EECS 428 Computer Communications NetworksII:

Software-Defined Networking and Emerging Applications

An Wang
Case Western Reserve University

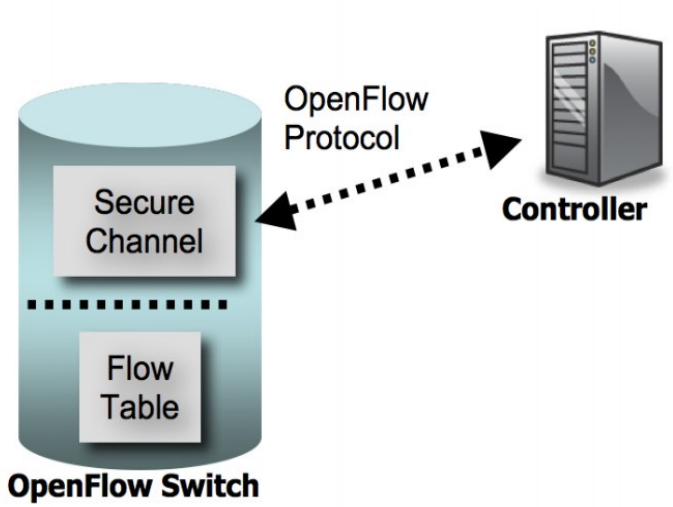
Part I: Recap of SDN and OpenFlow

- SDN Architecture
- OpenFlow 1.0 and Beyond
- OpenFlow Switch Components

What is SDN?

- What is the definition of SDN?
  - The *separation* of control plane from data plane
  - A specific SDN: configuration, distribution and forwarding abstraction
- What is one API between control plane and data plane?
  - OpenFlow protocol

OpenFlow Protocol Specification



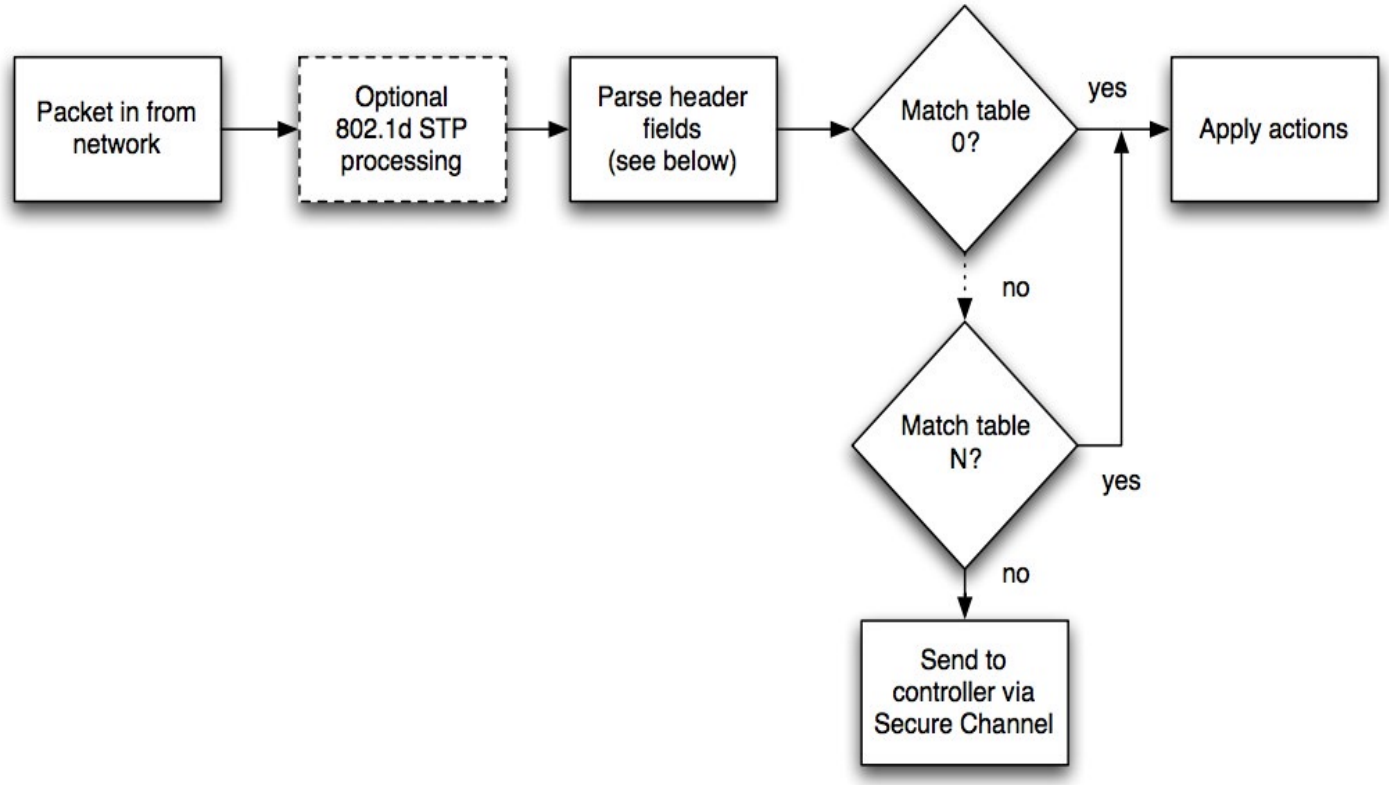
- OpenFlow controller communicates with switch over a secure channel
  - OpenFlow protocol defines message format
  - Purpose of control channel: update flow table
  - Logic is executed at **controller**

Switch Components

- **Flow table:** Performs packet lookup
  - All packets compared to flow table for **match**
  - **Actions** depend on match being found
  - If no match, traffic is sent to controller

- **Secure channel:** Communication to external controller

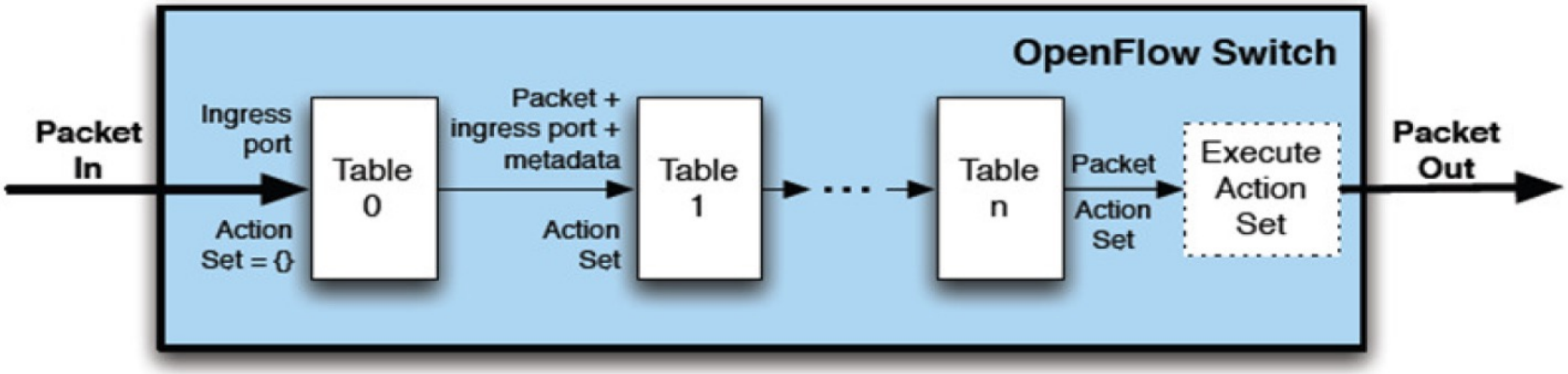
Matching – OpenFlow v1.0



- Packet header fields matched against one of N tables
- If no match, packet is sent to controller
- Otherwise, switch performs action

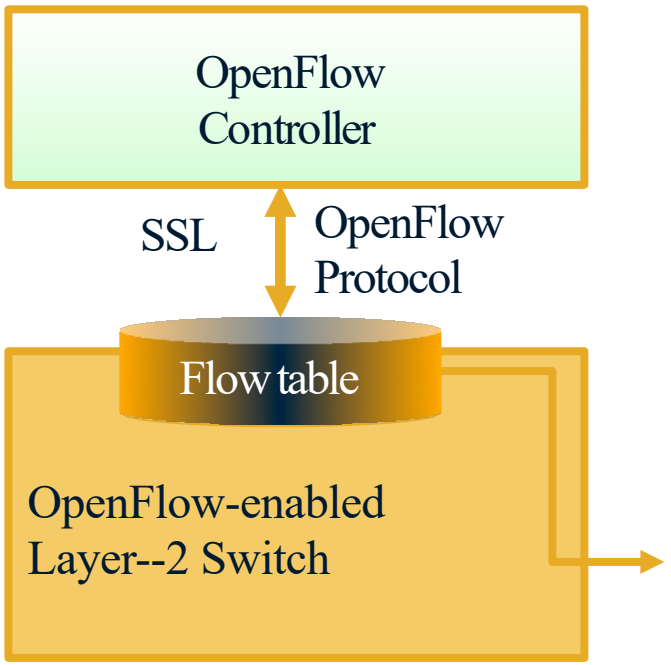
Switch Pipeline Processing

- A switch can have multiple flow tables that are matched in a pipeline fashion.



