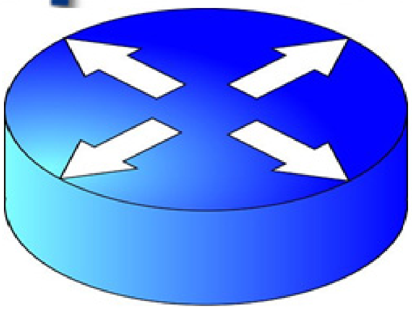
Software Defined Networking

Past, Current, and Future

David Hausheer

Department of Electrical Engineering
and Information Technology
Technische Universität Darmstadt

E-Mail: hausheer@ps.tu-darmstadt.de
http://www.ps.tu-darmstadt.de/teaching/sdn



*Original slides for this lecture provided by Scott Shenker (UC Berkeley) and Bernhard Plattner (ETH Zürich)

Lecture Overview

- A Brief History of the Internet
- The Networking World Prior to SDN
- The Future of Networking, and the Past of Protocols

A Brief History of the Internet

Bernhard Plattner

A Brief History of the Internet

- 1962: ARPANET – network of computers
- 1973: Internet – network of networks
  - Gradually, ARPANET becomes core network
  - Applications: e-mail, interactive host access, simple c/s apps (e.g. gopher, 1991)
- 1984: First international e-mail in Switzerland (UUCP, X.400)
- 1986: NSFNet becomes core network
- 1987: The Internet starts in Europe
- 1993: Classless inter-domain routing
- 1993: IP Next Generation white paper solicitation
- 1994: Internet discovered by the public
  - The web
  - The era of global client/server communication starts

A Brief History of the Internet

TECHNISCHE UNIVERSITÄT DARMSTADT

- 1998: First P2P App, Napster
  - Age of overlay networks starts
  - Google founded as a private company (IPO 2004)

- Late 90-ties: Mobile ad-hoc networks

- 2001: „Structured“ P2P systems using DHT
- 2002: SIP-based Internet telephony
- 2002: Interstellar Internet => DTN => Opportunistic Networks
- 2003: Skype
- 2004: Facebook
  - Social network era starts

A Brief History of the Internet

- 2005: YouTube
  - TV distribution with P2P technology (Zattoo)

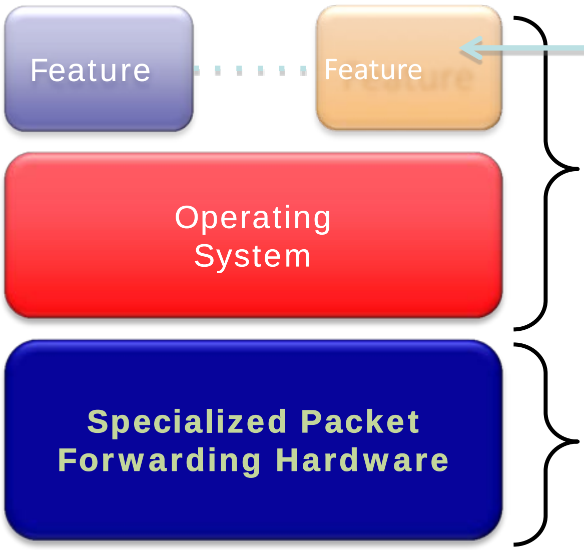
- 2006: Research towards the Future Internet: US NSF
  - Find Program, FP6 Situated and Autonomic Communication, EU FIRE Program

- 2006: Cloud computing (by Amazon)
- 2007: iPhone
- 2009: SDN, currently one of the hottest topics
- 2010: Start of EU Project OFELIA: OpenFlow/SDN testbed

The Networking World Prior to SDN

Bernhard Plattner

Reminder: The Networking Industry (2007)



Routing, management, mobility management, access control, VPNs, …

Million of lines of source code | 5400 RFCs | Barrier to entry

Billions of gates | Complex | Power Hungry

- Closed, vertically integrated, boated, complex, proprietary
- Many complex functions baked into the infrastructure
  - *OSPF, BGP, multicast, differentiated services, Traffic Engineering, NAT, firewalls, MPLS, redundant layers, …*
- Little ability for non-telco network operators to get what they want
- Functionality defined by standards, put in hardware, deployed on nodes

Research Stagnation: Faster networks but not *better* networks

- Lots of *deployed* innovation in other areas
  - Operating Systems: various filesystems, schedulers, virtualization
  - Distributed Systems: DHTs, CDNs, MapReduce
  - Compilers: JITs, new language paradigms

- Networks are largely the same as years ago
  - Ethernet, IPv4,WiFi – IPv6 took ages

- Rate of change in networking seems slow in comparison
  - Need better tools and abstractions to demonstrate and deploy

- Researchers need flexible and changeable platforms for experimentation

Another problem: Closed Systems (Vendor Hardware)

