Arista Scale with Symmetry Guide

The ever-increasing traffic levels across data networks have created a challenge for even the most high performance inline network appliances. While network devices are processing packets at higher and higher rates, ever increasing demand can require seamless expansion of a network.

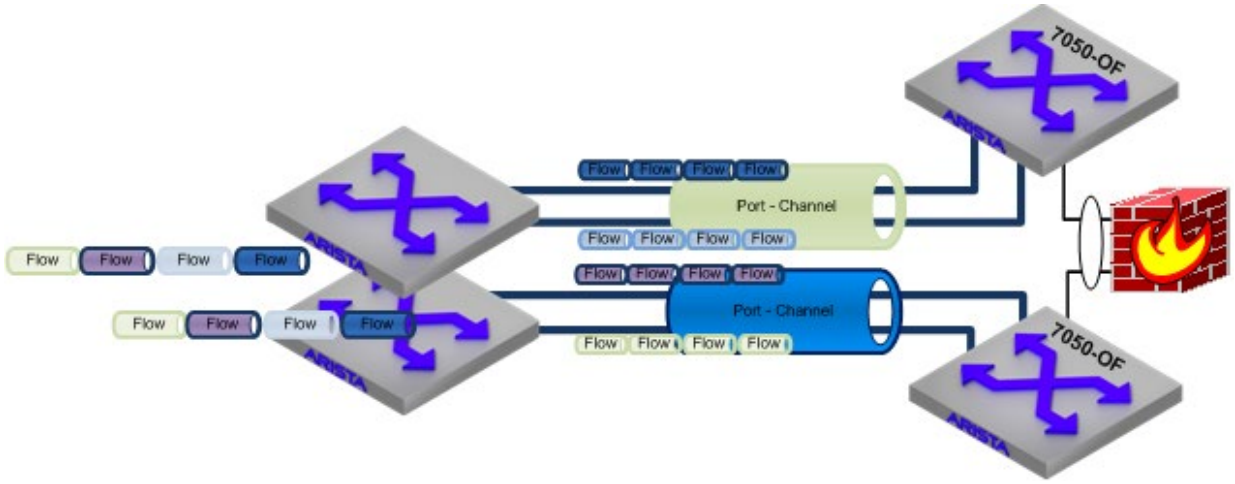
To tackle this problem, Arista Networks has partnered with Palo Alto Networks to help our customers scale to the performance demands on their networks. While the initial goal was to build a 100 Gbps firewall cluster, the actual solution scale is only limited by the physical number of ports on the switch and can be used by any network-based appliance. Leveraging the extensibility built into every Arista Extensible Modular Operating System (EOS)TM, the Arista OpenFlow Direct Mode, and the unique deployment options offered by the Palo Alto Networks virtual-wire mode, we are able to deliver scale while ensuring flow symmetry.

Palo Alto Networks next-generation firewalls enable unprecedented visibility and granular policy control of applications and content – by user, not just IP address – at 20 Gbps network throughput levels. Based on patent-pending App-IDTM technology, Palo Alto Networks firewalls accurately identify and control applications – regardless of port, protocol, evasive tactic or SSL encryption – and scan content to stop threats and prevent data leakage. Palo Alto Networks' products and services can address a broad range of network security requirements, from the data center to the network perimeter, as well as the distributed enterprise, which includes branch offices and a growing number of mobile devices.

This Arista Networks and Palo Alto Networks joint solution focuses on delivering investment protection by horizontally scaling firewall performance to add capacity while maintaining sessions. While this document is focused on firewall scaling, this concept can be applied to many other types of devices, for example IDS/IPS devices, proxies, antivirus pods, DDOS mitigation devices, WAN Optimizers.

It's All About the Hash

Network devices have gone from simple source / destination IP hashing algorithms to fine grained advanced hashing on many fields within the packet. This has created excellent flow distribution across many parallel Layer 2 and 3 paths. The algorithms also ensure single session flows all hash the same links so that packets in the same flow are not distributed across multiple interfaces. While these advanced hash algorithms are great for load balancing they tend to not be friendly to devices that inspect traffic flows.

