Software Defined Networks

Chen Avin

Based on Lecture notes from Scott Shenker, Nick Mckown and Google

How do we build large-scale software systems?

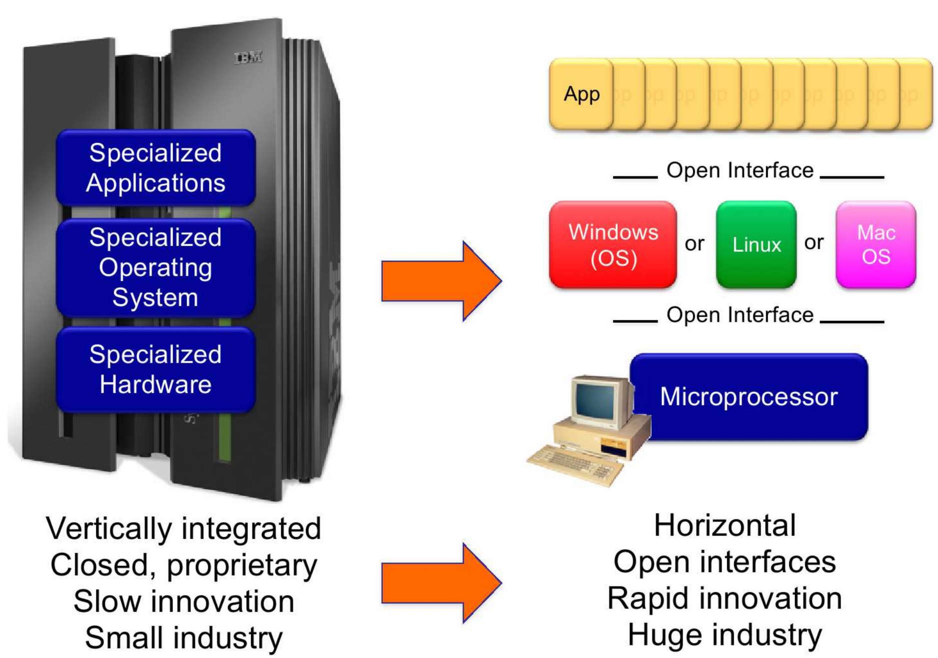
Liskov: "The Power of Abstractions"

"Modularity based on abstraction is the way things get done"

Abstractions ➔ Interfaces ➔ Modularity

- Modularity provides:
  - Code reuse
  - Flexibility of implementation
  - Conceptual separation of concerns

Vertically integrated
Closed, proprietary
Slow innovation
Small industry

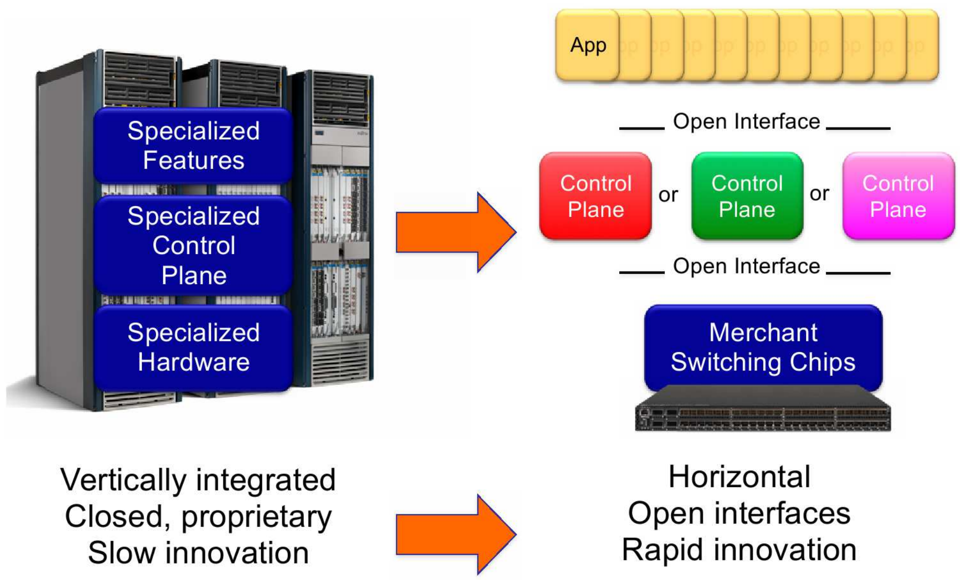

Specialized Applications
Specialized Operating System
Specialized Hardware
App
Open Interface
Windows (OS)
or
Linux
or
Mac OS
Open Interface
Microprocessor
Vertically integrated
Closed, proprietary
Slow innovation
Small industry
Horizontal
Open interfaces
Rapid innovation
Huge industry

How do we find abstractions?