- Can’t extend – vertically integrated
- Stuck with interfaces (CLI, SNMP, etc)
- Hard to meaningfully extend
- Hard to meaningfully collaborate

Future Internet Research

- What should be the architecture of a future Internet?
  - Beyond IPv4 and IPv6
- Future Internet Architecture (FIA) project by the US National Science Foundation (2010)
- Global Environment for Network Innovations (GENI) 2006
- Future Internet and Experimentation (FIRE) Program by the European Commission's 7th Framework Program(2006)
- Clean Slate Approach

The networking researcher's playground

Approach	Example	Performance Fidelity	Scale	Real User Traffic?	Complexity	Open
"ideal system"	n/a	High	High	Yes	Low	Yes
Simulation	NS-2, NS-3, Opnet	medium	medium	no	medium	yes
Emulation	Mininet	medium	low	no	medium	yes
Software Switches	Linux box, Click router	poor	low	yes	medium	yes
HW impl.	NetFPGA	high	low	yes	high	yes
Network Processors	Intel IXP family	high	medium	yes	high	yes
Vendor Switches		**high**	**high**	**yes**	**low**	No

So let's open up the vendor switches!

The Future of Networking, and the Past of Protocols

Scott Shenker

with Martín Casado, Teemu Koponen, Nick McKeown

(and many others….)

Key to Internet Success: Layers

Applications

...built on...

Reliable (or unreliable) transport

...built on...

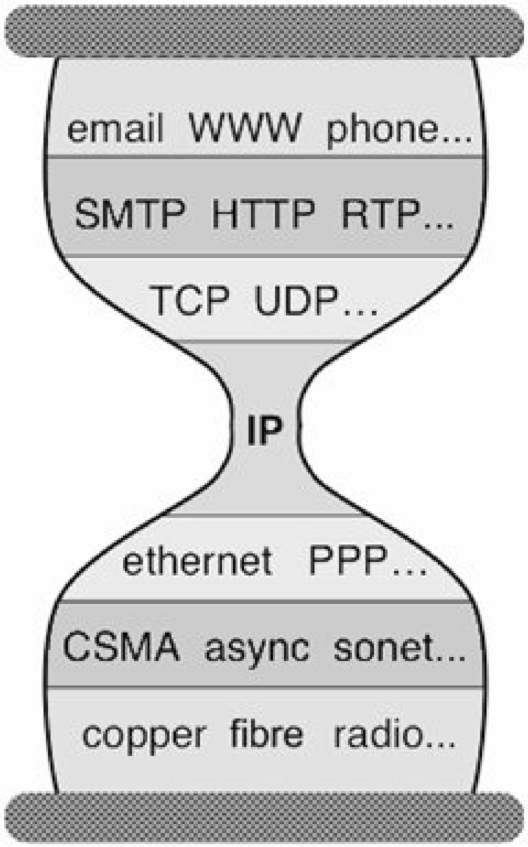
Best-effort global packet delivery

...built on...

Best-effort local packet delivery

...built on...

Physical transfer of bits



Why Is Layering So Important?

- Decomposed delivery into fundamental components
- Independent but compatible **innovation** at each layer
- A practical success of unprecedented proportions…
- **…but an academic failure**

Built an Artifact, Not a Discipline

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

- Networking:
  - Teach big bag of protocols
  - Notoriously difficult to manage
  - Evolves very slowly

Why Does Networking Lag Behind?

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

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

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

Two Key Definitions

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

- **Control Plane**: establishing the state in routers
  - Determines how and where packets are forwarded
  - Routing, traffic engineering, firewall state, ...
  - Slow time-scales (per control event)

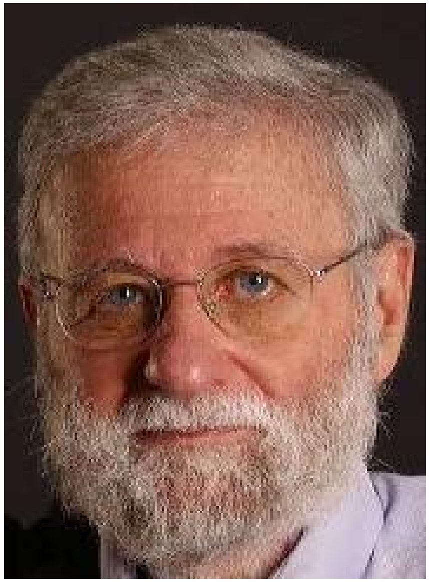
Infrastructure Still Works!

- **Only** because of “our” ability to master complexity

- This ability to master complexity is both a blessing…
  - **…and a curse!**

A Simple Story About Complexity

- ~1985: Don Norman visits Xerox PARC
  - Talks about user interfaces and stick shifts



What Was His Point?