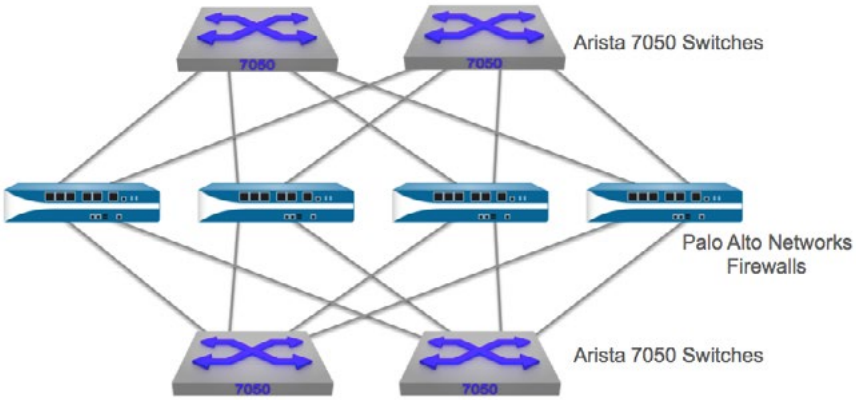


Symmetric Hash

Horizontal scale provides the ability to expand capacity by adding devices in a parallel fashion. The main benefit to this solution is that the addition requires no architectural changes to the infrastructure. In the past this would require an engineering effort to investigate, and potentially remediate by splitting services into separate pods, which is both service impacting and costly.

Validation

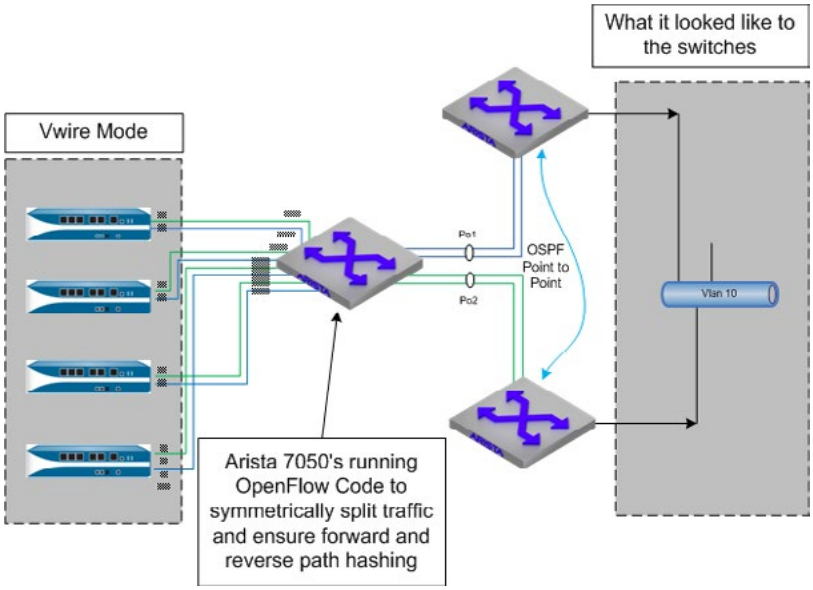
This solution was validated through a joint effort between Arista and Palo Alto Networks. Validation included two Arista 7050 switches sending traffic across four Palo Alto Networks PA-5060 firewalls.



The Building Blocks

The solution is built upon basic layer 2 connectivity between the "devices requiring symmetric scale" and the Arista 7050 OpenFlow enabled switches. If two switches are used in the topology, connectivity to both switches are required. This removes the potential for asymmetric flows being sent to the devices.

Initially the solution was tested on a single switch in the below topology. It was further enhanced to provide link level redundancy across a pair of 7050s.

