big switch networks

Controller User Guide

Version 2011.04.19.0050-build.bigswitchcontroller.release

April 19, 2011 - Alpha

1. Getting started

Big Switch Controller is bundled in two formats:

- a virtual hard disk image that can be installed and booted on a virtual machine platform
- an upgrade package that can be used to upgrade all the software in an existing controller

This section will guide you through the procedures of installing, getting started with, and upgrading Big Switch Controller.

1.1. Prerequisites

While the controller is known to work on a variety of VM platforms, we strongly recommend you use VMware vSphere Hypervisor (ESXi) unless you are an expert with some other platform.

VMware ESXi host:

The following resources are required on the host:

- memory: 1024 MB
- disk space: 20 GB
- network: one virtual network interface

The virtual network interface should be bridged to a host interface that is connected to a non-OpenFlow network.

VMware vSphere Client:

Install the vSphere Client tool (included with ESXi) onto a Windows machine to administer the controller VM running on your ESXi host.

VMware ovftool:

Install ovftool onto your laptop or server to transfer the controller virtual disk image to the ESXi host and install it.

1.2. Installing the controller

Download the controller archive `bigswitchcontroller.zip` into a directory, and unzip it. The files in the archive will be extracted into a new subdirectory called `bigswitchcontroller-VERSION` where `VERSION` is the full version identifier of the controller virtual disk image.

Assuming you have configured your VMware ESXi host `my.esx.host` with a virtual network called `My Control Network`, and that you can log into the host as `root`, run `ovftool` as follows:

```
$ ovftool --name=bigswitchcontroller1 --net:bridged="My Control
Network" bigswitchcontroller.vmx vi://root@my.esx.host
Opening VMX source: bigswitchcontroller.vmx
Opening VI target: vi://root@my.esx.host/
Deploying to VI: vi://root@my.esx.host/
Disk Transfer Completed
Completed successfully
```

This will transfer the controller virtual disk image to the ESXi host and create a new VM named `bigswitchcontroller1`.

If you don't remember the name of your virtual network, run the same `ovftool` command without the `--net` option to see a list of all networks on the ESXi host (target):

```
$ ovftool bigswitchcontroller.vmx vi://root@my.esx.host
Opening VMX source: bigswitchcontroller.vmx
Opening VI target: vi://root@my.esx.host/
Error: No network mapping specified
OVF networks:
  bridged
Target networks:
  My Control Network
  Some Other Network
```

After you finish transferring the virtual disk image and creating the VM, open vSphere Client and locate the controller VM in the inventory. Power on the controller VM, and switch to the Console tab to access the VM console.

1.3. Alternative VM platforms

This section contains tips for running the controller on an alternative VM platform, for users who are expert in one of these platforms. If you are using VMware ESXi, skip to Initial bringup and sanity check below.

1.3.1. KVM (libvirt and virt-manager)

First use your Linux distribution's tools for setting up a bridge interface on the host; these instructions assume that a bridge `br0` exists with the correct physical network interface enslaved. To create bridge `br0` on an Ubuntu or Debian server with physical interface `eth0`, modify `/etc/network/interfaces`, replacing the `eth0` configuration with something like:

```
# remove the existing eth0 interface
auto br0
iface br0 inet static
# copy your existing address, netmask, gateway here
bridge_ports eth0
```

and then run `sudo /etc/init.d/networking restart` to apply the changes. Note this will not work if the address info is present on both eth0 and br0. This can also work with dhcp.

You may need to install virtinst tools:

```
$ sudo apt-get install virtinst
```

To create the controller VM, run the following commands (substituting version for some identifier that keeps VM disk images and domain definitions separate). It is important to make sure you do not use the same name for these images: doing so can lead to multiple VMs sharing a single disk image, and that will lead to disk corruption.

```
$ sudo cp bigswitchcontroller.vmdk
/var/lib/libvirt/images/bigswitchcontroller-<version>.vmdk
$ sudo virt-install --connect qemu:///system --virt-type kvm \
  --import --noautoconsole --name bigswitchcontroller-<version>
--ram 1024 \
  --network bridge=br0,model=virtio \
  --disk
path=/var/lib/libvirt/images/bigswitchcontroller-<version>.vmdk,forma
```

If you get an error message `does not support domain type 'kvm'`, check that the kernel modules are loaded by running `lsmod | grep kvm`. You should see at least `kvm` and one of `kvm_intel` or `kvm_amd` depending on the host CPU.

If you are setting up KVM on a machine for the first time, make sure the CPU supports virtualization extensions and that they are enabled in the BIOS. Many machines older than 2007 do not support KVM.

See the KVM FAQ for troubleshooting tips.

1.3.2. KVM (Proxmox VE)

Copy bigswitchcontroller.vmdk to the Proxmox host, and log into the host as root via SSH. Choose an unused VMID and MAC address for the controller VM. On the host, run:

```
# mkdir -p /var/lib/vz/images/VMID
# cp bigswitchcontroller.vmdk /var/lib/vz/images/VMID/
# qm create VMID --name bigswitchcontroller --onboot yes \
  --memory 1024 --ostype l26 --vlan0 virtio=MAC --bootdisk ide0 \
  --ide0 local:VMID/bigswitchcontroller.vmdk
```

Be sure to substitute your chosen values for VMID and MAC above.

1.3.3. VMware Workstation, Fusion, or Player

The controller archive includes a configuration file `bigswitchcontroller.vmx` that can be opened directly in VMware Worksation, Fusion or Player.

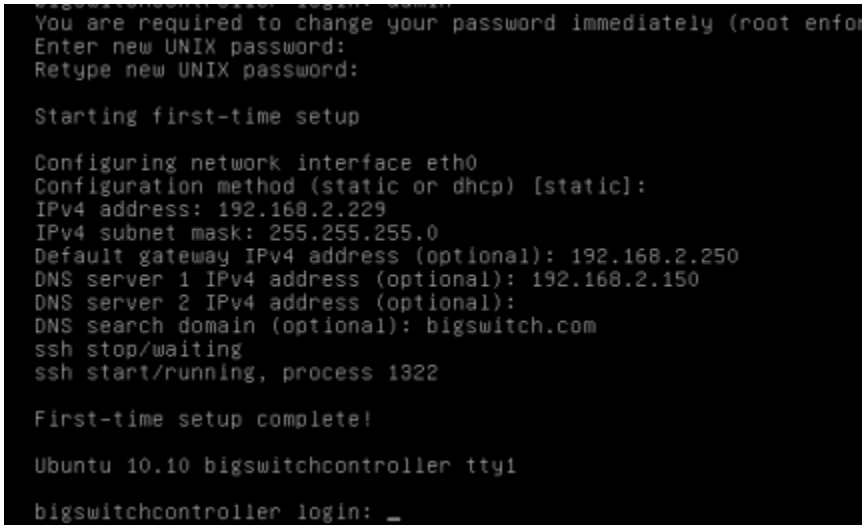
1.3.4. Microsoft Hyper-V

Convert bigswitchcontroller.vmdk to VHD format using Microsoft System Center Virtual Machine Manager or a third-party tool such as WinImage.

Using the Hyper-V Manager, create a new virtual machine. In the setup wizard, set the RAM size to 1024 MB. Attach the virtual hard disk to the converted VHD image. Leave other settings unchanged. When the wizard completes, remove the existing network adapter (if any) and create a new "Legacy Network Adapter" (Big Switch Controller does not currently include Hyper-V a driver for the normal Hyper-V network adapter).

1.4. Initial bringup and check

The first time you boot Big Switch Controller, log in as user `admin` on the main console (no password is required). You will be prompted to set the `admin` login password and configure the management network interface (eth0) as well as configure the timezone and other options. For example:



Now log in as user `admin` and run some commands to check the system is up and running. For example:

```
192.168.2.129> show version
Big Switch Controller 0.1
(2011.03.28.1314-build.bigswitchcontroller.release)
192.168.2.129> show interfaces
eth0      Link encap:Ethernet  HWaddr 52:54:00:25:e3:9b
inet addr:192.168.2.129  Bcast:192.168.2.255  Mask:255.255.255.0
inet6 addr: fe80::5054:ff:fe25:e39b/64 Scope:Link
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:218435 errors:0 dropped:0 overruns:0 frame:0
TX packets:136260 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:34582552 (34.5 MB)  TX bytes:27675992 (27.6 MB)
```

```
lo        Link encap:Local Loopback
inet addr:127.0.0.1  Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING  MTU:16436  Metric:1
RX packets:1614375 errors:0 dropped:0 overruns:0 frame:0
TX packets:1614375 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:457096230 (457.0 MB)  TX bytes:457096230 (457.0 MB)


192.168.2.129> show switch
Switch DPID             Alias         Active Last Connect Time
IP Address    Socket Address       Max Packets Max Tables
----------------------|-------------|------|------------------|----
pronto-3290   False  2011-03-28 15:24:52 192.168.2.31
/192.168.2.31:33083  256         3
00:00:00:00:00:37:80:01 pronto-3780   True   2011-03-28 15:24:51
192.168.2.32  /192.168.2.32:47260  256         3
00:00:00:00:00:73:28:02 netgear-1     True   2011-03-28 15:24:52
192.168.2.28  /192.168.2.28:40924  256         3
00:00:00:00:00:73:52:01 netgear-2     True   2011-03-28 15:24:51
192.168.2.52  /192.168.2.52:57690  256         3
00:00:00:00:00:d0:d0:03 wireless-ap-1 True   2011-03-28 15:24:56
192.168.2.34  /192.168.2.34:57244  256         2
00:00:00:00:00:d0:d0:05 wireless-ap-2 True   2011-03-28 15:24:56
192.168.2.35  /192.168.2.35:54597  256         2
00:00:00:30:48:f9:cd:0b vswitch       True   2011-03-28 15:24:53
192.168.2.152 /192.168.2.152:44946 256         2
00:0a:00:21:f7:de:e9:00 hp-5400       True   2011-03-28 15:25:10
192.168.2.102 /192.168.2.102:62122 256         2
00:0a:00:24:a8:c4:69:00 hp-6600       True   2011-03-28 15:25:10
192.168.2.101 /192.168.2.101:43138 256         2


192.168.2.129> show switch 00:0a:00:24:a8:c4:69:00 desc
Switch  Serial #   Vendor  Make
Model      SW Version
-------|----------|-------|-----------------------------------|----
CN9202M063 HP-Labs HP Procurve 5400zl/3500yl/6600 series
hp6600-101 2.02v

192.168.2.129> show switch 00:0a:00:24:a8:c4:69:00 flow
Switch  Bytes Packets Dur(s) Cookie  In Port Src MAC           Dst
MAC           Ether Type Src IP         Dst IP         Protocol
Src Port Dst Port
-------|-----|-------|------|-------|-------|-----------------|------
102   1       3      lswitch 29 (5)  00:11:43:d8:7a:62
c4:2c:03:02:5a:11 ip(2048)   199.59.148.139 192.168.10.237 *
*        *
hp-6600 0     0       4      lswitch 43 (19) c4:2c:03:02:5a:11
```

```
00:11:43:d8:7a:62 arp(2054)  192.168.10.237 192.168.10.1   *
*        *
hp-6600 78    220007  6952   lswitch 29 (5)  00:11:43:d8:7a:62
10:9a:dd:4b:c4:dd ip(2048)   192.168.2.152  192.168.10.248 *
*        *
hp-6600 0     1       1      lswitch 43 (19) 10:9a:dd:4b:e5:cb
00:11:43:d8:7a:62 ip(2048)   192.168.10.242 98.136.48.221  *
```

```
*           *
```

1.5. Connecting your switches to the controller

The OpenFlow connection between a switch and the Big Switch Controller is initiated by the switch. The switch configuration steps will vary based on the type of switch - essentially, the switch needs to be configured with the IP address of the Big Switch Controller and the port (6633) that the OpenFlow daemon is listening to on the controller.

1.6. Upgrading an existing controller

Beginning with version 2011.03.28 of Big Switch Controller, the virtual disk has room for two complete installations of the controller software, in separate partitions. Initially partition 1 contains the controller software and partition 2 is empty. You can install a new release into the unused partition of an existing controller rather than installing a new virtual machine from scratch.

The Big Switch Controller upgrade feature

- reduces the need to interact with the virtual machine platform configuration interface, which may be accessible only by a separate administrative staff
- migrates first-time setup parameters to the new installation: network interface configuration (IP address, gateway, etc.), admin user password, hostname, NTP server, time zone, SSH host keys
- minimizes downtime by performing most of the upgrade steps in the background while the controller continues to work normally; the final reboot step usually takes less than a minute
- keeps the previous installation available as a fail-safe (contact Big Switch support for assistance with rolling back to a previous installation)

1.6.1. Exporting configuration data

The upgrade process does not migrate configuration data from the existing installation to the new one, except for the first-time setup parameters listed above. Therefore whether you use the upgrade feature or reinstall the controller from scratch, if you want to preserve the controller's configuration, you must export and store it elsewhere prior to the upgrade and reapply the configuration afterwards.

Currently the configuration data you may want to save include static flow table entry definitions, switch aliases and host aliases. To display or export this configuration data, run the `show running-config` command. This output should be saved in a file outside the controller.

```
192.168.2.129# show running-config
!
! Big Switch Controller 0.1
(2011.03.28.1314-build.bigswitchcontroller.release)
!
!
switch 00:00:00:00:00:00:00:01
  flow-entry deny-h11-h31
    active True
    ether-type 2048
    src-ip 10.0.0.11
    dst-ip 10.0.0.31
!
switch 00:00:00:00:00:00:00:02
!
switch 00:00:00:00:00:00:00:03
```

1.6.2. Performing the upgrade

(log in as admin; stash away current running-config; debug bash; scp or wget bigswitchcontroller-upgrade.pkg; sudo upgradebsc bigswitchcontroller-upgrade.pkg; look for "successfully installed" message; sudo reboot)

1.6.3. Troubleshooting

(debug using the vm console if the controller is unreachable after rebooting)

1.6.4. Restoring configuration data

Once the new controller is up and running, use the copy command with a URL argument (either an FTP server or an HTTP server) to copy the file back into the running-config.

1.7. New to this version, known issues, and tickets

New to this version:

This version of the controller now supports an in-VM upgrade (but only from version 2011.04.05.1050-build.bigswitchcontroller.release/). If you have an earlier version that 2011.04.05, then you must reinstall the entire controller VM.

This major additions to this version are:

- "show host" now indicates whether a host has been found at multiple places in the network. This happens in a network with a mix of OpenFlow-enabled and non-OpenFlow-enabled switches.
- "trace" command and "show switch <dpid> trace" command that shows controller <-> switch messages in varying levels of detail
- statistics collection and graphing - the controller now collects a few set of statistics, including the number of OpenFlow packet-in messages received from switches and the number of flow-mods

sent by the controller to the switches. It also tracks system information such as memory usage and CPU utilization. These statistics are presented in a web-based graphical user interface.

Deployment Considerations:

The Big Switch Controller has support for VLANs and mixed networks with a combination of OpenFlow-enabled and non-OpenFlow switches. There are a number of considerations to review when deploying with VLANs or in a mixed network:

- What VLAN mode the switch is configured for - in the HP switches, either virtualized mode or aggregation mode can be enabled.
- Whether links are tagged or untagged
- Whether spanning tree is enabled and whether the spanning tree information is available across a mixed network.

We recommend a quick overview of your network/VLAN topology before deploying the controller into mixed networks or networks with VLANs.

Known issues:

Tickets:

- #1 - flow entry name completion in config mode completes flow entries for other switches
- #13 - deleting a switch doesn't delete flow entries
  - The workaround is to delete the flow entries first and then delete the switch
- #48 - log files can be too large for show logging
  - The workaround is to use the "debug bash" command and cd to /opt/bigswitch and find . -name *.log to get the specific log file.
- #55 - switch active status in "show switch" may not be accurate
  - The "show switch realtime" command shows accurately what switches are connected to the controller.
- #63 - controller send no reply to HELLO message
- #64 - display shows incorrect byte values on NEC switches

2. Concepts

2.1. OpenFlow

OpenFlow defines a way for networking switches to receive detailed packet-handling instructions from an external controller. These instructions have *match rules* that instruct the switch to match specific frames, and *actions* that instruct the switch what actions to take on those packets.

The match rules can match a variety of layer 2 and layer 3 fields in the header, including source/destination MAC addresses, source/destination IP addresses, port numbers, and others. The rules

can also *wildcard* any of these fields (called *wide* flow entries as they can match lots of different traffic), or the rules can specify a value for every field (called *narrow* flow entries as they can match only very specific traffic).

The actions can forward to a port or multiple ports or to drop the packet. Before forwarding, the switch can rewrite parts of the header.

In order for a switch to receive these instructions (called *flow entries* or *flow mods*), the switch must have OpenFlow enabled - effectively, it must implement the agent side of the OpenFlow protocol. An external controller implements the controller side of the OpenFlow protocol.

A switch can also ask the controller how to handle a given packet for which it has no matching rule. In response, a controller can tell the packet how to forward the packet, and it can dynamically insert a flow mod to the switch (or even multiple flow mods to multiple switches) tells the switch how to handle subsequent packets.