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

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

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

- **You will never succeed in extracting simplicity**
  - If don't recognize it is different from mastering complexity

Scott Shenker's Point

- Networking still focused on mastering complexity
  - Little emphasis on extracting simplicity from control plane
  - No recognition that there's a difference….
- Extracting simplicity builds intellectual foundations
  - **Necessary for creating a discipline….**
  - **That's why networking lags behind**

A Better Example: Programming

- Machine languages: no abstractions
  - Mastering complexity was crucial

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

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

Abstractions key to extracting simplicity

“The Power of Abstraction”

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

Barbara Liskov

Abstractions → Interfaces → Modularity

What abstractions do we have in networking?

Layers are Great Abstractions

- Layers only deal with the **data plane**
- We have no powerful ***control plane*** abstractions!
- How do we find those control plane abstractions?
- Two steps: ***define*** problem, and then ***decompose*** it.

The Network Control Problem

- Compute the configuration of each physical device
  - E.g., Forwarding tables, ACLs,...
- Operate without communication guarantees
- Operate within given network-level protocol

Only people who love complexity would find this a reasonable request

Programming Analogy

- What if programmers had to:
  - Specify where each bit was stored
  - Explicitly deal with all internal communication errors
  - Within a programming language with limited expressability
- Programmers would redefine problem:
  - Define a higher level abstraction for memory
  - Build on reliable communication abstractions
  - Use a more general language
- **Abstractions** divide problem into tractable pieces
  - And make programmer’s task easier

From Requirements to Abstractions

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

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

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

Once these abstractions are in place, control mechanism has much easier job!

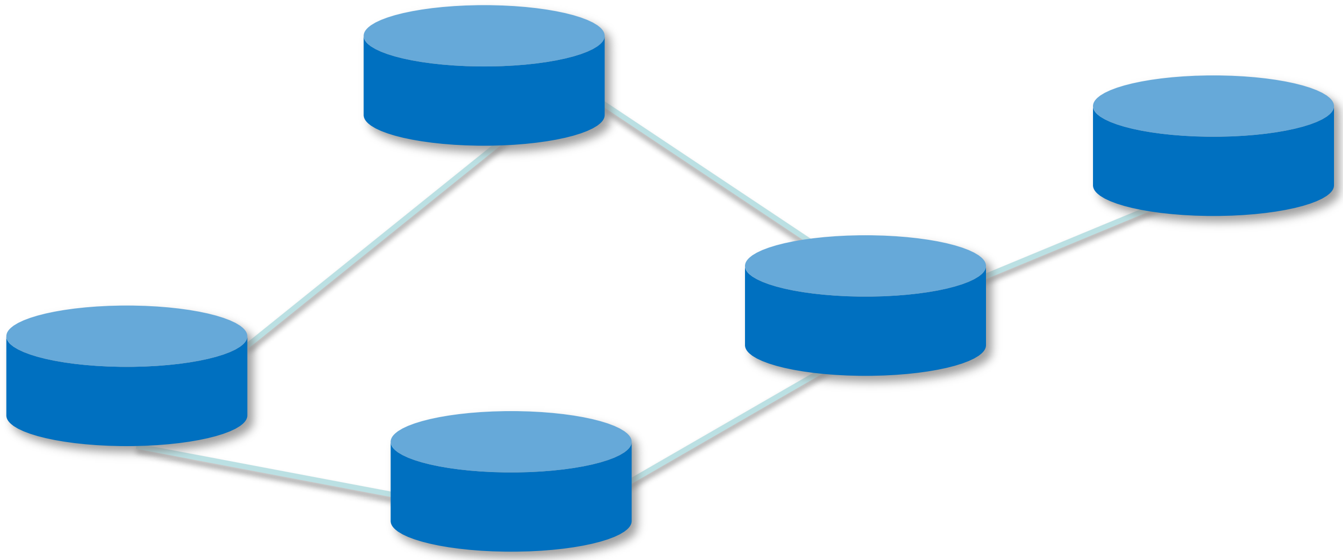
Shenker’s Entire Talk in One Sentence

- SDN is defined *precisely* by these three abstractions
  - Distribution, forwarding, configuration
- SDN not just a random good idea...
  - Fundamental validity and general applicability
- SDN may help us *finally* create a discipline
  - Abstractions enable reasoning about system behavior
  - Provides environment where formalism can take hold....
- OK, but what are these abstractions?