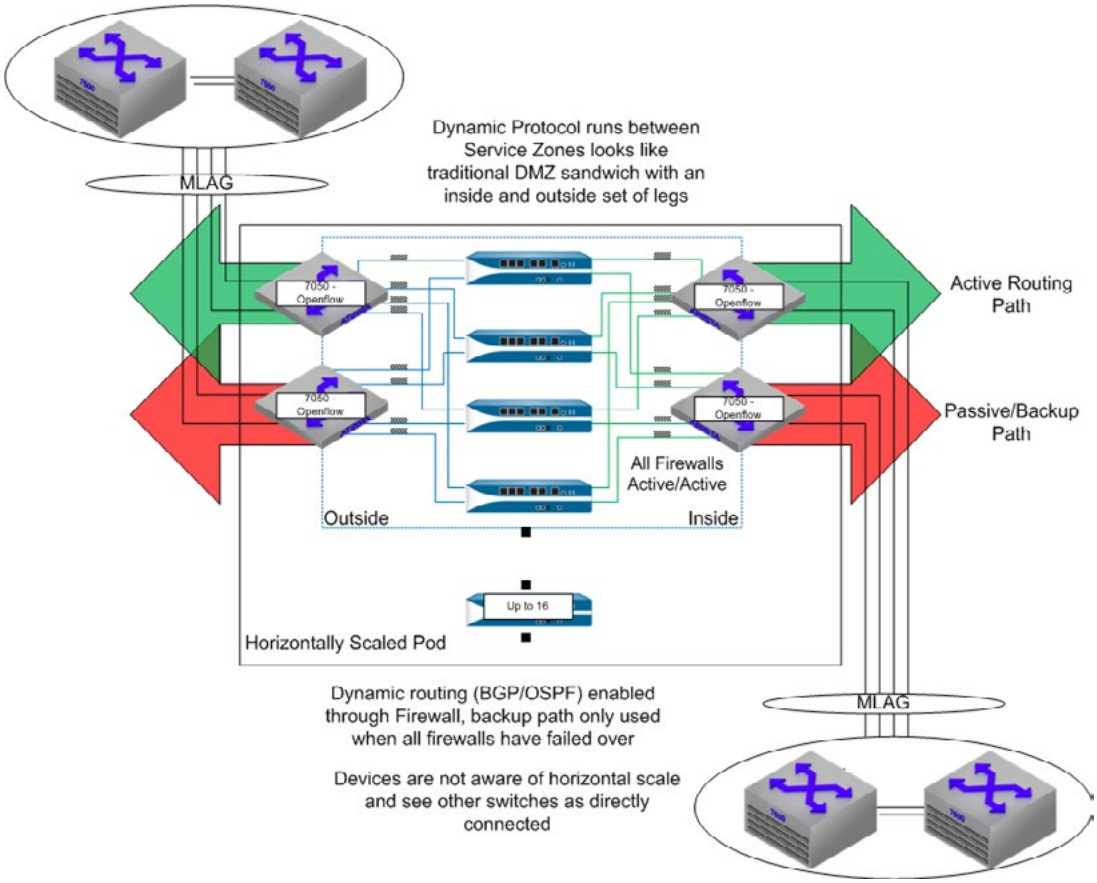


Solution Overview High Level Operation

The basic operation of the solution is as follows; an event handler is configured on the 7050 switches to trigger a script to run based on interface status change. The event handler feeds variables into the script to create the hash buckets based on the number of devices to be included in the hash. A priority value is used to ensure that if a link or firewall fails, the next device in the hash takes over. Below is a basic flow chart illustrating the operation.

