Enterprise GENI

Guido Appenzeller, Nick McKeown,
Rob Sherwood, Guru Parulkar
Stanford University

GENI DESING PROCESS

Build from two sides: The Transcontinental Railroad Model

Network Substrate

Bottom Up:

- Experimental Network Infrastructure
- Local Infrastructure Management

Control Framework

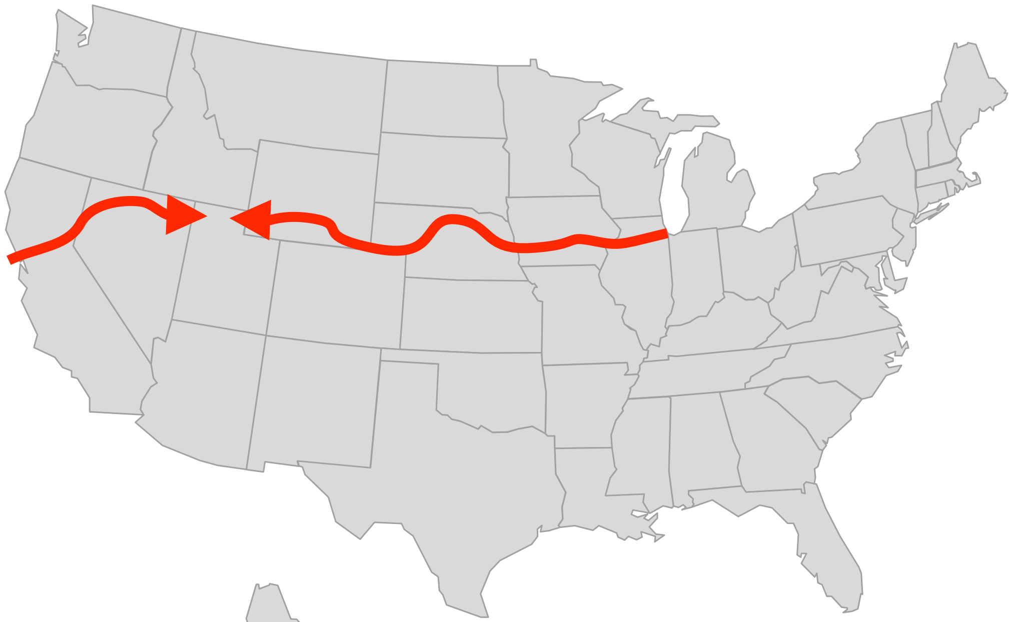
Top Down:

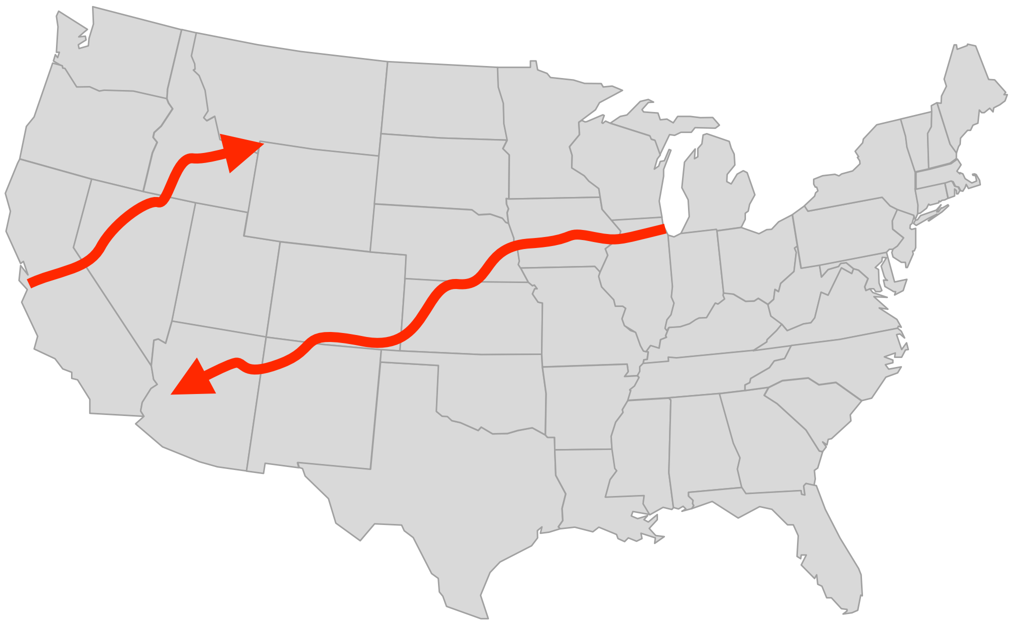
- Design of the control frameworks
- Naming, Identities, Authorization, RSpec

THE GOAL

Eventually both sides meet

WHERE ARE WE?