Abstractions ≈ Problem Decomposition

- Decompose problem into basic components (tasks)
- Define an abstraction for each component
- Implementation of abstraction can focus on one task
- If tasks still too hard to implement, return to step 1

Vertically integrated
Closed, proprietary
Slow innovation


Specialized Features
Specialized Control Plane
Specialized Hardware
App
Open Interface
Control Plane
or
Control Plane
or
Control Plane
Open Interface
Merchant Switching Chips
Vertically integrated
Closed, proprietary
Slow innovation
Horizontal
Open interfaces
Rapid innovation

What abstractions have been applied to networking?

The Two Networking "Planes"

- **Data plane**: process packets with local fwding state
  - Fwding state + packet header ➔ forwarding decision

- **Control plane**: compute the forwarding state
  - Distributed protocols
  - Manual configuration (and scripting)
  - Centralized computation

- These different planes require different abstractions

Data Plane Abstractions: Layers

Applications

...built on...

Reliable (or unreliable) transport

...built on...

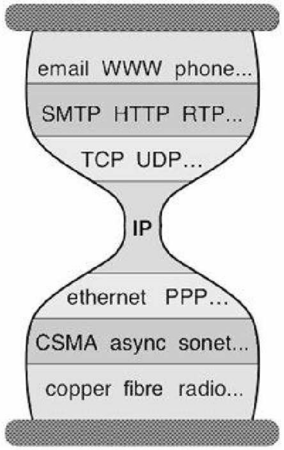
Best-effort global packet delivery

...built on...

Best-effort local packet delivery

...built on...

Local physical transfer of bits



Control Plane Abstractions

?

(Too) Many Control Plane Mechanisms

- Variety of goals:
  - **Routing:** distributed routing algorithms
  - **Isolation**: ACLs, VLANs, Firewalls,…
  - **Traffic engineering**: adjusting weights, MPLS,…

- No modularity, limited functionality

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

This is crazy!

Programming Analogy

- What if you were told to write a program that must…
  - Be aware of the hardware you were running on
  - Specify where each bit was stored
- Programmer would immediately define abstractions:
  - Machine-independent interface
  - Virtual memory interface
- Programmers use abstractions to separate concerns
  - Network designers should too!

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

Forwarding Abstraction

- Express intent independent of implementation
  - Hardware (e.g., ASIC structure and capabilities)
  - Software (e.g., vendor-independent)
- 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

Network State Abstraction

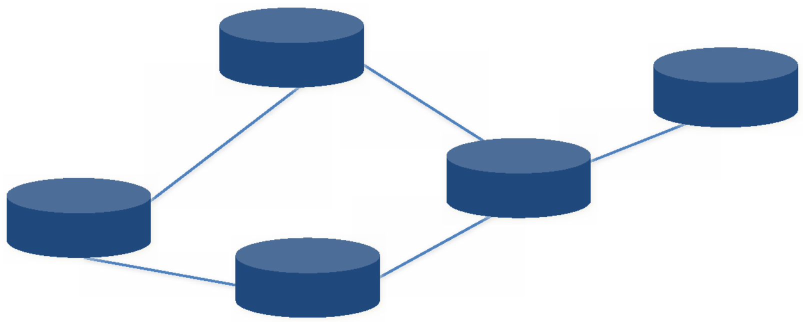
- Abstract away complicated distributed mechanisms

- Abstraction: **global network view**
  - Annotated network graph provided through an API
  - Network elements can be controlled via this API

- Implementation: “Network Operating System”
  - Runs on servers in network (replicated for reliability)

- Information flows both ways
  - Information *from* routers/switches to form “view”
  - Configurations *to* routers/switches to control forwarding

