

# Developing SDN Applications

Rev. 14.21

## HP ExpertOne LEARNER GUIDE



© Copyright 2014 Hewlett-Packard Development Company, L.P.

The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

This is an HP copyrighted work that may not be reproduced without the written permission of HP. You may not use these materials to deliver training to any person outside of your organization without the written permission of HP.

**Developing SDN Applications**

Learner Guide

Rev. 14.21

# Contents

## Module 1: Application History

Overview .....	1-1
Welcome to HP's Developing SDN Applications Course .....	1-2
SDN Backstory .....	1-3
A Brief History of Networking .....	1-4
A Brief History of Networking .....	1-5
A Brief History of Networking (cont.) .....	1-6
Networking Today .....	1-7
Network Change: ForCES .....	1-8
Network Change: Clean Slate .....	1-9
First SDN Application: Ethane .....	1-10
OpenFlow is Born .....	1-11
OpenFlow Interest: Initially .....	1-12
Networking is a Closed System .....	1-13
Why is a Closed System a Bad Thing? .....	1-14
What Caused Interest in SDN to Increase? .....	1-15
MAC Address Table Issue .....	1-16
Spanning Tree Issue .....	1-17
VLAN Issue .....	1-18
Traffic Engineering Issue .....	1-19
Agility and Automation Issue .....	1-20
Cost Issue .....	1-21
OpenFlow Interest: after Data Center .....	1-22
Using Management to Address DC Issues .....	1-23
Orchestration Solutions .....	1-24
VM Plug-in Solutions .....	1-25
RADIUS Solutions .....	1-26
Using Tunnels to Address DC Issues .....	1-27
Tunnels: VXLAN .....	1-28
Tunnels: NVGRE .....	1-29
Tunnels: Stateless Transport Tunneling .....	1-30
Using Protocols to Address DC Issues .....	1-31
Protocols: Trill .....	1-32
Protocols: Shortest Path Bridging (VLAN) .....	1-33
Protocols: Shortest Path Bridging (MAC-in-MAC) .....	1-34
SDN Definitions: What is SDN anyway? .....	1-35
SDN: It Depends on Who You Ask .....	1-36
Open SDN: Separate Control & Forwarding .....	1-37
Open SDN: OpenFlow Protocol .....	1-38
Open SDN: The Big Picture .....	1-39
Open SDN: Today's Devices .....	1-40
Open SDN: Devices with OpenFlow .....	1-41
Open SDN: Summary .....	1-42
Open SDN: Summary (continued) .....	1-43
SDN via APIs: Review of Open SDN / Separation of Forwarding and Control Planes .....	1-44
SDN via APIs: Programmability / no Separation .....	1-45

SDN via APIs: Review of Open SDN / OpenFlow .....	1-46
SDN via APIs: Proprietary Protocol .....	1-47
SDN via APIs: Today's Devices .....	1-48
SDN via APIs: A Better API .....	1-49
SDN via APIs: Summary .....	1-50
SDN via APIs: Summary (continued) .....	1-51
SDN via Overlays: Virtualized Networks .....	1-52
SDN via Overlays: Doesn't Touch Devices .....	1-53
SDN via Overlays: MAC-in-IP Tunnels .....	1-54
SDN via Overlays: Tunneling Operation .....	1-55
SDN via Overlays: Summary .....	1-56
SDN via Overlays: Summary (continued) .....	1-57
A Closer Look at SDN .....	1-58
Anatomy of an SDN Hardware Device .....	1-59
Anatomy of an SDN Software Device .....	1-60
SDN Device Hybrid Modes .....	1-61
SDN Controller APIs .....	1-62
SDN Controller: Northbound API .....	1-63
SDN Controller Applications .....	1-65
SDN Controller Considerations .....	1-66
OpenFlow Overview .....	1-67
OpenFlow Match Fields .....	1-69
OpenFlow Flow Entries .....	1-70
OpenFlow Flow Tables .....	1-71
OpenFlow Flow Entry Types .....	1-72
OpenFlow Ports .....	1-73
OpenFlow Flow Entry Examples (1) .....	1-74
OpenFlow Flow Entry Examples (2) .....	1-76
OpenFlow 1.1 Changes .....	1-78
OpenFlow 1.1 Changes (cont'd) .....	1-80
OpenFlow 1.1 Changes (cont'd) .....	1-81
OpenFlow 1.1. Changes (cont'd) .....	1-82
OpenFlow 1.1 Changes (cont'd) .....	1-83
Openflow 1.2 Changes .....	1-85
OpenFlow 1.2 Changes (cont'd) .....	1-86
OpenFlow 1.3 Changes .....	1-87
Environments for SDN Applications .....	1-88
SDN Application Environments: Data Centers .....	1-89
SDN Application Environments: WANs .....	1-90
Google WAN Implementation .....	1-91
Google WAN Implementation (cont'd) .....	1-92
SDN App Environments: Routed Networks .....	1-93
SDN App Environments: Carrier / Provider .....	1-94
SDN App Environments: Load Balancing .....	1-95
SDN App Environments: Firewalls .....	1-96
SDN App Environments: Campus / Enterprise .....	1-97
Summary .....	1-98

## Module 2: SDN Application Basics

Objectives .....	2-1
SDN Applications: Two Types .....	2-2

SDN App Type: Reactive .....	2-3
SDN App Type: Proactive .....	2-4
Reactive Application .....	2-5
Reactive Application: Basics .....	2-6
Reactive Application: Java .....	2-7
Reactive Application: Java APIs .....	2-9
Reactive Application: Java API Objects .....	2-10
Reactive Application: Components .....	2-11
Reactive Application: Listeners .....	2-12
Reactive Application: Packet Handlers .....	2-13
Reactive Application: Packet Handlers (cont'd) .....	2-14
Reactive Application: Packet Actions .....	2-15
Reactive Application: Flow Management .....	2-16
Proactive Application .....	2-17
Proactive Application: REST Basics .....	2-18
Proactive Application: REST Basics examples .....	2-19
Proactive Application: REST Basics examples .....	2-20
Proactive Application: REST Basics .....	2-21
Proactive Application: REST Resources .....	2-22
Proactive Application: Flow Management .....	2-23
Proactive Application: Similar to Network Management, with a Major Difference .....	2-24
Summary .....	2-25

## Module 3: SDN Application Design

Objectives .....	3-1
Designing SDN Solutions .....	3-2
Why SDN Solutions May Be Harder? .....	3-4
Steps for Designing an SDN Application .....	3-5
Step #0: Define the Problem .....	3-6
Step #1: Choose Application Design Type .....	3-7
Step #2: Draw Initial Diagram .....	3-8
Sample Diagram: Reactive Application .....	3-9
Sample Diagram: Proactive Application .....	3-10
Step #3: Define Initial (Default) Flows .....	3-11
Step #4: Define Operation of System .....	3-12
Step #5: Perform Complete Walkthrough(s) .....	3-13
Blacklist App Design: Step 2 Diagram .....	3-15
Blacklist App Design: Step 3 (Initial Flows) .....	3-16
Blacklist App Design: Step 4 (Operation) .....	3-17
Blacklist App Design: Step 5 (ARP) .....	3-19
Blacklist App Design: Step 5 (DNS Request) .....	3-20
Blacklist App Design: Step 5 (IP packet) .....	3-21
Blacklist App Design: Step 5 (non-IP packet) .....	3-23
Blacklist App Design: Final Diagram .....	3-24
SDN Application Design Labs .....	3-25
Lab 3-1: Patch Panel .....	3-26
Debrief for Lab .....	3-26
Lab 3-2: Learning Switch .....	3-27
Debrief for Lab .....	3-27
Lab 3-3: Firewall .....	3-28
Debrief for Lab .....	3-28

	.....	3-29
Lab 3-4: Load Balancer .....	.....	3-29
Debrief for Lab .....	.....	3-30
Lab 3-5: WAN Router .....	.....	3-30
Debrief for Lab .....	.....	3-31
Lab 3-6: Overlay .....	.....	3-31
Debrief for Lab .....	.....	3-32
Lab 3-7: Offload .....	.....	3-32
Debrief for Lab .....	.....	3-33
Example: Blacklist (DNS operation) .....	.....	3-34
Example: Blacklist (IP) .....	.....	3-35
Example: Data Center Tunneling .....	.....	3-37
Example: Data Center Offload .....	.....	3-39
Example: Network Access Control (Edge) .....	.....	3-41
Example: Network Access Control (Campus) .....	.....	3-42
Example: Traffic Engineering (Proactive) .....	.....	3-44
Example: Traffic Engineering (Reactive) .....	.....	3-46
Summary .....	.....	3-46

## Module 4: SDN Application Integration

Objectives .....	.....	4-1
SDN Application Development Environment .....	.....	4-2
Environment: Overview .....	.....	4-3
Ubuntu Basics .....	.....	4-4
Environmental Lab 4-1: Ubuntu .....	.....	4-6
Debrief for Lab .....	.....	4-6
Eclipse Basics .....	.....	4-7
Environmental Lab 4-2: Eclipse .....	.....	4-9
Debrief for Lab .....	.....	4-9
Mininet: Overview .....	.....	4-10
Mininet: Installation .....	.....	4-11
Mininet: Options .....	.....	4-12
Environmental Lab 4-3: Mininet .....	.....	4-13
Debrief for Lab .....	.....	4-13
Wireshark (OpenFlow Dissector) .....	.....	4-14
Environmental Lab 4-4: Wireshark .....	.....	4-15
Debrief for Lab .....	.....	4-15
OVS-OFCTL (for examining switch state) .....	.....	4-16
Environmental Lab 4-5: OVS-OFCTL .....	.....	4-17
Debrief for Lab .....	.....	4-17
Reactive Application Review (from module 02) .....	.....	4-18
Reactive Application Components .....	.....	4-19
Listeners: Switch   Datapath .....	.....	4-20
Listeners: Switch   Datapath - example .....	.....	4-21
Listeners: Message   Packet .....	.....	4-22
Listeners: Message   Packet - example .....	.....	4-23
Listeners: Device   Node   Host .....	.....	4-24
Listeners: Device   Node   Host - example .....	.....	4-25
Packet Handler: Packet-In .....	.....	4-26
Packet Handler: Packet-In (example) .....	.....	4-27
Packet Handler: Packet-Out .....	.....	4-28
Packet Handler: Packet-Out (example) .....	.....	4-30

