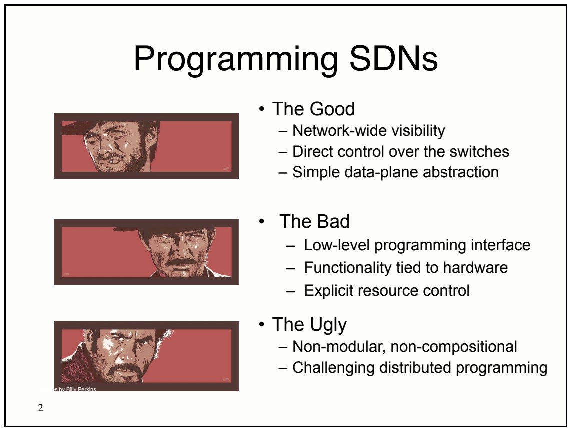
SDN Programming Languages

COS 597E: Software Defined Networking

Jennifer Rexford

Princeton University

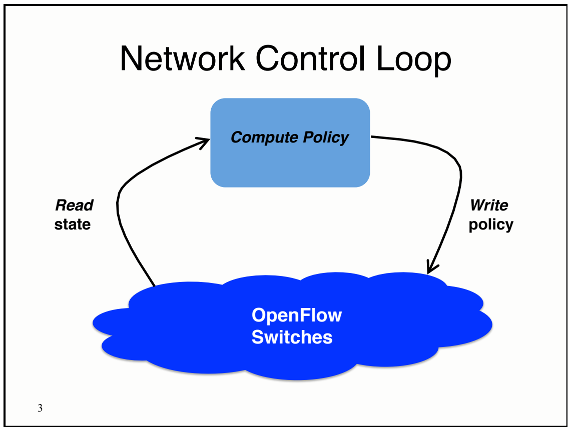
MW 11:00am-12:20pm



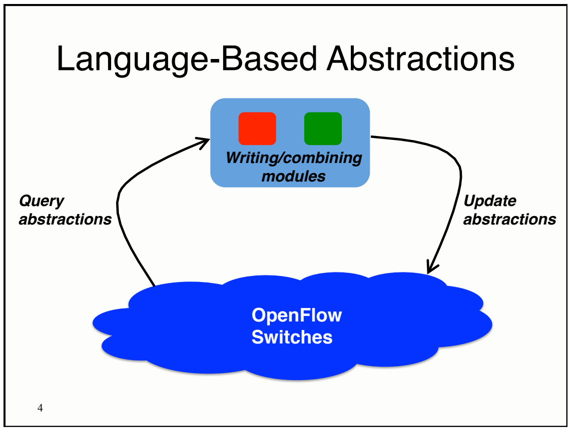
Programming SDNs

- The Good
  - Network-wide visibility
  - Direct control over the switches
  - Simple data-plane abstraction
- The Bad
  - Low-level programming interface
  - Functionality tied to hardware
  - Explicit resource control
- The Ugly
  - Non-modular, non-compositional
  - Challenging distributed programming