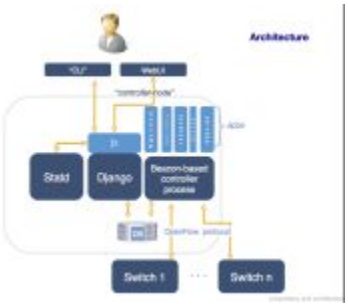
Based on OpenFlow's simple concept, a controller can implement a variety of applications/logic to flexibly provide connectivity and security in a network.

For more information, please refer to the OpenFlow 1.0 specification.

2.2. Big Switch Controller architecture

Definitions:

- **controller system:** a set of cooperating controller nodes that are managing a set of switches in a network.
- **controller node:** refers to a single virtual machine that runs the controller software. In this release, a controller-system can only have one controller node.
- Beacon-based controller process or just **Beacon:** the component in the controller node that speaks the OpenFlow protocol to the switches. This component is based on the open-source, Java-based Beacon controller. Beacon also allows different applications to plug into it to extend the functionality of the controller.
- **CLI:** the component that allows for configuration of the controller and viewing of data from the switches and the controller.
- **WebUI:** the component that allows for viewing of statistics collected from the switches and the controller.
- **Django:** the component that provides a REST API that is used by the CLI and can be leveraged by third parties for integration.
- **DB:** persistent storage of configuration as well as observed hosts, links, switches, and ports and their accompanying statistics.
- **Statd:** the component that collects system and network statistics.



2.3. Core applications

A number of applications are enabled by default in the controller. The most relevant ones from the user's perspective are the following:

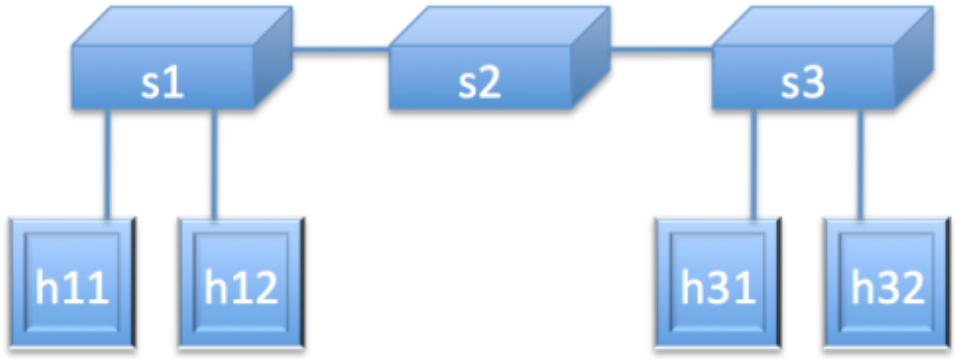
- Learning Switch
- Device Manager
- Topology
- Routing
- Static Flow Entry

To check the status of all these applications, use the `show bundles` command:

```
10.0.2.15# show bundles | grep -e topo -e device -e outing -e
static -e learning
Active   1  net.beaconcontroller.routing
Active   3  net.beaconcontroller.learningswitch
Active   7  net.beaconcontroller.routing.apsp
Active   16 net.beaconcontroller.devicemanager
Active   23 net.beaconcontroller.topology
Active   37 com.bigswitch.beacon.staticflowentry
```

The applications can be started and stopped using the `bundles` command. However, this should not be done unless instructed by Big Switch Networks. By default, all applications should be enabled. Stopping/starting applications will have an impact on the correctness of forwarding in the network.

The applications will be covered below and will refer to the following example network:



2.3.1. Learning Switch

The Learning Switch application allows any switch that is connected to the controller to support forwarding by learning the ports for given hosts on the network and inserting flow entries into the switch on demand.

When a switch sees a packet for which it has no rule, it then gives that packet to the controller for examination. The Learning Switch application looks in its own per-switch tables and learns the source MAC address is located on the ingress port of the packet. The application then looks to see if it knows the port for the destination MAC address. If it does, then it creates a flow mod that is an exact match for the packet and it inserts that flow mod into the switch with a very low timeout (less than 10 seconds). If it does not know where the destination MAC address lives, it floods the packet out to all ports.

In the example above, when h11 pings h12, the following flow mods can be seen on the s1 switch:

```
10.0.2.15# show switch 00:00:00:00:00:00:00:01 flow brief
Cookie  Src MAC           Dst MAC           Src IP    Dst IP
Protocol Src Port    Dst Port
-------|-----------------|-----------------|---------|---------|-----
00:00:00:00:00:0c 00:00:00:00:00:0b 10.0.0.12 10.0.0.11 icmp     0
lswitch 00:00:00:00:00:0b 00:00:00:00:00:0c 10.0.0.11 10.0.0.12
icmp     8
```

Note the `Cookie` column that shows the source of the flow mod as being the Learning Switch application.

2.3.2. Device Manager

The Device Manager application is a monitoring application that learns about host locations across the network and inserts host information into the database. The hosts can be viewed from the CLI using the `show host` command. The host information is also used by the Routing application.

Warning: The Routing application will not function properly if the Device Manager application is not started, and packets will not be forwarded if Routing is started without Device Manager.

When the Device Manager application is running, hosts will be populated in the database like below:

```
10.0.2.15(config)# show host
MAC Address       Switch ID                Ingress Port IP Address
Name
------------------|------------------------|------------|----------|---
00:00:00:00:00:00:00:01 1             0.0.0.0
00:00:00:00:00:0c 00:00:00:00:00:00:00:01 2             0.0.0.0
00:00:00:00:00:1f 00:00:00:00:00:00:00:03 1             0.0.0.0
00:00:00:00:00:20 00:00:00:00:00:00:00:03 2             0.0.0.0
```

Note that host 11 (which has a MAC address of 0b) is located on switch 1, port 1 (as indicated in our topology). Similarly, host 31 (MAC address of 1f) is located on switch 3, port 1.

2.3.3. Topology

The Topology application is a monitoring application that learns about link information connecting the switches and inserts that link information into the database. The links can be viewed from the CLI using the `show link` command. The link information is also used by the Routing application (just as the host information is).

Warning: The Routing application will not function if the Topology application is not started, and packets will not be forwarded if Topology is started without Topology.

When the Topology application is running, links will be populated in the database like below:

```
10.0.2.15(config)# show link
Src Switch DPID         Src Port Dst Switch DPID         Dst Port
----------------------|--------|----------------------|--------
00:00:00:00:00:00:00:01 3        00:00:00:00:00:00:00:02 1
00:00:00:00:00:00:00:02 1        00:00:00:00:00:00:00:01 3
00:00:00:00:00:00:00:02 2        00:00:00:00:00:00:00:03 3
00:00:00:00:00:00:00:03 3        00:00:00:00:00:00:00:02 2
```

2.3.4. Routing

Note: the Routing application is temporarily disabled for this release. Forwarding across the network will still function, but the system will rely on the learning switch application for this.

The Routing application uses the host and link information from the Device Manager application and the Topology application to develop a graph of the network and its connectivity. When a switch sees a packet for which it has no rule, it forwards the packet to the controller, and the Routing application will compute the shortest path across the network for the packet to reach its destination. With the Routing application, the controller is serving as a centralized routing table for the network.

In the example above, if pings are initiated from all hosts to all other hosts, then a combination of flow mods from the Learning Switch application and the Routing application will be inserted into the switches.

For example, below is the output from issuing `show switch <dpid> flow brief` for all three switches in the example network. Note switch 1 and switch 3 have a combination of flow mods marked `routing` and `lswitch` while switch 2 has flow mods only with `lswitch`.

Executing

```
show switch 00:00:00:00:00:00:00:01 flow brief; show switch
00:00:00:00:00:00:00:02 flow brief; show switch
00:00:00:00:00:00:00:03 flow brief
Cookie  Src MAC           Dst MAC           Src IP    Dst IP
Protocol   Src Port Dst Port
-------|-----------------|-----------------|---------|---------|-----
00:00:00:00:00:1f 00:00:00:00:00:0b 10.0.0.31 10.0.0.11 arp-rep(2)
0        0
routing 00:00:00:00:00:20 00:00:00:00:00:0b 10.0.0.32 10.0.0.11
arp-rep(2) 0        0
routing 00:00:00:00:00:1f 00:00:00:00:00:0c 10.0.0.31 10.0.0.12
arp-rep(2) 0        0
routing 00:00:00:00:00:20 00:00:00:00:00:0c 10.0.0.32 10.0.0.12
arp-rep(2) 0        0
routing 00:00:00:00:00:0c 00:00:00:00:00:0b 10.0.0.12 10.0.0.11
arp-rep(2) 0        0
lswitch 00:00:00:00:00:0c 00:00:00:00:00:1f 10.0.0.12 10.0.0.31
arp-req(1) 0        0
lswitch 00:00:00:00:00:0c 00:00:00:00:00:20 10.0.0.12 10.0.0.32
arp-req(1) 0        0
routing 00:00:00:00:00:0b 00:00:00:00:00:0c 10.0.0.11 10.0.0.12
arp-req(1) 0        0
lswitch 00:00:00:00:00:0b 00:00:00:00:00:1f 10.0.0.11 10.0.0.31
arp-req(1) 0        0
lswitch 00:00:00:00:00:0b 00:00:00:00:00:20 10.0.0.11 10.0.0.32
arp-req(1) 0        0
Cookie  Src MAC           Dst MAC           Src IP    Dst IP
Protocol   Src Port Dst Port
-------|-----------------|-----------------|---------|---------|-----
00:00:00:00:00:1f 00:00:00:00:00:0b 10.0.0.31 10.0.0.11 arp-rep(2)
0        0
lswitch 00:00:00:00:00:20 00:00:00:00:00:0b 10.0.0.32 10.0.0.11
arp-rep(2) 0        0
lswitch 00:00:00:00:00:1f 00:00:00:00:00:0c 10.0.0.31 10.0.0.12
arp-rep(2) 0        0
lswitch 00:00:00:00:00:20 00:00:00:00:00:0c 10.0.0.32 10.0.0.12
arp-rep(2) 0        0
lswitch 00:00:00:00:00:0b 00:00:00:00:00:1f 10.0.0.11 10.0.0.31
arp-req(1) 0        0
lswitch 00:00:00:00:00:0b 00:00:00:00:00:20 10.0.0.11 10.0.0.32
arp-req(1) 0        0
lswitch 00:00:00:00:00:0c 00:00:00:00:00:1f 10.0.0.12 10.0.0.31
arp-req(1) 0        0
lswitch 00:00:00:00:00:0c 00:00:00:00:00:20 10.0.0.12 10.0.0.32
```

```
arp-req(1) 0        0
Cookie  Src MAC           Dst MAC           Src IP    Dst IP
Protocol   Src Port Dst Port
-------|-----------------|-----------------|---------|---------|-----
00:00:00:00:00:0b 00:00:00:00:00:1f 10.0.0.11 10.0.0.31 arp-req(1)
0        0
routing 00:00:00:00:00:0b 00:00:00:00:00:20 10.0.0.11 10.0.0.32
arp-req(1) 0        0
routing 00:00:00:00:00:0c 00:00:00:00:00:1f 10.0.0.12 10.0.0.31
arp-req(1) 0        0
routing 00:00:00:00:00:0c 00:00:00:00:00:20 10.0.0.12 10.0.0.32
arp-req(1) 0        0
lswitch 00:00:00:00:00:20 00:00:00:00:00:0b 10.0.0.32 10.0.0.11
arp-rep(2) 0        0
lswitch 00:00:00:00:00:20 00:00:00:00:00:0c 10.0.0.32 10.0.0.12
arp-rep(2) 0        0
routing 00:00:00:00:00:20 00:00:00:00:00:1f 10.0.0.32 10.0.0.31
arp-req(1) 0        0
lswitch 00:00:00:00:00:1f 00:00:00:00:00:0b 10.0.0.31 10.0.0.11
arp-rep(2) 0        0
lswitch 00:00:00:00:00:1f 00:00:00:00:00:0c 10.0.0.31 10.0.0.12
arp-rep(2) 0        0
```

```
routing 00:00:00:00:00:1f 00:00:00:00:00:20 10.0.0.31 10.0.0.32
arp-rep(2) 0        0
```

2.3.5. Static Flow Entry

The Static Flow Entry application inserts statically defined flow entries created by the user into the switches. Flow entries can be defined in the CLI with custom priority and cookie fields, and matches can include wildcards for any fields (support for both *narrow* and *wide* flow entries). Actions can be a list of multiple output actions.

The Static Flow Entry application can override or supplement the default connectivity provided by the Learning Switch and Routing applications.

For example, in the network above, the Learning Switch application can provide connectivity for all hosts to all others hosts. However, suppose the user wanted to block pings from h11 to h31. The following flow entry could be defined:

```
10.0.2.15(config)# switch 00:00:00:00:00:00:00:01
10.0.2.15(config-switch)# flow-entry deny-h11-h31
10.0.2.15(config-flow-entry)# src-ip 10.0.0.11
10.0.2.15(config-flow-entry)# dst-ip 10.0.0.31
10.0.2.15(config-flow-entry)# ether-type 2048
10.0.2.15(config-flow-entry)# active True
10.0.2.15(config-flow-entry)# end
```

Recall that the OpenFlow protocol specifies that if no action is specified, as in this flow entry, packets matched by the flow are dropped.

This static flow entry would appear like this on the switch:

```
10.0.2.15# show switch 00:00:00:00:00:00:00:01 flow brief
Cookie              Src MAC Dst MAC Src IP    Dst IP    Protocol
Src Port Dst Port
-------------------|-------|-------|---------|---------|--------|----
*       *       10.0.0.11 10.0.0.31 *        *        *
```

Now traffic between h11 and h31 is blocked.

The Static Flow Entry application works by continually reasserting the flow entries to the switches every 10 seconds with a hard timeout of 30 seconds. If a flow entry is modified or deleted, then the old flow entry will expire in 30 seconds. In this way, the application ensures only the intended flow entries are on the switches.

Note that because of this design, the packet/byte counters for the static flow entries are continually reset. This limitation will be addressed in future releases.

2.4. Statd

Statd is a daemon module running within the controller VM to collect system and OF network statistics. Statd was designed with extensibility in mind. It is a simple plugin framework with four types of pluggable python modules, collectors, formatters, outputters, and throttlers.

- Collectors implement a collect() function and publish data by invoking either addControllerData() or addSwitchData() functions. Currently, two collectors are built into the VM.
  - osStats is an OS statistics collector that gathers CPU, memory, and swap statistics.
  - ofStats is an Open-Flow statistics collector that gets the number packetIns and flowMods per switch.
- Formatters implement a format() function and return the formatted string.
  - JsonFormatter is the lone formatter supported in this release.
- Outputters implement a post() function.
  - RestOutput is the lone outputter supported in this release.
- Throttler implement a throttle() function that throttles the amount of event data that is published. Throttlers are assigned to Outputters and are optional. A throttler name and rate is required.
  - LeakyBucketThrottler is the lone throttler supported in this release.

/opt/bigswitch/statd/statd.conf is the configuration file used by the statd to determine its behavior. Here is a sample statd.conf:

```
{
    "logLevel": "info",{    "logLevel": "info",
    "logFile": "/opt/bigswitch/statd/log/statd.log",
    "input": [
        {
            "name":"OsStats",
            "period":60
        },
        {
            "name":"OfStats",
            "period":5
        },
        {
            "name":"LogCollector",
            "period":10,
            "level":"info",
            "module":"beacon"
        }
    ],
    "output": [
        {
            "name":"RestOutput",
            "period":15,
            "formatter":"Json",
            "throttler":"LeakyBucketThrottler",
            "stats":"v1/stats/default",
            "log":"v1/events/default"
        }
    ],
    "formatter": [
        {
            "name":"Json"
        }
    ],
    "throttler": [
        {
            "name":"LeakyBucketThrottler",
            "rate":10
        }
    ]
}
```

The "period" field in each entry is in seconds and determines the execution frequency for each module.

3. WebUI reference

3.1. Introduction

In the current release, webUI only supports the viewing of system and OF-related statistics.

On the top of each page, a time range can be selected by specifying the start and end time, or one of the shortcut links, 1hr, 1day, and 1wk, which means the last selected interval (i.e. 1hr means the last hour). A "Refresh" button is at the top right corner to submit the request.

The system statistics include two time-series charts, Memory Usage and CPU Usage. Memory usage chart trends the VM's used memory, free memory, and used swap over a selected period of time. The y-axis is in absolute amount. CPU usage chart trends the VM's cpu usages such as system, user, nice, and idle. The y-axis is in percentage.

The OF statistics include packetIn rate and flow-mod rate per switch over a specified period of time and the corresponding CDF chart. In addition to the time selector on the top of the page, a drop-down switch selector is added to the page to specify the target switch.

3.2. Setup

The controller's web server listens on TCP port 8000. By default, the VM's firewall blocks the port. The controller VM's administrator must open the port in order to access the web UI.

