Network Management and Software-Defined Networking (SDN)

EE122 Fall 2012

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

This Week

- No sections
- No lecture on Thursday
- I’m not holding office hours tonight, or on Thursday
  - I just held office hours, so you missed your chance!
- Project 3 part 2 is out today
  - Panda will now take questions

Coming Soon: EE122 Top Gun

Who can design the best routing algorithm?

- We will use routing framework from project 2
  - You design a routing algorithm
  - We come up with test cases
- Whoever delivers the most packets wins
- Prizes:
  - First prize: dinner with the TAs at a fine restaurant
  - Second prize: chocolate from my private collection
  - **No extra credit, just bragging rights**

You can play with the code now…

- Code and rules are online….
- Quick preview:
  - Do not ask TAs anything, ever, period.
  - Do not post on Piazza about your solution
  - Submit bugs by email to Panda
- After you've looked at the rules and code, email Panda if you want to participate (see rules)

Last three lectures

- Today: Introduction to SDN
  - An exercise in thinking *radically*
- Tuesday: Implications of SDN
  - An exercise in thinking *architecturally*
- Thursday: P2P and the triumph of the Internet
  - An exercise in thinking *egotistically*

Software-Defined Networking: *Caveats and Context*

Caveats

- I cofounded a startup (Nicira) that worked on SDN
  - My views may be biased
  - I have no financial interest in the outcome, just ego
- SDN is not a revolutionary technology…
  - …just a way of organizing network functionality
- But that's all the Internet architecture is….
  - The Internet architecture isn't clever, but it is deeply wise
  - We know SDN isn't clever, but we hope it is wise….

Some context before we go further…

- *Where did SDN come from?*
- *And what is the state of networking as a field?*
- **Keep context in mind as you learn about SDN…**

Where did SDN come from?

- ~2004: Research on new management paradigms
  - RCP, 4D [Princeton, CMU,….]
  - SANE, Ethane [Stanford/Berkeley]
  - Industrial efforts with similar flavor (not published)
- 2008: Software-Defined Networking (SDN)
  - NOX Network Operating System [Nicira]
  - OpenFlow switch interface [Stanford/Nicira]
- 2011: Open Networking Foundation (72 members)
  - **Board**: Google, Yahoo, Verizon, DT, Msoft, F'book, NTT
  - **Members**: Cisco, Juniper, HP, Dell, Broadcom, IBM,…..

Where did SDN really come from?

Martín Casado (from a Wired profile)

"Martin Casado is fucking amazing," says Scott Shenker, the physics PhD, UC Berkeley computer science professor, and former Xerox PARC researcher who has worked closely with Casado for the past several years on the networking problems Nicira is trying to solve. "I've known a lot of smart people in my life, and on any dimension you care to mention, he's off the scale."

Current Status of SDN

- SDN widely accepted as "**future of networking**"
  - ~1000 engineers at latest Open Networking Summit
  - Commercialized, in production use (few places)
    - E.g., controls Google's WAN; NTT moving to deploy
  - *Much more acceptance in industry than in academia*
- An insane level of SDN hype, and backlash…
  - SDN doesn't work miracles, merely makes things easier
- But the real question is: *why the rapid adoption?*

The Field of Networking…

- CS networking now largely the study of the Internet
- Also interesting research in wireless, optical
  - Much of it is EE research into underlying technologies
  - Some wireless research (such at Katabi at MIT) broader
- This Internet research effort built a great artifact
  - Mostly unrelated to academic research which came later
- But it has failed to create an academic discipline
  - The fact that EE122 sucks is not ***my*** fault!

Building an Artifact, Not a Discipline

- Other fields in “systems”: OS, DB, etc.
  - Teach basic principles
  - Are easily managed
  - Continue to evolve
- Networking:
  - Teach big bag of protocols
  - Notoriously difficult to manage
  - Evolves very slowly
- ***Networks are much more primitive and less understood than other computer systems***

We are left with two key questions

- *Why the rapid adoption of SDN?*
  - What problem is it solving?
- *Why is networking behind other fields in CS?*
  - What is missing in the field?
- The answers are related, but will unfold slowly

Network Management

What is Network Management?