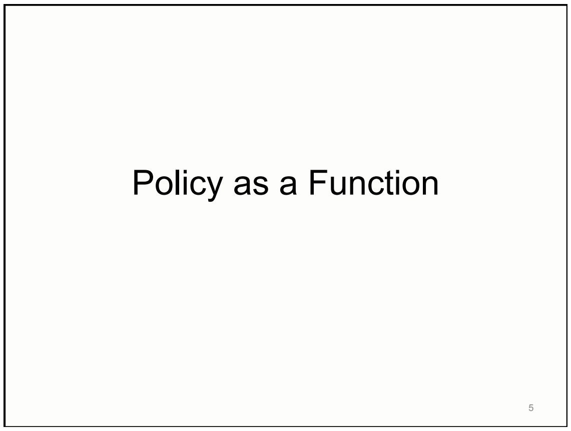
s by Billy Perkins



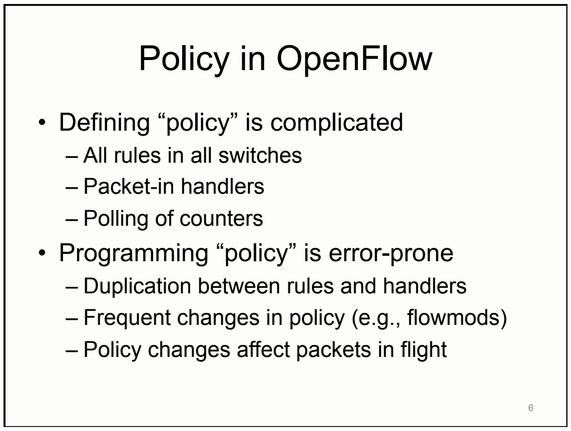
Network Control Loop



Language-Based Abstractions

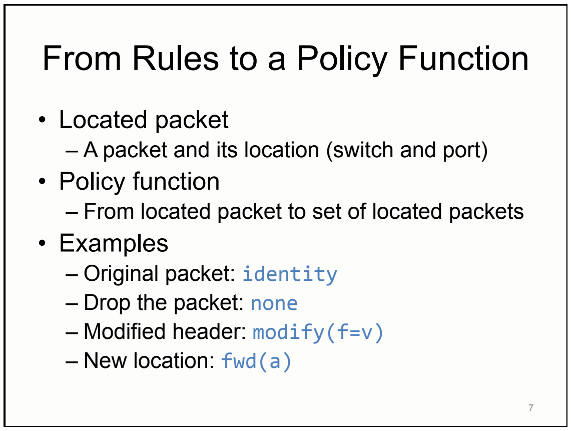


Policy as a Function



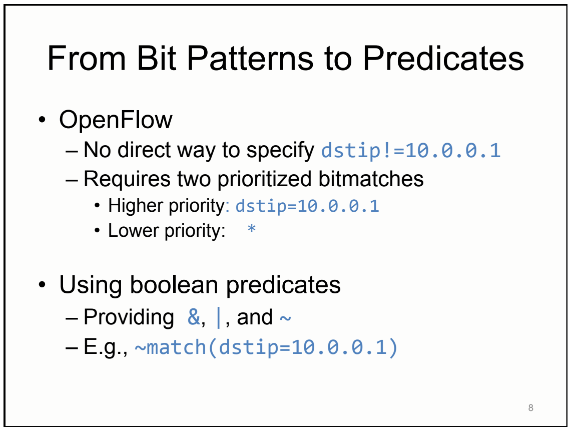
Policy in OpenFlow

- Defining “policy” is complicated
  - All rules in all switches
  - Packet-in handlers
  - Polling of counters
- Programming “policy” is error-prone
  - Duplication between rules and handlers
  - Frequent changes in policy (e.g., flowmods)
  - Policy changes affect packets in flight



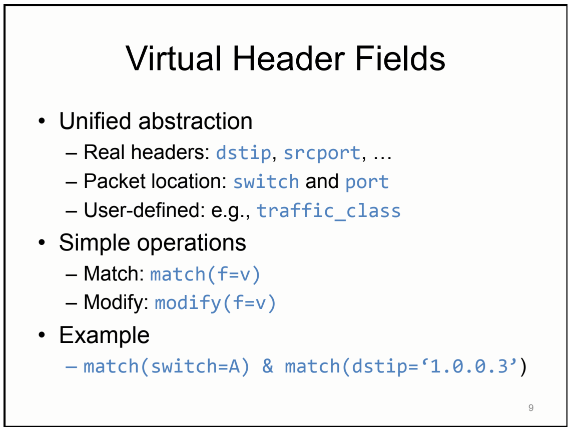
From Rules to a Policy Function

- Located packet
  - A packet and its location (switch and port)
- Policy function
  - From located packet to set of located packets
- Examples
  - Original packet: `identity`
  - Drop the packet: `none`
  - Modified header: `modify(f=v)`
  - New location: `fwd(a)`



