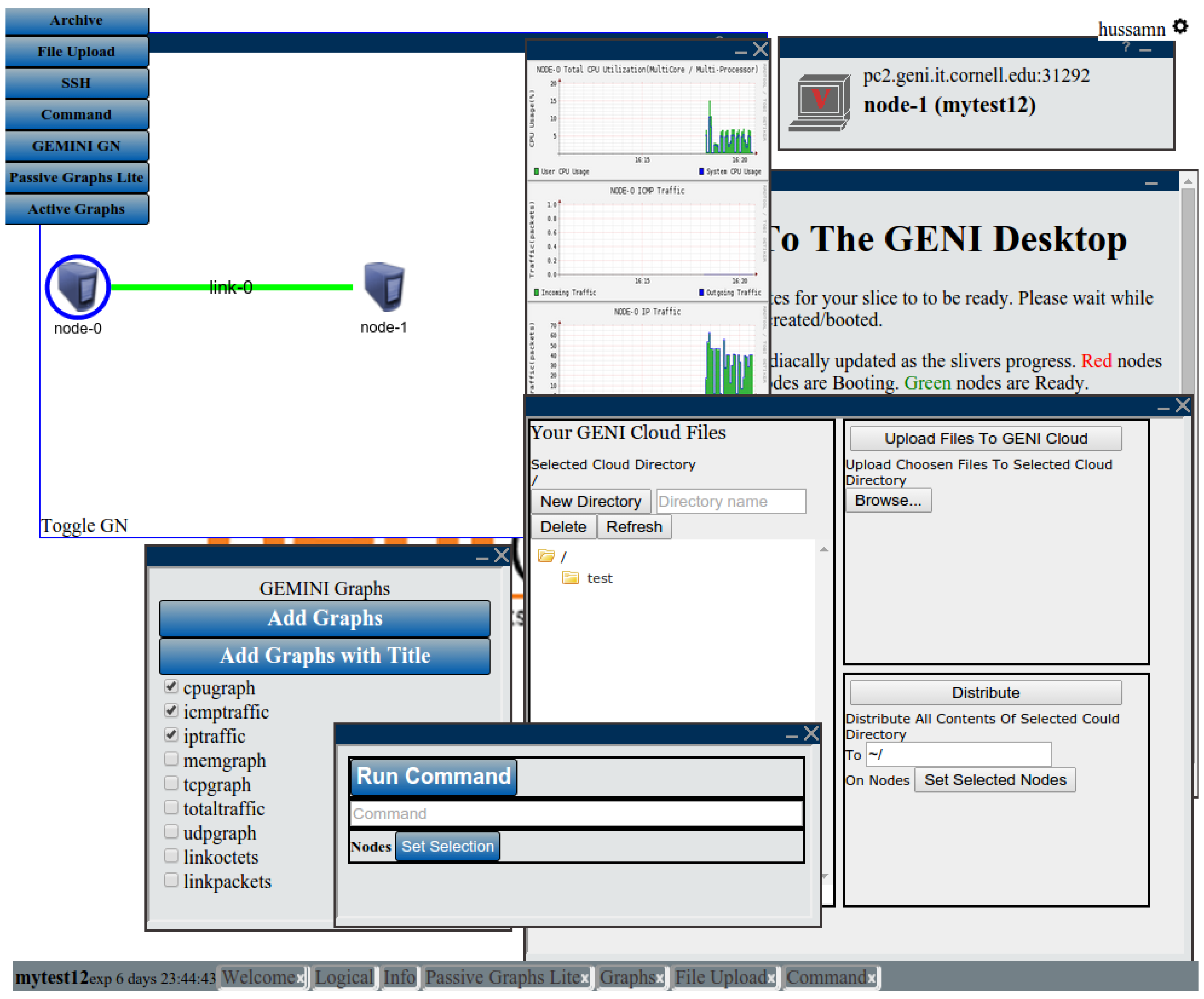
The GENI Desktop (GD) and Adopt-A-GENI (AAG)

A User Interface for Creating, Running, and Monitoring GENI Experiments

James Griffioen, Zongming Fei, Hussamuddin Nasir, Charles Carpenter, Xiongqi Wu, Jeremy Reed, Lowell Pike, Sergio Rivera P.
(*Laboratory for Advanced Networking, University of Kentucky*)

Overview

- ❑ The goal of the GENI Desktop is to make it easier for experimenters to create, control, interact with, and evaluate the performance of the resources that comprise their slice.
- ❑ The GENI Desktop provides an easy-to-use graphical user interface with windowing-system style features inside a web browser to create the look-and-feel of locally running tools (without having to actually install, manage, and run tools locally).
- ❑ Users interact with their resources using a single abstraction that involves (1) selecting resources, and (2) apply operations on those resources.



