Project 3 and Software-Defined Networking (SDN)

EE122 Fall 2011

Scott Shenker

http://inst.eecs.berkeley.edu/~ee122/

Materials with thanks to Jennifer Rexford, Ion Stoica, Vern Paxson and other colleagues at Princeton and UC Berkeley
Introducing Project 3

- Scott: Background on Software-Defined Networking (40 minutes)
- Yahel: Project Overview (10 minutes)
- Murphy: Software Architecture (10 minutes)
- TD and Kyriakos: Demo and Details (15 minutes)
Preliminaries

• Wanted to let you program a real device
  – Marvell donated 250 of these “plug computers”, which we are sharing with NUST (Pakistan)

• Be gentle with us, we’ll be gentle with you…
  – You break it early, we’ll fix it early

• If you are interested in doing something neat with your box, send me a proposal and we’ll let you continue to play with the box after end of semester.

• Do not lose your device…we need it back.
• SDN is a new approach to networking
  – Not about “architecture”: IP, TCP, etc.
  – But about design of network control (routing, TE, …)

• Full Disclosure: SDN invented by Nicira Networks
  – Based on earlier work at Stanford, UCB, Princeton, CMU

• But this is not a sales pitch for Nicira
  – Nicira sells products that happen to use SDN internally
  – It does not sell SDN, nor market itself as an SDN company
Status of SDN

• Open Networking Foundation is standards body
  – SDN endorsed by 49 companies
  – Almost everyone who matters…..

• A few products on market, many more coming
  – Some large companies using SDN internally

• SDN has won the war of words, the real battle over customer adoption is just beginning…..
How is Project 3 Related to SDN?

• Project 3 uses SDN technology
  – But SDN will be invisible to you (as it should be!)

• You will write program to control single switch
  – **Easy** (in principle)!

• **Similar program could control entire network**
  – Impossible without SDN…and whole goal of SDN

• I will provide motivation and context for SDN
  – Absolutely no design details
Rules of Engagement

• Because short on time, I will not ask questions

• If you don’t understand what I’m saying, stop me.

• To pursue points more deeply, do so after class
  – Goal here is not depth, but general intuition about SDN
Two Key Definitions

• **Data Plane**: processing and delivery of packets
  – Based on state in routers and endpoints
  – E.g., IP, TCP, Ethernet, etc.
  – Fast timescales (per-packet)

• **Control Plane**: establishing the state in routers
  – Determines how and where packets are forwarded
  – Routing, traffic engineering, firewall state, …
  – Slow time-scales (per control event)
The Future of Networking,
and the Past of Protocols

Scott Shenker

with Martín Casado, Teemu Koponen, Nick McKeown
(and many others....)
Key to Internet Success: Layers

Applications
...built on...
Reliable (or unreliable) transport
...built on...
Best-effort global packet delivery
...built on...
Best-effort local packet delivery
...built on...
Physical transfer of bits
Why Is Layering So Important?

• Decomposed delivery into fundamental components

• Independent but compatible innovation at each layer

• A practical success of unprecedented proportions…

• …but an academic failure
Built an Artifact, Not a Discipline

• Other fields in “systems”: OS, DB, DS, etc.
  - Teach basic principles
  - Are easily managed
  - Continue to evolve

• Networking:
  - Teach big bag of protocols
  - Notoriously difficult to manage
  - Evolves very slowly
Why Does Networking Lag Behind?

• Networks used to be simple: Ethernet, IP, TCP....

• New control requirements led to great complexity
  - Isolation ➔ VLANs, ACLs
  - Traffic engineering ➔ MPLS, ECMP, Weights
  - Packet processing middleboxes ➔ Firewalls, NATs,
  - Payload analysis ➔ Deep packet inspection (DPI)
  - ..... 

• Mechanisms designed and deployed independently
  - Complicated “control plane” design, primitive functionality
  - Stark contrast to the elegantly modular “data plane”
Infrastructure Still Works!

- Only because of “our” ability to master complexity
- This ability to master complexity is both a blessing…
  - …and a curse!
A Simple Story About Complexity

• ~1985: Don Norman visits Xerox PARC
  - Talks about user interfaces and stick shifts
What Was His Point?

- The ability to *master complexity* is not the same as the ability to *extract simplicity*

- When first getting systems to work….
  - Focus on mastering complexity

- When making system easy to use and understand
  - Focus on extracting simplicity

- You will never succeed in extracting simplicity
  - If don’t recognize it is different from mastering complexity
What Is *My* Point?

- Networking still focused on mastering complexity
  - Little emphasis on extracting simplicity from control plane
  - No recognition that there’s a difference….

- Extracting simplicity builds intellectual foundations
  - Necessary for creating a discipline….
  - That’s why networking lags behind
A Better Example: Programming

• Machine languages: no abstractions
  - Mastering complexity was crucial

• Higher-level languages: OS and other abstractions
  - File system, virtual memory, abstract data types, ...

• Modern languages: even more abstractions
  - Object orientation, garbage collection, ...

Abstractions key to extracting simplicity
“The Power of Abstraction”

“Modularity based on abstraction is the way things get done”

Barbara Liskov

Abstractions ➔ Interfaces ➔ Modularity

What abstractions do we have in networking?
Layers are Great Abstractions

• Layers only deal with the data plane

• We have no powerful control plane abstractions!

