g+1 1 More Next Blog» Create Blog Sign In

Monitoring converged networks using the sFlow® standard

Wednesday, April 23, 2014

Mininet integrated hybrid OpenFlow testbed

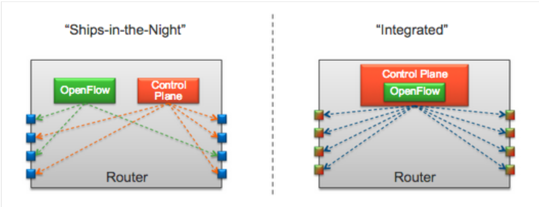

Figure 1: Hybrid Programmable Forwarding Planes

Integrated hybrid OpenFlow combines OpenFlow and existing distributed routing protocols to deliver robust software defined networking (SDN) solutions. Performance optimizing hybrid OpenFlow controller describes how the sFlow and OpenFlow standards combine to deliver visibility and control to address challenges including: DDoS mitigation, ECMP load balancing, LAG load balancing, and large flow marking.

A number of vendors support sFlow and integrated hybrid OpenFlow today, examples described on this blog include: Alcatel-Lucent, Brocade, and Hewlett-Packard. However, building a physical testbed is expensive and time consuming. This article describes how to build an sFlow and hybrid OpenFlow testbed using free Mininet network emulation software. The testbed emulates ECMP leaf and spine data center fabrics and provides a platform for experimenting with analytics driven feedback control using the sFlow-RT hybrid OpenFlow controller.

First build an Ubuntu 13.04 / 13.10 virtual machine then follow instructions for installing Mininet - Option 3: Installation from Packages.

Next, install an Apache web server:

```
sudo apt-get install apache2
```

Install the sFlow-RT integrated hybrid OpenFlow controller, either on the Mininet virtual machine, or on a different system (Java 1.6+ is required to run sFlow-RT):

```
wget http://www.inmon.com/products/sFlow-RT/sflow-rt.tar.gz
tar -xvzf sflow-rt.tar.gz
```

Copy the *leafandspine.py* script from the *sflow-rt/extras* directory to the Mininet virtual machine.

The following options are available:

```
./leafandspine.py --help
Usage: leafandspine.py [options]

Options:
  -h, --help            show this help message and exit
  --spine=SPINE         number of spine switches, default=2
  --leaf=LEAF           number of leaf switches, default=2
  --fanout=FANOUT       number of hosts per leaf switch, default=2
  --collector=COLLECTOR
                        IP address of sFlow collector, default=127.0.0.1
  --controller=CONTROLLER
                        IP address of controller, default=127.0.0.1
  --topofile=TOPOFILE   file used to write out topology, default topology.txt
```

Figure 2 shows a simple leaf and spine topology consisting of four hosts and four switches:

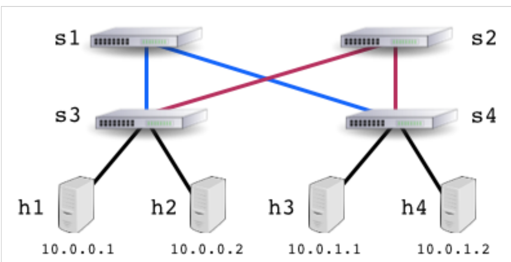

Figure 2: Simple leaf and spine topology

The following command builds the topology and specifies a remote host (10.0.0.162) running sFlow-RT as the hybrid OpenFlow controller and sFlow collector:

Subscribe To

Posts
Comments

Follow by Email

Email address... Submit

Links

- What is sFlow?
- sFlow.org - Industry consortium
- Switches with sFlow
- Switch configuration settings
- Host sFlow - Free server performance agent
- sFlowTrend - Free, graphical sFlow analyzer
- sFlow-RT - Real-time analytics

Popular Posts

Docker performance monitoring
IT'S HERE: DOCKER 1.0 recently announced the first production release of the Docker Linux container platform. Docker is seeing explosive ...

Installing Host sFlow on a Linux server
The Host sFlow agent supports Linux performance monitoring, providing a lightweight, scalable solution for monitoring large numbers of Li...

Sampling rates
A previous posting discussed the scalability and accuracy of packet sampling and the advantages of packet sampling for network-wide visib...