Packet Actions .....	4-31
Packet Actions - example .....	4-33
Packet Matches .....	4-34
Packet Matches - examples .....	4-35
Flow Modifications .....	4-36
Flow Modifications - example .....	4-37
Packet Handling Precedence .....	4-38
Coding Lab 4-6: Learning Switch .....	4-39
Debrief for Lab .....	4-39
Coding Lab 4-7: Learning Switch w/ Loops .....	4-40
Debrief for Lab .....	4-40
Coding Lab 4-8: Learning Switch w/ OF1.3 .....	4-41
Debrief for Lab .....	4-41
Coding Lab 4-9: Overlays .....	4-42
Debrief for Lab .....	4-42
Device Limitations .....	4-43
Device Limitations (cont'd) .....	4-45
Application Coordination .....	4-46
Performance and Scale .....	4-48
High Availability and redundancy .....	4-50
High Availability .....	4-51
Security .....	4-52
Summary .....	4-53

## Module 5: Applications on the HP VAN SDN Controller

Objectives .....	5-1
HP VAN SDN Controller Installation .....	5-2
Eclipse Basics .....	5-3
Maven Basics .....	5-4
OSGi Basics .....	5-6
Keystone Basics .....	5-7
Cassandra Basics .....	5-8
Java Basics .....	5-9
Java Basics .....	5-10
Java Basics .....	5-11
Javadocs .....	5-12
HP Controller: Java API Overview .....	5-13
HP Controller: Application Overview .....	5-15
Factory Example .....	5-16
DataPath Listeners .....	5-17
DataPathListener Example .....	5-18
Message Listeners .....	5-19
Node Listeners .....	5-20
Sequenced Packet Listeners .....	5-21
Sequenced Packet Listeners (cont'd) .....	5-22
SequencedPacketListener Example .....	5-23
SequencedPacketListener Example (cont'd) .....	5-24
Message Context .....	5-25
Message Context (cont'd) .....	5-26
Incoming Packets: OfmPacketIn Object .....	5-27
OfmPacketIn Example .....	5-28

---

Packet Contents .....	5-29
Outgoing Packets: PacketOut Interface .....	5-30
PacketOut Example .....	5-31
Outgoing Packets: Ofm[Mutable]PacketOut .....	5-32
Factories in the HP Controller .....	5-33
Match Object .....	5-34
MatchField Object .....	5-35
Match and MatchField Example .....	5-36
Actions .....	5-37
Actions Example .....	5-38
Flow Modifications: OfmFlowMod Object .....	5-39
OfmFlowMod Example .....	5-40
Sending the Flow Modification Message .....	5-41
Sending Flow Mod Message Example .....	5-42
Mutable and Immutable .....	5-43
Mutability Example .....	5-44
Lab 5-3: Default Application .....	5-45
Debrief for Lab .....	5-46
Lab 5-4: Structure, Initialization, Debug .....	5-46
Debrief for Lab .....	5-47
Lab 5-5: Switch Listener .....	5-47
Debrief for Lab .....	5-48
Lab 5-6: Packet Listener .....	5-48
Debrief for Lab .....	5-49
Lab 5-7: Packet Handler .....	5-49
Debrief for Lab .....	5-50
Lab 5-8: Action List .....	5-50
Debrief for Lab .....	5-51
Lab 5-9: Flow Modification Message .....	5-51
Debrief for Lab .....	5-52
Lab 5-10: Match Object and Match Fields .....	5-52
Debrief for Lab .....	5-53
Lab 5-11: Flow Modification Completion .....	5-53
Debrief for Lab .....	5-54
HA Overview: Primitives, Synchronization .....	5-55
HP Controller Teaming .....	5-56
Controller Teaming Example .....	5-57
High Availability Services .....	5-58
Distributed Bus Example .....	5-59
Regions and Role Orchestration .....	5-60
Distributed Persistence .....	5-61
Metrics Overview .....	5-62
Summary .....	5-63

# **Application History**

## **Module 1**

### **Overview**

This module introduces, and gives the history of, Software Defined Networking Applications. At the end of this module you should understand the history behind the emergence of SDN as a means of controlling networking behavior – and how SDN applications have the opportunity to drive the manner in which networks are controlled in the future.

#### **NOTES**

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

# Welcome to HP's Developing SDN Applications Course

Primary goal of this course:

- Provide the student with the required skills necessary to successfully develop an SDN application on the HP VAN SDN Controller

During the course the student will:

- Learn about the history of SDN and SDN applications
- Learn, and experience, designing an SDN application
- Learn, and experience, developing the code necessary to create SDN applications on the HP VAN SDN Controller

Figure 1-1: Introduction

The goal of this course is simple – when you leave you will be able to develop SDN applications on the HP SDN VAN Controller.

The course is designed to be beneficial to you, whether you are a networking expert but a programming novice, or whether you are a programming novice and a networking novice.

In addition, you will learn more background information about the development of SDN, and SDN applications, over the past ten years.

## NOTES

---

---

---

---

## SDN Backstory

In this section we will examine some of the history of SDN. Quite often it is helpful to understand how we got to where we are, in order to understand where we are going. Or, where we should be going.

This is true in the development of SDN applications as well. So for a moment we will look back at the historical context of SDN, and then we will consider where SDN is at this point in time.

Understanding the different definitions of SDN will help provide a framework for making application development decisions, and so we consider each definition in turn.

### NOTES

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

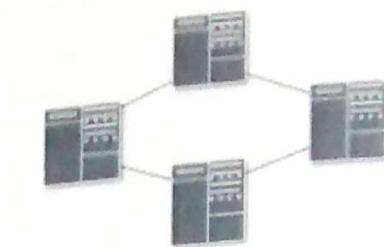
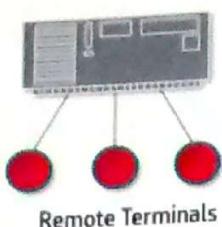
---

# A Brief History of Networking

**1970s: Mainframes & remote terminals**

Centralized control

Point-to-point



**1980s: Minicomputers**

- Distributed control

- Point-to-point

Figure 1-2 – 1970s and 1980s: Mainframes and Minicomputers

Back in the old days, before networking as we know it today existed, there were forms of networking. In the 1970s, mainframe computers were the dominant form of computing, and these computers would support remote terminals for either user input, or for support of remote card readers. Networking in these cases was driven centrally, by the mainframe. Each connection was point-to-point; the idea of forwarding packets did not exist.

In the 1980s, compute was moving from mainframes to minicomputers, which allowed for more systems, operating in a distributed environment. This environment also fostered the idea of the distributed communication between systems. Although this form of network communication was distributed, it was still point-to-point; forwarding and routing by networking devices was not prevalent.

## NOTES

---



---



---



---



---



---

# A Brief History of Networking

## 1990s: PCs and networking: LANs and WANs

"Networking devices"

Distributed control  
Autonomous devices

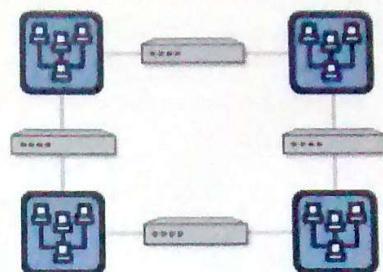


Figure 1-3: 1990s: PCs and LANs

In the late 1980s and early 1990s, networking as we know it today had been invented and was being deployed widely. We developed Local Area Networks, which allowed locally-connected computers to pass data back and forth. These LANs could even be connected to each other by devices originally called "bridges" (later, "switches").

These network devices were called "transparent" because they were able to be inserted into the network at various places, and they would automatically learn how to forward packets between their network segments. This was possible because the devices were autonomous, meaning that they could make decisions about what to do about packets that arrived on their ports *all by themselves*. The network devices did not require any outside control from a central system.

These network devices were so smart that they could even operate in environments in which loops existed. They did this by enforcing a logical tree structure ("spanning tree") over the network, in order to eliminate loops. The result was that certain links between devices were blocked – traffic was not forwarded on these links. If a link went down, the devices in the network would re-configure (aka "converge") the spanning tree all over again, creating a new hierarchy.

So to summarize, devices were independent, autonomous, and the networking intelligence was distributed amongst all the networking devices.

## NOTES

---

---

---

# A Brief History of Networking (cont.)

## 1990s: PCs and networking: Switches

Distributed control

Independent,  
intelligent,  
and autonomous  
network devices

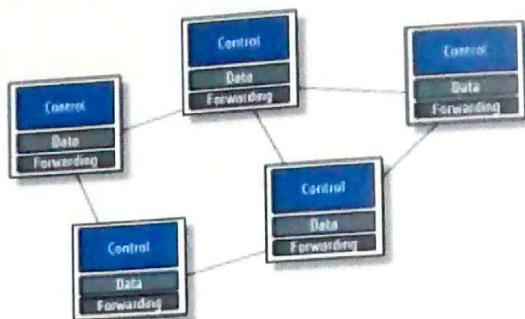


Figure 1-4: 1990s: Switches

Very quickly, these networking devices advanced in their capabilities, so that they no longer just supported links between two LANs, but they supported links between *many* LANs. These devices evolved in their speed and capacity and the word "bridge" was eventually replaced by the word "switch".

