Software Defined Networking

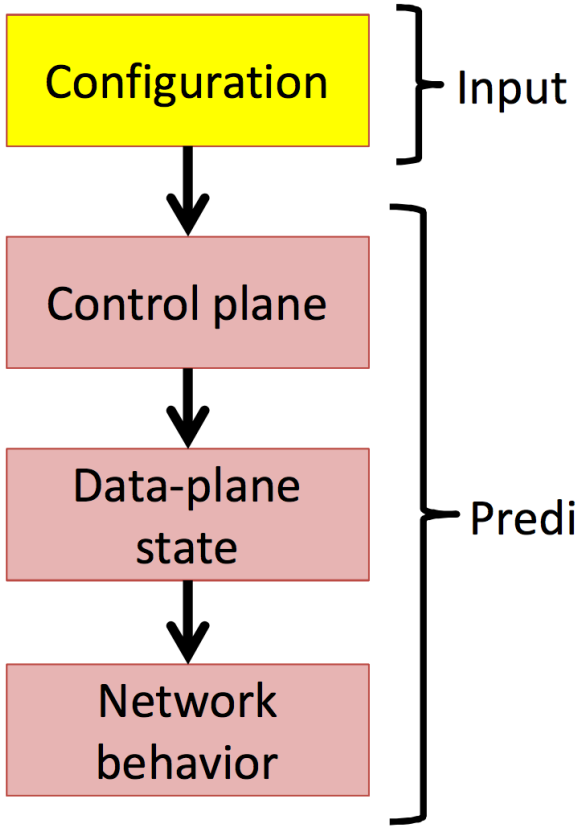
Dr. Nick Feamster
Professor

In this course, you will learn about software defined networking and how it is changing the way communications networks are managed, maintained, and secured.

This Module: Verification

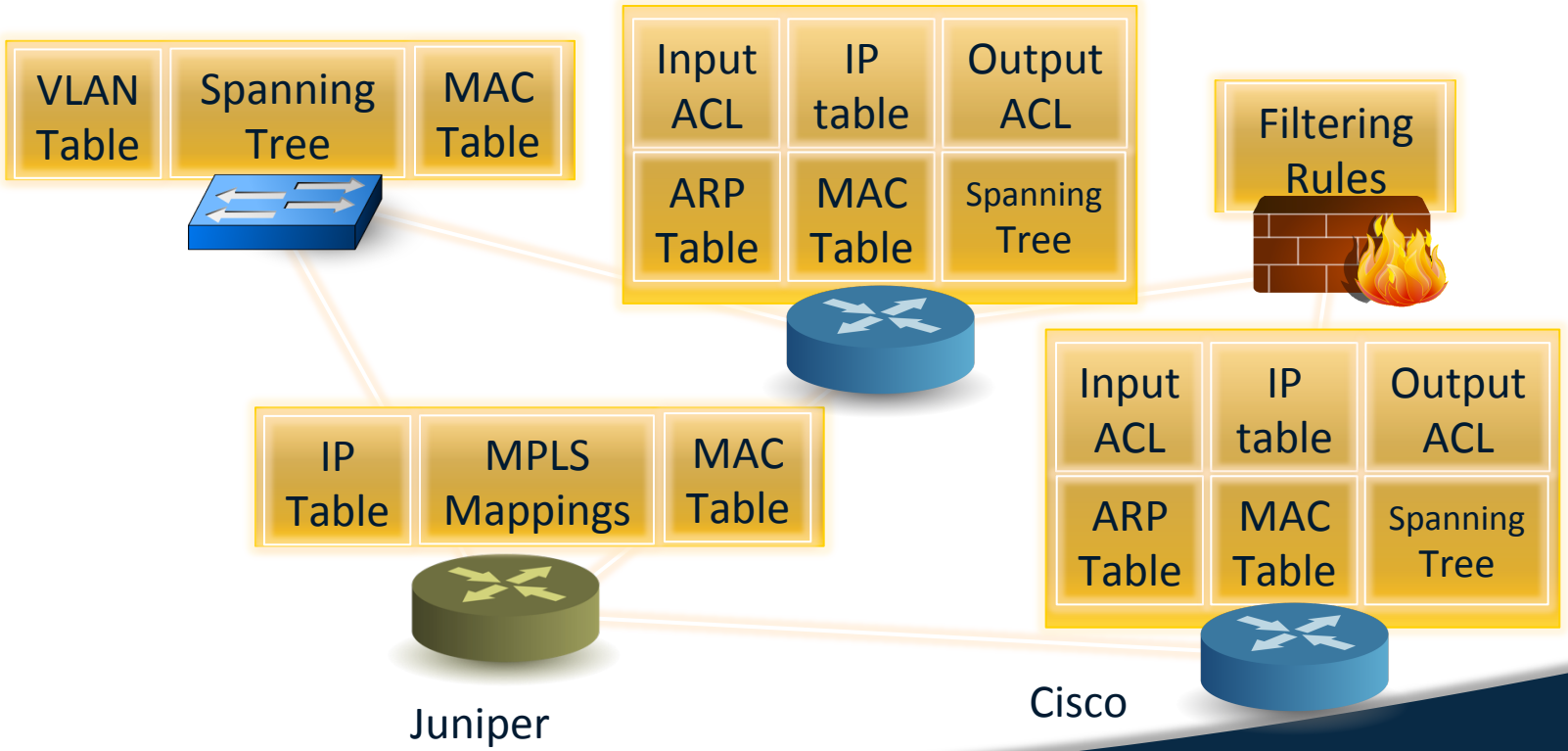
- **Motivation:** How do you know the network is doing the right thing?
- Verification techniques
  - Configuration Verification: rcc (pre-SDN)
  - Control Plane Verification: Kinetic
  - Data Plane Verification
    - Header Space Analysis
    - Veriflow