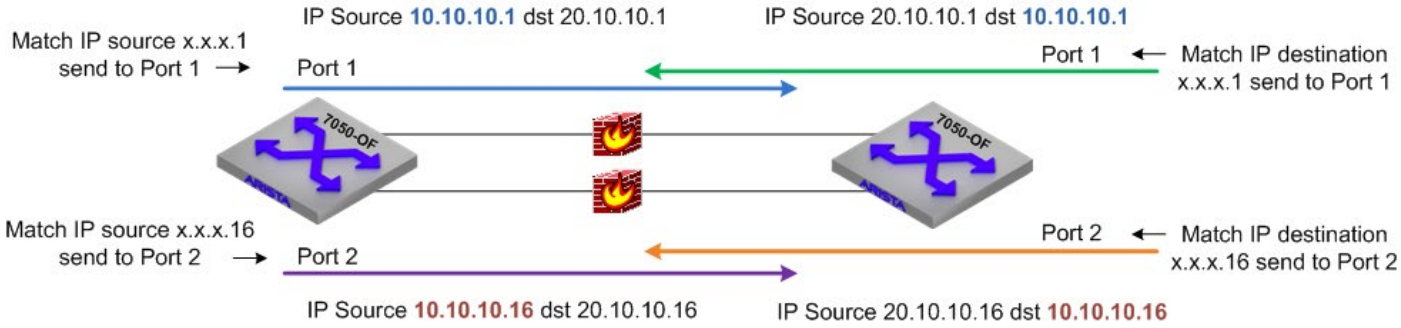
Topology Diagram

A typical tiered firewall diagram is shown below; this topology separates the inside and outside interfaces on to discrete switches. While this model may be preferred for some security organizations, it does pose some minor issues that need to be resolved. Without a view of both firewall legs, continuity through the firewalls needs to be verified before bringing the upstream interfaces online.



The Script Foundation

Simplicity is key; while the operation of the flow splitting seems complex, the operation is actually quite simple. To ensure symmetric hashing on flows through these types of devices the switching layer at each side of the firewall pairs needs to operate in a reverse hashing mode. Flows from one side are hashed based on source IP while the return traffic is hashed based on destination IP to ensure the response traffic traverses the same firewall.



This simple but effective hashing leads to load splitting based on real world traffic patterns (I.E. lots of varied sources). And, this can be modified to be even finer by changing the number of buckets on the inverse mask hash. The limit of the buckets is based upon the free ACL entries. Currently OpenFlow allows up to 750 entries in the default table profile or up to 1500 entries in the L2-only profile. In all tested scenarios, the amount of flow entries would not come close to these limits. What should be noted is that source IP addresses have an effect on firewall loading, for example if all the source IP's are (.1) the solution would not balance traffic appropriately. It's best to ensure a good mix of varied sources.

Switch OpenFlow Configuration

In OpenFlow mode, the switch will not act as a traditional learning switch, and will only forward flows when they exist in the flow table. In our example, the flows must be programed by the event handler to forward traffic. The default action if no flows are programed is to drop all traffic. When using OpenFlow, it is best to have a thorough understanding of the OpenFlow switch specification.

Enable OpenFlow on the switch globally:

```
Switch(config)#openflow <cr>
```

Enable Bind mode and Bind interfaces to OpenFlow both the physical and the incoming port-channel :

```
Switch(config-openflow)# bind mode interface <cr>

Switch(config-openflow)#bind interface ethernet <Num>

$        end of range

<1-52>  Port number

Switch(config-openflow)#bind interface Port-Channel <Num>

Switch(config-openflow)#profile full-match

Switch(config-openflow)#default-action drop

*must enable the configuration to take effect:

Switch(config-openflow)#enable
```

Event Handler Configuration

In order to provide failover capabilities and generate the OpenFlow Direct Mode commands, event handlers are used to trigger the creation of the flow entries and hash mask. The event handler is configured to trigger a script to fire when an interface operational state transitions. The event handler is configured via standard Cli. Below is the syntax of the event handler configuration.

