Project Status for Elf

Reporter: Ze Yu and Xiaolin Li

1. Current Project Status

The project is completed. We have built a fully-functional SDN service and integrated it with Hadoop (see the documentation for details). We have deployed in our local datacenter to test its functionality.

Some facts of this project:

Lines of code: ~3000 lines of Java code

New Java classes: ~40 new classes

2. Test results

We planned to test it on both local datacenter and ExoGeni testbed. Here are our testing results.

2.1 Local datacenter

2.1.1 Environments

We deployed our application on a local cluster consisting of 8 nodes. These nodes are distributed across 8 different racks. These racks are then connected into a tree-topology like most datacenter networks. Two racks form a datacenter. Thus, we have 4 datacenters in total. The topology is shown in Figure. 1 and devices are listed in Figure. 2.

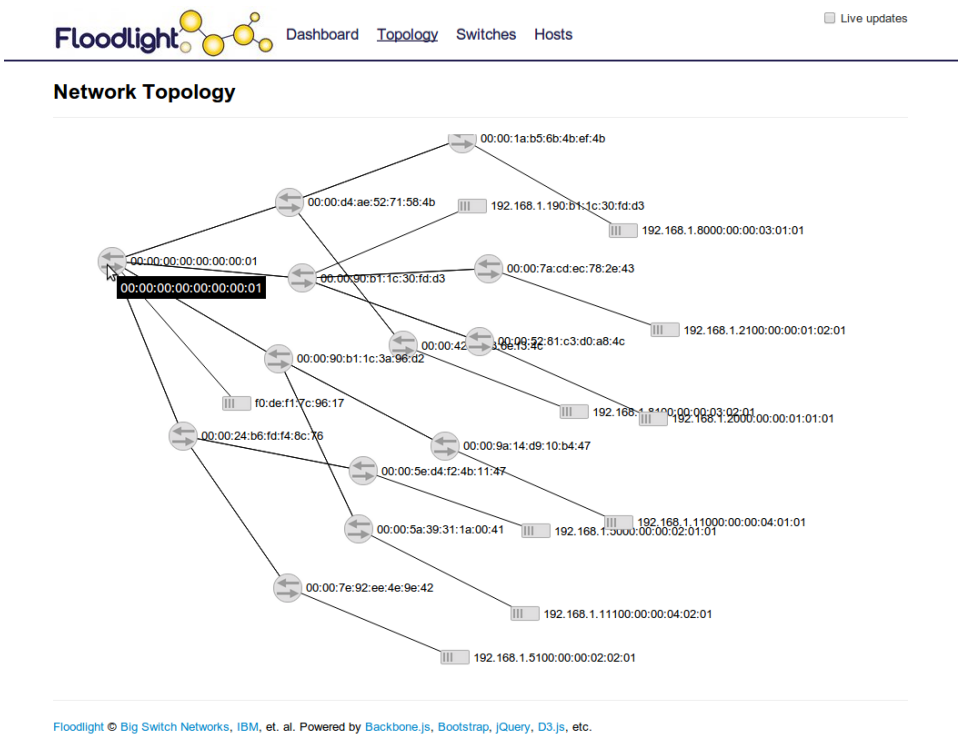

Figure 1. Local Testbed Topology

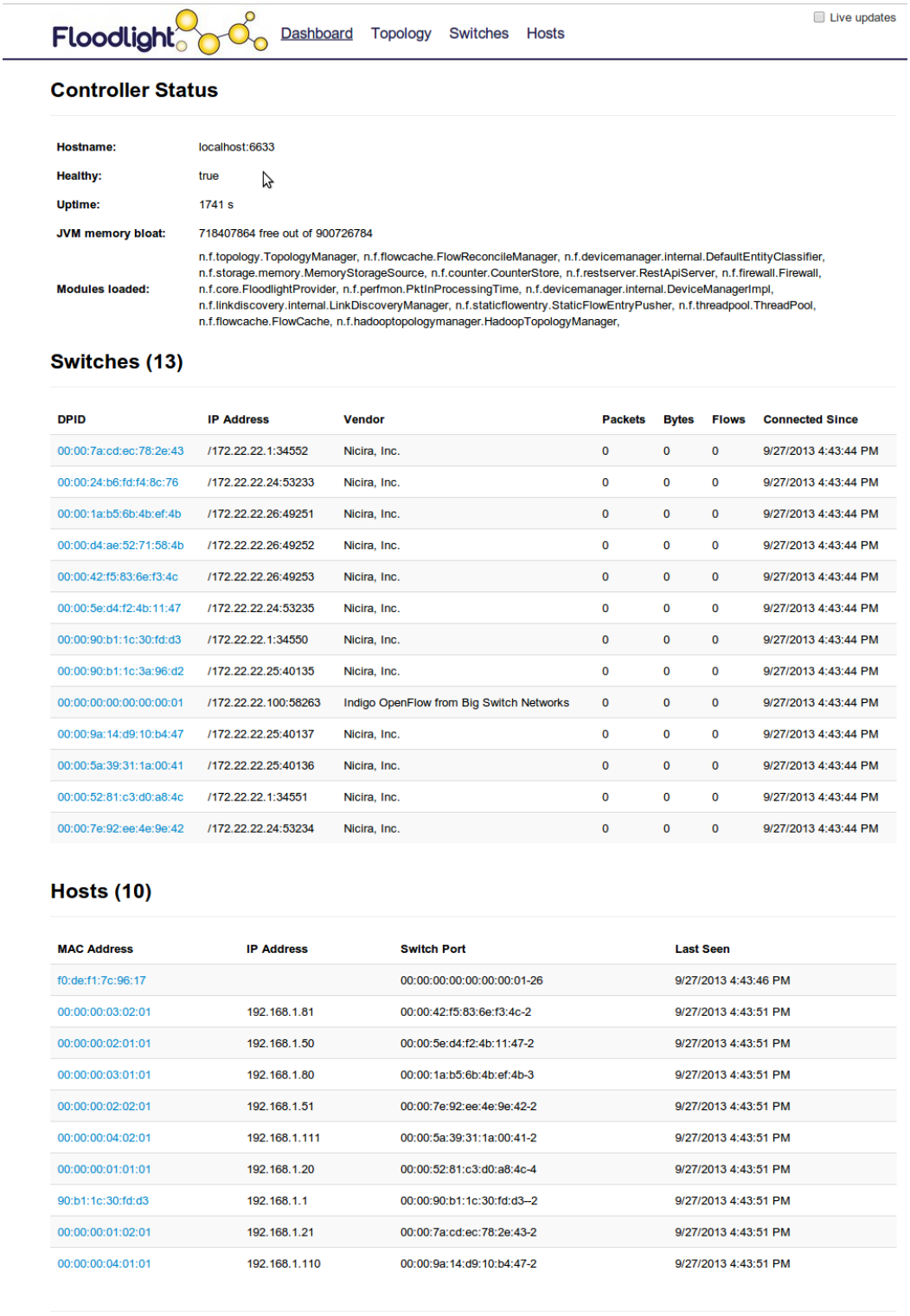

Figure 2. Device List

The nodes run in Ubuntu Server 13.04, with 3.2 Linux kernels. All codes are compiled using Sun JDK 1.6 and Apache Ant 1.8.

2.1.2 Test Results

We have tested both the SDN based topology management and the Hadoop integration. After setting up the testbed and configuring the network, we test the network Elf network topology management service by querying host locations via RESTful API that the Elf provides:

```
s3lab@headnode:~/elf$ curl -s http://10.227.119.160:8080/wm/hadooptopology/?host=192.168.1.50
/00:00:00:00:00:00:01/dc2/00:00:5e:d4:f2:4b:11:47
```

Figure 3. Elf Service Query Result

We can see that given a host IP address, the Elf service returns its network location in the multi-datacenter topology.