Top of rack network virtualization
Network virtualization (credit Brad Hedlund) Support for tunneling protocols (NVGRE, VxLAN etc.) and OpenFlow in top of rack switche...

Configuring FortiGate appliances
The recent FortiOS 4.0 MR2 release adds sFlow support to Fortinet's FortiGate® appliances. The following commands configure a FortiGat...

sflowtool
The sflowtool command line utility is used to convert standard sFlow

```
sudo ./leafandspine.py --collector 10.0.0.162 --controller 10.0.0.162 --topofile
/var/www/topology.json
```

Note: All the links are configured to 10Mbit/s and the sFlow sampling rate is set to 1-in-10. These settings are equivalent to a 10Gbit/s network with a 1-in-10,000 sampling rate - see Large flow detection.

The network topology is written to */var/www/topology.json* making it accessible through HTTP. For example, the following command retrieves the topology from the Mininet VM (10.0.0.61):

```
curl http://10.0.0.61/topology.json
{"nodes": {"s3": {"ports": {"s3-eth4": {"ifindex": "392", "name": "s3-eth4"},
"s3-eth3": {"ifindex": "390", "name": "s3-eth3"}, "s3-eth2": {"ifindex": "402",
"name": "s3-eth2"}, "s3-eth1": {"ifindex": "398", "name": "s3-eth1"}}, "tag":
"edge", "name": "s3", "agent": "10.0.0.61", "dpid": "0000000000000003"}, "s2":
{"ports": {"s2-eth1": {"ifindex": "403", "name": "s2-eth1"}, "s2-eth2": {"ifindex":
"405", "name": "s2-eth2"}}, "name": "s2", "agent": "10.0.0.61", "dpid":
"0000000000000002"}, "s1": {"ports": {"s1-eth1": {"ifindex": "399", "name":
"s1-eth1"}, "s1-eth2": {"ifindex": "401", "name": "s1-eth2"}}, "name": "s1",
"agent": "10.0.0.61", "dpid": "0000000000000001"}, "s4": {"ports": {"s4-eth2":
{"ifindex": "404", "name": "s4-eth2"}, "s4-eth3": {"ifindex": "394", "name":
"s4-eth3"}, "s4-eth1": {"ifindex": "400", "name": "s4-eth1"}, "s4-eth4":
{"ifindex": "396", "name": "s4-eth4"}}, "tag": "edge", "name": "s4", "agent":
"10.0.0.61", "dpid": "0000000000000004"}}, "links": {"s2-eth1": {"ifindex1": "403",
"ifindex2": "402", "node1": "s2", "node2": "s3", "port2": "s3-eth2", "port1":
"s2-eth1"}, "s2-eth2": {"ifindex1": "405", "ifindex2": "404", "node1": "s2",
"node2": "s4", "port2": "s4-eth2", "port1": "s2-eth2"}, "s1-eth1": {"ifindex1":
"399", "ifindex2": "398", "node1": "s1", "node2": "s3", "port2": "s3-eth1",
"port1": "s1-eth1"}, "s1-eth2": {"ifindex1": "401", "ifindex2": "400", "node1":
"s1", "node2": "s4", "port2": "s4-eth1", "port1": "s1-eth2"}}}
```

Don't start sFlow-RT yet, it should only be started after Mininet has finished building the topology.

Verify connectivity before starting sFlow-RT:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

This test demonstrates that the Mininet topology has been constructed with a set of default forwarding rules that provide connectivity without the need for an OpenFlow controller - emulating the behavior of a network of integrated hybrid OpenFlow switches.

The following sFlow-RT script *ecmp.js* demonstrates ECMP load balancing in the emulated network:

```
include('extras/json2.js');

// Define large flow as greater than 1Mbits/sec for 1 second or longer
var bytes_per_second = 1000000/8;
var duration_seconds = 1;

var top = JSON.parse(http("http://10.0.0.61/topology.json"));
setTopology(top);

setFlow('tcp',
 {keys:'ipsource,ipdestination,tcpsourceport,tcpdestinationport',
  value:'bytes', t:duration_seconds}
);

setThreshold('elephant',
 {metric:'tcp', value:bytes_per_second, byFlow:true, timeout:2}
);

setEventHandler(function(evt) {
 var rec = topologyInterfaceToLink(evt.agent,evt.dataSource);
 if(!rec || !rec.link) return;
 var link = topologyLink(rec.link);
 logInfo(link.node1 + "-" + link.node2 + " " + evt.flowKey);
},['elephant']);
```

Modify the sFlow-RT start.sh script to include the following arguments:

```
RT_OPTS="-Dopenflow.controller.start=yes -Dopenflow.controller.flushRules=no"
SCRIPTS="-Dscript.file=ecmp.js"
```

Some notes on the script:

1. The topology is retrieved by making an HTTP request to the Mininet VM (10.0.0.61)
2. The 1Mbits/s threshold for large flows was selected because it represents 10% of the bandwidth of the 10Mbits/s links in the emulated network

records into a variety of different formats. While there are a large ...