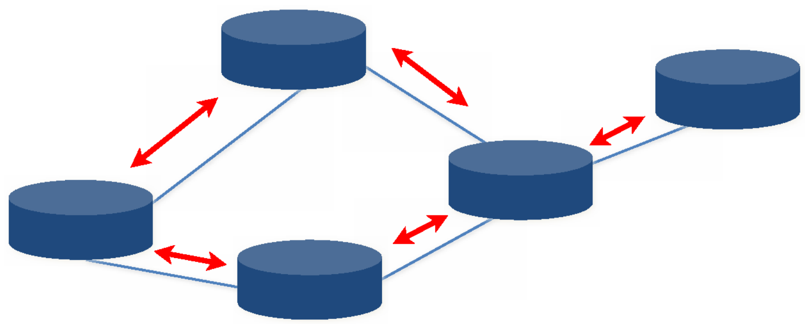
Network of Switches and/or Routers



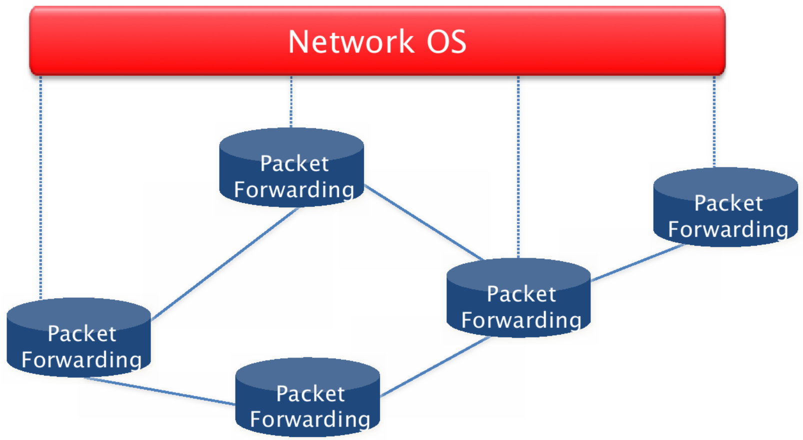
Traditional Control Mechanism

Distributed algorithm running between neighbors

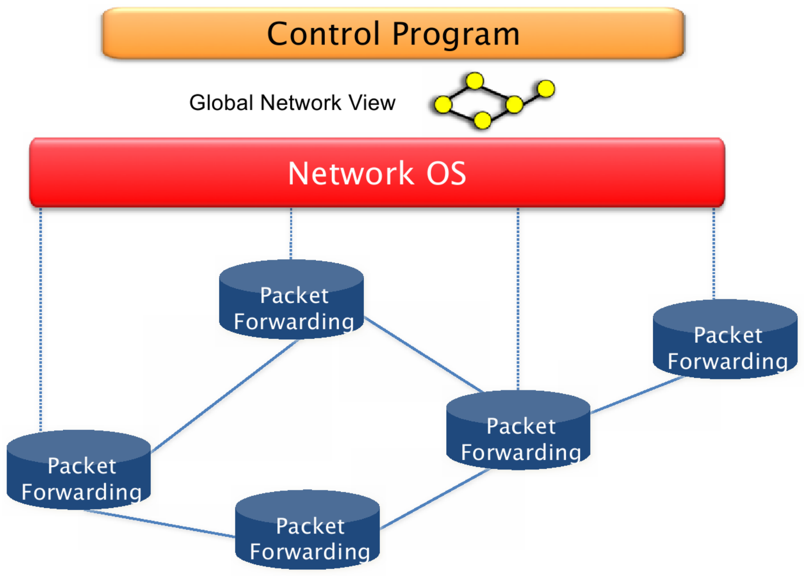
Complicated task-specific distributed algorithm



Software Defined Network (SDN)



Software Defined Network (SDN)



