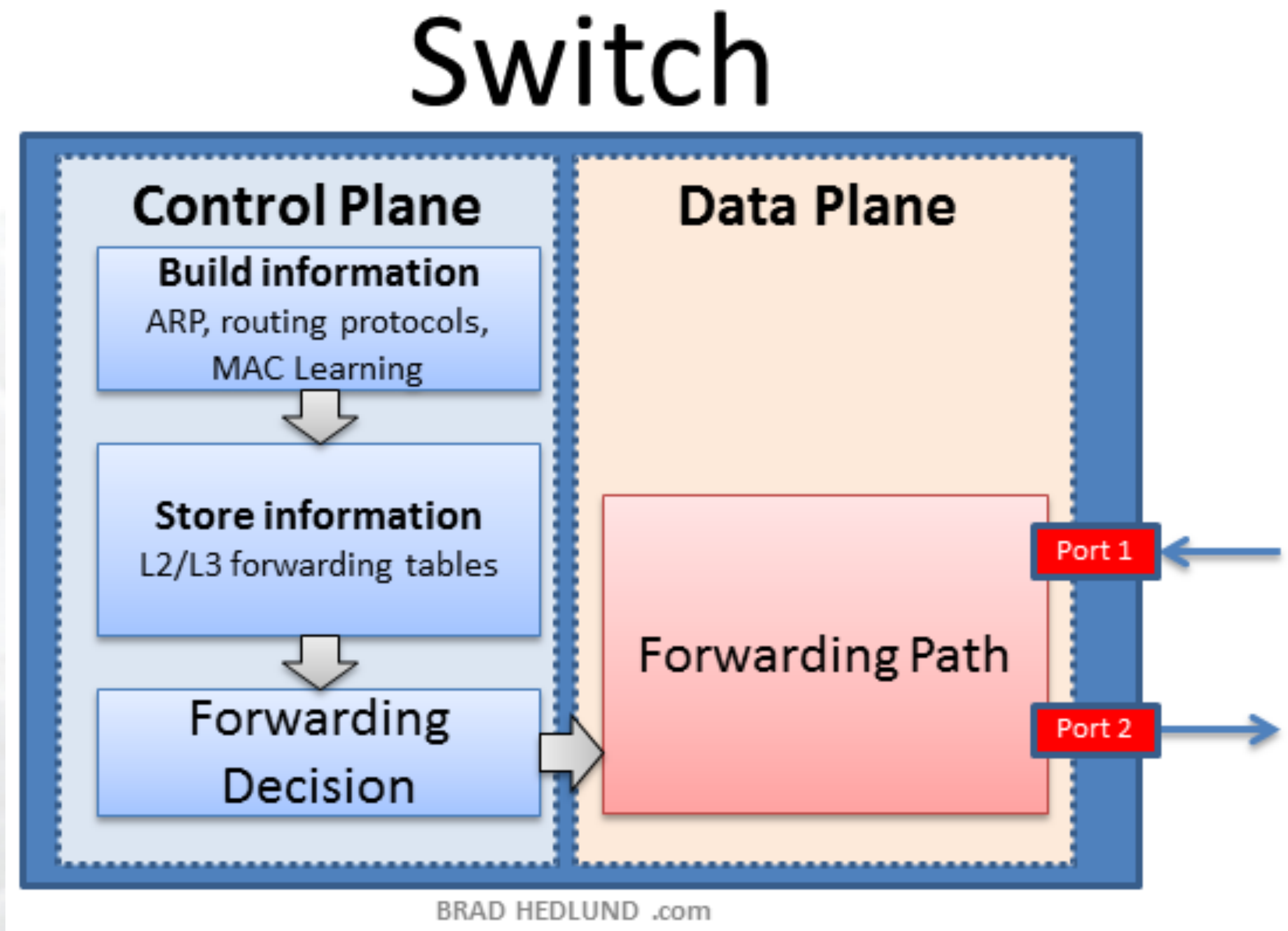
# Tutorial: OpenFlow and GENI

GENI Engineering Conference 18

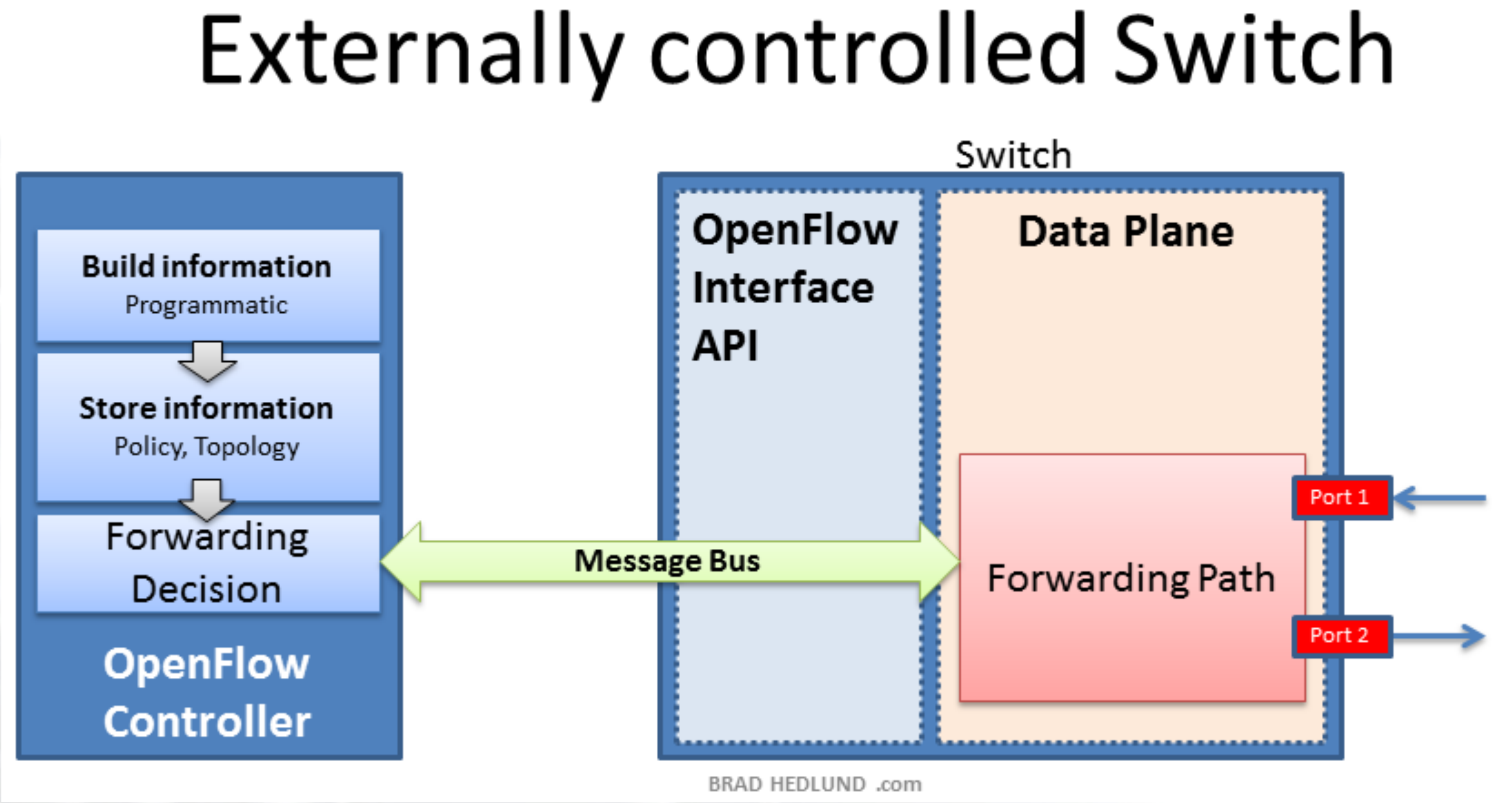
October 2013


Sponsored by the National Science Foundation

# Switch Architecture



# Moving Control out of the Switch

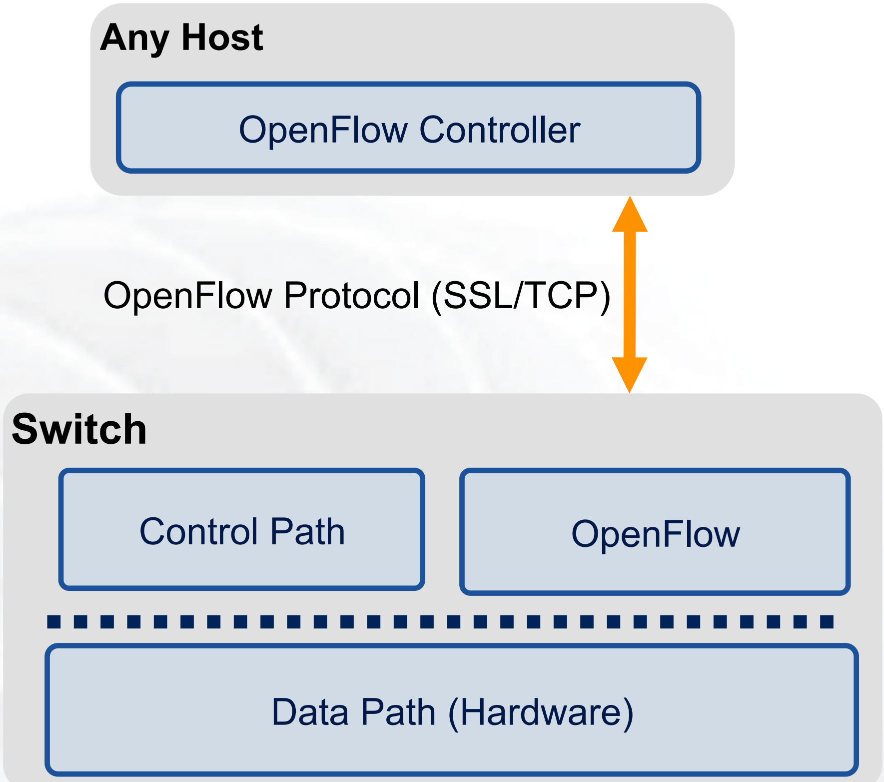


# OpenFlow is an API

- Control how packets are forwarded
- Implementable on COTS hardware
- Make deployed networks programmable
  - not just configurable
- Makes innovation easier

Modified slide from : http://www.deutsche-telekom-laboratories.de/~robert/GENI-Experimenters-Workshop.ppt