1. Distributed State Abstraction

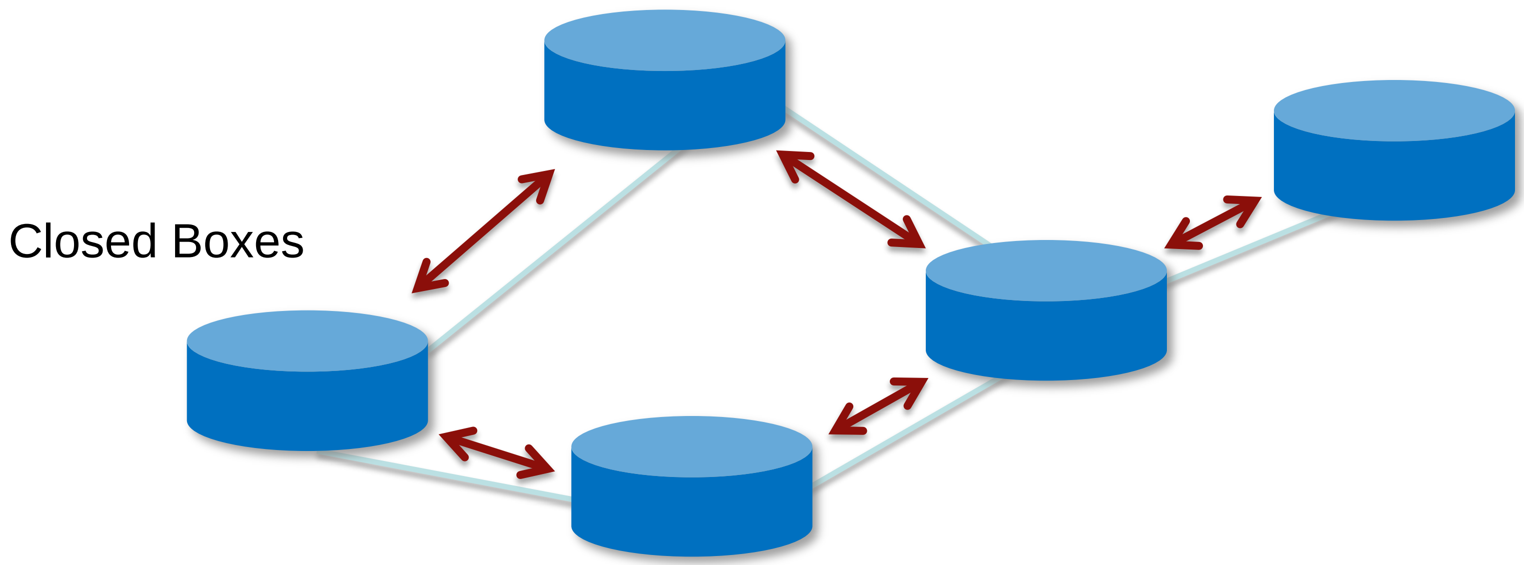
- Shield control mechanisms from state distribution
  - While allowing access to this state
- Natural abstraction: ***global network view***
  - Annotated network graph provided through an API
- Implemented with “Network Operating System”
- Control mechanism is now program using API
  - No longer a distributed protocol, now just a graph algorithm
  - E.g. Use Dijkstra rather than Bellman-Ford