The procedure to open the firewall:

1. login as admin

2. open the linux shell, "debug bash"

3. permit port 8000, "sudo ufw allow 8000/tcp"

3.3. Screen Captures

System Graphs
OpenFlow Graphs

4. CLI reference

4.1. CLI concepts

4.1.1. Access and modes

The CLI can be accessed through the console window of the virtual machine or by using ssh to connect to the virtual machine. Logging in as the admin user gives access to the CLI.

The admin user will be logged into the CLI in *login mode*. The prompt will have a > to indicate this mode.

Enter *enable mode* by typing the `enable` command. The prompt will have a # to indicate this mode.

Enter *config mode* by typing the `configure` command. The prompt will have a `(config)#` to indicate this mode.

There are a number of *config submodes* which will be indicated in the prompt as `(config-<submode>)#`, and each submode allows configuration of a specific type of object in the database.

Type `exit` to return to the previous mode, or type `end` to exit all config modes and return to enable mode.

4.1.2. Navigation and help

There are a number of ways users can navigate the CLI and receive help at any time.

For navigation, the CLI implements a Linux/shell-style navigation - for example:

- Ctrl-B - back one character
- Ctrl-F - forward one character
- Ctrl-A - move to the start of the line
- Ctrl-E - move to the end of the line
- Ctrl-P - display the previous command - can be repeated to go through history
- Ctrl-R - search for text among previous commands

Refer to http://tiswww.case.edu/php/chet/readline/readline.html for more information on all keyboard shortcuts and facilities.

Help is also accessible by typing the command `help`. The output will vary based on the specific mode the user is in.

Command completion at any point can be pressing tab once or twice. Pressing tab once will complete the value if possible, and pressing tab twice will show all possible completions. Type the `?` character to show completions at any time.

The CLI will accept shortened versions of commands and options so long as there is no ambiguity in the commands. For example, `sh run` will be interpreted as `show running-config`.

4.1.3. CLI conveniences: pipes, watching commands, and other tricks

The output of any CLI command can be piped to common Unix shell utilities such as `grep`, `awk`, `wc`, `tail`, `more`, or `less`. This can make searching for data or browsing through output significantly easier. For example, to find all the ports of a given switch, type:

```
10.0.2.15> show port | grep :24
00:0a:00:24:a8:c4:69:00 52    28    00:24:a8:c4:69:cc
00:0a:00:24:a8:c4:69:00 54    30    00:24:a8:c4:69:ca
00:0a:00:24:a8:c4:69:00 56    32    00:24:a8:c4:69:c8
00:0a:00:24:a8:c4:69:00 57    33    00:24:a8:c4:69:c7
00:0a:00:24:a8:c4:69:00 58    34    00:24:a8:c4:69:c6
10.0.2.15>
```

The CLI provides `begin`, `include`, and `exclude` as pipe options that may be familiar to administrators of existing network devices. `begin` is useful for starting to view large output at a particular point. For example, to see just the `lsof` output of show tech-support, type:

```
10.0.2.15> show tech-support | begin lsof |more
Executing os command: sudo lsof
COMMAND    PID       USER   FD      TYPE             DEVICE
SIZE/OFF       NODE NAME
init         1       root  cwd       DIR               8,16
4096          2 /
init         1       root  rtd       DIR               8,16
4096          2 /
init         1       root  txt       REG               8,16
125640       4081 /sbin/init
init         1       root  mem       REG               8,16
51712      19738 /lib/libnss_files-2.12.1.so
init         1       root  mem       REG               8,16
43552      26679 /lib/libnss_nis-2.12.1.so
init         1       root  mem       REG               8,16
97256      19579 /lib/libnsl-2.12.1.so
...
```

The CLI also allows users to repeatedly invoke a command by prepending the command `watch`. For example, `watch show switch <dpid> flow` will allow the user to monitor the flows on a specific switch. This may be familiar to Unix users who use top or the corresponding `watch` command in Unix.

Multiple CLI commands can be entered into the CLI on a single line, separated by a semi-colon. For example, to get all the way into a configuring a flow entry on a specific switch, type:

```
10.0.2.15> enable; conf; switch 00:0a:00:24:a8:c4:69:00;
flow-entry foo
10.0.2.15(config-flow-entry)#
```

Finally, the CLI allows redirecting the output of a command to a local file or to the URL of an HTTP server or an FTP server. The local file is stored in the controller database (as general access to the local filesystem is not given). This is convenient for storing away show output or versions of configurations. For example, to store the output of show host, type:

```
10.0.2.15# show host > localhost://hostoutput
user data created
10.0.2.15# show config
Name       Length Version Timestamp
----------|------|-------|-------------------
hostoutput 1775   1       2010-12-15.08:29:31

10.0.2.15# show config hostoutput | head -10
MAC Address       Name Switch ID               Ingress Port IP
Address     Vendor
-----------------|----|----------------------|------------|---------
00:0a:00:24:a8:c4:69:00 38           192.168.10.207 VMware, Inc.
00:21:f7:de:e9:00      00:0c:00:21:f7:de:e9:00 local(65534)
192.168.12.102 ProCurve Networking by HP
00:25:90:08:94:d5      00:0a:00:21:f7:de:e9:00 2
128.8.109.145  Super Micro Computer, Inc.
00:30:48:f9:cc:f7      00:00:00:30:48:f9:cc:f7 local(65534)
192.168.2.99   Supermicro Computer, Inc.
00:30:48:f9:cd:0b      00:00:00:30:48:f9:cd:0b local(65534)
0.0.0.0        Supermicro Computer, Inc.
00:30:48:f9:cd:0d      00:00:00:00:00:00:00:01 17
192.168.11.2   Supermicro Computer, Inc.
00:50:8d:65:be:4b      00:00:00:00:00:00:00:01 22
192.168.11.1   ABIT COMPUTER CORPORATION
10:00:00:67:ad:07      00:00:00:30:48:f9:cc:f7 4
192.168.12.132 unknown
```

4.1.4. Conventions

Text in courier font indicates either text that should be typed in by the user or output from a command.

Text enclosed in parentheses ( ) indicates a set of required arguments. The options are separated by a vertical bar |.

Text enclosed in square brackets [ ] indicates optional arguments. The options are separated by a vertical bar |.

Text enclosed in angle brackets < > indicates a value that should be entered/substituted by the user.

4.2. Login mode commands

4.2.1. date

S y n t a x :

```
date
```

Examples:

```
192.168.2.129> date
2011-03-29 21:04:59
192.168.2.129>
```

Description:
This command prints out the current date/time of the controller node. This is useful for interpreting log output or timestamps in `show` commands.

4.2.2. debug

Syntax:
`debug ( bash | python | netconfig )`

Examples:

```
10.0.2.15> debug bash

***** Warning: this is a debug command - use caution! *****
***** Type "exit" or Ctrl-D to return to the BigOS CLI *****

bsn@bigswitchcontroller:~$
```

Description:

This command gives access to a number of debug-level commands in the controller node. The `bash` and `python` options run the respective Unix command shell that allows access to the internals of the controller node. The `netconfig` reruns the first-time networking setup script. After running netconfig, you should restart the controller.

This command should be used as advised by Big Switch Networks.

4.2.3. enable

Syntax:
`enable`

Examples:

```
10.0.2.15> enable
10.0.2.15#
```

Description:
This command puts the user into enable mode. This mode gives the user access to more commands and capabilities, including copying configurations to be stored on the controller and changing the configuration of the controller. The CLI prompt will change from a > to a # - see above example.

Typically, network devices require a password to enter enable mode. In this release of the controller, passwords are not supported.

4.2.4. exit

Syntax:

```
exit
```

Examples:

```
10.0.2.15(config)# exit
10.0.2.15#
10.0.2.15# exit
10.0.2.15>
```

Description:

This command leaves the current mode and returns the user to the previous mode. In the example above, the user is first leaving config mode and then leaving enable mode.

4.2.5. help

Syntax:

```
help [ <topic> ]
```

Examples:

```
10.0.2.15(config-switch)# help
For help on specific commands type help <topic>
Available commands:
  bundles
  copy
  date
  debug
  end
  exit
  help
  history
  logout
  no
  show
  write

Available config objects:
  flow-entry

Available fields for switch:
  actions
  active
  alias
  buffers
  capabilities
  connected-since
  controller
  dpid
  ip-address
  socket-address
  tables
```

Description:
This command lists the commands that are available in the current mode. The output will vary based on the current mode. In the example above, the user is in config-switch submode, so there are three types of commands that available: commands that can be executed directly (e.g., `copy` or `show`), objects that can be created (e.g., `flow-entry`), or fields for the current submode that can be configured.

4.2.6. logout

Syntax:
`logout`

Examples:

```
10.0.2.15> logout

Ubuntu 10.10 bigswitchcontroller tty1

bigswitchcontroller login:
```

Description:
This command logs out the user from the CLI session and returns either to the shell where the user connected from, or to the login prompt if at a console window.

4.2.7. show <keyword>

Syntax:
`show ( bundles | config | controller-node | flow-entry | host | interfaces | link | logging | port | running-config | startup-config | switch | tech-support )`

Examples:

```
10.0.2.15> show version
Big Switch Controller 0.1
(2011.03.28.1314-build.bigswitchcontroller.release)
10.0.2.15> show switch hp-6600 desc
Switch  Serial #   Vendor  Make
Model      SW Version
-------|----------|-------|-------------------------------------|----
CN9202M063 HP-Labs HP Procurve 5400zl/3500yl/6600 series
hp6600-101 2.02v
```

Description:
This command takes a variety of options that will be covered in subsections below. Several options display information about the system while other options display information about objects in the database (controller-node, flow-entry, host, link, port, and switch).

4.2.7.1. show bundles

Syntax:
`show bundles`

Examples:

```
192.168.2.129> sh bundles | grep beacon
Active     3  com.bigswitch.beacon.staticflowentry
Active     4  com.bigswitch.beacon.storage.cassandra
Active     5  com.bigswitch.beacon.storage.nosql
Active     13 net.beaconcontroller.core
Resolved   14 net.beaconcontroller.core.web
Active     15 net.beaconcontroller.devicemanager
Resolved   16 net.beaconcontroller.jetty.config
Active     17 net.beaconcontroller.learningswitch
Active     18 net.beaconcontroller.logging.bridge
Resolved   19 net.beaconcontroller.logging.config
Active     20 net.beaconcontroller.packet
Resolved   21 net.beaconcontroller.routing
Resolved   22 net.beaconcontroller.routing.dijkstra
Resolved   23 net.beaconcontroller.storage
Resolved   24 net.beaconcontroller.storage.memory
Active     25 net.beaconcontroller.topology
Resolved   26 net.beaconcontroller.topology.visualweb
Resolved   27 net.beaconcontroller.topology.web
Active     28 net.beaconcontroller.util
Active     29 net.beaconcontroller.web
```

D e s c r i p t i o n :

This command shows the underlying bundles that make up the Java Beacon process (see earlier architecture discussion). This command is useful in conjunction with the `bundles` command that allows starting and stopping of individual bundles, e.g., a deployment where flows are managed entirely via static flow entries and the default routing application is not required.

4.2.7.2. show config

S y n t a x :

```
show config [ all | <name> [ all | <version number> | diff <ver a> <ver b> ] ]
```

Examples:

```
10.0.2.15# show config
Name        Length Version Timestamp
-----------|------|-------|-------------------
startup     287    4       2010-12-14.21:29:26
mattspecial 254    1       2010-12-14.19:40:44
```

```
10.0.2.15# show config startup all
Name    Length Version Timestamp
-------|------|-------|-------------------
startup 132    2       2010-12-13.22:11:04
startup 287    4       2010-12-14.21:29:26
startup 132    3       2010-12-14.04:32:09
startup 132    1       2010-12-13.22:09:43
```

```
10.0.2.15# show config startup 3
!
! Big Switch Controller 0.1
(2011.03.28.1314-build.bigswitchcontroller.release)
!
!
switch 00:00:00:00:00:73:28:03
!
switch 00:0a:00:21:f7:de:e9:00
!
switch 00:0a:00:24:a8:c4:69:00
10.0.2.15# show config startup diff 3 4
4a5,6
> switch 00:00:00:00:00:00:00:01
> !
9a12,15
>   flow-entry ssh-block
>     active False
>     dst-port 22
>     actions drop
```

Description:
This command shows configuration files that have been saved on the controller. A configuration file can be a complete configuration (such as the special `startup` file) or a partial "configlet." The options allow listing versions for a particular file, viewing a specific version of a file, or diff'ing two versions of a file.

4.2.7.3. show interfaces

Syntax:
`show interfaces`

Examples:

```
192.168.2.229# show interfaces
eth0      Link encap:Ethernet  HWaddr 52:54:00:25:e3:9b
inet addr:192.168.2.229  Bcast:192.168.2.255  Mask:255.255.255.0
inet6 addr: fe80::5054:ff:fe25:e39b/64 Scope:Link
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:285120 errors:0 dropped:0 overruns:0 frame:0
TX packets:162749 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:44291880 (44.2 MB)  TX bytes:27358754 (27.3 MB)

lo        Link encap:Local Loopback
inet addr:127.0.0.1  Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING  MTU:16436  Metric:1
RX packets:2253128 errors:0 dropped:0 overruns:0 frame:0
TX packets:2253128 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:405056167 (405.0 MB)  TX bytes:405056167 (405.0 MB)
```

D e s c r i p t i o n :
This command runs the Unix `ifconfig` command to show interface status.

4.2.7.4. show logging

S y n t a x :
`show logging ( all | console-error | syslog | beacon | console-access )`

Examples:

```
10.0.2.15# show logging syslog | grep UFW
[90655.094966] [UFW BLOCK] IN=eth0 OUT=
MAC=52:54:00:25:e3:9b:c4:2c:03:37:c3:8d:08:00 SRC=192.168.2.23
DST=192.168.2.229 LEN=48 TOS=0x00 PREC=0x00 TTL=64 ID=9384 DF
PROTO=TCP SPT=60385 DPT=8080 WINDOW=65535 RES=0x00 SYN URGP=0
[90671.112565] [UFW BLOCK] IN=eth0 OUT=
MAC=52:54:00:25:e3:9b:c4:2c:03:37:c3:8d:08:00 SRC=192.168.2.23
DST=192.168.2.229 LEN=48 TOS=0x00 PREC=0x00 TTL=64 ID=39100 DF
PROTO=TCP SPT=60385 DPT=8080 WINDOW=65535 RES=0x00 SYN URGP=0
```

```
10.0.2.15# show logging beacon | grep DEBUG | grep Static | tail
-2
21:36:54.830 [Timer-68] DEBUG c.b.b.s.StaticFlowEntryPusher -
pushEntriesForSwitch OFSwitchImpl
[Socket[addr=/192.168.2.28,port=36307,localport=6633]
DPID[00:00:00:00:00:00:00:01]]
21:37:04.815 [Timer-68] DEBUG c.b.b.s.StaticFlowEntryPusher -
pushEntriesForSwitch OFSwitchImpl
[Socket[addr=/192.168.2.29,port=41930,localport=6633]
DPID[00:00:00:00:00:73:28:03]]
```

D e s c r i p t i o n :
This command provides access to logging files for the controller, including general syslog as well as log files for specific components of the controller. As shown in the examples, the output can be piped to various shell utilities to make the output more manageable.

4.2.7.5. show running-config

S y n t a x :
show running-config

Examples:

```
10.0.2.15# show running-config
!
! Big Switch Controller 0.1
(2011.03.28.1314-build.bigswitchcontroller.release)
!
!
switch 00:00:00:00:00:00:00:01
!
switch 00:00:00:00:00:73:28:03
!
switch 00:0a:00:21:f7:de:e9:00
!
switch 00:0a:00:24:a8:c4:69:00
  flow-entry ssh-block
    active False
    dst-port 22
    actions drop
```

D e s c r i p t i o n :
This command shows the running configuration of the controller. In this release, this consists of the current switches that are known to the controller and any static flow entries that have been configured for each switch.

4.2.7.6. show startup-config

S y n t a x :
`show startup-config`

Examples:

```
10.0.2.15# show startup-config
!
! Big Switch Controller 0.1
(2011.03.28.1314-build.bigswitchcontroller.release)
!
!
switch 00:00:00:00:00:00:00:01
!
switch 00:00:00:00:00:73:28:03
!
switch 00:0a:00:21:f7:de:e9:00
```

D e s c r i p t i o n :
This command shows the startup configuration of the controller. The user should have issued a `copy running startup/write memory` command or some other `copy <src> startup` command.

4.2.7.7. show tech-support

S y n t a x :
`show tech-support`

Examples:

```
192.168.2.129> show tech-support |more
Big Switch Controller 0.1
(2011.03.28.1314-build.bigswitchcontroller.release)
Current date/time: 2011-03-29.21:17:56

------------------------------------------------------------------------
controller-node

IP Address Port Active Start Time
----------|----|------|-------------------------
127.0.0.1  6633 True   2011-03-28 15:24:49.533000


------------------------------------------------------------------------
host

MAC Address       Name  Switch ID     Ingress Port IP Address
Vendor                    Last new flow from the host
-----------------|-----|-------------|------------|--------------|---
blah  netgear-2     2 (2)        192.168.10.239 VMware, Inc.
2011-03-29 20:35:02
00:0c:29:74:64:47       netgear-2     2 (2)        192.168.10.200
VMware, Inc.              2011-03-28 19:55:35
00:0c:29:8d:71:a9       netgear-2     23 (23)      192.168.10.223
VMware, Inc.              2011-03-29 20:36:03
00:0c:29:ad:6f:43       netgear-2     2 (2)        192.168.10.239
VMware, Inc.              2011-03-28 21:59:34

...
```

Description:
This command dumps information that is useful for debugging the controller operations. This includes contents of the database, process/system information, command history, and log files.