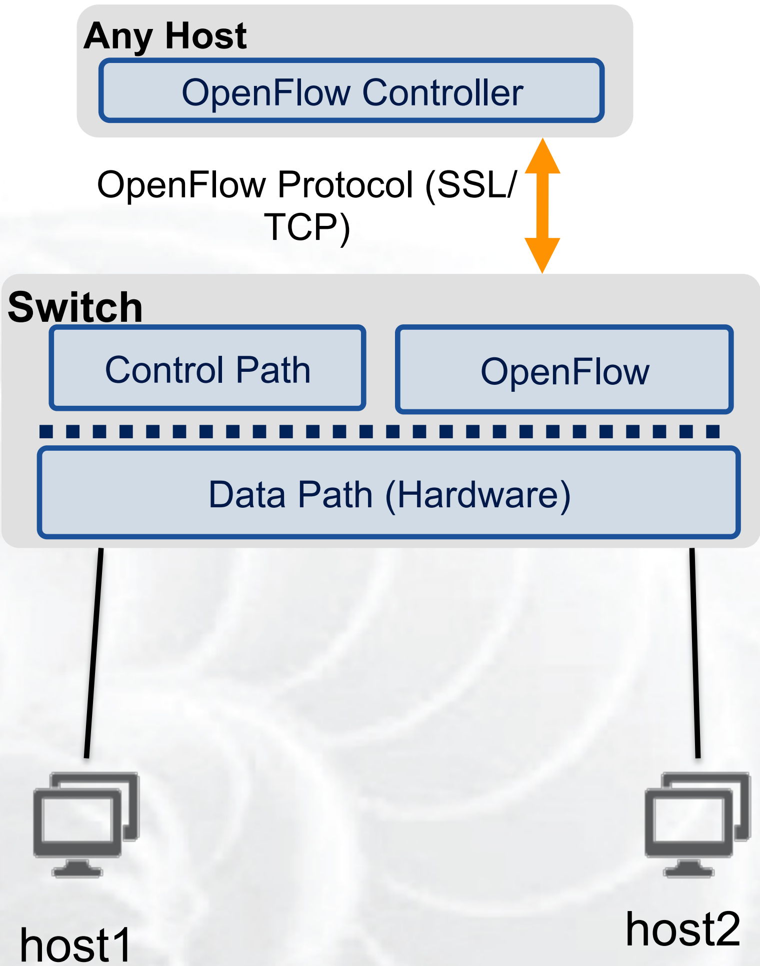
# OpenFlow




- The controller is responsible for populating forwarding table of the switch
- In a table miss the switch asks the controller

Modified slide from : http://www.deutsche-telekom-laboratories.de/~robert/GENI-Experimenters-Workshop.ppt

# OpenFlow in action




- Host1 sends a packet
- If there are no rules about handling this packet
  - Forward packet to the controller
  - Controller installs a flow
- Subsequent packets do not go through the controller

Modified slide from : http://www.deutsche-telekom-laboratories.de/~robert/GENI-Experimenters-Workshop.ppt

# OpenFlow Basics
## Flow Table Entries

| Rule | Action | Stats |
|---|---|---|

Packet + byte counters

1. Forward packet to port(s)
2. Encapsulate and forward to controller
3. Drop packet
4. Send to normal processing pipeline
5. Modify Fields

| Switch Port | VLAN ID | VLAN PCP | MAC src | MAC dst | Eth type | IP Src | IP Dst | IP Prot | IP ToS | TCP sport | TCP dport |
|---|---|---|---|---|---|---|---|---|---|---|---|

+ mask what fields to match

slide from : http://www.deutsche-telekom-laboratories.de/~robert/GENI-Experimenters-Workshop.ppt

# Use Flow Mods

- Going through the controller on every packet is inefficient
- Installing Flows either proactively or reactively is the right thing to do:
- A Flow Mod consists off :
  - A match on any of the 12 supported fields
  - A rule about what to do matched packets
  - Timeouts about the rules:
    - Hard timeouts
    - Idle timeouts
  - The packet id in reactive controllers