Network of Switches and / or Routers

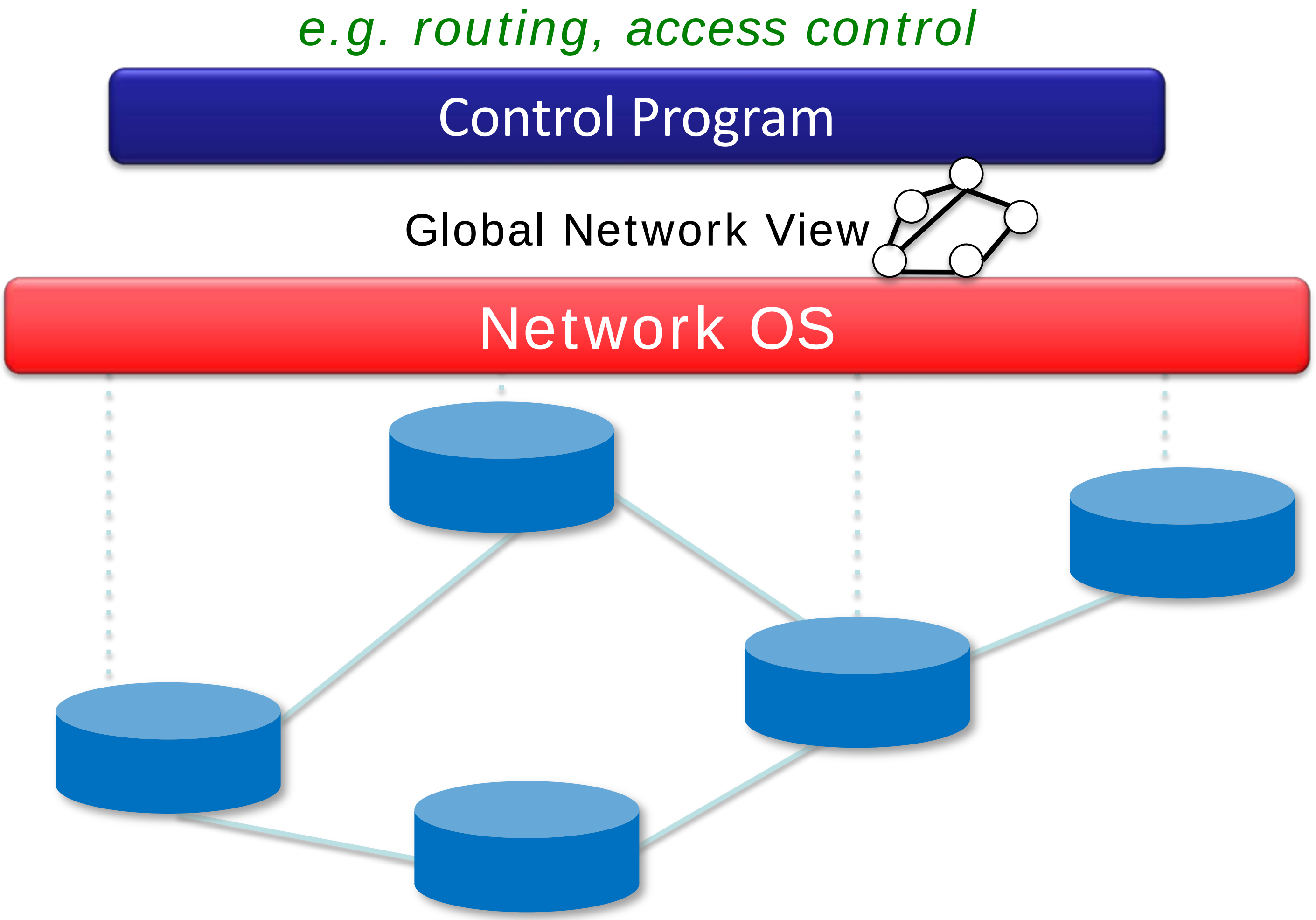


Traditional Control Mechanisms

Distributed algorithm running between neighbors



Software Defined Network (SDN)



Major Change in Paradigm

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

- Now just defining a centralized control ***function***

Configuration = Function(view)

2. Specification Abstraction

- Control program should express desired behavior

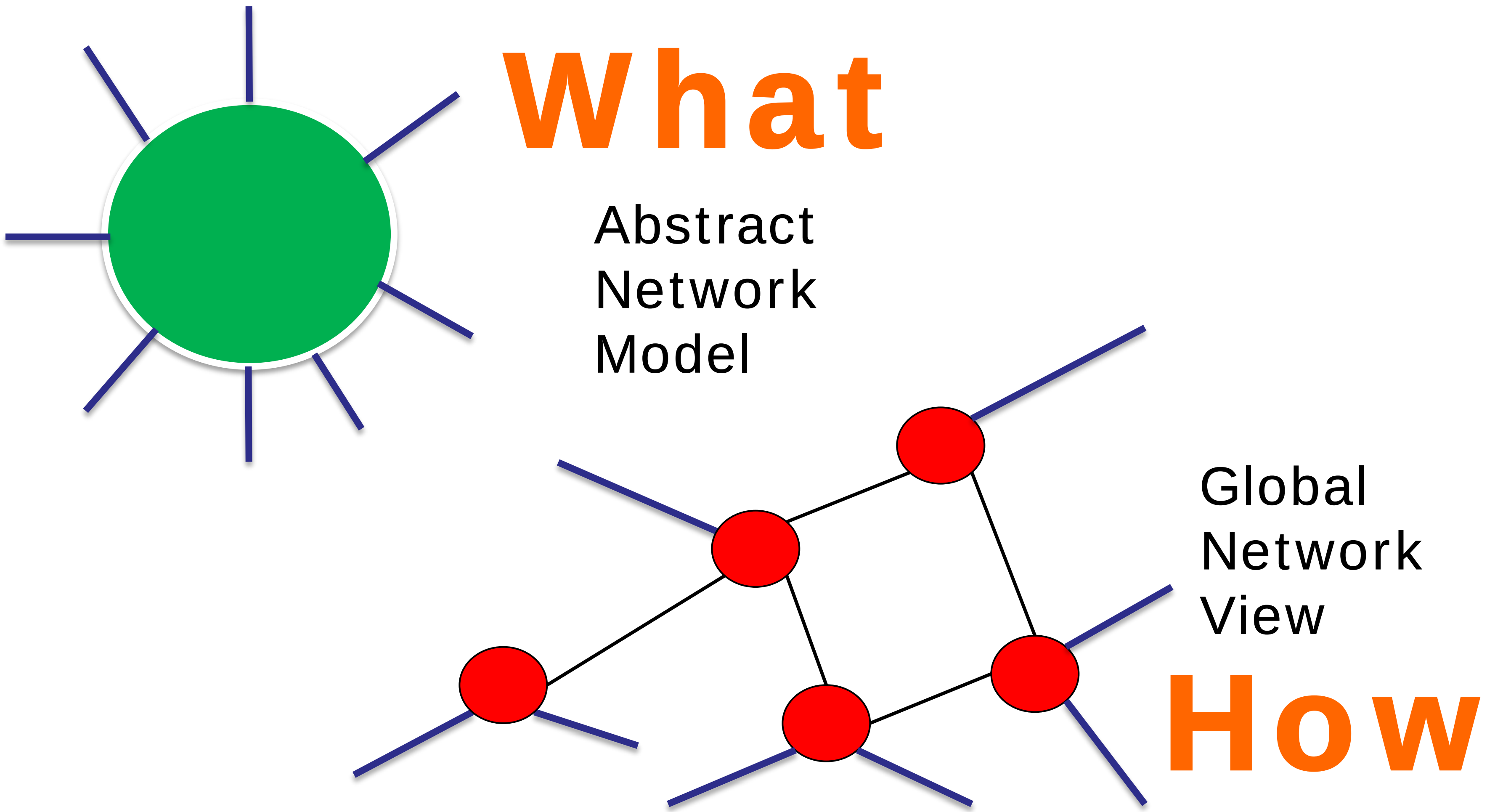
- It should not be responsible for implementing that behavior on physical network infrastructure