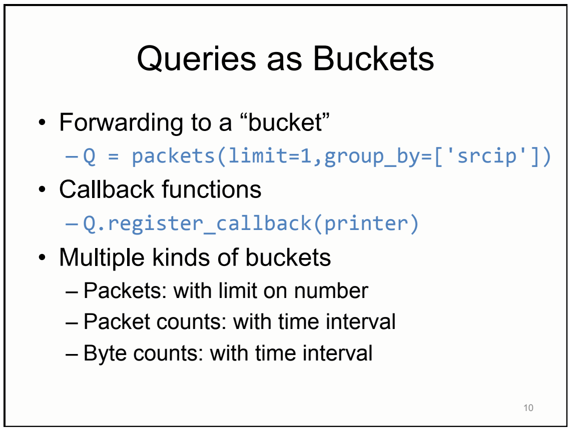
From Bit Patterns to Predicates

- OpenFlow
  - No direct way to specify `dstip!=10.0.0.1`
  - Requires two prioritized bitmatches
    - Higher priority: `dstip=10.0.0.1`
    - Lower priority: `*`
- Using boolean predicates
  - Providing `&`, `|`, and `~`
  - E.g., `~match(dstip=10.0.0.1)`



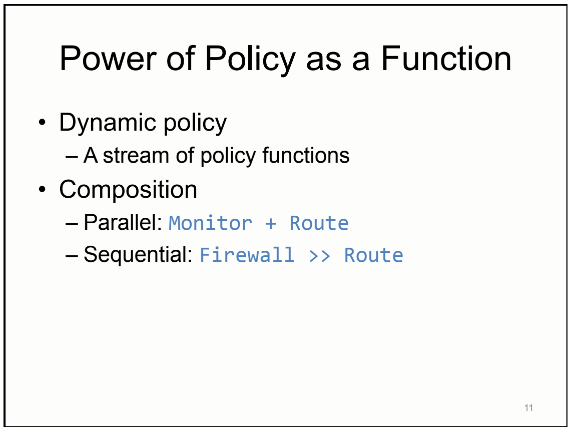
Virtual Header Fields

- Unified abstraction
  - Real headers: `dstip`, `srcport`, …
  - Packet location: `switch` and `port`
  - User-defined: e.g., `traffic_class`
- Simple operations
  - Match: `match(f=v)`
  - Modify: `modify(f=v)`
- Example
  - `match(switch=A) & match(dstip=‘1.0.0.3’)`



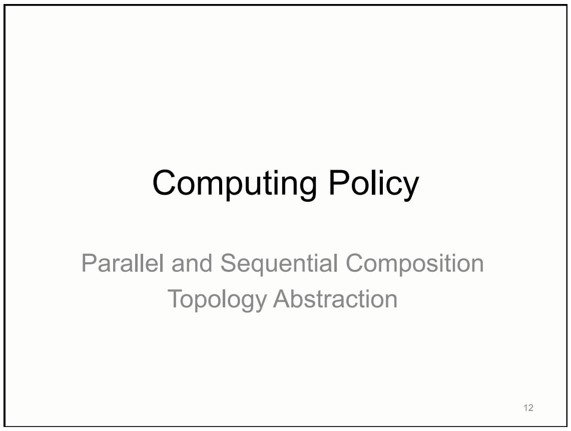
Queries as Buckets

- Forwarding to a “bucket”
  - `Q = packets(limit=1,group_by=['srcip'])`
- Callback functions
  - `Q.register_callback(printer)`
- Multiple kinds of buckets
  - Packets: with limit on number
  - Packet counts: with time interval
  - Byte counts: with time interval