(a) Packets are matched against multiple tables in the pipeline

Matching: Fields in Lookup



- 12-tuple
  (also, VLAN priority and ToS)
- Support for wildcard matching

Matches subsets of packet header fields

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport

Actions: Forward/Drop

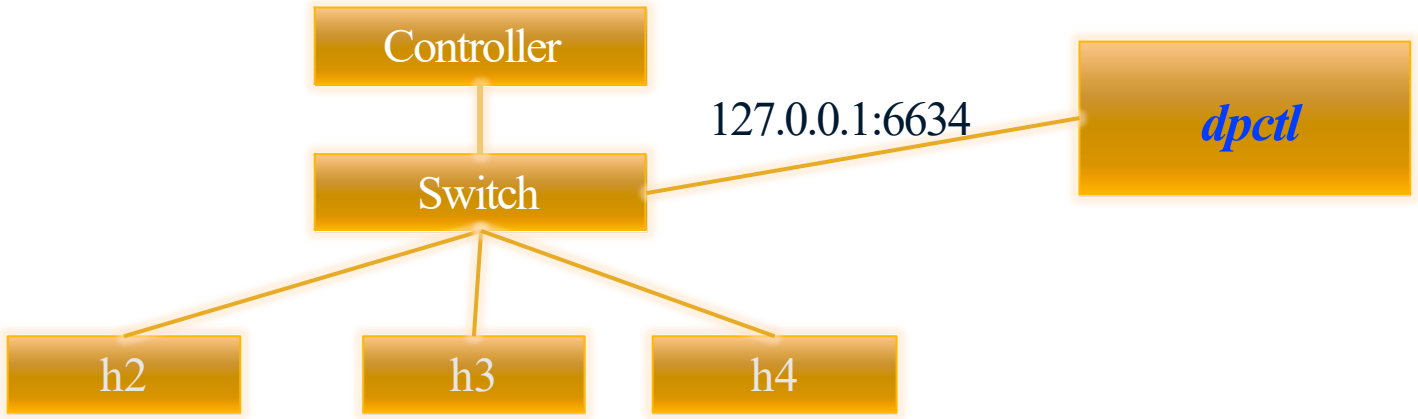
- **Forward**
  - **ALL:** Send out all interfaces, not including the incoming interface.
  - **CONTROLLER:** Encapsulate and send to the controller.
  - **LOCAL:** Send to the switch's local networking stack.
  - **TABLE:** Perform actions in flow table. Only for packet-out messages.
  - **IN PORT:** Send the packet out the input port
  - **Optional:** Normal forwarding, spanning tree
- **Drop:** A flow-entry with no specified action indicates that all matching packets should be dropped.

Optional Actions: Modify/Enqueue

- **Modify:** Option to modify packet header values in the packet (e.g., VLAN ID)
  - Set VLAN ID, priority, etc.
  - Set destination IP address

- **Enqueue:** Send the packet through a queue attached to a port

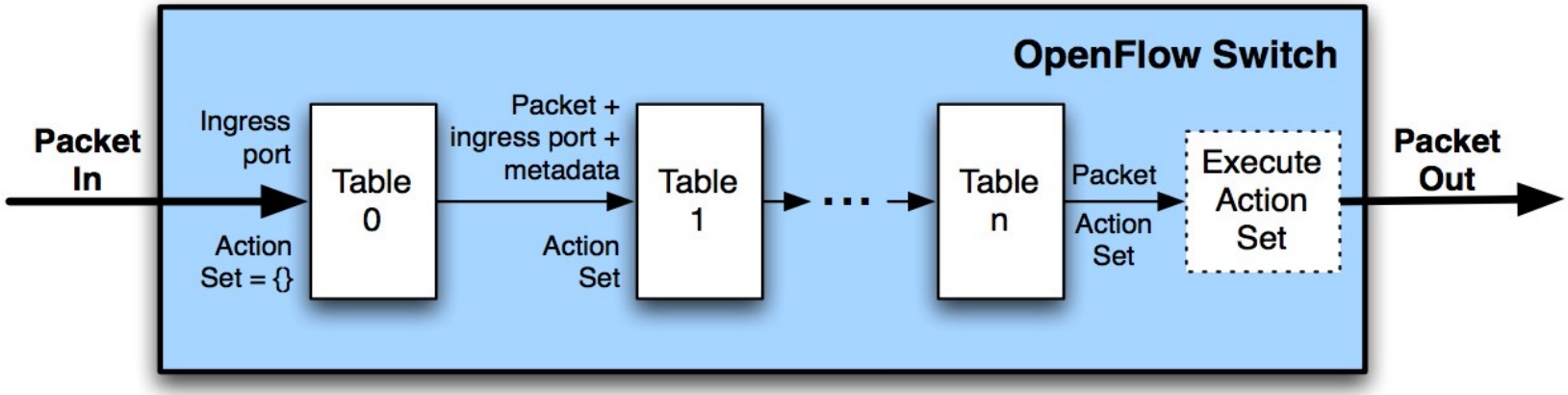
Example: dpctl Control Channel



- `$ sudo mn --topo single,3 --mac --switch ovsk --controller remote`

- `dpctl` to communicate with switches
  - Switches listen on port 6634
  - Can inspect flow table entries, modify flows, etc.

OpenFlow (v1.3) Enhancement



- **Action set:** Set of actions to be performed on each packet
- **Group:** A list of action sets
- Each table updates fields, modifies action set

Action Group Options

- Execute all action sets in a group
  - Useful for implementing multicast: One packet is cloned for each action set in the group
- Indirect groups
  - Execute the one defined bucket in the group.

    Useful for pointing multiple flow entries to a common action

Other SDN Control Architecture

- Juniper's Contrail Controller (Linux)
  - XMPP as control plane
  - L2 and L3 virtual networks
  - Contributions to OpenDaylight

- Cisco's Open Network Environment
  - Centralized software controller
  - Programmable data plane
  - Ability to provide virtual overlays

Summary: SDN Basics

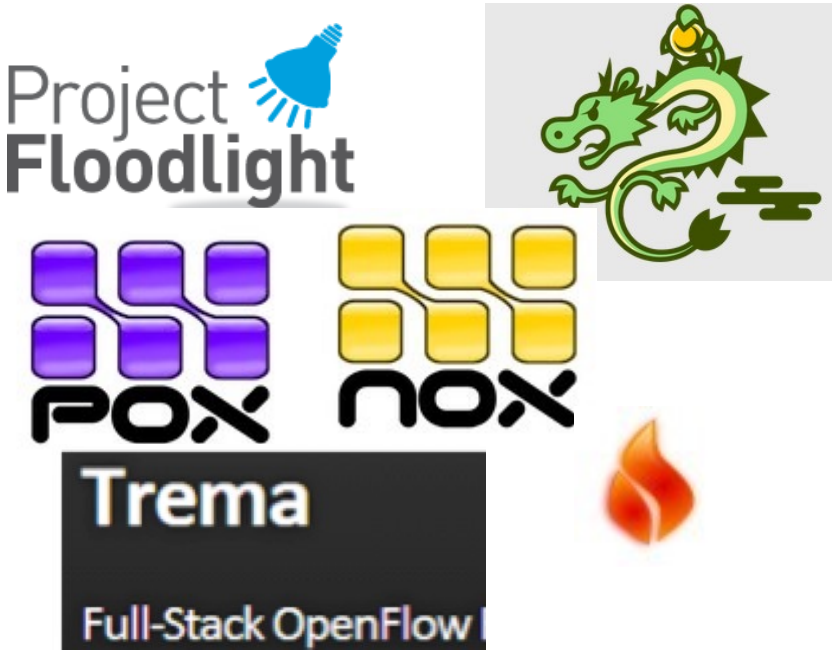
- OpenFlow Switch Components
  - Secure channel
  - Flow tables (match and action)
  - (New) Group tables
- OpenFlow Protocol is evolving
- `dpctl` connects directly to a switch to poll, manipulate, etc.

