LACP Fallback Test Plan

Related documents
LACP Fallback Design Document <https://github.com/Azure/SONiC/blob/gh-pages/doc/LACP%20Fallback%20Feature%20for%20SONiC_v0.3.docx>

Overview

This feature test suite is targeting on testing LACP fallback feature on SONiC. In our testbed, 't0', 't1-lag' and 't0-64' have LAG configurations. Each test covers a basic functionality of LACP fallback feature and ensures the switch works as expected under production scenarios.

Scope

The test is targeting a running SONIC system with fully functioning configuration.
The purpose of the test is not to test specific SAI API, but functional testing of LACP fallback on SONIC system, making sure that traffic flows correctly, according to BGP routes advertised by BGP peers of SONIC switch, and the LAG configuration.

NOTE: Test will be able to run **only** in the testbed specifically created for LAG, such as t0 and t1-lag.

Scale / Performance

No scale/performance test involved in this test plan.

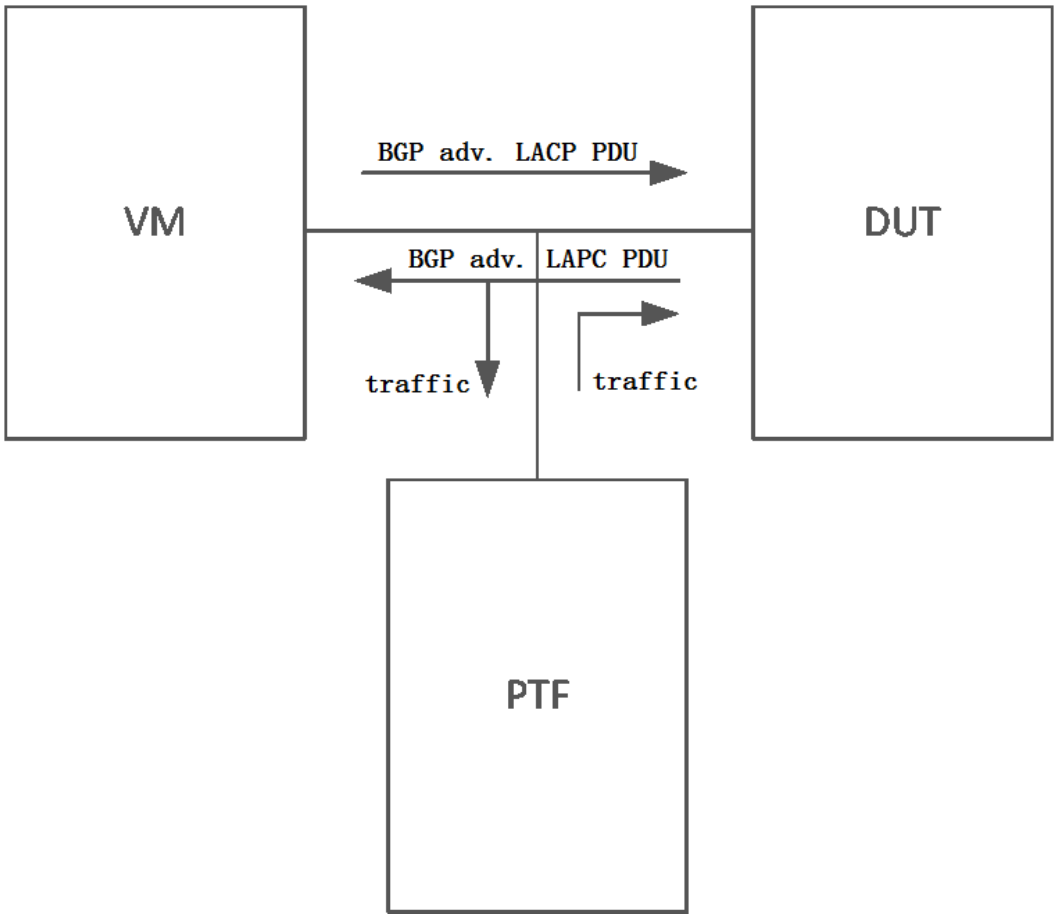
Related SAI APIs

sai_create_lag_member
sai_remove_lag_member

Related DUT CLI commands

DUT configuration is done via minigraph. See more information below

Setup configuration



- 8 LAGs per switch from the DUT to 8 EOS devices.
- Each of the LAG contains 2 members.
- BGP sessions:
  - 16 front panel ports north bound towards spine devices
  - 16 front panel ports combine each two to have 8 LAGs south bound towards servers
- All TORs advertise 6 routes and all spine routers advertise 6402 routes. It is similar to the test environment set up for leaf devices without LAGs.