# OpenFlow common Pitfalls

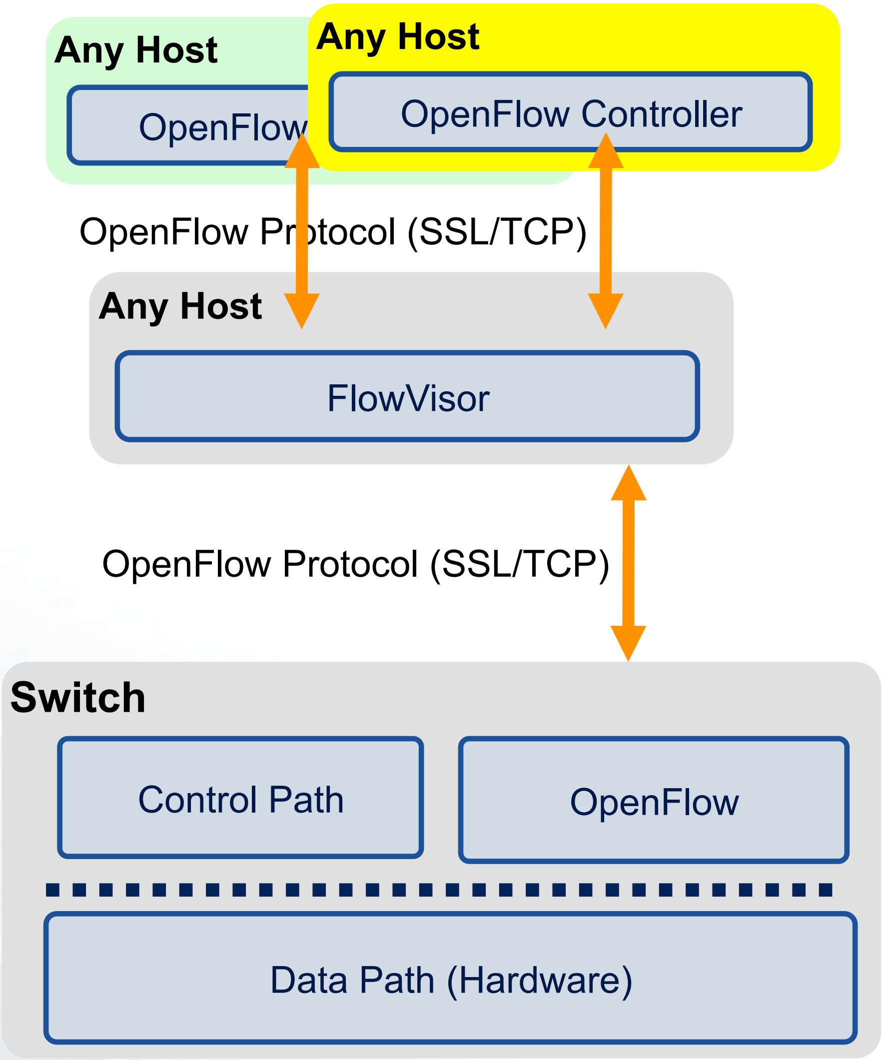
- Controller is responsible for all traffic, not just your application!
  - ARPs
  - DHCP
  - LLDP
- Reactive controllers
  - UDP
- Performance in hardware switches
  - Not all actions are supported in hardware
- No STP
  - Broadcast storms

# FlowVisor

- Only one controller per switch
- FlowVisor is a proxy controller that can support multiple controllers

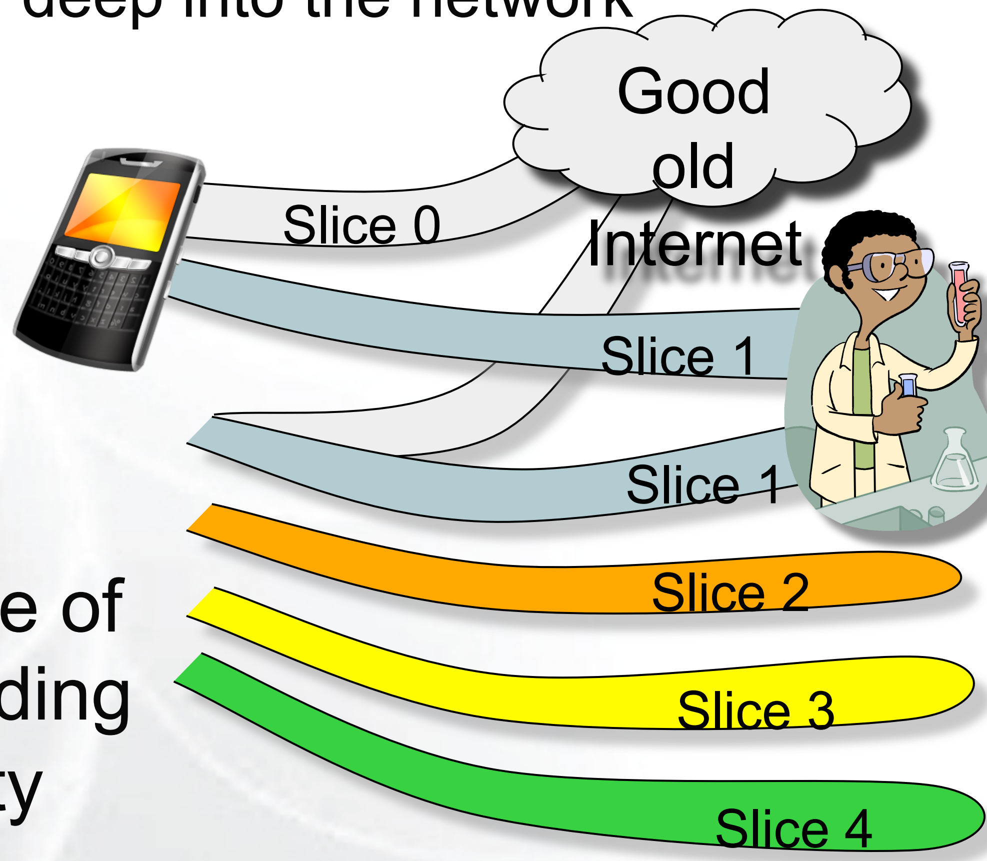
FlowSpace describes packet flows :

  - Layer 1: Incoming port on switch
  - Layer 2: Ethernet src/dst addr, type, vlanid, vlanpcp
  - Layer 3: IP src/dst addr, protocol, ToS
  - Layer 4: TCP/UDP src/dst port