Part II: SDN Controllers

- Overview of different SDN Controllers
- Basic Understanding of Each Controller
- Pros and Cons & Ideal Situations

Many Different SDN Controllers

- **NOX/POX**
- **Ryu**
- **Floodlight**
- Pyretic
- Frenetic
- Procera
- RouteFlow

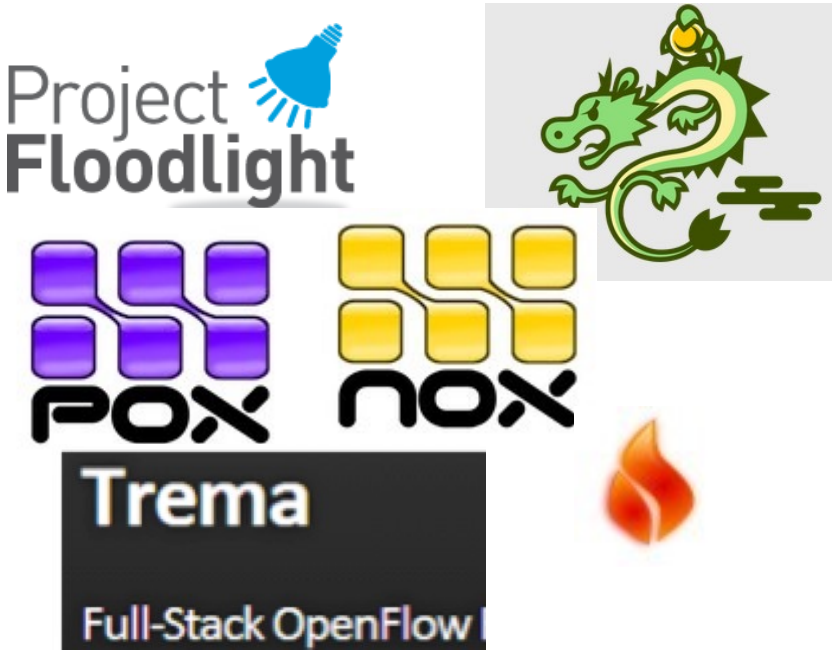


Many Considerations

- Programming Language (can affect performance)
- Learning curve
- User base and community support
- Focus
  - Southbound API
  - Northbound API / “Policy Layer”
  - Support for OpenStack
  - Education, Research, or Production?

Many Different SDN Controllers

- **NOX/POX**
- **Ryu**
- **Floodlight**
- Pyretic
- Frenetic
- Procera
- RouteFlow
- Trema



NOX: Overview

- First-generation OpenFlow controller
  - Open source, stable, widely used

- Two "flavors" of NOX
  - **NOX-Classic:** C++/Python. No longer supported
  - **NOX (the "new NOX")**
    - C++ only
    - Fast, clean codebase
    - Well maintained and supported

NOX: Characteristics

- Users implement control in C++
- Supports OpenFlow v.1.0
  - A fork (CPqD) supports 1.1, 1.2, and 1.3
- Programming model
  - Controller registers for events
  - Programmer writes event handler

When to Use NOX

- You know C++
- You are willing to use low-level facilities and semantics of OpenFlow
- You need good performance

POX: Overview

- NOX in Python
  - Supports OpenFlow v1.0 only

- **Advantages**
  - Widely used, maintained, supported
  - Relatively easy to read and write code

- **Disadvantages:** Performance

When to Use POX

- If you know (or can learn) Python and are not concerned about controller performance

- Rapid prototyping and experimentation
  - Research, experimentation, demonstrations
  - Learning concepts

Ryu

- Open source Python controller
  - Supports OpenFlow 1.0, 1.2, 1.3, Nicira extensions
  - Works with OpenStack
- Aims to be an "Operating System" for SDN
- **Advantages**
  - OpenStack integration, OpenFlow 1.2, 1.3 and 1.4
- **Disadvantages:** Performance

Floodlight

- Open-source Java controller
  - Supports OpenFlow v1.0
  - Fork from the Beacon Java OpenFlow controller
  - Maintained by Big Switch Networks
- **Advantages**
  - Good documentation
  - Integration with REST API
  - Production-level performance, OpenStack
- **Disadvantages:** Steep learning curve

When to Use Floodlight

- You know Java
- You need production-level performance and support
- You will use the REST API to interact with the controller

Summary

	NOX	POX	Ryu	Floodlight
Language	C++	Python	Python	Java
Performance	Fast	Slow	Slow	Fast
OpenFlow	1.0 (CPqD: 1.1, 1.2, 1.3)	1.0	1.0, 1.1, 1.3	1.0
OpenStack	No	No	Yes	Yes
Learning Curve	Moderate	Easy	Moderate	Steep

- Choice of controller depends on needs, language, etc.
- **So far:** Southbound API implementations

Part II: SDN Prototyping Tools

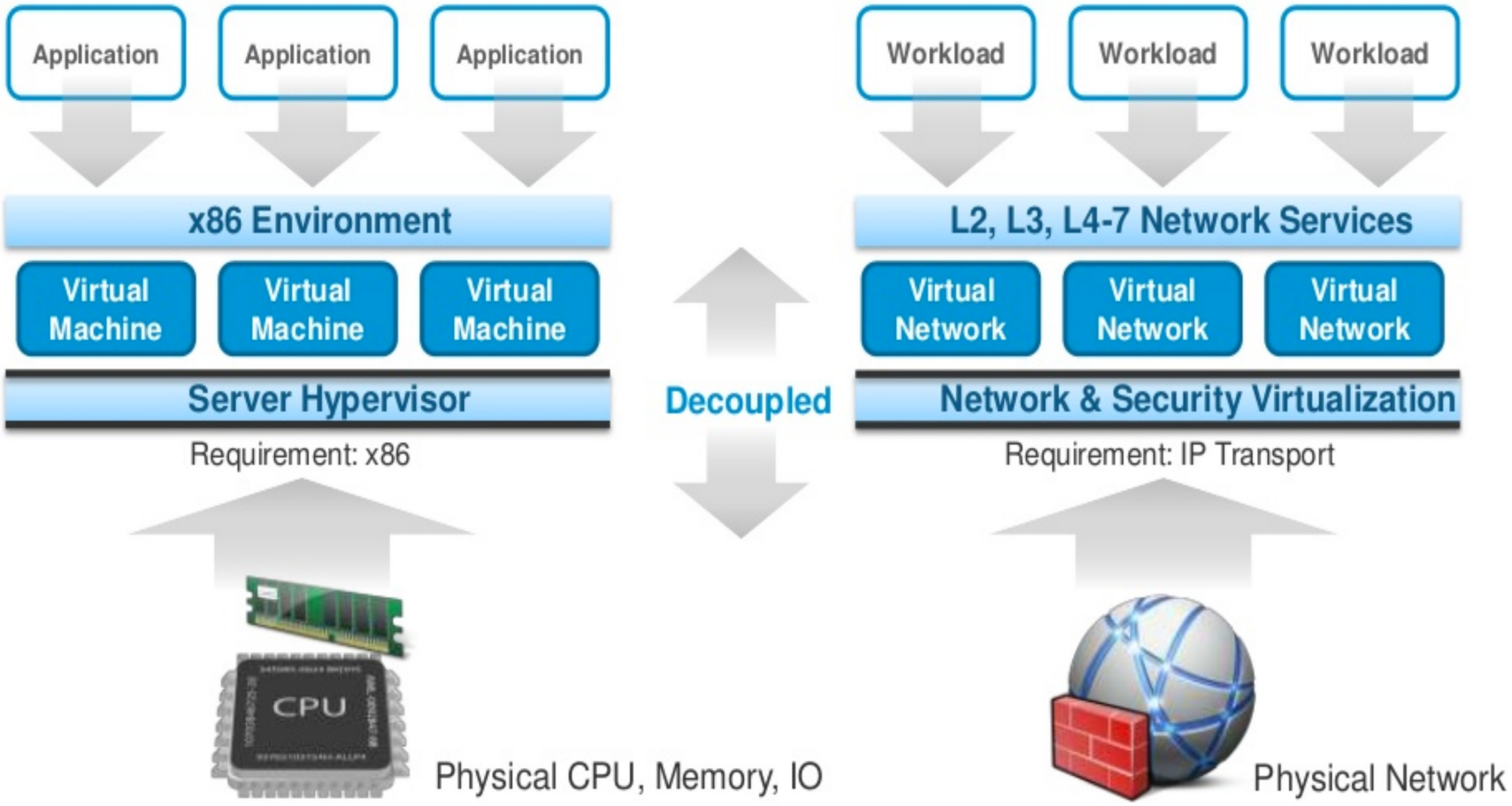
- What is Network Virtualization?
- Examples of Network Virtualization and Applications
- Virtual Networking in Mininet
- Virtual Switch – Open vSwitch

What is Network Virtualization?

