Software Defined Networking

ECE/CS598HPN

Radhika Mittal

Acknowledgements: Scott Shenker, UC Berkeley

Software-defined Networking

- My favorite example for the impact of academic research.

- Transformed the way we manage networks.
  - Particularly within and across datacenters.

What is Network Management?

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

Original goals for the control plane

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

Isolation

- L2 broadcast protocols often used for discovery
  - Useful, unscalable, invasive
- Want multiple logical LANs on a physical network
  - Retain usefulness, cope with scaling, provide isolation
- Use VLANs (virtual LANs) tags in L2 headers
  - Controls where broadcast packets go
  - Gives support for logical L2 networks
  - Routers connect these logical L2 networks
- No universal method for setting VLAN state

Access Control

- Operators want to limit access to various hosts
  - "Don't let laptops access backend database machines"
- This can be imposed by routers using ACLs
  - ACL: Access Control List
- Example entry in ACL: <header template; drop>
  - If not port 80, drop
  - If source address = X, drop

Traffic Engineering

- Want to avoid persistent overloads on links
- Choose routes to spread traffic load across links
- Two main methods:
  - Setting up MPLS tunnels *(MPLS is layer 2.5)*
  - Adjusting weights in OSPF
- Often done with centralized computation
  - Take snapshot of topology and load
  - Compute appropriate MPLS/OSPF state
  - Send to network

Net management has many goals

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

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

Managing networks is extremely complicated!

- Many different control plane mechanisms
- Each designed from scratch for their intended goal
- Encompassing a wide variety of implementations
  - Distributed, manual, centralized,…
- And none of them particularly well designed

Adding to the complications

- When running distributed algorithms and protocols.
  - Need to deal with standardization and interoperability.
- When configuring individual network devices.
  - Interface varies across vendors and protocols.
- Indirect control.
  - Policy specification had to workaround existing routing mechanisms.

How have we managed to survive?

- Net. admins miraculously master this complexity
  - Understand all aspects of networks
  - Must keep myriad details in mind

- No longer possible…..

Large datacenters

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

Multiple tenancy

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

Net Operators Were Now Weeping…

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

Abstractions and Layering!

Network Abstractions

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

Abstractions for Data Plane: Layers

Applications

…built on…

Reliable (or unreliable) transport

…built on…

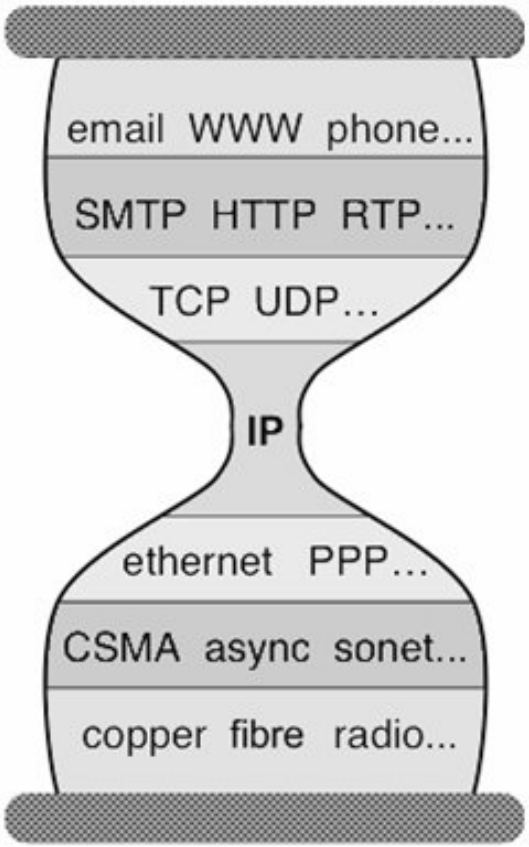
Best-effort global packet delivery

…built on…

Best-effort local packet delivery

…built on…

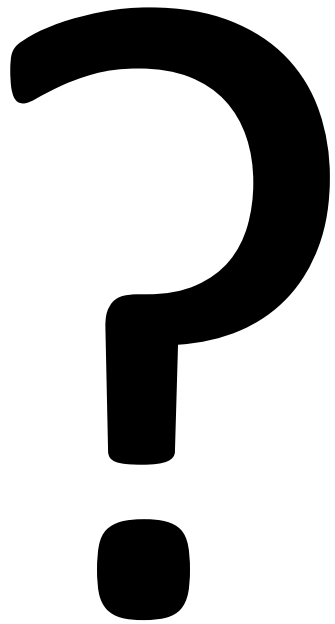
Physical transfer of bits



The Importance of Layering

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

Control Plane Abstractions



Control Plane Task

Compute forwarding state.

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

Our Pre-SDN approach

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

Separate Concerns with Abstractions

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

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

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

Abs#1: Forwarding Abstraction

- Express intent independent of implementation
  - Don't want to deal with proprietary HW and SW
- OpenFlow is current proposal for forwarding
  - Standardized interface to switch
  - We will discuss in next class.

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

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

Abs#2: Network State Abstraction

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

Software-Defined Networking (SDN)