4.2.7.8. show version

Syntax:
`show version`

Examples:

```
192.168.2.129> show version
Big Switch Controller 0.1
(2011.03.28.1314-build.bigswitchcontroller.release)
```

Description:
This command shows the specific software version installed on this controller node.

4.2.8. show <database object>

4.2.8.1. show controller-node

Syntax:

```
show controller-node [ <controller-node id> ] [stats [<stats options>]]
```

Examples:

```
10.0.2.15# show controller-node
IP Address Port Active Start Time
----------|----|------|-------------------------
127.0.0.1  6633 True   2011-02-01 00:04:49.330000
```

Description:

This command shows information about the current controller node that the CLI is connected to. In this release, the controller node will just be the local machine and will always show 127.0.0.1. In later releases, this will show the multiple controller nodes that comprise the overall controller system.

If the `stats` option is specified, you will be able to see a number of statistics gathered on the status of the system hosting the controller. See the section on `show <object> <id> stats` to learn more about the options here.

4.2.8.2. show flow-entry

Syntax:

```
show flow-entry [ <name> ]
```

Examples:

```
10.0.2.15> show flow-entry
Switch DPID             Name      Active Idle Timeout Hard Timeout
Cookie Priority Wildcards Ingress Port Src MAC Dst MAC Ether Type
VLAN ID VLAN Priority Src IP Dst IP Protocol TOS Bits Src Port Dst
Port Actions
----------------------|---------|------|------------|------------|--
ssh-block False  60           0            0      32768    0
*                                          *
22       drop
```

Description:

This command shows all user-defined, static flow entries in the database. The command `show switch <dpid> flow` shows all flow entries currently on a given switch, including static flow entries that have configured with active set to True.

4.2.8.3. show host

S y n t a x :

```
show host [ <mac address> ]
```

Examples:

```
bigswitchcontroller> sh host
MAC Address       Name Vendor                    IP Address(es)
Switch/OF Port (Physical Port)
-----------------|----|-------------------------|--------------|-----
DELL INC.                 multiple (3)   00:0a:00:21:f7:de:e9:00/4
(4)
00:30:48:f9:cc:f7      Supermicro Computer, Inc. Unknown
multiple (2)
00:30:48:f9:cc:f8      Supermicro Computer, Inc. Unknown
00:00:00:00:00:73:52:02/31 (31)
00:50:8d:65:be:4b      ABIT COMPUTER CORPORATION 10.0.0.9
00:00:00:00:00:32:90:01/16 (16)
10:00:00:67:ad:07      unknown                   192.168.12.132
multiple (2)
10:00:00:6f:ca:67      unknown                   192.168.12.131
multiple (2)
10:9a:dd:4b:c4:dd      Apple Inc                 192.168.10.249
00:00:00:00:00:73:52:01/28 (28)
10:9a:dd:4b:e5:cb      Apple Inc                 192.168.10.243
00:00:00:00:00:73:52:01/23 (23)
10:9a:dd:52:39:d5      Apple Inc                 192.168.10.237
00:00:00:00:00:73:52:01/2 (2)
58:b0:35:f5:f2:8f      Apple, Inc                multiple (2)
00:00:00:00:00:73:52:01/19 (19)
c0:3f:0e:75:83:65      NETGEAR                   Unknown
00:0a:00:24:a8:c4:69:00/25 (25)
c4:2c:03:02:5a:11      Apple                     192.168.10.238
00:00:00:00:00:73:52:01/23 (23)
c4:2c:03:37:c3:8d      Apple                     192.168.10.216
00:00:00:00:00:73:52:01/23 (23)
c8:2a:14:05:de:ac      unknown                   192.168.10.246
00:00:00:00:00:73:52:01/23 (23)
```

D e s c r i p t i o n :

This command shows host information that has been discovered by the Device Manager application. In addition to the MAC address, the specific switch and port where this host was learned is displayed.

If a host was learned in multiple places or has multiple IP addresses the word 'multiple' along with how many places it was learned or how many IPs it has is shown. If no IP address is known for a host 'Unknown' is displayed instead.

Note that the ingress port is the logical OpenFlow port and the name of the port is in parentheses.

Name can be written in by the user for convenience in identifying particular hosts.

4.2.8.4. show link

S y n t a x :

```
show link [ <link-id> ]
```

Examples:

```
192.168.2.129> show link
Src Switch DPID Src Port Src Port State Dst Switch DPID Dst Port
Dst Port State
---------------|--------|--------------|---------------|--------|----
5        listen(0x0)    netgear-2       33       link-down(0x1)
netgear-2       33       listen(0x0)    netgear-1       5
listen(0x0)
netgear-2       9        listen(0x0)    hp-6600         43
listen(0x0)
wireless-ap-2   3        listen(0x0)    hp-6600         62
listen(0x0)
vswitch         1        listen(0x0)    hp-6600         67
listen(0x0)
hp-5400         5        forward(0x200) hp-6600         29
listen(0x0)
```

D e s c r i p t i o n :

This command shows link information that has been discovered by the Topology application. Links are added from the perspective of a given switch based on LLDP packets, so two link rows will exist for a single "logical" point-2-point link between two switches.

A link id is the concatenation of the source switch DPID/port and destination switch DPID/port, joined by the – character. In the example above, the id for the first line is `00:00:00:00:00:00:00:01-2-00:00:00:00:00:00:00:02-2`

4.2.8.5. show port

S y n t a x :

```
show port [ <port-id> ]
```

Examples:

```
10.0.2.15> show port
Switch DPID             OF #  Name  MAC Address
Configuration State Current Advertised Supported Peer
-----------------------|-----|-----|-----------------|--------------|
10    10    00:26:e1:cb:db:0a                0x1
00:00:00:00:00:00:00:01 11    11    00:26:e1:cd:5d:0b
0x1
00:00:00:00:00:00:00:01 12    12    00:26:e1:fe:72:0c
0x1
00:00:00:00:00:00:00:01 13    13    00:26:e1:b5:02:0d
0x1
00:00:00:00:00:00:00:01 14    14    00:26:e1:6a:4c:0e
0x1
00:00:00:00:00:00:00:01 15    15    00:26:e1:db:52:0f
0x1
```

D e s c r i p t i o n :

This command shows port information that has been discovered by the controller. Both the logical OpenFlow port number and the name of the interface (often the externally visible physical port number on a switch) are given.

`<port-id>` is the concatenation of the switch DPID and the logical OpenFlow port number joined by a `:` . In the example above, the id for the first port is `00:00:00:00:00:00:00:01:1`.

4.2.8.6. show switch

S y n t a x :

```
show switch [ realtime | <dpid> | all | <alias> ] [ aggregate | desc |
flow [brief | details] | host | port | table | features ] ]
```

Examples:

```
192.168.2.129> show switch
Switch DPID             Alias         Active Last Connect Time
IP Address    Socket Address       Max Packets Max Tables
----------------------|-------------|------|------------------|----
pronto-3290   False  2011-03-28 15:24:52 192.168.2.31
/192.168.2.31:33083  256         3
00:00:00:00:00:37:80:01 pronto-3780   True   2011-03-28 15:24:51
192.168.2.32  /192.168.2.32:47260  256         3
00:00:00:00:00:73:28:02 netgear-1     True   2011-03-28 15:24:52
192.168.2.28  /192.168.2.28:40924  256         3
00:00:00:00:00:73:52:01 netgear-2     True   2011-03-28 15:24:51
192.168.2.52  /192.168.2.52:57690  256         3
00:00:00:00:00:d0:d0:03 wireless-ap-1 True   2011-03-28 15:24:56
192.168.2.34  /192.168.2.34:57244  256         2
00:00:00:00:00:d0:d0:05 wireless-ap-2 True   2011-03-28 15:24:56
192.168.2.35  /192.168.2.35:54597  256         2
```

```
10.0.2.15> show switch realtime
Switch DPID
-----------------------
00:00:00:30:48:f9:cd:0b
00:0a:00:21:f7:de:e9:00
00:00:00:30:48:f9:cc:f7
00:0a:00:24:a8:c4:69:00
00:00:00:00:00:00:00:01

192.168.2.129> show switch 00:0a:00:24:a8:c4:69:00 desc
Switch  Serial #   Vendor  Make
Model      SW Version
-------|----------|-------|-------------------------------------|----
CN9202M063 HP-Labs HP Procurve 5400zl/3500yl/6600 series
hp6600-101 2.02v

192.168.2.129> show switch 00:0a:00:24:a8:c4:69:00 features
Switch  OF Port #    Name  HW Mac Address    Config State
Current                 Advertised Supported
Peer
-------|------------|-----|-----------------|------|--------------|--
25 (1)       1     00:24:a8:c4:69:e7 0x0    forward(0x200)
1gb-fd,autoneg(0x220)   0x0
10mb-hd,10mb-fd,100mb-hd,100mb-fd,1gb-fd,autoneg(0x22f) 0x0
hp-6600 29 (5)       5     00:24:a8:c4:69:e3 0x0    forward(0x200)
1gb-fd,autoneg(0x220)   0x0
10mb-hd,10mb-fd,100mb-hd,100mb-fd,1gb-fd,autoneg(0x22f) 0x0
hp-6600 30 (6)       6     00:24:a8:c4:69:e2 0x0    block(0x300)
1gb-fd,autoneg(0x220)   0x0
10mb-hd,10mb-fd,100mb-hd,100mb-fd,1gb-fd,autoneg(0x22f) 0x0
hp-6600 43 (19)      19    00:24:a8:c4:69:d5 0x0    forward(0x200)
1gb-fd,autoneg(0x220)   0x0
10mb-hd,10mb-fd,100mb-hd,100mb-fd,1gb-fd,autoneg(0x22f) 0x0
hp-6600 62 (38)      38    00:24:a8:c4:69:c2 0x0    forward(0x200)
1gb-fd,autoneg(0x220)   0x0
10mb-hd,10mb-fd,100mb-hd,100mb-fd,1gb-fd,autoneg(0x22f) 0x0
hp-6600 64 (40)      40    00:24:a8:c4:69:c0 0x0    forward(0x200)
100mb-fd,autoneg(0x208) 0x0
10mb-hd,10mb-fd,100mb-hd,100mb-fd,1gb-fd,autoneg(0x22f) 0x0
hp-6600 67 (43)      43    00:24:a8:c4:69:bd 0x0    forward(0x200)
1gb-fd,autoneg(0x220)   0x0
10mb-hd,10mb-fd,100mb-hd,100mb-fd,1gb-fd,autoneg(0x22f) 0x0
hp-6600 68 (44)      44    00:24:a8:c4:69:bc 0x0    forward(0x200)
1gb-fd,autoneg(0x220)   0x0
10mb-hd,10mb-fd,100mb-hd,100mb-fd,1gb-fd,autoneg(0x22f) 0x0
hp-6600 local(65534) local 00:24:a8:c4:69:00 0x0    listen(0x0)
0x0                     0x0        0x0
0x0

192.168.2.129> show switch 00:0a:00:24:a8:c4:69:00 host
```

```
Switch  MAC Address         VLAN Switch Port
-------|------------------|----|-----------
hp-6600 c4:2c:03:02:5a:11:L 10   43
hp-6600 00:c2:92:45:2d:7L   10   43
hp-6600 01:14:3d:87:a6:2L   10   29
hp-6600 10:9a:dd:52:39:d5:L 10   43
hp-6600 10:9a:dd:4b:c4:dd:L 10   43
hp-6600 02:54:bc:d0:89:0L   10   43

192.168.2.129> show switch 00:0a:00:24:a8:c4:69:00 port
Switch  OF Port #    Rcv Bytes   Rcv Pkts Rcv Errs Rcv Dropped Rcv
CRC Rcv Overruns Rcv Frame Errs Xmit Bytes  Xmit Pkts Xmit Errs
Xmit Dropped Collisions
-------|------------|-----------|--------|--------|-----------|------
25 (1)       3297786525  20532074          0           n/a     n/a
n/a            4014392970  22421628  0         0            n/a
hp-6600 29 (5)       44898224795 55976242          0           n/a
n/a          n/a            13607647730 55432025  0         0
n/a
hp-6600 30 (6)       23048783577 34488214          0           n/a
n/a          n/a            3005116197  16835159  0         0
n/a
hp-6600 43 (19)      9997933560  42702605          0           n/a
n/a          n/a            50611767271 57349665  0         0
n/a
hp-6600 62 (38)      12593367    196630            0           n/a
n/a          n/a            578185690   5711527   0         0
n/a
hp-6600 64 (40)      13247627    40730             0           n/a
n/a          n/a            611940635   5753084   0         0
n/a
hp-6600 67 (43)      967734799   2660588           0           n/a
n/a          n/a            4406391087  9026356   0         0
n/a
hp-6600 68 (44)      2556554522  7647534           0           n/a
n/a          n/a            10349780825 14700074  0         0
n/a
hp-6600 local(65534) n/a         n/a               n/a         n/a
n/a          n/a            n/a         n/a       n/a       n/a
n/a

192.168.2.129> show switch 00:0a:00:24:a8:c4:69:00 table
Switch  Name       Table ID Wildcards Maximum Entries # Lookups #
Matched # Active
-------|----------|--------|---------|---------------|---------|-----
HW TCAM    0        532707    1500            0         0
16
hp-6600 hash       1        0         65536           6078274
606977    0
hp-6600 classifier 2        4194303   65536           0         0
20
```

```
192.168.2.129> show switch 00:0a:00:24:a8:c4:69:00 flow
Switch  Bytes Packets Dur(s) Cookie  In Port Src MAC           Dst
MAC           Ether Type Src IP         Dst IP         Protocol
Src Port Dst Port
-------|-----|-------|------|-------|-------|-----------------|------
70    1       2      lswitch 29 (5)  00:11:43:d8:7a:62
c4:2c:03:02:5a:11 ip(2048)   205.234.25.153 192.168.10.237 *
*        *
hp-6600 0     1       0      lswitch 43 (19) c4:2c:03:02:5a:11
00:11:43:d8:7a:62 ip(2048)   192.168.10.237 74.125.224.85  *
*        *
hp-6600 78    273364  8688   lswitch 29 (5)  00:11:43:d8:7a:62
10:9a:dd:4b:c4:dd ip(2048)   192.168.2.152  192.168.10.248 *
*        *
hp-6600 125   22      1      lswitch 43 (19) c4:2c:03:02:5a:11
00:11:43:d8:7a:62 ip(2048)   192.168.10.237 74.125.47.109  *
*        *
hp-6600 0     51      9      lswitch 43 (19) 10:9a:dd:52:39:d5
00:11:43:d8:7a:62 ip(2048)   192.168.10.236 208.85.146.99  *
*        *
hp-6600 0     0       2      lswitch 29 (5)  00:11:43:d8:7a:62
c4:2c:03:02:5a:11 ip(2048)   74.125.224.80  192.168.10.237 *
*        *
hp-6600 72    1       2      lswitch 43 (19) c4:2c:03:02:5a:11
00:11:43:d8:7a:62 ip(2048)   192.168.10.237 74.125.224.80  *
*        *
hp-6600 0     0       0      lswitch 67 (43) 10:00:00:67:ad:07
00:11:43:d8:7a:62 ip(2048)   192.168.12.132 192.168.2.150  *
*        *
hp-6600 329   4088    683    lswitch 43 (19) c4:2c:03:37:c3:8d
00:11:43:d8:7a:62 ip(2048)   192.168.10.215 208.85.146.99  *
*        *
hp-6600 0     0       2      lswitch 43 (19) c4:2c:03:37:c3:8d
00:11:43:d8:7a:62 ip(2048)   192.168.10.215 98.136.48.116  *
*        *
hp-6600 66    40      9      lswitch 29 (5)  00:11:43:d8:7a:62
10:9a:dd:52:39:d5 ip(2048)   208.85.146.99  192.168.10.236 *
*        *
hp-6600 64    1       3      lswitch 29 (5)  00:11:43:d8:7a:62
c4:2c:03:02:5a:11 ip(2048)   118.169.32.181 192.168.10.237 *
*        *
hp-6600 0     19531   19539  lswitch 43 (19) c4:2c:03:37:c3:8d
00:11:43:d8:7a:62 ip(2048)   192.168.10.215 192.168.10.1   *
*        *
hp-6600 0     0       3      lswitch 43 (19) c4:2c:03:02:5a:11
00:11:43:d8:7a:62 ip(2048)   192.168.10.237 118.169.32.181 *
*        *
hp-6600 772   32447   19539  lswitch 29 (5)  00:11:43:d8:7a:62
c4:2c:03:37:c3:8d ip(2048)   192.168.10.1   192.168.10.215 *
*        *
```

```
hp-6600 0     0       1      lswitch 29 (5)  00:11:43:d8:7a:62
c4:2c:03:02:5a:11 ip(2048)   95.88.104.171  192.168.10.237 *
*        *
hp-6600 70    116930  8688   lswitch 43 (19) 10:9a:dd:4b:c4:dd
00:11:43:d8:7a:62 ip(2048)   192.168.10.248 192.168.2.152  *
*        *
hp-6600 149   1       1      lswitch 43 (19) c4:2c:03:02:5a:11
00:11:43:d8:7a:62 ip(2048)   192.168.10.237 95.88.104.171  *
*        *
hp-6600 160   1       0      lswitch 29 (5)  00:11:43:d8:7a:62
10:00:00:67:ad:07 ip(2048)   192.168.2.150  192.168.12.132 *
*        *
hp-6600 70    1       2      lswitch 29 (5)  00:11:43:d8:7a:62
c4:2c:03:37:c3:8d ip(2048)   98.136.48.116  192.168.10.215 *
*        *
hp-6600 0     21      1      lswitch 29 (5)  00:11:43:d8:7a:62
c4:2c:03:02:5a:11 ip(2048)   74.125.47.109  192.168.10.237 *
*        *
hp-6600 384   1       0      lswitch 29 (5)  00:11:43:d8:7a:62
c4:2c:03:02:5a:11 ip(2048)   74.125.224.85  192.168.10.237 *
*        *
hp-6600 66    3289    683    lswitch 29 (5)  00:11:43:d8:7a:62
c4:2c:03:37:c3:8d ip(2048)   208.85.146.99  192.168.10.215 *
*        *
hp-6600 0     1       2      lswitch 43 (19) c4:2c:03:02:5a:11
00:11:43:d8:7a:62 ip(2048)   192.168.10.237 205.234.25.153 *
*        *
hp-6600 0     0       2      lswitch 29 (5)  00:11:43:d8:7a:62
c4:2c:03:37:c3:8d ip(2048)   192.168.2.1    192.168.10.215 *
*        *
hp-6600 72    1       2      lswitch 43 (19) c4:2c:03:37:c3:8d
00:11:43:d8:7a:62 ip(2048)   192.168.10.215 192.168.2.1    *
```

```
*        *
```

