Network Verification

Peng Zhang
School of Computer Science and Technology
Xi'an Jiaotong University

Spring 2024

Network Outages Are Common

Google accidentally broke the internet throughout Japan

A mistake led to internet outages for about

Southwest CEO: Router failure that grounded flights equated to 'once-in-a-thousand-year flood'

Microsoft: misconfigured network device led to Azure outage

BlackBerry outage for three days caused by faulty router says former RIM staffer

Alibaba Cloud services go down for 30 minutes, interrupts Taobao and Tmall backend

JUN 28, 2018 | IN HEAVY HITTERS | BY JIEFEI LIU

BUSINESS › LOCAL COMPANIES

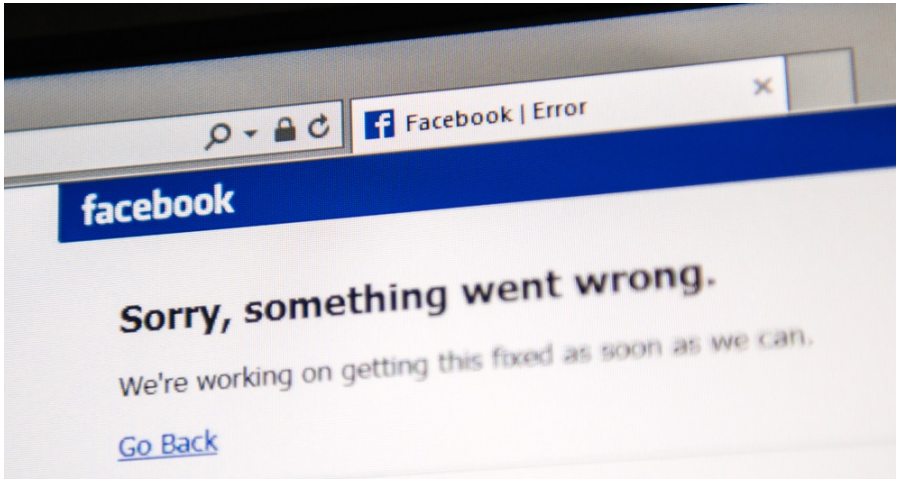
Southwest CEO: Router failure that grounded flights equated to 'once-in-a-thousand-year flood'

In an industry built on managing risk and preparing for the unexpected, this was something Southwest executives never saw coming.

Over the next four days, **2,300 flights were canceled**, and hundreds of thousands of customers were impacted, ultimately costing the airline tens of millions of dollars.

Stranded passengers at the New Orleans Airport after a Southwest Airlines computer outage caused massive delays. PETER KING/CBS NEWS

On Oct. 4, 2021, Facebook's apps including Facebook, Instagram, Messenger, WhatsApp and Oculus, disappeared from the internet for nearly six hours.



Facebook's massive outage costs the company an estimated $60 million in revenue

The social network is losing ad dollars every second its platforms remain offline.

Wealth

Zuckerberg Loses $6 Billion in Hours as Facebook Plunges

POSTED ON OCTOBER 4, 2021 TO NETWORKING & TRAFFIC

Update about the October 4th outage

Our engineering teams have learned that configuration changes on the backbone routers that coordinate network traffic between our data centers caused issues that interrupted this communication. This disruption to network traffic had a cascading effect on the way our data centers communicate, bringing our services to a halt.

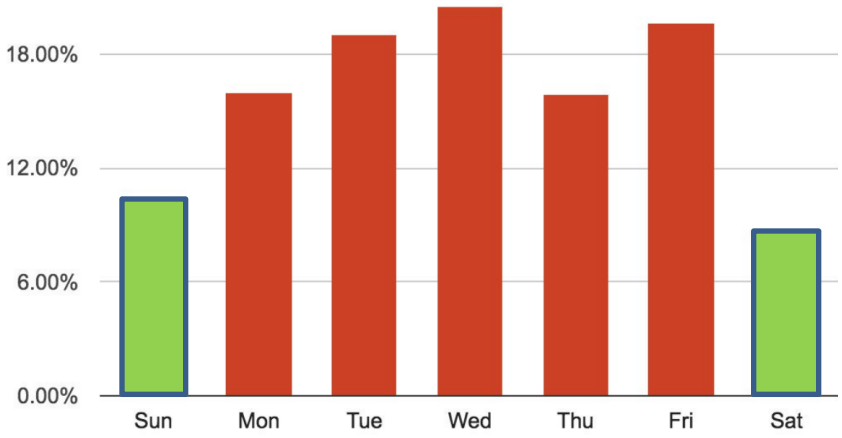
The Root Cause of Network Outages

"75% of network outages and performance issues result from misconfiguration"

-- Google cloud

"Human factors are responsible for 50% to 80% of network outages"

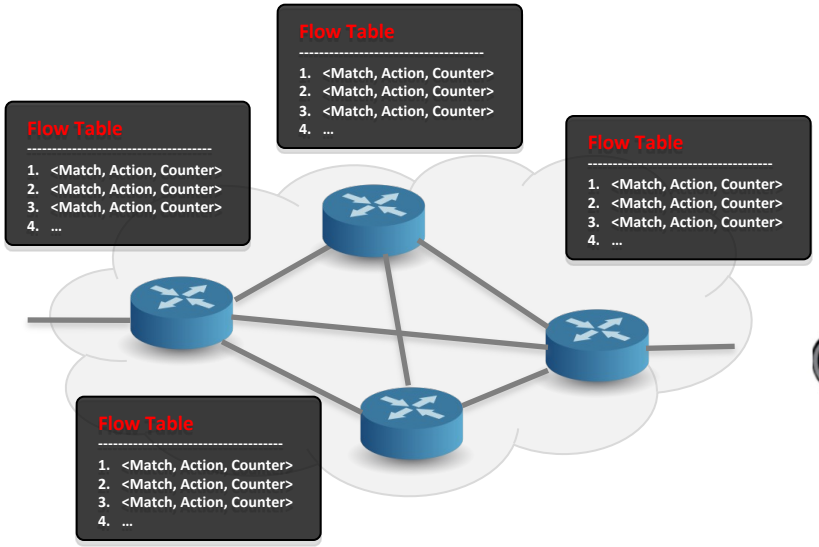
-- Juniper network



"Fun fact: most BGP route leaks happen on Wednesdays, but in the weekend us humans collectively take a break! :-)"