Frenetic, Pyretic and Resonance
Northbound APIs for traffic engineering describes some of the limitations with current OpenFlow controllers and describes some of the fea...

Choosing an sFlow analyzer
sFlow achieves network-wide visibility by shifting complexity away from the switches to the sFlow analysis application. Simplifying the mo...

Installing Host sFlow on a Windows server
The Host sFlow agent supports Windows performance monitoring, providing a lightweight, scalable solution for monitoring large numbers of W...

DDoS mitigation with Cumulus Linux
Figure 1: Real-time SDN Analytics for DDoS mitigation Figure 1 shows how service providers are ideally positioned to mitigate large flo...

Labels

802.1Q 802.1aq accuracy ACI ADC agent analyzer AoE application ASIC bandwidth baseline blade server blade switch cloud computing cluster complexity configure control convergence cost data center dashboards DevOps DoS ECMP Ethernet FCoE Ganglia GPU Graphite Hadoop header HTTP Hyper-V IEEE IETF integration IP IPFIX IPv6 iSCSI java LAG LAN Linux load balancer management map measurement measurement delay Memcache MIB Mininet mirror multi-vendor NetFlow network network adapter observability OpenFlow overhead ovs polling power probe QoS RMON router sampling SAN scalability standard SDN server sFlow sFlow-RT sflowtool sFlowTrend SNMP SPAN spanning tree SR-IOV standard storage switch syslog topology TRILL utilization VDI VEPA virtualization visibility VoIP vSwitch WAN Windows wireless WLAN Xen

Blog Archive

- ▼ 2014 (25)
  - ► July (1)
  - ► June (4)
  - ► May (2)
  - ▼ April (4)
    - Mininet integrated hybrid OpenFlow testbed
    - Configuring Mellanox switches

3. The event handler prints the link the flow traversed - identifying the link by the pair of switches it connects

Start sFlow-RT:

```
./start.sh
```

Now generate some large flows between h1 and h3 using the Mininet iperf command:

```
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['9.58 Mbits/sec', '10.8 Mbits/sec']
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['9.58 Mbits/sec', '10.8 Mbits/sec']
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['9.59 Mbits/sec', '10.3 Mbits/sec']
```

The following results were logged by sFlow-RT:

```
2014-04-21T19:00:36-0700 INFO: ecmp.js started
2014-04-21T19:01:16-0700 INFO: s1-s3 10.0.0.1,10.0.1.1,49240,5001
2014-04-21T19:01:16-0700 INFO: s1-s4 10.0.0.1,10.0.1.1,49240,5001
2014-04-21T20:53:19-0700 INFO: s2-s4 10.0.0.1,10.0.1.1,49242,5001
2014-04-21T20:53:19-0700 INFO: s2-s3 10.0.0.1,10.0.1.1,49242,5001
2014-04-21T20:53:29-0700 INFO: s1-s3 10.0.0.1,10.0.1.1,49244,5001
2014-04-21T20:53:29-0700 INFO: s1-s4 10.0.0.1,10.0.1.1,49244,5001
```

The results demonstrate that the emulated leaf and spine network is performing equal cost multi-path (ECMP) forwarding - different flows between the same pair of hosts take different paths across the fabric (the highlighted lines correspond to the paths shown in Figure 2).