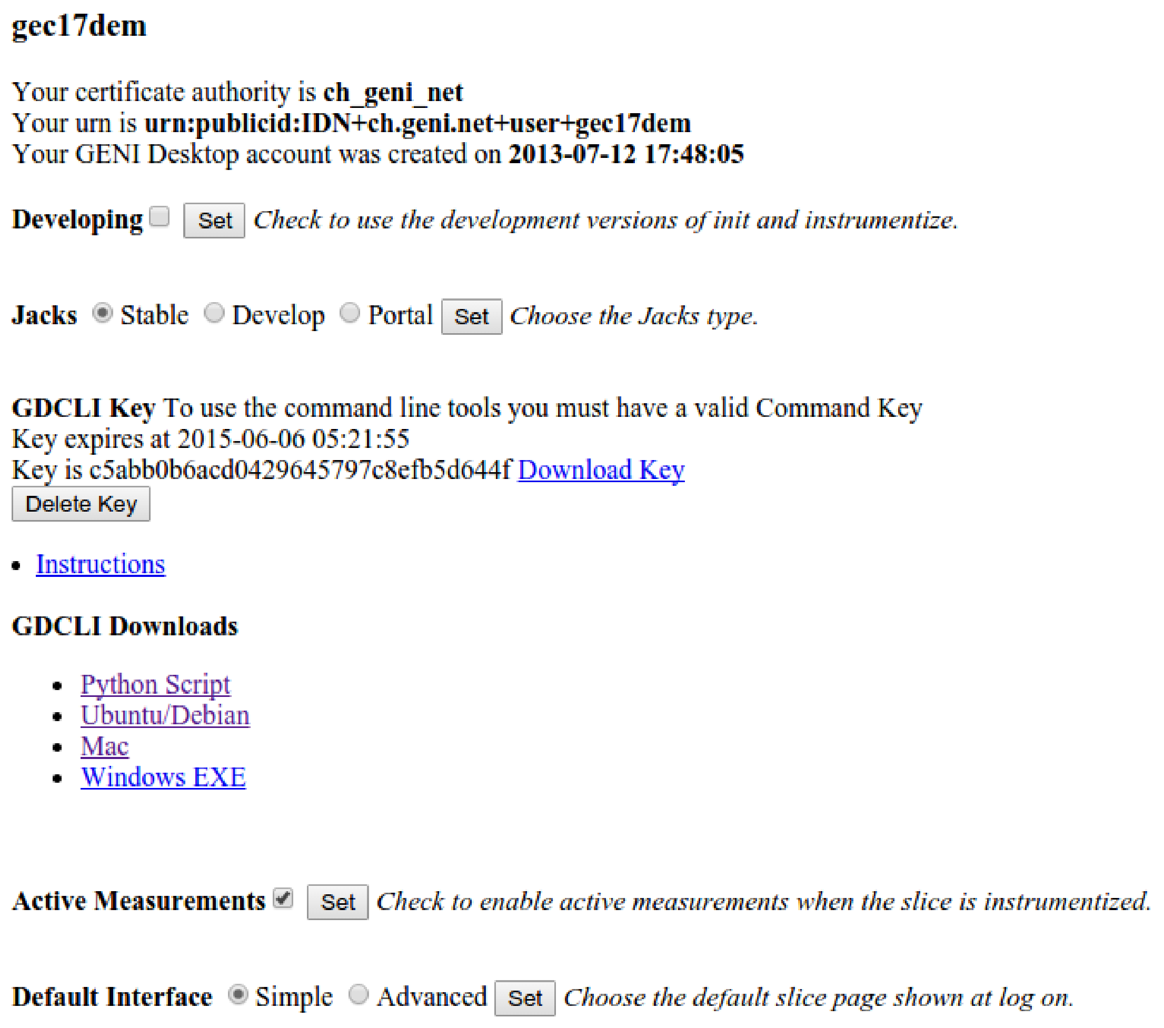
To The GENI Desktop

es for your slice to to be ready. Please wait while reated/booted.

diacally updated as the slivers progress. Red nodes des are Booting. Green nodes are Ready.