# GENI Programmable Network

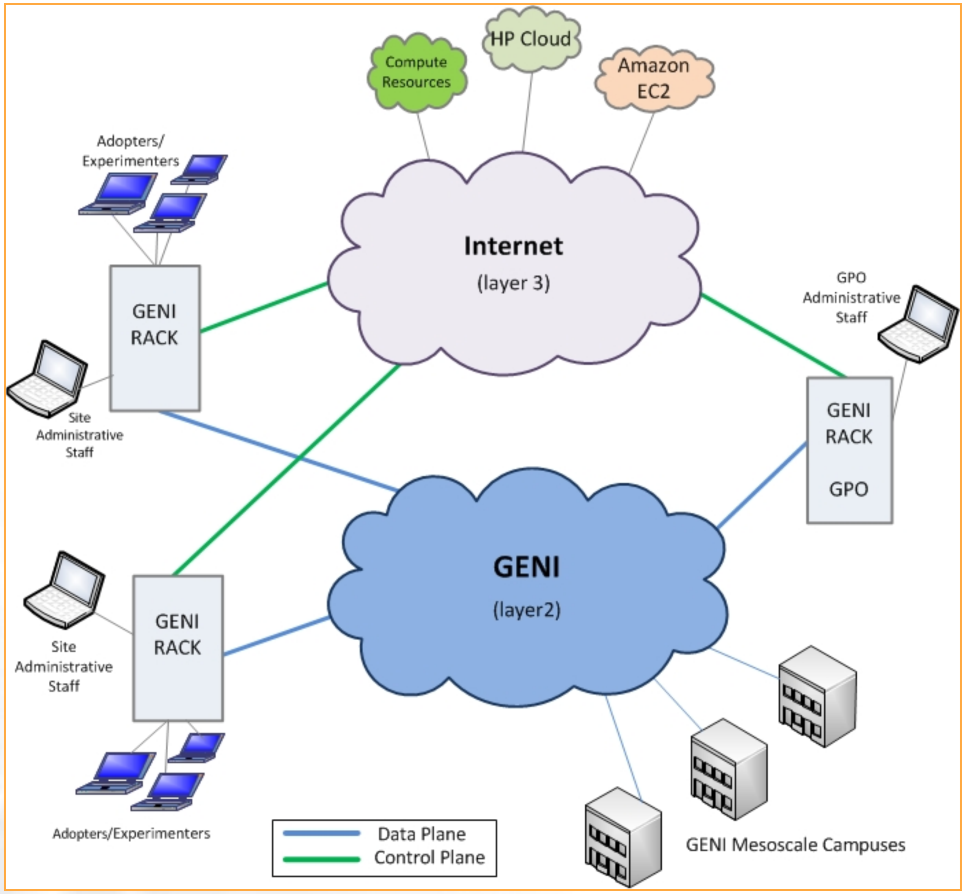
- Key GENI concept: slices & deep programmability
  - Internet: open innovation in application programs
  - GENI: open innovation deep into the network