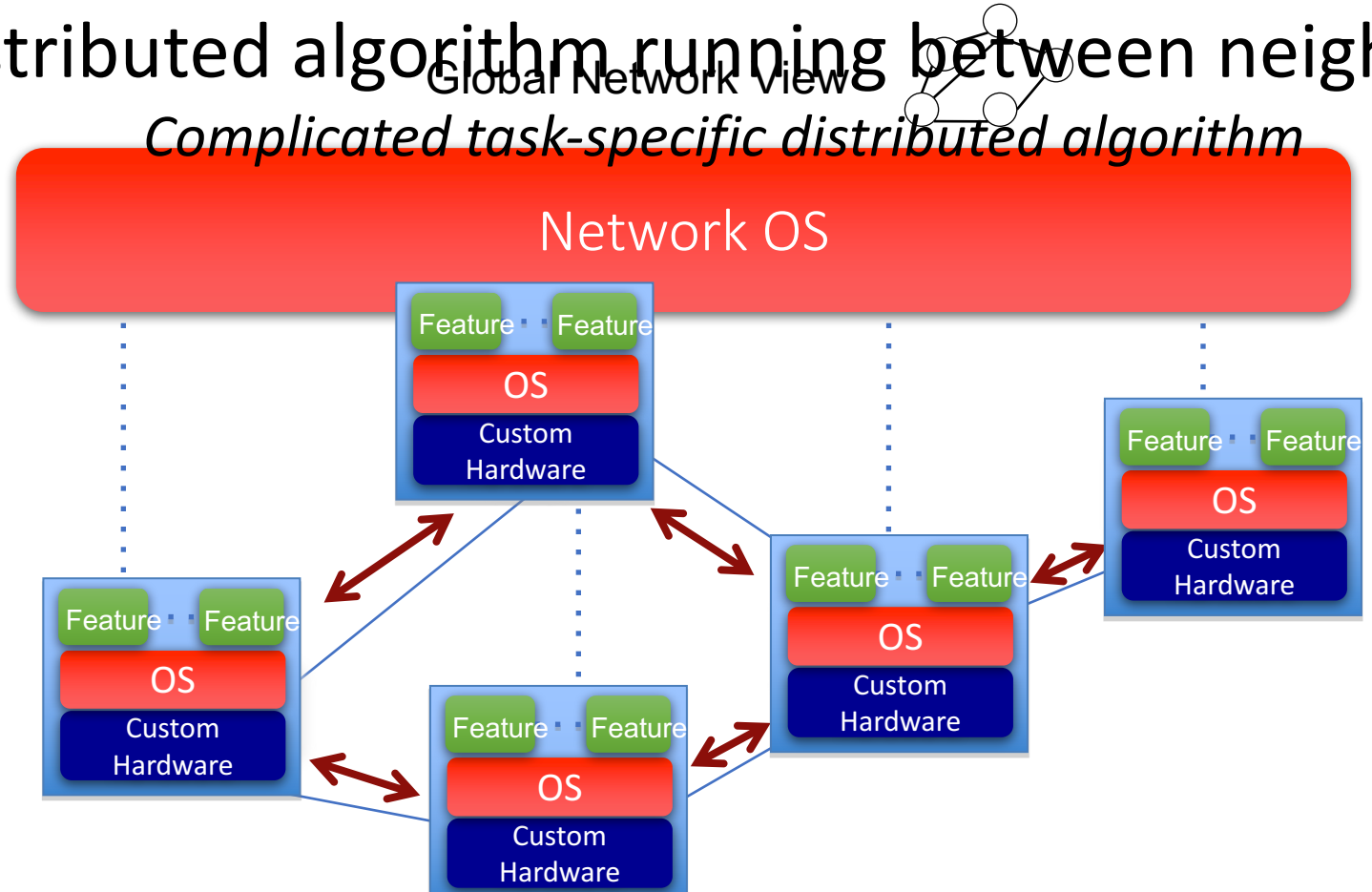
routing, access control, etc.

Control Program

Distributed algorithm running between neighbors

Global Network View

Complicated task-specific distributed algorithm



Implication of global network view

- We can now use centralized algorithms!
- Example: link-state routing
  - Dijkstra's algorithm: 4 pages
  - OSPF: RFC 2328, 245 pages

Thanks to Nick McKeown

Key idea behind SDN

- Separate *control plane* from *data plane*.
- Data plane: hardware that handles packet forwarding on individual switches.
- Control plane: centralized software that remotely and directly controls switch hardware.

Separate Concerns with Abstractions

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

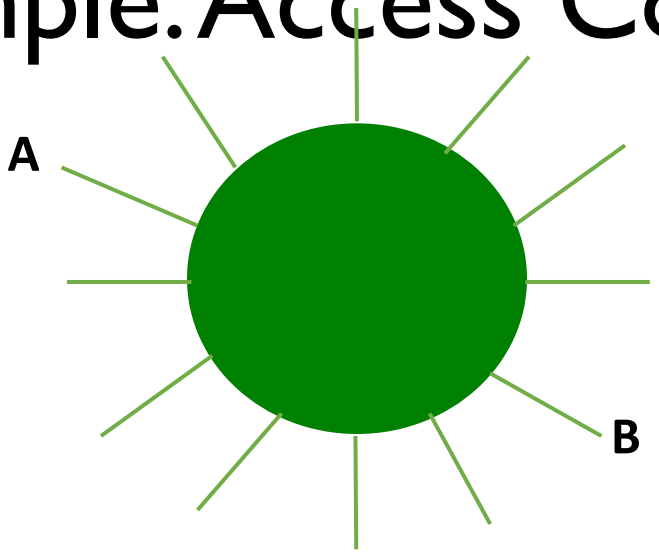
2. Make decisions based on entire network
   Need an abstraction for network state