Next we test the multi-datacenter aware optimization of Hadoop based on the network topology management service that Elf provides. First, we show that our datacenter-aware data placement works well. Our placement strategy now is to place 3 replicas in two different datacenters: one datacenter has two replicas on two different racks and the other has one more replica.

```
/user/ubuntu/input/b 329123080 bytes, 5 block(s):  OK
0. blk_2292545193883778511_1001 len=67108864 repl=3 [/00:00:00:00:00:00:01/dc4/00:00:5a:39:31:1a:00:41/192.168.1.111:50
010, /00:00:00:00:00:00:01/dc1/00:00:52:81:c3:d0:a8:4c/192.168.1.20:50010, /00:00:00:00:00:00:01/dc4/00:00:9a:14:d9:10:
b4:47/192.168.1.110:50010]
1. blk_1941964895107307826_1001 len=67108864 repl=3 [/00:00:00:00:00:00:01/dc4/00:00:9a:14:d9:10:b4:47/192.168.1.110:50
010, /00:00:00:00:00:00:01/dc4/00:00:5a:39:31:1a:00:41/192.168.1.111:50010, /00:00:00:00:00:00:01/dc1/00:00:52:81:c3:d0
:a8:4c/192.168.1.20:50010]
2. blk_2005226386850894849_1001 len=67108864 repl=3 [/00:00:00:00:00:00:01/dc3/00:00:1a:b5:6b:4b:ef:4b/192.168.1.80:500
10, /00:00:00:00:00:00:01/dc3/00:00:42:f5:83:6e:f3:4c/192.168.1.81:50010, /00:00:00:00:00:00:01/dc1/00:00:52:81:c3:d0:a
8:4c/192.168.1.20:50010]
3. blk_5584102875604240394_1001 len=67108864 repl=3 [/00:00:00:00:00:00:01/dc2/00:00:7e:92:ee:4e:9e:42/192.168.1.51:500
10, /00:00:00:00:00:00:01/dc2/00:00:5e:d4:f2:4b:11:47/192.168.1.50:50010, /00:00:00:00:00:00:01/dc1/00:00:52:81:c3:d0:a
8:4c/192.168.1.20:50010]
4. blk_6095365225741781855_1001 len=60687624 repl=3 [/00:00:00:00:00:00:01/dc2/00:00:7e:92:ee:4e:9e:42/192.168.1.51:500
10, /00:00:00:00:00:00:01/dc1/00:00:52:81:c3:d0:a8:4c/192.168.1.20:50010, /00:00:00:00:00:00:01/dc2/00:00:5e:d4:f2:4b:1
1:47/192.168.1.50:50010]
```

Figure 4. Datacenter-aware Block Placement Strategy

Finally, we run a Hadoop job under the datacenter-aware prefetching and scheduling policy:

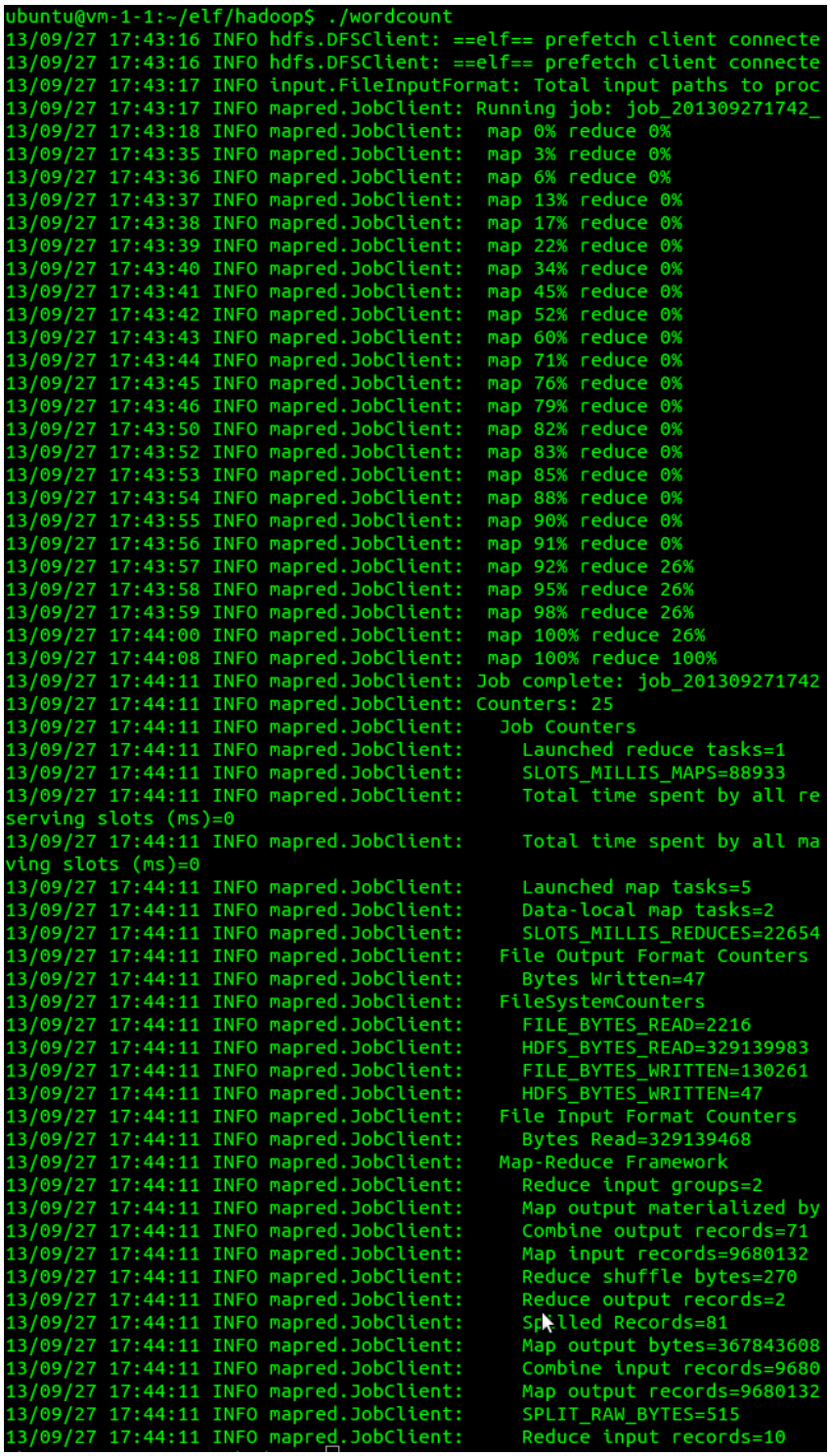

Figure 5. Hadoop Job Test

2.2 ExoGeni Testbed

When we started our development, we discussed the deployment problem with our Internet2 collaborators. As our application requires heavy use of VM resources, our Internet2 collaborators suggest us to use ExoGeni's resource to do the experiments across multiple sites.

After discussing with ExoGeni's manager, we found it impossible for us to use ExoGeni's hardware switch resources since they only support flat tag-based virtual Openflow-based network. This is not sufficient for us since we are going to build a hierarchical network topology.

Then we are suggested to use software switches instead of hardware switches to build the topology. This means that we run Open vSwtich instances inside VMs to forward the packets. Following this suggestion we have successfully built the topology we want, as shown in Figure 6. We sent test traffic through ping and everything worked fine.