- Abstraction of the physical network
  - Support for multiple logical networks running on a common shared physical substrate
  - A container of network services

- Aspects of the network that can be virtualized
  - **Nodes:** Virtual machines
  - **Links:** Tunnels (e.g., Ethernet GRE)
  - Storage

Network Virtualization



Motivation of Network Virtualization

- "Ossification" of the Internet architecture
  - Lots of work on overlay networks in the 2000s
  - One-size-fits all architectures are difficult
  - Why not allow for easier evolution?

- Instead, why not create a substrate where "1,000 flowers can bloom"?

The Promise of Network Virtualization

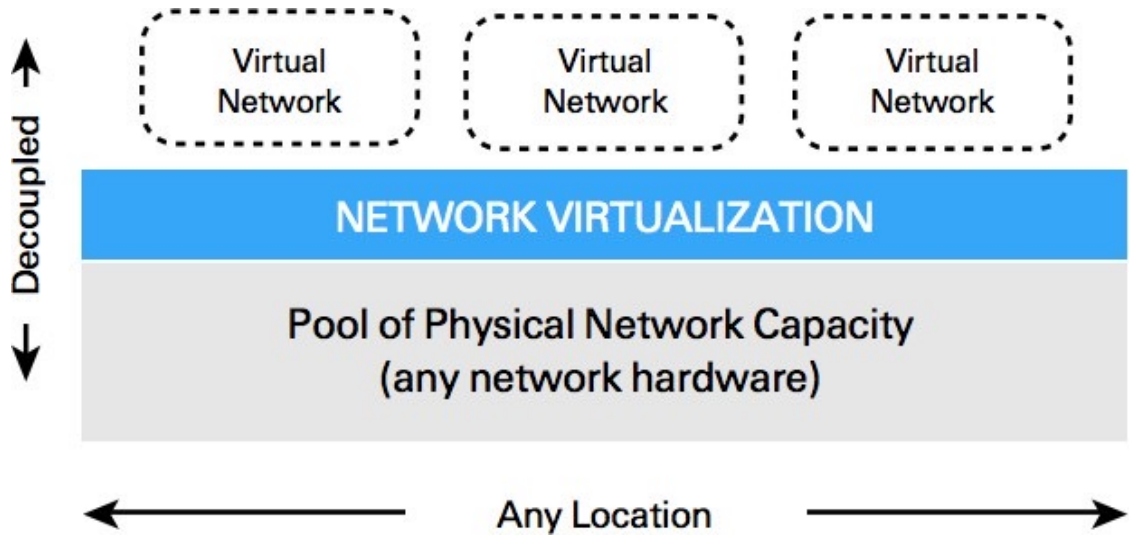
- Rapid innovation: services delivered at software speeds (vswitch and controller)
- New forms of network control
- Vendor choice
- Simplified programming and operations

Distinction: SDN does not inherently abstract the details of the physical network

Applications of Virtual Networking

- Experimentation on production networks (FlowVisor)
  - Can run (virtual) experimental infrastructure in parallel with production
- Rapid deployment and development (Nicira)
  - Can deploy services independently from underlying vendor hardware
- Dynamic scaling of resources
  - Can allocate from “pool” of resources

Rapid Deployment of Services: Nicira Network Virtualization Platform



- Abstraction layer between hosts & underlying network
- Open vSwitch in host hypervisors: abstraction layer
- Managed by distributed controller

Nicira NVP: Applications

- Dynamic workload placement
  - Multi-tenant data centers
  - Creation of isolated virtual networks for each tenant

- Dynamic security
  - Central management of security policies
  - Enforcement per virtual network
  - Independence from VLAN limits

What is Mininet?

- A **virtual network environment** that can run on a single PC
- Runs real kernel, switch, and application code on a single machine
  - Command-line, UI, Python interfaces
- Many **OpenFlow features** are built-in
  - Useful: developing, deploying, and sharing

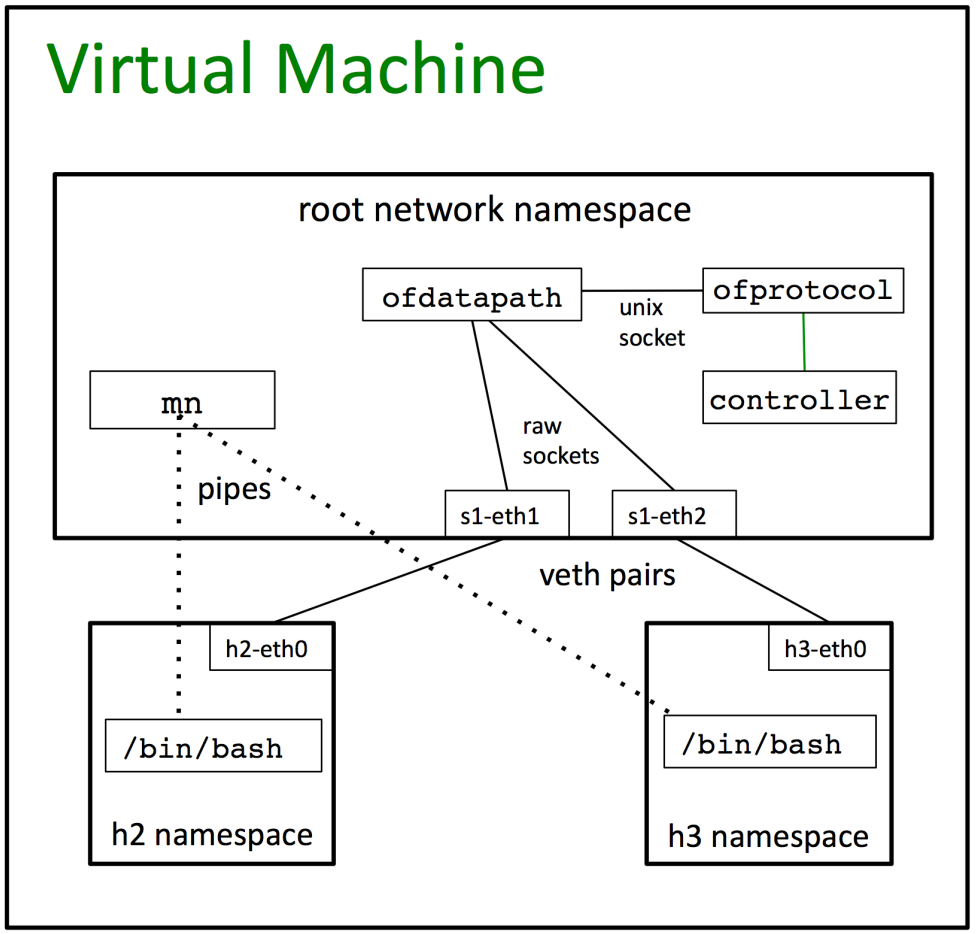
Why Use Mininet

- Fast
- Possible to create custom topologies
- Can run real programs (anything that can run on Linux can run on a Mininet host)
- Programmable OpenFlow switches
- Easy to use
- Open source

Alternatives

- **Real system:** Pain to configure
- **Networked VMs:** Scalability
- **Simulator:** No path to hardware deployment

The Mininet VM in a Nutshell



- Launch mininet process
- Per host
  - Bash process
  - Network namespace
- Create veth pairs and assign to namespaces
- Create OpenFlow switch to connect hosts
- Create OpenFlow controller

Summary

- Mininet is a network emulator that runs in a Virtual Machine
  - Lightweight OS virtualization to achieve scale
  - Fast, easy, sharable

- Download & Tutorials
  - Download: http://mininet.org/download/
  - Tutorial: http://mininet.org/walkthrough/
  - Python API: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

Testing a Simple Mininet Setup

- Try setting up a simple topology with three hosts connected to a single switch:
  - `sudo mn --test pingall --topo single,3`