Originally, all these switches implemented their intelligence in firmware running on the device itself. But eventually that functionality migrated into hardware (ASICs), such that forwarding was done in that layer. The intelligence of the device, however, was driven by software running 'above', but in the same device.

Hence, devices had a lower level, which had data tables and forwarding mechanisms; and they had the higher 'control' level, in which the distributed intelligence driving decisions about how to forward and route packets was resident.

These switches, like the bridges that preceded them, were independent, intelligent, and autonomous, capable of individually making forwarding and routing decisions. Certain information was shared between these devices (e.g. spanning tree, subnet responsibilities, etc.), but ultimately it was the device itself that made decisions that allowed the lower 'forwarding' layer to be programmed appropriately.

## NOTES

---



---



---

# Networking Today

2013: PCs, laptops, tablets, phones, cloud, datacenters, virtualization

Distributed control

Independent,  
intelligent,  
and autonomous  
network devices

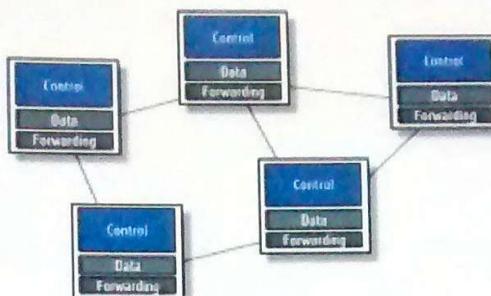


Figure 1-5: Networking Today (same as 20 years ago)

So then, set your brains ahead a couple decades, to today (2013 or 2014). Looking at these wonderfully intelligent networking devices, you can see that – nothing has changed.

Of course there have been improvements. For example, the spanning tree standards have resulted in convergence time improvements of an order of magnitude – or more. New protocols have been developed to address the wastefulness of having blocked links that don't forward traffic. However, that doesn't escape the fact that networking devices are fundamentally the same as they were almost two and a half decades ago.

Consider other technologies. How much have computers changes in the past couple of decades? We now have desktop PCs that sport multiple cores, huge amounts of memory, threads, etc. The form factors have changed such that we have laptops, tablets, and phones that have far more processing power than even the most advanced computers of the past.

Server technology has moved huge amounts of storage and processing up into large data centers and into private, public, and hybrid clouds. Virtualization has made it possible to move compute functionality across the country in a matter of minutes – or less.

Advances in storage have been similar. Storage Area Networks, virtual storage capacity, etc., have kept pace with compute technology.

But not so with networking. As we see when we look a bit deeper, networking has fallen far behind other technologies. People in the industry – specifically researchers and innovators – have looked at this situation and begun to ask whether we can do something to address the problem of network technological deficiencies.

We examine some of these attempts in the next few pages.

## Network Change: ForCES

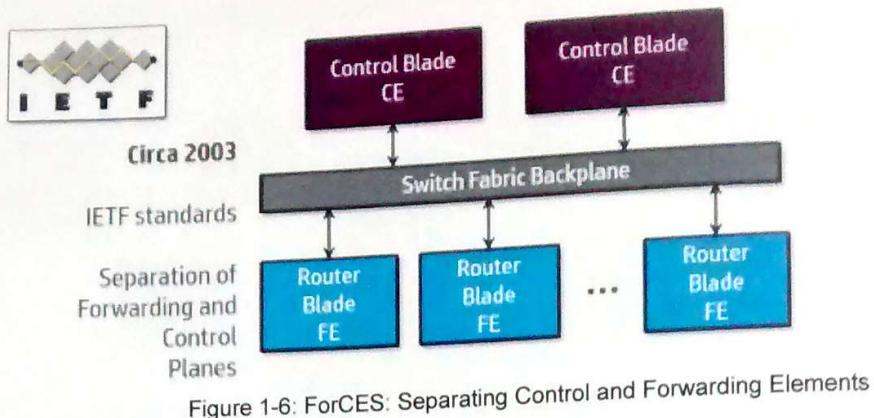


Figure 1-6: ForCES: Separating Control and Forwarding Elements

As much as 10 years ago, some folks in the IETF were looking at better ways to design networking devices. What became apparent to them was that it would be wise to separate the *forwarding* functionality in these devices, from the *control* functionality.

And so, the first of many *ForCES* standards was conceived. *ForCES* (Forwarding and Control Element Separation) proposed making the forwarding elements distinct from the control elements in a network device. They even have gone further, to specify that one could theoretically move the control functionality off of the device altogether, running it on a different device, or even on a server.

This is the precursor to the fundamental SDN concept of the separation of the forwarding functionality from the control functionality.

### NOTES

---



---



---



---



---



---



---



---

# Network Change: Clean Slate

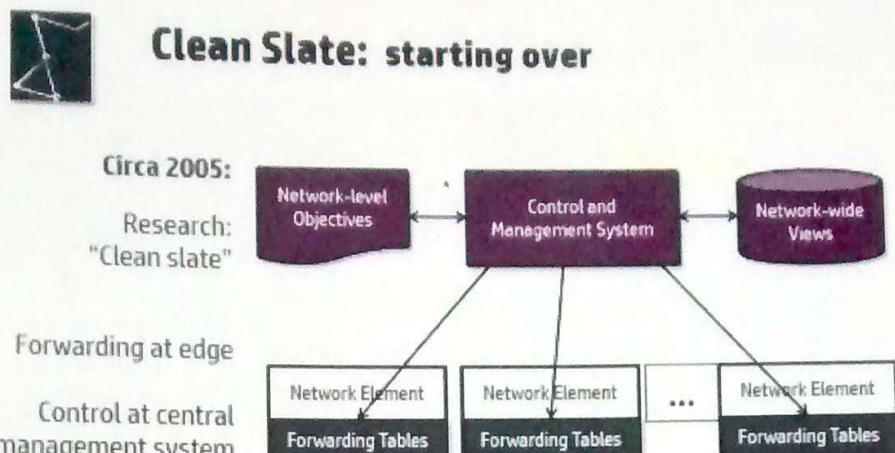


Figure 1-7: Clean Slate: starting over

Following on the heels of ForCES is the “Clean Slate” program, initiated at Stanford University in the US. The Clean Slate program asked the question: if we were to begin building networks anew – without any existing traditional methods – how would we build them? Would networking technology look like it does today, or would it look different?

The answer arrived at by Clean Slate, as with other technologies, is that there should be a control and management system “running the show”. That the network itself should be driven by network-level objectives (rather than distributed device configurations). And furthermore, that this centralized system would be able to look at the entire network, and make optimal, intelligent, and predictable decisions about how traffic should be forwarded and routed throughout that network.

## NOTES

---



---



---



---



---



---



---



---

## First SDN Application: Ethane

**Ethane: Complete SDN(ish) System**

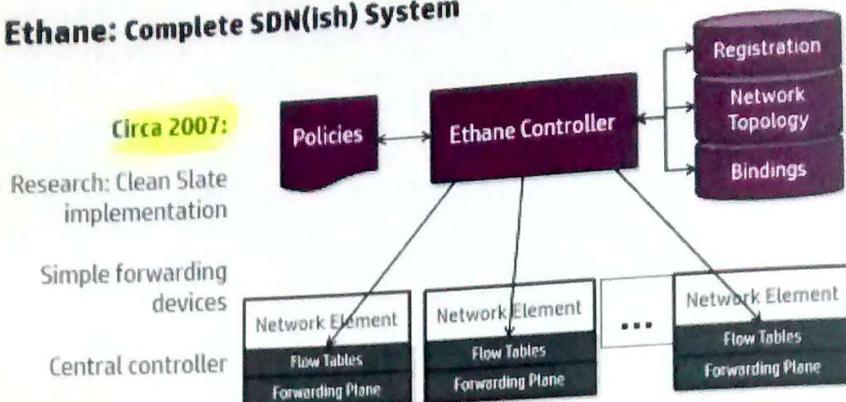


Figure 1-8: Ethane: first SDN application

In a like manner, following on the heels of Clean Slate was the first implementation of what we'd call SDN today. This implementation grew out of Clean State at Stanford.

The implementation was aided by a number of vendors, including HP, NEC, and others, who implemented specifications in their devices which allowed them to make their forwarding functionality available to a central control system.

This central control system – called ‘Ethane’ – used policy information and a database of various information (topology, registration, bindings) to administer rules regarding network access by individual devices. In this way, Ethane is a sort of a Network Access Control (NAC) solution. It is still being used by Stanford today.

Typical NAC systems require control functionality on the device (e.g. RADIUS or captive portal), or else functionality in a special in-line appliance, to achieve their desired functionality. This solution was achieved with no special software on the device or appliance – it was all done with simple devices that exposed their ‘flow tables’ to the central controller.

Note that the primary researcher who developed this technology – Martin Casado – created a company (Nicira) that created a data center solution which a few years later was acquired by VMware for 1.26 billion US dollars. Not a bad return for a little bit of software defined networking.