Major Change in Paradigm

- Control program: **Configuration = Function(view)**

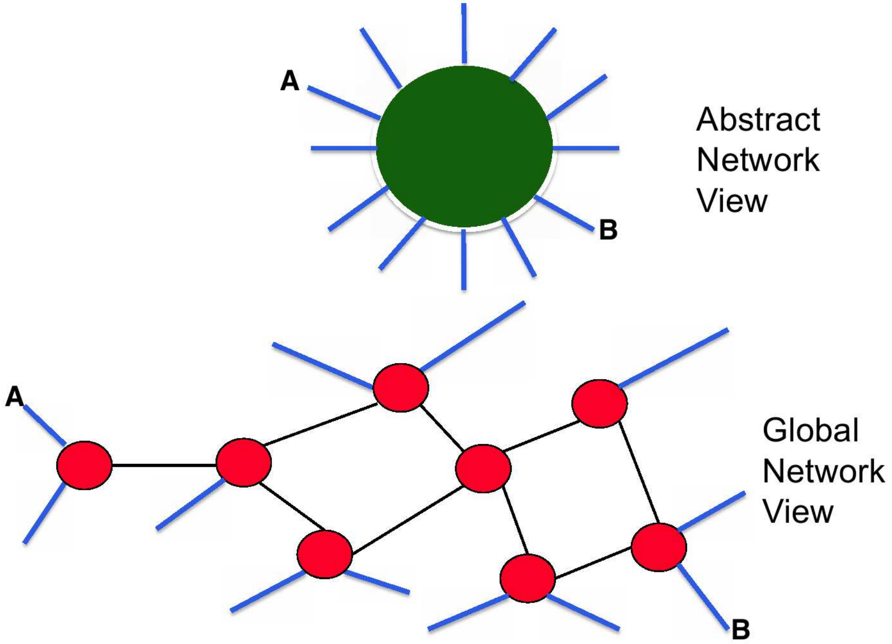
- Control mechanism is now program using NOS API
  - Not a distributed protocol, now just a graph algorithm

- Easier to write, maintain, verify, reason about, …

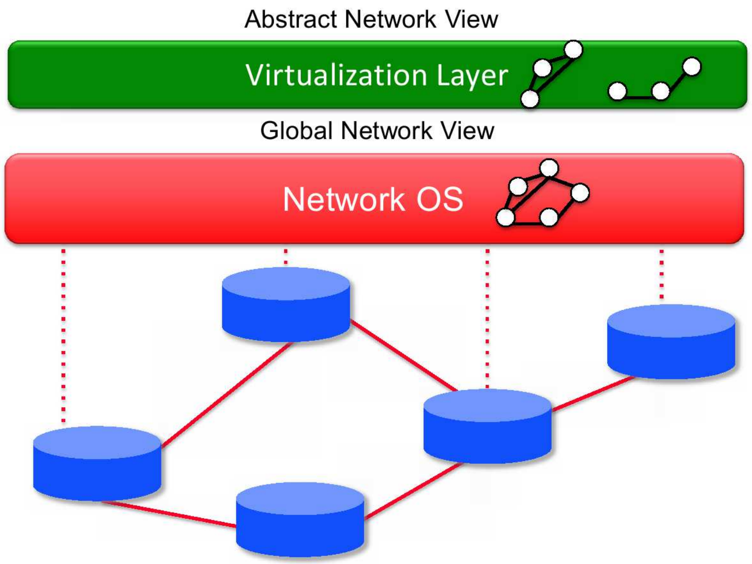
Specification Abstraction

- Control mechanism must ***express*** 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
- Analogy: programming languages and compilers