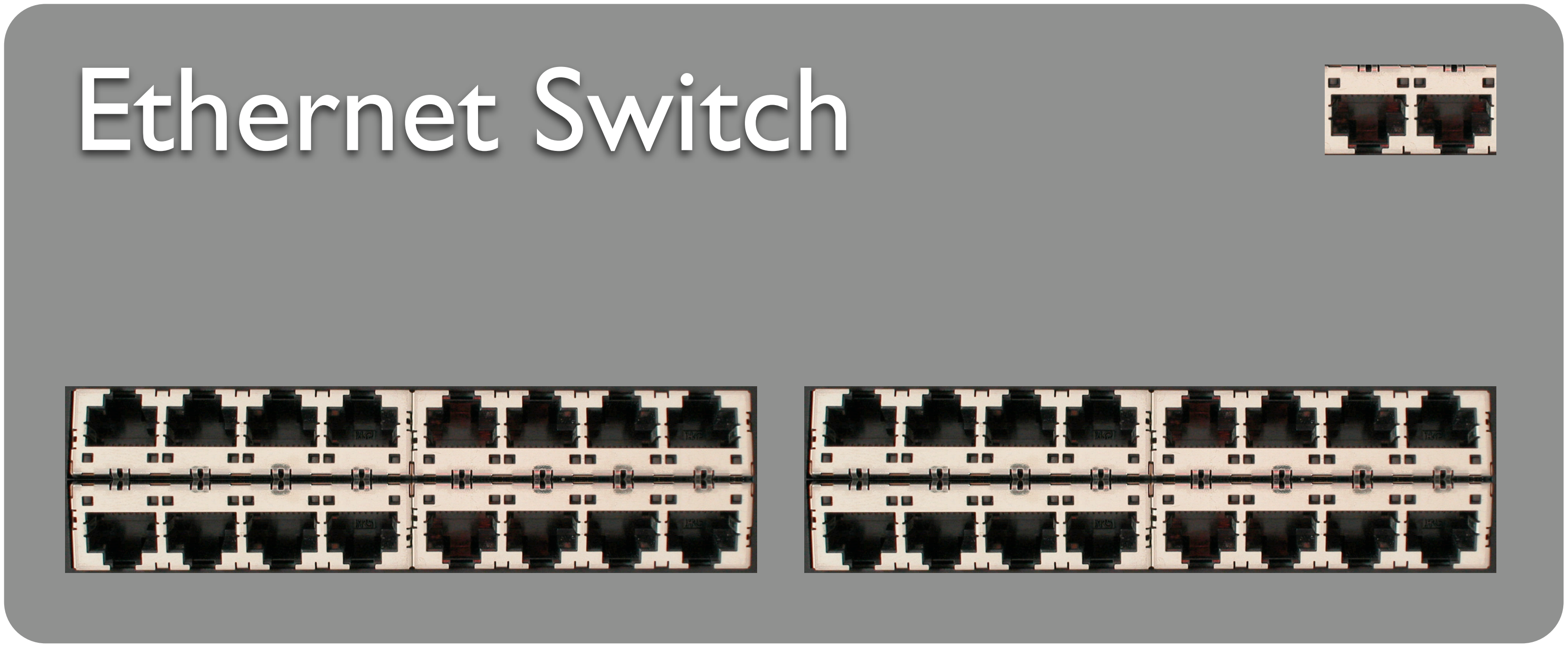


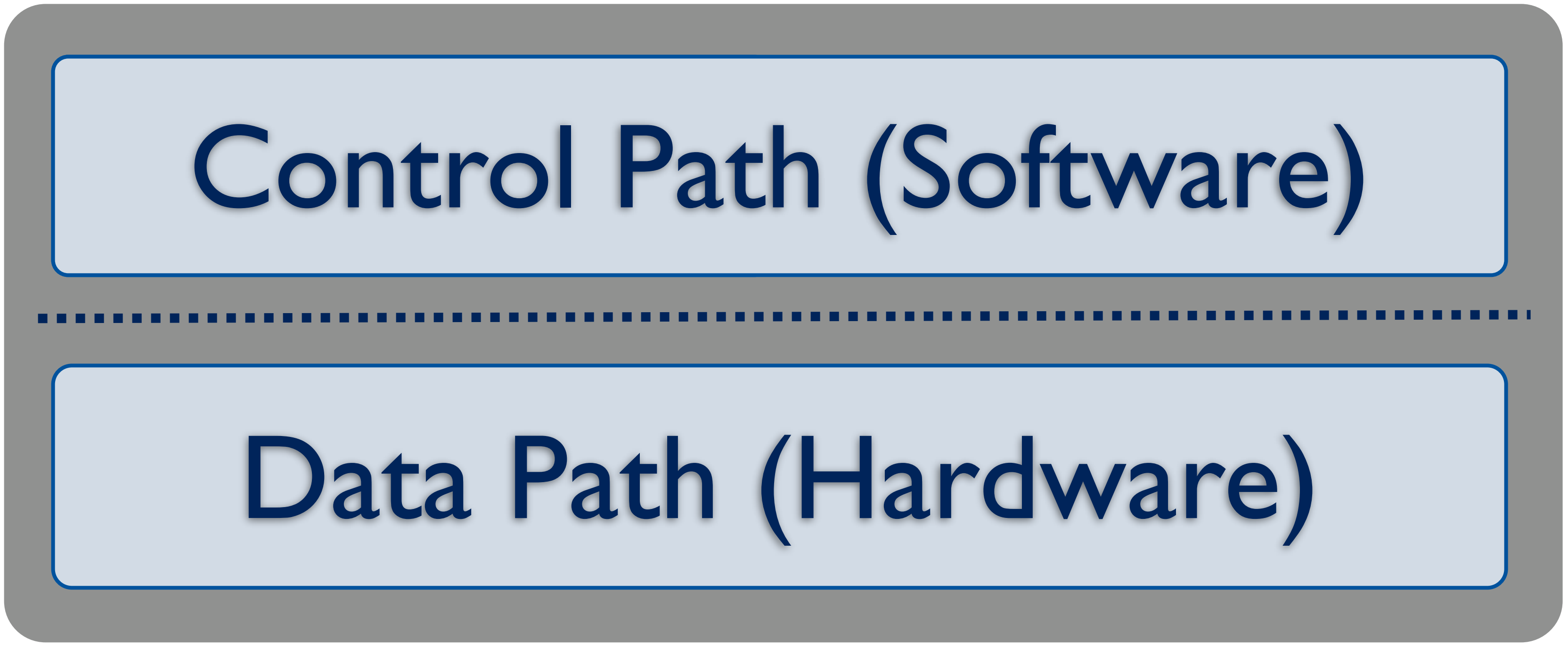
AGENDA

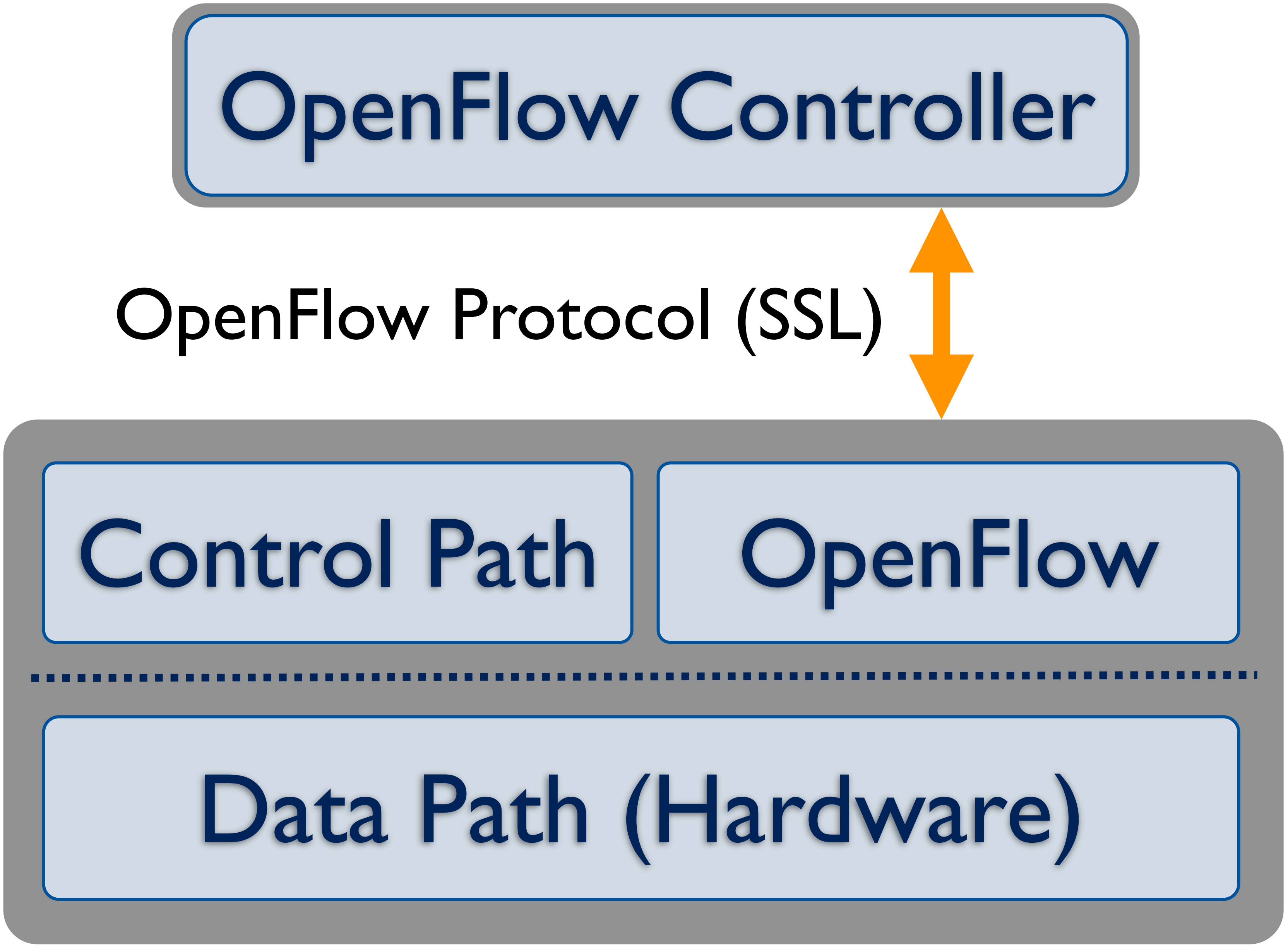
- Enterprise GENI
  - Review: OpenFlow
  - Status Enterprise GENI
  - Control Framework Requirements
- Control Framework
  - Our Perspective
- Proposal

AGENDA

- Enterprise GENI
  - Review: OpenFlow
  - Status Enterprise GENI
  - Control Framework Requirements
- Control Framework
  - Our Perspective
- Proposal

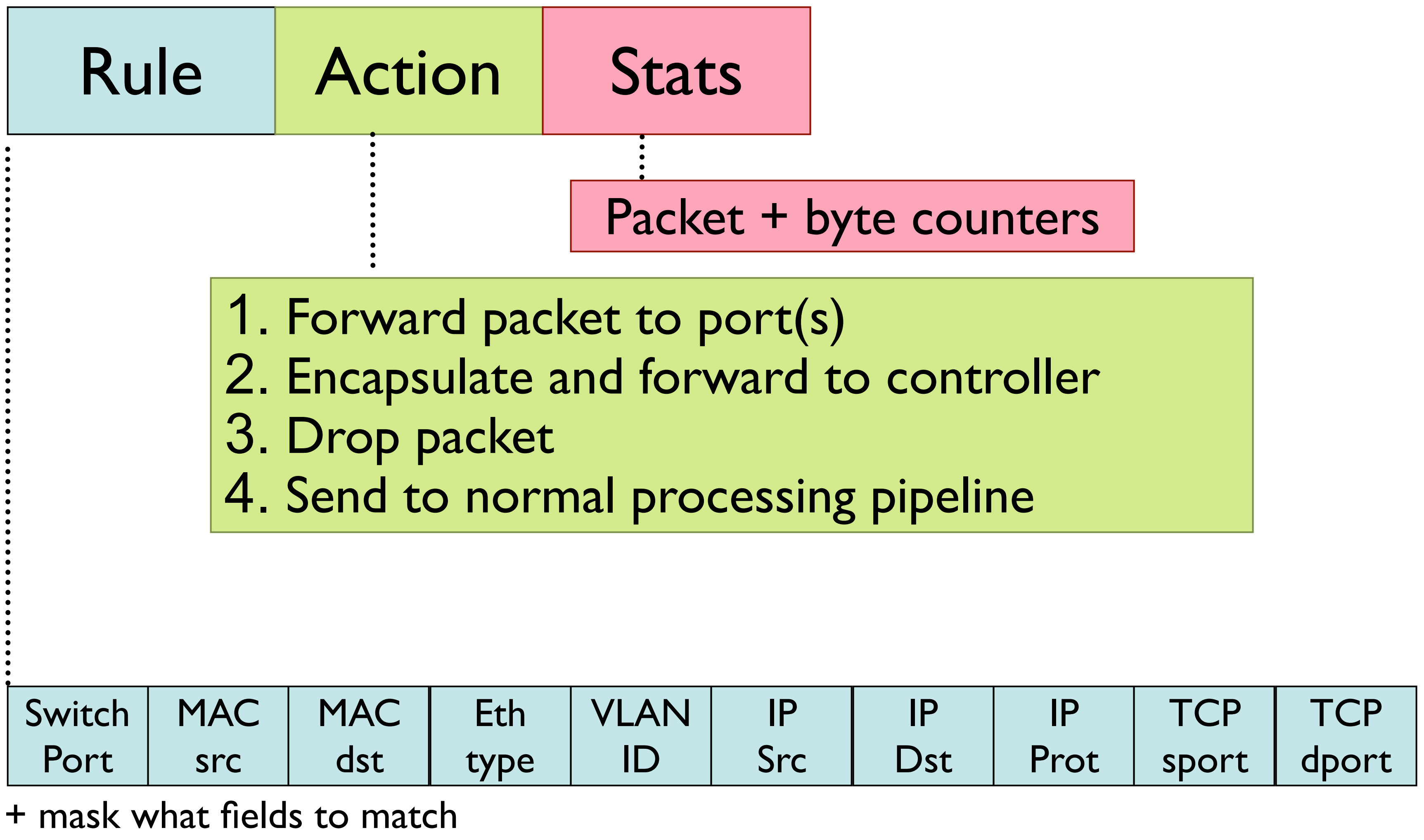

Ethernet Switch


Control Path (Software)
Data Path (Hardware)


OpenFlow Controller
OpenFlow Protocol (SSL)
Control Path
OpenFlow
Data Path (Hardware)