- This setup uses a default switch controller and switch
  - Mininet also allows you to use custom remote controllers (and custom switches)

Basic Mininet Command Line

- **--topo** – defines a topology via command line upon mininet start-up
- **--switch** – defines the switch to be used. By default the OVSK software switch is used
- **--controller** – defines the controller to be used. If unspecified default controller is used with a default hub behavior

Trying out Different Mininet Topologies

- Minimal network with two hosts, one (1) switch
  - `sudo mn –topo minimal`
- Example with 4 hosts and 4 switches
  - `sudo mn --topo linear,4`
- Example with 3 hosts all connected to one switch.
  - `sudo mn --topo single,3`
- Tree topology with defined depth and fan-out.
  - `sudo mn --topo tree,depth=2,fanout=2`

Writing Your Own Mininet Topologies

- Example: Two hosts, one switch
- **mininet.cli.CLI(net)** before net.stop() will escape to interactive CLI before script terminates
- **addLink** allows you to specify: Bandwidth (bw) in Mbps, Delay (delay), Maximum Queue Size (max_queue_size), Loss (loss) in percentage

```
from mininet.net import Mininet
net = Mininet()

# Creating nodes in the network.
c0 = net.addController()
h0 = net.addHost('h0')
s0 = net.addSwitch('s0')
h1 = net.addHost('h1')

# Creating links between nodes in
network(2--ways)

net.addLink(h0, s0)
net.addLink(h1, s0)

# Configuration of IP addresses in
interfaces

h0.setIP('192.168.1.1', 24)
h1.setIP('192.168.1.2', 24)

net.start()
net.pingAll()
net.stop()
```

What is Open vSwitch

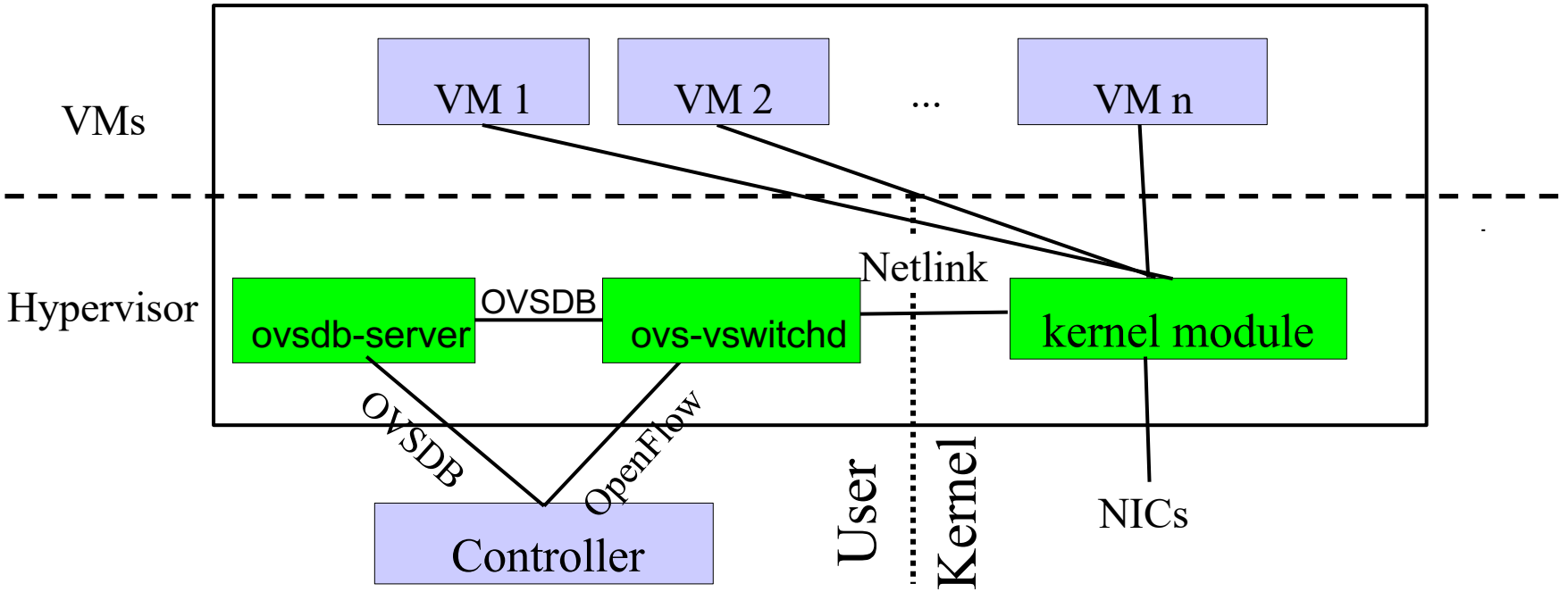
From openvswitch.org:

"Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols (e.g. NetFlow, sFlow, SPAN, RSPAN, CLI, LACP, 802.1ag)."

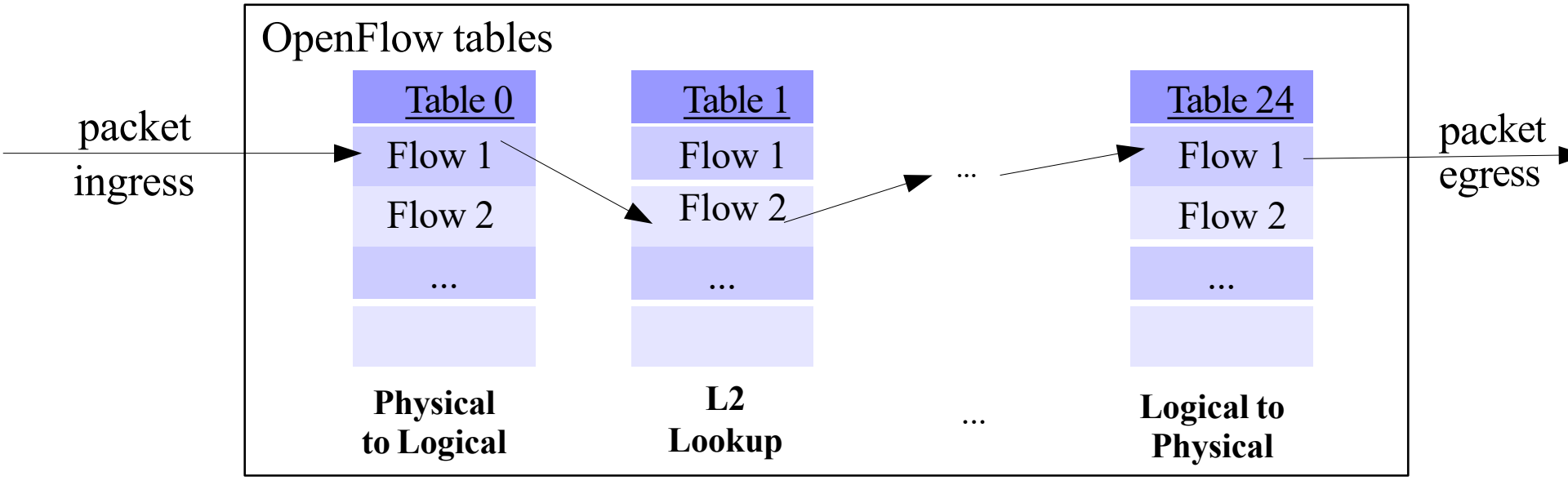
Where is Open vSwitch Used?