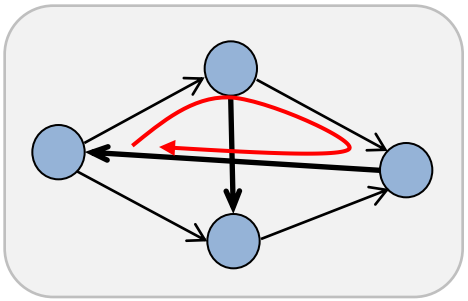
-- Job Snijders, NTT

Data Plane Verification



Physical/Virtual Network

Take a Snapshot of the network

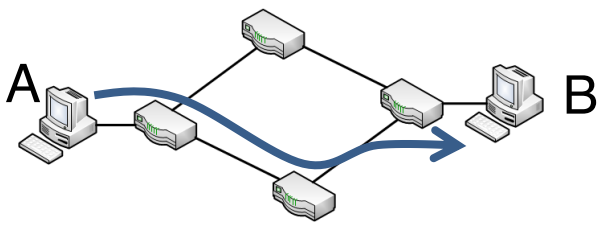


Network Model

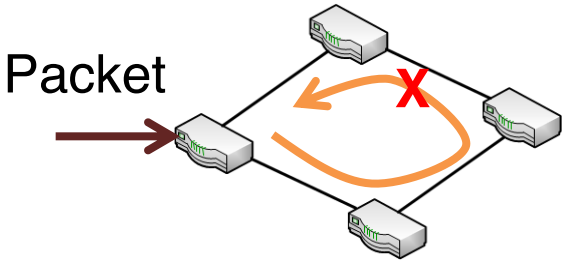
What to Verify

- We want to verify

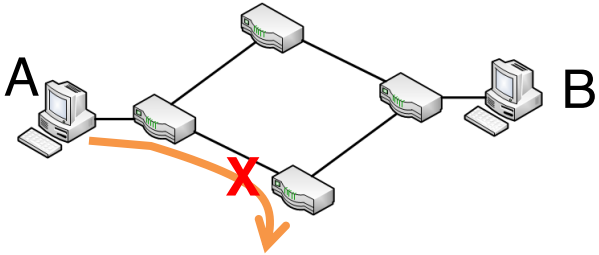
Pairwise Reachability



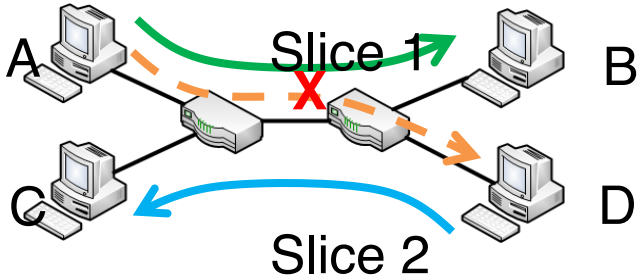
Loop Freedom



Blackhole Freedom

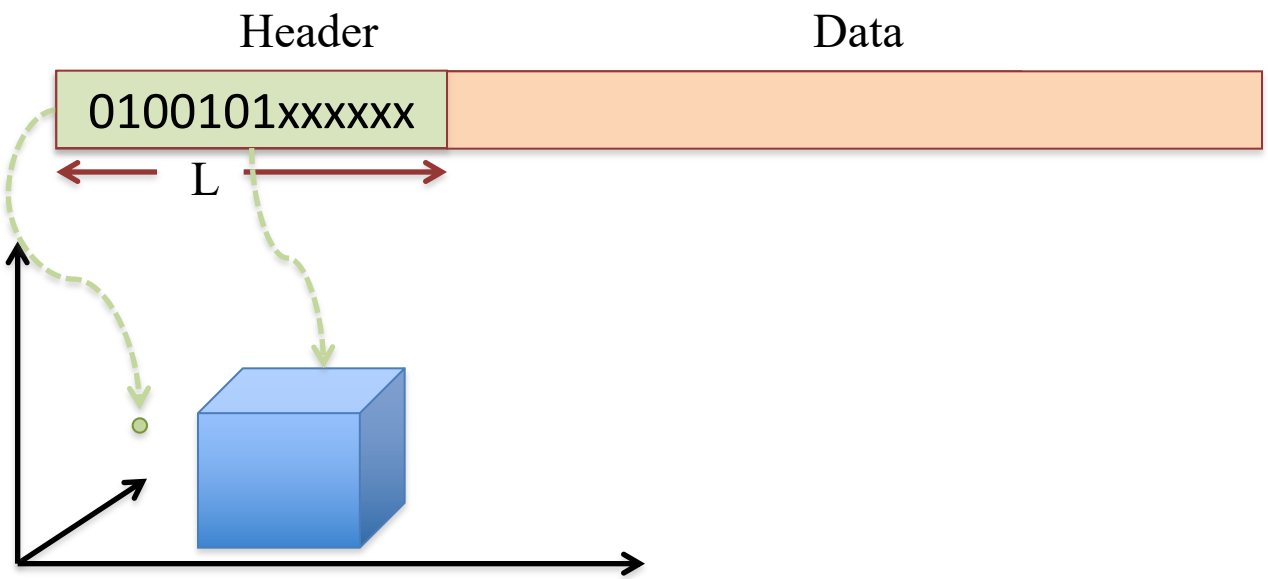


Traffic Isolation



Header Space Analysis

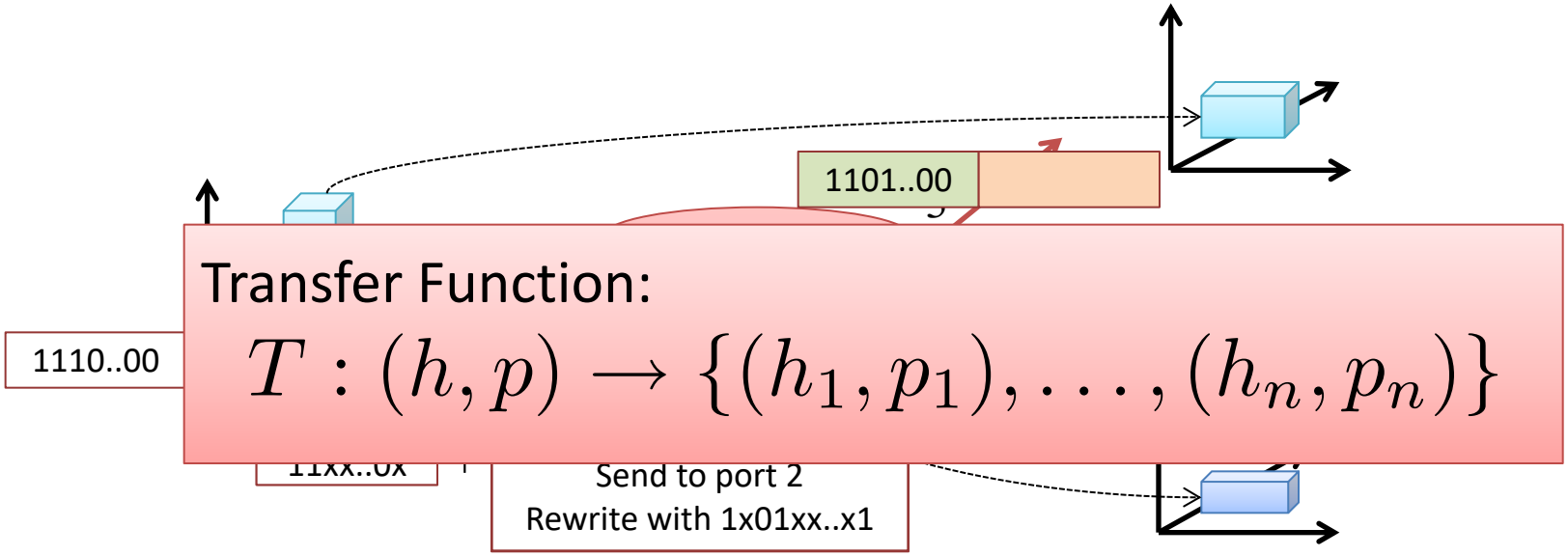
- Step 1 – Model a packet (header) as a point in $\{0,1\}^L$ space, i.e., the header space
  - Flat space, protocol oblivious
  - Wildcard expression (e.g., IP prefix) -> hyper cubes



Kazemian et al., Header Space Analysis: Static Checking For Networks, in NSDI'12

Header Space Analysis

- Step 2 – Model all networking boxes as transformers of header space

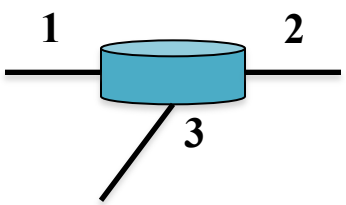


Transfer Function:

$$T : (h, p) \longrightarrow \{(h_1, p_1), \ldots, (h_n, p_n)\}$$

Transfer Function Example

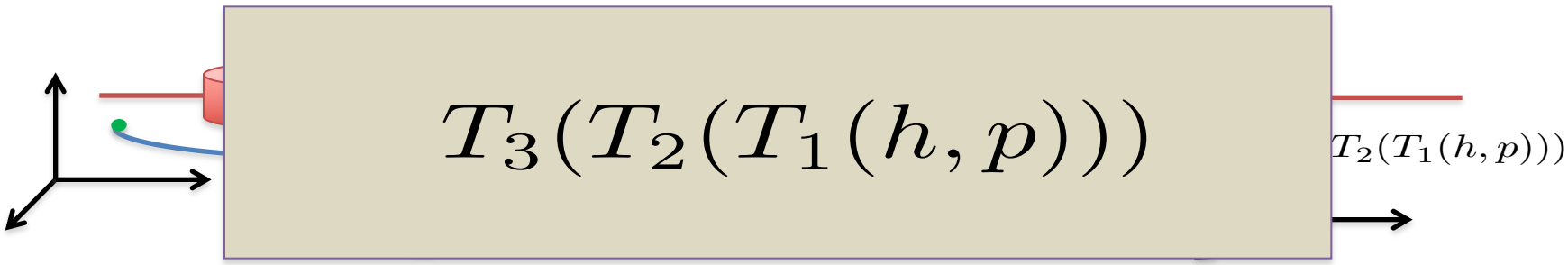
- IPv4 Router – forwarding + TTL + MAC rewrite
  - 172.24.74.x Port1
  - 172.24.128.x Port2
  - 171.67.x.x Port3



$$T(h, p) = \begin{cases} (\text{rw_mac}(\text{dec_ttl}(h), \text{next_mac}), 1) & \text{if dst_ip}(h) = 172.24.74.x \\ (\text{rw_mac}(\text{dec_ttl}(h), \text{next_mac}), 2) & \text{if dst_ip}(h) = 172.24.128.x \\ (\text{rw_mac}(\text{dec_ttl}(h), \text{next_mac}), 3) & \text{if dst_ip}(h) = 171.67.x.x \end{cases}$$

Theorems

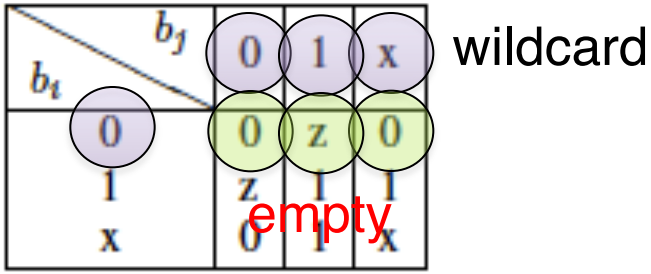
- Composition Theorem
  - Network behavior = Composition of transfer functions



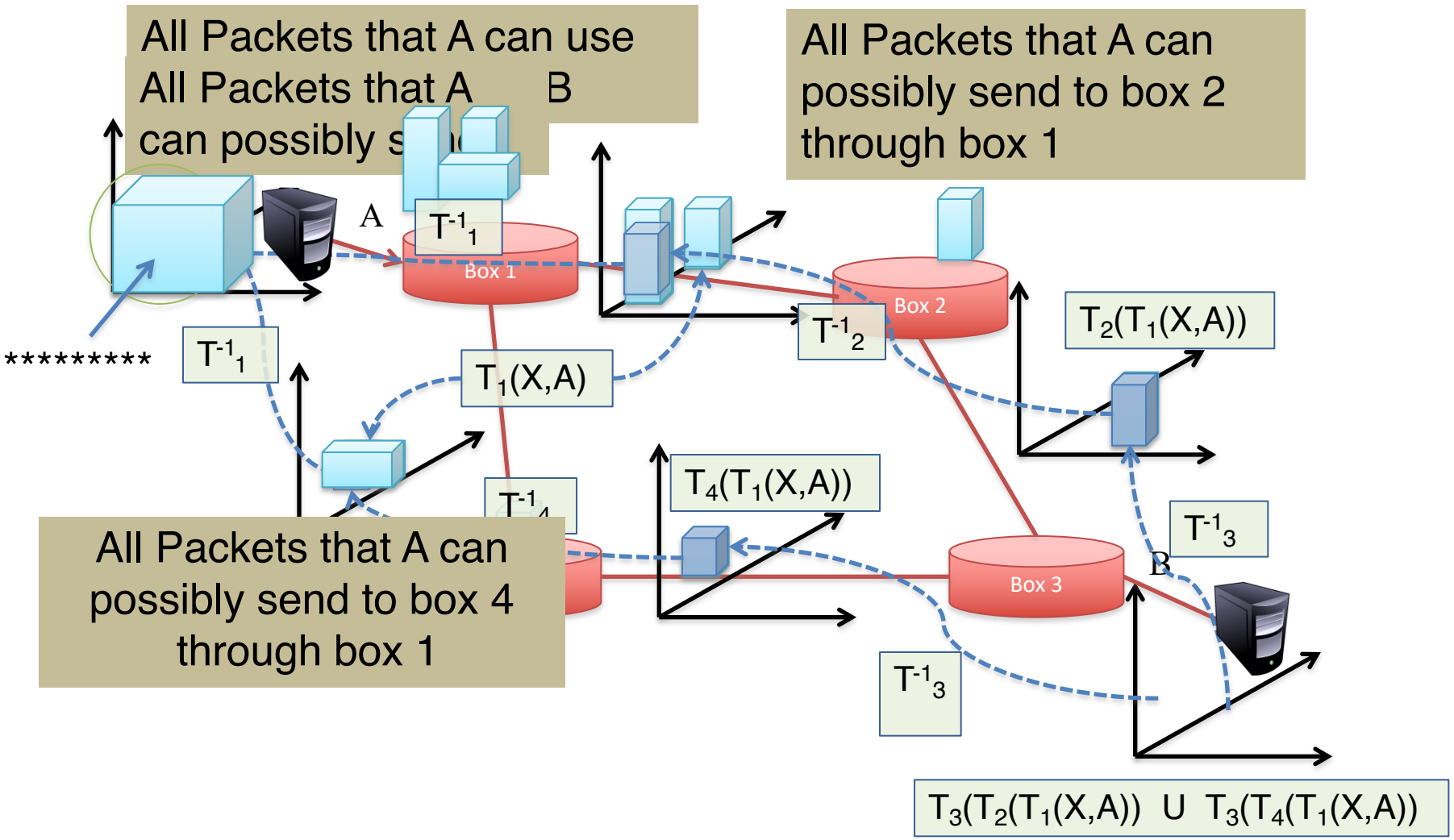
- Inversion Theorem
  - given header h at destination p, we can invert to find (h',p'): headers sent at source s' to produce (h,p)