Ansible infrastructure changes

Minigraph file

sonic-mgmt uses minigraph files to described VM set (Arista vEOS devices) and the DUT switch.
We will need to create a new minigraph file, describing VMs and the switch, with LAGs.
The switch file will be named switch-lacp-fallback.yml, and located at https://github.com/Azure/sonic-mgmt/blob/master/ansible/minigraph <https://github.com/Azure/sonic-mgmt/blob/master/ansible/minigraph> .

LAG related minigraph data

The LAG related data will be built from minigraph XML (minigraph_facts), namely from section.

Sample:

```
<PortChannelInterfaces>
  <PortChannel>
    <Name>PortChannel0</Name>
    <AttachTo>Ethernet0,Ethernet1</AttachTo>
    <Fallback>false</Fallback>
    <SubInterface/>
  </PortChannel>
  <PortChannel>
    <Name>PortChannel2</Name>
    <AttachTo>Ethernet2,Ethernt3</AttachTo>
    <Fallback>true</Fallback>
    <SubInterface/>
  </PortChannel>
</PortChannelInterfaces>
```

This information will be consumed by teamd.j2 template introduced in LAG testbed Pull Reqeust <https://github.com/Azure/sonic-mgmt/pull/69> (see port_channel variable) to produce LAG json configuration for teamd.
see Setup of DUT switch for details on teamd json.

LAG route information file

To properly validate traffic flow, PTF test will need to have know:

- route ip prefix
- list of LAGs which are members of ECMP group for given route
- member ports of a LAG

route_info.txt example

```
1.1.1.1 [0,1],[2,3] // ECMP members are LAG0, LAG2. LAG0 with members p0,p1, LAG2 with members p2,p3
2.2.2.2 [2,3],[4,5] // ECMP members are LAG2, LAG4. LAG2 with members p2,p3, LAG4 with members p4,p5
```

Ansible test setup script for LAG will generate route_info.txt file containing information mentioned above.
When invoking LAG PTF test, Ansible script will pass this file to the test.

Scripts for generating LAG configuration on SONIC

There will be lag.j2 script which will iterate over minigraph_facts and generate route_info.txt described above.
lag.j2 will be invoked by Ansible playbook which will setup and start LAG PTF test.

Ansible scripts to setup and run LAG test

lag.common.yml

We'll introduce lag.common.yml ansible playbook to setup and run LAG test case.

In high level description, the script will perform following steps:

1. Run lognanalyzer 'init' phase
2. Run LAGTest and pass to it route_info.txt
3. Run loganalyzer 'analyze' phase

lag.yml

We'll introduce lag.yml playbook, which will perform setups specific for each test case, and invoke the test.

lag.yml will:

- Generate BGP route_info.txt file with information about all BGP routes - see /test/files/template/fib.j2
- Generate route_info.txt once and pass to each test run.
- perform test-case-specific setup

- and invoke lag.common.yml

Setup of DUT switch

Setup of SONIC DUT will be done by Ansible scripts. During setup Ansible will push json file containing configuration for LAG. Data will be consumed by teamd.

J2 template to generate LAG configuration on SONIC

LAG testbed anbisle playbooks are using j2 scripts to generate JSON content to define the LAG structure for DUT. For each LAG port on DUT there be 1 json file is generated. So, in case of the setup with 8 LAGs, there will be 8 json files generated, each describing LAG and its member ports. Dedicated instance of teamd will be started with each json file.

An ansible playbook will push these files to the DUT during testbed setup.

Sample:

```
{
     "device": "PortChannel0",
     "runner": {
         "name": "lacp",
         "active": true,
         "fallback": true,
         "tx_hash": ["eth", "ipv4", "ipv6"]
     },
     "link_watch": {
         "name": "ethtool"
     },
     "ports": {
         Ethernet0:{},
         Ethernet4:{}
     }
}
```

Here `PortChannel0` is the name of LAG port.
`Ethernet0,Ethernet4` is the front panel ports-members of the `PortChannel0` lag port.

NOTE

- According to current implementation in SONIC, there will be 1 json file for each LAG port.
- For each LAG a separate teamd process will be started 1 json file.

Setup of fanout switch

Same as regular LAG testbed.

Setup of VMs

vEOS VMs will be setup during testbed setup with proper LAG layout.