Note the interfaces are a matching pair, the below event handler sends 4 variables into the script. The variables are used to configure the flow entries when the interface changes state. These variables are based on the below key and need to be configured on each switch participating in the flow hashing. When finished each firewall interface should have an event handler associated with it.

```
Switch(config)# event-handler fw1-east

Switch(config-event-handler-fw1-east)#trigger onIntf eth1 operstatus

Switch(config-event-handler-fw1-east)#delay 0

Switch(config-event-handler-fw1-east)#action bash /mnt/flash/fw_change.sh 1 4 src Po1

Switch(config-event-handler-fw1-east)#exit
```

Key: (configuration assumes 4 firewalls)
eth1 = Ethernet1 (operational status)
1 = first interface/device in the hash
4 = total number of interfaces (devices connected to the switch) src = hash based on source IP
Po1 = incoming interface to upstream switches (source of traffic)

Configure the opposite Switch to match the other direction with the below event handler.

```
Switch(config)# event-handler fw1-west
```

```
Switch(config-event-handler-fw1-west)#trigger onIntf eth2 operstatus
Switch(config-event-handler-fw1-west)#delay 0
Switch(config-event-handler-fw1-west)#action bash /mnt/flash/fw_change.sh 1 4 dst Po2
Switch(config-event-handler-fw1-west)#exit
```

Key:
eth2 = Ethernet2 (operational status)
1 = first interface/device in the hash
4 = total number of interfaces
dst = hash based on destination IP
Po2 = incoming interface to upstream switches (source of traffic)

The ACL "Hash Mask"

To create the load balancing of IP packets to symmetric device interface, a simple hash on the source and or destination IP is used. This hash makes use of TCAM don't care bits, and OpenFlow flexible matching criteria. An ACL match on src-ip 1.1.1.0 netmask 0.0.0.224 will match

- 11.22.33.0 thru 11.22.33.31
- 22.33.44.0-32 thru 22.33.44.31
- 203.9.111.0 thru 203.9.111.31
- ...

By using this style of matching the script creates 'buckets' where each bucket has a policy to send to a different device. In reality it can create as many 'buckets' as there are free ACL entries. More 'buckets' = more granular traffic distribution, by default we use 32, which when using 8 firewalls yields approximately 150 flow entries. To provide more granular results we could increase the number to 64, which with 8 devices would be around 300 flow entries.

Script Operation

A simple shell script is called when the port operation state changes from up to down. The interface parameters are feed into the script and OpenFlow rules are created and applied or removed. This is the breakdown of the script for illustration purposes:

View the detailed script in the EOS script library on Google Drive (fw_change.sh). Note this library is open to all Arista registered users.

https://eos.aristanetworks.com/home.php

The script can be loaded onto the flash using the 'copy' command. eg. copy http://myserver.example.com/fw_change.sh flash:
Description of operation within script:

```
#!/bin/sh
# Copyright (c) 2013 Arista Networks, Inc. All rights reserved.
# Arista Networks, Inc. Confidential and Proprietary.
#
# fw_change.sh <n> <m> <src or dst> <matching_if>
#
# Script to do traffic steering to firewall <n> of <m> firewalls based on <src or dst>
# direction. Traffic is steered from/to <matching_if>
#
# Script picks up interface status update (up/down) from $OPERSTATE and interface
# $INTF both passed from an event-handler
```

Within the script we first create a flow entry so that ARP's are flooded.

```
# blanket openflow rule to make sure all devices get ARPs
echo "flow arp-flood" >> $FILE
echo " match ethertype arp" >> $FILE
echo " action output flood" >> $FILE
echo " hard timeout 0" >> $FILE
echo " exit" >> $FILE
```

In order to create the inverse mask for the IP hash mask we use a BUCKETS variable to configure the number of buckets. By default we use 32, which when using 8 firewalls yields approximately 150 flow entries. To provide more granular results we could increase the number to 64, which with 8 devices would be around 300 flow entries.