- Broad Support
  - Linux, FreeBSD, NetBSD, Windows, ESX KVM,
  - Xen, Docker, VirtualBox, Hyper-V, …
  - OpenStack, CloudStack, OpenNebula, …
- Widely Used
  - Most popular OpenStack networking backend
  - Default network stack in XenServer
  - 1,440 hits in Google Scholar
  - Thousands of subscribers to OVS mailing lists

Open vSwitch Architecture

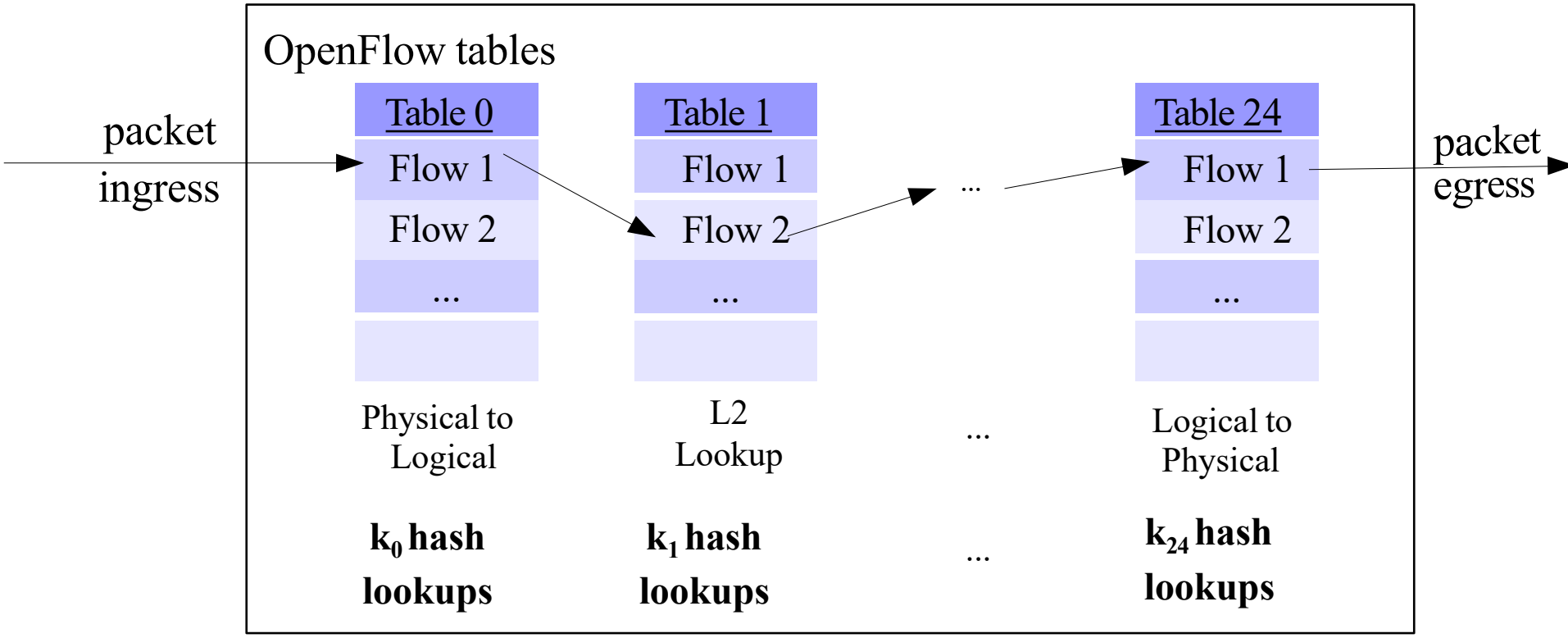


Use Case: Network Virtualization



OpenFlow Pipeline

Implications for Forwarding Performance

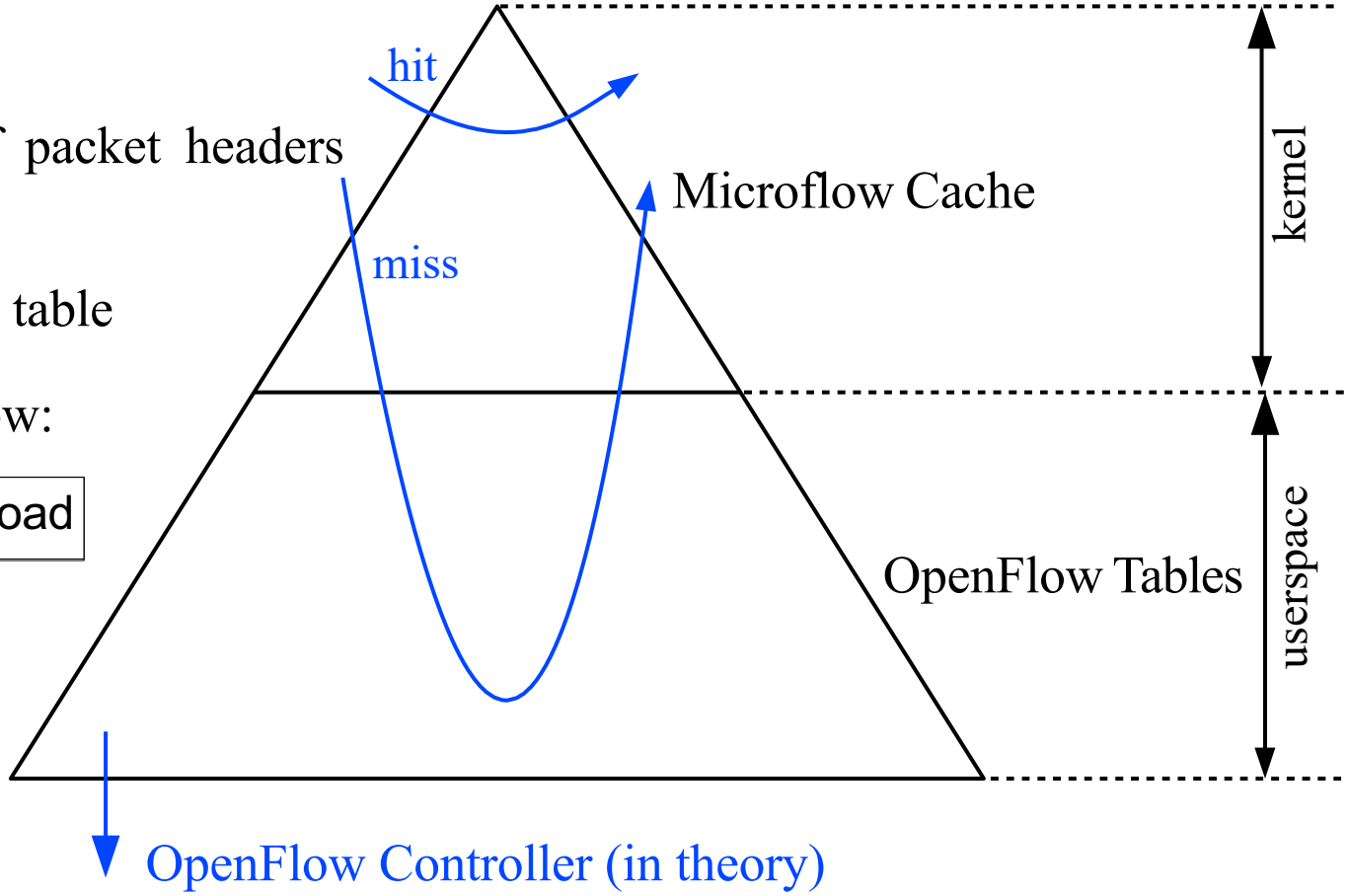


100+ hash lookups per packet for tuple space search?