Limitations of Configuration Verification

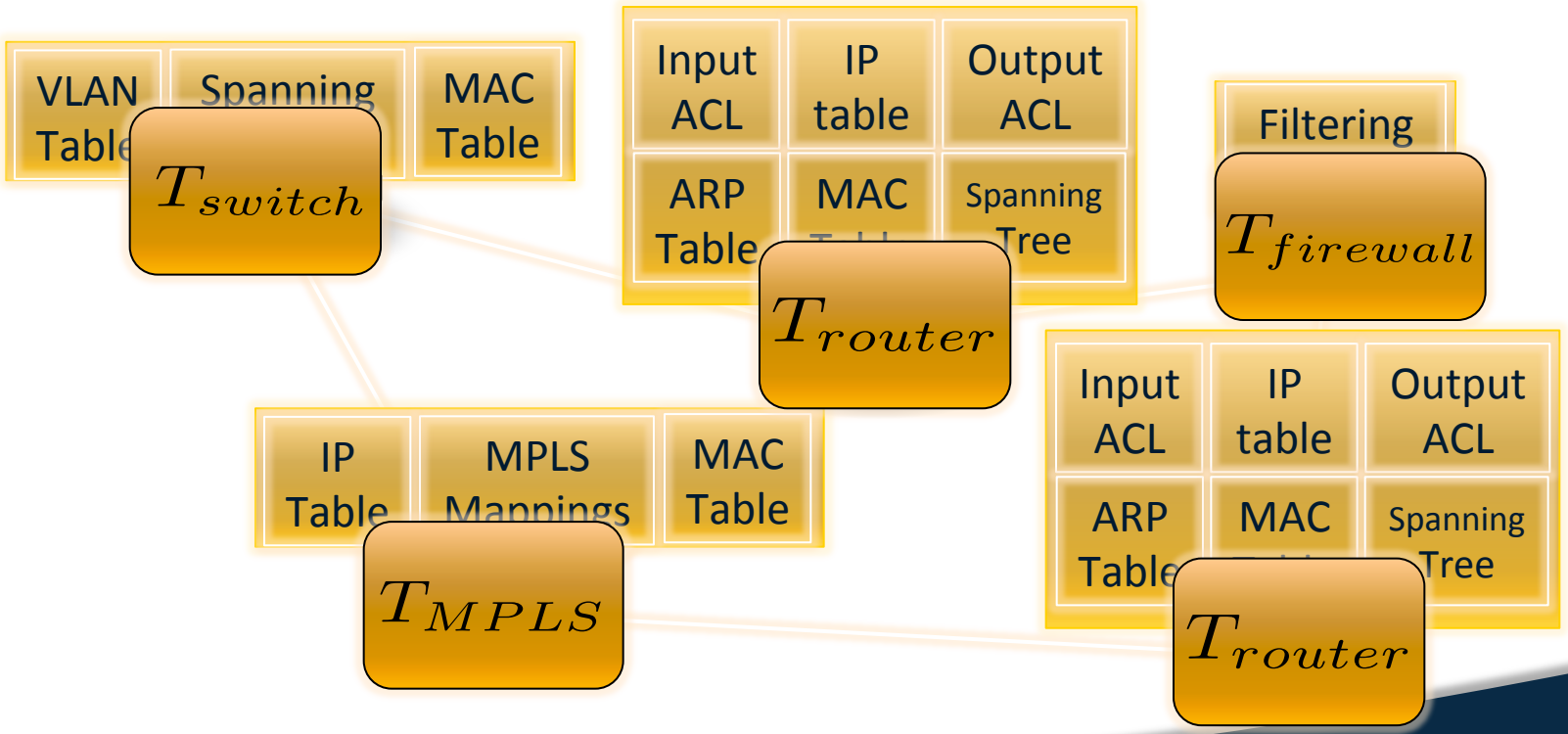


- Prediction is difficult
  - Various configuration languages
  - Dynamic distributed protocols
- Prediction misses implementation bugs in control plane