Inverse mask calculation:

```
BUCKETS=32
HASHINCR=$(( 256/$BUCKETS ))
HASHMASK=$(( 256-$HASHINCR ))
```

The main calculations of the script can be seen in the following snipit. Note that there are also calculations to create priorities for each flow entry. These priorities allow for a device failure to re-shuffle the traffic to the next device based on a configured priority value.

```
MY_NUM=$1
TOTAL_NUM=$2
HASH_ON=$3
MATCHING_IF=$4

BUCKETS=64
HASHINCR=$(( 256/$BUCKETS ))
HASHMASK=$(( 256-$HASHINCR ))

# N rules for traffic TO firewall
for BUCKET in `seq $BUCKETS`
do
      BUCKET_IP=$[($BUCKET-1) * $HASHINCR]
      PRIORITY=$[(1000-((($BUCKET + $MY_NUM - 2) % $TOTAL_NUM) * 10))]
      FLOWNAME="to-fw$MY_NUM-$MATCHING_IF-$INTF-b$BUCKET"

      if [ "$OPERSTATE" = "linkup" ] ;
            then echo "flow $FLOWNAME" >> $FILE
            echo " match input interface $MATCHING_IF" >> $FILE
            echo " match ethertype ip" >> $FILE
            if [ "$HASH_ON" = "src" ] ; then
            echo " match source ip 1.1.1.$BUCKET_IP mask 0.0.0.$HASHMASK" >> $FILE
      elif [ "$HASH_ON" = "dst" ; then
            echo " match destination ip 1.1.1$BUCKET_IP mask 0.0.0$HASKMASK" >>
            $FILE
```

```
        if
        echo " hard timeout 0" >> $FILE
        echo " priority $PRIORITY" >> $FILE
        echo " action output interface $INTF >> $FILE
        echo " exit" >> $FILE
```

The output of the above script creates 32 flow entries, starting at bucket1 (b1) with a priority of 1000. Below is the output of the script showing what commands are created.

```
conf t
openflow
flow arp-flood
      match ethertype arp
      action output flood
      hard timeout 0
      exit
flow from-fw1-et1-Po1
      match input interface et20
      hard timeout 0
      action output interface Po1
      exit
flow to-fw1-Po1-1-b1
      match input interface Po1
      match ethertype ip
      match source ip 1.1.1.0 mask 0.0.0.248
      hard timeout 0
      priority 1000
      action output interface et20
      exit
...
```

Finally we trigger a log message based on the calculations for display purposes.

```
      logger -p CRIT -t FW_CHANGE added flow $FLOWNAME inputif $MATCHING_IF x.x.x.$BUCKET_IP
mask
0.0.0.$HASHMASK priority $PRIORITY outputif $INTF
      elif [ "$OPERSTATE" = "linkdown" ] ; then
            echo "no flow $FLOWNAME" >> $FILE
            logger -p CRIT -t FW_CHANGE removed flow $FLOWNAME
```

Sample Topologies - Use Cases

Apart from the firewall use case, some other potential solutions: Split flows symmetrically across a set of IDS/IPS boxes that can't support higher traffic rates.

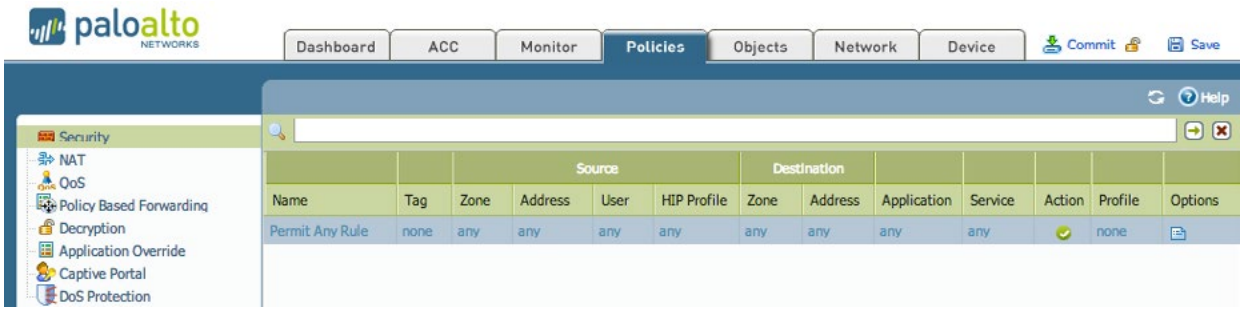
Palo Alto Networks Firewall Configuration