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

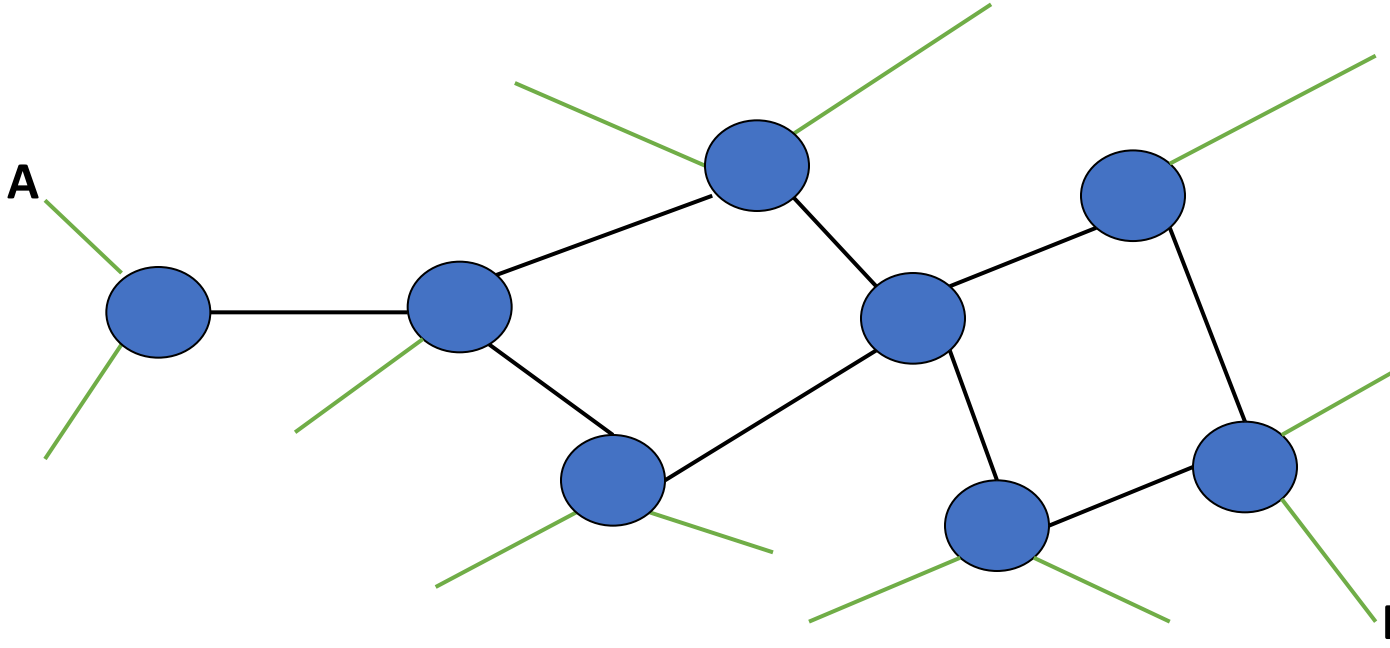
Abs#3: Specification Abstraction

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

Simple Example: Access Control

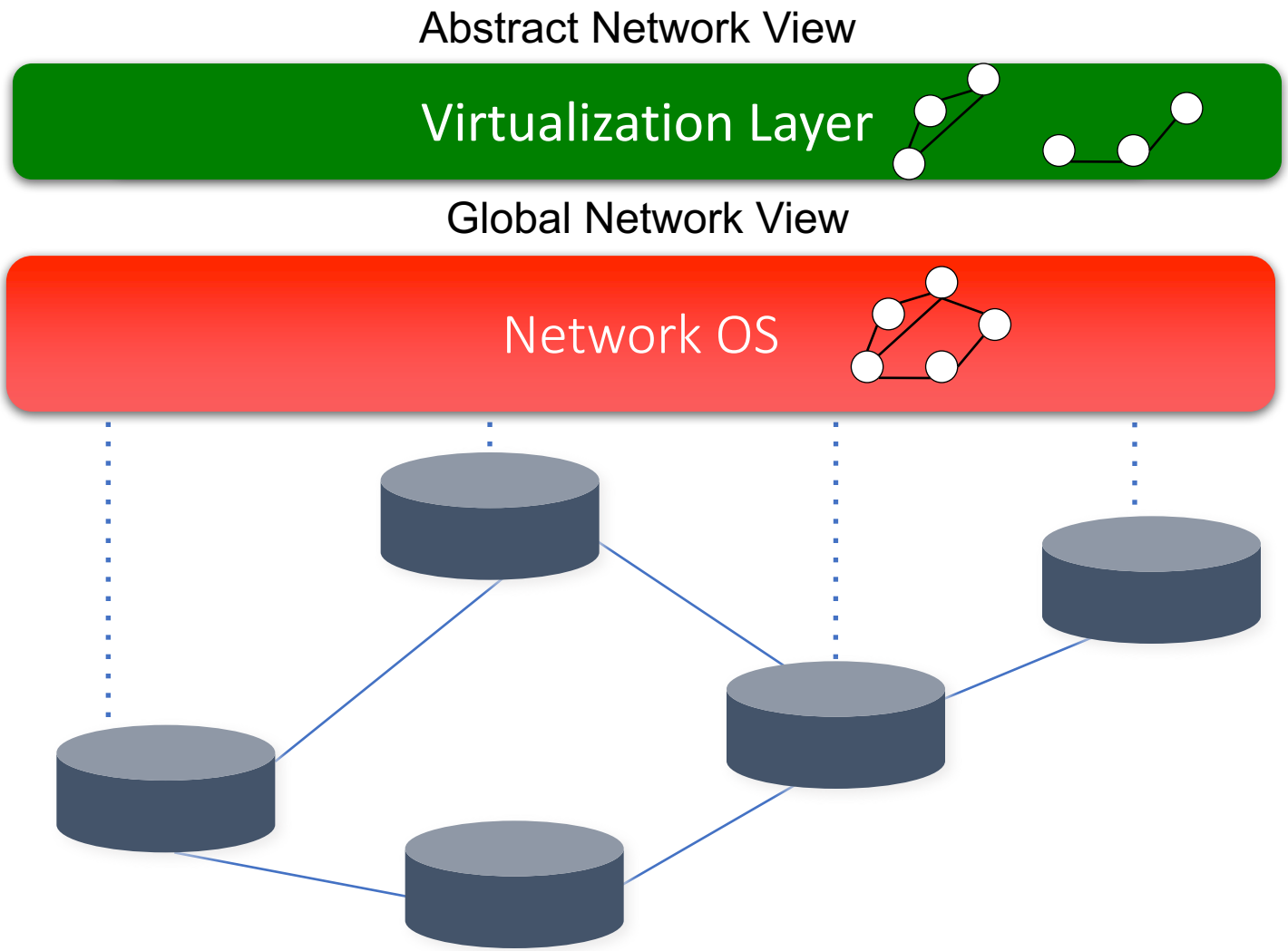


Abstract
Network
View



Global
Network
View

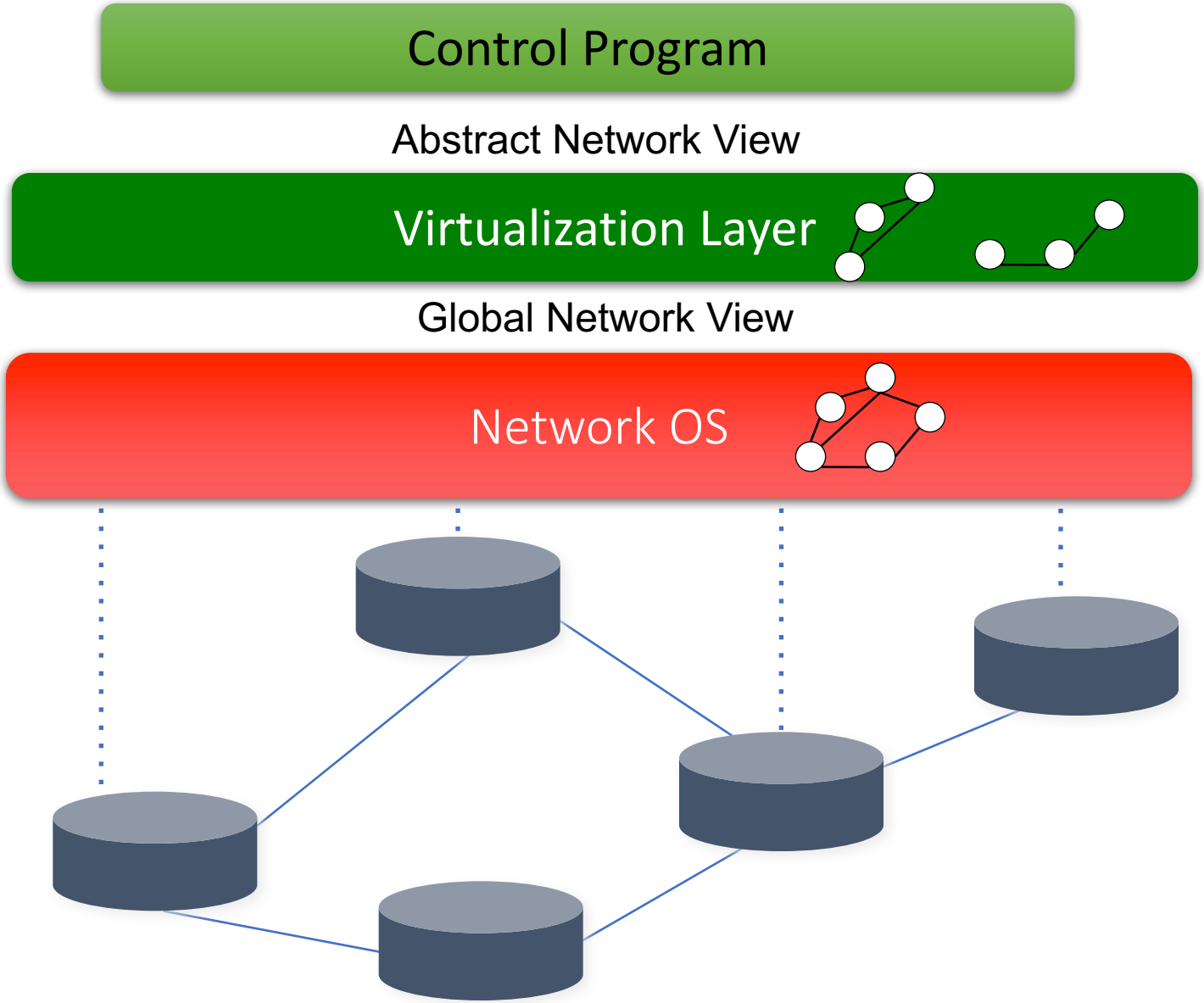
Software Defined Network



Clean Separation of Concerns

- **Control program:** express goals on abstract view
  - *Driven by Operator Requirements*
- **Virt. 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



Key idea behind SDN

- Separate *control plane* from *data plane*.
- Data plane: hardware that handles packet forwarding on individual switches.
- Control plane: centralized software that remotely and directly controls switch hardware.
  - Use software design principles to modularize the control plane.

When and how did it start?

2D HotNets (2004)

- Switches have bulky control plane running distributed algorithms.
- Separate management plane.
- Difficult to configure and manage network state.
- Based on 2D's:
  - Decision plane
  - Dissemination plane
- Wafer-thin control plane on switch.

Network-Wide Decision Making: Toward A Wafer-Thin Control Plane*

Jennifer Rexford, Albert Greenberg, Gisli Hjalmtysson
{jrex,albert,gisli}@research.att.com
AT&T Labs–Research

David A. Maltz, Andy Myers, Geoffrey Xie, Jibin Zhan, Hui Zhang
{dmaltz,acm, geoffxie,jibin,hzhang}@cs.cmu.edu
Carnegie Mellon University

Abstract

We argue for the refactoring of the IP control plane to provide direct expressibility and support for network-wide goals relating to all fundamental functionality: reachability, performance, reliability and security. This refactoring is motivated by trends in operational practice and in networking technology. We put forward a design that decomposes functionality into information *dissemination* and *decision* planes. The decision plane is formed by lifting out of the routers all decision making logic currently found there and merging it with the current management plane where network-level objectives are specified. What is left on each router is a wafer-thin control plane focused on information dissemination and response to explicit instructions for configuring packet forwarding mechanisms. We discuss the consequences, advantages and challenges associated with this design.

1. Introduction

Despite the early design goal of minimizing the state in network elements, tremendous amounts of state are distributed across routers and management platforms in today's IP networks. We believe that the many, loosely-coordinated actors that create and manipulate the distributed state introduce substantial complexity that makes both backbone and enterprise networks increasingly fragile and difficult to manage. In this paper, we argue that the current division of functionality across the data, control, and management planes is antithetical to the desire for network-wide control. Instead, we advocate moving the *decision logic* for running the network from the individual routers into the management system. In our framework, the routers simply *disseminate* timely information about the network and respond to explicit instructions for configuring the packet forwarding behavior.

We argue that our approach will significantly reduce the complexity of IP routers while making the resulting network easier to manage. We first describe the status quo, and then present and contrast our design. Then, we give concrete examples of how operators are forced to run their networks today and how they could be better served by our design, and we end by considering the challenges facing our design.