HSA Algebra

- Bit by bit intersect using intersection table:
  - Example: $10xx \cap 1xx0 = 10x0$
  - If result has any ‘z’, then intersection is empty:
  - Example: $10xx \cap 0xx0 = z0x0 = \phi$

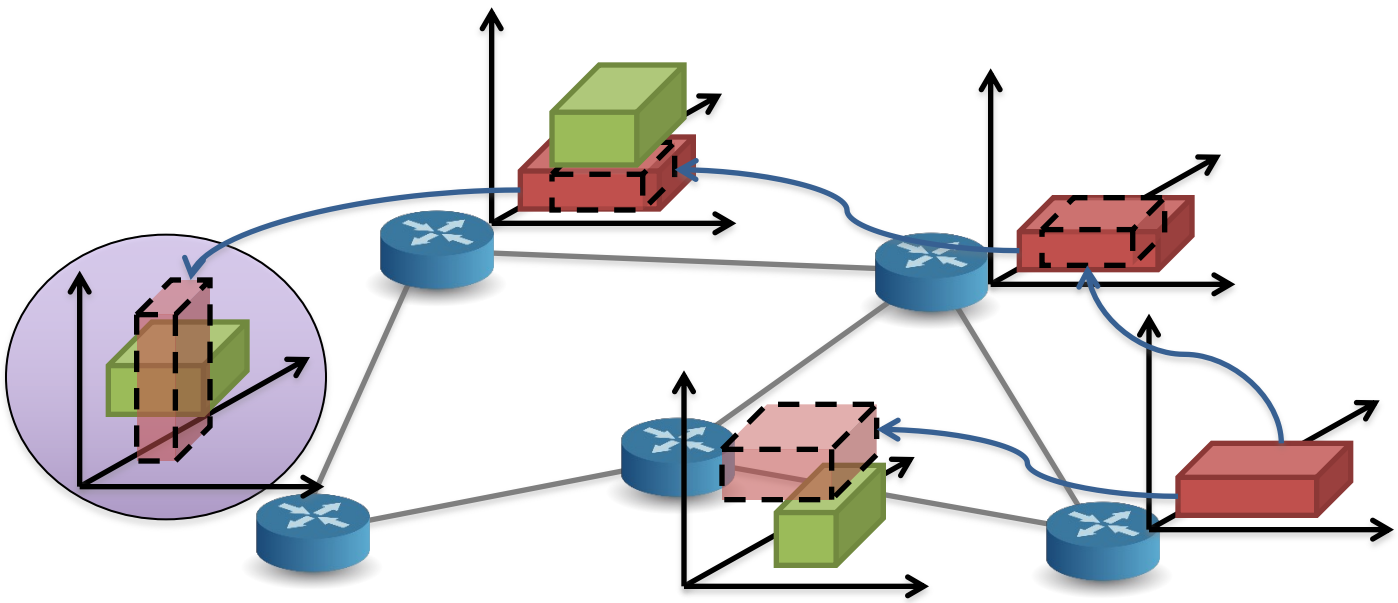


Use Case 1: Computing Reachability



Use Case 2: Checking Isolation

- Slice definitions don’t intersect.
- Packets don’t leak after forwarding.



Problems with HSA

- Only check a snapshot of network configuration

- Networks are dynamically changing
- What if a new rule is inserted?
  - checking the entire network's state every time a new flow is wasteful and slow

What is VeriFlow?

VeriFlow: Verifying Network-Wide Invariants in Real Time

Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, P. Brighten Godfrey
University of Illinois at Urbana-Champaign
{khurshi1, xuanzou2, wzhou10, caesar, pbg}@illinois.edu

Abstract

Networks are complex and prone to bugs. Existing tools that check network configuration files and the data-plane state operate offline at timescales of seconds to hours, and cannot detect or prevent bugs as they arise.

Is it possible to *check network-wide invariants in real time*, as the network state evolves? The key challenge here is to achieve extremely low latency during the checks so that network performance is not affected. In this paper, we present a design, VeriFlow, which achieves this goal. VeriFlow is a layer between a software-defined networking controller and network devices that checks for network-wide invariant violations dynamically as each forwarding rule is inserted, modified or deleted. VeriFlow supports analysis over multiple header fields, and an API for checking custom invariants. Based on a prototype implementation integrated with the NOX OpenFlow controller, and driven by a Mininet OpenFlow network and Route Views trace data, we find that VeriFlow can perform rigorous checking within hundreds of microseconds per rule insertion or deletion.

1 Introduction

Packet forwarding in modern networks is a complex process, involving codependent functions running on hundreds or thousands of devices, such as routers, switches, and firewalls from different vendors. As a result, a substantial amount of effort is required to ensure networks' correctness, security and fault tolerance. However, faults in the network state arise commonly in practice, including loops, suboptimal routing, black holes and access control violations that make services unavailable or prone to attacks (e.g., DDoS attacks). Software-Defined Networking (SDN) promises to ease the development of network applications through logically-centralized network programmability via an open interface to the data plane, but bugs are likely to remain problematic since the complexity of software will increase. Moreover, SDN allows multiple applications or even multiple users to program the same physical network simultaneously, potentially resulting in conflicting rules that alter the intended behavior of one or more applications [25].

One solution is to rigorously check network software or configuration for bugs prior to deployment. Symbolic execution [12] can catch bugs through exploration of all possible code paths, but is usually not tractable for large software. Analysis of configuration files [13, 28] is useful, but cannot find bugs in router software, and must be designed for specific configuration languages and control protocols. Moreover, using these approaches, an operator who wants to ensure the network's correctness must have access to the software and configuration, which may be inconvenient in an SDN network where controllers can be operated by other parties [25]. Another approach is to statically analyze snapshots of the network-wide data-plane state [9, 10, 17, 19, 27]. However, these previous approaches operate offline, and thus only find bugs after they happen.

This paper studies the question, *Is it possible to check network-wide correctness in real time as the network evolves?* If we can check each change to forwarding behavior before it takes effect, we can raise alarms immediately, and even prevent bugs by blocking changes that violate important invariants. For example, we could prohibit changes that violate access control policies or cause forwarding loops.

However, existing techniques for checking networks are inadequate for this purpose as they operate on timescales of seconds to hours [10, 17, 19]. [1] Delaying updates for processing can harm consistency of network state, and increase reaction time of protocols with real-time requirements such as routing and fast failover; and processing a continuous stream of updates in a large

[1] The average run time of reachability tests in [17] is 13 seconds, and it takes a few hundred seconds to perform reachability checks in Anteater [19].