Simple Example: Access Control



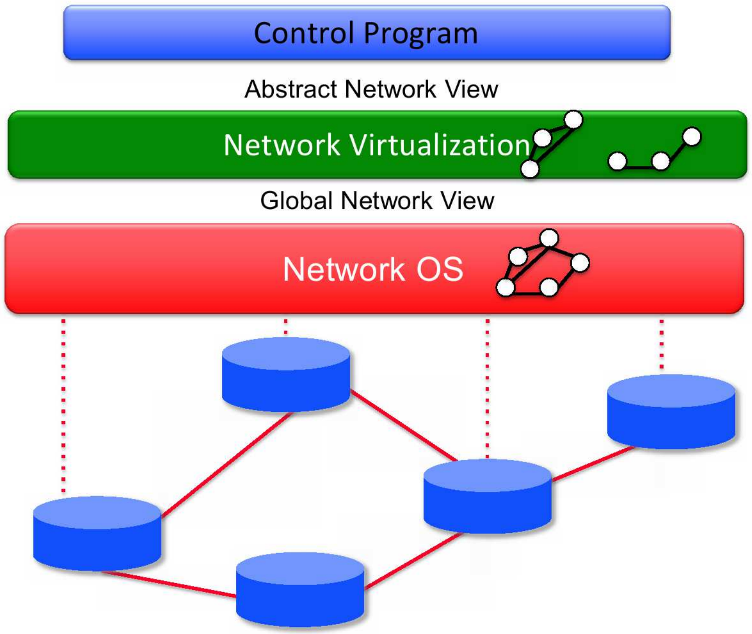
Software Defined Network



Clean Separation of Concerns

- **Control program:** express goals on abstract view
  - Driven by **Operator Requirements**
- **Virtualization Layer:** 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 Eliminate Complexity

- Every component of system is tractable
  - NOS, Virtualization are still complicated pieces of code

- SDN main achievements:
  - Simplifies interface for control program (user-specific)
  - Pushes complexity into reusable code (SDN platform)

- Just like compilers….

What Should I Remember About SDN?

Four Crucial Points

- SDN is merely set of abstractions for control plane
  - Not a specific set of mechanisms
  - OpenFlow is least interesting aspect of SDN, technically
- SDN involves computing a function….
  - NOS handles distribution of state
- …on an abstract network
  - Can ignore actual physical infrastructure
- Network virtualization is the “killer app”
  - Already virtualized compute, storage; network is next

SDN Vision: Networks Become "Normal"

- Hardware: Cheap, interchangeable, Moore's Law
- Software: Frequent releases, decoupled from HW
- Functionality: Mostly driven by SW
  - Edge (software switch)
  - Control program
- Solid intellectual foundations