1.1 Today's Data, Control, and Management Planes

State distributed across interconnected routers defines how a network "works." Yet, our understanding of how this state is created and maintained (and, perhaps more importantly, how it *should* be created and maintained) is surprisingly limited. Just as great care went in to splitting the Internet's functionality between the smart edge devices (such as end host computers) and the "dumb" core devices (such as routers), we need to revisit the separation of functionality between the three "planes" that affect the operation of an IP network:

- **Data plane:** The data plane is local to an individual router, or even a single interface card on the router, and operates at the speed of packet arrivals. For example, the data plane performs packet forwarding, including the longest-prefix match that identifies the outgoing link for each packet, as well as the access control lists (ACLs) that filter packets based on their header fields. The data plane also implements functions such as tunneling, queue management, and packet scheduling.
- **Control plane:** The control plane consists of the network-wide distributed algorithms that compute parts of the state in the data plane. For example, the control plane includes BGP update messages and the BGP decision process, as well as the Interior Gateway Protocol (such as OSPF), its link-state advertisements (LSAs), and the Dijkstra's shortest-path algorithm. A primary job of the control plane is to compute routes between IP subnets, including combining information from each routing protocol's Routing Information Base (RIB) to construct a single Forwarding Information Base (FIB) that drives packet forwarding decisions.
- **Management plane:** The management plane stores and analyzes measurement data from the network and generates the configuration state on the individual routers. For example, the management plane collects and combines SNMP (Simple Network Management Protocol) statistics, traffic flow records, OSPF LSAs, and BGP update streams. A tool that configures the OSPF link weights and BGP policies to satisfy traffic engineering goals would be part of the management plane. Similarly, a system that analyzes traffic measurements to detect denial-of-service attacks and configures ACLs to block offending traffic would be part of the management plane.

In today's IP networks, the data plane operates at the timescale of packets and the spatial scale of individual routers, the control plane operates at the of timescale of seconds with an incomplete view of the entire network, and the management plane operates at the timescale of minutes or hours and the spatial scale of the entire network.

In this paper, we argue that this three-level division of functionality leads to complex decision logic split across multiple

*Research sponsored by the NSF under ANI-0085920 and ANI-0331653. Views and conclusions contained in this document are those of the authors.

Routing Control Platform (2005)

- iBGP: protocol to exchange routes to external destination within AS.
- Scalability concerns.
- Prone to forwarding loops and oscillations.
- RCP: centralized platform for iBGP routing.

Design and Implementation of a Routing Control Platform

Matthew Caesar, *UC Berkeley*
Donald Caldwell, *AT&T Labs-Research*
Nick Feamster, *MIT*
Jennifer Rexford, *Princeton University*
Aman Shaikh, *AT&T Labs-Research*
Jacobus van der Merwe, *AT&T Labs-Research*

Abstract

The routers in an Autonomous System (AS) must distribute the information they learn about how to reach external destinations. Unfortunately, today's internal Border Gateway Protocol (iBGP) architectures have serious problems: a "full mesh" iBGP configuration does not scale to large networks and "route reflection" can introduce problems such as protocol oscillations and persistent loops. Instead, we argue that a Routing Control Platform (RCP) should collect information about external destinations and internal topology and select the BGP routes for each router in an AS. RCP is a logically-centralized platform, separate from the IP forwarding plane, that performs route selection on behalf of routers and communicates selected routes to the routers using the unmodified iBGP protocol. RCP provides scalability without sacrificing correctness. In this paper, we present the design and implementation of an RCP prototype on commodity hardware. Using traces of BGP and internal routing data from a Tier-1 backbone, we demonstrate that RCP is fast and reliable enough to drive the BGP routing decisions for a large network. We show that RCP assigns routes correctly, even when the functionality is replicated and distributed, and that networks using RCP can expect comparable convergence delays to those using today's iBGP architectures.

1 Introduction

The Border Gateway Protocol (BGP), the Internet's interdomain routing protocol, is prone to protocol oscillation and forwarding loops, highly sensitive to topology changes inside an Autonomous System (AS), and difficult for operators to understand and manage. We address these problems by introducing a Routing Control Platform (RCP) that computes the BGP routes for each router in an AS based on complete routing information and higher-level network engineering goals [1, 2]. This paper describes the design and implementation of an RCP prototype that is fast and reliable enough to coordinate routing for a large backbone network.

1.1 Route Distribution Inside an AS

The routers in a single AS exchange routes to external destinations using a protocol called internal BGP (iBGP). Small networks are typically configured as a "full mesh" iBGP topology, with an iBGP session between each pair of routers. However, a full-mesh configuration does not scale because each router must: (i) have an iBGP session with every other router, (ii) send BGP update messages to every other router, (iii) store a local copy of the advertisements sent by each neighbor for each destination prefix, and (iv) have a new iBGP session configured whenever a new router is added to the network. Although having a faster processor and more memory on every router would support larger full-mesh configurations, the installed base of routers lags behind the technology curve, and upgrading routers is costly. In addition, BGP-speaking routers do not always degrade gracefully when their resource limitations are reached; for example, routers crashing or experiencing persistent routing instability under such conditions have been reported [3]. In this paper, we present the design, implementation, and evaluation of a solution that behaves like a full-mesh iBGP configuration with much less overhead and no changes to the installed base of routers.

To avoid the scaling problems of a full mesh, today's large networks typically configure iBGP as a hierarchy of *route reflectors* [4]. A route reflector selects a single BGP route for each destination prefix and advertises the route to its clients. Adding a new router to the system simply requires configuring iBGP sessions to the router's route reflector(s). Using route reflectors reduces the memory and connection overhead on the routers, at the expense of compromising the behavior of the underlying network. In particular, a route reflector does not necessarily select

4D (2005)

- Difficult to configure management policies (e.g. access control).
- 4D architecture.

A Clean Slate 4D Approach to Network Control and Management *

Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers,
Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, Hui Zhang
{dmaltz,acm,yh,jibin,hzhang}@cs.cmu.edu
gisli@ru.is jrex@cs.princeton.edu albert@research.att.com xie@nps.edu

ABSTRACT

Today's data networks are surprisingly fragile and difficult to manage. We argue that the root of these problems lies in the complexity of the control and management planes—the software and protocols coordinating network elements—and particularly the way the decision logic and the distributed-systems issues are inexorably intertwined. We advocate a complete refactoring of the functionality and propose three key principles—network-level objectives, network-wide views, and direct control—that we believe should underlie a new architecture. Following these principles, we identify an extreme design point that we call "4D," after the architecture's four planes: decision, dissemination, discovery, and data. The 4D architecture completely separates an AS's decision logic from protocols that govern the interaction among network elements. The AS-level objectives are specified in the decision plane, and enforced through direct configuration of the state that drives how the data plane forwards packets. In the 4D architecture, the routers and switches simply forward packets at the behest of the decision plane, and collect measurement data to aid the decision plane in controlling the network. Although 4D would involve substantial changes to today's control and management planes, the format of data packets does not need to change; this eases the deployment path for the 4D architecture, while still enabling substantial innovation in network control and management. We hope that exploring an extreme design point will help focus the attention of the research and industrial communities on this crucially important and intellectually challenging area.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Packet Switching Networks; C.2.2 [**Network Protocols**]: Routing Protocols; C.2.3 [**Network Operations**]: Network Management

*This research was sponsored by the NSF under ITR Awards ANI-0085920 and ANI-0331653. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of AT&T, NSF, or the U.S. government.