### NOTES

---



---

## OpenFlow is Born

### OpenFlow: Evolves from prior work

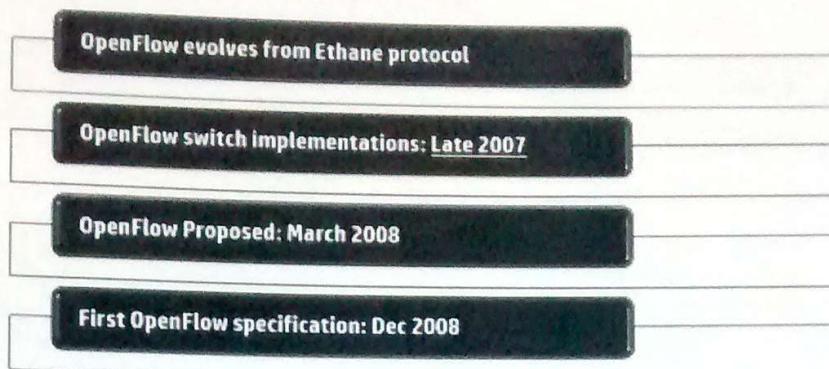


Figure 1-9: OpenFlow becomes a standard

The protocol that was used in the Ethane project eventually matured into the OpenFlow protocol, which lets centralized controllers communicate with these new types of devices. Vendor switches have been implementing OpenFlow since it was first developed in 2007. It is notable that HP was one of the primary vendors that contributed OpenFlow-supporting switches to Ethane and the Clean Slate program at that time.

The first official OpenFlow specification was developed in December 2008 – over five years ago. So to say that SDN is ‘new’ can be a bit misleading, since people have been working in this area for quite a few years.

The main OpenFlow standard that is most widely implemented today is OpenFlow 1.0, which was standardized in December of 2009. Since that point in time, there have been a number of subsequent significant versions of OpenFlow – 1.1, 1.2, 1.3, and most recently, 1.4.

### NOTES

---

---

---

---

---

---

---

## OpenFlow Interest: Initially

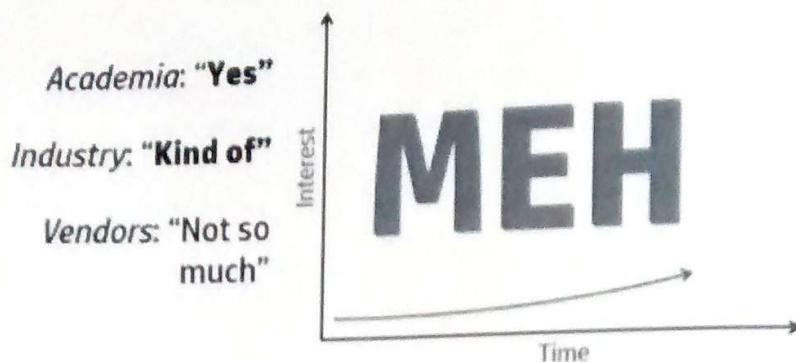


Figure 1-10: OpenFlow Initial Interest

So one might assume that this new technology, aimed at re-defining how networking works, would take the networking world by storm, correct? As it turns out, although academia and innovators were quite excited, the networking vendors were not, seeing it as a threat to their profit margins.

The networking vendors that did show interest were the ones trying to unseat the incumbents. However, until the networking heavyweights got on board with this new disruptive technology, the general response to SDN and OpenFlow back in 2008 and up through 2012 was "meh" (*exclamation*: expressing a lack of interest or enthusiasm.)

### NOTES

---

---

---

---

---

---

---

# Networking is a Closed System

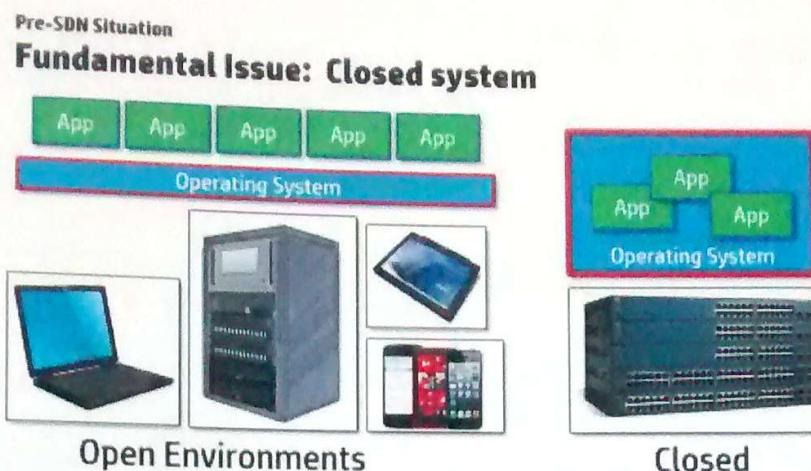


Figure 1-11: Networking is a closed system

Taking a step back, consider why the current status quo - what the networking vendors are trying to protect - is a bad idea. One of the fundamental issues that has driven SDN is the fact that networking functionality today operates as a closed system. That is to say, only engineers belonging to the company that created the hardware, are able to design and write software for those devices.

Contrast this to the open environment of the compute world. You have devices such as laptops, servers, tablets, and phones - different hardware from different vendors. And yet just about anyone can write software to make those devices do innovative and special things. The reason is because there is an operating system which runs on top of the devices, abstracting their various differences, and providing a common interface to the applications above.

And for the most part, anybody can write software to run on those devices. Sadly this is not true for networking, and this had led to some of the issues we show on the next pages.

## NOTES

---



---



---



---



---



---

## Why is a Closed System a Bad Thing?

### Why is this bad?

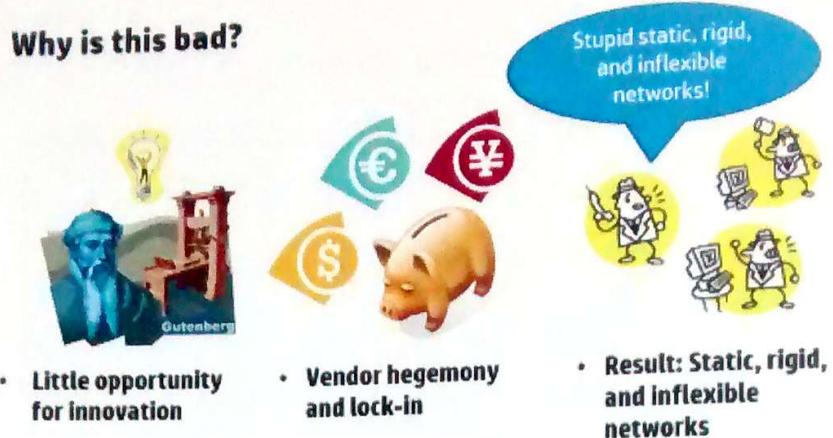


Figure 1-122: Why is a Closed System a Bad Thing?

A closed system is bad for a number of reasons:

- Because of the closed nature, there is little opportunity for innovation in the networking space. Compare this to the innovation that has taken place on phones, tablets, etc.
- Because the vendors control the hardware and software, they can develop products that are proprietary and expensive. The ones to suffer here are the customers.
- As a result of the relative stagnation in terms of networking innovation, networks today are rigid, static, and inflexible. Which only causes problems when compared to advances that have been achieved in other realms.

### NOTES

---



---



---



---



---



---



---



---

# What Caused Interest in SDN to Increase?

## Data centers

*DCs are the "straw that broke the camel's back"*

Massive scale of DCs and cloud threaten to break many network technologies:

- **MAC table issues:** overflowing
- **Spanning tree issues:** unused links
- **VLAN issues:** 4K not enough for multi-tenant clouds
- **Traffic engineering:** not a special case any more

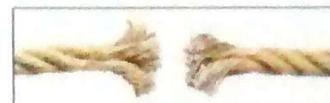


Figure 1-133: What Caused Interest in SDN to Increase?

In spite of these disadvantages in the networking space, the SDN revolution did not start to gather steam until data centers and the cloud became ubiquitous. And the sad fact is that, in data centers, networking is broken. The next pages describe some of the manners in which this is true, including:

- MAC tables overflowing because of the massive increase in the number of compute nodes due to server virtualization.
- Spanning Tree deficiencies become intolerable, both in terms of unused links, and convergence times.
- The number of possible VLANs, used to support multi-tenancy, is no longer sufficient (who would ever need more than 4096 VLANs?!).
- Traffic engineering was always a luxury; in this day and age, it has become mandatory, as providers and customers seek to make the most use of their available bandwidth.

## NOTES

---

---

---

---

---

---

---

## MAC Address Table Issue

### MAC Address Table Overflow

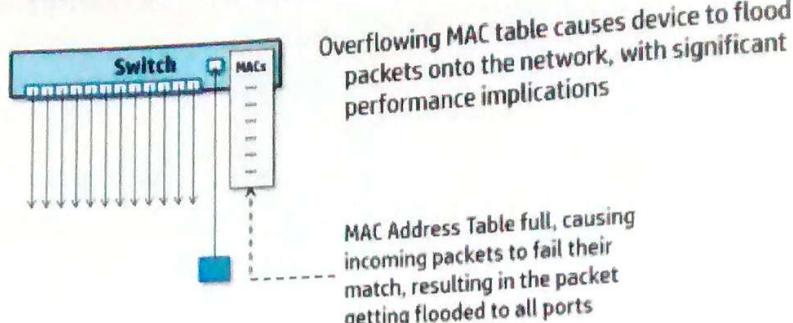


Figure 1-144: What Caused Interest in SDN to Increase?

Why is MAC table overflowing an issue? Because when the table's length is exceeded, then the switch must begin to flood packets on the network. This creates general problems as well as increasing bandwidth used by one or two orders of magnitude.