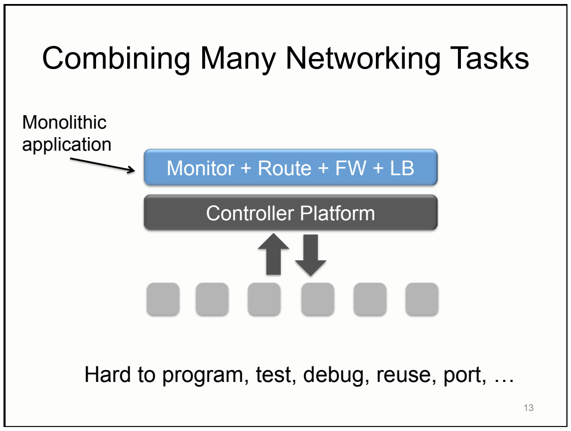
Power of Policy as a Function

- Dynamic policy
  - A stream of policy functions
- Composition
  - Parallel: `Monitor + Route`
  - Sequential: `Firewall >> Route`



Computing Policy

Parallel and Sequential Composition
Topology Abstraction



Combining Many Networking Tasks

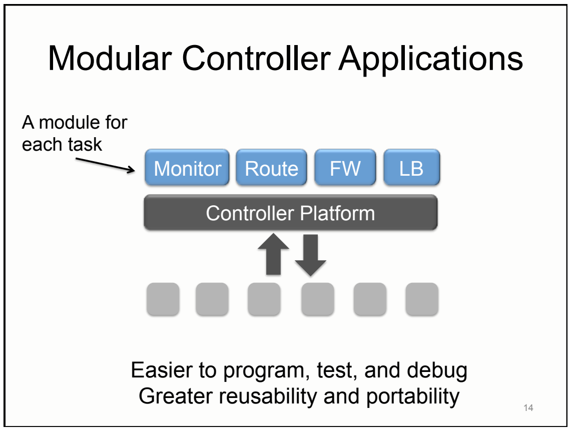
Monolithic application

Monitor + Route + FW + LB

Controller Platform

Hard to program, test, debug, reuse, port, …

13



Modular Controller Applications

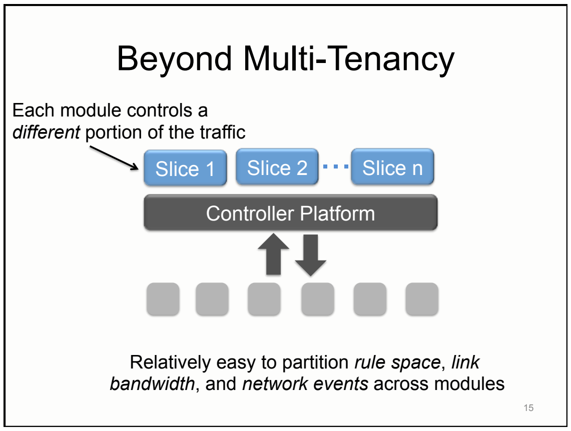
A module for each task

Monitor | Route | FW | LB

Controller Platform

Easier to program, test, and debug
Greater reusability and portability

14



Beyond Multi-Tenancy

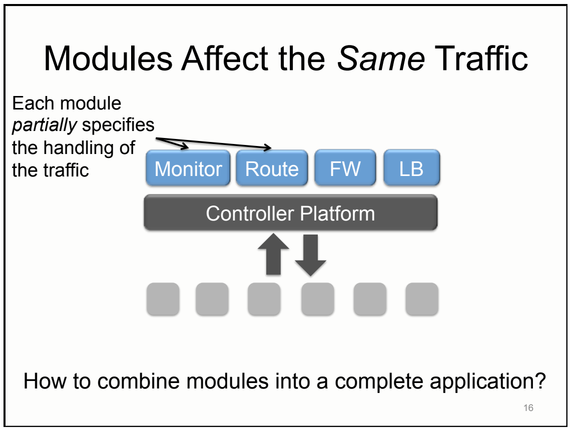
Each module controls a *different* portion of the traffic

Slide 1 | Slide 2 ··· Slice n

Controller Platform

Relatively easy to partition *rule space*, *link bandwidth*, and *network events* across modules