Simple example

OSPF

- RFC 2328: **245 pages**

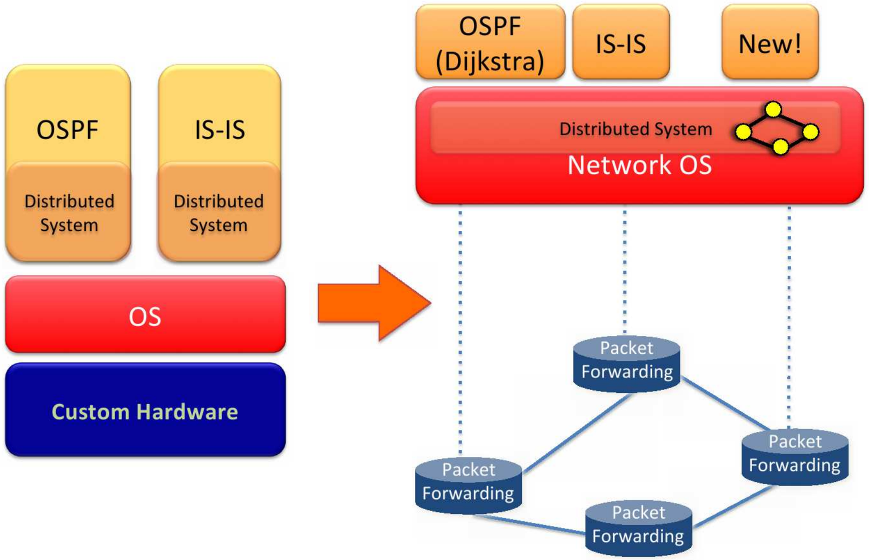
Distributed System

- Builds consistent, up-to-date map of the network: **101 pages**

Dijkstra's Algorithm

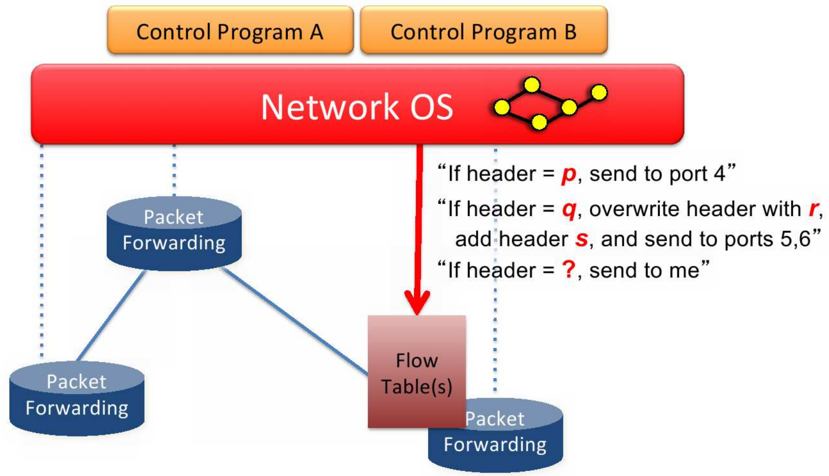
- Operates on map: **4 pages**

Example



OSPF
IS-IS
Distributed System
Distributed System
OS
Custom Hardware
OSPF (Dijkstra)
IS-IS
New!
Distributed System
Network OS
Packet Forwarding
Packet Forwarding
Packet Forwarding
Packet Forwarding

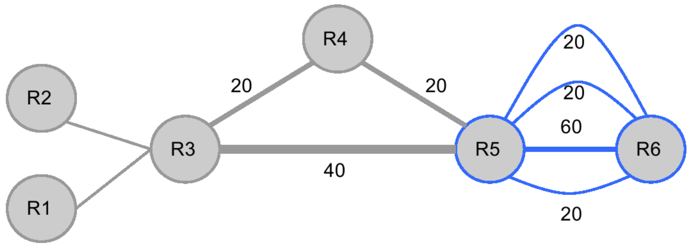
OpenFlow Forwarding Abstraction



OpenFlow @ Google

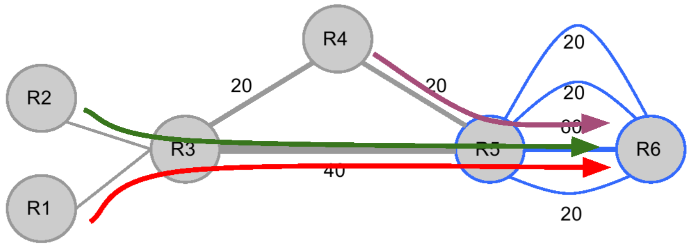
Convergence After Failure

- Flows: R1->R6: 20; R2->R6: 20; R4->R6: 20



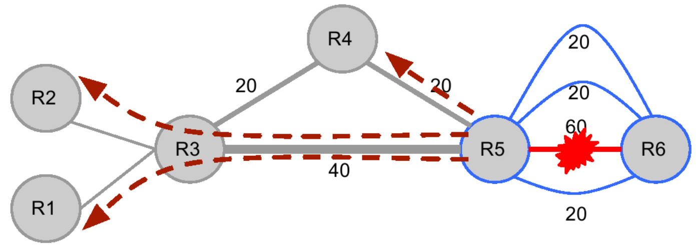
Convergence After Failure

- Flows: R1->R6: 20; R2->R6: 20; R4->R6: 20



Convergence After Failure

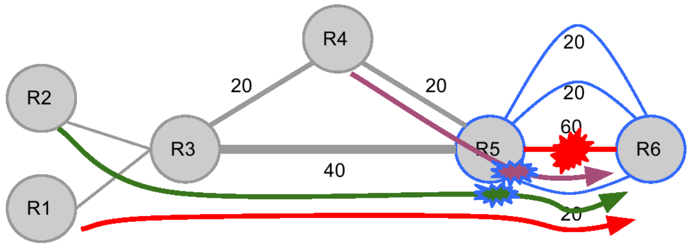
- Flows: R1->R6: 20; R2->R6: 20; R4->R6: 20



- R5-R6 link fails
  - R1, R2, R4 *autonomously* try for next best path

Convergence After Failure

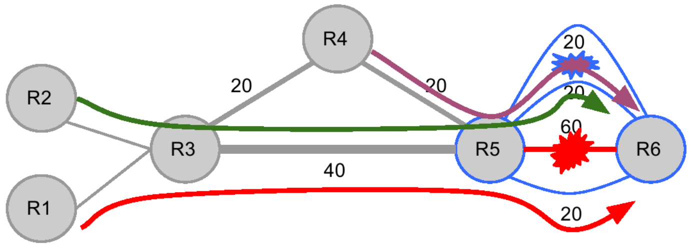
- Flows: R1->R6: 20; R2->R6: 20; R4->R6: 20