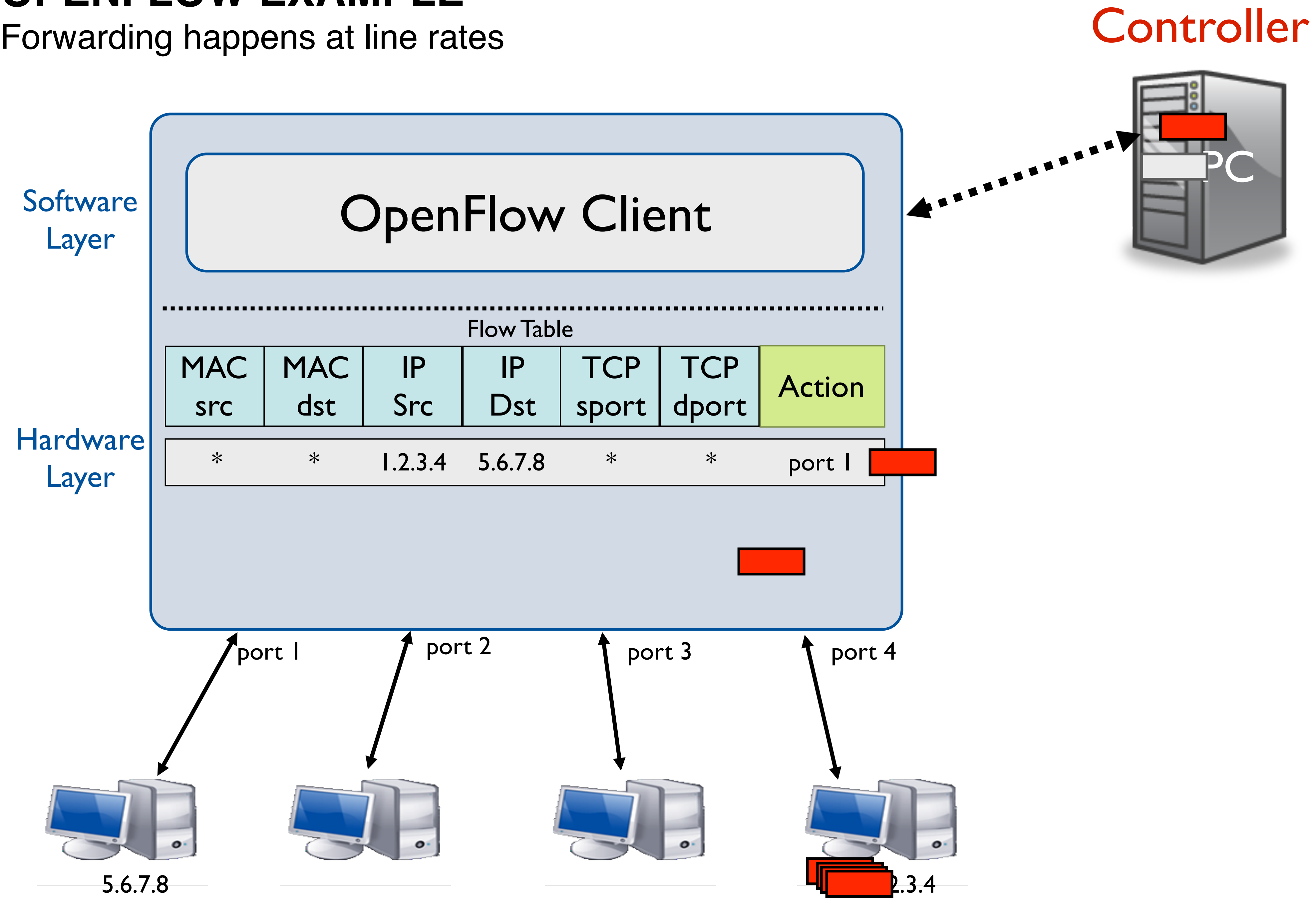
FLOW TABLE ENTRY

Allows to make forwarding decisions based on Layers 1-4


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
Switch Port
MAC src
MAC dst
Eth type
VLAN ID
IP Src
IP Dst
IP Prot
TCP sport
TCP dport
+ mask what fields to match

OPENFLOW EXAMPLE

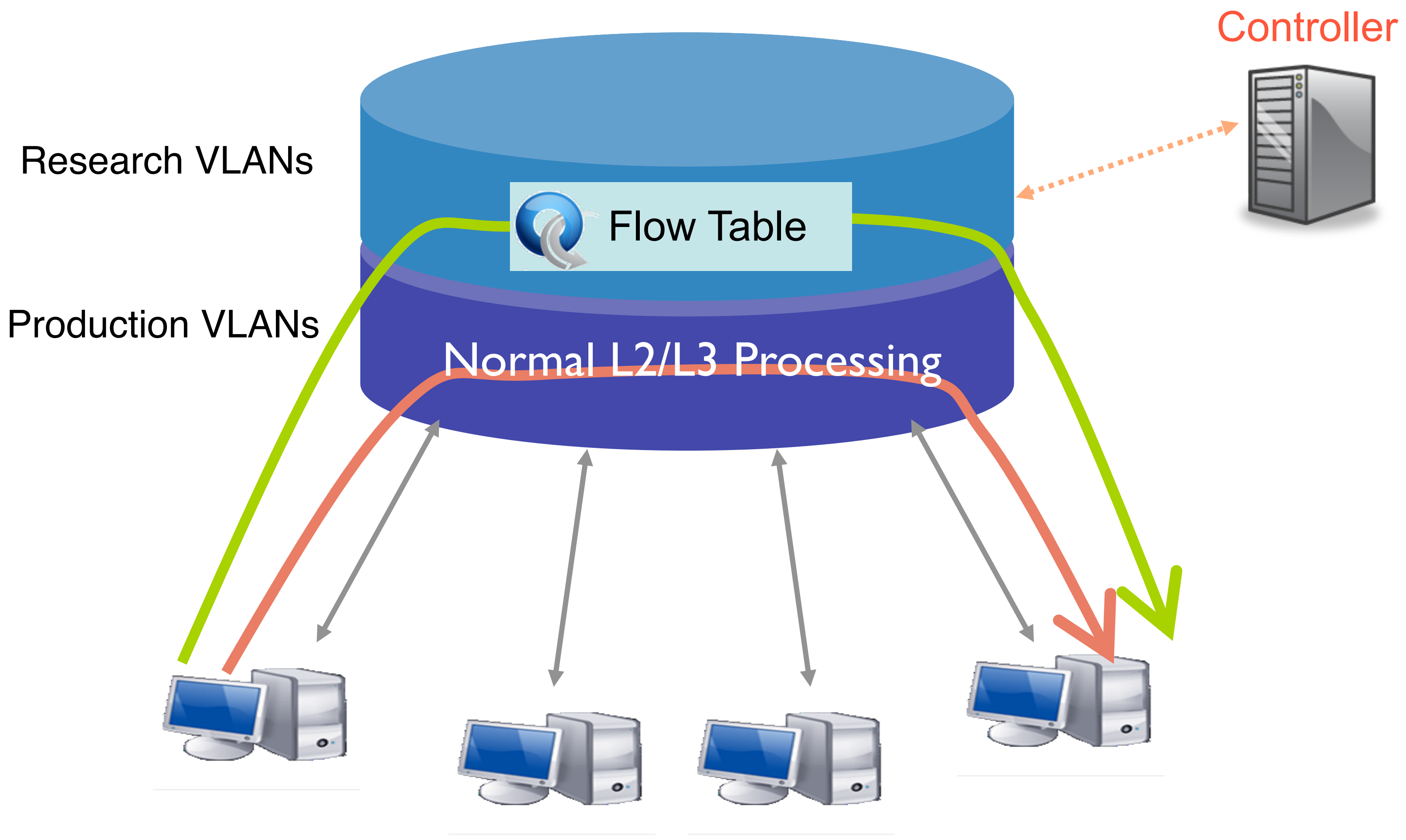
Forwarding happens at line rates



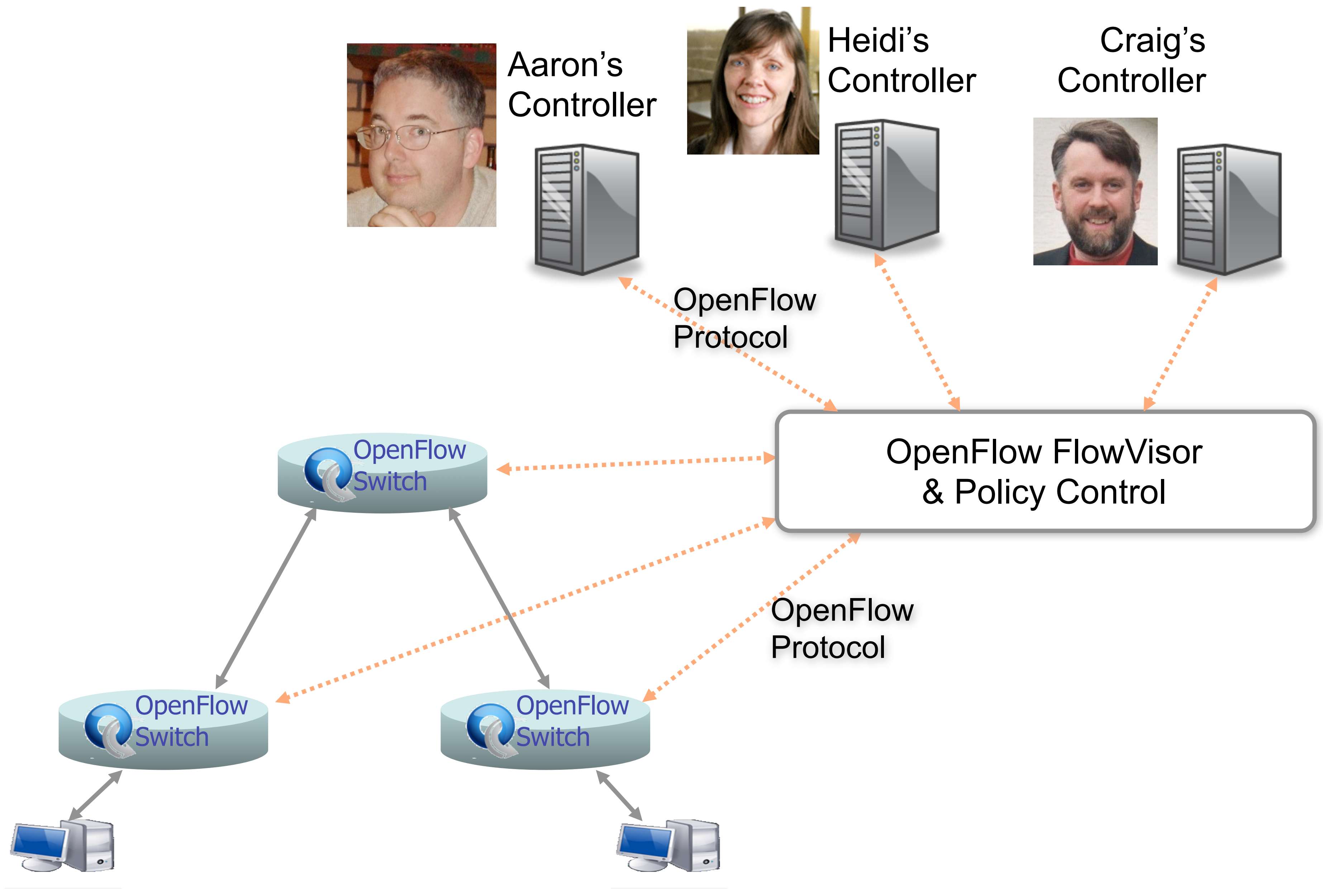
MAC src	MAC dst	IP Src	IP Dst	TCP sport	TCP dport	Action
*	*	1.2.3.4	5.6.7.8	*	*	port 1