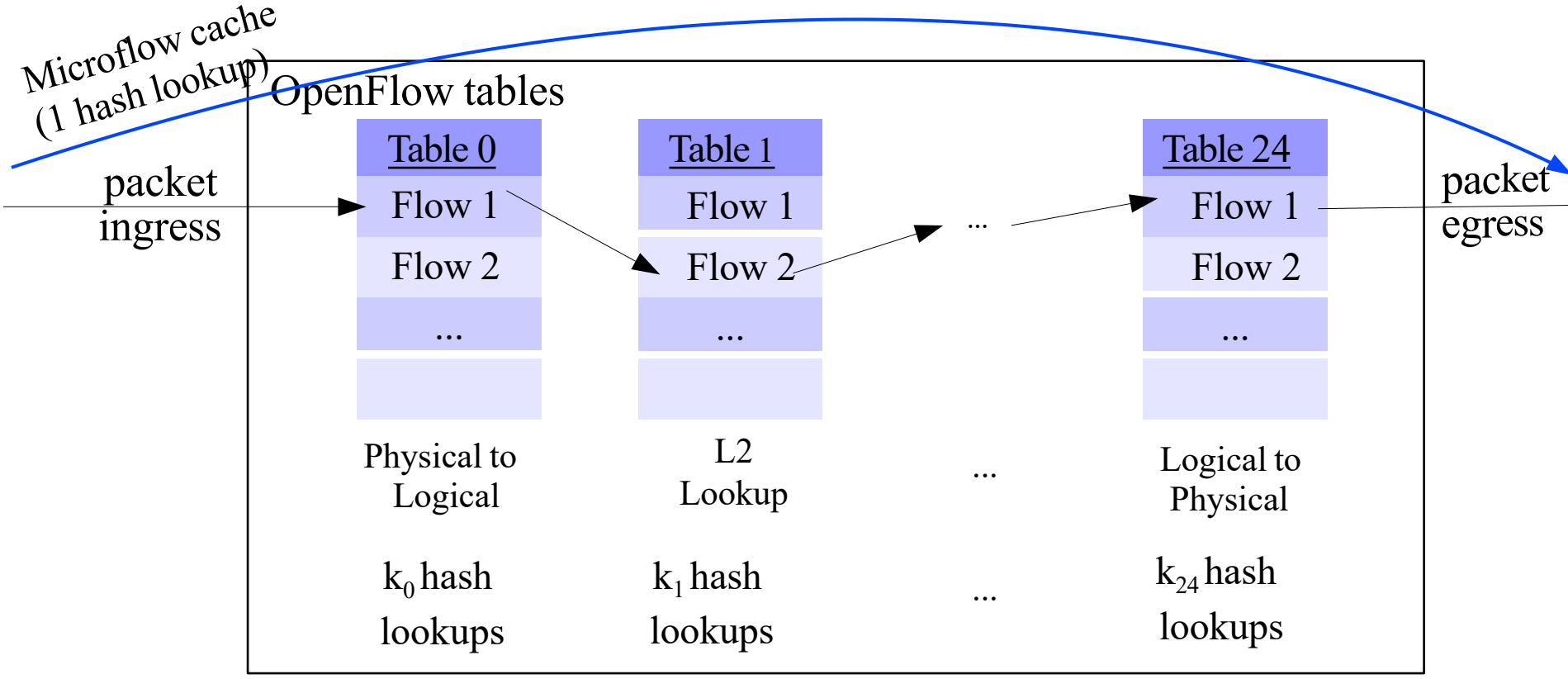
OVS Cache v1: Microflow Cache

- Microflow:
  - Complete set of packet headers and metadata
  - Suitable for hash table
  - Shaded data below:

Eth	IP	TCP	payload



Speedup with Microflow Cache

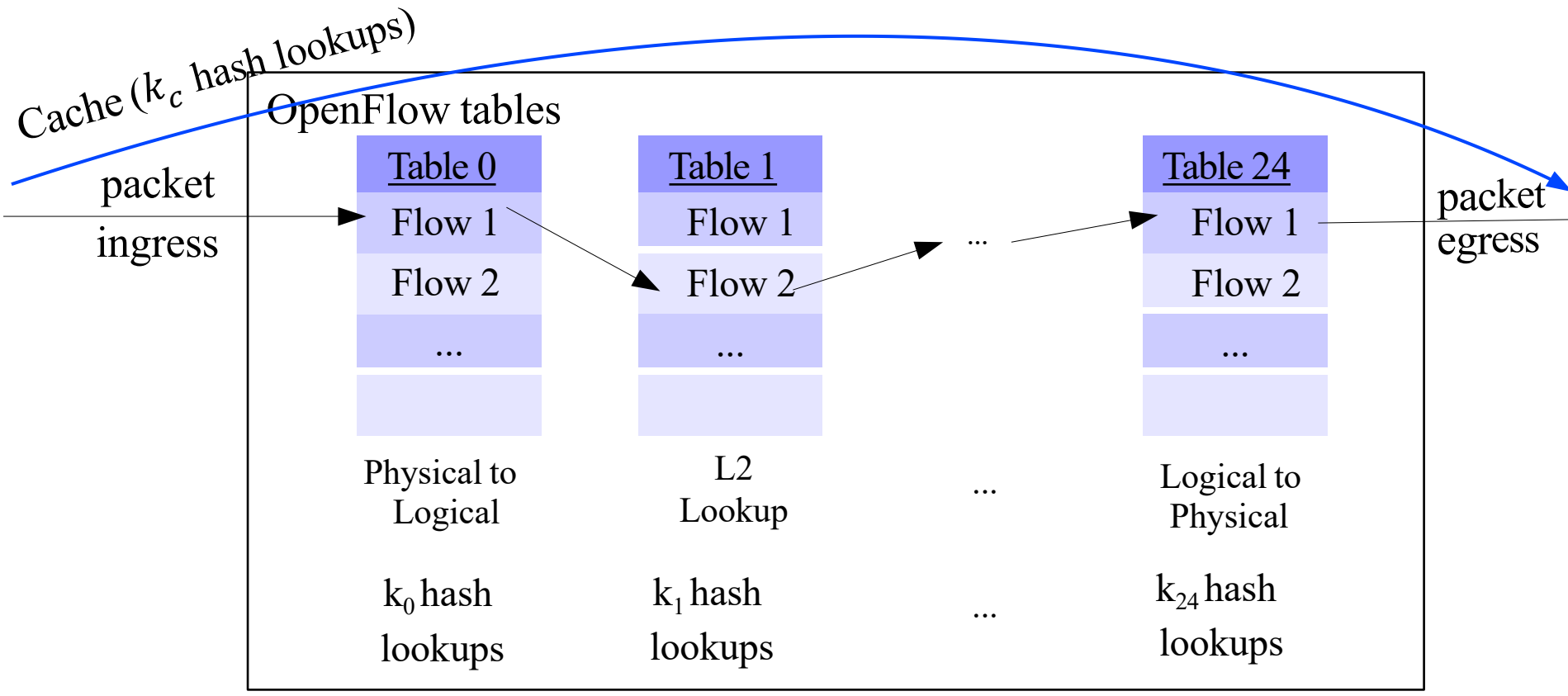


From 100+ hash lookups per packet, to just 1!

Microflow Caching in Practice

- Tremendous speedup for most workloads
- Problematic traffic patterns:
  - Port scans
    - Malicious
    - Accidental (!)
  - Peer-to-peer rendezvous applications
  - Some kinds of network testing
- All of this traffic has lots of short-lived microflows
  - Fundamental caching problem: low hit rate

Using a More Expensive Cache



If $k_c << k_0 + k_1 + \ldots + k_{24}$: benefit!

Naive Approach to Populating Cache

Combine tables 0...24 into one flow table

Easy! Usually, $k_c << k_0 + k_1 + \ldots + k_{24}$. But:

Table 0
ip_src=*a*
ip_src=*b*
ip_src=*c*
ip_src=*d*

n_1 flows

$\times$

Table 1
ip_dst=*e*
ip_dst=*f*
ip_dst=*g*
ip_dst=*h*

n_2 flows

$\times \cdots \times$

Table 24
eth_dst=*i*
eth_dst=*j*
eth_dst=*k*
eth_dst=*m*

n_{24} flows

$=$

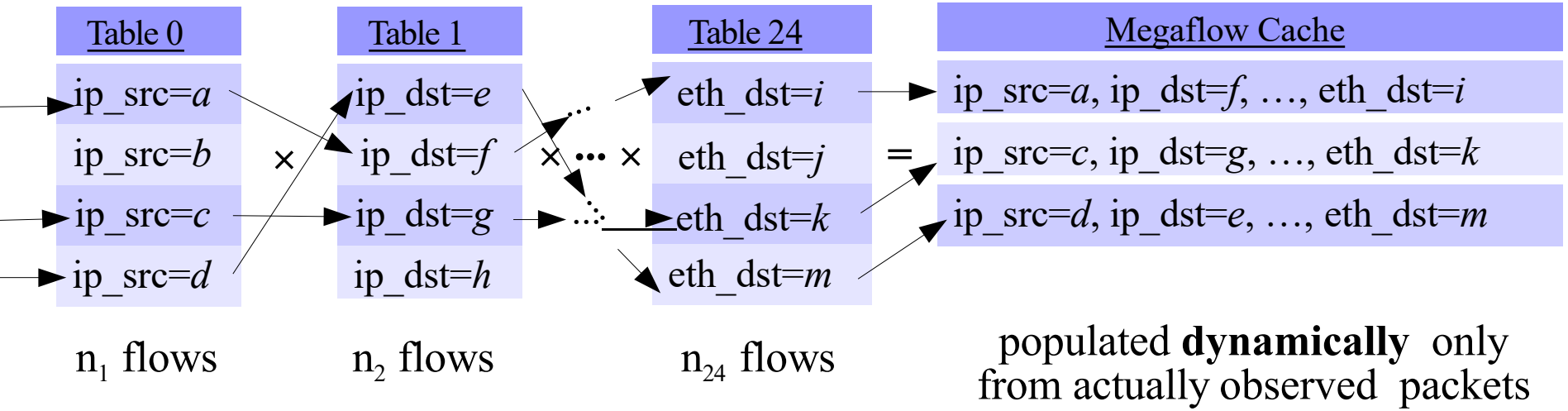
Table 0+1+...+24
ip_src=*a*, ip_dst=*e*, …, eth_dst=*i*
ip_src=*a*, ip_dst=e, …, eth_dst=*j*
ip_src=*a*, ip_dst=*e*, …, eth_dst=*k*
...
ip_src=*d*, ip_dst=*h*, …, eth_dst=*k*
ip_src=*d*, ip_dst=*h*, …, eth_dst=*m*

up to $n_1 \times n_2 \times \cdots \times n_{24}$ flows

“Crossproduct Problem”