- Recall the two “planes”
- **Data plane**: forwarding packets
  - Based on local forwarding state
- **Control plane**: computing that forwarding state
  - Involves coordination with rest of system
- Broad definition of “network management”:
  - ***Everything having to do with the control plane***

Original goals for the control plane

- **Basic connectivity**: route packets to destination
  - Local state computed by routing protocols
  - Globally distributed algorithms
- **Interdomain policy**: find policy-compliant paths
  - Done by fully distributed BGP
- For long time, these were the only relevant goals!
  - What other goals are there in running a network?

Isolation

- Want multiple LANs on single physical network
- Packets on LAN don't pass through routers
  - But routers used to impose various controls (later)
- Use VLANs (virtual LANs) tags in L2 headers
  - Controls where broadcast packets go
  - Gives support for logical L2 networks
  - Routers connect these logical L2 networks
- No universal method for setting VLAN state

Access Control

- Operators want to limit access to various hosts
  - Don’t let laptops access backend database machines
- This can be imposed by routers using ACLs
  - ACL: Access control list
- Example entry in ACL: <header template; drop>

Traffic Engineering

- Want to avoid persistent overloads on links
- Choose routes to spread traffic load across links
- Two main methods:
  - Setting up MPLS tunnels
  - Adjusting weights in OSPF
- Often done with centralized computation
  - Take snapshot of topology
  - Compute appropriate MPLS/OSPF state
  - Send to network

Summarizing

- Network management has many goals
- Achieving these goals is job of the control plane…
- …which currently involves many mechanisms

Control Plane Mechanisms

- **Globally distributed:** routing algorithms
- **Manual/scripted configuration:** ACLs, VLANs
- **Centralized computation:** Traffic engineering

Bottom Line

- Many different control plane mechanisms
- Each designed from scratch for their intended goal
- Encompassing a wide variety of implementations
  - Distributed, manual, centralized,…
- **Network control plane is a complicated mess!**

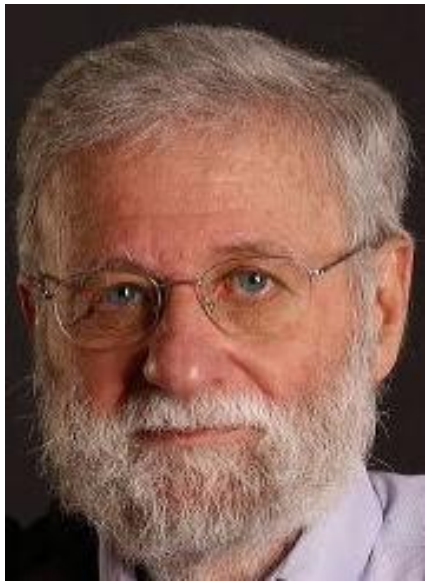
How Did We Get Into This Mess?

How Have We Managed To Survive?

- Net. admins miraculously master this complexity
  - Understand all aspects of networks
  - Must keep myriad details in mind
- 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 valuable
  - But not the same as the ability to **extract simplicity**
- Each has its role:
  - When first getting systems to work, *master complexity*
  - When making system easy to use, *extract simplicity*
- You will never succeed in extracting simplicity
  - If you don't recognize it is a different skill set than mastering complexity

What Is *My* Point?

- Networking has never made the distinction…
  - And therefore has never made the transition from mastering complexity to extracting simplicity
- Still focused on mastering complexity
  - Networking "experts" are those that know all the details
- *Extracting simplicity lays intellectual foundations*
  - This is why networking has weak foundation
  - We are ***still*** building the artifact, not the discipline

Have answered one of our questions

- The reason networking is not a discipline is because it has not sought to extract simplicity
  - Other fields, such as OS, DB, etc, have
  - Those fields are more mature
- Extracting simplicity is also how you generalize to larger, more complicated systems
  - So it has practical advantages as well….

Forcing people to make the transition

- We are really good at mastering complexity
  - And it has worked for us for decades, why change?
- How do you make people change?
  - Make them cry!
- A personal story about algebra and complexity
  - School problems:

    $3x + 2y = 8$ $\quad$ $x + y = 3$
  - My father's problems:

    $327x + 26y = 8757$ $\quad$ $45x + 57y = 7776$

Making Network Operators Cry…

Step 1: Large datacenters

- 100,000s machines; 10,000s switches
- This is pushing the limits of what we can handle….