#### NOTES

---

---

---

---

---

---

---

## Spanning Tree Issue

### **Spanning Tree issues:**

**Inefficiency:** wasting bandwidth due to blocked links

**Latency:** delays introduced due to re-convergence after change

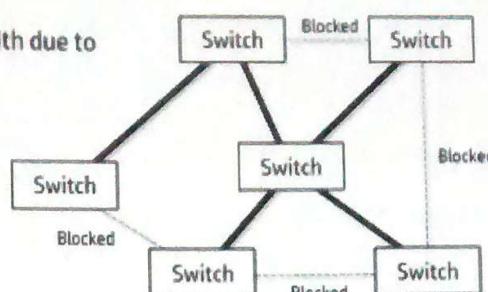


Figure 1-155: Spanning Tree Issue

Spanning tree is sub-optimal because of the blocked links in a network (in the picture above, 44% of the bandwidth of the network is being unused).

Not only that, but also when there are changes to configuration, it takes the switches a while to re-configure themselves into a new spanning tree. Having a network go down, even for a short time, these days is intolerable.

## NOTES

## VLAN Issue

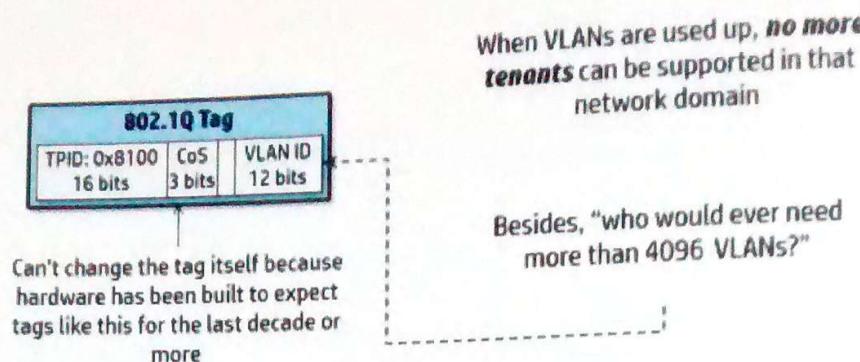


Figure 1-166: VLAN Issue

There are only 12 bits available for the VLAN Identifier in the 802.1Q tag inside an Ethernet header. And most devices have hardware which is programmed to recognize this tag, so it cannot be altered.

Consequently we are stuck with about 4K possible VLANs. In the past this was not an issue; however, with large data centers and especially the public cloud, this is a limitation which cannot be accommodated.

### NOTES

---



---



---



---



---



---



---



---

# Traffic Engineering Issue

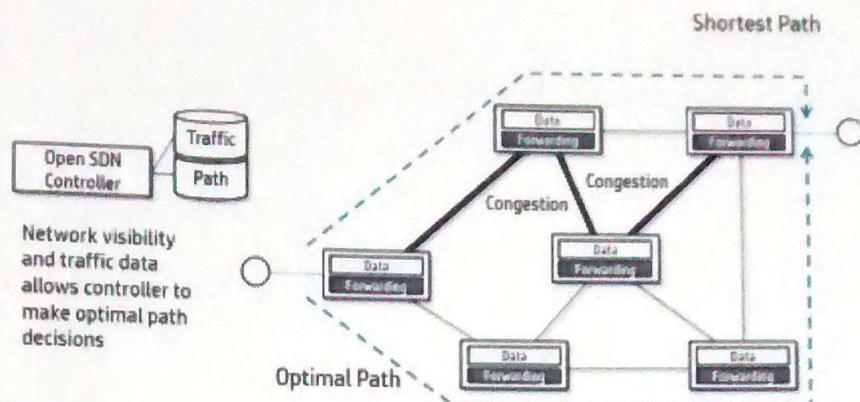


Figure 1-177: Traffic Engineering Issue

Today's routing mechanisms take into account (a) shortest number of hops, and possibly (b) potential maximum bandwidth. However, that is not always the best way to route traffic, as shown in the diagram. When there is congestion in the path, it may be wiser to route traffic in the optimal way, which is different by any current shortest-path algorithm.

In order to take into account this type of information, it is required that there be some central authority which has a view of the entire network, and can make decisions which are appropriate. This is not possible today with the current independent, autonomous device model that persists in networking.

## NOTES

---



---



---



---



---



---



---



---

## Agility and Automation Issue

### Data centers

Today, networking is the **Hammer Pants** of data centers

No Agility: to quickly move networks  
from one physical location to another

No Automation: to use programmatic  
methods to make changes

No Virtualization: to instantly create,  
destroy, and move network resources  
anywhere, anytime in the DC



Figure 1-188: Agility and Automation Issue

To put a picture to this situation, networking is the “hammer pants” of networking; it seemed cool at the time (in the 90s), but today isn’t looking so cool any more.

In addition to the deficiencies mentioned already, there is a huge gap in the areas of agility, automation, and virtualization. These are all sides of the same deficiency, which is directly related to the manual and static nature of networks.

Data centers can move servers and storage instantly, on the order of minutes. However doing the same thing for networks takes on the order of days rather than minutes, because it is a manual process, not lending itself to automation, and thus is not agile. And the virtual nature of servers and storage - which enables the agility - is not possible with networks.

### NOTES

---

---

---

---

---

---

---

---

---

---

## Cost Issue

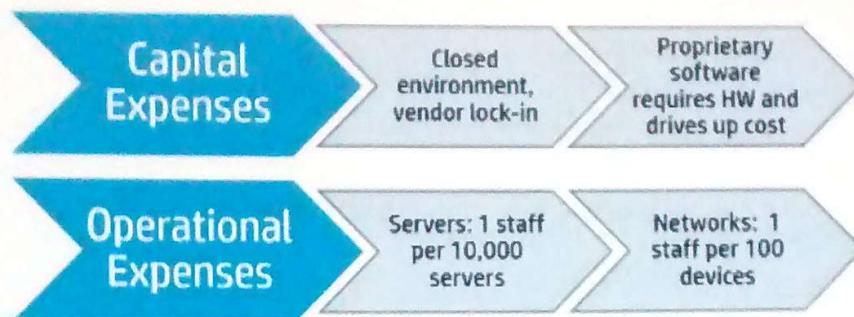


Figure 1-199: Cost Issue

We referred to this earlier - the high cost of networking. Cost can be separated into two facets:

- Capital Expenses (CapEx): the actual cost of the devices. Independent, autonomous devices require much more hardware (CPU, memory), as well as software (which must be developed, debugged, and maintained). Then there is the fact that with a closed system, competition is reduced, and thus vendors are free to charge higher prices.
- Operational Expenses (OpEx): the cost of maintaining those rigid, inflexible, and manual devices is much higher than would be the case for simpler devices which are controlled in an automated manner by a central controller.

### NOTES

---

---

---

---

---

---

## OpenFlow Interest: after Data Center



Figure 1-20: OpenFlow Interest: after Data Center

Suddenly with the acute need to evolve networking technology made apparent by the data center issues, there is now a much increased level of interest from everybody involved.

### NOTES

---

---

---

---

---

---

---

# Using Management to Address DC Issues

## Solving DC Issues: *management*

Different attempts have been made to make datacenter networking more agile and able to adapt to changes:

- **Orchestration** solutions
- VM Management **Plug-In** solutions
- **RADIUS** solutions

Figure 1-21: Using Management to Address DC Issues

### NOTES

---

---

---

---

---

---

---

---

# Orchestration Solutions

## Orchestration Solution:

Scripting to automate certain common tasks

Good for firmware updates

***Not dynamic enough for datacenter automation needs***

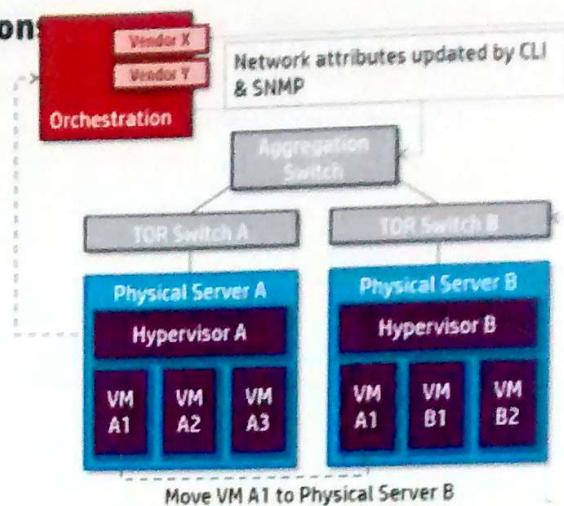


Figure 1-22: Orchestration Solutions

One early attempt to solve data center issues was to use standard Network Management Orchestration software. Unfortunately, this technology was limited to using the old CLI and SNMP, so the functionality was limited and the 'automation' of moving networks in this way never really caught on.