Lazy Approach to Populating Cache

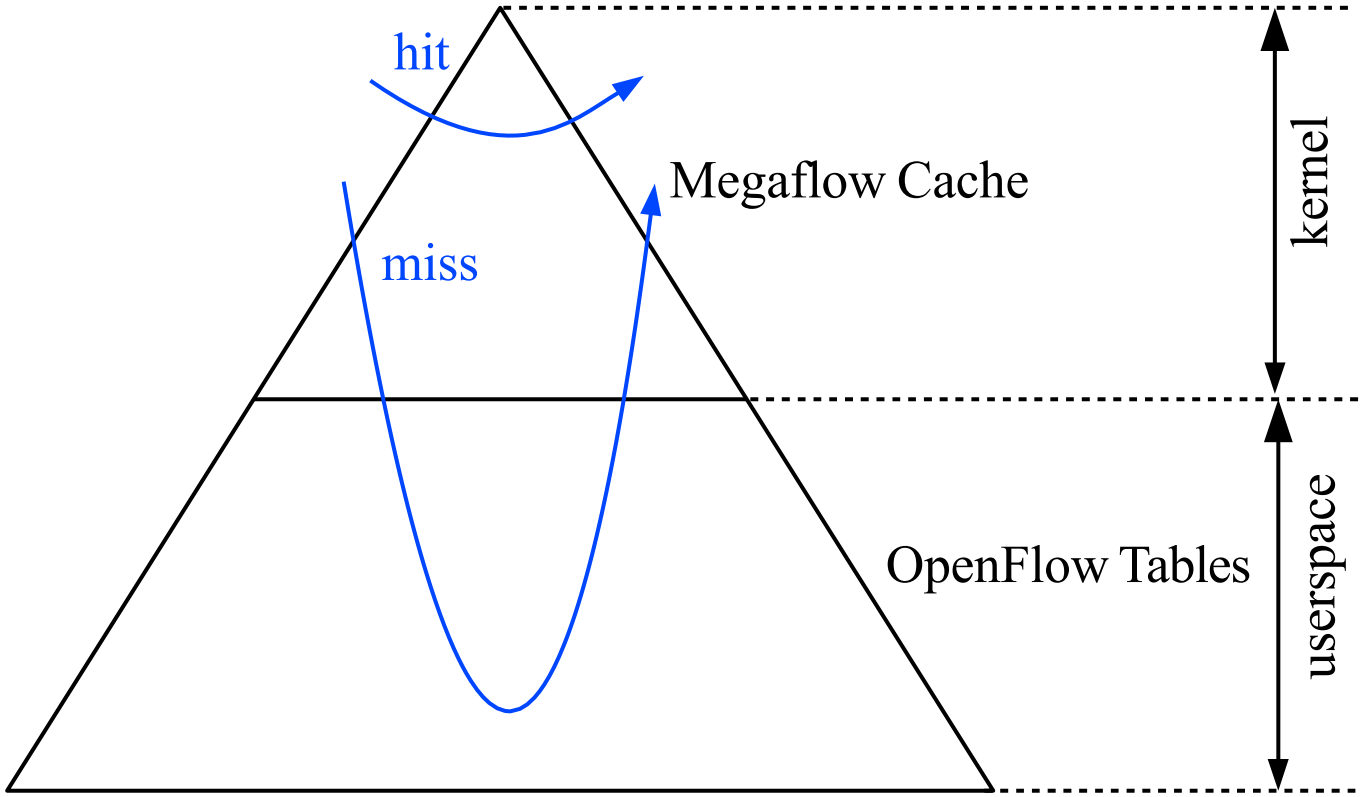
Solution: Build cache of combined "megaflows" **lazily** as packets arrive



Same (or better!) table lookups as naive approach

Traffic locality yields practical cache size

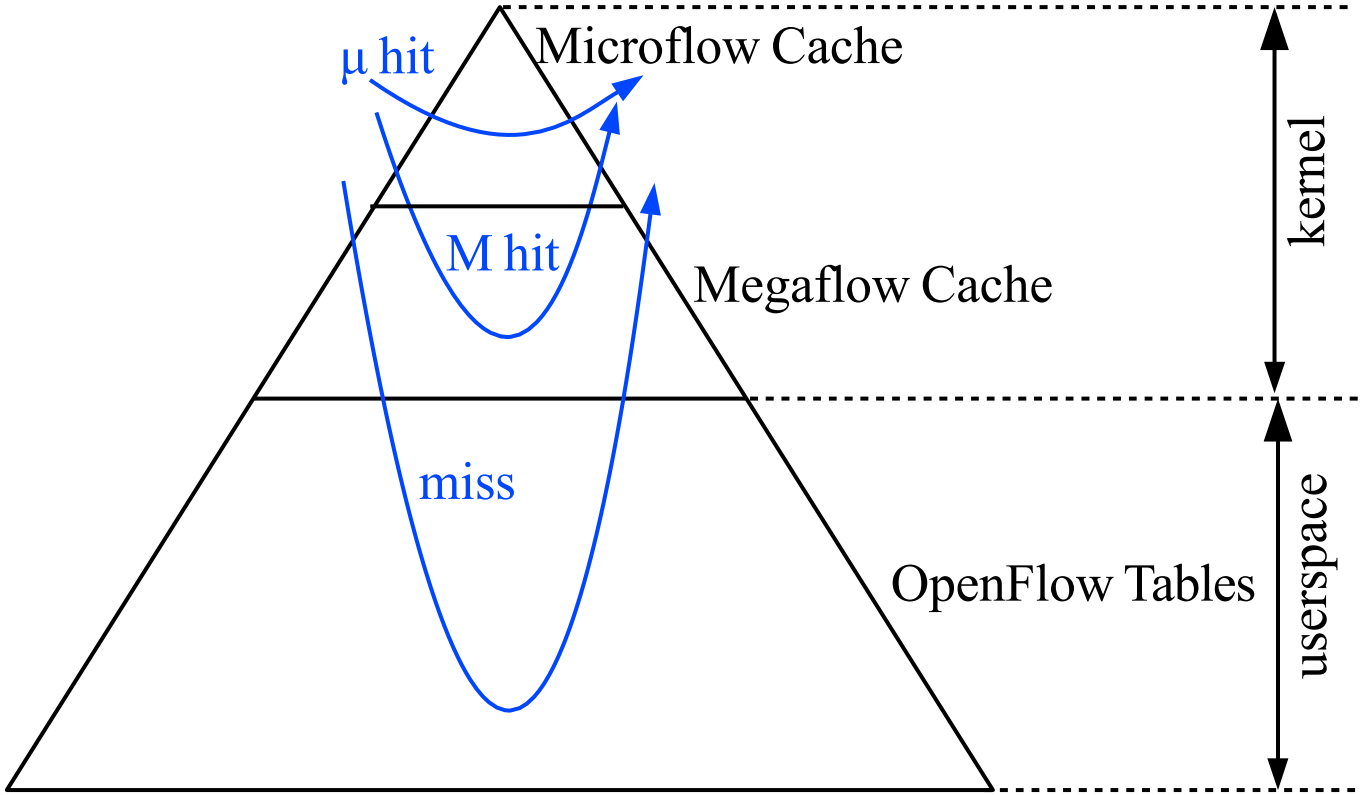
OVS Cache v2: “Megaflow” Cache



Megaflow vs. Microflow Cache Performance

- Microflow cache:
  - $k_0 + k_1 + \cdots + k_{24}$ lookups for first packet in microflow
  - 1 lookup for later packets in microflow
- Megaflow cache:
  - k_c lookups for (almost) every packet
- $k_c > 1$ is normal, so megaflows perform worse in common case!
- Best of both worlds would be:
  - k_c lookups for first packet in megaflow
  - 1 lookup for later packets in microflow

OVS Cache v3: Dual Caches



Pfaff, B., Pettit, J., Koponen, T., Jackson, E.J., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P. and Amidon, K., 2015, May. The Design and Implementation of Open vSwitch. In *NSDI.[pdf]*