15



Modules Affect the *Same* Traffic

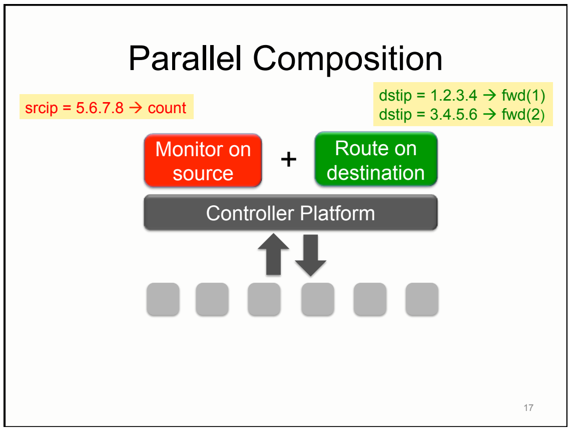
Each module *partially* specifies the handling of the traffic

Monitor | Route | FW | LB

Controller Platform

How to combine modules into a complete application?

16



Parallel Composition

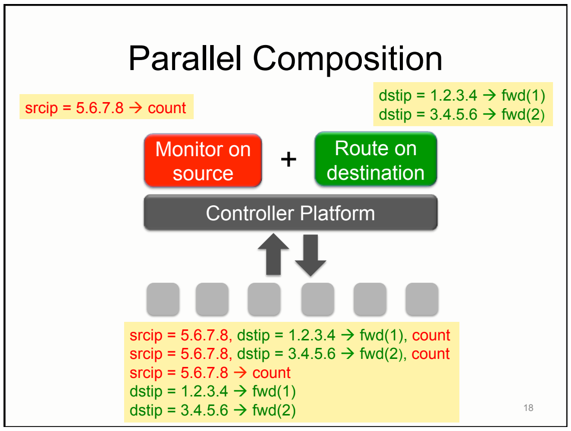
srcip = 5.6.7.8 → count

dstip = 1.2.3.4 → fwd(1)
dstip = 3.4.5.6 → fwd(2)

Monitor on source + Route on destination

Controller Platform

17



Parallel Composition

srcip = 5.6.7.8 → count

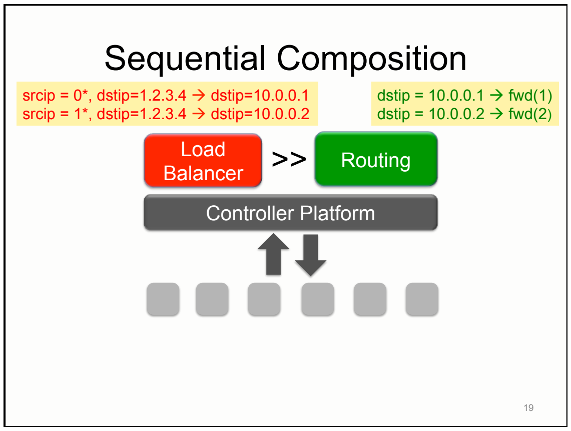
dstip = 1.2.3.4 → fwd(1)
dstip = 3.4.5.6 → fwd(2)

Monitor on source + Route on destination

Controller Platform

srcip = 5.6.7.8, dstip = 1.2.3.4 → fwd(1), count
srcip = 5.6.7.8, dstip = 3.4.5.6 → fwd(2), count
srcip = 5.6.7.8 → count
dstip = 1.2.3.4 → fwd(1)
dstip = 3.4.5.6 → fwd(2)

18



Sequential Composition

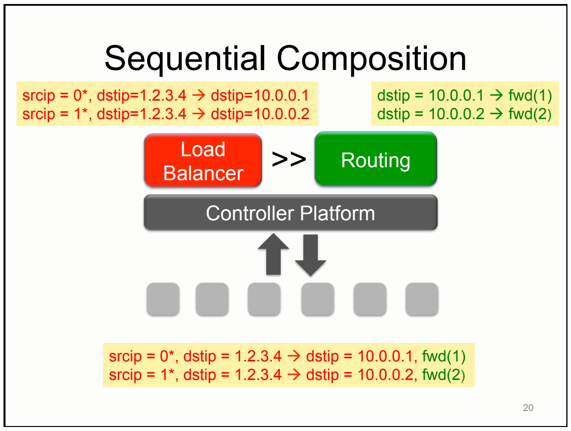
srcip = 0*, dstip=1.2.3.4 → dstip=10.0.0.1
srcip = 1*, dstip=1.2.3.4 → dstip=10.0.0.2

dstip = 10.0.0.1 → fwd(1)
dstip = 10.0.0.2 → fwd(2)