VeriFlow: Verifying Network-Wide Invariants in Real Time, **USENIX NSDI 2013**

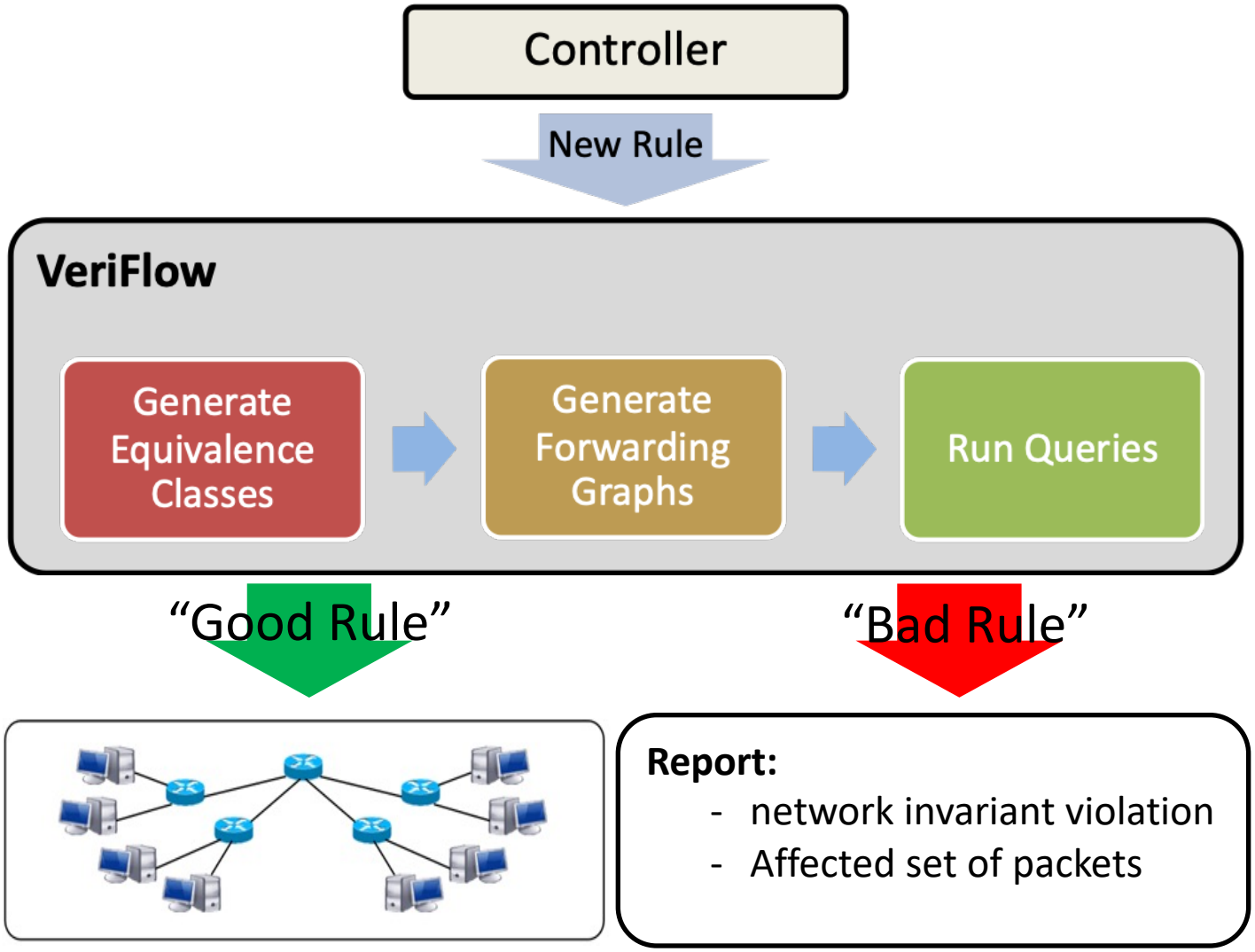
VERIFLOW

Aug. 2019
acquired by

vmware®

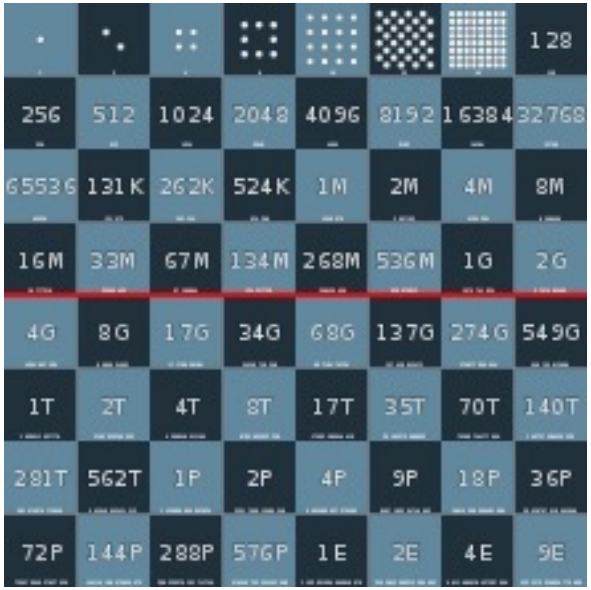
https://www.vmware.com/company/acquisitions/veriflow.html

VeriFlow


Controller
New Rule
VeriFlow
Generate Equivalence Classes
Generate Forwarding Graphs
Run Queries
"Good Rule"
"Bad Rule"
Report:
- network invariant violation
- Affected set of packets

Why Challenging?

- Huge space of headers, impossible to enumerate
  - IP destination (IPv4): 2^{32}
  - IP 5-tuple: $2^{32} \times 2^{32} \times 2^{16} \times 2^{16} \times 2^{8}$

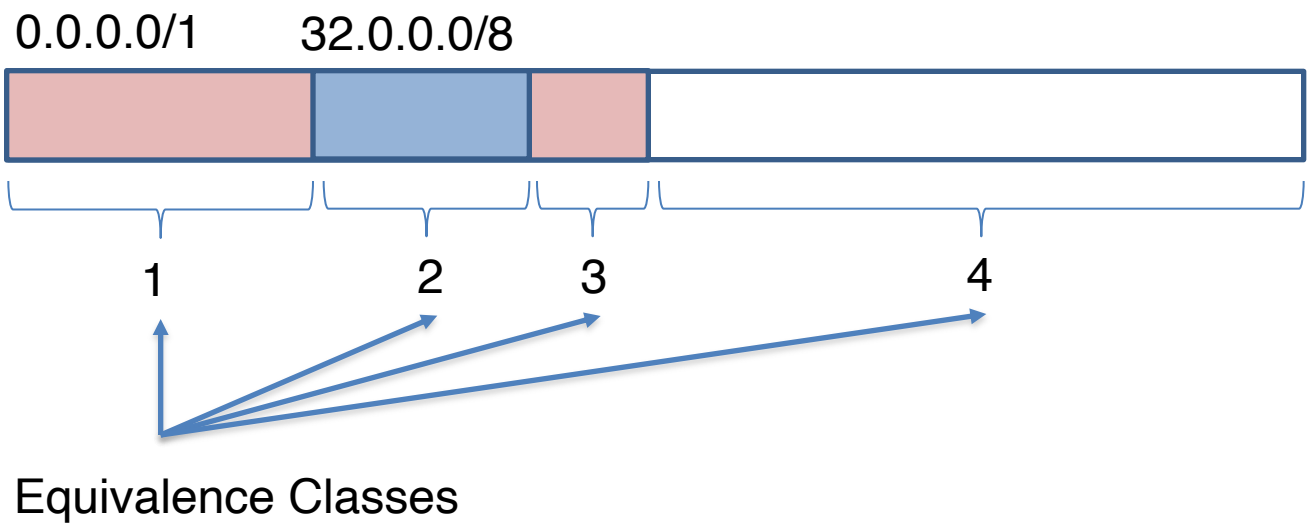


$2^{64} - 1 = 18{,}446{,}744{,}073{,}709{,}551{,}615$

https://en.wikipedia.org/wiki/Wheat_and_chessboard_problem

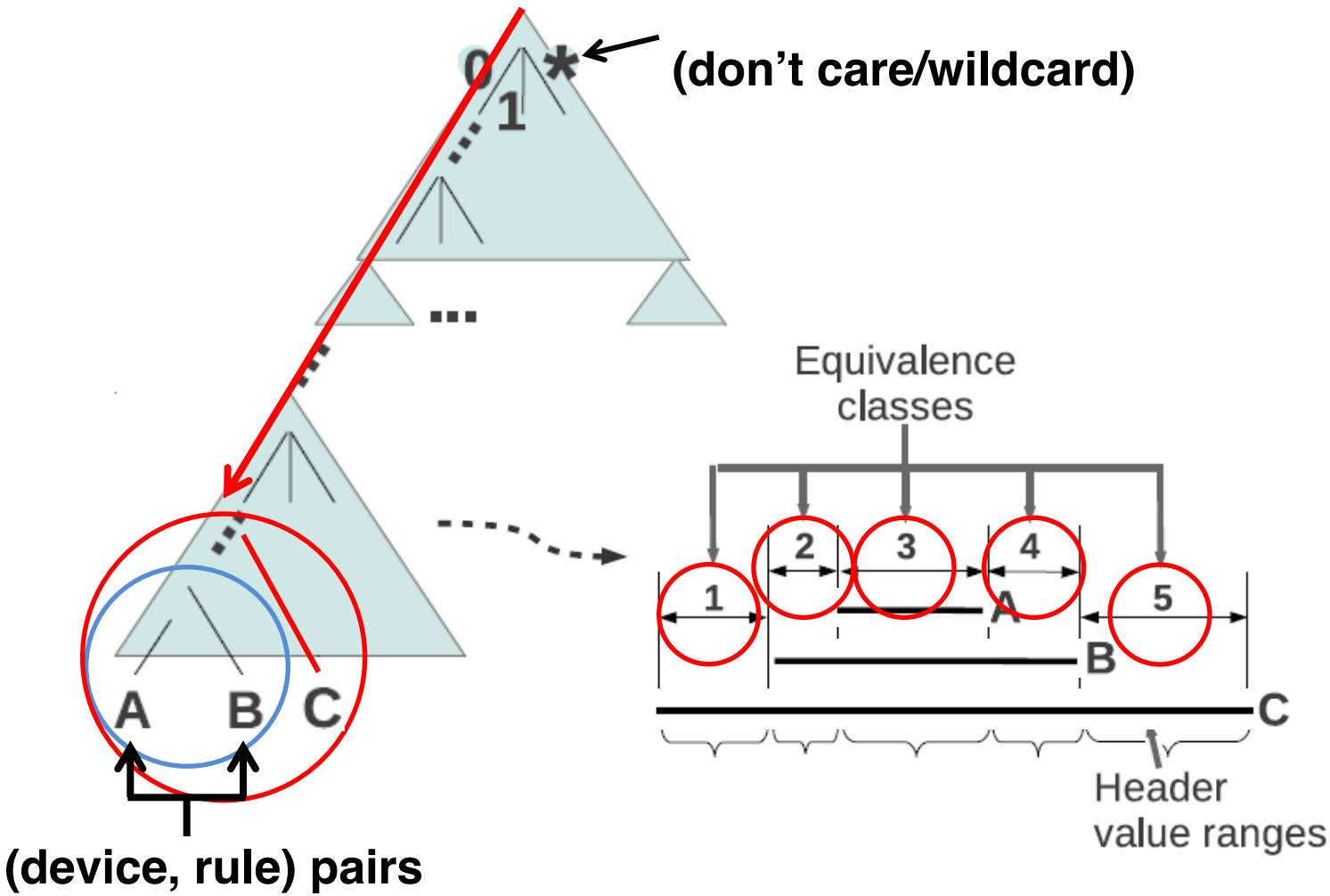
Step1: Generate Equivalence Class

- **Equivalence class (EC):** packets experiencing the same forwarding actions throughout the network