Command Line Interface (GDCLI) of the GENI Desktop

- ❑ The GENI Desktop provides a command line interface (GDCLI) for experienced users to use commands to manage their slices.
- ❑ The user can run the commands in Linux, MAC and Windows operating systems, after downloading a command key from the GENI Desktop.



gec17dem

Your certificate authority is **ch_geni_net**
Your urn is **urn:publicid:IDN+ch.geni.net+user+gec17dem**
Your GENI Desktop account was created on **2013-07-12 17:48:05**

Developing ☐ Set *Check to use the development versions of init and instrumentize.*

Jacks ● Stable ○ Develop ○ Portal Set *Choose the Jacks type.*

GDCLI Key To use the command line tools you must have a valid Command Key
Key expires at 2015-06-06 05:21:55
Key is c5abb0b6acd0429645797c8efb5d644f Download Key
Delete Key

- Instructions

GDCLI Downloads

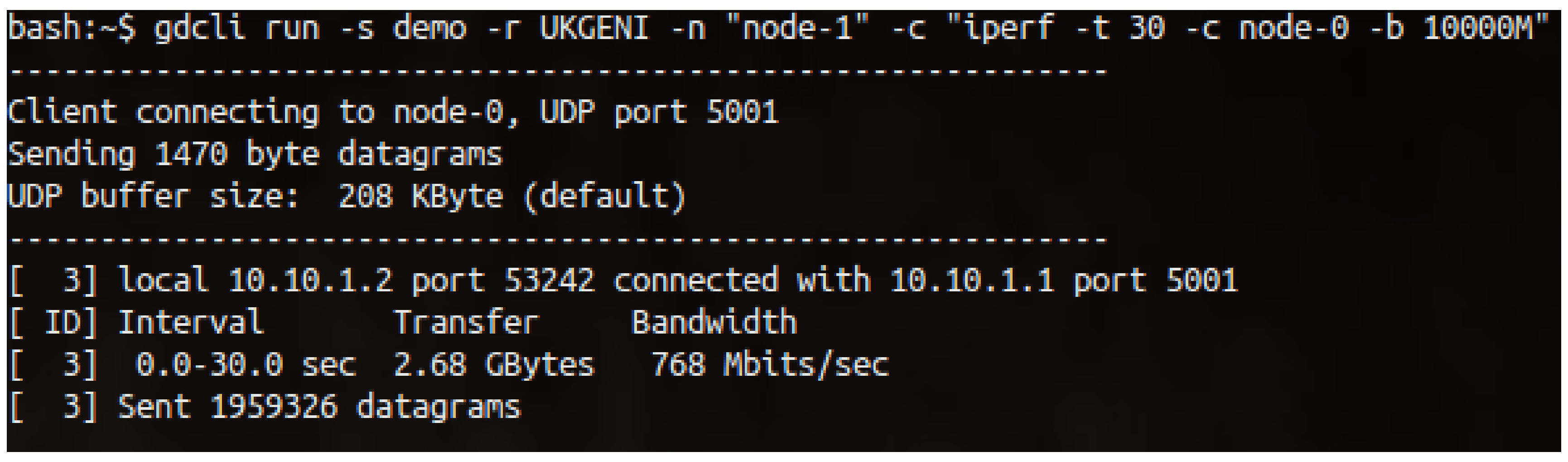
- Python Script
- Ubuntu/Debian
- Mac
- Windows EXE

Active Measurements ☑ Set *Check to enable active measurements when the slice is instrumentized.*

Default Interface ● Simple ○ Advanced Set *Choose the default slice page shown at log on.*