Use of all as an argument to show switch:

```
192.168.2.129> sh switch netgear-1 table
Switch    Name           Table ID Wildcards Maximum Entries #
Lookups # Matched # Active
---------|--------------|--------|---------|---------------|---------
BCM FlowDriver 0        4194303   512             17264224  49895
0
netgear-1 hash2          1        0         131072          0
0         0
netgear-1 linear         2        4194303   100             0
0         0
```

Use of alias as an argument to show switch:

```
192.168.2.129> sh switch netgear-1 table
Switch    Name           Table ID Wildcards Maximum Entries #
Lookups # Matched # Active
---------|--------------|--------|---------|---------------|---------
BCM FlowDriver 0        4194303   512             17264224  49895
0
netgear-1 hash2          1        0         131072          0
0         0
netgear-1 linear         2        4194303   100             0
0         0
```

D e s c r i p t i o n :

This command shows both switch information currently in the database as well as real-time information directly collected from the switch. For convenience, the "all" keyword can be given to retrieve information from all the switches at once. Switch aliases can also be given instead of giving the full 8-byte DPIDs.

`show switch` and {{show switch < dpid | alias >} }} will show switches in the database, including those that are no longer connected to the controller.

`show switch realtime` shows a list of the switches that are currently connected to the controller.

{{show switch < dpid | alias | all > host }} shows a list of hosts and their ports and VLAN IDs for a given switch.

`show switch aggregate`, `desc`, `features`, `flow`, `port`, and `table` correspond to the OpenFlow statistics that a switch can collect and present.

- `aggregate` shows the total flows, bytes, and packets that this switch has seen from an OpenFlow perspective
- `desc` shows the configured serial #, vendor, make, model, and software version
- `features` shows the port features when the switch initially connected to the controller.
- `flow` shows the flow entries that are currently active on this switch. The `details` and `brief` options show more or less information per flow entry - in particular, details will show the actions for a flow-entry. Wildcarded match fields are shown with a `*`, and the `cookie` field shows the source of the flow mod (either `static` for statically configured, `lswitch` for Learning Switch application, or `routing` for the Routing application).
- `port` shows the real-time counters (received/transmitted bytes/packets/errors, collisions...)
- `table` shows counters for each table on the switch (see the OpenFlow spec or the appropriate switch manual for more details on this).

4.2.8.7. show <object> <id> stats

S y n t a x :

```
show (switch | controller-node) <object id> stats [<stat-name> [display (graph | table | latest-value)] [duration <interval>] [start-time <time>] [end-time <time>] [sample-count <count>] [sample-interval <integer milliseconds>] [sample-window <integer milliseconds>] [data-format (rate | value)]]
```

E x a m p l e s :

View most recent values for statistics on the specified controller node:

```
bigswitchcontroller> show controller-node 192.168.2.129:6633 stats
CPU User       : 3 %
CPU Nice       : 0 %
CPU System     : 0 %
Memory Used    : 906092 kB
Swap Used      : 98832 kB
/ Used         : 13 %
/log Used      : 21 %
/sysboot Used  : 2 %
Beacon CPU     : 1 %
Database CPU   : 0 %
Apache CPU     : 1 %
Cli CPU        : 0 %
Statd CPU      : 6 %
Memory Free    : 118884 kB
CPU Idle       : 95 %
```

Show most recent values for statistics on the specified switch:

```
bigswitchcontroller> show switch 00:0a:00:21:f7:de:e9:00 stats
OF Packet In    : 21590 Packets
OF Flow Mod     : 41584 Flow Mods
OF Active Flow : 20 Flows
```

Show the most recent value for a specific statistic

```
bigswitchcontroller> show controller-node 192.168.2.129:6633 stats
cpu-user
CPU User      : 2 %
```

Display a graph of the CPU system stat for the last hour on the controller node (some load has been artificially generated):

```
roba-controller> show controller-node 192.168.2.129:6633 stats
cpu-system display graph
CPU System (%)
100|
 96|
 92|
 88|
 84|
 80|
 76|
 72|
 68|
 64|
 60|
 56|
 52|
 48|
 44|
 40|
 36|
 32|
 28|
 24|                                                    #
 20|                                                    #
 16|                                                    #
 12|                                                    #
###
  8|                                                    ##
###

0|####################________________________________##############
^12:30 ^12:35 ^12:41 ^12:46 ^12:52 ^12:57 ^13:02 ^13:08 ^13:13
^13:19  13:29^
   ^05/26
05/26^
                                        Time
```

Display a graph of the CPU user stat for the last 30 minutes on the controller node (some load has been artificially generated):

```
roba-controller> show controller-node 172.16.195.129:6633 stats
cpu-user display graph duration 20m
CPU User (%)
100|  ############################
 96|############################
 92|############################
 88|############################
 84|############################
 80|############################
 76|############################
 72|############################
 68|############################
 64|############################
 60|############################
 56|############################
 52|############################
 48|############################
 44|############################
 40|############################
 36|############################
 32|############################
 28|############################
 24|############################
 20|############################                                #  #
# # # #     #
 16|############################
###################
 12|############################
###################
  8|############################
###################

0|#################################################################
^13:12 ^13:14 ^13:15 ^13:17 ^13:19 ^13:21 ^13:23 ^13:24 ^13:26
^13:28  13:31^
   ^05/26
05/26^
                                        Time
```

Display 5 minutes of the CPU user statistic for the controller starting at 1:24 PM today in a table format. Note that not all stored data is displayed necessarily in the table format because it will be down-sampled. You can control the downsampling using the `sample-count` and `sample-window` parameters:

```
roba-controller> show controller-node 172.16.195.129:6633 stats
cpu-user display graph start-time 13:24 duration 5m display table
CPU User (%) Timestamp
------------|------------------------
2            Thu May 26 13:24:07 2011
0            Thu May 26 13:24:17 2011
0            Thu May 26 13:24:27 2011
2            Thu May 26 13:24:37 2011
0            Thu May 26 13:24:47 2011
0            Thu May 26 13:24:57 2011
3            Thu May 26 13:25:07 2011
0            Thu May 26 13:25:17 2011
0            Thu May 26 13:25:27 2011
2            Thu May 26 13:25:37 2011
0            Thu May 26 13:25:47 2011
0            Thu May 26 13:25:57 2011
2            Thu May 26 13:26:07 2011
0            Thu May 26 13:26:17 2011
1            Thu May 26 13:26:27 2011
2            Thu May 26 13:26:37 2011
7            Thu May 26 13:26:47 2011
17           Thu May 26 13:26:57 2011
19           Thu May 26 13:27:07 2011
17           Thu May 26 13:27:17 2011
16           Thu May 26 13:27:27 2011
18           Thu May 26 13:27:37 2011
16           Thu May 26 13:27:47 2011
16           Thu May 26 13:27:57 2011
19           Thu May 26 13:28:07 2011
17           Thu May 26 13:28:17 2011
16           Thu May 26 13:28:27 2011
19           Thu May 26 13:28:37 2011
16           Thu May 26 13:28:47 2011
17           Thu May 26 13:28:57 2011
```

D e s c r i p t i o n :

Some objects support displaying statistics about that object, including seeing how these change over time. Currently, you can request statistics for the `controller-node` and `switch` objects. For the controller node, the following statistics are available:

- **cpu-user** - Total CPU user percentage
- **cpu-nice** - Total CPU nice percentage
- **cpu-system** - Total CPU system percentage
- **cpu-idle** - Total CPU idle percentage
- **mem-used** - Total memory used in kilobytes
- **mem-free** - Memory free in kilobytes
- **swap-used** - Swap space used in kilobytes
- **disk-root** - Percentage of the root partition disk space used
- **disk-log** - Percentage of the log partition disk space used

- **disk-boot** - Percentage of the boot partition disk space used
- **beacon-cpu** - CPU usage for the controller processes
- **database-cpu** - CPU usage for the local database processes
- **apache-cpu** - CPU usage for the web server processes
- **cli-cpu** - CPU usage for the command-line interface processes
- **statd-cpu** - CPU usage for the stats gathering daemon processes

For switches, the following statistics are available:

- **OFActiveFlow** - Number of active flows
- **OFFlowMod** - Number of flow mods processed
- **OFPacketIn** - Number of openflow packets processed

The following arguments are accepted in any order:

- **display** - `graph | table | latest-value` - Default is `latest-value` unless a time range is specified, in which case it will default to `graph`. `latest-value` will display the most recent value for that statistic. `graph` will display the time series data on an ascii graph. `table` will display the time series data in tabular form.
- **duration** - `<integer>weeks | <integer>days | <integer>hours | <integer>mins | <integer>secs | <integer>ms` - Select the length of the time interval to display. If `start-time` is specified this will be starting from the start time. If `end-time` is specified it will be ending at the end time. If neither is specified it will be ending at the current time.
- **start-time** - `"YYYY-MM-DD HH:MM:SS" | YYYY-MM-DDTHH:MM:SS | "YYYY-MM-DD HH:MM:SS+TTT" | YYYY-MM-DDTHH:MM:SS+TTT | YYYY-MM-DD | MM-DD HH:MM | now` - Specify the start time for the time series in one of the indicated formats
- **end-time** - `"YYYY-MM-DD HH:MM:SS" | YYYY-MM-DDTHH:MM:SS | "YYYY-MM-DD HH:MM:SS+TTT" | YYYY-MM-DDTHH:MM:SS+TTT | YYYY-MM-DD | MM-DD HH:MM | now` - Specify the start time for the time series in one of the indicated formats
- **sample-count** - count - An integer number of samples to target to return. Note that it's possible more or less samples could be returned.
- **sample-interval** - `<integer>weeks | <integer>days | <integer>hours | <integer>mins | <integer>secs | <integer>ms` - A number of milliseconds as the interval between samples
- **sample-window** - `<integer>weeks | <integer>days | <integer>hours | <integer>mins | <integer>secs | <integer>ms` - Average all the samples in the neighborhood of each sample and return the average value. The window is specified as an integer number of milliseconds.

4.2.9. `trace`

Syntax:

```
trace [detail |oneline] [echo_reply | echo_request | features_rep | flow_mod | flow_removed | get_config_rep | hello | packet_in | packet_out | port_status | set_config | stats_reply | stats_request]*
show switch <dpid | alias> trace [detail |oneline] [echo_reply | echo_request | features_rep | flow_mod | flow_removed | get_config_rep | hello | packet_in | packet_out | port_status | set_config | stats_reply | stats_request]*
```