- R5-R6 link fails
  - R1, R2, R4 *autonomously* try for next best path
  - R1 wins, R2, R4 retry for next best path

Convergence After Failure

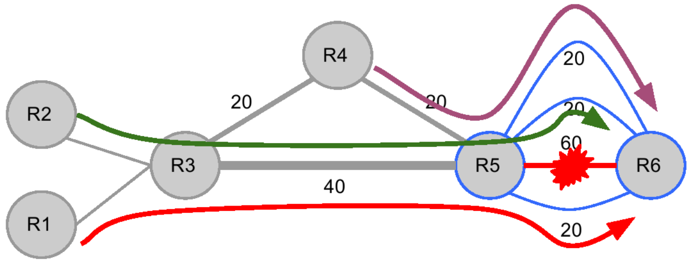
- Flows: R1->R6: 20; R2->R6: 20; R4->R6: 20



- R5-R6 link fails
  - R1, R2, R4 *autonomously* try for next best path
  - R1 wins, R2, R4 retry for next best path
  - R2 wins this round, R4 retries again

Convergence After Failure

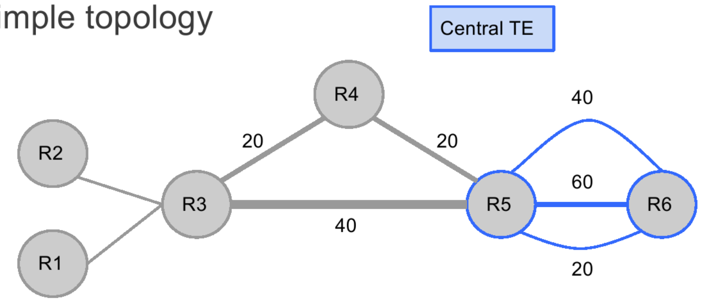
- Flows: R1->R6: 20; R2->R6: 20; R4->R6: 20



- R5-R6 link fails
  - R1, R2, R4 *autonomously* try for next best path
  - R1 wins, R2, R4 retry for next best path
  - R2 wins this round, R4 retries again
  - R4 finally gets third best path

Centralized Traffic Engineering

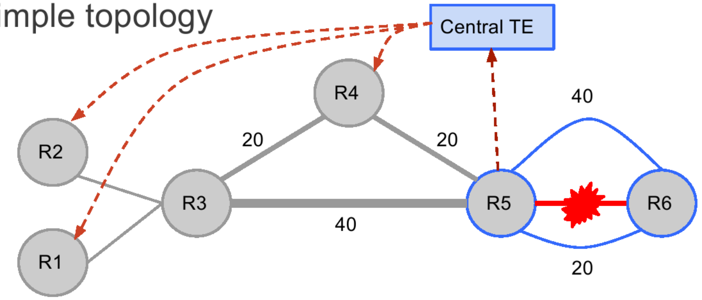
- Simple topology



- Flows:
  - R1->R6: 20; R2->R6: 20; R4->R6: 20

Centralized Traffic Engineering Google

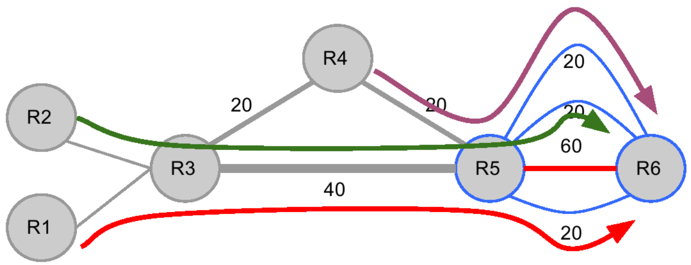
- Simple topology



- Flows:
  - R1->R6: 20; R2->R6: 20; R4->R6: 20
- R5-R6 fails
  - R5 informs TE, which programs routers in one shot

Centralized Traffic Engineering Google

- Simple topology



- Flows:
  - R1->R6: 20; R2->R6: 20; R4->R6: 20
- R5-R6 link fails
  - R5 informs TE, which programs routers in one shot
  - Leads to faster realization of target optimum