Use GDCLI to Manage Experiments

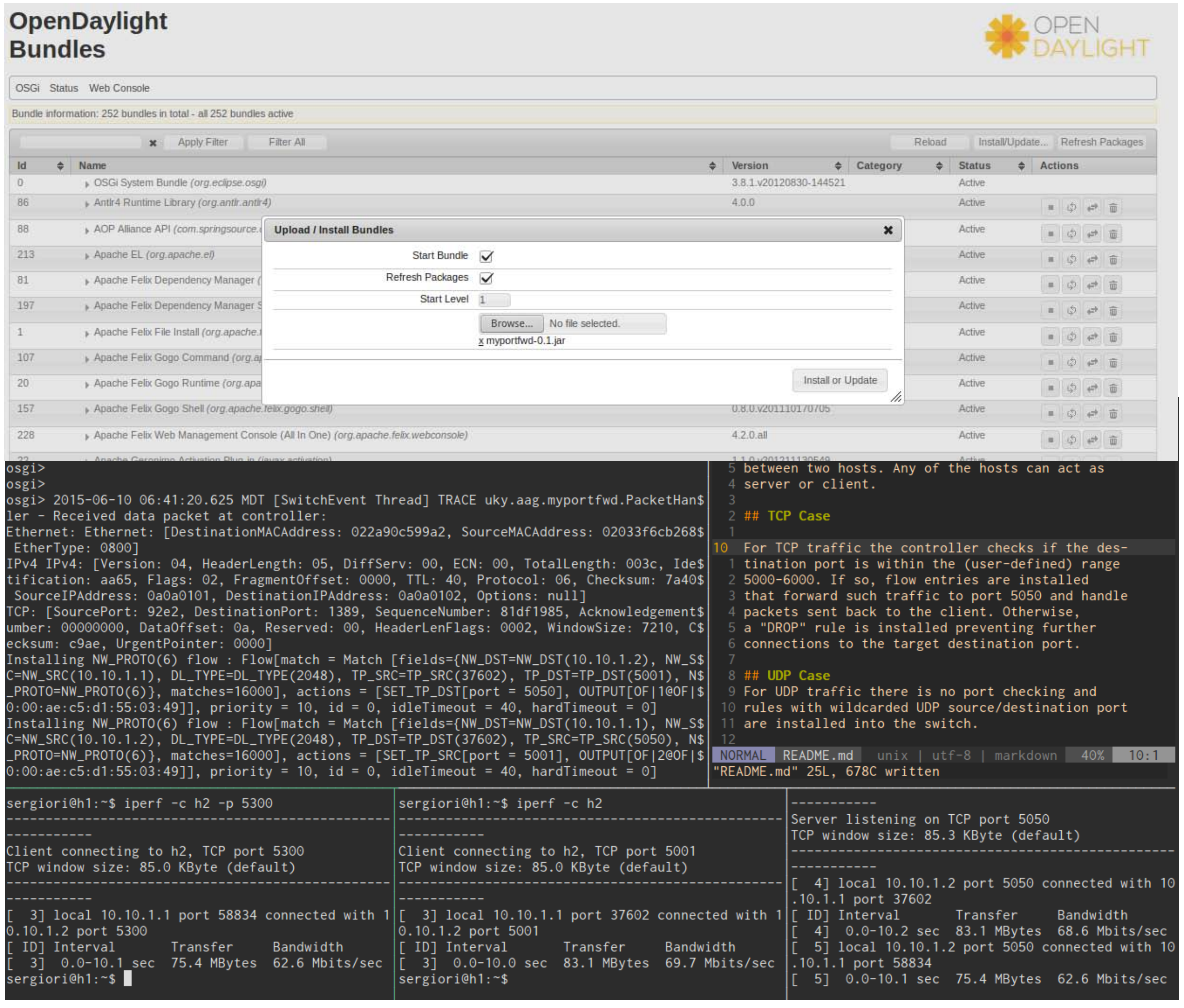
- ❑ The GDCLI provides most functions implemented in the GUI of the GENI Desktop, including
  - ❑ Upload files to a selected set of nodes
  - ❑ Run a command on a selected set of nodes
  - ❑ Download a traffic measurement graph (as PNG or CSV) from a selected set of nodes
  - ❑ Download a normal file from a selected set of nodes
  - ❑ Get a list of slices
  - ❑ Check the status of a slice
  - ❑ Get the topology of a slice
  - ❑ Validate the setup of a slice
  - ❑ List the nodes in a slice
  - ❑ List the links in a slice
- ❑ The user can write a script using these commands to run the whole experiment process. The following figure shows an example of the GDCLI run command.



```
bash:~$ gdcli run -s demo -r UKGENI -n "node-1" -c "iperf -t 30 -c node-0 -b 10000M"
------------------------------------------------------------
Client connecting to node-0, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.10.1.2 port 53242 connected with 10.10.1.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-30.0 sec  2.68 GBytes   768 Mbits/sec
[  3] Sent 1959326 datagrams
```

User-defined AAG Subcontroller

- ❑ The OpenDaylight-based AAG controller allows users to write a bundle as an AAG-sub-controller to control the traffic within an SDN-controlled AAG slice.
- ❑ The following figures shows a user-defined port-forwarding subcontroller to allow traffic to ports 5300 and 5001 to be forwarded to port 5050.