General Terms

Measurement, Control, Performance, Reliability

Keywords

Network management, robustness, control

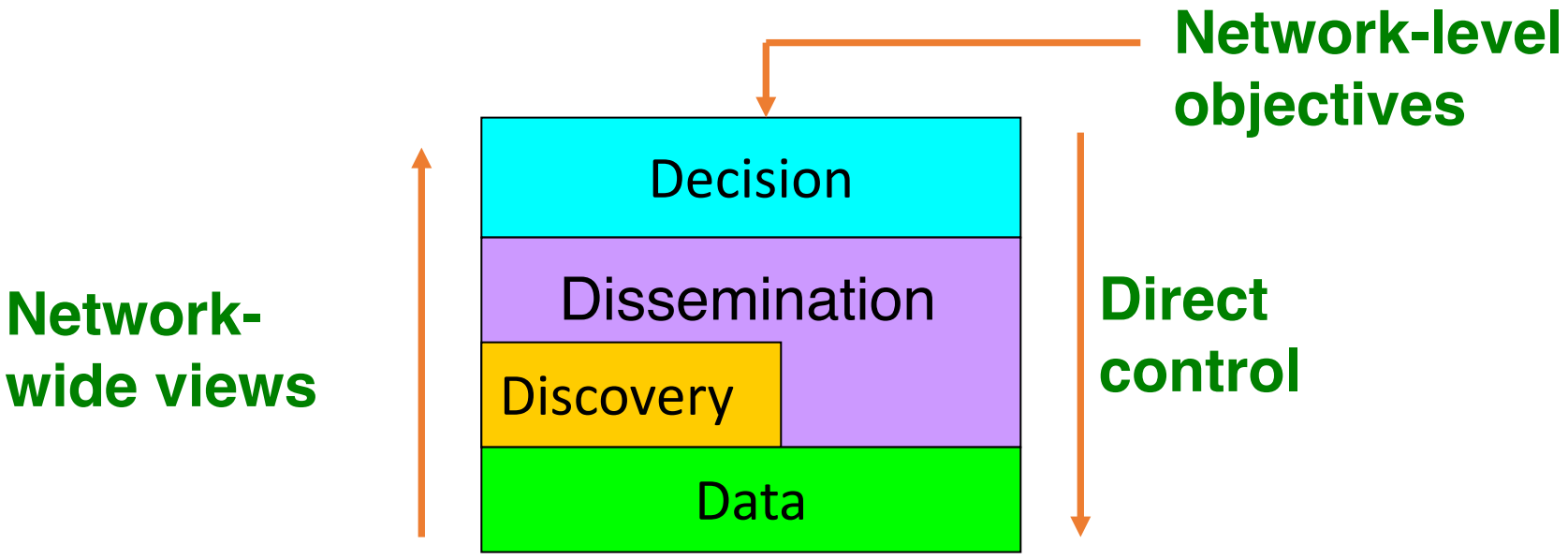
1. INTRODUCTION

Although IP networking has been wildly successful, there are serious problems lurking "under the hood." IP networks exhibit a defining characteristic of unstable complex systems—a small local event (e.g., misconfiguration of a routing protocol on a single interface) can have severe, global impact in the form of a cascading meltdown. In addition, individual Autonomous Systems (ASes) must devote significant resources to "working around" the constraints imposed by today's protocols and mechanisms to achieve their goals for traffic engineering, survivability, security, and policy enforcement. We believe the root cause of these problems lies in the control plane running on the network elements and the management plane that monitors and configures them. In this paper, we argue for revisiting the division of functionality and advocate an extreme design point that completely separates a *network's decision logic* from the *the protocols that govern interaction of network elements* We initially focus our attention on the operation of a single Autonomous System (AS), though we also discuss how multiple ASes can coordinate their actions.