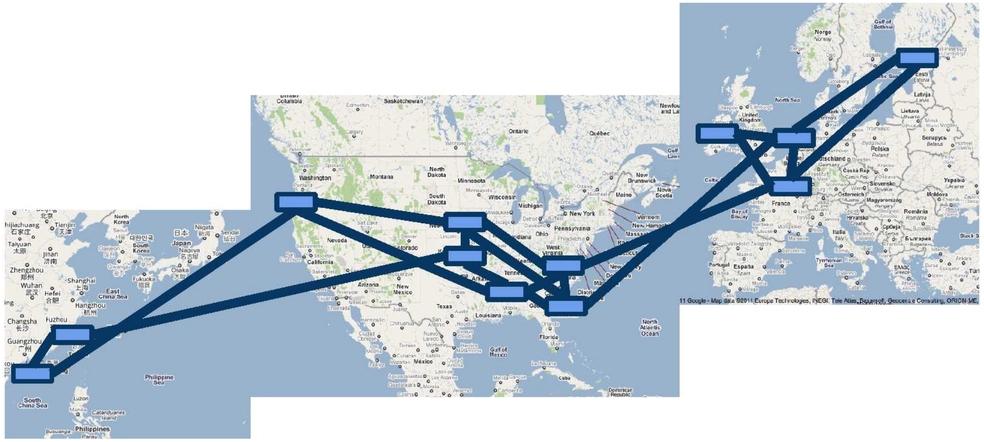
Advantages of Centralized TE

- Better network utilization with global picture
- Converges faster to target optimum on failure
- Allows more control and specifying intent
  - Deterministic behavior simplifies planning vs. overprovisioning for worst case variability
- Can mirror production event streams for testing
  - Supports innovation and robust SW development
- Controller uses modern server hardware
  - 50x (!) better performance

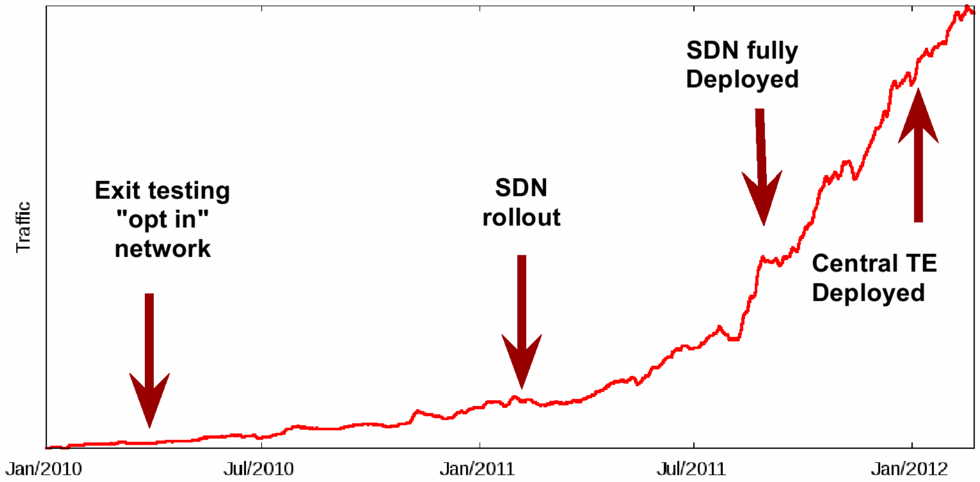
Google's WAN

- Two backbones
  - Internet facing (user traffic)
  - Datacenter traffic (internal)
- Widely varying requirements: loss sensitivity, availability, topology, etc.
- Widely varying traffic characteristics: smooth/diurnal vs. bursty/bulk
- Therefore: built two separate logical networks
  - I-Scale (bulletproof)
  - G-Scale (possible to experiment)

Google's OpenFlow WAN



G-Scale WAN Usage



How SDN will shape networking

Nick McKeown
Stanford University

With: **Martín Casado**, Teemu Koponen, Scott Shenker
… and many others

With thanks to: NSF, GPO, Stanford Clean Slate Program,
Cisco, DoCoMo, DT, Ericsson, Google, HP, Huawei, NEC, Xilinx

A Gentle Introduction to

Software Defined Networks

Scott Shenker

*with **Martín Casado**, Teemu Koponen, Nick McKeown*

(and many others….)