SWITCH BASED VIRTUALIZATION

Exists for NEC, HP switches but not flexible enough for GENI

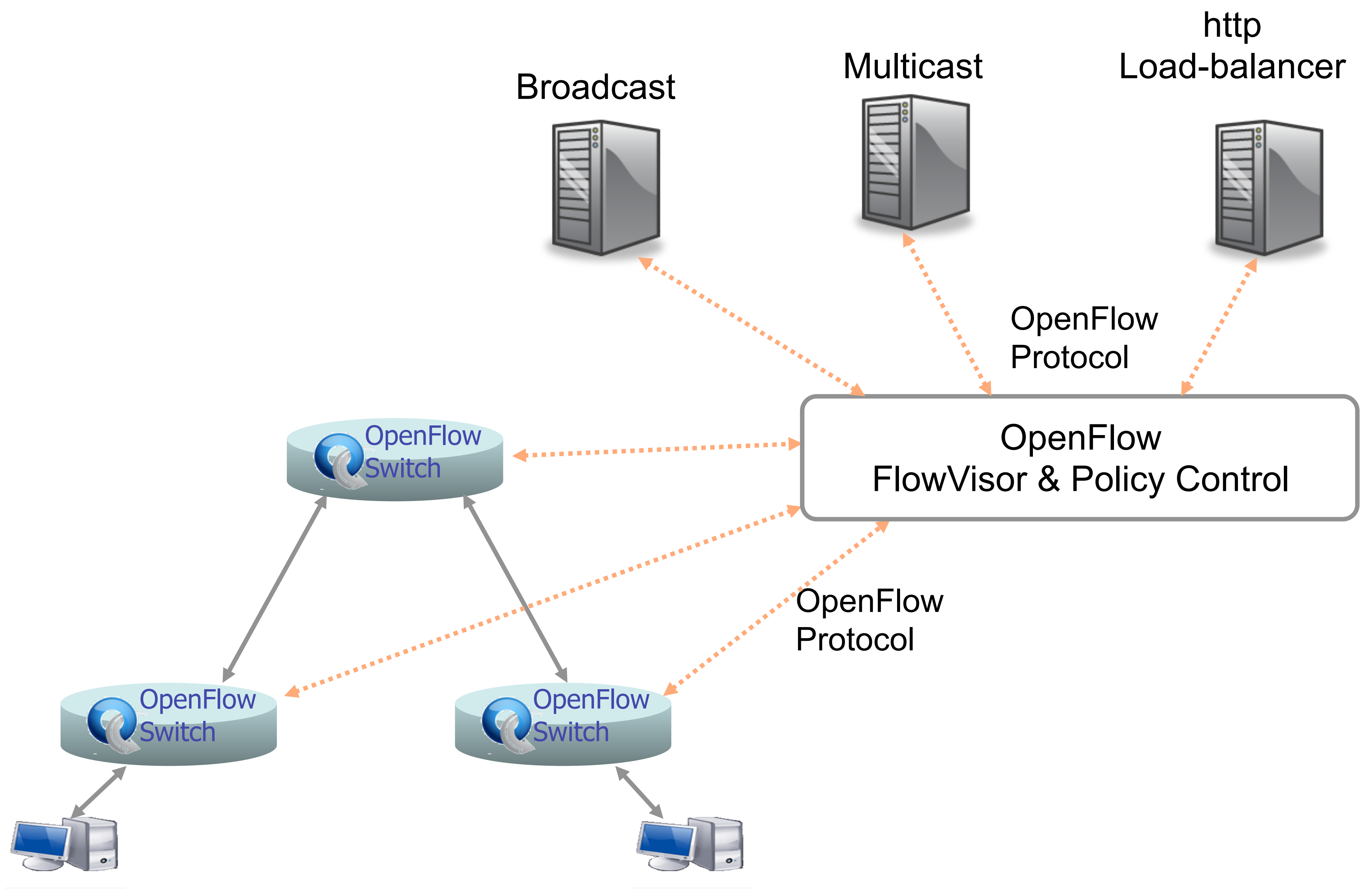


FLOWVISOR BASED VIRTUALIZATION



FLOWVISOR BASED VIRTUALIZATION

Separation not only by VLANs, but any L1-L4 pattern

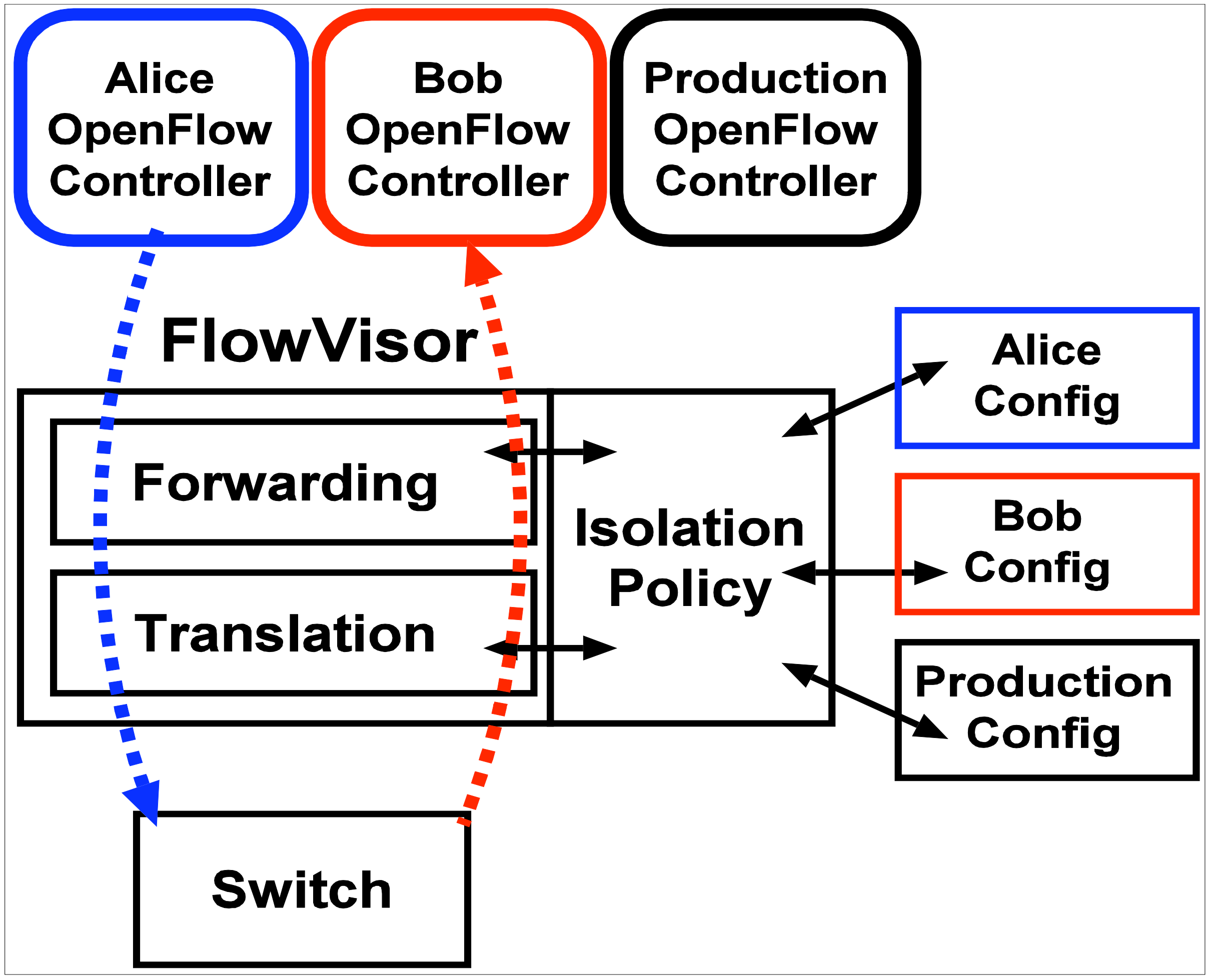


AGENDA

- Enterprise GENI
  - Review: OpenFlow
  - Status Enterprise GENI
  - Control Framework Requirements
- Control Framework
  - Our Perspective
- Proposal

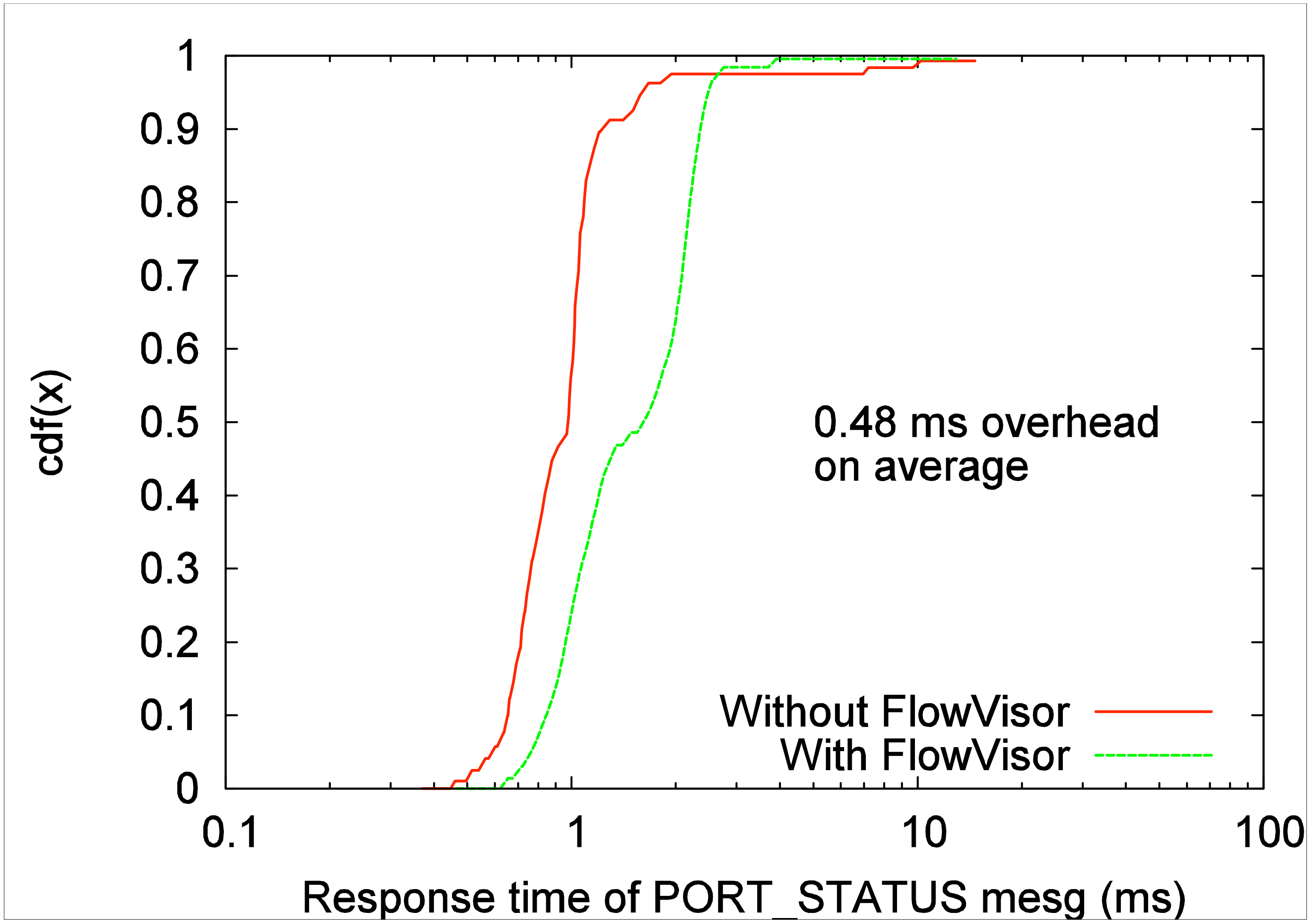
FLOWVISOR STATUS

- Tested in 15 node, intercontinental WAN
- Tested with five experiments concurrently