Load Balancer >> Routing

Controller Platform

19



Sequential Composition

srcip = 0*, dstip=1.2.3.4 → dstip=10.0.0.1
srcip = 1*, dstip=1.2.3.4 → dstip=10.0.0.2

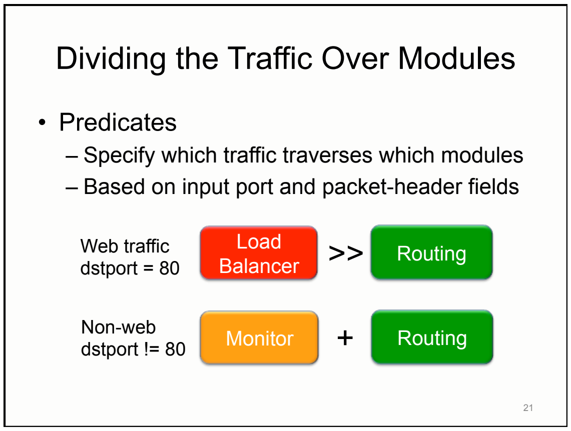
dstip = 10.0.0.1 → fwd(1)
dstip = 10.0.0.2 → fwd(2)

Load Balancer >> Routing

Controller Platform

srcip = 0*, dstip = 1.2.3.4 → dstip = 10.0.0.1, fwd(1)
srcip = 1*, dstip = 1.2.3.4 → dstip = 10.0.0.2, fwd(2)

20



Dividing the Traffic Over Modules

- Predicates
  - Specify which traffic traverses which modules
  - Based on input port and packet-header fields

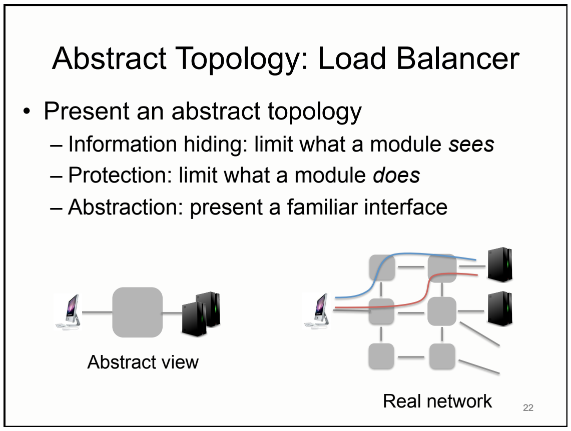
Web traffic
dstport = 80

Load Balancer >> Routing

Non-web
dstport != 80

Monitor + Routing

21



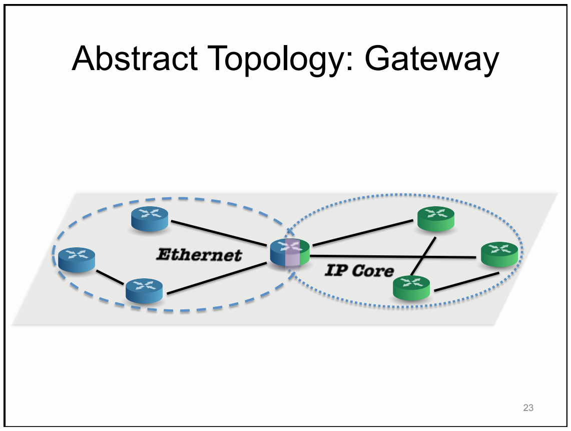
Abstract Topology: Load Balancer

- Present an abstract topology
  - Information hiding: limit what a module *sees*
  - Protection: limit what a module *does*
  - Abstraction: present a familiar interface