Step 2: Multiple tenancy

- Large datacenters can host many customers
- Each customer gets their own logical network
  - Customer should be able to set policies on this network
  - ACLs, VLANs, etc.
- If there are 1000 customers, that adds 3 oom
  - Where oom = orders of magnitude
- This goes *way* beyond what we can handle

Network Operators Are Now Weeping…

- They have been beaten by complexity
- The era of ad hoc control mechanisms is over
- We need a simpler, more systematic design
- ***So how do you "extract simplicity"?***

An Example Transition: Programming

- Machine languages: no abstractions
  - Had to deal with low-level details
  - 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 About Networking Abstractions?

- Consider the data and control planes separately
- Different tasks, so naturally different abstractions

Abstractions for Data Plane: Layers

Applications

…built on…

Reliable (or unreliable) transport

…built on…

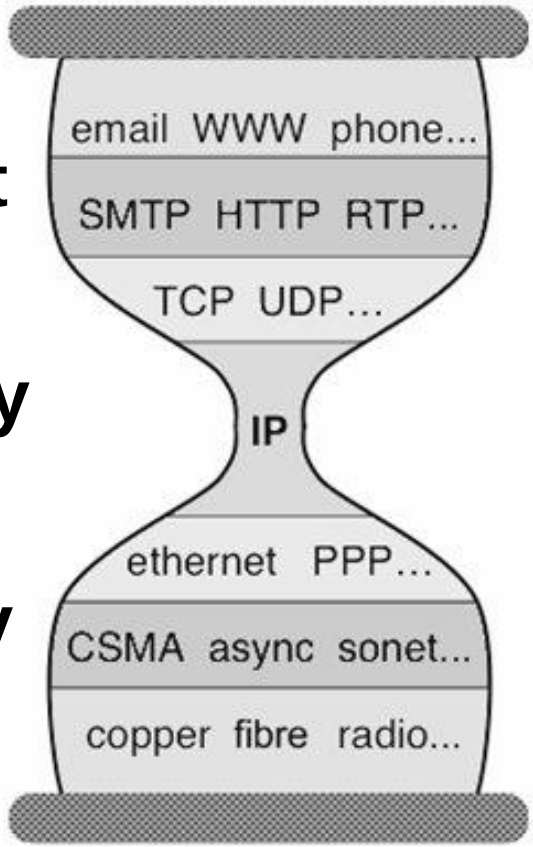
Best-effort global packet delivery

…built on…

Best-effort local packet delivery

…built on…

Physical transfer of bits



The Importance of Layering

- Decomposed delivery into basic components
- Independent, compatible innovation at each layer
  - Clean “separation of concerns”
  - Leaving each layer to solve a tractable problem
- Responsible for the success of the Internet!
  - Rich ecosystem of independent innovation

Control Plane Abstractions

(Too) Many Control Plane Mechanisms

- Variety of goals, no modularity:
  - **Routing:** distributed routing algorithms
  - **Isolation**: ACLs, VLANs, Firewalls,…
  - **Traffic engineering**: adjusting weights, MPLS,…

- **Control Plane: mechanism without abstraction**
  - ***Too many mechanisms, not enough functionality***

Finding Control Plane Abstractions

How do you find abstractions?

- You first decompose the problem….
- …and define abstractions for each subproblem
- **So what is the control plane problem?**

Task: Compute forwarding state:

- Consistent with low-level hardware/software
  - Which might depend on particular vendor
- Based on entire network topology
  - Because many control decisions depend on topology
- For all routers/switches in network
  - Every router/switch needs forwarding state

Our current approach

- Design one-off mechanisms that solve all three
- A sign of how much we love complexity
- No other field would deal with such a problem!
- They would define abstractions for each subtask
- …and so should we!

Separate Concerns with Abstractions

1. Be compatible with low-level hardware/software
   Need an abstraction for general **forwarding model**

2. Make decisions based on entire network
   Need an abstraction for **network state**

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

Abs#1: Forwarding Abstraction

- Express intent independent of implementation
  - Don’t want to deal with proprietary HW and SW
- OpenFlow is current proposal for forwarding
  - Standardized interface to switch
  - Configuration in terms of flow entries: <header, action>
- Design details concern exact nature of:
  - Header matching
  - Allowed actions

Two Important Facets to OpenFlow

- Switches accept external control messages
  - Not closed, proprietary boxes
- Standardized flow entry format
  - So switches are interchangable

Abs#2: Network State Abstraction

- Abstract away various distributed mechanisms
- Abstraction: **global network view**
  - Annotated network graph provided through an API
- Implementation: “Network Operating System”
  - Runs on servers in network (“controllers”)
  - Replicated for reliability
- Information flows both ways
  - Information *from* routers/switches to form “view”
  - Configurations *to* routers/switches to control forwarding

Network Operating System

- Think of it as a centralized link-state algorithm
- Switches send connectivity info to controller
- Controller computes forwarding state
  - Some control program that uses the topology as input
- Controller sends forwarding state to switches
- Controller is replicated for resilience
  - System is only “logically centralized”

Software-Defined Networks (SDN)

routing, access control, etc.