Examples:

```
bigswitchcontroller> trace
Starting openflow trace, use ^C to quit
11:39:51.570389    flow_mod       [   192.168.2.129:6633  ->
192.168.2.101:42390 ]
11:39:51.570951    flow_mod       [   192.168.2.129:6633  ->
192.168.2.102:45682 ]
11:39:51.570967    flow_mod       [   192.168.2.129:6633  ->
192.168.2.31:45636 ]
11:39:51.571071    flow_mod       [   192.168.2.129:6633  ->
192.168.2.32:44175 ]
11:39:51.571084    flow_mod       [   192.168.2.129:6633  ->
192.168.2.36:53820 ]
11:39:51.609319    port_status    [    192.168.2.53:37396 ->
192.168.2.129:6633  ]
11:39:51.872062    port_status    [    192.168.2.53:37396 ->
192.168.2.129:6633  ]
11:39:52.017197    flow_removed   [   192.168.2.102:45682 ->
192.168.2.129:6633  ]
11:39:52.017251    flow_removed   [   192.168.2.102:45682 ->
192.168.2.129:6633  ]
11:39:52.018221    flow_mod       [   192.168.2.129:6633  ->
192.168.2.102:45682 ]
11:39:52.026433    flow_removed   [   192.168.2.101:42390 ->
192.168.2.129:6633  ]
11:39:52.026459    flow_removed   [   192.168.2.101:42390 ->
192.168.2.129:6633  ]
11:39:52.027282    flow_mod       [   192.168.2.129:6633  ->
192.168.2.101:42390 ]
11:39:52.100353    packet_in      [   192.168.2.102:45682 ->
192.168.2.129:6633  ]
11:39:52.101537    packet_out     [   192.168.2.129:6633  ->
192.168.2.102:45682 ]
11:39:52.103353    packet_in      [   192.168.2.101:42390 ->
192.168.2.129:6633  ]
11:39:52.103564    packet_out     [   192.168.2.129:6633  ->
192.168.2.101:42390 ]
11:39:52.107072    packet_in      [    192.168.2.34:36938 ->
192.168.2.129:6633  ]
11:39:52.107360    packet_out     [   192.168.2.129:6633  ->
192.168.2.34:36938 ]
11:39:52.130533    port_status    [    192.168.2.53:37396 ->
192.168.2.129:6633  ]
11:39:52.178312    packet_out     [   192.168.2.129:6633  ->
192.168.2.101:42390 ]
11:39:52.181437    packet_out     [   192.168.2.129:6633  ->
192.168.2.102:45682 ]
11:39:52.191252    packet_out     [   192.168.2.129:6633  ->
```

```
192.168.2.31:45636 ]
11:39:52.201603    packet_out      [   192.168.2.129:6633  ->
192.168.2.32:44175 ]
11:39:52.208931    packet_out      [   192.168.2.129:6633  ->
192.168.2.36:53820 ]
^Cbigswitchcontroller>

bigswitchcontroller> trace detail flow_mod
Starting openflow trace, use ^C to quit
11:40:11.890486    flow_mod        [   192.168.2.129:6633  ->
192.168.2.102:45682 ]
    arp
    in_port=*
    dl_vlan=0x000a
    dl_vlan_pcp=0x02
    dl_src=10:9a:dd:4b:e5:cb
    dl_dst=00:11:43:d8:7a:62
    nw_src=192.168.10.242
    nw_dst=192.168.10.1
    nw_tos=*
    nw_proto=*
    tp_src=*
    tp_dst=*
    DEL: cookie:4503599627370496 idle:5 hard:0 pri:100
buf:0xffffffff flg:0 actions=output:4
11:40:11.994601    flow_mod        [   192.168.2.129:6633  ->
192.168.2.32:44175 ]
    arp
    in_port=*
    dl_vlan=0xffff
    dl_vlan_pcp=0x00
    dl_src=00:11:43:d8:7a:62
    dl_dst=10:9a:dd:4b:e5:cb
    nw_src=192.168.10.1
    nw_dst=192.168.10.242
    nw_tos=*
    nw_proto=*
    tp_src=*
    tp_dst=*
    DEL: cookie:4503599627370496 idle:5 hard:0 pri:100
buf:0xffffffff flg:0 actions=output:1
11:40:11.997367    flow_mod        [   192.168.2.129:6633  ->
192.168.2.101:42390 ]
    ip
    in_port=43
    dl_vlan=0x000a
    dl_vlan_pcp=0x02
    dl_src=10:9a:dd:4b:c4:dd
    dl_dst=00:11:43:d8:7a:62
    nw_src=192.168.10.248
    nw_dst=192.168.10.1
```

```
    nw_tos=*
    nw_proto=*
    tp_src=*
    tp_dst=*
    ADD: cookie:4503599627370496 idle:5 hard:0 pri:100
buf:0x9c8486 flg:0x1 actions=output:29
^Cbigswitchcontroller>

bigswitchcontroller> trace oneline packet_in
Starting openflow trace, use ^C to quit
11:50:22.819522    packet_in      [   192.168.2.102:45682 ->
192.168.2.129:6633  ], total_len=205 in_port=5 data_len=205
buffer=0x00589eec, in_port=5, dl_vlan=0xffff, dl_vlan_pcp=0x00,
dl_src=00:24:a8:c4:69:e3, dl_dst=01:80:c2:00:00:0e,
dl_type=0x88cc, nw_src=0.0.0.0, nw_dst=0.0.0.0, nw_tos=0x00,
nw_proto=0, tp_src=0, tp_dst=0
11:50:22.853040    packet_in      [   192.168.2.102:45682 ->
192.168.2.129:6633  ], total_len=183 in_port=5 data_len=183
buffer=0x00589eed, in_port=5, dl_vlan=0xffff, dl_vlan_pcp=0x00,
dl_src=00:24:a8:c4:69:e3, dl_dst=01:80:c2:00:00:00,
dl_type=0x05ff, nw_src=0.0.0.0, nw_dst=0.0.0.0, nw_tos=0x00,
nw_proto=0, tp_src=0, tp_dst=0
11:50:22.853077    packet_in      [   192.168.2.102:45682 ->
192.168.2.129:6633  ], total_len=183 in_port=6 data_len=183
buffer=0x00589eee, in_port=6, dl_vlan=0xffff, dl_vlan_pcp=0x00,
dl_src=00:24:a8:c4:69:e2, dl_dst=01:80:c2:00:00:00,
dl_type=0x05ff, nw_src=0.0.0.0, nw_dst=0.0.0.0, nw_tos=0x00,
nw_proto=0, tp_src=0, tp_dst=0
11:50:22.857064    packet_in      [    192.168.2.34:36938 ->
192.168.2.129:6633  ], total_len=183 in_port=4 data_len=183
buffer=0x002ce628, in_port=4, dl_vlan=0xffff, dl_vlan_pcp=0x00,
dl_src=00:24:a8:c4:69:c0, dl_dst=01:80:c2:00:00:00,
dl_type=0x05ff, nw_src=0.0.0.0, nw_dst=0.0.0.0, nw_tos=0x00,
nw_proto=0, tp_src=0, tp_dst=0
11:50:22.858939    packet_in      [    192.168.2.52:39124 ->
192.168.2.129:6633  ], total_len=183 in_port=9 data_len=183
buffer=0x00af0b91, in_port=9, dl_vlan=0xffff, dl_vlan_pcp=0x00,
dl_src=00:24:a8:c4:69:d5, dl_dst=01:80:c2:00:00:00,
dl_type=0x05ff, nw_src=0.0.0.0, nw_dst=0.0.0.0, nw_tos=0x00,
nw_proto=0, tp_src=0, tp_dst=0
11:50:22.859429    packet_in      [    192.168.2.31:45636 ->
192.168.2.129:6633  ], total_len=183 in_port=22 data_len=183
buffer=0x00085a8d, in_port=22, dl_vlan=0xffff, dl_vlan_pcp=0x00,
dl_src=00:24:a8:c4:69:d0, dl_dst=01:80:c2:00:00:00,
dl_type=0x05ff, nw_src=0.0.0.0, nw_dst=0.0.0.0, nw_tos=0x00,
nw_proto=0, tp_src=0, tp_dst=0
11:50:22.976736    packet_in      [   192.168.2.101:42390 ->
192.168.2.129:6633  ], total_len=183 in_port=29 data_len=183
buffer=0x009c8bf1, in_port=29, dl_vlan=0xffff, dl_vlan_pcp=0x00,
dl_src=00:21:f7:de:f9:fb, dl_dst=01:80:c2:00:00:00,
dl_type=0x05ff, nw_src=0.0.0.0, nw_dst=0.0.0.0, nw_tos=0x00,
```

```
nw_proto=0, tp_src=0, tp_dst=0
11:50:22.976765    packet_in      [   192.168.2.101:42390 ->
192.168.2.129:6633  ], total_len=183 in_port=30 data_len=183
buffer=0x009c8cf2, in_port=30, dl_vlan=0xffff, dl_vlan_pcp=0x00,
dl_src=00:21:f7:de:f9:fa, dl_dst=01:80:c2:00:00:00,
dl_type=0x05ff, nw_src=0.0.0.0, nw_dst=0.0.0.0, nw_tos=0x00,
nw_proto=0, tp_src=0, tp_dst=0
11:50:23.210654    packet_in      [   192.168.2.101:42390 ->
192.168.2.129:6633  ], total_len=123 in_port=25 data_len=123
buffer=0x009c8bf3, in_port=25, dl_vlan=0x0001, dl_vlan_pcp=0x02,
dl_src=c0:3f:0e:75:83:65, dl_dst=01:80:c2:00:00:00,
dl_type=0x0069, nw_src=0.0.0.0, nw_dst=0.0.0.0, nw_tos=0x00,
nw_proto=0, tp_src=0, tp_dst=0
```

```
^Cbigswitchcontroller>
```

Description:
This command prints the trace of openflow messages being exchanged between the controller and the switches. It can be executed in the login context (as '`trace [options]`'), or the switch context (as '`switch show <dpid|alias> trace [options]`'). In the login context, it displays messages to/from all switches. And in the switch context, it displays the messages to/from that specific switch. The trace is terminated by a ^C on the terminal.

The default output format is a summary that identifies the type of message and the src/dest nodes. You can view additional openflow message details using the options `detail` or `oneline`. `detail` lists openflow message details in a multi-line format that is easier to use interactively, and `oneline` formats the same content as one line which is easier to use with grep/other tools.

The output can be further filtered to specific type(s) of messages by specifying them on the commandline.

4.3. Enable mode commands

4.3.1. `bundles`

Syntax:
`bundles <bundle-id> ( stop | start )`

Examples:

```
10.0.2.15# show bundles | grep static
Active   3  com.bigswitch.beacon.staticflowentry
10.0.2.15# bundles 3 stop
10.0.2.15# show bundles | grep static
Resolved 3  com.bigswitch.beacon.staticflowentry
10.0.2.15# bundles 3 start
10.0.2.15# show bundles | grep static
Active   3  com.bigswitch.beacon.staticflowentry
10.0.2.15#
```

Description:

Warning: this command should only be run when instructed by Big Switch Networks.

This command allows the user to stop/start specific bundles in the Java Beacon process. Generally, all bundles should be running, but there may be specific deployments where certain bundles (for example, the routing application) may need to be disabled. Alternatively, there may be errors conditions where a bundle needs to be manually restarted.

The bundle ids are shown in the `show bundles` command. The example above shows obtaining the bundle id for a specific bundle using the `show` command, deactivating it and checking its status, and then

reactivating it.

4.3.2. configure

Syntax:
`configure`

Examples:

```
10.0.2.15# configure
10.0.2.15(config)# help
For help on specific commands type help <topic>
Available commands:
  bundles
  connect
  copy
  date
  debug
  end
  exit
  help
  history
  logout
  no
  reload
  show
  write

Available config objects:
  controller-node
  host
  link
  port
  switch
```

Description:
This command puts the user into config mode where objects can be created, updated, and deleted. The CLI prompt will change to `(config)#` (see example above). In this mode, commands are available to create, update, and delete objects.

The controller-node, host, link, port, and switch objects are typically written in by the controller applications such as Device Manager and Topology. The typical user-defined object are static flow entries. Flow entries are not directly available in config mode but are in the config-switch submode (which is entered when a switch object is being edited).

4.3.3. copy

Syntax:
`copy <src> <dst>`

where <src> and <dst> can be:

{{ (running-config | startup-config | localhost://<name> | url)}}

where <url> must start with either http: or ftp:.

Examples:

```
10.0.2.15# copy run startup
10.0.2.15# show config
Name        Length Version Timestamp
-----------|------|-------|-------------------
startup     287    5       2010-12-14.22:55:48

10.0.2.15# copy run startup
10.0.2.15# show config
Name        Length Version Timestamp
-----------|------|-------|-------------------
startup     287    6       2010-12-14.22:56:00

10.0.2.15# show config
Name        Length Version Timestamp
-----------|------|-------|-------------------
startup     287    6       2010-12-14.22:56:00
sampleblock 90     1       2010-12-14.23:07:31
10.0.2.15# show config sampleblock
switch 00:0a:00:24:a8:c4:69:00
  flow-entry ssh-block
    active False
    dst-port 22
10.0.2.15# copy localhost://sampleblock running
Num lines applied: 4
```

Description:
This command copies configuration files from/to various files, including the running-config, the startup-config, local files and URLs for http or ftp servers.

Common examples include:

- copy running-config startup-config: copies the current running configuration to the startup configuration.
- copy localhost://<name> running-config: applies a stored config file to the running configuration. The stored config file can be a "configlet": a partial config file.
- copy running-config localhost://<name>: copies the current running configuration to a named file - useful for "checkpointing" the running config without writing it to startup.

4.3.4. write

Syntax:

```
write ( memory | terminal )
```

Examples:

```
10.0.2.15# write mem
10.0.2.15# write t
!
! Big Switch Controller 0.1
(2011.03.28.1314-build.bigswitchcontroller.release)
!
!
switch 00:00:00:00:00:00:00:01
!
switch 00:00:00:00:00:73:28:03
```

Description:
`write memory` is an alias for `copy running-config startup-config` and `write terminal` is an alias for `show running-config`.

4.4. Config mode commands

4.4.1. cloud

Syntax:

```
cloud connect [auth-token <auth-token-value>] [cluster <cluster-id>]
[sync-period                                   <period-in-secs>]
cloud                                              disconnect
cloud                                                    reset
cloud status
```

Examples

```
10.0.2.15(config)# cloud connect auth-token 3X4E-XYF9-EKX3-GFP7
cluster acme:engineering
Successfully configured controller node for cloud access
10.0.2.15(config)# cloud status
Syncing to the cloud has been configured and is enabled.

Sync Period: 300
Target Cluster: acme:engineering
Cloud Server: update.bigswitch.com

10.0.2.15(config)# cloud disconnect
Successfully disconnected controller node from the cloud
10.0.2.15(config)# cloud connect
Successfully configured controller node for cloud access
10.0.2.15(config)#
```

Description

The `cloud` command is used to configure the controller to upload stats and log data from the controller to a cloud server. Before connecting to the cloud you need to work with your contact at Big Switch Networks to create a customer account on the cloud server and provision user(s) and cluster(s). Once this has been done you should have at least one user/password credential, a customer account name, and a cluster name. You are then ready to use the cloud command to finish the configuration.

The `connect` subcommand is used to specify the information needed to connect to the cloud. There are two required arguments for the initial cloud configuration: `auth-token` and `cluster`. The auth token is used to authenticate the local controller node to the cloud server. To obtain the auth token value:

1. Log in to the cloud server (`https://support.bigswitch.com/`) using the credentials obtained from Big Switch Networks.
2. Navigate to the token generation page using the *Token* link in the upper right corner.
3. Click the "Generate new API token" button. This generates a 16 character API/auth token (e.g. 3X4E-XYF9-EKX3-GFP7).
4. Copy the generated API/auth token value to paste to the CLI cloud command.

The `cluster` argument is a cluster ID, which is the cluster name qualified by the customer name. It is composed of the customer name and the cluster name separated by a colon (e.g. "acme:engineering").

There are a few other optional arguments to the `connect` subcommand:

- `sync-period`: Time (in secs) between successive syncs of data to the cloud. Default is 5 minutes.
- `sync-overlap`: Overlap (in secs) of the time ranges of successive syncs of data to the cloud. Default is 30 secs.
- `cloud-host`: Host name of the cloud server to use to sync data. Default is support.bigswitch.com.

The only one of these you should change typically is the `sync-period` if you want syncing to happen more or less frequently. The other ones you should only change if advised by Big Switch Networks.

After you've configured the controller to sync data to the cloud you can discontinue syncing using the `disconnect` subcommand. This preserves the auth token value and other configuration info but disables the sync process. You can later reenable syncing using the `connect` subcommand again but without without having to specify the `auth-token` or `cluster` arguments.

To completely reset the cloud configuration use the `reset` subcommand. This deletes the cloud configuration info including the auth token value.

Currently the cloud server only supports a single auth token per user. If you regenerate a new auth/API token for the user the previous token value will now be invalid. This means that if you have a controller that was configured to sync to the cloud using the old token value it will no longer be able to authenticate to the cloud. You will need to update the token value using the `connect` subcommand with the *auth-token* argument and the new token value (e.g. `cloud connect auth-token EFG4-5YPW-23BC-88SA`).

4.4.2. controller-node

Syntax:

```
controller-node <controller-node-id>
```

Examples:

```
10.0.2.15(config)# controller-node 127.0.0.1:6633
10.0.2.15(config-controller-node)# show this
IP Address      : 127.0.0.1
Port            : 6633
Active          : True
Start Time      : 2010-12-13 18:36:58.987000
10.0.2.15(config-controller-node)# listen-address 192.168.2.229
```

Description:

This command allows creation and/or update of a specific controller node by placing the user into config-controller-node mode.

The <controller-node-id> IP address + **:** + the port.

In this release, updating or creating controller nodes will not affect the operations of the core Beacon applications. The information is written to the database strictly for informational purposes.

4.4.3. flow-entry

Syntax:

```
flow-entry <name>
```

Note this command must be entered while in a config-switch submode.

Examples:

```
10.0.2.15(config)# switch 00:00:00:00:00:00:00:01
10.0.2.15(config-switch)# flow-entry sample
10.0.2.15(config-flow-entry)# help
For help on specific commands type help <topic>
<...>
Available fields for flow-entry:
  actions
  active
  cookie
  dst-ip
  dst-mac
  dst-port
  ether-type
  hard-timeout
  idle-timeout
  ingress-port
  name
  priority
  protocol
  src-ip
  src-mac
  src-port
  switch
  tos-bits
  vlan-id
  vlan-priority
  wildcards

10.0.2.15(config-flow-entry)# ether-type 2048
10.0.2.15(config-flow-entry)# src-ip <tab-key-pressed>
< Src IP >: IP v4 source address in dotted decimal a.b.c.d w/
optional mask (ex: /24)
10.0.2.15(config-flow-entry)# src-ip 10.0.0.3
10.0.2.15(config-flow-entry)# dst-ip 10.0.0.4
10.0.2.15(config-flow-entry)# actions drop
```

D e s c r i p t i o n :

This command creates or updates a static flow entry for a switch. It also puts the user into a config-flow-entry submode, where specific fields for the flow entry can be configured (see list of available fields above). A flow entry has header fields, match rules, and actions - each of these can be configured as fields in a flow entry in the configuration. The list of options and specific format can be obtained using tab-completion in the CLI (note in the example above where it says tab-key-pressed).

More details on these fields can be found in the OpenFlow specification.

Header fields:

- priority - 16-bit unsigned integer, higher has more priority. By default, the priority of a statically defined flow entry will be higher than the flow entries of the other applications, such as Learning

Switch and Routing.

- cookie - 32-bit integer
- hard-timeout - not used/hardwired in this release
- idle-timeout - not used/hardwired in this release
- active - True/False

Match rules:

- wildcards - not used/automatically determined in this release
- src-mac,dst-mac - MAC addresses in xx:xx:xx:xx:xx:xx format
- src-ip, dst-ip - IP addresses in dotted decimal format with an optional mask
- ether-type - 16-bit unsigned integer. Common values are 2048: IP, 2054:ARP. This field is required in order for matching on IP address or ports.
- protocol - 8-bit unsigned integer. Common values are 1: icmp, 6: tcp, 17: udp for IP ether-type.
- src-port/dst-port - 16-bit unsigned integer. Corresponds to TCP/UDP ports or ICMP type/code.
- vlan-id/vlan-priority - 16-bit unsigned integer and 8-bit unsigned integer.

NOTE

- Match rules that include information for higher-layer fields must include the fields from lower layers or else they will not be applied. For example, a common mistake is to match just on src-ip address and not specify the ether-type as being IP (2048). In this case, the switch will decide it doesn't not know how to interpret the src-ip address and it will wildcard the src-ip address field.

A c t i o n s :
This is a comma-separated list of actions, where the actions include:

- drop
- output=<value> where value is one of:
  - <port number>
  - normal
  - flood
  - all
  - local
  - ingress-port
- enqueue=<openflow port id>:<queue id>
- strip-vlan
- set-vlan-priority=<priority>
- set-vlan-id=<vlan id>
- set-src-mac=<mac address>
- set-dst-mac=<mac address>
- set-tos-bits=<tos bits values>
- set-src-ip=<ip address>
- set-dst-ip=<ip address>
- set-src-port=<port number>
- set-dst-port=<port number>

Examples:

- To forward traffic to output ports 3 and 4, use the command:

```
actions output=3,output=4
```

> **NOTE**
>
> - 'actions' command is not currently validated on entry and will allow typos, mistakes, etc. to be entered. Please use care to enter the command correctly.
> - If no actions are specified for the flow-entry, the 'drop' action is implied

Flow entries will not appear on the switch unless the active field has been set to True. Once active has been set to True, the Static Flow application will push this down to the switch every 10 seconds with a hard-timeout of 30 seconds.