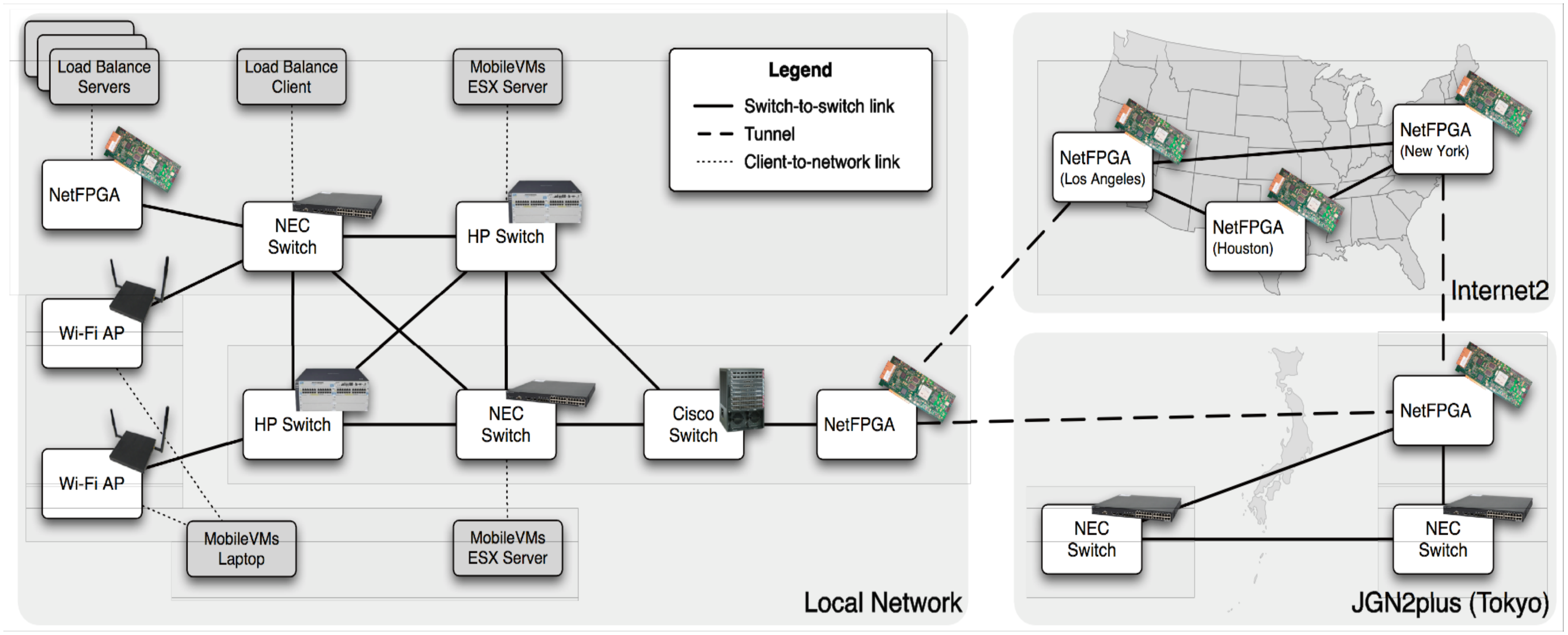
FLOWVISOR PERFORMANCE

Performance is ready for production deployment



FLOWVISOR EXPERIMENTAL TOPOLOGY

Tested in WAS and LAN environment



STANFORD OPENFLOW DEPLOYMENT

Phase 1 (ongoing)

- Gates Building, 3A Wing only
- Two switches (HP ProCurve 5400)
- 4 Wireless APs
- ~25 users

Phase 2 (1H2009)

- Gates Building, All Floors
- 23 Switches (HP ProCurve 5400)
- Wireless TBD
- Hundreds of users

Phase 3 (2H2009)

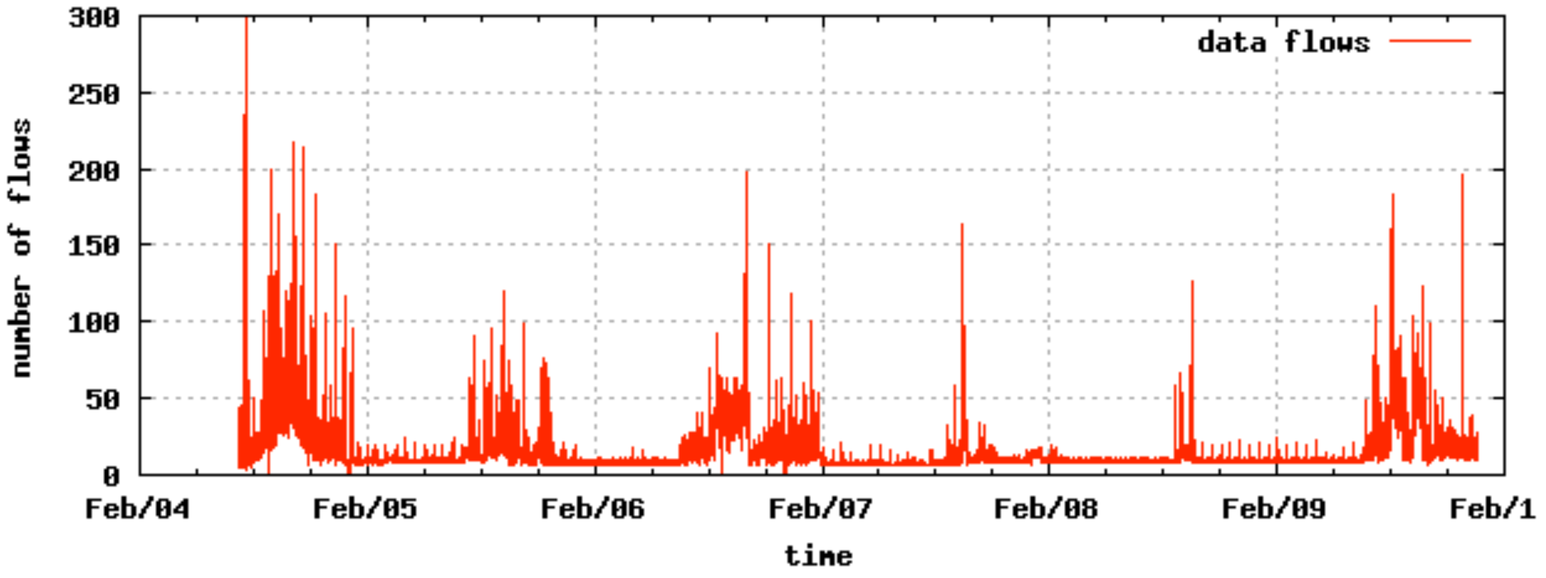
- Packard and CIS Buildings
- Switch Count TBD (HP ProCurve 5400)
- Wireless TBD
- > 1000 users

Stanford Enterprise GENI

- Small production deployment can be virtualized today
- Stanford Enterprise GENI targeted for GEC5 time frame

STANFORD OPENFLOW USAGE

OpenFlow Switches are carrying production traffic today



OPENFLOW ROADMAP 2009

OpenFlow Campus Trials (2H2009)

- Idea originated in CIO Meeting organized by GPO
- OpenFlow deployments at 8 Universities across the U.S.
- Four Vendors: Cisco, Juniper, HP, NEC

OpenFlow Internet2 Backbone (today)

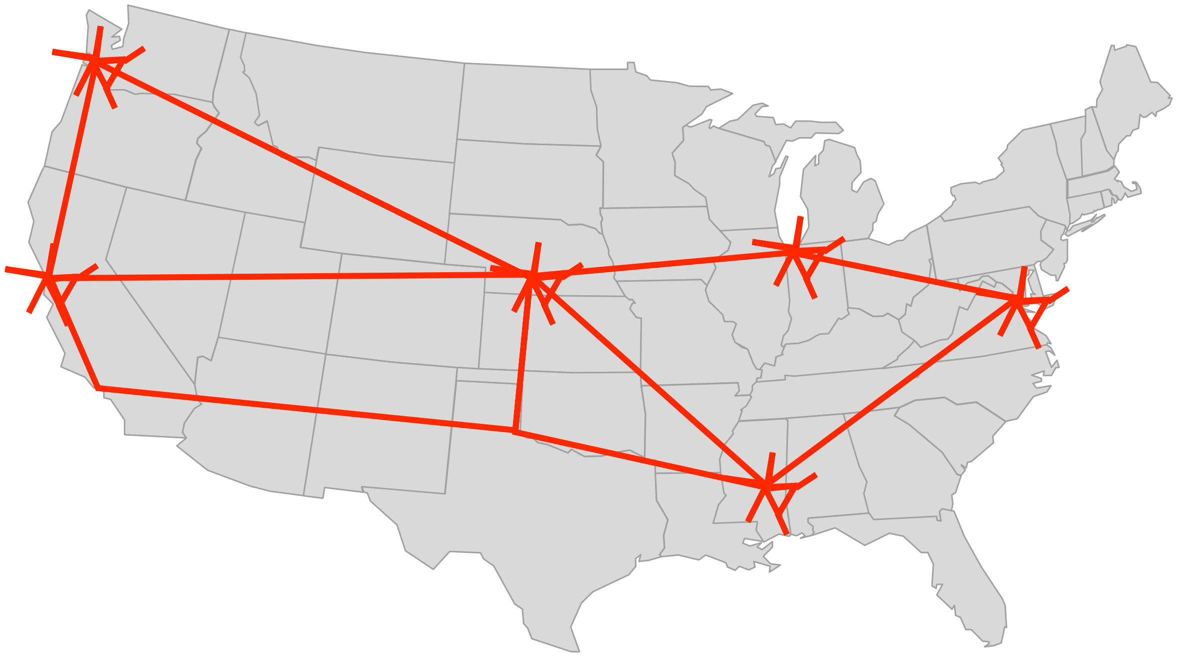
- Part of the GEC4 Demo
- Based on NetFPGA, Juniper MX

Obvious Next Steps:

- Virtualize them using the FlowVisor
- Make them accessible via Aggregate Manager

GOAL FOR OPENFLOW SUBSTRATE LATE 2009

5-6 Deployments plus Backbone



Goal is in late 2009 we will have an initial Enterprise GENI substrate that spans the country.

Now we need is a Control Framework!