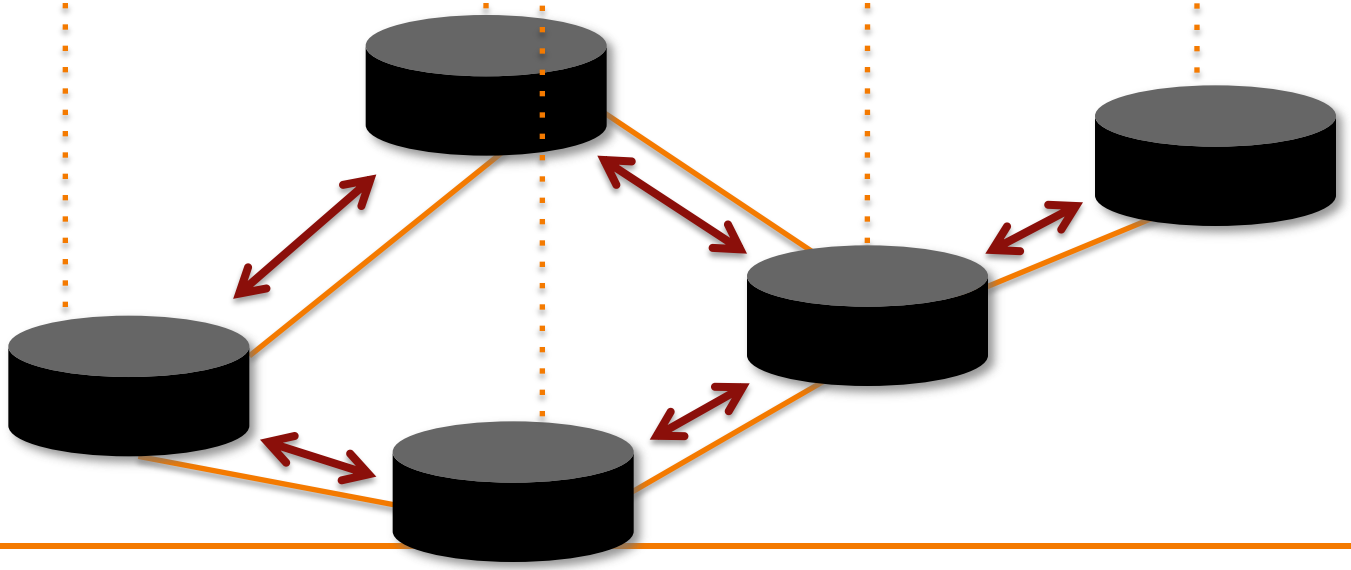
Control Program

Distributed algorithm running between neighbors

Global Network View

Complicated task-specific distributed algorithm

Network OS



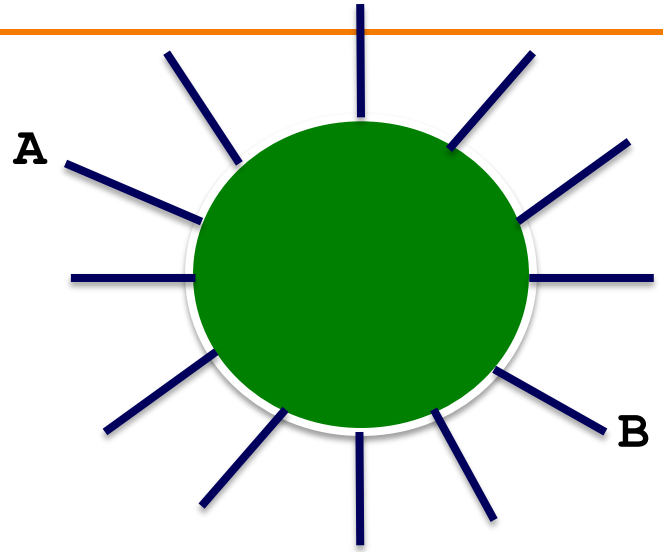
Major Change in Paradigm

- Control program: **Configuration = Function(view)**
- Control mechanism now program using NOS API
- Not a distributed protocol, just a graph algorithm