Step1: Generate Equivalence Class

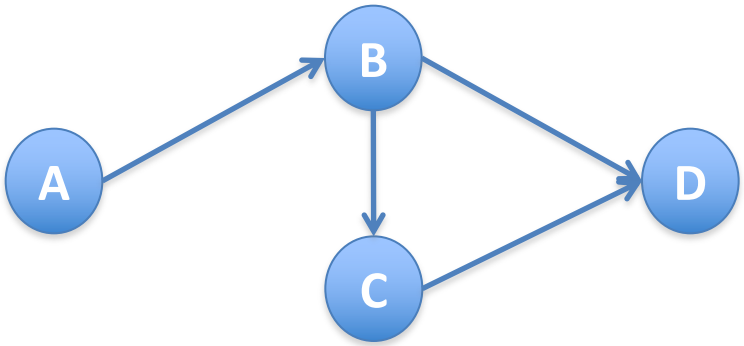
- Data structure for EC computation
  - Multidimensional Prefix Tree (Trie)



Step2: Generate Forwarding Graph

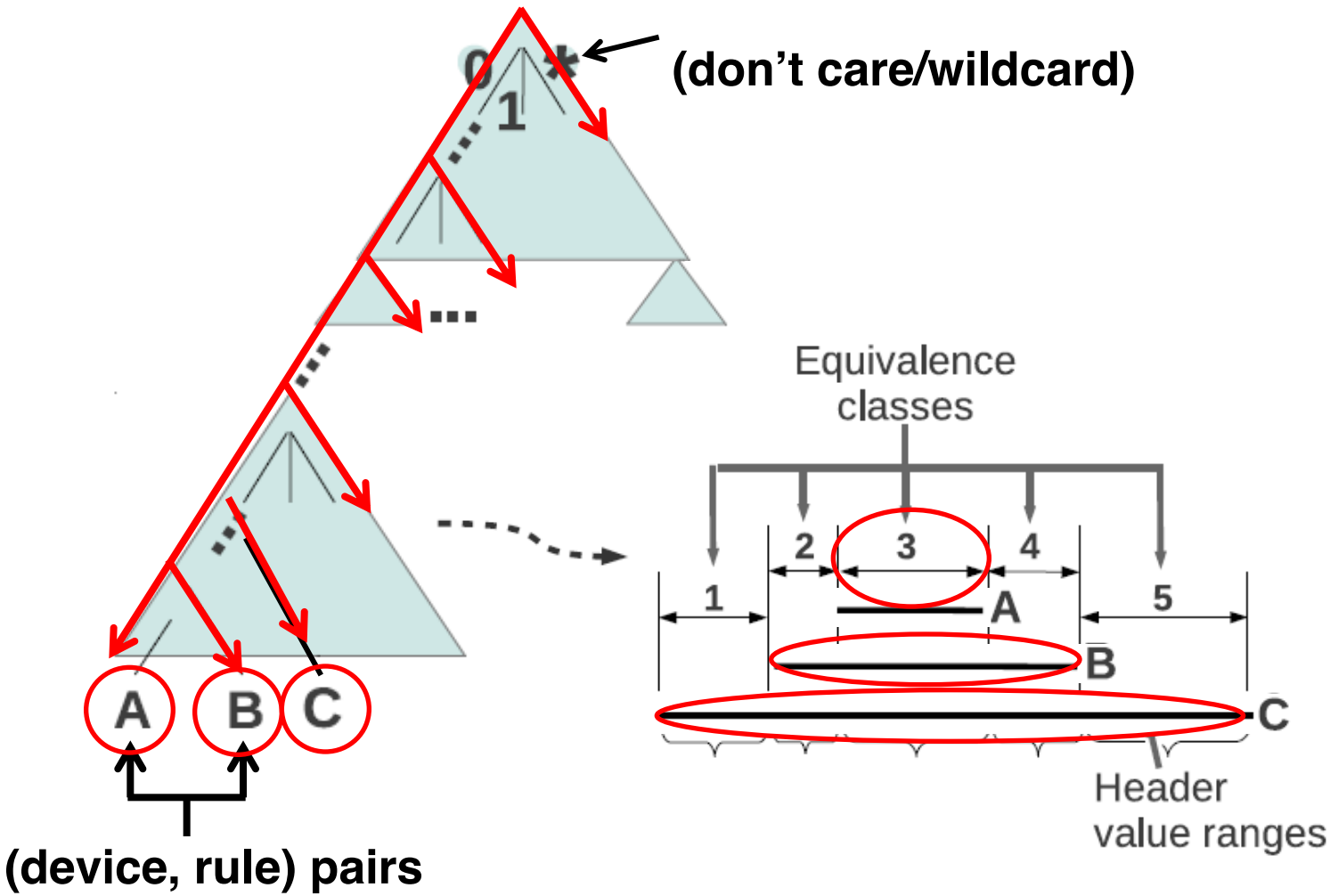
- Generate a forwarding graph for each EC
  - Defines how packets within that EC will be forwarded through the network
  - Node: device, edge: forwarding rule
  - Example:

EC:
dst_ip=10.0.0.0/24



Step2: Generate Forwarding Graph

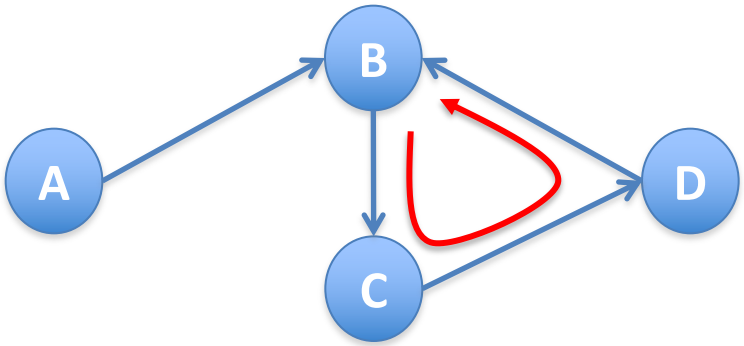
- Let the EC traverse the Trie for the second time



Step3: Run Queries (Verifying)

- Check whether the forwarding graph of each EC satisfies
  - Reachability
  - Loop-freedom
  - Blackhole-freedom