## NOTES

---



---



---



---



---



---



---



---

# VM Plug-in Solutions

## VM Plug-in Solutions

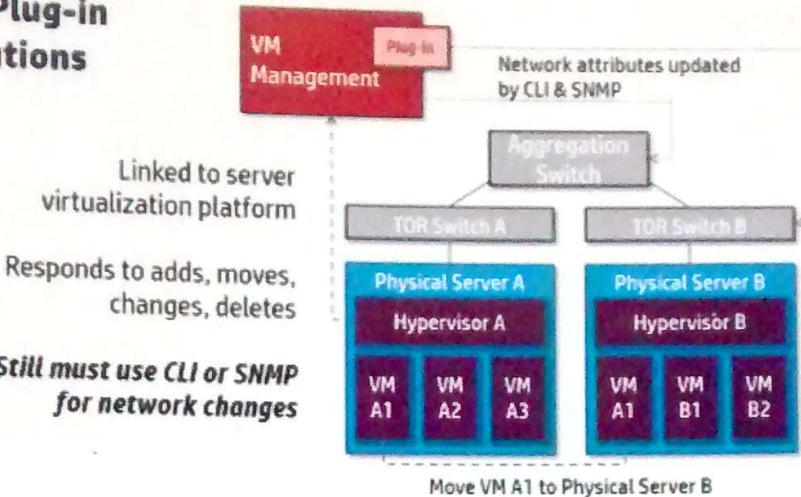


Figure 1-23: VM Plug-in Solutions

VM management plug-in solutions were a similar attempt; the idea was to trigger network changes based on events generated within the VM management subsystem. Unfortunately, these also had limited value, since they still used CLI and SNMP to make changes.

## NOTES

---



---



---



---



---



---



---



---

# RADIUS Solutions

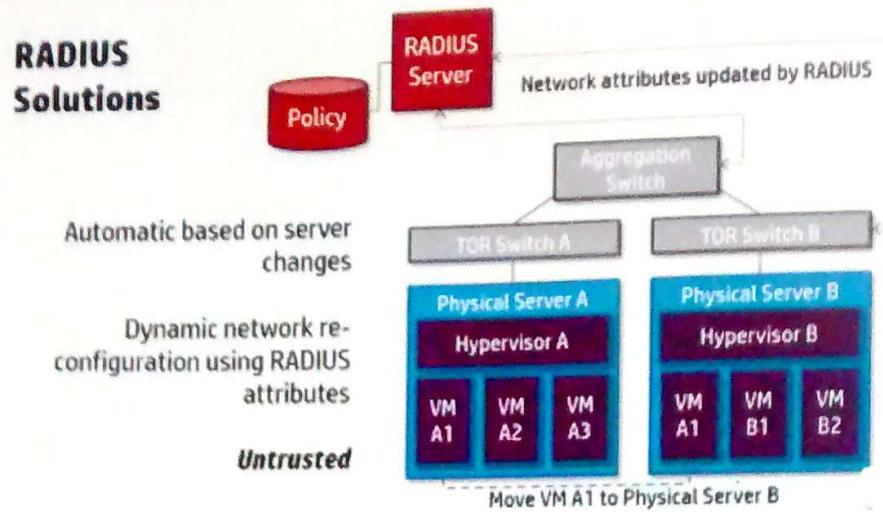


Figure 1-24: RADIUS Solutions

One other type of solution - which also never caught on - was to use RADIUS to trigger the automation of network changes. The positive sides of this were:

- Naturally event driven (a VM moves to a new location, its MAC address is seen on the network, and this starts the automation process)
- Generic (non-CLI and non-SNMP) mechanism for moving networks around, using RADIUS attributes defined for the purpose (e.g. RFC 3580)

Unfortunately this solutions was not adopted either, leaving it up to network device technology to address these issues.

## NOTES

---



---



---



---



---



---



---



---

# Using Tunnels to Address DC Issues

## Solving DC Issues: *tunnels*

Different attempts have been made to make networking better able to handle datacenter issues such as MAC address table and VLAN exhaustion using tunnels:

- **Virtual Networking** solutions using tunnels
- **Spanning Tree** replacement protocols

Figure 1-25: Using Tunnels to Address DC Issues

One approach to addressing the issues related to MAC address table overflow, VLAN exhaustion (multi-tenancy), and spanning tree, was to use new network technologies called 'tunnels'. The idea here is to encapsulate the original packet and place it inside another outer packet shell, which then gets sent across the network to the other side (hence 'tunnel'), where it is decapsulated and sent to the destination host.

There are two major areas where tunnels are proposed:

- Tunnels to provide Network Virtualization across the top of a physical network
- Tunnels to replace spanning tree and make more efficient use of the network links

We explore these types of tunneling technologies in the following pages.

There are three Network Virtualization tunneling technologies we will look at: VXLAN, NVGRE, and STT. There are two path-related tunneling technologies we will look at: SPB and TRILL.

## NOTES

---

---

---

---

# Tunnels: VXLAN

## Network virtualization: VXLAN

- MAC-in-IP tunnel
- Unicast between switches
- UDP (VXLAN port 8472)
- $2^{24}$  Virtual Networks

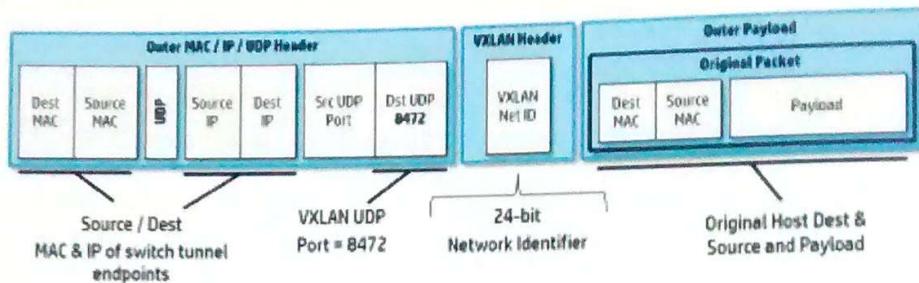


Figure 1-26: Tunnels: VXLAN

VXLAN stands for Virtual eXtensible LAN.

One of the first tunneling technologies aimed at network virtualization for data centers was NVGRE, created primarily by Cisco and VMware. Things to note about VXLAN:

- Like the other tunneling technologies for network virtualization, it uses MAC-in-IP tunneling, meaning that the entire original Ethernet packet - including MAC addresses - is encapsulated within a new IP packet.
- VXLAN uses UDP, with the UDP destination port shown in the diagram (8472).
- VXLAN includes a Network Identifier which is 24 bits, allowing for plenty of unique tunnels, and solving (for now anyway!) the multi-tenancy issue.
- Like the other tunneling technologies - and really, like all tunneling technologies - there are unicast packets exchanged between the two tunnel endpoints.

You will see in subsequent slides that VXLAN is similar in many respects to NVGRE and STT.

## NOTES

---



---



---



---



---

# Tunnels: NVGRE

## Network virtualization: NVGRE

- MAC-in-IP tunnel
- Unicast between switches
- GRE (IP Protocol 0x2F)
- $2^{24}$  Virtual Networks

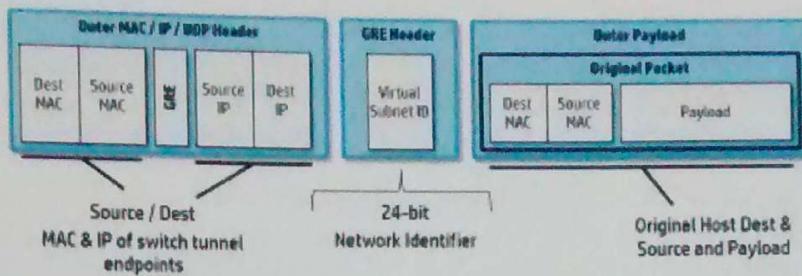


Figure 1-27: Tunnels: NVGRE

NVGRE stands for Network Virtualization Generic Routing Encapsulation

NVGRE is an extension of GRE, which has been around for some time. NVGRE is promoted by Microsoft and others.

Some things to note about NVGRE:

- Like the other tunneling protocols doing network virtualization, NVGRE uses MAC-in-IP tunnels.
- Rather than using IP/UDP as does VXLAN, NVGRE uses the Ethernet protocol 'GRE' as its packet type.
- Like VXLAN, NVGRE uses a Network Identifier which is 24 bits.
- As with the others, packets are carried across the network from tunnel endpoint to tunnel endpoint in a unicast manner.

## NOTES

---



---



---



---



---



---



---



---

# Tunnels: Stateless Transport Tunneling

## Network virtualization: STT

- MAC-in-IP tunnel
- Unicast between switches
- TDP (STT port 7471)
- 64 bits of Context ID

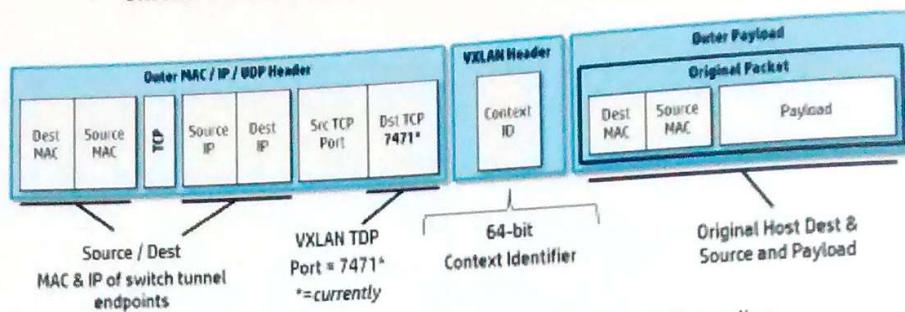


Figure 1-28: Tunnels: Stateless Transport Tunneling

STT stands for Stateless Transport tunneling.

Proponents of STT state that this tunneling protocol is superior because it utilizes capabilities in server NICs to more efficiently pass encapsulated packets across the network.

Things to note about STT:

- STT uses MAC-in-IP tunnels, as with the others (VXLAN and NVGRE)
- STT uses TCP, although as the name states, it is “stateless”. The reason for using TCP is to make use of functionality in server NICs.
- The Network Identifier for STT is 64 bits.
- Once again, packets are passed in a unicast manner from one tunnel endpoint to the other.