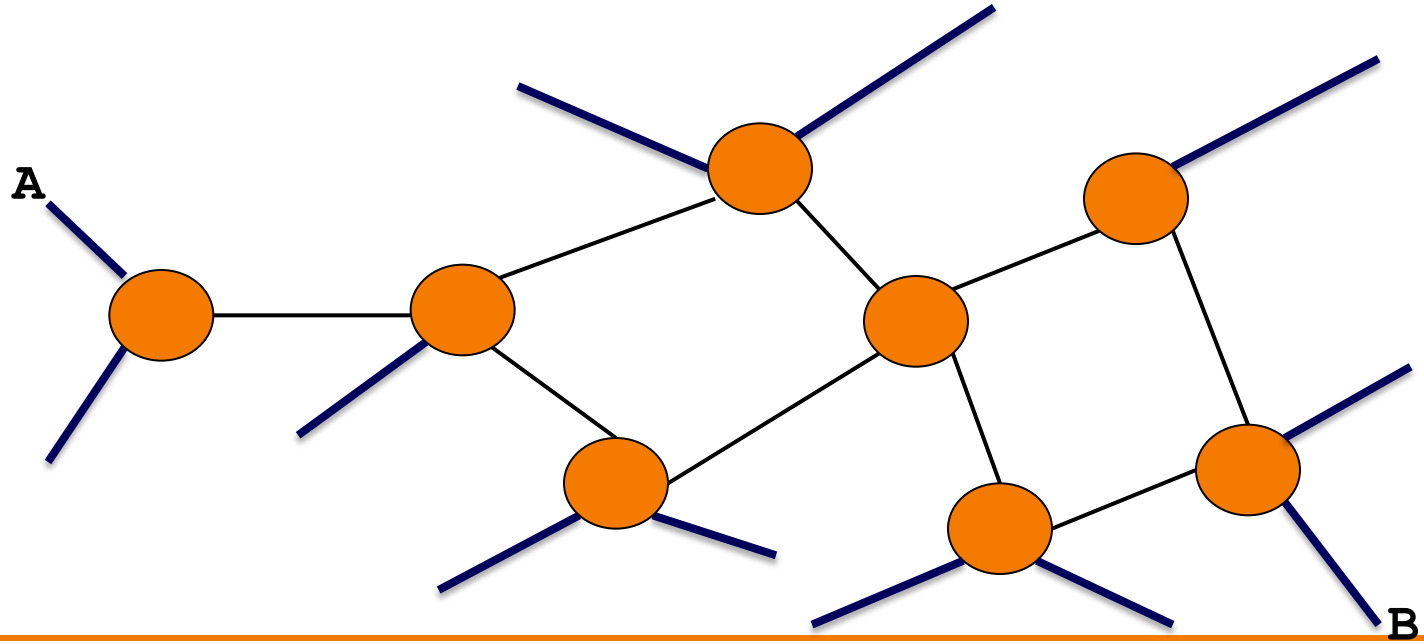
Abs#3: Specification Abstraction

- Control mechanism expresses desired behavior
  - Whether it be isolation, access control, or QoS
- It should not be responsible for ***implementing*** that behavior on physical network infrastructure
  - Requires configuring the forwarding tables in each switch
- Proposed abstraction: **abstract view** of network
  - Abstract view models only enough detail to *specify goals*
  - Will depend on task semantics