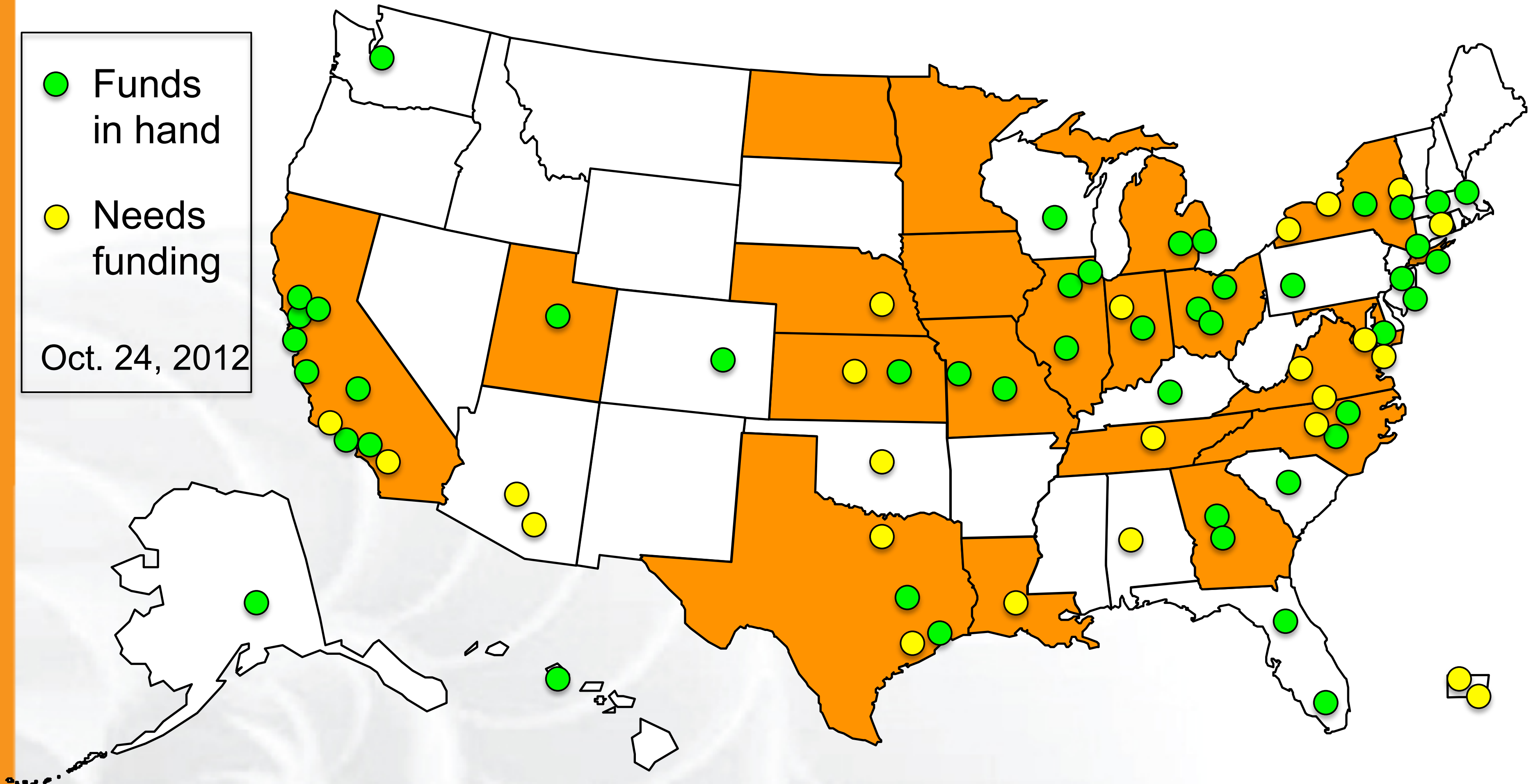

OpenFlow switches one of the ways GENI is providing deep programmability

# Racks and Campuses

- GENI Rack projects are expanding available GENI infrastructure in the US.
- Racks provide reservable, sliceable compute and network resources using Aggregate Managers.
- GENI AM API compliance

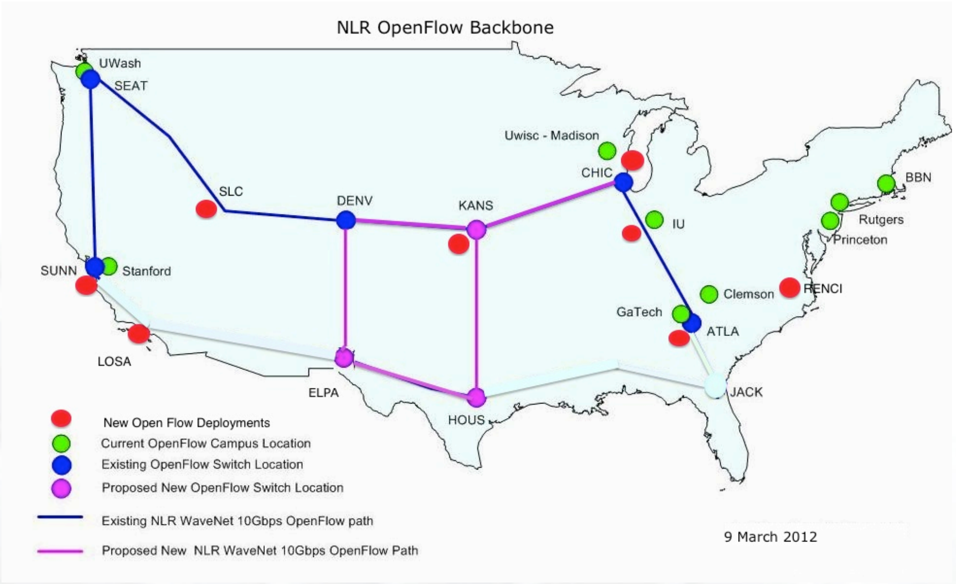


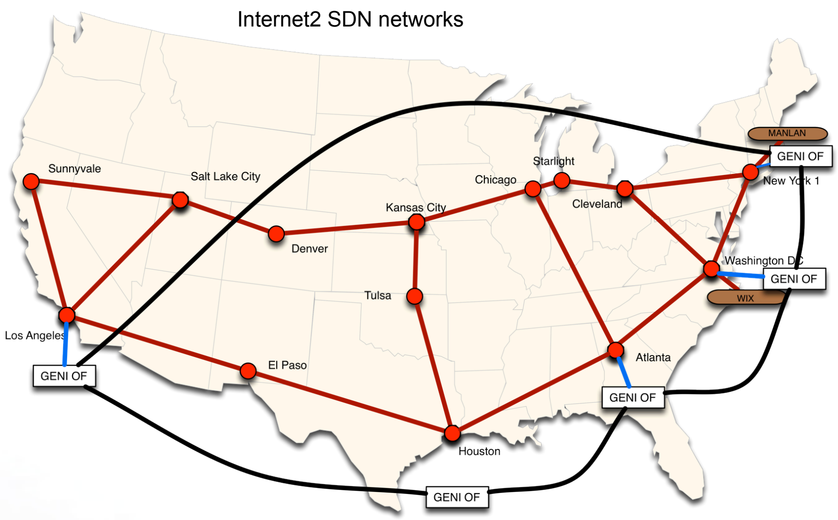
# GENI Rack Campuses




- 43 racks planned this year
- Each rack has an **OpenFlow-enabled** switch

# Core Networks




- NLR committed to 2013 meso-scale expansion following reorganization
- Internet2 adding 10GbE paths to Advanced Layer 2 Services (AL2S) at 4 of 5 OpenFlow meso-scale/ProtoGENI Pops
- GENI Aggregate Manager in Internet2 AL2S and dynamic stitching with GENI coming in Spiral 5