AGENDA

- Enterprise GENI
  - Review: OpenFlow
  - Status Enterprise GENI
  - Control Framework Requirements
- Control Framework
  - Our Perspective
- Proposal

USE CASE: VLAN BASED PARTITIONING

Basic Idea: Partition Flows based on Ports and VLAN Tags

- Traffic entering system (e.g. from end hosts) is tagged
- VLAN tags consistent throughout substrate

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport
*	*	*	*	1,2,3	*	*	*	*	*
*	*	*	*	4,5,6	*	*	*	*	*
*	*	*	*	7,8,9	*	*	*	*	*

USE CASE: NEW CDN - TURBO CORAL ++

Basic Idea: Build a CDN where you control the entire network

- All traffic to or from Coral IP space controlled by Experimenter
- All other traffic controlled by default routing
- Topology is entire network
- End hosts are automatically added (no opt-in)

	Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport
	*	*	*	*	*	84.65.*	*	*	*	*
	*	*	*	*	*	*	84.65.*	*	*	*
	*	*	*	*	*	*	*	*	*	*

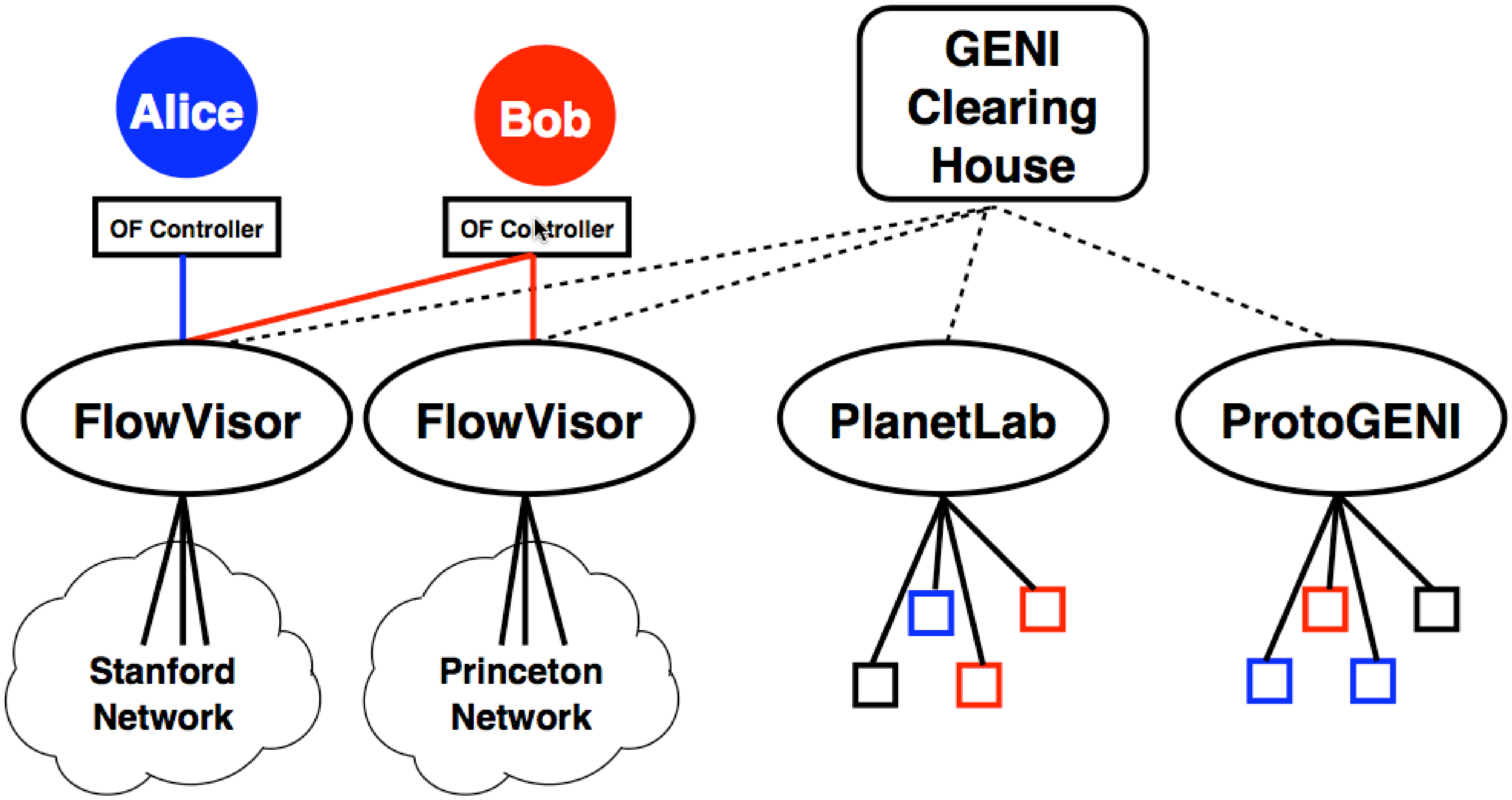
USE CASE: AARON'S IP

- A new layer 3 protocol
- Replaces IP
- Defined by a new Ether Type

	Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport
	*	*	*	AaIP	*	*	*	*	*	*
	*	*	*	!AaIP	*	*	*	*	*	*

ENTERPRISE GENI ARCHITECTURE

Needs to be changed: Multiple switches, controller under control of experimenter, graphics.



OPENFLOW BASED GENI

Differences to other Implementations

- No Components or Component Managers (in the GENI sense)
  - Aggregate is managed centrally by the FlowVisor
  - No external login into components of any kind
- Forwarding decisions done by external service
  - FlowVisor connects to Experimenter’s OpenFlow Controller
- The Network can be sliced at any layer
  - Separation by physical topology or VLAN (L1)
  - Separation by MAC addresses (L2)
  - Separation by IP blocks (L3)
  - Separation by port (L4)

OVERALL REQUIREMENTS

From a Control Framework, we initially need:

- Protocol to talk to GCH
- Infrastructure Description (Switches, Links)
- Slice Management Operations (Initialize, Release)
- Define and Manage Traffic Sources and Sinks
  - Opt-in or Opt-out for hosts
  - Connection with Internet, GENI Backbone, other networks
- Mechanism for specifying external controllers

TECHNICAL REQUIREMENTS

Protocol

- Clearly Specified and Verifiable
- Language, OS and Development Tool Agnostic
- As few external dependencies as possible (if any)
- Ideally: Independent reference implementations and test suite

RSpec

- Clearly Specified (allow some interoperability)
- Extensible (GENI is work in progress)
- Easy to implement

AGENDA

- Enterprise GENI
  - Review: OpenFlow
  - Status Enterprise GENI
  - Control Framework Requirements
- Control Framework
  - Our Perspective
- Proposal

RSPEC

- Currently XML with and XSD definition
  - Clearly typed, works great
- Specification in flux, no complete spec proposed yet
  - Initially we expect everyone to define their own
  - This means initially no or little interoperability
  - Defining a standard that works for everyone will be hard work

- Some existing XSD’s have heavy external dependencies
  - For our substrate, this seems undesirable

RSPEC
XSD definition is dependent on external references, development tool

```
<xsd:complexType name="LinkSpec">
    <xsd:sequence>
      <xsd:element default="0" ecore:unique="true" maxOccurs="unbounded"
        minOccurs="0" name="bw" type="ecore:EInt"/>
      <xsd:element default="0" ecore:name="max_alloc" ecore:unique="true"
        maxOccurs="unbounded" minOccurs="0" name="max_alloc" type="ecore:EInt"/>
    </xsd:sequence>
    <xsd:attribute default="&quot;&quot;" ecore:changeable="false"
        ecore:unsettable="false" name="type" type="ecore:EString"/>
    <xsd:attribute ecore:name="init_params" name="init_params"
        type="ecore:EByteArray"/>
    <xsd:attribute default="0" ecore:name="min_alloc" ecore:unsettable="false"
        name="min_alloc" type="ecore:EInt"/>
    <xsd:attribute ecore:changeable="false" ecore:reference="pl:IfSpec"
        name="endpoint" use="required">
      <xsd:simpleType>
        <xsd:list itemType="xsd:anyURI"/>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute ecore:name="start_time" name="start_time" type="ecore:EDate"/>
    <xsd:attribute name="duration" type="ecore:EDate"/>
</xsd:complexType>
```

AUTHENTICATION, AUTHORIZATION AND NAMING

Current Architecture

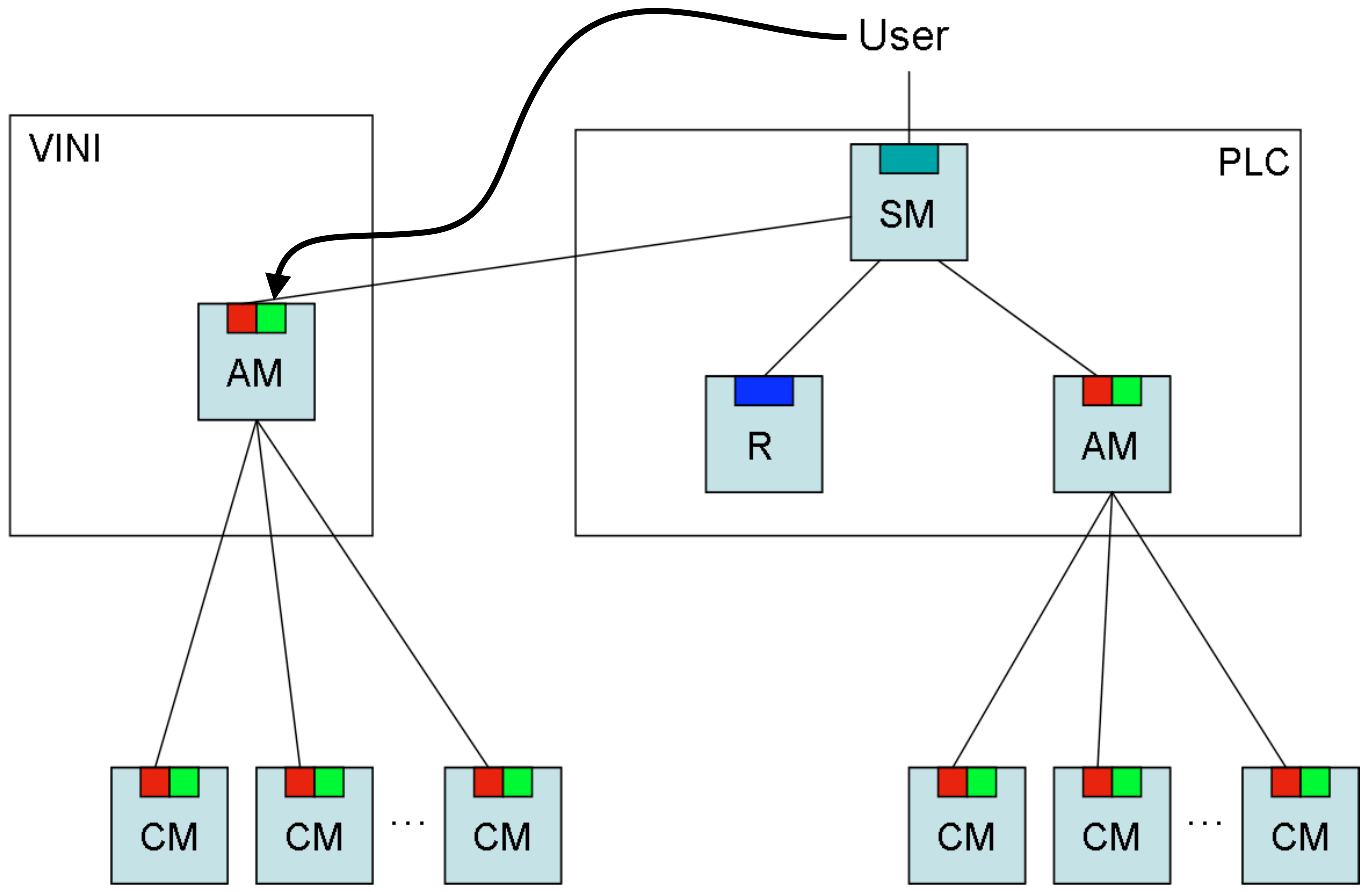
- Client and Server certificates to authenticate connections
- Certificate Infrastructure for signing tickets, GIDs etc.