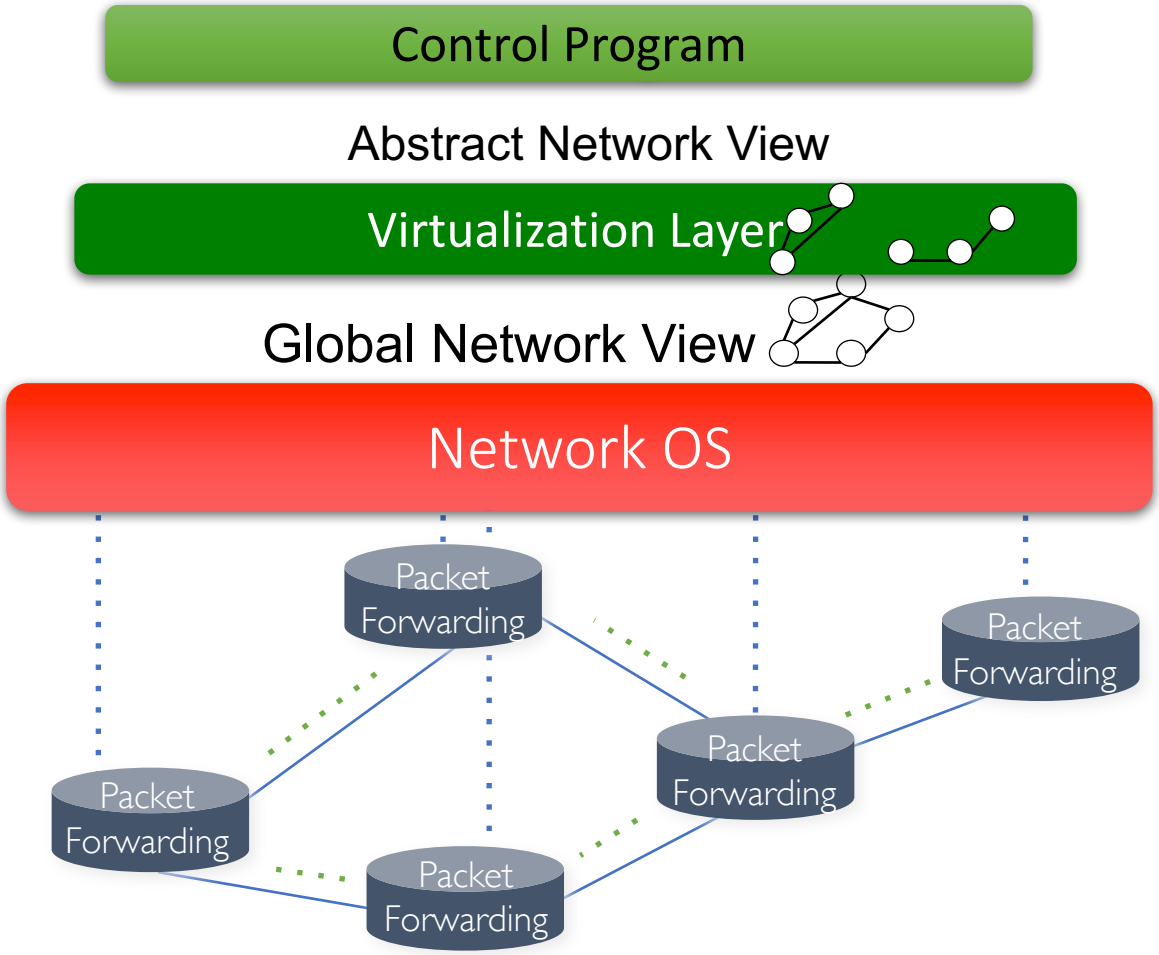
The Internet architecture bundles control logic and packet handling into the individual routers and switches distributed throughout an AS. As a result, each router/switch[1] participates in distributed protocols that implicitly *embed* the decision logic. For example, in IP networks, the path-computation logic is governed by distributed protocols such as OSPF, IS-IS, and EIGRP. The routing protocols dictate not only how the routers learn about the topology, but also how they select paths. Similarly, in Ethernet networks, the path-computation logic is embedded in the Spanning Tree protocol [1]. However, today's data networks, operated by numerous institutions and deployed in diverse environments, must support network-level objectives and capabilities far more sophisticated than best-effort packet delivery. These ever-evolving requirements have led to incremental changes in the control-plane protocols, as well as complex management-plane software that tries to "coax" the control plane into satisfying the network objectives. The resulting complexity is responsible for the increasing fragility of IP networks and the tremendous difficulties facing people trying to understand and manage their networks.

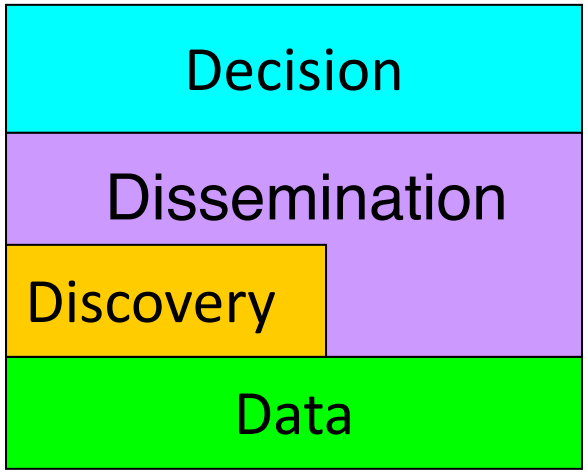
[1]We use the terms "network element" and "router/switch" interchangeably throughout the paper.

4D (2005)


Network-level objectives
Decision
Dissemination
Discovery
Data
Network-wide views
Direct control

Software Defined Network (SDN)





SANE (2006)

- Security achieved via ACLs, packet filters, firewalls, NATs, VLANs.
- Prone to misconfigurations due to human-error.
- Architecture support for:
  - Natural, simple, policy expression.
  - One trusted component.
- A Domain Controller that grants source routing capability.

SANE: A Protection Architecture for Enterprise Networks

Martin Casado, Tal Garfinkel, Aditya Akella, Michael J. Freedman
Dan Boneh, Nick McKeown, Scott Shenker
{*casado,talg,mfreed,dabo,nickm*}*@cs.stanford.edu*
aditya@cs.cmu.edu, shenker@icsi.berkeley.edu

Abstract

Connectivity in today's enterprise networks is regulated by a combination of complex routing and bridging policies, along with various interdiction mechanisms such as ACLs, packet filters, and other middleboxes that attempt to retrofit access control onto an otherwise permissive network architecture. This leads to enterprise networks that are inflexible, fragile, and difficult to manage.

To address these limitations, we offer SANE, a protection architecture for enterprise networks. SANE defines a single *protection layer* that governs all connectivity within the enterprise. All routing and access control decisions are made by a logically-centralized server that grants access to services by handing out capabilities (encrypted source routes) according to declarative access control policies (e.g., "Alice can access http server *foo*"). Capabilities are enforced at each switch, which are simple and only minimally trusted. SANE offers strong attack resistance and containment in the face of compromise, yet is practical for everyday use. Our prototype implementation shows that SANE could be deployed in current networks with only a few modifications, and it can easily scale to networks of tens of thousands of nodes.

1 Introduction

The Internet architecture was born in a far more innocent era, when there was little need to consider how to defend against malicious attacks. Moreover, many of the Internet's primary design goals, such as universal connectivity and decentralized control, which were so critical to its success, are at odds with making it secure.

Worms, malware, and sophisticated attackers mean that security can no longer be ignored. This is particularly true for enterprise networks, where it is unacceptable to lose data, expose private information, or lose system availability. And so security measures have been retrofitted to enterprise networks via many mechanisms, including router ACLs, firewalls, NATs, and other middleboxes, along with complex link-layer technologies such as VLANs.

Despite years of experience and experimentation, these mechanisms are far from ideal. They require a significant amount of configuration and oversight [43], are often limited in the range of policies they can enforce [45], and produce networks that are complex [49] and brittle [50]. Moreover, even with these techniques, security within the enterprise remains notoriously poor. Worms routinely cause significant losses in productivity [9] and potential for data loss [29, 34]. Attacks resulting in theft of intellectual property and other sensitive information are similarly common [19].