# FOAM

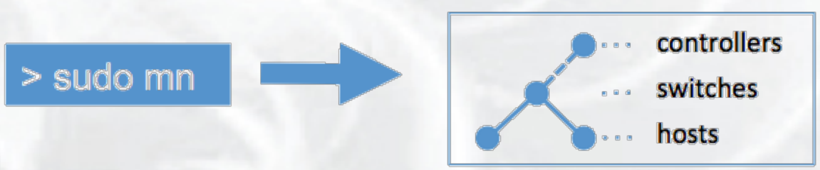
- An OpenFlow Aggregate Manager
- It's a GENI compliant reservation service
  - Helps experimenters reserve flowspace in the FlowVisor
- Speaks AM API v1 and AM API v2
- RSpecs GENI v3, OpenFlow v3 extension

# OpenFlow Experiments

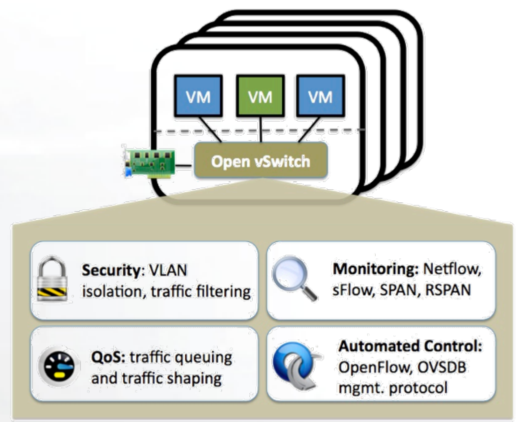
Debugging OpenFlow experiments is hard:

– Network configuration debugging requires coordination
– Many networking elements in play
– No console access to the switch

Before deploying your OpenFlow experiment test your controller.




http://mininet.github.com/

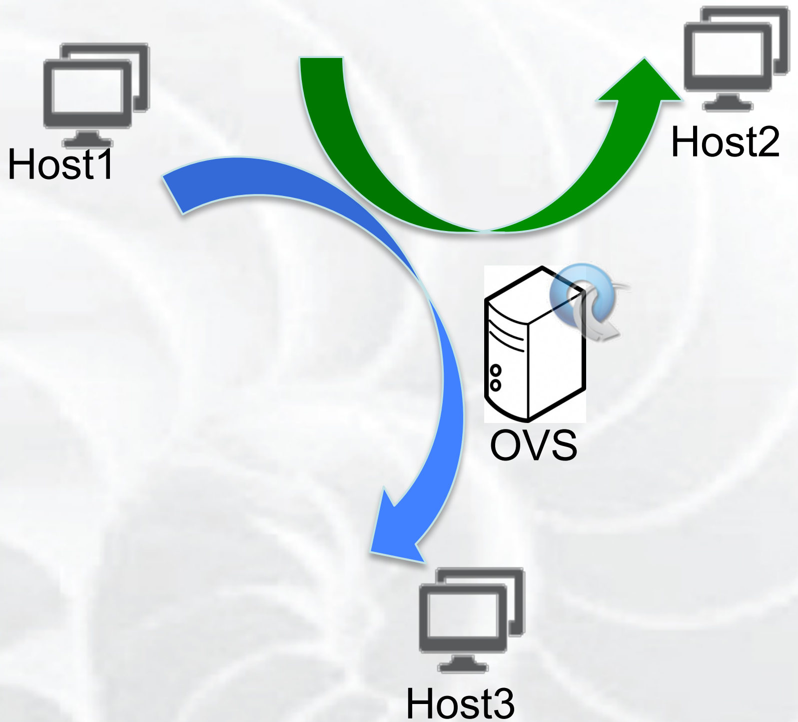



http://openvswitch.org/

# Run an OpenFlow experiment

1 Xen VM as OVS switch

3 OpenVZ VMs connected to OVS




- Setup OVS
- Write simple controllers
  - e.g. divert traffic to a different server
  - Use Python controller PoX

# To Save Time…

- Slices have been created for you:
  - Slice name: oftutnn
- Resources have been added to your slice:
  - 1 Xen VM running OVS
  - 3 OpenVZ VMs that act as traffic sources & sinks
  - Resources are from various InstaGENI racks
- Get shared SSH key installed on the resources:

  $ wget http://www.gpolab.bbn.com/experiment-support/gec18.key

  $ mv gec18.key ~/.ssh/gec18.key

  $ chmod 0600 ~/.ssh/gec18.key
- Add the key to your ssh-agent:

  $ ssh-add ~/.ssh/gec18.key (password: gec!8)
- Example login: (DON"T DO THIS NOW)

  $ ssh lnevers@pcxx.instageni.northwestern.edu -p 34106
- We will add your account to the slice!