PKI is extremely heavyweight, more complexity than we need

- We only need one server side certificate for SSL
- All other authentication can be done with shared secrets
  - Server side SSL certs and shared secrets are the de-facto standard in the internet
  - Why would we need anything else for GENI?
- Existing naming schemes are sufficient (don’t need GIDs)

CURRENT CONTROL FRAMEWORK PROPOSAL

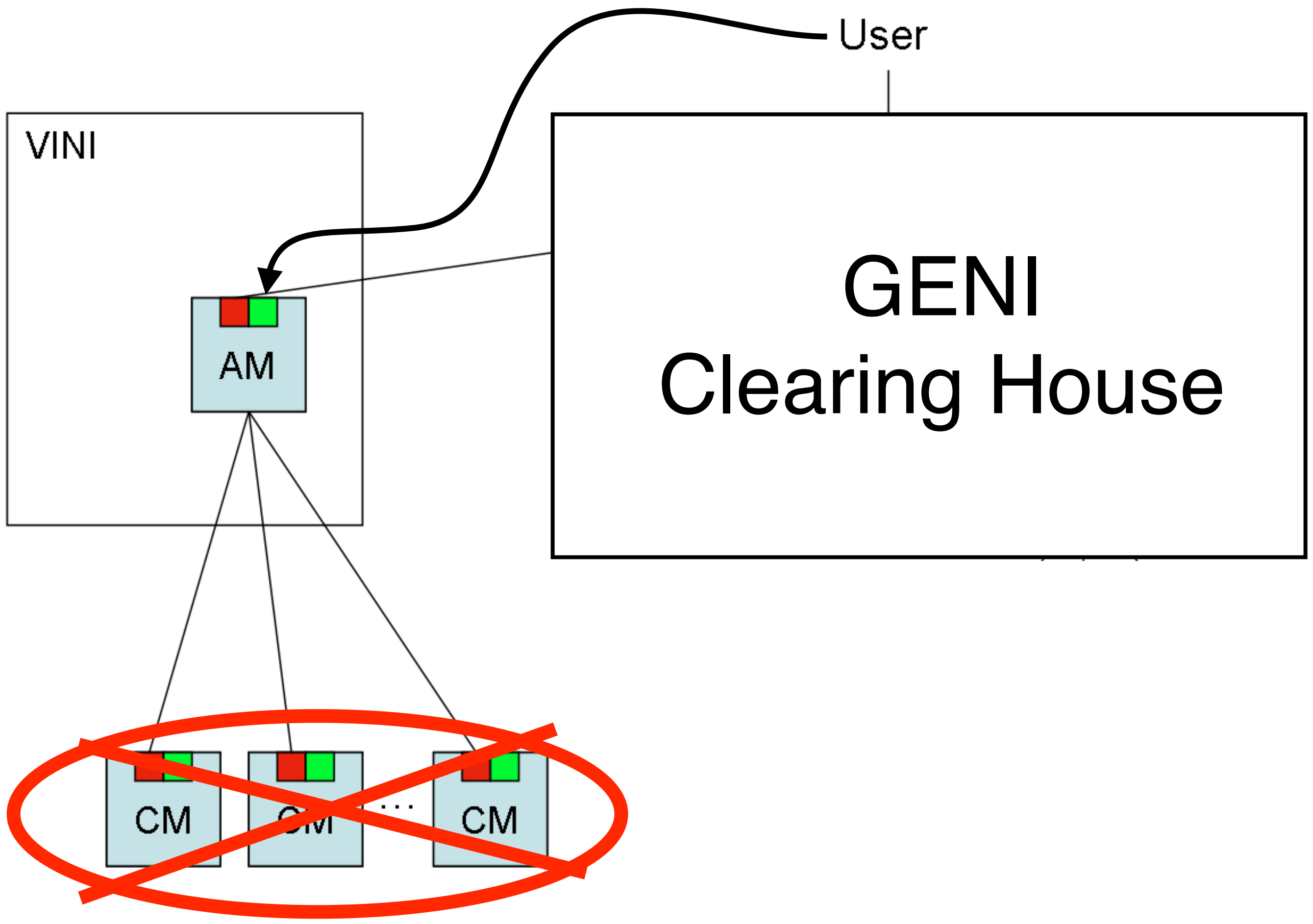
Based on PlanetLab Central



Source: PlanetLab Implementation of the Slice-Based Facility Architecture. February 10, 2009.

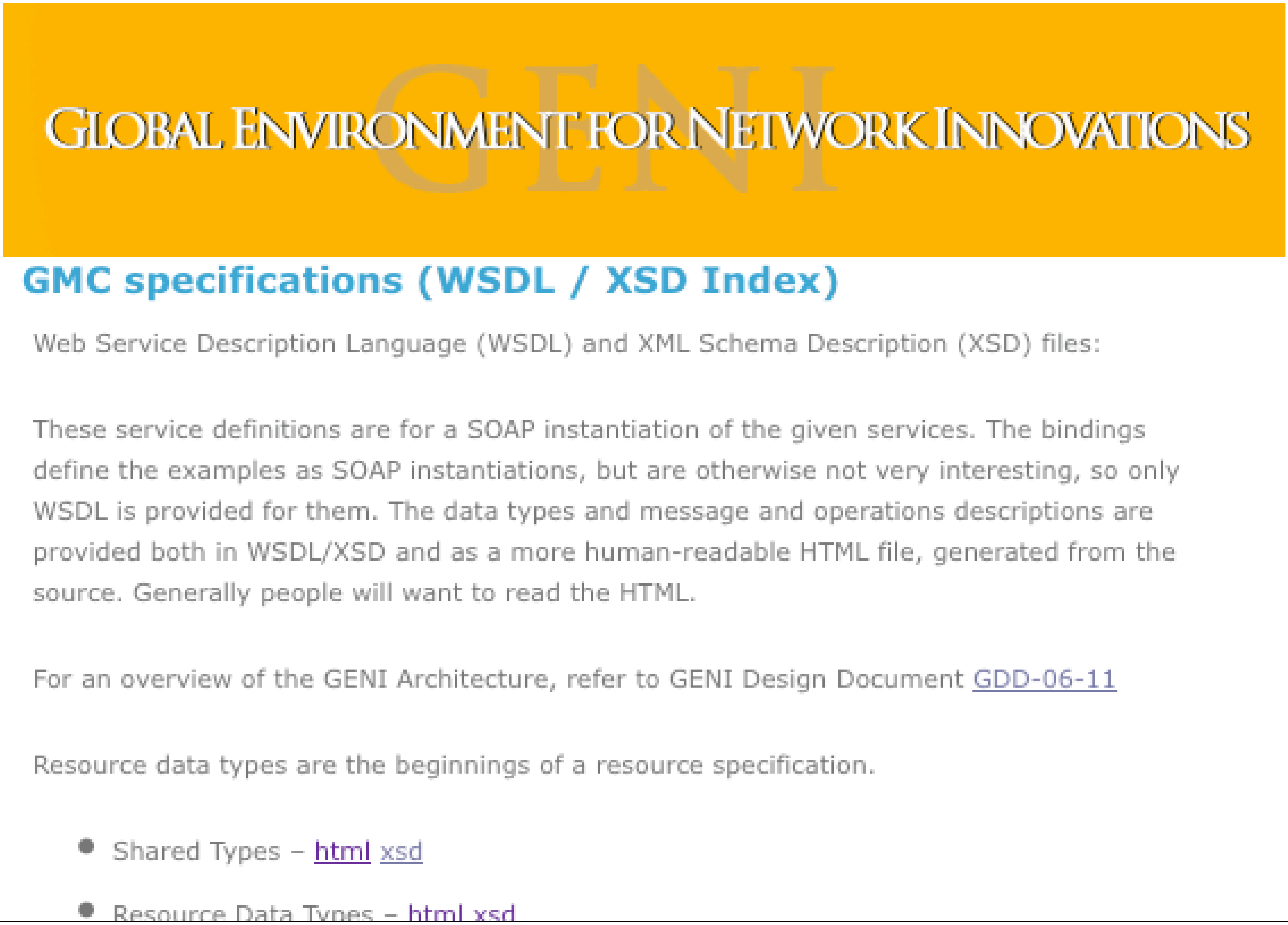
CURRENT CONTROL FRAMEWORK PROPOSAL

Based on PlanetLab Central



Source: PlanetLab Implementation of the Slice-Based Facility Architecture. February 10, 2009.

SOAP BASED GENI PROTOCOL



GLOBAL ENVIRONMENT FOR NETWORK INNOVATIONS

GMC specifications (WSDL / XSD Index)

Web Service Description Language (WSDL) and XML Schema Description (XSD) files:

These service definitions are for a SOAP instantiation of the given services. The bindings define the examples as SOAP instantiations, but are otherwise not very interesting, so only WSDL is provided for them. The data types and message and operations descriptions are provided both in WSDL/XSD and as a more human-readable HTML file, generated from the source. Generally people will want to read the HTML.

For an overview of the GENI Architecture, refer to GENI Design Document GDD-06-11

Resource data types are the beginnings of a resource specification.