EC:
dst_ip=10.0.0.0/24

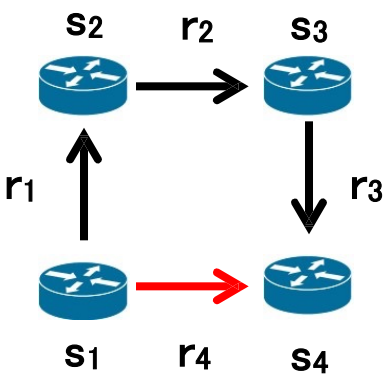


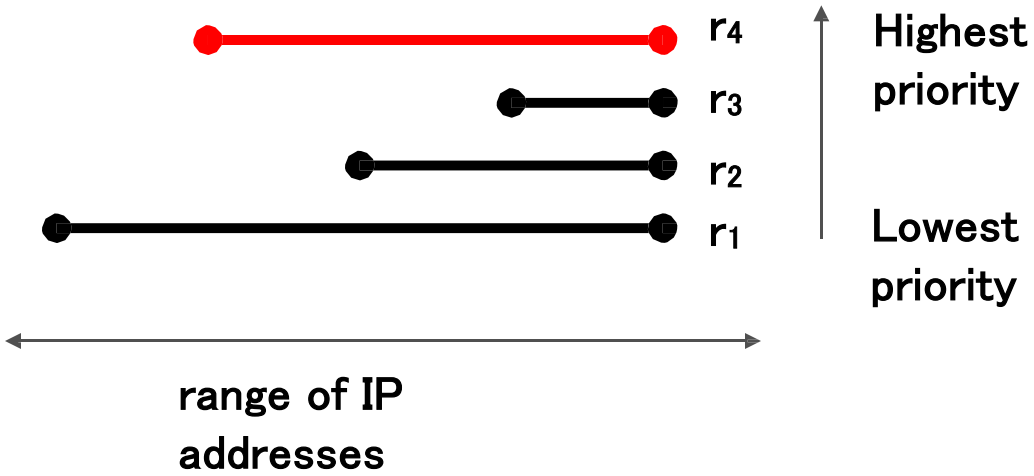
Source Code

- Not Open Sourced
- But…

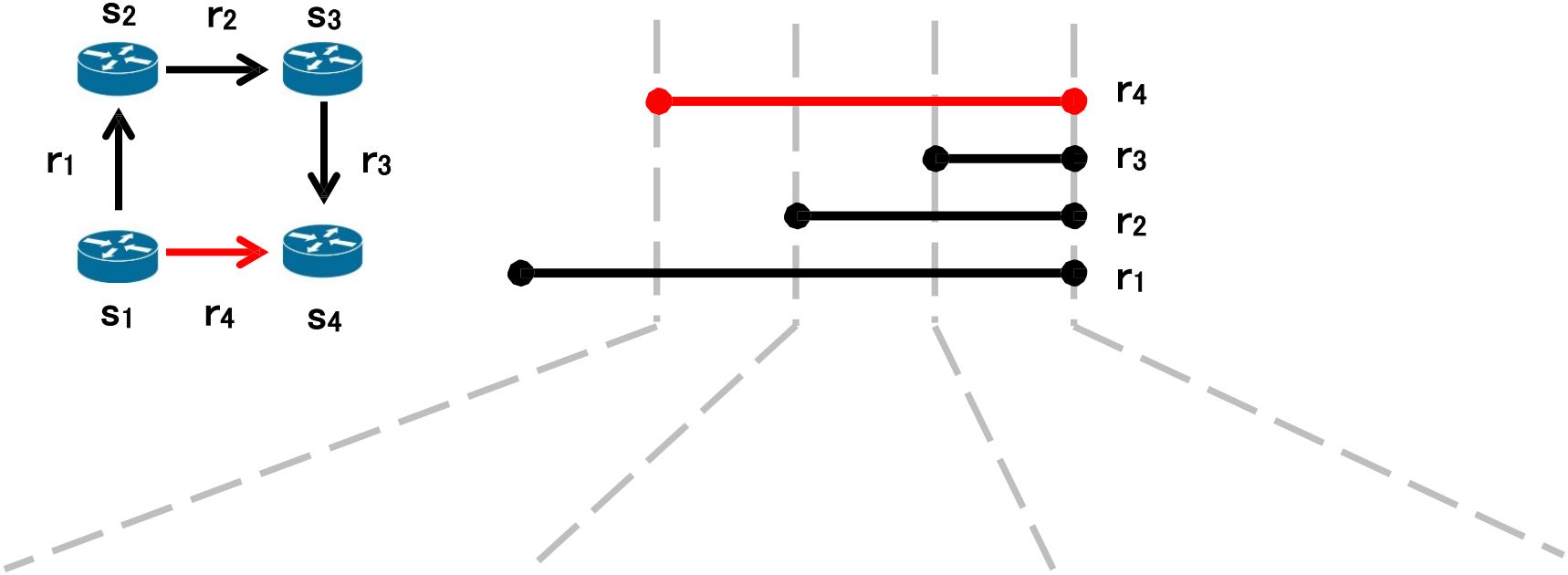
https://github.com/samueljero/BEADS/tree/master/veriflow