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

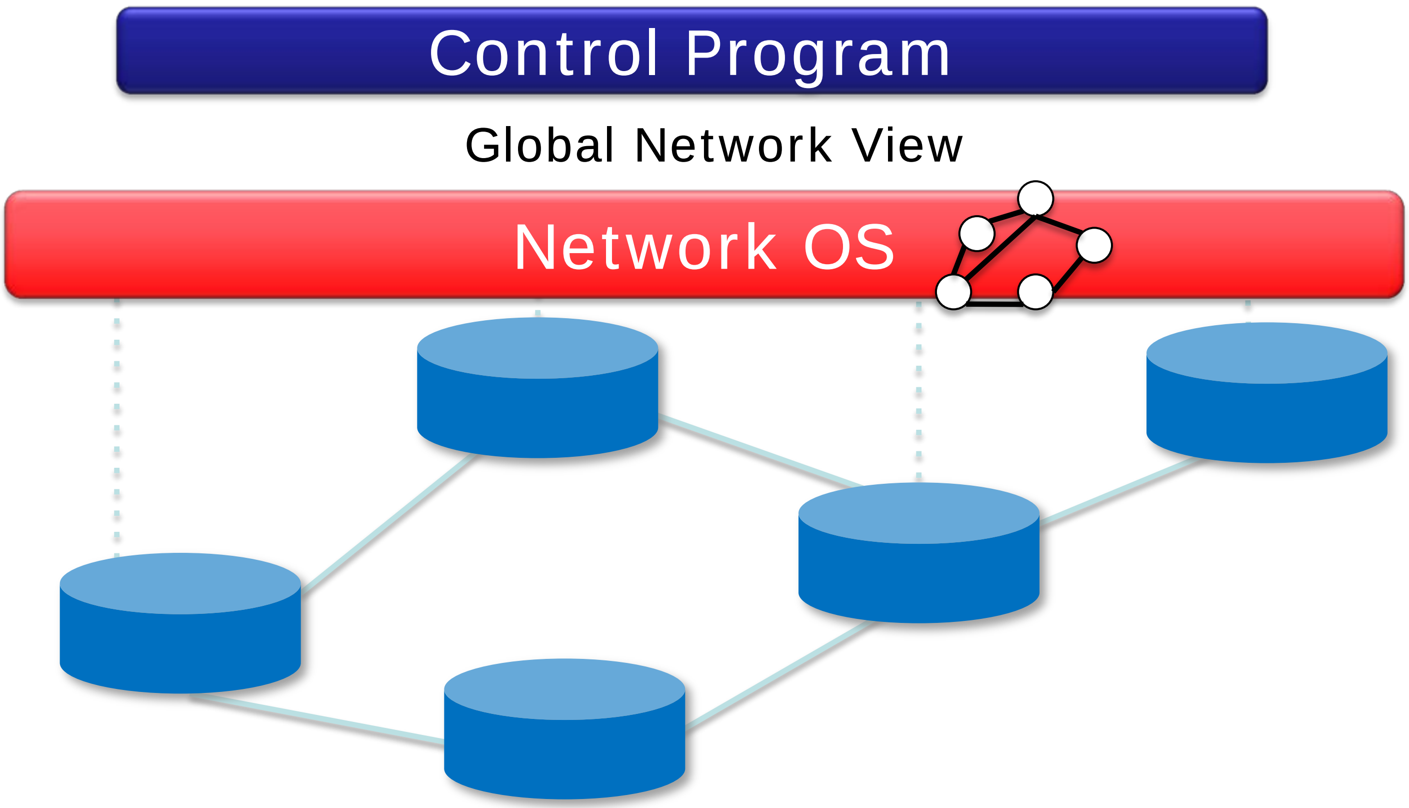
- Requires a new shared control layer:
  - **Map abstract configuration to physical configuration**

- This is “network virtualization”