- Shared Types – html xsd
- Resource Data Types – html xsd

GENIWRAPPER PROTOCOL

Essentially XMPRPC between Python libraries

Implemented in geniwrapper:

- Basic idea: Encode messages in XML
- Implemented with Python's SimpleXMLRPCServer/XLMRPCLib
  - Create a proxy object `server = xmlrpclib.ServerProxy()`
  - Now I can call `server.function(parameters)`
  - Python will marshal/de-marshal object on each side automatically

```
server = ServerProxy("http://clearinghouse.geni.net")
try:
    self.server.get_ticket(credential, name, rspec)
```

Source: Geniwrapper from http://svn.planet-lab.org/svn/geniwrapper, version Feb 10th

GENIWRAPPER PROTOCOL

Wire Format (Simplified)

```
<?xml version='1.0'?>
  <methodCall>
    <methodName>create_slice</methodName>
    <params>
      <param>
        <value><string>-----BEGIN CERTIFICATE-----
MFcwTQIBATADBgEAMAAwHhcNMDkwMjA0MTEzNTM0WhcNMTQwMjAzMTEzNTM0WjAX
MRUwEwYDVQQDEwxSb2IgU2hlcndvb2QwCDADBgEAAwEAMAMGAQADAQA=
-----END CERTIFICATE-----</string></value>
      </param>
    <param>
      <value><string>rob.sherwood@stanford.edu</string></value>
    </param>
    <param>
      <value><string>some stuff</string></value>
    </param>
  </params>
</methodCall>
```

GENIWRAPPER PROTOCOL

Interface is dynamically generated at runtime

XMLRPC Implementation

- XML structure of the protocol is based on types of python objects
- Python is a dynamically typed language
- Programmer can change types of objects at run time

Consequences

- Protocol may change at run time
- No real definition what the “correct” protocol is
- Essentially impossibly to implement protocol in other languages.
- Requires use of Geniwrapper Python code.
- Security Implications

Source: Analysis of Geniwrapper from http://svn.planet-lab.org/svn/geniwrapper, version Feb 10th

AGENDA

- Enterprise GENI
  - Review: OpenFlow
  - Status Enterprise GENI
  - Control Framework Requirements
- Control Framework
  - Our Perspective
- Proposal

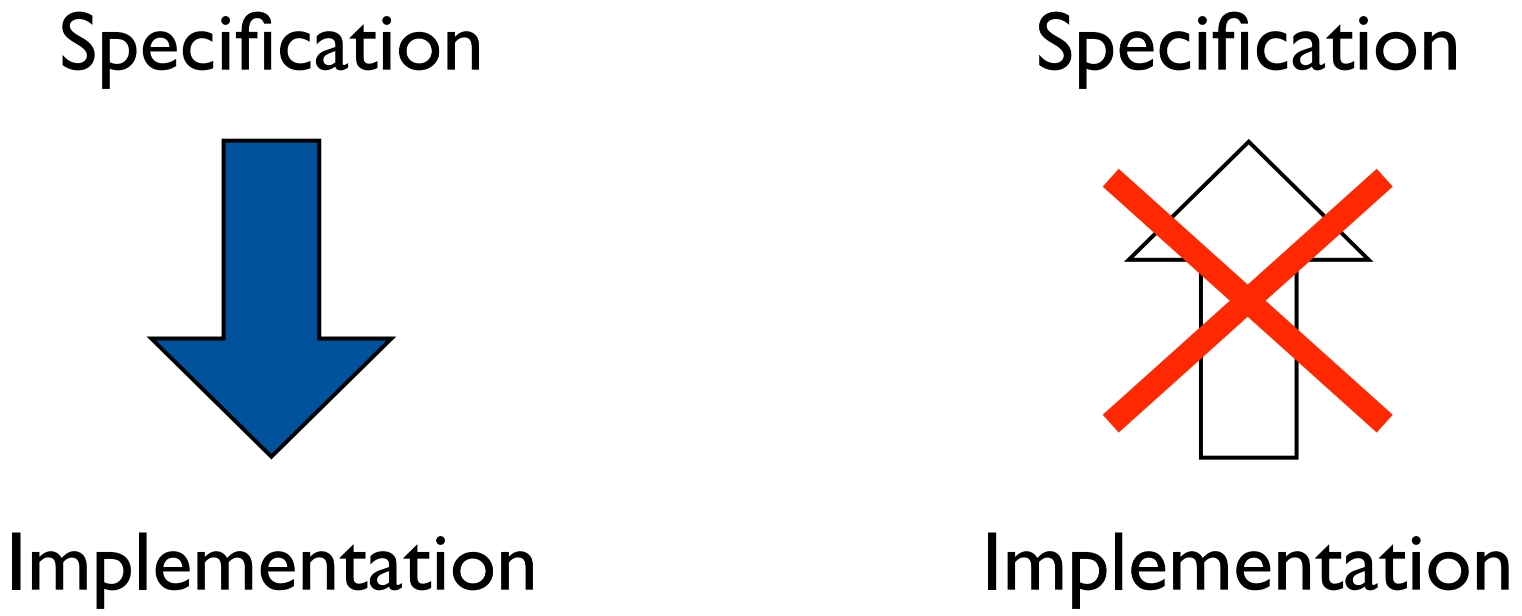
SUCCESSFUL PROTOCOLS

The most frequently used remote APIs today are all SOAP or REST

API Name	API Type	Auth Type
Google Search API	REST/JSON*	SSL+Secret
PayPal	SOAP	SSL+Secret
EBay	SOAP	SSL+Secret
Microsoft Live Search	SOAP	SSL+Secret
Amazon EC2	SOAP	SSL, Secret, SSH
Yahoo Search	REST	SSL+Secret
OpenID	REST (sort of)	SSL+HMAC
Geniwrapper	Dynamic XML RPC	SSL+Full PKI
Geni Ultralight	SOAP	SSL+Secret

INTERFACE DESIGN

For protocols, specification should drive implementation



PROTOCOL PROPOSAL
Use Lightweight SOAP Protocol

- Back to Plan A: Use SOAP to define the protocol
  - WSDL “Contract” between control framework and substrates
  - Easy to achieve interoperability
  - Language independent
  - Platform independent
- If done right, this is extremely lightweight and developer friendly
  - Marshalling/Demarshalling code is generated automatically
    - Tools to do this exist for any language
  - Great debugging tools exist

CHANGES TO GENIWRAPPER

Changes are minimal, only topmost Python layer is affected

Current implementation creates API interface via the server proxy at run time:

```
server = ServerProxy("http://clearinghouse.geni.net")
// This generates XML structures at run time
self.server.get_ticket(credential, name, rspec)
```

With SOAP, interface is defined in the WSDL and server proxy is generated via command line tool:

```
wsdl2py http://www.geni.net/wsdl.xml
```

```
from GeniClearingHouse import *
server = GCHServiceLocator("http://clearinghouse.geni.net")
r = GCHGetTicketRequest()
r.credential, r.name, r.rspec = credential, name, rspec
server.GetTicker(r)
```

PROTOCOL CHANGES

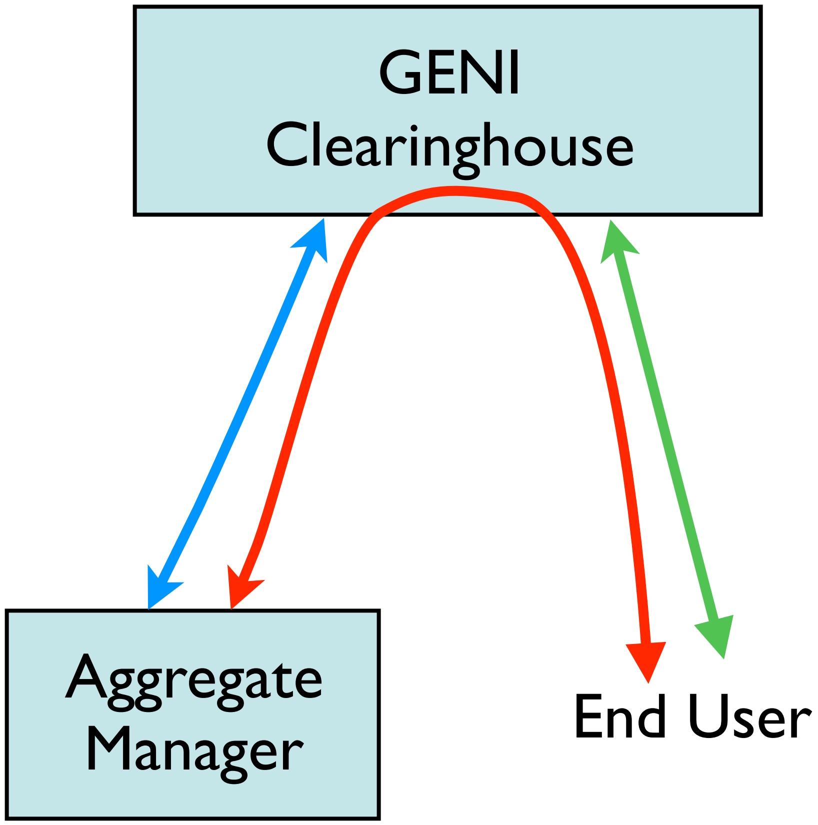
Eliminate use of PKI by centralizing control

All Communication via GCH

- e.g. slice management

Vastly simplified Security Model

- One SSL Cert at GCH
- Shared secret to authenticate Aggregate Manager
- No more ticket signing
- No more certificates in global identities
- No more PKI required
- If we need assertions in the future, use HMAC



SIMPLIFIED AGGREGATE MANAGER API

Very small set of mandatory messages

Messages from AM to GCH

- Registry Interface
- Update, Add, Delete Information

Messages from GCH to AM

- CreateSlice(RSpec)
- DeleteSlice(RSpec)

Observations

- This is sufficient to manage Enterprise GENI
  - What would ResetSlice mean for a stateless aggregate?
- Aggragates should not be burdened by complexity they don’t need
  - Make other messages optional

NAMING

De-facto internet standards are email addresses, host names and URLs

- Principals/Users: email addresses
  - Human readable
  - Can be used for bootstrapping authenticators
  - Future integration with external authentication (OpenID etc.)

- Hosts/Nodes: domain names
  - Hirarchical, globally unique

- Everything else: URLs
  - Essentially add information to host names
  - What else do we really need?

For Enterprise GENI, we don’t need complex structure of GID

WHY A SIMPLER CONTROL FRAMEWORK?

Today

- We are ready to start integration with a GCH
- The currently proposed framework does not fit our needs
  - Substantial complexity that we (at least initially) don’t require
  - Dependency on large code base that is actively being developed
- Not a short-term solution for us

By 2H2009

- We expect to have a backbone substrate
- We expect to have 5-7 local substrates with 20-100 switches
- Other groups will have their own (potentially larger) substrates

If we have a simple control framework, we hope a first GENI deployment would be usable for researchers by December 2009

Thank you!