• How do we find those control plane abstractions?

• Two steps: define problem, and then decompose it.
The Network Control Problem

• Compute the configuration of each physical device
  - E.g., Forwarding tables, ACLs,…

• Operate without communication guarantees

• Operate within given network-level protocol

Only people who love complexity would find this a reasonable request
Programming Analogy

• What if programmers had to:
  - Specify where each bit was stored
  - Explicitly deal with all internal communication errors
  - Within a programming language with limited expressability

• Programmers would redefine problem:
  - Define a higher level abstraction for memory
  - Build on reliable communication abstractions
  - Use a more general language

• Abstractions divide problem into tractable pieces
  - And make programmer’s task easier
From Requirements to Abstractions

1. Operate without communication guarantees
   Need an abstraction for **distributed state**

2. Compute the configuration of each physical device
   Need an abstraction that **simplifies configuration**

3. Operate within given network-level protocol
   Need an abstraction for general **forwarding model**

*Once these abstractions are in place, control mechanism has a much easier job!*
• SDN is defined *precisely* by these three abstractions
  - Distribution, forwarding, configuration

• SDN not just a random good idea…
  - Fundamental validity and general applicability

• SDN may help us *finally* create a discipline
  - Abstractions enable reasoning about system behavior
  - Provides environment where formalism can take hold….  

• OK, but what are these abstractions?
1. Distributed State Abstraction

- Shield control mechanisms from state distribution
  - While allowing access to this state

- Natural abstraction: *global network view*
  - Annotated network graph provided through an API

- Implemented with “Network Operating System”

- Control mechanism is now program using API
  - No longer a distributed protocol, now just a graph algorithm
  - E.g. Use Dijkstra rather than Bellman-Ford
Software Defined Network (SDN)

Control Program

Global Network View

Distributed algorithm running between neighbors

Network OS

e.g. routing, access control
Major Change in Paradigm

- No longer designing distributed control protocols
  - Design one distributed system (NOS)
  - Use for all control functions

- Now just defining a centralized control function

Configuration = Function(view)

- If you understand this, raise your hand.
2. Specification Abstraction

• Control program should express desired behavior

• It should not be responsible for implementing that behavior on physical network infrastructure

• Natural abstraction: simplified model of network
  - Simple model with only enough detail to specify goals

• Requires a new shared control layer:
  - Map abstract configuration to physical configuration

• This is “network virtualization”
Simple Example: Access Control

What
Abstract Network Model

How
Global Network View
Software Defined Network: Take 2

Abstract Network Model

Network Virtualization

Global Network View

Network OS
What Does This Picture Mean?

• Write a simple program to configure a simple model
  - Configuration merely a way to specify what you want

• Examples
  - ACLs: who can talk to who
  - Isolation: who can hear my broadcasts
  - Routing: only specify routing to the degree you care
    • Some flows over satellite, others over landline
  - TE: specify in terms of quality of service, not routes

• Virtualization layer “compiles” these requirements
  - Produces suitable configuration of actual network devices

• NOS then transmits these settings to physical boxes
Software Defined Network: Take 2

- Specifies behavior
- Compiles to topology
- Transmits to switches

Control Program
Abstract Network Model
Network Virtualization
Global Network View
Network OS

Specifies behavior:
- Control Program

Compiles to topology:
- Abstract Network Model
- Network Virtualization
- Global Network View

Transmits to switches:
- Network OS

Diagram of network topology with connecting switches and control program components.
Two Examples Uses

• Scale-out router:
  - Abstract view is single router
  - Physical network is collection of interconnected switches
  - Allows routers to “scale out, not up”
  - Use standard routing protocols on top

• Multi-tenant networks:
  - Each tenant has control over their “private” network
  - Network virtualization layer compiles all of these individual control requests into a single physical configuration

• Hard to do without SDN, easy (in principle) with SDN
3. Forwarding Abstraction

• Switches have two “brains”
  - Management CPU (smart but slow)
  - Forwarding ASIC (fast but dumb)

• Need a forwarding abstraction for both
  - CPU abstraction can be almost anything

• ASIC abstraction is much more subtle: OpenFlow

• OpenFlow:
  - Control switch by inserting <header;action> entries
  - Essentially gives NOS remote access to forwarding table
  - Instantiated in OpenvSwitch
Does SDN Work?

• Is it scalable? Yes
• Is it less responsive? No
• Does it create a single point of failure? No
• Is it inherently less secure? No
• Is it incrementally deployable? Yes
SDN: Clean Separation of Concerns

• Control prgm: specify behavior on abstract model
  - Driven by Operator Requirements

• Net Virt’n: map abstract model to global view
  - Driven by Specification Abstraction

• NOS: map global view to physical switches
  - API: driven by Distributed State Abstraction
  - Switch/fabric interface: driven by Forwarding Abstraction
We Have Achieved Modularity!

• Modularity enables independent innovation
  - Gives rise to a thriving ecosystem

• Innovation is the true value proposition of SDN
  - SDN doesn’t allow you to do the impossible
  - It just allows you to do the possible much more easily

• *This is why SDN is the future of networking*...