Bundle information: 252 bundles in total - all 252 bundles active

Id	Name	Version	Category	Status	Actions
0	OSGi System Bundle *(org.eclipse.osgi)*	3.8.1.v20120830-144521		Active	
86	Antlr4 Runtime Library *(org.antlr.antlr4)*	4.0.0		Active	
88	AOP Alliance API *(com.springsource.*			Active	
213	Apache EL *(org.apache.el)*			Active	
81	Apache Felix Dependency Manager (			Active	
197	Apache Felix Dependency Manager S			Active	
1	Apache Felix File Install *(org.apache.*			Active	
107	Apache Felix Gogo Command *(org.a*			Active	
20	Apache Felix Gogo Runtime *(org.apa*			Active	
157	Apache Felix Gogo Shell *(org.apache.felix.gogo.shell)*	0.8.0.v201110170705		Active	
228	Apache Felix Web Management Console (All In One) *(org.apache.felix.webconsole)*	4.2.0.all		Active	

```
osgi>
osgi>
osgi> 2015-06-10 06:41:20.625 MDT [SwitchEvent Thread] TRACE uky.aag.myportfwd.PacketHan$
ler - Received data packet at controller:
Ethernet: Ethernet: [DestinationMACAddress: 022a90c599a2, SourceMACAddress: 02033f6cb268$
 EtherType: 0800]
IPv4 IPv4: [Version: 04, HeaderLength: 05, DiffServ: 00, ECN: 00, TotalLength: 003c, Ide$
tification: aa65, Flags: 02, FragmentOffset: 0000, TTL: 40, Protocol: 06, Checksum: 7a40$
 SourceIPAddress: 0a0a0101, DestinationIPAddress: 0a0a0102, Options: null]
TCP: [SourcePort: 92e2, DestinationPort: 1389, SequenceNumber: 81df1985, Acknowledgement$
umber: 00000000, DataOffset: 0a, Reserved: 00, HeaderLenFlags: 0002, WindowSize: 7210, C$
ecksum: c9ae, UrgentPointer: 0000]
Installing NW_PROTO(6) flow : Flow[match = Match [fields={NW_DST=NW_DST(10.10.1.2), NW_S$
C=NW_SRC(10.10.1.1), DL_TYPE=DL_TYPE(2048), TP_SRC=TP_SRC(37602), TP_DST=TP_DST(5001), N$
_PROTO=NW_PROTO(6)}, matches=16000], actions = [SET_TP_DST[port = 5050], OUTPUT[OF|1@OF|$
0:00:ae:c5:d1:55:03:49]], priority = 10, id = 0, idleTimeout = 40, hardTimeout = 0]
Installing NW_PROTO(6) flow : Flow[match = Match [fields={NW_DST=NW_DST(10.10.1.1), NW_S$
C=NW_SRC(10.10.1.2), DL_TYPE=DL_TYPE(2048), TP_DST=TP_DST(37602), TP_SRC=TP_SRC(5050), N$
_PROTO=NW_PROTO(6)}, matches=16000], actions = [SET_TP_SRC[port = 5001], OUTPUT[OF|2@OF|$
0:00:ae:c5:d1:55:03:49]], priority = 10, id = 0, idleTimeout = 40, hardTimeout = 0]
```

```
between two hosts. Any of the hosts can act as
server or client.

## TCP Case

For TCP traffic the controller checks if the des-
tination port is within the (user-defined) range
5000-6000. If so, flow entries are installed
that forward such traffic to port 5050 and handle
packets sent back to the client. Otherwise,
a "DROP" rule is installed preventing further
connections to the target destination port.

## UDP Case
For UDP traffic there is no port checking and
rules with wildcarded UDP source/destination port
are installed into the switch.

NORMAL  README.md   unix | utf-8 | markdown   40%   10:1
"README.md" 25L, 678C written
```

```
sergiori@h1:~$ iperf -c h2 -p 5300
------------------------------------------------------------
Client connecting to h2, TCP port 5300
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 10.10.1.1 port 58834 connected with 10.10.1.2 port 5300
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.1 sec  75.4 MBytes  62.6 Mbits/sec
sergiori@h1:~$
```

```
sergiori@h1:~$ iperf -c h2
------------------------------------------------------------
Client connecting to h2, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 10.10.1.1 port 37602 connected with 10.10.1.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  83.1 MBytes  69.7 Mbits/sec
sergiori@h1:~$
```

```
------------------------------------------------------------
Server listening on TCP port 5050
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 10.10.1.2 port 5050 connected with 10.10.1.1 port 37602
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.2 sec  83.1 MBytes  68.6 Mbits/sec
[  5] local 10.10.1.2 port 5050 connected with 10.10.1.1 port 58834
[  5]  0.0-10.1 sec  75.4 MBytes  62.6 Mbits/sec
```

Supported by the National Science Foundation

GEC 23 – UIUC – June 2015