## NOTES

---



---



---



---



---



---



---



---



---

# Using Protocols to Address DC Issues

## Solving dc issues: protocols

Different attempts have been made to overcome datacenter networking issues, especially spanning tree, using new protocols

- Trill (MAC-in-MAC)
- Shortest Path Bridging (Q-in-Q, MAC-in-MAC)

Figure 1-29: Using Protocols to Address DC Issues

Trill and Shortest Path Bridging are two attempts to address the issues related to spanning tree that currently make layer 2 networks less efficient than they could be. Trill has been championed by Cisco, and SPB by various other networking vendors.

## NOTES

---

---

---

---

---

---

---

# Protocols: Trill

## Trill

**T**Ransparent Interconnection of  
Lots of Links  
MAC-in-MAC encapsulation  
IS-IS link-state protocol for  
determining best path  
No spanning tree

I hope that we shall one day see  
A graph more lovely than a tree.  
A graph to boost efficiency  
While still configuration-free,  
A network where RBridges can  
Route packets to their target LAN.  
The paths they find, to our elation,  
Are least cost paths to destination!  
With packet hop counts we now see,  
The network need not be loop-free!  
Rbridges work transparently,  
Without a common spanning tree.

Figure 1-30: Protocols: Trill

A detailed understanding of Trill is not within the scope of this class, but it is enough to say that:

- Trill bridges are renamed 'Rbridges' in this scheme.
- Trill uses MAC-in-MAC encapsulation, meaning that a new MAC header is pre-pended onto the original packet in order to take it across the network to the destination RBRidge.
- Trill uses the Intermediate-System-to-Intermediate-System link state protocol to determine the topology of the RBRidges, and thence the shortest path between two nodes.

As with SPB, the goal here is to eliminate spanning tree and replace it with a protocol that allows full usage of all links in the broadcast domain.

## NOTES

---



---



---



---



---



---



---



---

# Protocols: Shortest Path Bridging (VLAN)

## SPB(v): USING Q-IN-Q

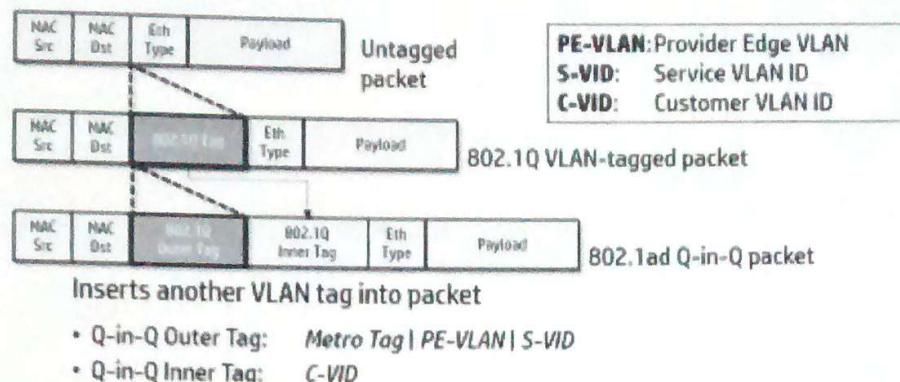


Figure 1-31: Protocols: Shortest Path Bridging (VLAN)

Shortest Path Bridging, like Trill, is intended as a replacement for spanning tree, using all available links in the broadcast domain, and routing packets through the network based on shortest path calculations.

SPB has two mechanisms for achieving its ends: Q-in-Q (aka VLAN tagging) and MAC-in-MAC. These embodiments are referred to as SPB(v) and SPB(m) respectively, the 'v' standing for VLAN, and the 'm' standing for MAC.

- SPB is an IEEE standard, IEEE 802.1aq to be precise.
- SPB uses either Q-in-Q (IEEE 802.1ad) or MAC-in-MAC (IEEE 802.1ah)
- SPB uses IS-IS for its control plan to discover topology and determine the shortest path between two nodes.

## NOTES

---



---



---



---



---



---



---



---

# Protocols: Shortest Path Bridging (MAC-in-MAC)

## SPB(M): USING MAC-IN-MAC

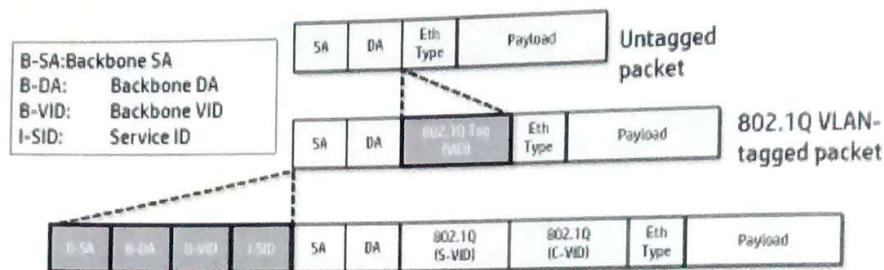


Figure 1-32: Protocols: Shortest Path Bridging (MAC-in-MAC)

The figure shows a generalization of the MAC-in-MAC frame for SPB.

## NOTES

---



---



---



---



---

## SDN Definitions: What is SDN anyway?

What the heck is SDN anyway?

Three SDN Definitions:

- **Open SDN:** *The original, Openflow-based SDN*
- **SDN via APIs:** *The response by incumbents*
- **SDN via Overlays:** *To address datacenter issues*

Figure 1-33: SDN Definitions: What is SDN anyway?

Understanding SDN can be difficult and confusing, in no small part because with all the hype surrounding it, and the movement of the networking vendors to claim SDN support, it has become a free-for-all, with organizations re-defining SDN for their own needs.

In this section we attempt to clear up the confusion by presenting the three categories of SDN that have come into vogue in the last year or so, and defining them precisely. This should help the customer to understand exactly what these definitions are, and who is promoting each.

In simplifying this landscape, we present three definitions of SDN:

- Open SDN
- SDN via APIs
- SDN via Overlays

We go into each of these in subsequent pages.

### NOTES

---

---

---

---

# SDN: It Depends on Who You Ask

## SDN Definitions

- Depends who you ask...



Figure 1-34: SDN: It Depends on Who You Ask

What is SDN? It goes back to the age-old adage, "it depends on who you ask".

- If you ask academia - where SDN started - or some 'open' focused vendors, then you will get one answer.
- If you ask established networking vendors who see to protect their bottom-lines, you will get another answer.
- If you ask software and server vendors, you will get another answer.
- If you ask customers, OpenStack developers, or others - they may be too confused to give you any answer at all.

In the following sections we will make this all clear for you.

## NOTES

---



---



---



---



---



---



---



---



---

# Open SDN: Separate Control & Forwarding

## Separate Control & Forwarding Planes

*Moving control functionality to centralized controller*

- Remove control software from device and put it into a controller
- Device handles the forwarding and data plane functionality
- Controller handles the control plane functionality

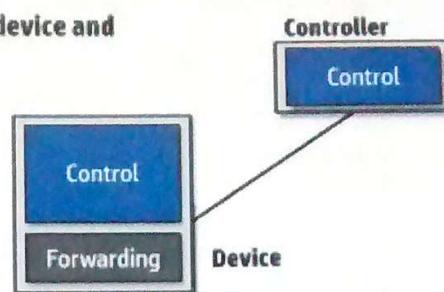


Figure 1-35: Open SDN: Separate Control & Forwarding

The first definition of SDN is what many call “Open SDN”. It is the definition that has been around since the beginning of the stirrings of SDN almost a decade ago. The general idea of Open SDN is:

- Take all that control software that today runs on the networking device, and move it off of the device and put it up into the controller.
- The device handles incoming packets using its hardware that has been programmed to recognize packets and take actions appropriately.
- The controller handles the more complex, compute-intensive functions that require more processing power and a network-wide view.

This is the fundamental change described by ForCES, Clean Slate, and which was first embodied in Ethane. Separating the forwarding (or data) plane from the control plane.

## NOTES

---



---



---



---



---



---



---



---

# Open SDN: OpenFlow Protocol

## Open Networking via SDN

- Simplified devices
- Centralized controller
- Enforcement implemented by devices
- Open environment for innovation

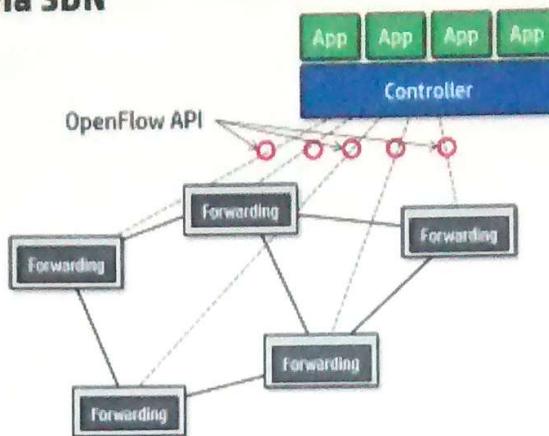


Figure 1-36: Open SDN: OpenFlow Protocol

The result of doing this separation is that you have what is shown in the picture above:

- Devices are simpler because they don't have all of that control plane software running on them.
- A controller is overseeing the operation of the devices in the network (we will explain this in the coming pages).
- The devices will be configured by the controller to make forwarding and routing decisions locally, without having to communicate with the controller.
- The protocol used between the controller and the devices is OpenFlow.
- The controller provides an API so that applications can be written in an open environment, for controlling the behavior of the network. This allows for innovation (as opposed to all software running on devices, which was closed).

This general design of the network - centralized control, simple devices, and open environment - differs greatly from the design that exists today, with independent, autonomous, and expensive devices. This is the goal of Open SDN.