Inefficiency of VeriFlow

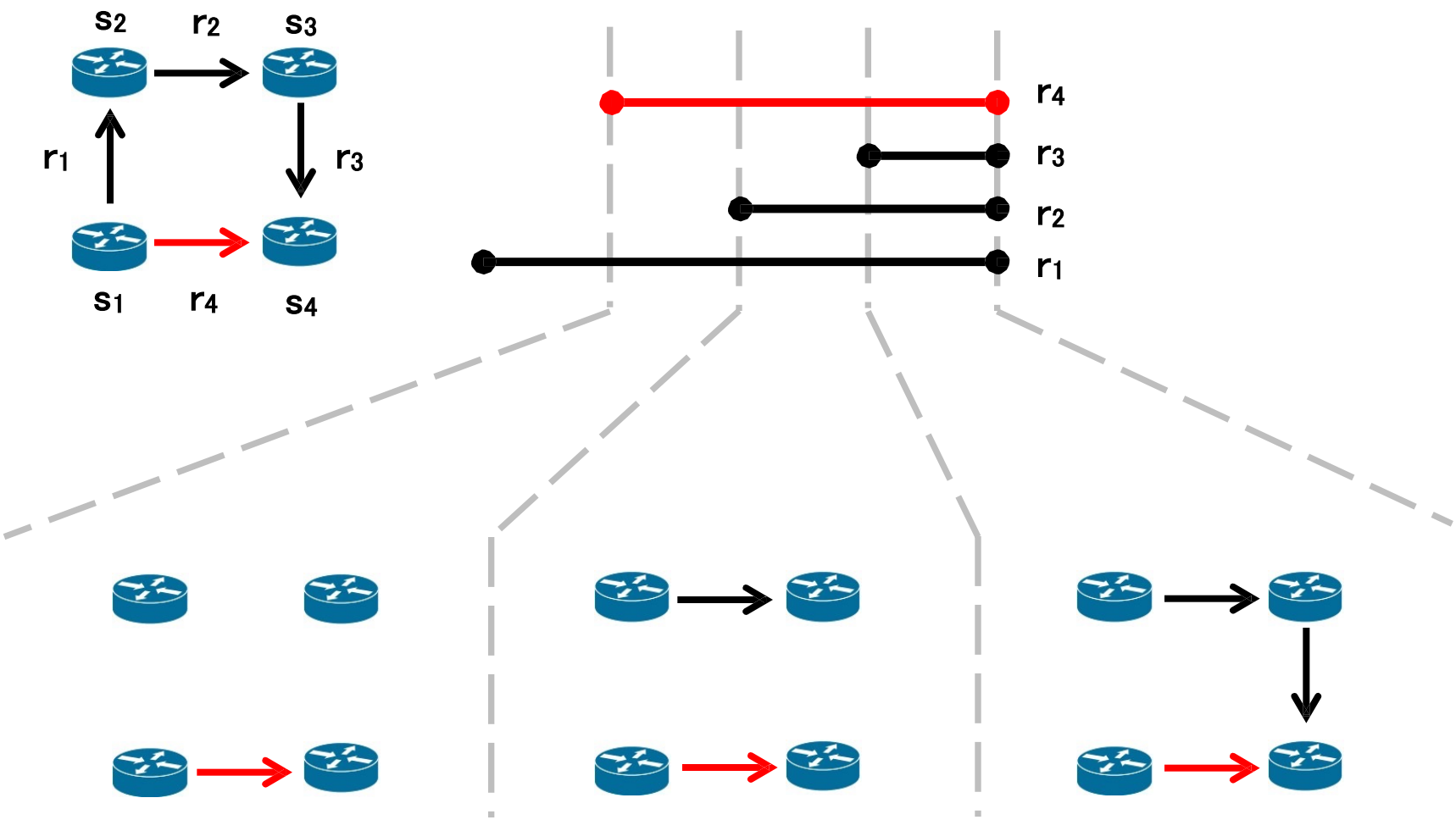




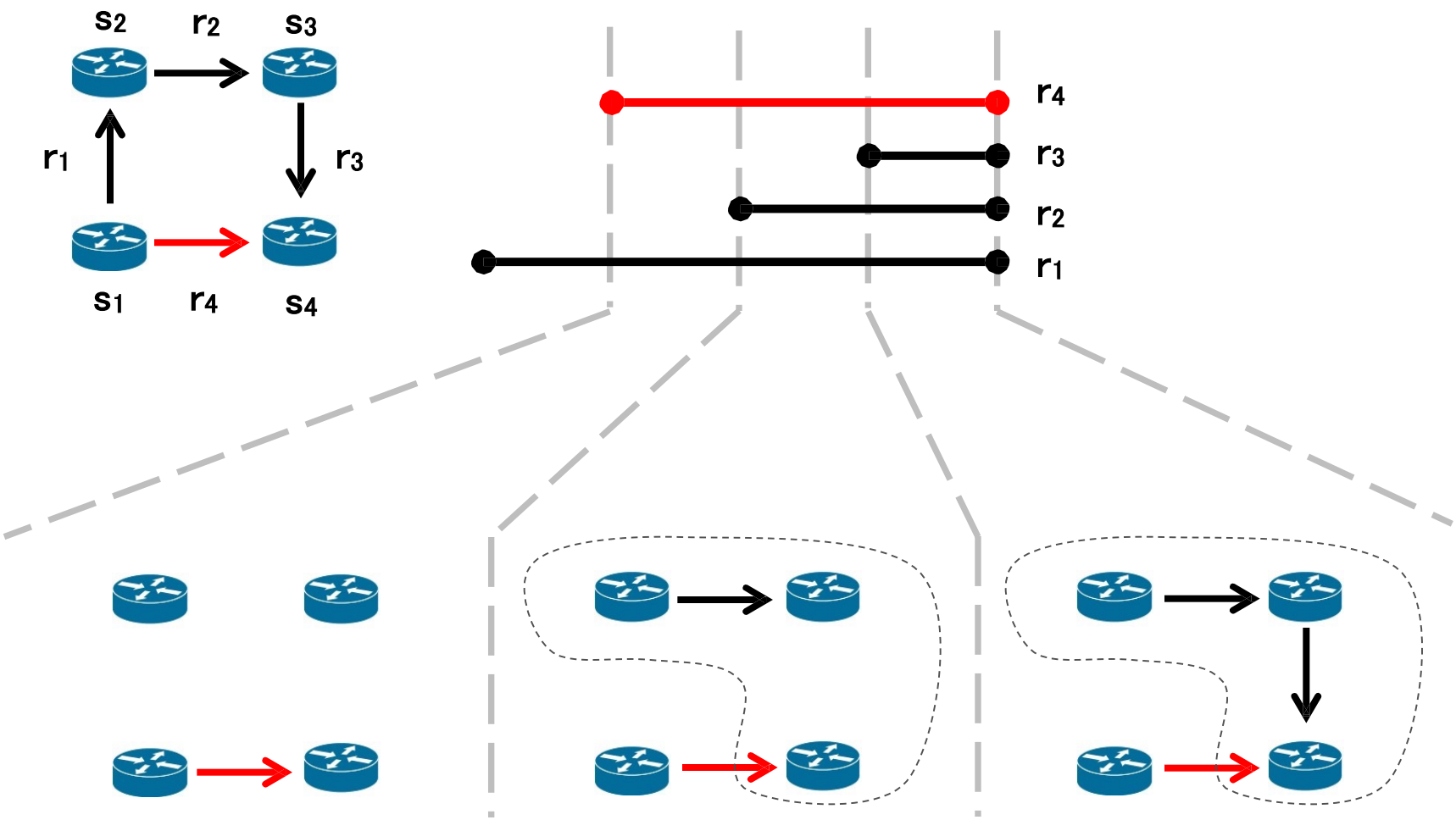
Inefficiency of VeriFlow



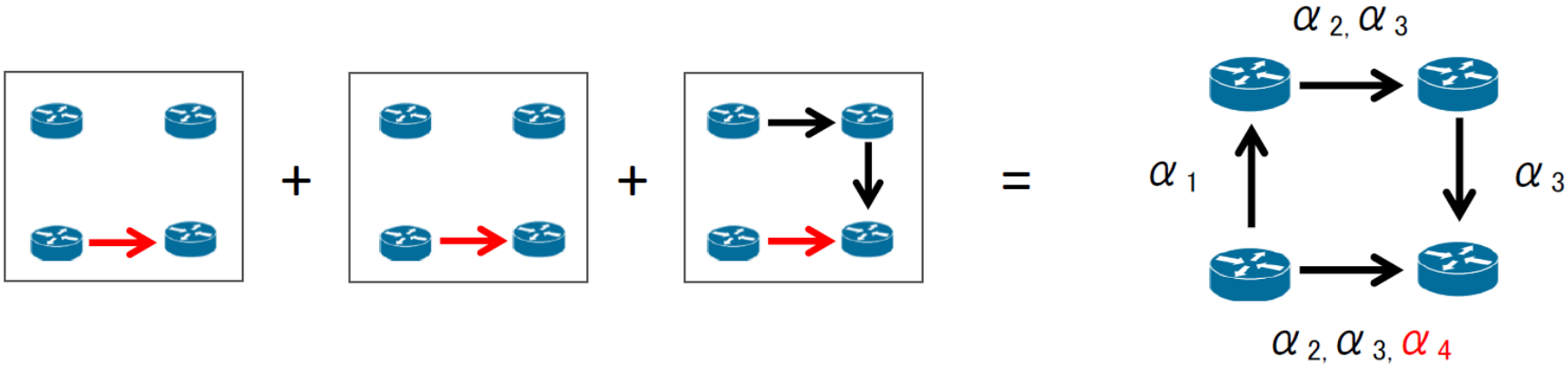
Inefficiency of VeriFlow



Inefficiency of VeriFlow



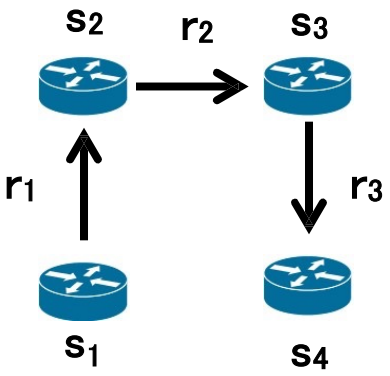
Delta-net [NSDI'17]

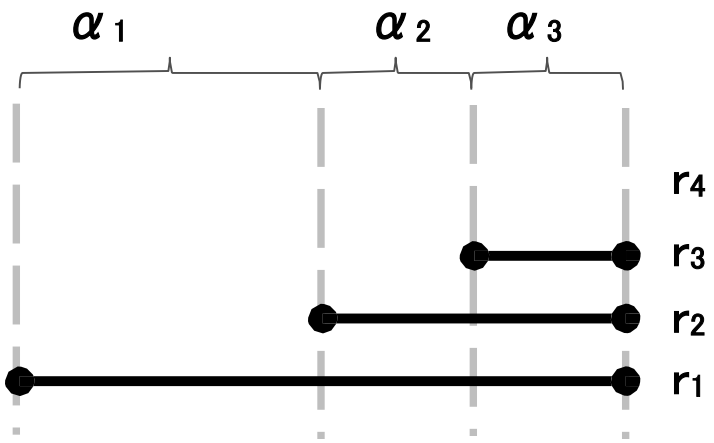


Rather than re-computing forwarding graphs ...

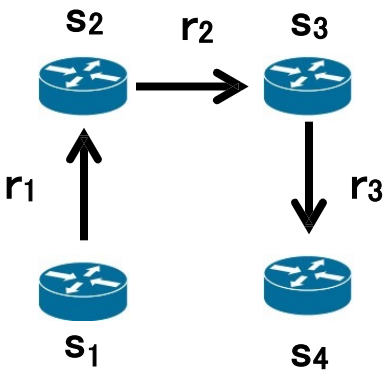
incrementally maintain a single edge-labelled graph; represents *all* packet flows.

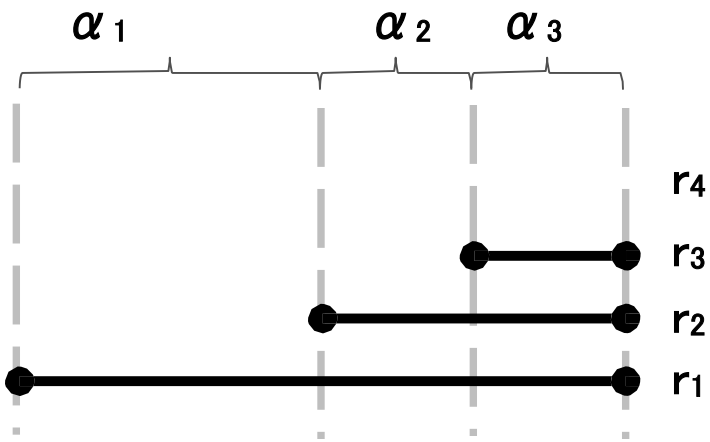
Delta-net [NSDI'17]

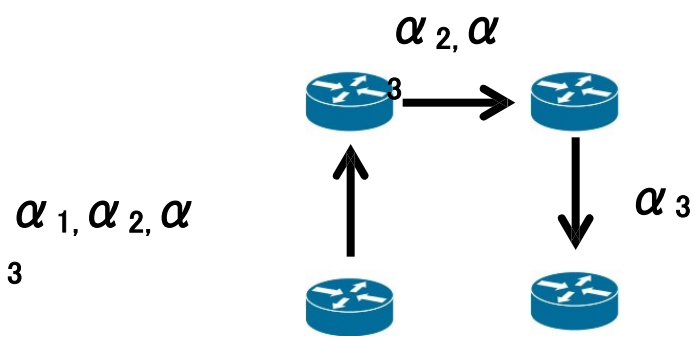




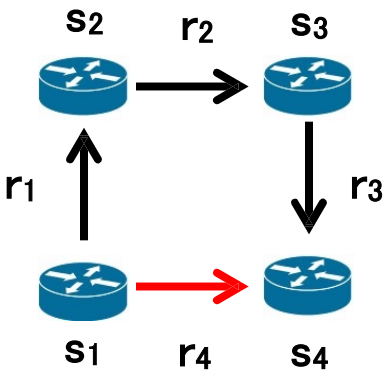
Delta-net [NSDI'17]

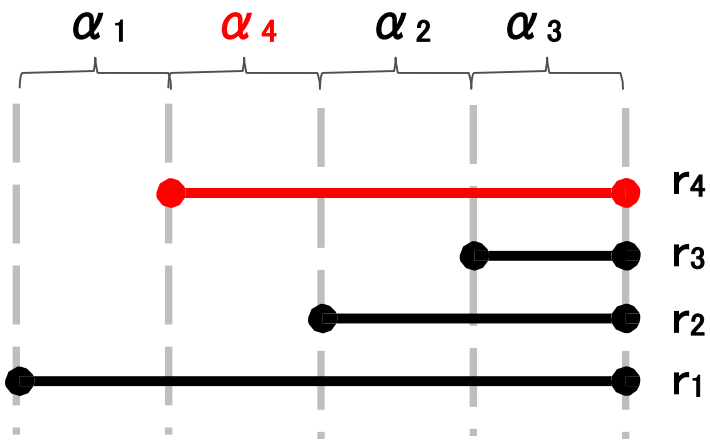


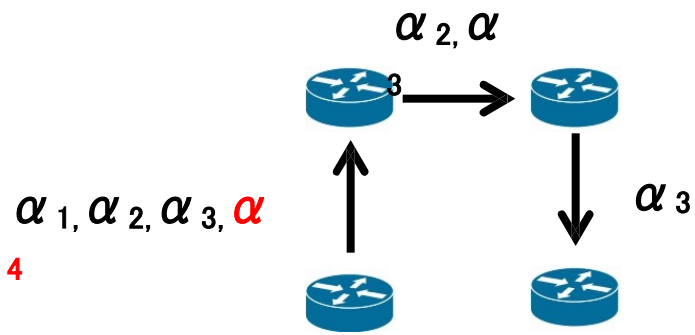




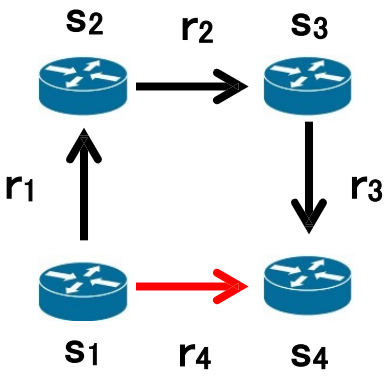
Delta-net [NSDI'17]

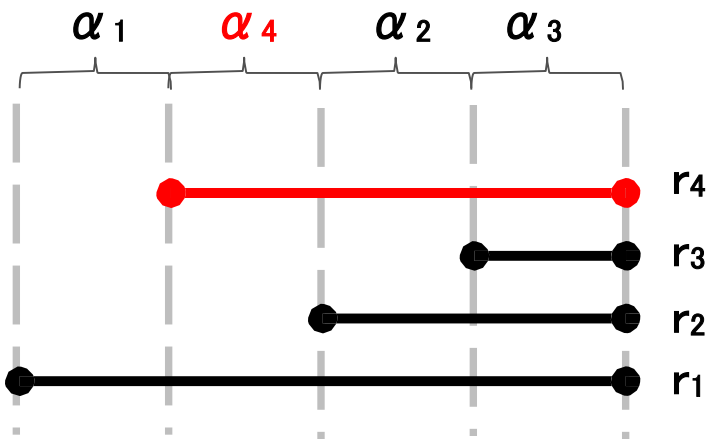


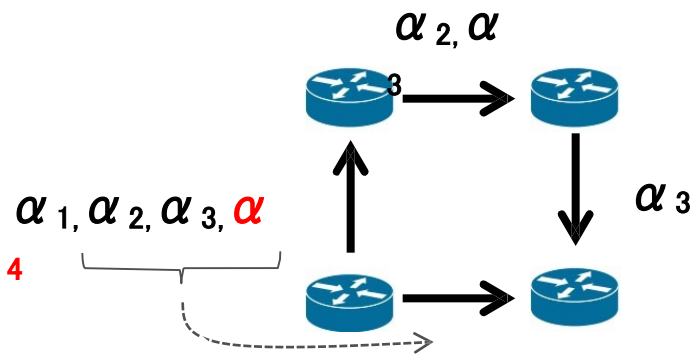




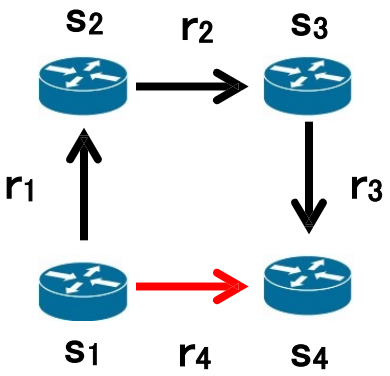
Delta-net [NSDI'17]