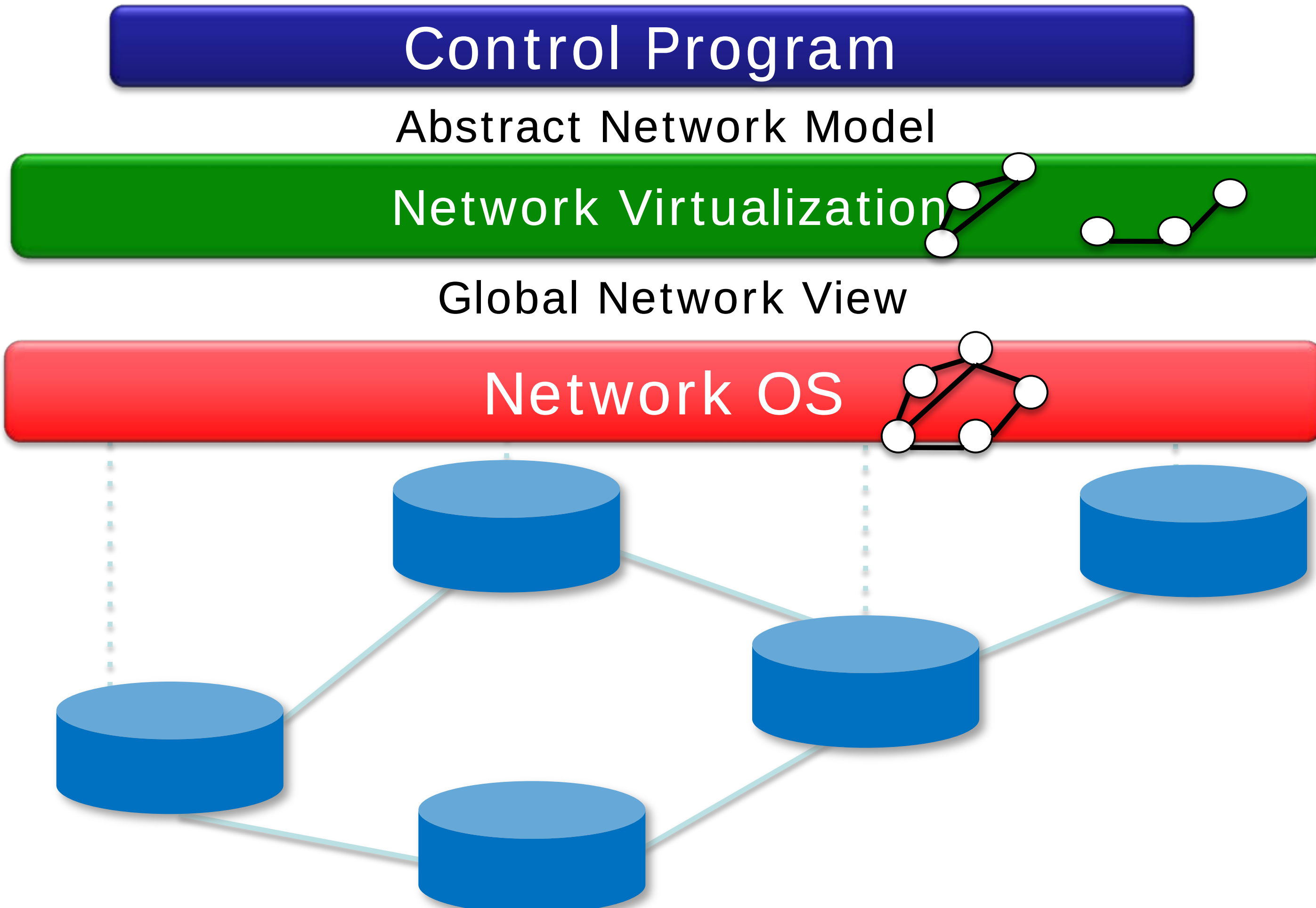
Simple Example: Access Control



Software Defined Network: Take 2



Software Defined Network: Take 2

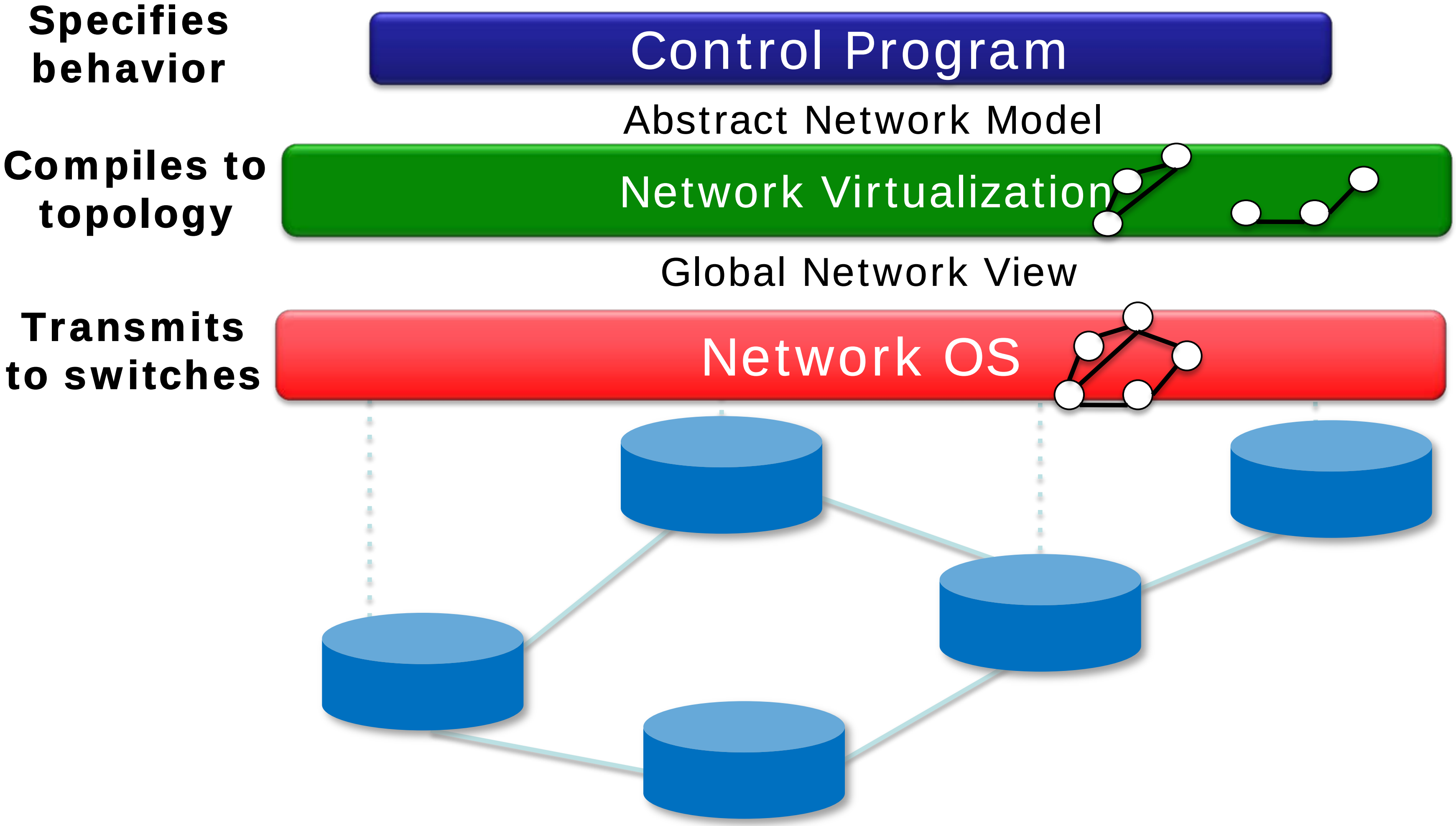


What Does This Picture Mean?

- Write a simple program to configure a simple model
  - Configuration merely a way to specify what you want
- Examples
  - ACLs: who can talk to who
  - Isolation: who can hear my broadcasts
  - Routing: only specify routing to the degree you care
    - Some flows over satellite, others over landline
  - TE: specify in terms of quality of service, not routes
- Virtualization layer “compiles” these requirements
  - Produces suitable configuration of actual network devices
- NOS then transmits these settings to physical boxes

Software Defined Network: Take 2

TECHNISCHE UNIVERSITÄT DARMSTADT



Two Examples Uses

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

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

- Hard to do without SDN, easy *(in principle)* with SDN

3. Forwarding Abstraction

- Switches have two “brains”
  - Management CPU (smart but slow)
  - Forwarding ASIC (fast but dumb)
- Need a forwarding abstraction for both
  - CPU abstraction can be almost anything
- ASIC abstraction is much more subtle: **OpenFlow**
- OpenFlow:
  - Control switch by inserting <header;action> entries
  - Essentially gives NOS remote access to forwarding table
  - Instantiated in OpenvSwitch

Does SDN Work?

- Is it scalable? **Yes**
- Is it less responsive? **No**
- Does it create a single point of failure? **No**
- Is it inherently less secure? **No**
- Is it incrementally deployable? **Yes**

SDN: Clean Separation of Concerns

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

- **Net Virt'n: map abstract model to global view**
  - Driven by **Specification Abstraction**

- **NOS: map global view to physical switches**
  - API: driven by **Distributed State Abstraction**
  - Switch/fabric interface: driven by **Forwarding Abstraction**

We Have Achieved Modularity!

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

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

- ***This is why SDN is the future of networking…***

Status of SDN

- Open Networking Foundation is standards body
  - SDN endorsed by 49 companies
  - Almost everyone who matters…..
- A few products on market, many more coming
  - Some large companies using SDN internally
- **SDN has won the war of words, the real battle over customer adoption is just beginning….**