Simple Example: Access Control

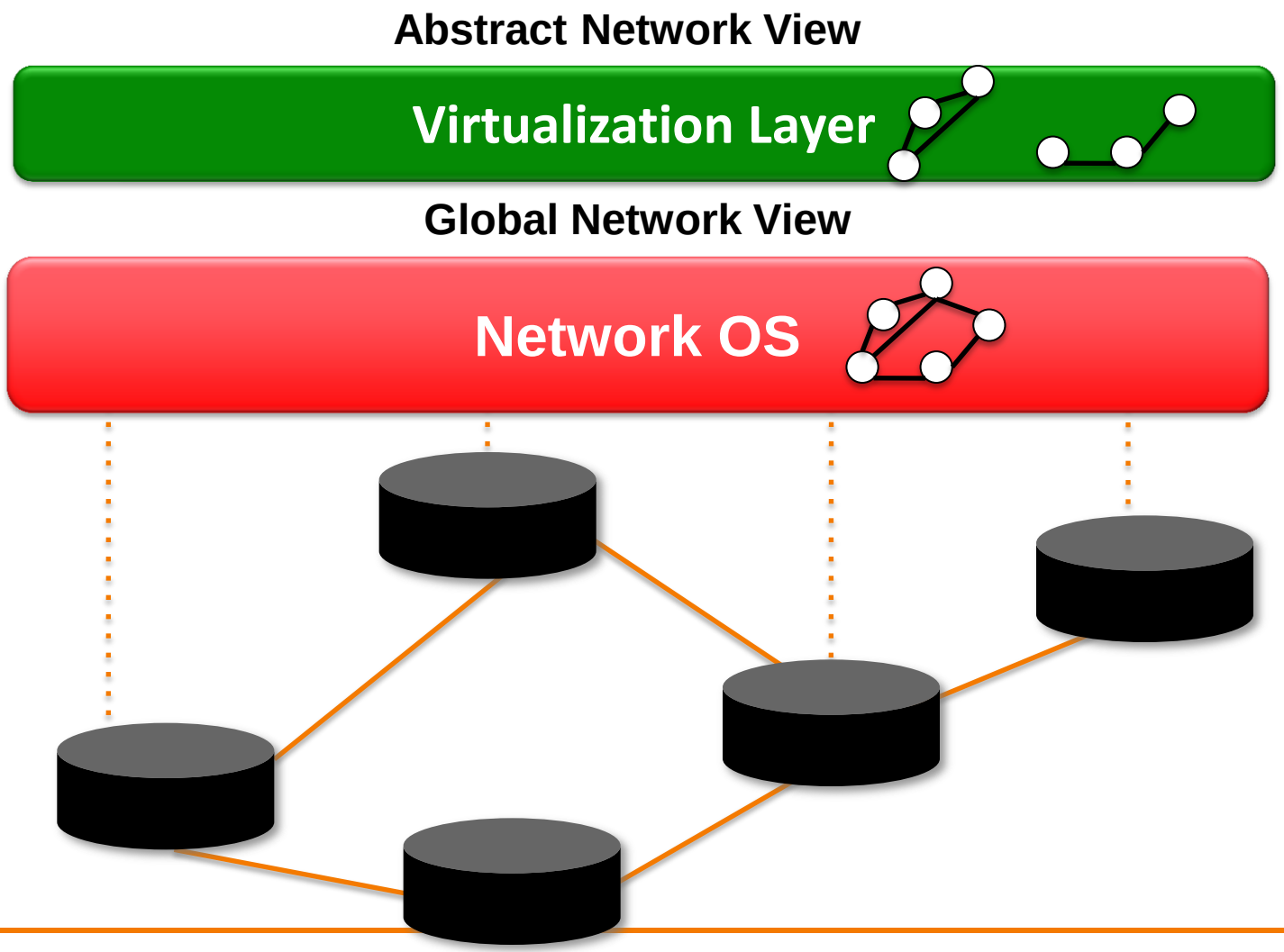


Abstract Network View



Global Network View

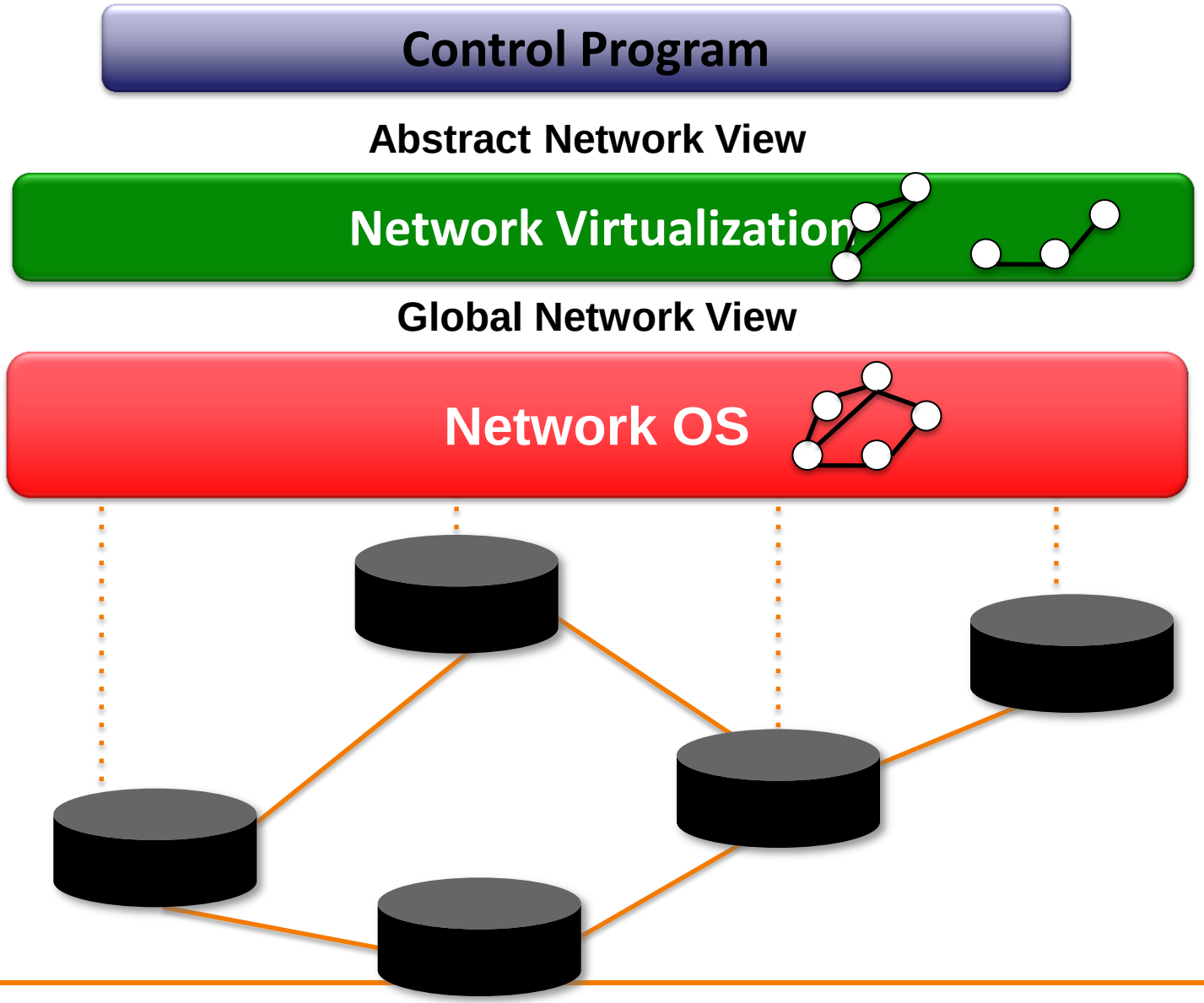
Software Defined Network



Clean Separation of Concerns

- **Control program:** express goals on abstract view
  - Driven by **Operator Requirements**
- **VirtualizationLayer:** abstract view ←→ global view
  - Driven by **Specification Abstraction** for particular task
- **NOS:** global view ←→ physical switches
  - API: driven by **Network State Abstraction**
  - Switch interface: driven by **Forwarding Abstraction**