Abstract view

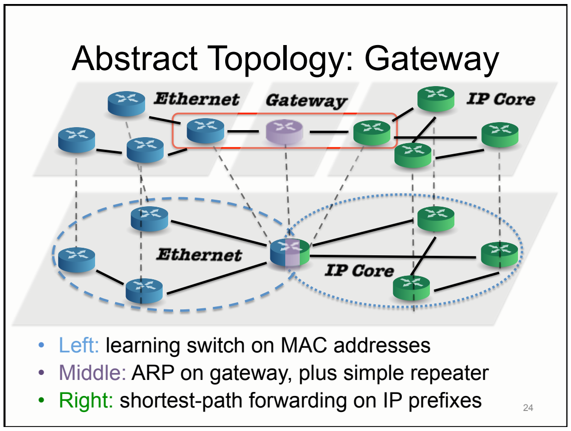
Real network

22



Abstract Topology: Gateway

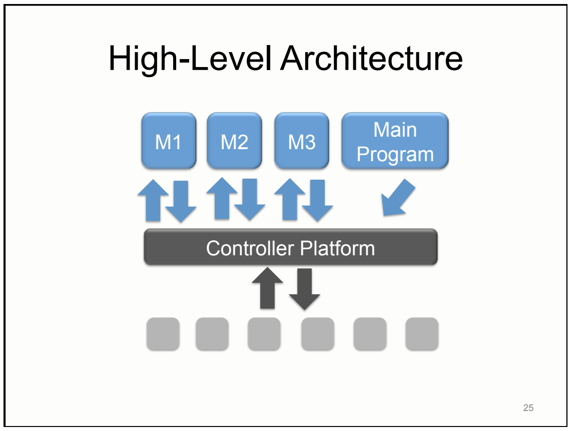
23



Abstract Topology: Gateway

- Left: learning switch on MAC addresses
- Middle: ARP on gateway, plus simple repeater
- Right: shortest-path forwarding on IP prefixes

24



High-Level Architecture

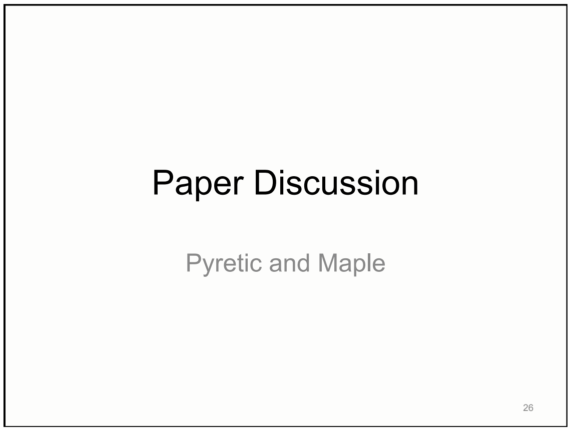
M1

M2

M3

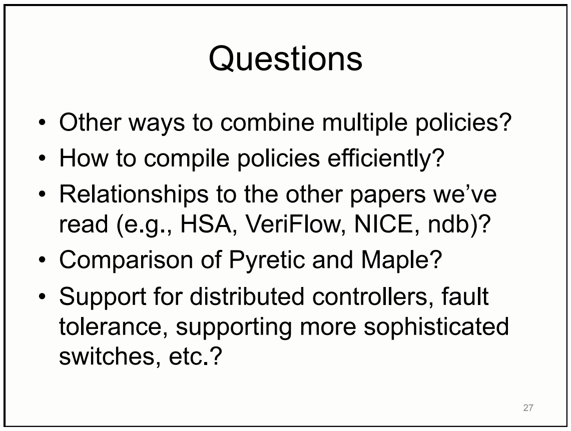
Main Program

Controller Platform



Paper Discussion

Pyretic and Maple



Questions

- Other ways to combine multiple policies?
- How to compile policies efficiently?
- Relationships to the other papers we've read (e.g., HSA, VeriFlow, NICE, ndb)?
- Comparison of Pyretic and Maple?
- Support for distributed controllers, fault tolerance, supporting more sophisticated switches, etc.?