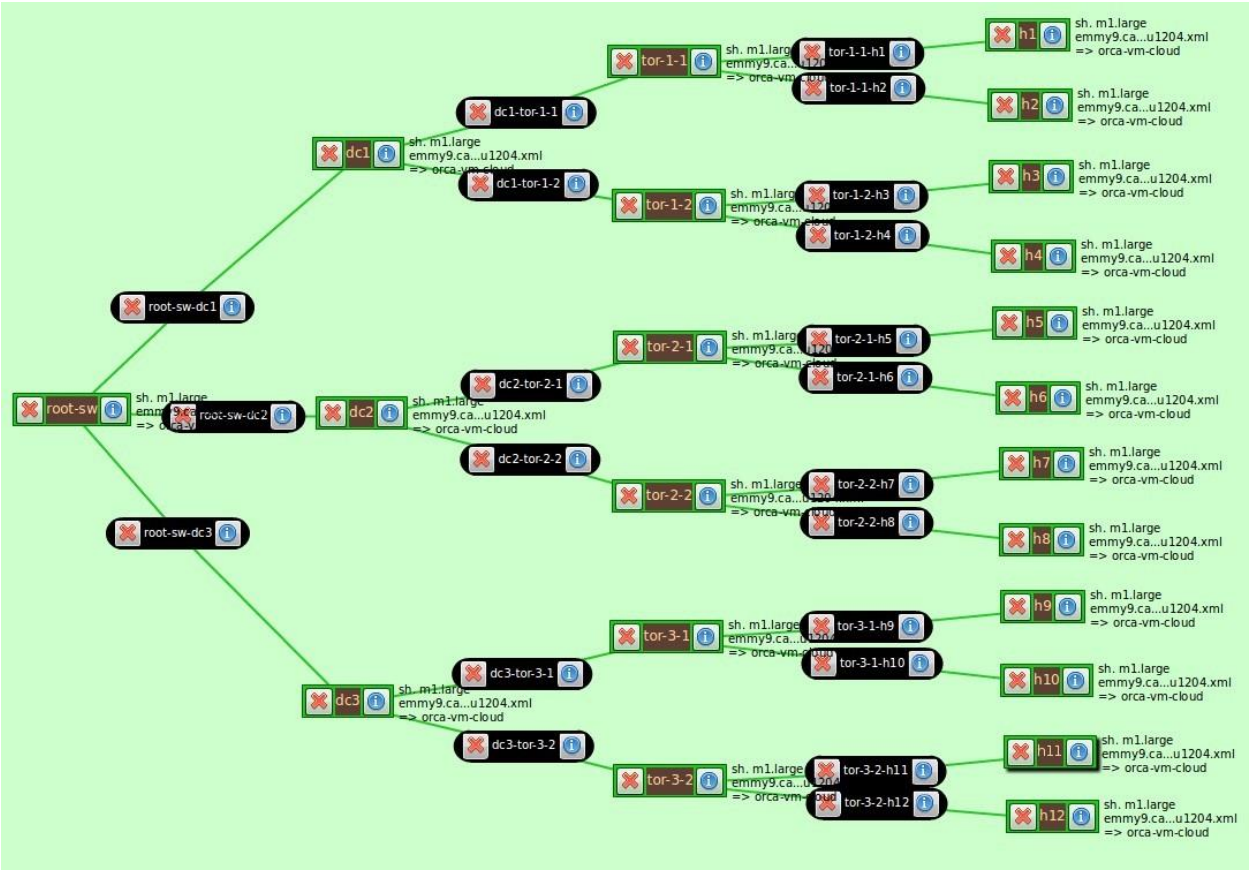

Figure 6. ExoGeni Testbed Topology

Unfortunately, when we were trying to connect these software switches to our controller, we found that the Floodlight controller cannot detect the links correctly. We believe it was not the problem of the Floodlight controller, but the testbed itself. We used wireshark and tcpdump toolkits to probe the problem. It seemed that our LLDP packet, which are used to construct the links by Floodlight, are absorbed by some unknown devices at somewhere inside the network. Since the underlying network is

hidden from us, we cannot further trace this problem. So we end up with using the local testbed to test our application and leave the wide-area development for the future work.

3. Other issues

Although we have passed all performed test suites, there are some potential improvements that could be done in the future. We have documented them in the code repository, under the /elf/doc directory.

We have revisited the original proposal because we could not get access to testbed resources that allow us to implement network bandwidth reservation. This issue has already been discussed during the follow-up meetings and the emails. Please see the design document for details.