The long and largely unsuccessful struggle to protect enterprise networks convinced us to start over with a clean slate, with security as a fundamental design goal. The result is our *Secure Architecture for the Networked Enterprise (SANE)*. The central design goals for our architecture are as follows:

- *Allow natural policies that are simple yet powerful*. We seek an architecture that supports natural policies that are independent of the topology and the equipment used, e.g., "Allow everyone in group sales to connect to the http server hosting documentation." This is in contrast to policies today that are typically expressed in terms of topology-dependent ACLs in firewalls. Through high-level policies, our goal is to provide access control that is restrictive (i.e., provides least privilege access to resources), yet flexible, so the network does not become unusable.
- *Enforcement should be at the link layer, to prevent lower layers from undermining it*. In contrast, it is common in today's networks for network-layer access controls (e.g., ACLs in firewalls) to be undermined by more permissive connectivity at the link layer (e.g., Ethernet and VLANs).

Ethane (2007)

- SANE was difficult to deploy.
- Incrementally deployable:
  - No endhost modifications.
  - Centralized controller manage simple dumb "Ethane switches".
  - Co-exist with regular switches.
- Flow-based policy decision.
- Ethane switch ⟶ OpenFlow

Ethane: Taking Control of the Enterprise

Martìn Casado, Michael J. Freedman,
Justin Pettit, Jianying Luo,
and Nick McKeown
Stanford University

Scott Shenker
U.C. Berkeley and ICSI

ABSTRACT

This paper presents Ethane, a new network architecture for the enterprise. Ethane allows managers to define a single network-wide fine-grain policy, and then enforces it directly. Ethane couples extremely simple flow-based Ethernet switches with a centralized controller that manages the admittance and routing of flows. While radical, this design is backwards-compatible with existing hosts and switches.

We have implemented Ethane in both hardware and software, supporting both wired and wireless hosts. Our operational Ethane network has supported over 300 hosts for the past four months in in Stanford University's network, and this deployment experience has significantly affected Ethane's design.

Categories and Subject Descriptors

C.2.6 [**Computer Communication Networks**]: Internetworking; C.2.1 [**Computer Communication Networks**]: Network Architecture and Design

General Terms

Design, Experimentation, Performance

Keywords

Network, Architecture, Security, Management

1. INTRODUCTION

Enterprise networks are often large, run a wide variety of applications and protocols, and typically operate under strict reliability and security constraints; thus, they represent a challenging environment for network management. The stakes are high, as business productivity can be severely hampered by network misconfigurations or break-ins. Yet the current solutions are weak, making enterprise network management both expensive and error-prone. Indeed, most networks today require substantial manual configuration by trained operators [11, 22, 23, 25] to achieve even moderate security [24]. A Yankee Group report found that 62% of network downtime in multi-vendor networks comes from human-error and that 80% of IT budgets is spent on maintenance and operations [16].

There have been many attempts to make networks more manageable and more secure. One approach introduces proprietary middleboxes that can exert their control effectively only if placed at network choke-points. If traffic accidentally flows (or is maliciously diverted) around the middlebox, the network is no longer managed nor secure [25]. Another approach is to add functionality to existing networks—to provide tools for diagnosis, to offer controls for VLANs, access-control lists, and filters to isolate users, to instrument the routing and spanning tree algorithms to support better connectivity management, and then to collect packet traces to allow auditing. This can be done by adding a new layer of protocols, scripts, and applications [1, 10] that help automate configuration management in order to reduce the risk of errors. However, these solutions hide the complexity, not reduce it. And they have to be constantly maintained to support the rapidly changing and often proprietary management interfaces exported by the managed elements.

Rather than building a new layer of complexity on top of the network, we explore the question: *How could we change the enterprise network architecture to make it more manageable?* Our answer is embodied in the architecture we describe here, called Ethane. Ethane is built around three fundamental principles that we feel are important to any network management solution:

The network should be governed by policies declared over high-level names. Networks are most easily managed in terms of the entities we seek to control—such as users, hosts, and access points—rather than in terms of low-level and often dynamically-allocated addresses. For example, it is convenient to declare which services a user is allowed to use and to which machines they can connect.

Policy should determine the path that packets follow. There are several reasons for policy to dictate the paths. First, policy might require packets to pass through an intermediate middlebox; for example, a guest user might be required to communicate via a proxy, or the user of an unpatched operating system might be required to communicate via an intrusion detection system. Second, traffic can receive more appropriate service if its path is controlled; directing real-time communications over lightly loaded paths, important communications over redundant paths, and private communications over paths inside a trusted boundary would all lead to better service. Allowing the network manager to determine the paths via policy—where the policy is in terms of high-level names—leads to finer-level control and greater visibility than is easily achievable with current designs.

The network should enforce a strong binding between a packet and its origin. Today, it is notoriously difficult to reliably determine the origin of a packet: Addresses are dynamic and change

SIGCOMM'07, August 27–31, 2007, Kyoto, Japan.

Key challenges / what's missing?

From student discussion:

- Which plane certain features go in?
- Scalability, single point of failure
- Cost-benefit trade-off
  - Performance wins?
- How to translate decision to data plane (is the intent same)?
- Control plane decisions in real-time given complex goals
- Control plane interface to network operators?

Your opinions

- Positives:
  - Innovative, ambitious, new, promising, comprehensive
  - Discussed flaws/limitations/challenges/future directions.
  - No change to packet format.
  - Decision plane is easy to customize, can enable more sohpisticated algorithms based on network-wide constraints.

Your opinions

- Negatives:
  - No evaluation, vague about implementation details
  - Concerns about scalability, resilience/reliability.
  - Overhead of such an approach, e.g. route convergence time with dynamic network setting.
  - Ignores middleboxes?
  - Concerns about physical separation between data and dissemination plane (?)
  - Privacy issues around sharing information with the decision plane.

Your opinions

- Ideas:
  - Implementing and evaluating the design.
  - Protocols for dissemination plane.
  - Greater focus on scalability and fault-tolerance.
  - Cost-benefit trade-off.
  - Data monitoring in the discovery plane.
  - Which functionality should go in data plane vs control plane?
  - Eliminate some of the key features of 4D to enable scalability. *
  - Analyze 4D through the lens of end-to-end arguments.*
  - How to incorporate in-network programmability? Another plane?