For example, below is a simple flow entry that blocks traffic for a specific src-ip and dst-ip - note it has no actions specified, and so by default, the action is drop

```
switch 00:00:00:00:00:00:00:01
  flow-entry deny-h4-h5
    active True
    ether-type 2048
    src-ip 10.0.0.4
    dst-ip 10.0.0.5
```

This flow entry appears on the switch as such:

```
10.0.2.15# show switch 00:00:00:00:00:00:00:01 flow
Switch                  Bytes Packets Dur(s) Cookie            In
Port Src MAC Dst MAC Ether Type Src IP   Dst IP   Protocol Src
Port Dst Port
-----------------------|-----|-------|------|-----------------|------
0     0       29     static-deny-h4-h5 *       *       *
ip(2048)   10.0.0.4 10.0.0.5 *        *        *
```

Note the wildcarded fields and note the duration field will not usually exceed 10 seconds so long as the flow entry is active.

If the user changes the match rules or deletes the flow entry, then the duration field will count up to the hard-timeout of 30 seconds and then the flow entry will expire and be deleted from the switch.

4.4.4. host

S y n t a x :

```
host <MAC address>
```

Examples:

```
10.0.2.15(config)# host 00:00:00:73:28:03
10.0.2.15(config-host)# name ceo-macbook-pro
10.0.2.15(config-host)# sh this
MAC Address  : 00:00:00:73:28:03
Switch ID    : 2814895659608320
Ingress Port : 43
IP Address   : 0.0.0.0
Name         : ceo-macbook-pro
```

D e s c r i p t i o n :
This command allows creation and/or update of a specific host by placing the user into config-host mode. Hosts are written into the database by the Device Manager application with their MAC address and switch/port that they were discovered on.

Currently, the name is blank but can be overridden manually. Switch ID and Ingress Port can also be overwritten and this will change the behavior of the Routing application as the host will now be found on a different port. For example, suppose host A on switch 1, port 1 is trying to reach host B on switch 2, port 2. If the user edits host B and configures it to be on switch 3, port 3, and a packet-in comes to the controller needing a route from A to B, then the Routing application will push down flow mods to get to switch 3, port 3.

4.4.5. link

S y n t a x :

```
link <link-id>
```

Examples:

```
10.0.2.15> show link
Src Switch DPID         Src Port Dst Switch DPID         Dst Port
----------------------|--------|----------------------|--------
00:00:00:00:00:00:00:01 15       00:0a:00:24:a8:c4:69:00 61
00:00:00:00:00:00:00:01 9        00:0a:00:21:f7:de:e9:00 43
00:0a:00:21:f7:de:e9:00 43       00:00:00:00:00:00:00:01 9
00:0a:00:24:a8:c4:69:00 61       00:00:00:00:00:00:00:01 15
10.0.2.15> enable; conf
10.0.2.15(config)# link
00:00:00:00:00:00:00:01-15-00:0a:00:24:a8:c4:69:00-61
10.0.2.15(config-link)# show this
Src Switch DPID : 00:00:00:00:00:00:00:01
Src Port        : 15
Dst Switch DPID : 00:0a:00:24:a8:c4:69:00
Dst Port        : 61
10.0.2.15(config-link)#
```

Description:
This command allows creation and/or update of a specific link by placing the user into config-link mode. Links are written into the database by the Topology application as it discovers the connectivity between the switches.

A link id is the concatenation of the source switch DPID/port and destination switch DPID/port, joined by the – character. See example above.

In this release, updating or creating links will not affect the operations of the Topology application. The information is written to the database strictly for informational purposes.

4.4.6. port

Syntax:
`port [ <port-id> ]`

Examples:

```
10.0.2.15# show port | grep -e 24 -e Switch -e '---'
Switch DPID             OF #  Name  MAC Address
Configuration State Current Advertised Supported Peer
----------------------|-----|-----|-----------------|--------------|
24    24    00:26:e1:81:13:18                0x1
00:0a:00:24:a8:c4:69:00 52    28    00:24:a8:c4:69:cc
00:0a:00:24:a8:c4:69:00 54    30    00:24:a8:c4:69:ca
00:0a:00:24:a8:c4:69:00 56    32    00:24:a8:c4:69:c8
00:0a:00:24:a8:c4:69:00 57    33    00:24:a8:c4:69:c7
00:0a:00:24:a8:c4:69:00 58    34    00:24:a8:c4:69:c6
00:0a:00:24:a8:c4:69:00 61    37    00:24:a8:c4:69:c3
00:0a:00:24:a8:c4:69:00 64    40    00:24:a8:c4:69:c0
00:0a:00:24:a8:c4:69:00 65534 local 00:24:a8:c4:69:00
00:0a:00:24:a8:c4:69:00 67    43    00:24:a8:c4:69:bd
10.0.2.15# conf
10.0.2.15(config)# port 00:0a:00:24:a8:c4:69:00:67
10.0.2.15(config-port)# show this
Switch DPID         : 00:0a:00:24:a8:c4:69:00
OF #                : 67
Name                : 43
MAC Address         : 00:24:a8:c4:69:bd
Configuration      : 0
State               : 0
Current             :
Advertised          :
Supported           :
Peer                :
10.0.2.15(config-port)#
```

Description:
This command allows creation and/or update of a specific port by placing the user into config-port mode.

Ports are written along with switches by the core Beacon application as it discovers switches connecting to it.

A port id is the concatenation of the switch DPID and the logical OpenFlow port number joined by a **:**.

In this release, updating or creating ports will not affect the operations of the core Beacon applications. The information is written to the database strictly for informational purposes.

4.4.7. switch

S y n t a x :

```
switch <switch-dpid>
```

Examples:

```
10.0.2.15(config)# switch 00:0a:00:21:f7:de:e9:00
10.0.2.15(config-switch)# show this
Switch DPID       : 00:0a:00:21:f7:de:e9:00
Active            : True
Last Connect Time : 2010-12-15 01:21:33.667000
IP Address        : 192.168.2.102
Socket Address    : /192.168.2.102:41886
Max Packets       : 256
Max Tables        : 2
Capabilities      : 135
Actions           : 2047
```

D e s c r i p t i o n :

This command allows creation and/or update of a specific switch by placing the user into config-switch mode. Switches are written along with ports by the core Beacon application as it discovers switches connecting to it.

In this release, switches appear in the running config as they connect to the controller. The main purpose of this command is allow the user to define static flow entries for a particular switch.

The "alias" field can be configured while in config-switch mode. This alias will be usable as an argument to show switch commands (instead of a full 8-byte DPID) and will be displayed instead of the DPID in various show commands.

4.4.8. no

S y n t a x :

```
no ( controller-node | flow-entry | host | link | port | switch ) <id>
```

Examples:

```
10.0.2.15(config-switch)# show run | begin switch
switch 00:00:00:00:00:00:00:01
  flow-entry deny-h4-h5
    active False
    ether-type 2048
    src-ip 10.0.0.4
    dst-ip 10.0.0.5
!
switch 00:00:00:00:00:00:00:02
!
switch 00:00:00:00:00:00:00:03
10.0.2.15(config-switch)# no flow-entry deny-h4-h5
10.0.2.15(config-switch)# show run | begin switch
switch 00:00:00:00:00:00:00:01
!
switch 00:00:00:00:00:00:00:02
!
switch 00:00:00:00:00:00:00:03
```

D e s c r i p t i o n :
This command allows the deletion of objects in the database. There is no confirmation, so this command should be used with caution. Operations of the controller will not be affected by the deletion of some of the informational-only data as specified earlier (host, link, port, controller node). However, deleting switches and flow entries will change the flow entries that are on the switches and may affect forwarding.

5. REST API

5.1. Introduction

This document describes the REST API supported by the Big Switch controller platform. The REST API provides access to the network database that includes both the configuration data for the controller (e.g. static flow table entries) as well as the data for dynamically created network entities (e.g. switches, hosts) populated by the beacon controller.

The REST API is implemented in a separate process (i.e. "bigcon") from the beacon controller. The beacon controller has its own REST API for obtaining real-time data like flow table entries. This document only describes the network database REST API, not the beacon REST API.

The REST API is accessed on port 8000 of the controller machine/VM. If the client application is running on a remote machine, then the firewall settings on the controller machine need to be configured to allow access to port 8000:

1. ssh to the controller machine: "ssh admin@<controller-ip>". This will put you in the Big Switch CLI.
2. Escape out to bash: "debug bash"
3. Allow access on port 8000: "sudo ufw allow 8000"

5.2. Querying Items

Items are queried using the HTTP GET method. The form of the URL is: http://<domain-or-ip-address>:8000/rest/<data-type>/<optional-id>/?<optional-query-params>

The data types currently supported are:

- **controller-node**: The individual controller nodes/instances in the overall controller platform
- **switch**: The switches associated with this controller
- **port**: The port info for the switches
- **host**: The host devices connected to the switches associated with this controller
- **link**: The links between switches
- **flow-entry**: The statically configured flow table entries

[TBD: Should have info about the fields for each of the data types. For now you can just experiment with queries for each of the types and look at what fields are there. The fields should be pretty self-explanatory for controller-node, host, and link. For switch, port and flow-entry you can look at the descriptions of the ofp_switch_features, ofp_phy_port and ofp_match structs in the OpenFlow spec. The fields in the REST API data types pretty much mirror those structs.]

If the optional ID and query params are omitted, then the REST API returns a list of all of the items of the given data type. The format of the return data is JSON. So, for example, to get all of the controller nodes:

URL	http://localhost:8000/rest/controller-node/
Return Text	[{"active": true, "listen-address": "192.168.1.1", "listen-port": 6633, "last-startup": "2010-12-29 19:28:03.429000", "id": "192.168.1.1:6633"}]

This is assuming the REST call is being made from the controller VM so it can use localhost for the IP address; otherwise use the actual IP address of the controller VM instead of the loopback address.

If you know the unique ID of a data item you can retrieve just that item by including the optional ID in the URL path:

URL	http://localhost:8000/rest/switch/00:00:00:00:00:00:00:01/
Return Text	{"tables": 3, "socket-address": "/192.168.2.29:60286", "connected-since": "2010-12-30 23:24:21.025000", "capabilities": 71, "controller": "192.168.1.1:6633", "actions": 2049, "ip-address": "192.168.2.29", "dpid": "00:00:00:00:00:00:00:01", "active": true, "buffers": 256}

Note that this returns a single item, not a list of items (i.e. no enclosing square brackets)

The unique ID fields for the different data types are:

Data Type	ID Field Name	Description
controller-node	id	<controller IP address>:<listen port>
switch	dpid	Datapath ID of the switch

port	id	<switch DPID>:<port #>
link	id	<src switch DPID>~~<src port>~~<dst switch DPID>-<dst port>
host	mac	MAC address of the host
flow-entry	name	User-defined name of the static flow entry

You can also do more complex queries using the optional query params. Note that if you do a query with the query params you should not set the optional ID in the URL path; those query modes are mutually exclusive. The format of the query params is the same as for standard query params: a semicolon-separated list of <name>=<value> pairs. In the simplest case the <name> is the name of a field in the data type being queried and the <value> is the value of that field you want to filter the query with. For example, to get all of the hosts connected to the switch whose DPID is 00:00:00:00:00:00:00:01:

URL	http://192.168.1.1:8000/rest/host/?switch=00:00:00:00:00:00:00:01
Return Text	[{"switch": "00:00:00:00:00:00:00:01", "mac": "00:11:11:e2:45:b7", "name": "", "ip-address": "192.168.1.19", "ingress-port": 1}, {"switch": "0:00:00:00:00:00:00:01", "mac": "00:16:cb:94:49:25", "name": "", "ip-address": "192.168.1.23", "ingress-port": 2}]

If you include multiple name/value pairs then the return text is the list of items that satisfies all of the conditions (i.e. the conditions are AND'd together).

You can also do more complex filtering operations using a special form of the name in the name/value pairs: <field-name>__<operation>. Note that that's a double underscore between the field name and the operation. For example to get all of the hosts whose MAC address starts with 00:00:00 you can use the startswith operation:

URL	http://localhost:8000/rest/host/?mac__startswith=00:00:00
Return Text	[{"switch": "00:0a:00:24:a8:c4:69:00", "mac": "00:00:00:00:00:01", "name": "", "ip-address": "192.168.1.19", "ingress-port": 1},{"switch": "00:0a:00:24:a8:c4:69:00", "mac": "00:00:00:00:00:02", "name": "", "ip-address": "192.168.1.23", "ingress-port": 2}]

The supported filtering operations are:

Operation Name	Description
exact	Exact match of specified value. Equivalent to setting the name to just the field name
iexact	Case insensitive match of the specified value
contains	The field contains the specified value, case-sensitive
icontains	The field contains the specified value, case-insensitive
startswith	The field starts with the specified value, case-sensitive
istartswith	The field starts with the specified value, case-insensitive
endswith	The field ends with the specified value, case-sensitive

iendswith	The field ends with the specified value, case-insensitive
gt	The field is greater than the specified value
gte	The field is greater than or equal to the specified value
lt	The field is less than the specified value
lte	The field is less than or equal to the specified value
regex	The field matches the specified regular expression, case-sensitive
iregex	The field matches the specified regular expression, case-insensitive

The syntax of the regular expressions for the regex and iregex operations is the Python regular expression syntax.

There are also some special query parameters that can be used to affect how the results are returned from a query:

Query Parameter Name	Description
callback	Return query results in the JSONP format with the specified value as the JSONP function name. Google "JSONP" for more info on why you'd want to use this feature.
nolist	If specified value is true (or True or 1), then return a single item, not a list (i.e. no enclosing square brackets]. This returns an error if there's not exactly one item that satisfies the query parameters.
orderby	Order the results by the specified value. The value is a comma-separated list of field names to use to sort the results, in decreasing precedence. So, for example, to order the results from a host query first on the switch, then on the ingress port, the orderby value would be "switch,ingress-port".

5.3. Creating Items

New items can be created using the HTTP PUT method with a URL path of: "/rest/<data-type>/". The PUT data is the JSON formatted data for the item (or items) to be created. Each item should be a JSON object (i.e. comma-separated list of field name/value pairs bracketed with curly brackets) with the fields of the given data type specified. In some cases some of the fields are optional and can be omitted. [TBD: Need descriptions of which fields are optional for each of the data types.] You can insert multiple items in a single REST call by specifying a JSON array of objects (i.e. comma-separated list of JSON objects bracketed with square brackets). Basically the format of the data is the same as you see when you query for items as described in the previous section.

Note that this use of the PUT method to create items isn't the conventional REST way of creating items (normal way is to use the POST method). But using the POST method with POST parameters it wouldn't be possible to create multiple items in one call, which is why we use the PUT method instead.

Most of the data types currently exposed in the REST API aren't intended to be modified by third party apps, only by the controller. You don't want to make changes to the controller-node, switch, port, and link types, unless you really know what you're doing. Soon there will be a real security model for the REST API with authentication/authorization that will prevent unprivileged apps from modifying many of these things, but until then, be aware that you can really mess up the controller if you change these things.

For now the main thing that third party apps will modify/create are the static flow table entries.

5.4. Updating Items

Items are updated using the HTTP PUT method. The difference between updating vs. creating is that the item (or items) to be updated must be specified using either the option ID component of the URL path or query parameters. If you use the option ID in the URL path, then only that single item is updated. If you use query parameters, then any items matching the query parameter conditions are updated. The PUT data is a JSON object that includes the field names/values to be updated. The return value is the string "saved" if the update succeeded. [Note: This will probably change sometime soon to return JSON-formatted text indicating that the operation succeeded to be consistent with the other calls]. See the "Error Handling" section below for information about how errors are handled.

For example, to set the name of the host whose MAC is "00:00:00:00:00:01" to "Test Server":

URL	http://localhost:8000/rest/host/00:00:00:00:00:01/
PUT Data	{"name":"Test Server"}

And to set the name of all hosts whose MAC starts with "c4:2c:03" to "MacBook":

URL	http://localhost:8000/rest/host/?mac__startswith=c4:2c:03
PUT Data	{"name":"MacBook"}

5.5. Deleting Items

Items are deleted using the HTTP DELETE method. Similar to updating items you specify the item(s) to be deleted using either the optional ID component in the URL path (to delete a single item) or the optional query parameters (to delete multiple items). Just to repeat the earlier warning: The only data type you should delete currently is the flow-entry data type. Bad things will probably happen if you delete switch, port, host, link info out from under the controller. The return value is the string "deleted" if the delete operation succeeded. [Note: This will probably change sometime soon to return JSON-formatted text indicating that the operation succeeded to be consistent with the other calls]. See the "Error Handling" section below for information about how errors are handled.