- Part I: Design/Setup
  - Obtain Resources
    - What is OpenFlow, what can I do with Openflow?
    - Demo: Using OpenFlow in GENI
- **Part II: Execute**
  - **Configure and Initialize Services**
  - Execute Experiment
- Part III: Finish
  - Teardown Experiment

# Configure OVS

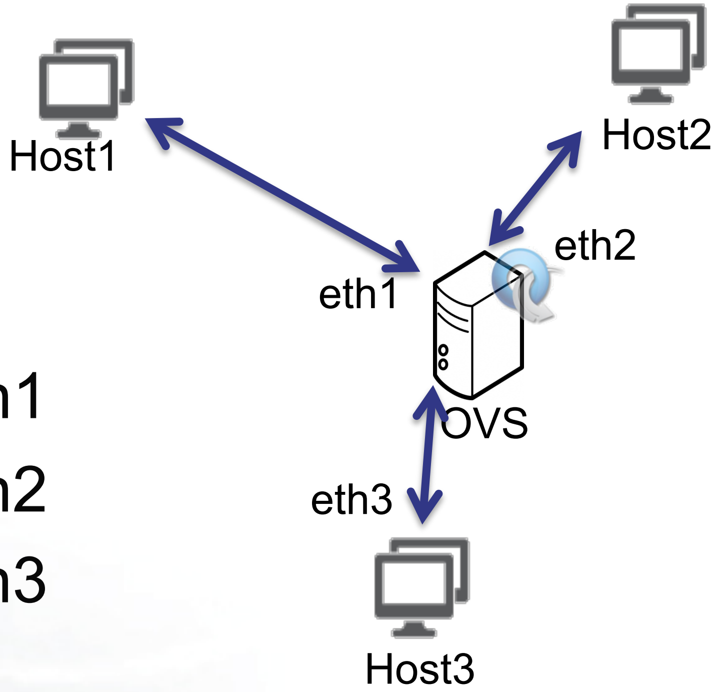
OVS is a virtual switch running on a Xen VM node.

- The interfaces of the Xen node are the ports of the switch
  - Configure an ethernet bridge
  - add all dataplane ports to the switch
- Can be an OpenFlow switch
  - Point OVS switch to the controller address and port (for convenience on the same host but it can be anywhere)
- Userspace OVS for this exercise

# Configure and Initialize OVS

- Log in to OVS host and configure software switch:

```
$ ifconfig
$ sudo ifconfig eth1 0
$ sudo ifconfig eth2 0
$ sudo ifconfig eth3 0
$ sudo ovs-vsctl add-port br0 eth1
$ sudo ovs-vsctl add-port br0 eth2
$ sudo ovs-vsctl add-port br0 eth3
$ sudo ovs-vsctl list-ports br0
$ sudo ovs-vsctl set-controller br0 tcp:127.0.0.1:6633
$ sudo ovs-vsctl set-fail-mode br0 secure
$ sudo ovs-vsctl show
```




- Part I: Design/Setup
  - Obtain Resources
  - What is OpenFlow, what can I do with Openflow?
  - Demo: Using OpenFlow in GENI
- **Part II: Execute**
  - Configure and Initialize Services
  - **Execute Experiment**
- Part III: Finish
  - Teardown Experiment

# Experiments (1/4)

1. Use a Learning Switch Controller:
    1. See the traffic flow changes between hosts as the controller is started or stopped.
    2. Soft versus hard timeouts for traffic flows.

# Experiments (1/4)

- Login host1 and start ping host2

  $ ping 10.

- Start learning switch controller:

  $ cd /local/pox

  $ ./pox.py --verbose forwarding.l2_learning

- Look at ping… *now works.*
- Kill controller (ctl c)
- Look at ping… *still running,*

# Experiments (2/4)

2. Write and run a Traffic Duplication Controller:
    1. Controller will duplicate traffic to a different port on the OVS switch.
    2. Use tcpdump to see the packet duplication.

# Experiments (2/4)

- Open 2 windows on OVS host
- Start tcpdump for on *OVS:if0* and *OVS:if1*
- Run duplication controller on *OVS:if1*

  $ cd /local/pox

  $ ./pox.py --verbose myDuplicateTraffic --duplicate_port=<data_interface_name>
- Look at ping from host1 to host2.
- Kill controller (ctl c)

# Experiments (3/4)

3. Write and run a port forwarding controller:

    1. Controller will do port forwarding on your OVS Switch to port specified.

    2. Use two netcat servers on host2 to see traffic delivery.

# Experiments (3/4)

- On host 3:

  $ nc –l 7000

- Run proxy controller:

  $ cd /local/pox

  $ ./pox.py --verbose myProxy

- On host1:

  $ nc 10.10.1.2 5000

- Look at host3 windows, should now be getting nc traffic.

# Experiments (4/4)

4. Write and run a server proxy controller
    1. To redirect packets to a proxy:
        - What fields do you need to overwrite?
        - Which packets needs special handling?
    2. Use netcat to see the deflection

# Experiments (4/4)

- Two windows on host2 run the following:

  $ nc -l 5000

  $ nc –l 6000

- Start learning switch controller:
- On host1:

  $ nc 10.10.1.2 5000

- See what happens to traffic
- Kill controller (ctl c)
- Retry with port forwarding controller and see what happens to traffic, and kill when done.

# Part III: Finish Experiment


When your experiment is done, you should always release your resources.

- Normally this is when you would archive your data
- Delete your slivers at **each** aggregate