PTF Test

Input files for PTF test

PTF test will be provided with a text input files describing the LAG layout and BGP routes on the DUT switch, and route_info.txt. Please see Ansible infrastructure changes section for description of both files.

Data in the files will be used to

- generate traffic (using route_info.txt)

- properly validate traffic is passing through valid LAGs physical ports.(using route_info.txt)

Traffic validation

Each LAG will be mentioned in route_info.txt, as member of ECMP group.
For each port well have port-to-lag mapping, see route_info.txt example

For each route (src_ip):

- validate that physical port through which packet was received belongs to one of the LAG ports, mentioned as ECMP members in the route_info.txt for given src__ip.
  1. using data from route_info.txt find mapping from physical port index to LAG index
  2. using route_info.txt check LAG index is in the ECMP group for the src__ip
- traffic distributed evenly between LAG member ports - by keeping packet counters in PTF test for each port
- traffic is distributed evenly between LAGs - by keeping counters in PTF for each LAG

The PTF test will keep per-port counter variables for counting packets arriving on different ports. The counters for LAG member ports will be used to compare for event traffic distribution.
SONIC doesn't have LAG counters.

Address types to validate

PTF test will send traffic for both IPV4 and IPV6 routes.

Test cases

The test assumes there is a mechanism to validate logs on the DUT where we should be able to analyze /var/log/syslog for error and warning messages related to the current test.
In case such messages are detected, the test is considered failed. See lognalyzer related comments in lag.common.yml section

Test case #1 - Fallback Verification

Test objective

Verify fallback functionality.

Test configurations

Shutdown/Unbundling vEOS VM port from LAG. Should be done using Arista command "shutdown" or "no channel-group" from **Ansible**. And all the BGP session are moved from LAGs to it's first member port.

Test description

Shutdown each LAG interface on VM.

To simulate the fallback environment we'll shutdown the EOS interface which simulates the halt of sending LACP packets on all of the LAG members. Please note the fanout switch interface will be kept up to simulate the fallback environment.

Following steps will be performed for each LAG port:

Shutdown LAG from vEOS VMs and validate no traffic flows.

1. Shutdown the EOS interface
   - lag.yml runs commands on VM to shutdown all member port.
2. Verify the LAG port is still selected and enabled on DUT
   - lag.yml invokes interface_facts.py to check LAG is selected and enabled on DUT.

Bring the LAG 'up' again and validate the traffic flows through it.

1. Bring 'up' the VM interface
   - [**clarify**][**how**] lag.yml runs commands on VM to bring the port up.
2. Verify LAG interface is 'up' on DUT
   - lag.yml invokes interface_facts.py to check LAG recovered on DUT.

Remove all member ports from LAG on vEOS vm port and verify that the lag interface is still up on DUT side if fallback is enabled.

For each LAG interface on vEOS vm, remove all member ports from LAG port on vEOS vm, and bring them up as individual physical ports.

- lag.yml uses “no channel-group” command to unbundle all member ports from the LAG on vEOS vm.
  - ansible will invoke interface_facts.py to validate corresponding LAG on DUT is up if fallback is enabled on LAG, or went down if fallback is disabled.
- lag_test.py will send packets to all LAGs (except lag-down-index) and validate
  - packets arrive on those LAGs
  - packets are received over fallback member port of LAGs

Test case #2 - Recovery Verification

Test objective

Verify traffic between legs evenly distributed after the LAG recovered from fallback state upon receiving LACP PDU and BGP routes over LAGs established on vEOS. Traffic is forwarded by SONIC DUT.

Test configurations

Bundling vEOS VM ports into LAG and configure BGP over the LAG according the normal t0/t1-LAG configuration. Bundling should be done using Arista command “channel-group" on vEOS VM from **Ansible**.

Test description

Recover DUT LAGs from fallback state.

Add the member ports back into the LAG port on DUT and verify that the LAG interface is back to normal.

- ansible will call “channel-group” cmd to adding member ports back to LAG on vEOS vm.
- ansible will invoke interface_facts.py to validate corresponding LAG on DUT is UP.
- PTF host will send packets according to the route_info.txt - will create packets with dst_ip according to route prefixes.
- When packet reaches to SONIC DUT, it will route packet according to BGP routes, and send it to one of vEOS BGP peers.
- PTF test will receive a copy of the packet and perform validations described in Validation of Traffic
  NOTE: We are not targeting testing traffic coming into the DUT from BGP peers.

Open Questions