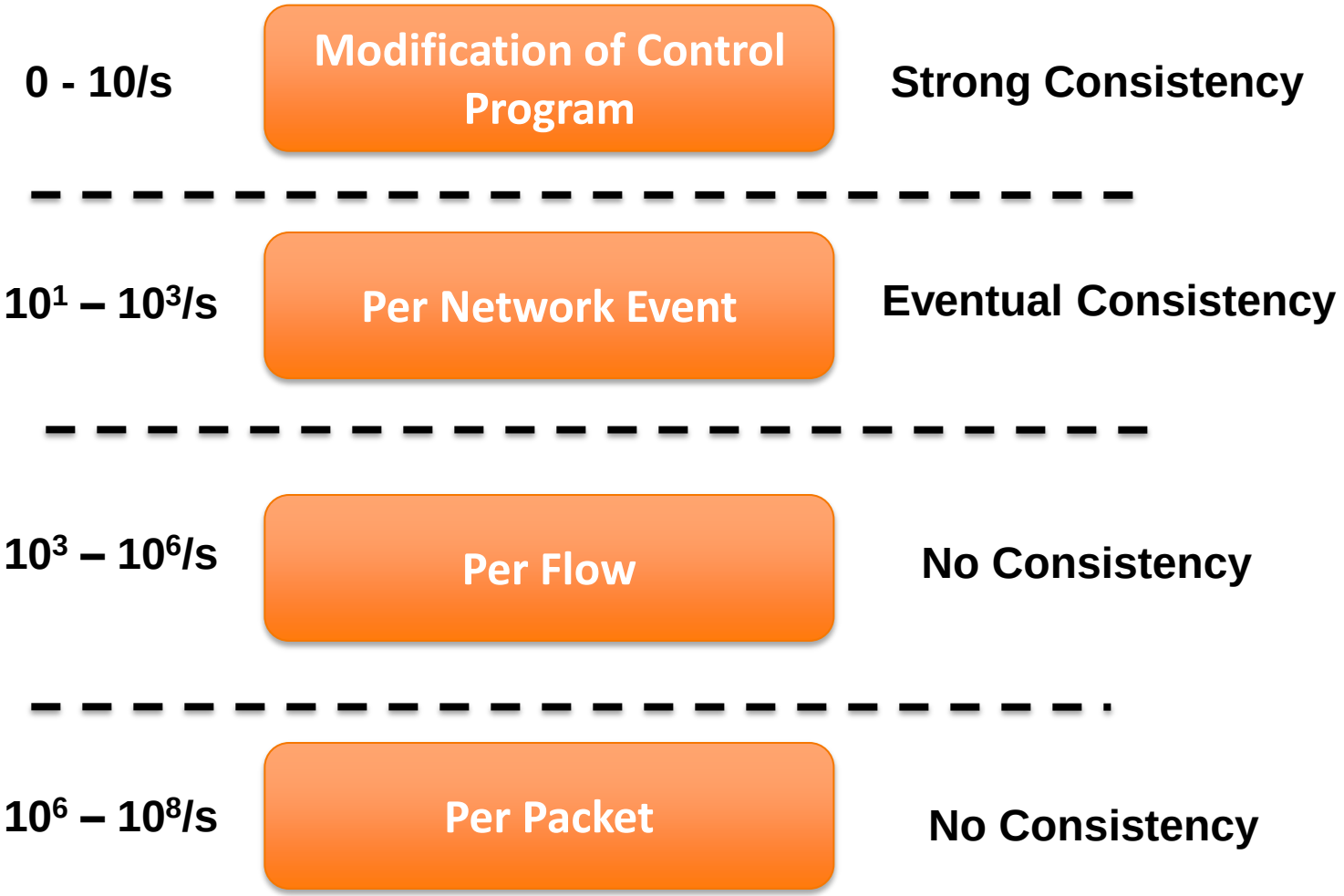
SDN: *Layers* for the Control Plane



Abstractions Don't Remove Complexity

- NOS, Virtualization are complicated pieces of code
- SDN merely localizes the complexity:
  - Simplifies interface for control program (user-specific)
  - Pushes complexity into ***reusable*** code (SDN platform)
- This is the big payoff of SDN: modularity!
  - The core distribution mechanisms can be reused
  - Control programs only deal with their specific function
- Note that SDN separates control and data planes
  - SDN platform does control plane, switches do data plane

Why Does SDN Scale?



What This Really Means

Routing

- Look at graph of network
- Compute routes
- Give to SDN platform, which passes on to switches

Access Control

- Control program decides who can talk to who
- Pass this information to SDN platform
- Appropriate ACL flow entries are added to network
  - In the right places (based on the topology)

Common Questions about SDN

Common Questions about SDN?

- Is SDN less scalable, secure, resilient,…?
- Is SDN incrementally deployable?
- Can SDN be extended to the WAN?
- Can you troubleshoot an SDN network?
- Is OpenFlow the right fwding abstraction?

Common Questions about SDN?

- Is SDN less scalable, secure, resilient,…? **No**
- Is SDN incrementally deployable? **Yes**
- Can SDN be extended to the WAN? **Yes**
- Can you troubleshoot an SDN network? **Yes**
- Is OpenFlow the right fwding abstraction? **No**

Next Time

- Extending SDN
- Thinking architecturally

Have a good holiday!

- Get some rest….