> Open vSwitch in Mininet is the key to this emulation, providing sFlow and multi-path forwarding support

The following script implements the large flow marking example described in Performance optimizing hybrid OpenFlow controller:

```
include('extras/json2.js');

// Define large flow as greater than 1Mbits/sec for 1 second or longer
var bytes_per_second = 1000000/8;
var duration_seconds = 1;

var idx = 0;

var top = JSON.parse(http("http://10.0.0.61/topology.json"));
setTopology(top);

setFlow('tcp',
 {keys:'ipsource,ipdestination,tcpsourceport,tcpdestinationport',
  value:'bytes', t:duration_seconds}
);

setThreshold('elephant',
 {metric:'tcp', value:bytes_per_second, byFlow:true, timeout:4}
);

setEventHandler(function(evt) {
 var agent = evt.agent;
 var ds = evt.dataSource;
 if(topologyInterfaceToLink(agent,ds)) return;

 var ports = ofInterfaceToPort(agent,ds);
 if(ports && ports.length == 1) {
  var dpid = ports[0].dpid;
  var id = "mark" + idx++;
  var k = evt.flowKey.split(',');
  var rule= {
    priority:1000, idleTimeout:2,
    match:{dl_type:2048, nw_proto:6, nw_src:k[0], nw_dst:k[1],
           tp_src:k[2], tp_dst:k[3]},
    actions:["set_nw_tos=128","output=normal"]
  };
  setOfRule(dpid,id,rule);
 }
},['elephant']);

setFlow('tos0',{value:'bytes',filter:'iptos=00000000',t:1});
setFlow('tos128',{value:'bytes',filter:'iptos=10000000',t:1});
```

Some notes on the script:

1. The topologyInterfaceToLink() function looks up link information based on agent and interface. The event

DDoS mitigation hybrid OpenFlow controller

Cisco, ACI, OpFlex and OpenDaylight

► March (3)

► February (4)

► January (7)

► 2013 (51)

► 2012 (37)

► 2011 (47)

► 2010 (55)

► 2009 (39)

Total Pageviews

712,322

Translate

Author's Profile

View my profile on Linked in

handler uses this function to exclude inter-switch links, applying controls to ingress ports only.

2. The OpenFlow rule priority for rules created by controller scripts must be greater than *500* to override the default rules created by *leafandspine.py*
3. The *tos0* and *tos128* flow definitions have been added to so that the re-marking can be seen.

Restart sFlow-RT with the new script and use a web browser to view the default **tos0** and the re-marked **tos128** traffic.

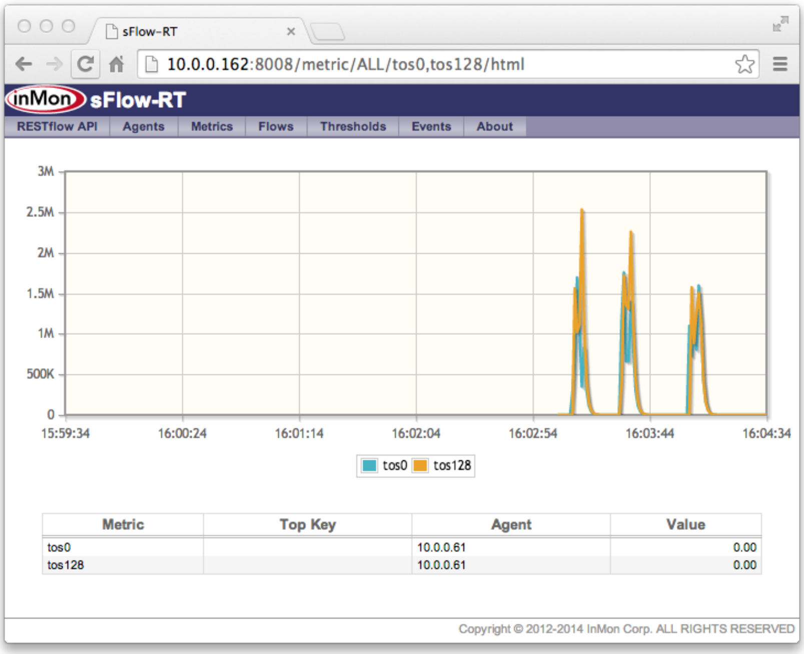

Figure 3: Marking large flows

Use *iperf* to generate traffic between h1 and h3 (the traffic needs to cross more than one switch so it can be observed before and after marking). The screen capture in figure 3 demonstrates that the controller immediately detects and marks large flows.

Posted by Peter at 10:06 PM

+1 Recommend this on Google

Labels: control, OpenFlow, sFlow, sFlow-RT

No comments:

Post a Comment

Enter your comment...

Comment as: Select profile...

Publish **Preview**

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)