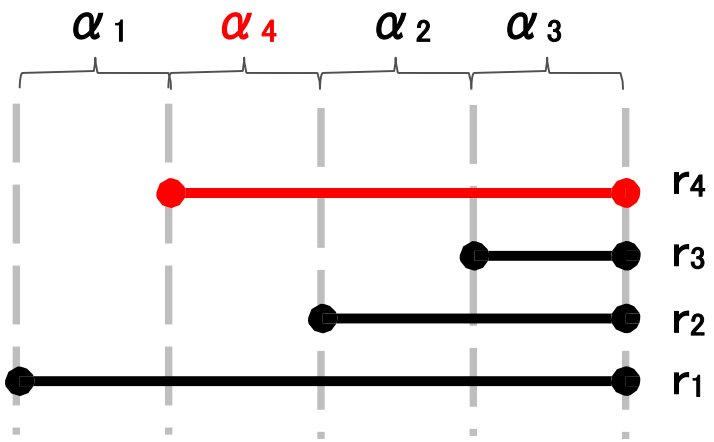


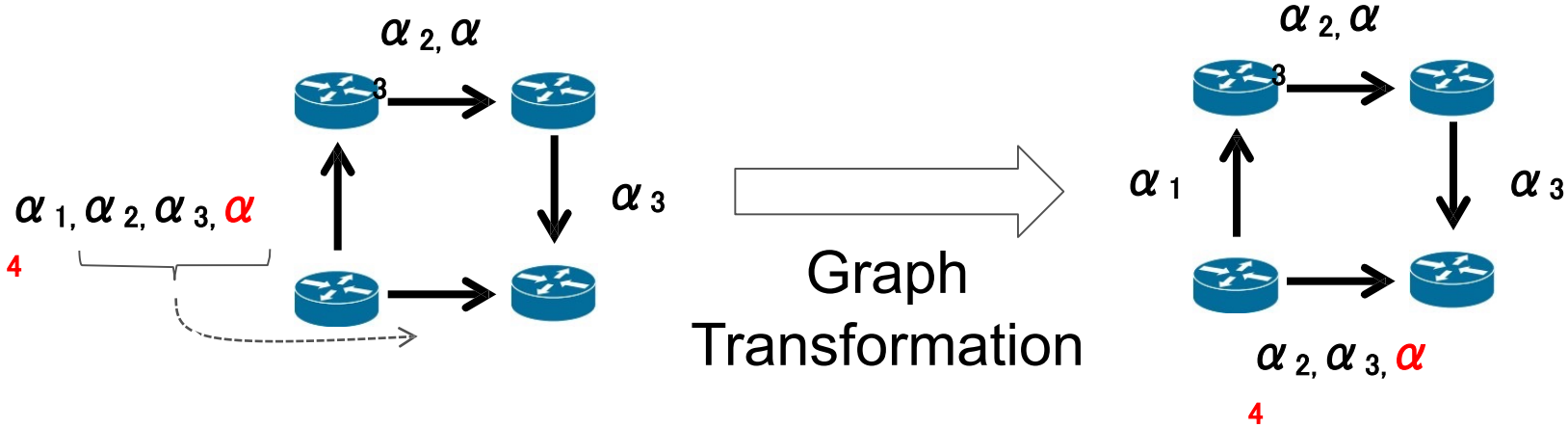




Delta-net [NSDI'17]

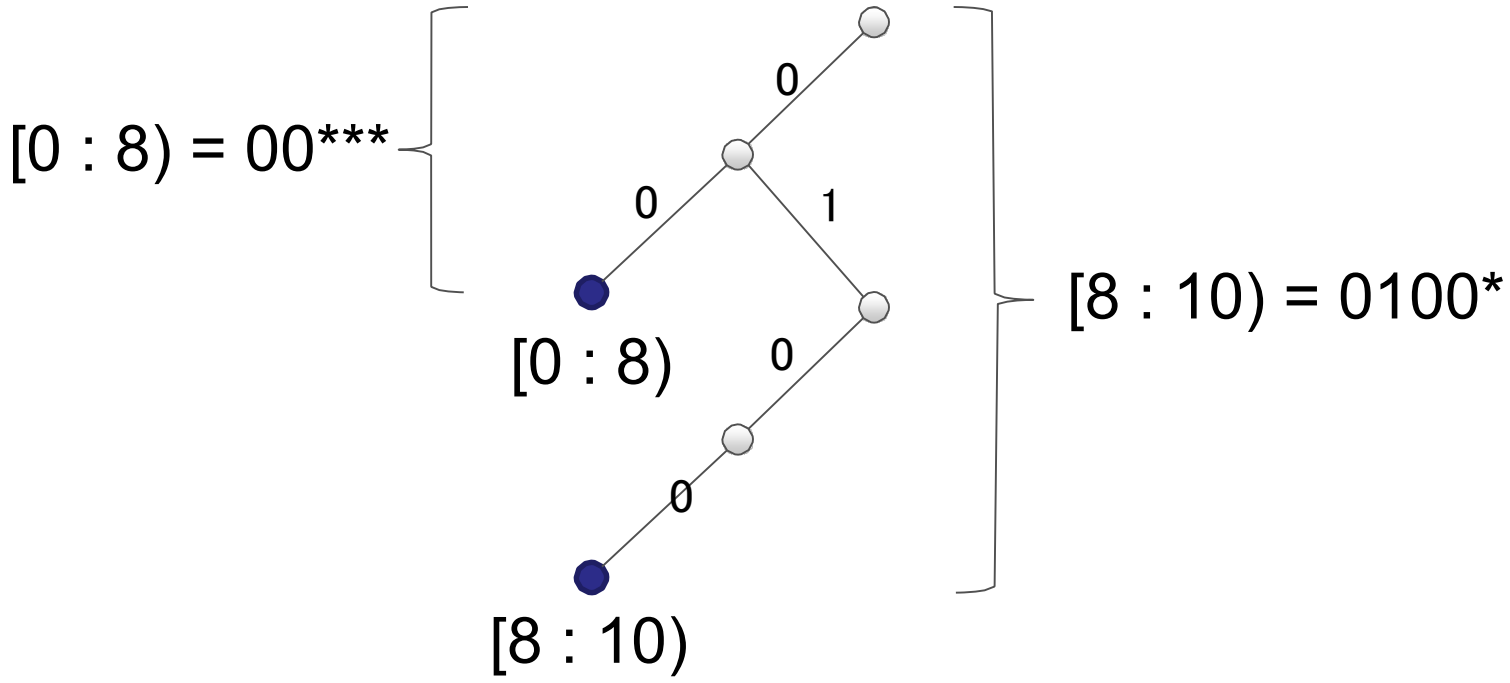






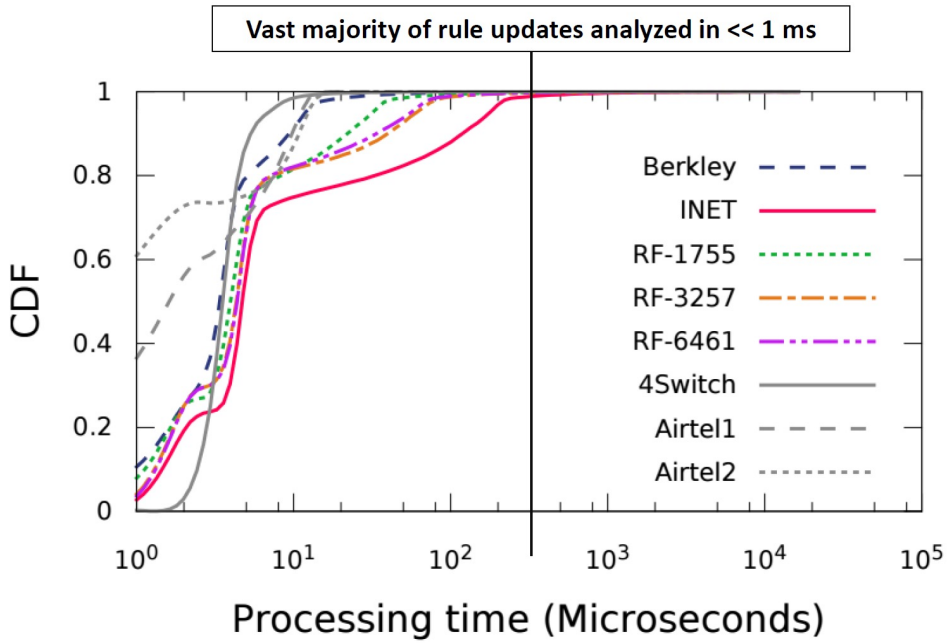
Delta-net [NSDI'17]

- Details: encode destination IP with ranges, binary search tree for rule insertion



Delta-net [NSDI'17]

- Advantages: faster than VeriFlow (1-2 orders)



- Limitations
  - Only considers a single dimension, i.e., destination IP
  - Only works for IP ranges, e.g., IP prefixes
  - The # of atoms is not minimum

Reference

[1] Kazemian et al., Header Space Analysis: Static Checking For Networks, in USENIX NSDI 2012

[2] A. Khurshid, W. Zhou, M. Caesar, and P. Godfrey. VeriFlow: Verifying network-wide invariants in real time. In USENIX NSDI, 2013
https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/khurshid

[3] A. Horn, A. Kheradmand, and M. R. Prasad. Delta-net: Real-time network verification using atoms. In USENIX NSDI, 2017