## NOTES

---



---



---



---



---

# Open SDN: The Big Picture

- A network operating system
- A suite of network applications
- Communication between controller and devices via OpenFlow API

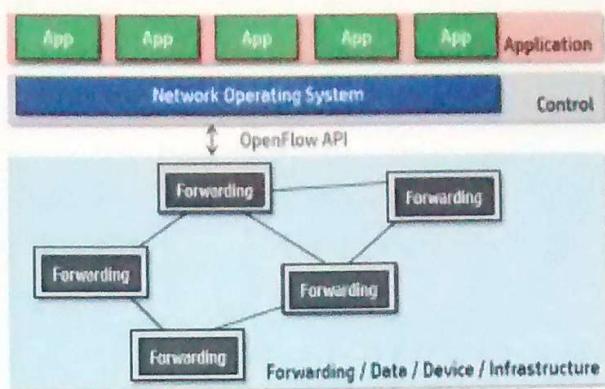


Figure 1-37: Open SDN: The Big Picture

Here is a broader view of what Open SDN is trying to achieve.

- At the bottom is the Data Plane, which is in some places called the Forwarding Plane, the Device Plane, and the Infrastructure Plane. At HP we call it the Data Plane.
- At the next level is the Control Plane, where the controller lives. Note that some have referred to this controller as the 'Network Operating System', the analogy being that of compute nodes where the operating system there serves the purpose of abstracting the devices interactions below (disk, memory, peripherals), and sharing those resources amongst the applications running above.
- At the top level is the Application Plane, in which all of the SDN applications run, utilizing APIs provided by the controller.
- The last thing to notice in this picture of Open SDN is that the protocol between the controller and the devices is OpenFlow.

## NOTES

---



---



---



---



---

# Open SDN: Today's Devices

## Inside Networking Devices Today

Proprietary, vendor-specific control software in network device

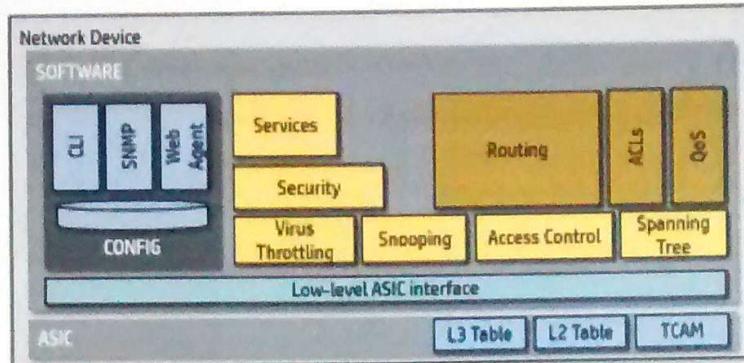


Figure 1-38: Open SDN: Today's Devices

In order to understand Open SDN at a deeper level, consider a typical switch today.

- At the lowest level you have the ASIC, which will contain a number of things, including
  - A layer 2 table for forwarding Ethernet traffic based on destination MAC address.
  - A layer 3 table for routing IP traffic based on destination IP address.
  - A TCAM (ternary content addressable memory) for matching traffic based on a number of fields in the header, which includes the ability to use wildcards.
- Above this hardware layer is a low-level interface to the ASIC, consisting of APIs for setting the appropriate hardware tables (L2, L3, and TCAM).
- Above this is software - LOTS of software - for performing the advanced control functions of the switch. These include functionality for doing spanning tree, access control, security, routing, ACLs, QoS, etc. In addition there is a large amount of configuration information for setting static values on the device.

## NOTES

---



---



---

# Open SDN: Devices with OpenFlow

## Inside Networking Devices *with OpenFlow*

Move software off the device, up to the controller

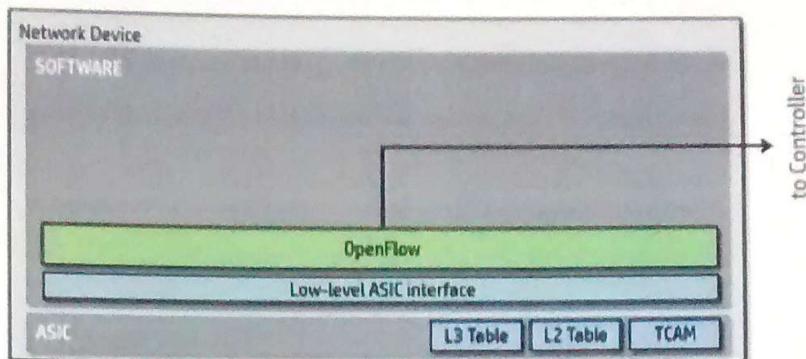


Figure 1-39: Open SDN: Devices with OpenFlow

Now consider that same device, but without all of the control software - replaced by an OpenFlow layer which is capable of communicating with the controller and controller-based applications, and setting the L2, L3, and TCAM tables appropriately.

This is the goal of Open SDN from a device standpoint - making devices simpler by removing control software from the device, and moving it off to the controller.

Consider the ramifications:

- The device requires less memory and CPU power because its compute burden has been greatly reduced.
- The device requires less specially-written software, which must otherwise be replicated on each device. Instead, it is written once - for the controller - and is applied to all current and future devices, independent of the hardware on which it is written.
- That software does not have to be supported on every device, reducing the ongoing maintenance load on the switch vendor.

These are just some of the advantages of this simplified device situation that results from moving towards Open SDN.

## NOTES

---



---



---



---

# Open SDN: Summary

## SDN Definition #1: Open SDN

**Open SDN:** The original, following FORCES, Clean Slate, and Ethane; re-thinking networking.

- Separate forwarding and control planes
- Simple devices, centralized controller
- Open for innovation and interoperability
- OpenFlow protocol

**Proponents:** BigSwitch, HP, IBM, NEC, Extreme, Brocade, Pica8, ...

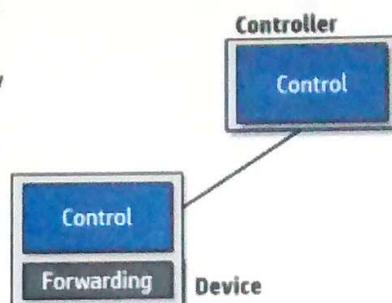


Figure 1-40: Open SDN: Summary

Here is a summary of what we have discussed for Open SDN

- Separating the control plane from the device and moving it up to the controller.
- Devices are now simpler and less expensive.
- The presence of the control functionality on the controller allows for open innovation, since it is no longer wrapped up inside the device.
- The mechanism for communication between controller and device is also open, which is the OpenFlow protocol.

There are a number of commercial vendors who promote this (in addition to academia). Some of these are BigSwitch, HP, IBM, NEC, Extreme, Brocade, Pica8, and others.

## NOTES

---



---



---



---



---



---



---



---

## Open SDN: Summary (continued)

### *Separate Control and Forwarding Planes*

- Simplified devices
- Centralized control
- Openness to enable research and innovation, and promote interoperability

Figure 1-41: Open SDN: Summary

### NOTES

---

---

---

---

---

---

---

# SDN via APIs: Review of Open SDN / Separation of Forwarding and Control Planes

## SDN Refresher

### Control & Forwarding Planes

*Moving control functionality to centralized controller*

- Removing control software from device
- With SDN, the device handles the forwarding and data planes
- With SDN, the controller hosts the control plane

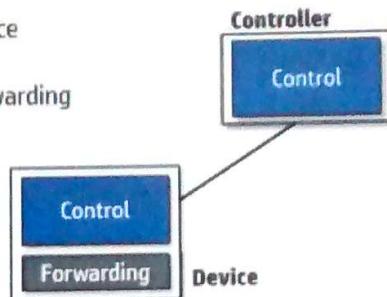


Figure 1-42: SDN via APIs: Review of Open SDN – The Planes

The second SDN definition we'd like to consider has been called by some, "SDN via APIs". In order to understand what this means, recall the previous definition: Open SDN". In this solution, control software was removed from the device, and moved over to the controller. The control functionality of the device has been moved to the controller, resulting in a simpler device, and an open environment on the controller, suitable for innovation and research.

### NOTES

---

---

---

---

---

---

---

# SDN via APIs: Programmability / no Separation

## Programmable APIs on Device

*Providing APIs to dynamically control devices*

- Much of the control software remains in device
- Controller dynamically controls devices via APIs
- Meets *some* SDN goals for programmability, but not for openness or simplicity

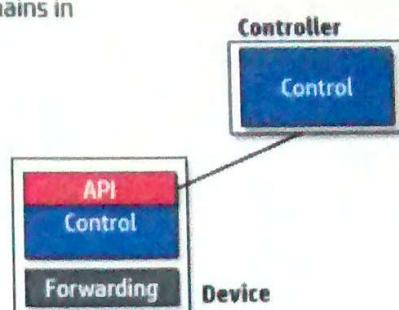


Figure 1-43: SDN via APIs: Programmability / no Separation

In contrast, SDN via APIs attempts to address the domain of SDN by providing a superior set of APIs on the device. The result of this type of SDN solution results in the following:

- Much of the complicated control-plane software remains on the device - thus to a certain degree preserving the status quo for vendors.
- There is a controller in the picture - it is able to control network devices using these superior APIs.
- As a result, the devices are more programmable than was possible with the old APIs such as CLI and SNMP. Some of the data center issues for automation can thus be met.
- However, as you can see, the device is not appreciably simpler or less expensive.
- In addition, while the goal of openness is potentially realized to a certain degree, it is nowhere near the level of openness as Open SDN.

We look at this last item in the next slides.

## NOTES

---



---



---



---