For example to delete the flow entry named "testping":

URL	http://localhost:8000/rest/flow-entry/testping/

5.6. Storing Data Blobs

The REST API also supports storing & retrieving arbitrary data blobs from the network database. For example, the Big Switch CLI uses this to store/archive configuration scripts in the database. The format of the URL for the data blobs is: "http://<domain>:8000/rest/data/<blob-name>/". To store blob data you use the HTTP PUT method where the PUT data is the raw blob data. To retrieve data you use the HTTP GET method. The data that's returned is the exact data that was stored in the earlier PUT call (i.e. there's no enclosing JSON formatting or anything else).

When storing data the REST API implementation looks for the Content-Type header in the HTTP request. If it's specified, then this value is saved along with the data and returned as the Content-Type header when the data is retrieved. You can also specify a query parameter of "binary" set to "true" to indicate that the data is binary data. In this case the REST API will set the content type to be "application/octet-stream". If neither the Content-Type header nor the binary query parameter is specified the data is assumed to be text data and the content type is set to "text/plain".

There's no particular format for the blob name imposed by the REST API. It can contain alphanumeric characters as well as any of the following punctuation characters: "_:./=;-". By convention it's useful to think of the name as an absolute path in a virtual file system with '/' as the path separator. That way you can group different types of data in logical directories and query for data blobs (described below) using the startswith filter operation. Currently we don't have any guidelines for reserved names in the global namespace, but for now you should prefix any blob names with something that you are reasonably certain will be globally unique to avoid conflicts with how Big Switch or another third party uses the data blob namespace.

You can also query to get a list of blobs matching some specified criteria. In this case the URL is "http://<domain>:8000/rest/data/?<query-params>" where the query parameters specify the criteria similar to querying other types of data in the REST API. The following fields are available for querying:

Field Name	Description
name	The name of the data blob
content_type	The content type of the data blob

The return value is a JSON formatted list of data blob references, where each blob reference contains the name of the blob and the URL path to access that data blob.

For example, to get all of the data blobs whose names starts with "/bigswitch/config/":

URL	http://localhost:8000/rest/data/?name__startswith=/bigswitch/config/
Return Text	[{"name":"/bigswitch/config/running", "url_path":"rest/data/bigswitch/config/running"},{"name":"/bigswitch/config/backup", "url_path":"rest/data/bigswitch/config/backup"}]

5.7. Retrieving Stats Data

The controller collects time-sampled stats from the controller node and the switches that are connected to the controller.

These stats data are retrieved using the following URL:

http://<domain-or-ip-address>:8000/rest/v1/stats/<target-type>/<customer-id>/<target-id>/<stat-type>?

The return text is a JSON-formatted array of time/value tuples or error info (see error handling section below). The time value is an integer UTC time in milliseconds since the epoch. The value is dependent on the type of stats value being retrieved. For example, the return text for a query for the mem-used stat type might be something like:

[[1303307359476, 870156], [1303308019478, 870416], [1303311619488, 869844]]

The <target-type> is the type of the source of the stats data. The <target-id> identifies a specific instance of the target type. Currently there are two target types:

Target Type	Description	Target ID
controller	Stats associated with a controller node	The ID of the controller node. Currently we only support a single node controller, so the ID is hard-coded to be "localhost"
switch	Stats associated with a switch	The DPID of the switch, e.g. 00:00:00:00:00:00:00:01

The <customer-name> is included for future multi-tenant stats support. Currently we only support a single tenant and the customer ID is hard-coded to be "default"

The <stat-type> identifies the specific time series stats data being retrieved, e.g. "cpu-user", "OFPacketIn").

The <optional-query-params> are used to specify the time range for the query and how the data is downsampled to return a manageable number of data points. The raw sample data for different stat types is sampled at frequencies that are configured for the controller process that collects stats. Currently these are only configurable by editing a config file on the controller system, but eventually we will support configuration via the CLI or web UI. The raw sampling frequencies can be as frequent as every few seconds. If an application retrieves stats data over a much longer period (e.g. days, weeks), then it would be too much data to return the raw sample data, so the REST API supports downsampling of the data, as specified in the query parameters described below.

The following query params are supported:

Query Param	Description
start-time	The start time for the data to return.
end-time	The end time for the data to return. Time format is the same as start-time
sample-interval	The interval between sample points
sample-count	The number of sample points returned
sample-window	The time window around each sample point used to calculate the value

data-format	The format of the returned data

All time values, both specific times and time intervals, are integer values in milliseconds. For specific times, the value is UTC time in milliseconds since the epoch (where epoch = Unix epoch = midnight 1/1/1970 UTC). For example, in Python this is: "int(time.time()*1000)". In general the epoch is system-dependent, so it may be necessary to adjust the time value to use time values that are consistent with what the controller uses.

If neither the start-time nor end-time are specified, then the REST call returns the most recently collected sample point. In this case the return value is a single data point (e.g. "[1303311619488, 869844]") not a list of sample points.

The sample-interval and sample-count params are mutually exclusive. Specifying the sample-count is equivalent to specifying a sample-interval of: (end-time - start-time) / sample-count. If the sample-interval is 0, then the raw data samples are returned. If neither the sample-interval nor sample-count params are specifies, then the REST call defaults to a sample-count value of 50.

The controller only supports certain pre-defined downsampling intervals. These are: 1 minute, 10 minutes, 1 hour, 4 hours, 1 day, 1 week and 4 weeks. The sample-interval (determined as described above) is rounded down to the nearest pre-defined sampling interval.

Currently two data-format values are supported:

Data Format	Description
value	The sample value
rate	The rate at which the sample value is changing (in units per millisecond)

The "value" data format returns the value of the sample point. For REST calls with downsampling, if the sample-window is 0, then each returned value is the value of the raw data point at the beginning of each downsampled interval. If the sample-window is non-zero, then the value return is the average of all of the raw sample data points over the specified window, centered around the time of the returned data point.

The "rate" data format returns the rate at which the sample value is changing. If the sample-window is 0, then the rate is calculated using the previous sample point, i.e. rate = (current.value - previous.value) / (current.time - previous.time)). The rate is returned in units per millisecond, where the units depend on the stats type being returned. If the sample-window is non-zero, then the rate is calculated using the first and list sample points in the specified window, centered around the time of the returned data point.

The following controller-related stat types are supported:

Stat Name	Description
cpu-idle	The CPU idle percentage reported by top
cpu-nice	The CPU nice percentage reported by top
cpu-user	The CPU user percentage reported by top

cpu-system	The CPU system percentage reported by top
mem-used	The memory used reported by top
mem-free	The memory free reported by top
swap-used	The swap used reported by top

The following switch-related stat types are supported:

Stat Name	Description
OFPacketIn	The count of OpenFlow packet in events received by the controller from the switch
OFFlowMod	The count of OpenFlow flow mod commands sent from the controller to the switch

The values are counters since the last time the controller was restarted.

The flow mod counter includes commands that add or delete flow mods on the switch. Due to the way the controller handles flow mods there are typically 3 flow mod commands per packet in event that is forwarded.

Here are some examples:

To get the latest mem-used

URL	http://<ip>:8000/rest/v1/stats/controller/default/localhost/mem-used

To get the last hour of mem-used:

URL	http://<ip>:8000/rest/v1/stats/controller/default/localhost/mem-used?start-time=1302888516260

To get the most recent OFPacketIn value for switch 00:00:00:00:00:73:28:02:

URL	http://<ip>:8000/rest/v1/stats/switch/default/00:00:00:00:00:73:28:02/OFPacketIn

To get the last day of OFPacketIn:

URL	http://<ip>:8000/rest/v1/stats/switch/default/00:00:00:00:00:73:28:02/OFPacketIn?start-time=130

To get the last day of OFPacketIn rates:

URL	http://<ip>:8000/rest/v1/stats/switch/default/00:00:00:00:00:73:28:02/OFPacketIn?start-time=130

5.8. Error Handling

If there's an error processing the REST request, then the REST API returns an appropriate HTTP error code and the body of the response is a JSON object containing several fields:

Field Name	Description
error_type	The type of the error/exception. This maps to the name of the exception that was thrown by the REST implementation.
description	A human-readable description of the error
model_error (optional)	The model-level validation error
field_errors (optional)	The per-field validation errors

The model_error and field_errors fields are only set if the error occurred during validation of the input data when creating or updating a data item. You get these errors if the format of the input doesn't match the expected format of the field you're setting (e.g. trying to set an IP address field to something that doesn't have the format of an IP address).

5.9. Sample Code

5.9.1. Python

```
import json
import urllib2

def construct_rest_url(host, type, id=None, query_params=None):
    """
    Construct the REST URL for the given host/type/id, including the
    items in the query_params dict as URL-encoded query parameters
    """
    url = 'http://%s:8000/rest/%s/' % (host, type)
if id:
       url += id
       url += '/'
    if query_params:
        url += '?'
        url += urllib.urlencode(query_params)
    return url

def print_rest_error_info(e):
    """
    Extract the error information and print it.
    This is mainly intended to demonstrate how to extract the
    error info from the exception. It may or may not make sense
    to print out this information, depending on the application.
    """
    # Extract the info from the exception
    error_code = e.getcode()
    response_text = e.read()
    obj = json.loads(response_text)
    error_type = obj.get('error_type')
    description = obj.get('description')

    # Print the error info
    print 'HTTP error code = ', error_code, '; error_type = ', error_type, '; description = ',
description

    # Print the optional validation error info
    model_error = obj.get('model_error')
```

```
    if model_error:
        print 'model_error = ', model_error
    field_errors = obj.get('field_errors')
    if field_errors:
        print 'field_errors = ', field_errors

def get_rest_data(host, type, id=None, query_param_dict=None):
    """
    Get the data for the given type/id/params from the give host.
    If both id and query_param_dict are None, then the return value
    is the list of all objects of the given type, represented as an
    array of dicts. If the id is set, the the return value is the
    single object with that id, represented as a dict. If filtering
    parameters are specified in the query_param_dict, thenn the return
    value is the list of objects that match the filtering criteria.
    """
    url = construct_rest_url(host, type, id, query_param_dict)
    request = urllib2.Request(url)
    try:
        response = urllib2.urlopen(request)
        response_text = response.read()
        obj = json.loads(response_text)
        return obj
    except urllib2.HTTPError, e:
        print_rest_error_info(e)
        raise e

def put_rest_data(obj, host, type, id=None, query_param_dict=None):
    """
    Put the given object data to the given type/id/params at the given host.
    If both the id and query_param_dict are empty, then a new item is created.
    Otherwise, existing data items are updated with the object data.
    """
    url = construct_rest_url(host, type, id, query_param_dict)
    post_data = json.dumps(obj)
    request = urllib2.Request(url, post_data, {'Content-Type':'application/json'})
    request.get_method = lambda: 'PUT'
    try:
        response = urllib2.urlopen(request)
        return response.read()
    except urllib2.HTTPError, e:
        print_rest_error_info(e)
        raise e

def delete_rest_data(host, type, id=None, query_param_dict=None):
    """
    Delete data items of the given type/id/params at the given host.
    If both id and query_param_dict are None, then all objects of the
    given type are deleted. If the id is set, then the object with that
    id is deleted. If filtering parameters are specified in the
    query_param_dict, then the objects that match the filtering criteria
    are deleted.
    """
    url = construct_rest_url(host, type, id, query_param_dict)
    request = urllib2.Request(url)
    request.get_method = lambda: 'DELETE'
    try:
        response = urllib2.urlopen(request)
        return response.read()
    except urllib2.HTTPError, e:
        print_rest_error_info(e)
        raise e

# Get all hosts
hosts = get_rest_data('localhost', 'host')

# Get host with MAC address of 00:00:00:00:00:01
one_host = get_rest_data('localhost', 'host', '00:00:00:00:00:01')

# Update the name of the host whose MAC address is 00:00:00:00:00:01
name_info = {'name':'test'}
saved_message = put_rest_data(name_info, 'localhost', 'host', '00:00:00:00:00:01')
```

```
# Delete the static flow table entry named "test1"
deleted_message = delete_rest_data('localhost', 'flow-entry', 'test1')

# Delete all static flow table entries whose name starts with "test"
deleted_message = delete_rest_data('localhost', 'flow-entry', None,
{'name__startswith':'test'})
```

5.9.2. Bash Shell with curl

Getting the list of models and getting a list of switches and flow-entries:

```
$ curl http://192.168.2.229:8000/rest/
[{"url_path": "rest/controller-node/", "name": "controller-node"}, {"url_path": "rest/switch/",
"name": "switch"}, {"url_path": "rest/host/", "name": "host"}, {"url_path": "rest/link/",
"name": "link"}, {"url_path": "rest/flow-entry/", "name": "flow-entry"}, {"url_path": "rest
/port/", "name": "port"}]

$ curl http://192.168.2.229:8000/rest/switch/
[{"tables": 3, "socket-address": "/192.168.2.28:41387", "connected-since": "2010-12-29
19:28:06.665000", "capabilities": 71, "controller": "127.0.0.1:6633", "actions": 2049,
"ip-address": "192.168.2.28", "dpid": "00:00:00:00:00:00:00:01", "active": true, "buffers":
256}, {"tables": 2, "socket-address": "/192.168.2.102:40973", "connected-since": "2010-12-29
19:28:07.334000", "capabilities": 135, "controller": "127.0.0.1:6633", "actions": 2047,
"ip-address": "192.168.2.102", "dpid": "00:0a:00:21:f7:de:e9:00", "active": true, "buffers":
256}, {"tables": 2, "socket-address": "/192.168.2.101:65152", "connected-since": "2010-12-29
19:28:07.116000", "capabilities": 135, "controller": "127.0.0.1:6633", "actions": 2047,
"ip-address": "192.168.2.101", "dpid": "00:0a:00:24:a8:c4:69:00", "active": true, "buffers":
256}]

$ curl http://192.168.2.229:8000/rest/flow-entry/
[{"wildcards": 0, "protocol": 1, "name": "pingblock", "idle-timeout": 10, "actions": "drop",
"priority": 32768, "switch": "00:0a:00:21:f7:de:e9:00", "cookie": 0, "ether-type": 2048,
"active": false, "hard-timeout": 10}]
```

Adding a flow-entry (note the PUT in the curl command-line and watching the quoting of the JSON) and then checking the list of flow-entries again:

```
$ curl -X PUT -d '{"name":"testing", "src-ip":"192.192.192.1", "active":"True", "switch":
"00:00:00:00:00:00:00:01", "ether-type":"2048"}' http://192.168.2.229:8000/rest/flow-entry/
saved

$ curl http://192.168.2.229:8000/rest/flow-entry/
[{"wildcards": 0, "protocol": 1, "name": "pingblock", "idle-timeout": 10, "actions": "drop",
"priority": 32768, "switch": "00:0a:00:21:f7:de:e9:00", "cookie": 0, "ether-type": 2048,
"active": false, "hard-timeout": 10}, {"wildcards": 0, "name": "testing", "idle-timeout": 60,
"src-ip": "192.192.192.1", "priority": 32768, "switch": "00:00:00:00:00:00:00:01", "cookie": 0,
"ether-type": 2048, "active": true, "hard-timeout": 0}]
```

Just for reference, this newly added flow-entry looks like this in the running config and in the show command in the CLI:

```
192.168.2.229:8000> sh run | begin switch | head -6
switch 00:00:00:00:00:00:00:01
  flow-entry testing
    active True
    ether-type 2048
    src-ip 192.192.192.1
!
192.168.2.229:8000> show switch 00:00:00:00:00:00:00:01 flow
Bytes Packets Dur(s) Cookie         In Port Src MAC Dst MAC Ether Type Src IP        Dst IP
Protocol Src Port Dst Port
-----|-------|------|--------------|-------|-------|-------|----------|-------------|------|------
0       10     static-testing *       *       *       ip(2048)   192.192.192.1 *      *
*        *

192.168.2.229:8000>
```

There are also REST commands at the console (port 8000) to get the realtime data from Beacon (which in turn gets it directly from a switch), but the formatting currently isn't done in the django/port 8000 REST layer (formatting is currently done in the CLI), so the output may not be directly understandable. Still, it's useful sometimes to get it from the switch:

```
$ curl http://192.168.2.229:8000/rest/realtimestats/desc/00:00:00:00:00:00:00:01/
[{"serialNumber":"None","manufacturerDescription":"Stanford University","hardwareDescription":
"Reference Userspace Switch","softwareDescription":"1.0.0","datapathDescription":"GSM-7328-1
pid=795","length":1056}]

$ curl http://192.168.2.229:8000/rest/realtimestats/flow/00:00:00:00:00:00:00:01/
[{"match":{"dataLayerDestination":"AAAAAAAA","dataLayerSource":"AAAAAAAA","networkSource"
:-1061109759,"dataLayerType":2048,"dataLayerVirtualLan":0,
"dataLayerVirtualLanPriorityCodePoint":0,"inputPort":0,"networkDestination":0,
"networkDestinationMaskLen":0,"networkSourceMaskLen":32,"networkProtocol":0,
"networkTypeOfService":0,"transportDestination":0,"transportSource":0,"wildcards":4178159},
"cookie":45732403745914880,"idleTimeout":0,"durationSeconds":7,"durationNanoseconds":192000000,
"packetCount":0,"byteCount":0,"hardTimeout":30,"tableId":0,"length":88,"actions":[],"priority"
:-32768}]
```