The configuration of each firewall is very simple. Four interfaces on each firewall are utilized, configured in two pairs of interfaces. Each pair is configured in virtual wire mode. The below snapshot provides an example of a Palo Alto Networks firewall configuration connected to an OpenFlow pair of 7050 switches. The firewall's four 10 Gbps interfaces are used in the following example:

ethernet1/21	Virtual Wire			none	none	Untagged	VWire1	vsys1	trust
ethernet1/22	Virtual Wire			none	none	Untagged	VWire1	vsys1	untrust
ethernet1/23	Virtual Wire			none	none	Untagged	VWire	vsys1	trust
ethernet1/24	Virtual Wire			none	none	Untagged	VWire	vsys1	untrust

Notice that the first two interfaces are part of one virtual-wire (vWire), while the second two interfaces are part of another vWire. Also notice that each vWire has a trust side, and an untrust side which connect to the switches on that side of the firewall.

Configure each firewall with a similar interface configuration. You may also want to configure a "permit any" policy on each firewall during initial testing of the solution, then remove it later when securing the network for production. Here is a screenshot of an example "permit any" policy.



Sample Topologies - Use Cases

```
show openflow
show openflow flows
show openflow statistics
clear openflow statistics
show openflow ports
show openflow profiles
show openflow queues
```

Example output:

```
show openfl fl | grep -A 9 fw1 | egrep '(fw|prior|matched)'
[..]
Flow fw1-Po1-Ethernet20-1:
  priority: 1000
  matched: 5893750 packets, 483287828 bytes
[..]
```

Reference Documents

OpenFlow Direct Mode:
https://docs.google.com/a/aristanetworks.com/document/d/1nw76A7EaBJN56qKkH5NHOQYfliYeqq2NraaL1E- GPRY/edit

7050 Series Data Sheet
http://www.aristanetworks.com/en/products/7050series

Palo Alto Networks Data Sheets
http://www.paloaltonetworks.com/literature/datasheets/

OpenFlow Information
http://www.openflow.org

Conclusion

Utilizing Arista's EOS and an OpenFlow-enabled pair of switches, traffic can be hashed symmetrically across several devices, basically up to the amount of ports supported by a pair of 7050 switches. The configuration makes use of CLI-based commands in conjunction with event triggered scripting. With this combination of flexibility and ease of deployment the potential use cases are endless.

Feature and Version Specific Notes

Currently OpenFlow Direct Mode is supported only in dedicated OpenFlow mode, once in this mode typical control plane protocols like MLAG and LACP are not supported.

When in OpenFlow Direct Mode in interface bind mode we disable MAC learning on the interface, this will be fixed in Q3CY2013 but something that may cause issues with lab testing as we will flood packets in the VLAN for unknown DA.

Palo Alto Networks software version tested was 5.0.3, however, any supported version would work in this solution. Arista 7050 with EOS 4.10.4

Santa Clara—Corporate Headquarters
5453 Great America Parkway,
Santa Clara, CA 95054

Phone: +1-408-547-5500
Fax: +1-408-538-8920
Email: info@arista.com

Ireland—International Headquarters
3130 Atlantic Avenue
Westpark Business Campus
Shannon, Co. Clare
Ireland

Vancouver—R&D Office
9200 Glenlyon Pkwy, Unit 300
Burnaby, British Columbia
Canada V5J 5J8

San Francisco—R&D and Sales Office
1390 Market Street, Suite 800
San Francisco, CA 94102

India—R&D Office
Global Tech Park, Tower A & B, 11th Floor
Marathahalli Outer Ring Road
Devarabeesanahalli Village, Varthur Hobli
Bangalore, India 560103

Singapore—APAC Administrative Office
9 Temasek Boulevard
#29-01, Suntec Tower Two
Singapore 038989

Nashua—R&D Office
10 Tara Boulevard
Nashua, NH 03062