Network Verification Vision

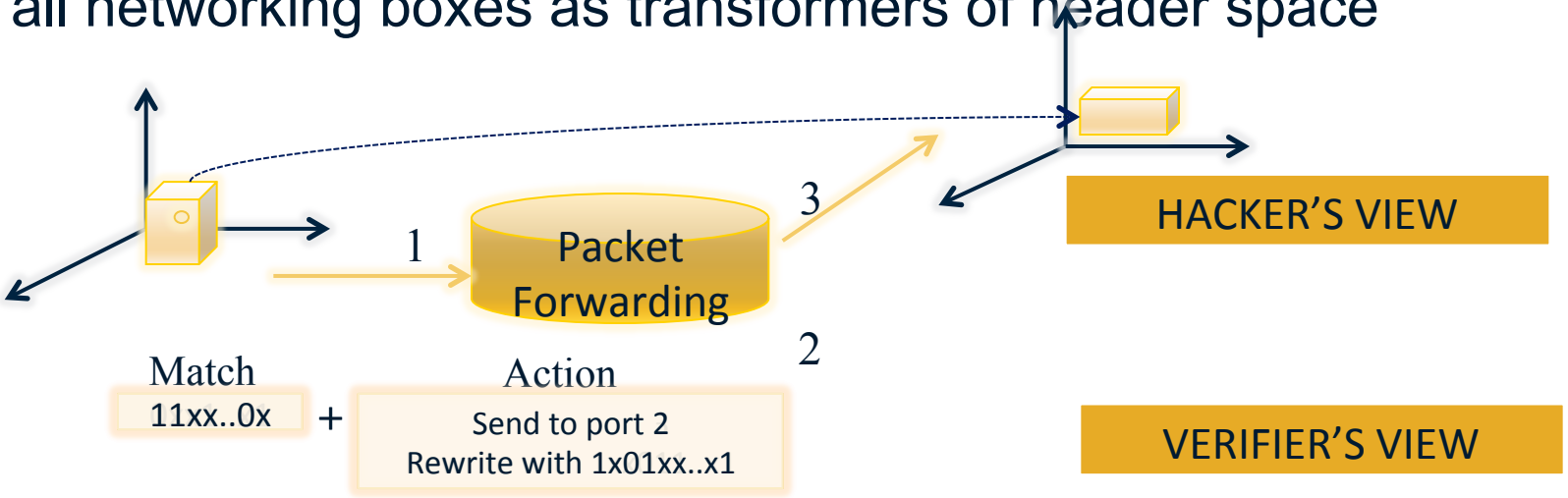


Network Verification Vision



Insight: Treat Network as a Program

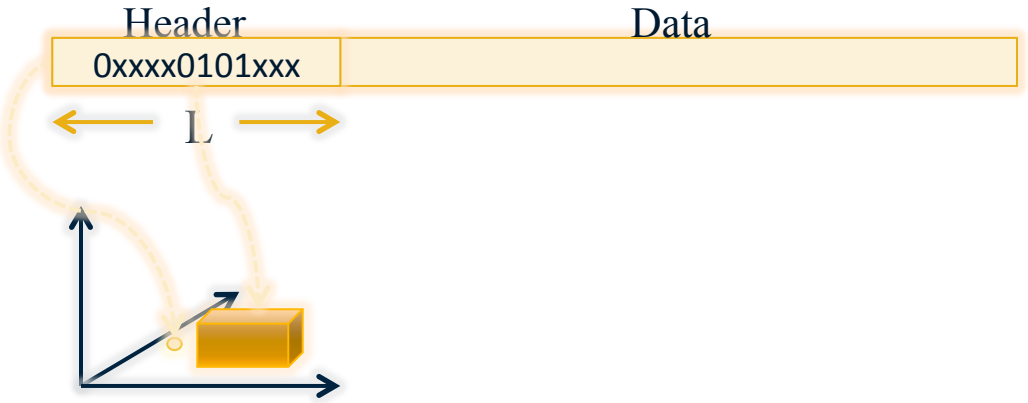
- Model header as point in high dimensional space and all networking boxes as transformers of header space



ROUTER ABSTRACTED AS SET OF GUARDED COMMANDS . .
NETWORK BECOMES A PROGRAM →**CAN USE PL TOOLS**

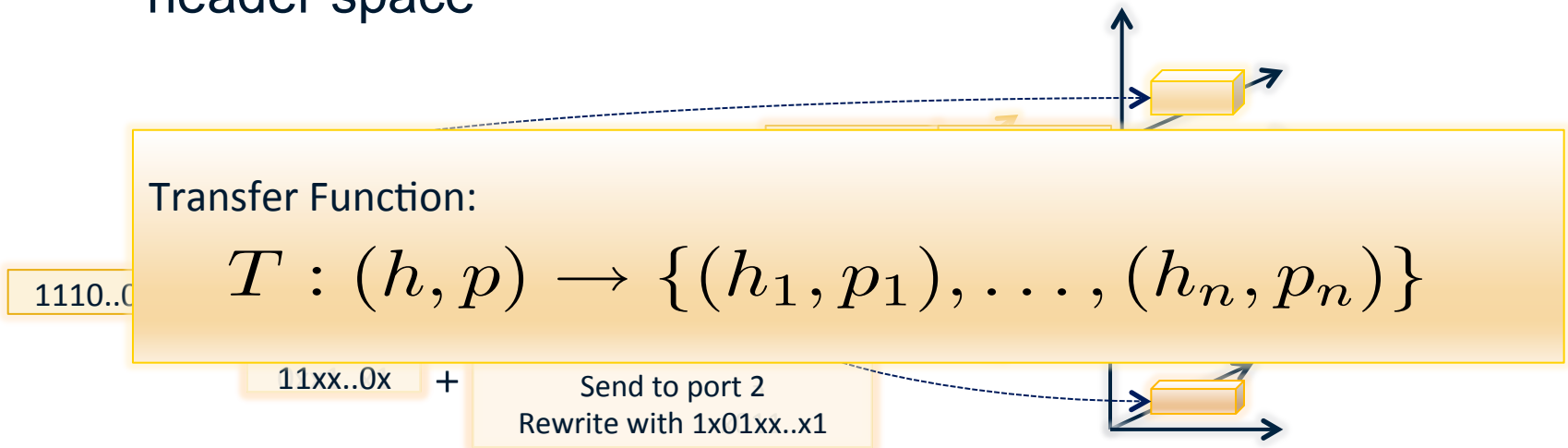
Header Space Framework

- Step 1 - Model a packet, based on its header bits, as a point in $\{0,1\}^L$ space – The Header Space



Header Space Framework

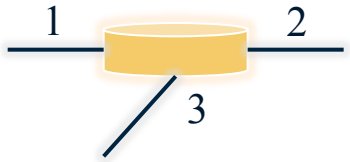
- Step 2 – Model all networking boxes as transformers of header space



Transfer Function Example

- IPv4 Router – Forwarding Behavior

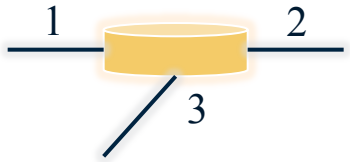
  - 172.24.74.x Port1
  - 172.24.128.x Port2
  - 171.67.x.x Port3



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

Transfer Function Example

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



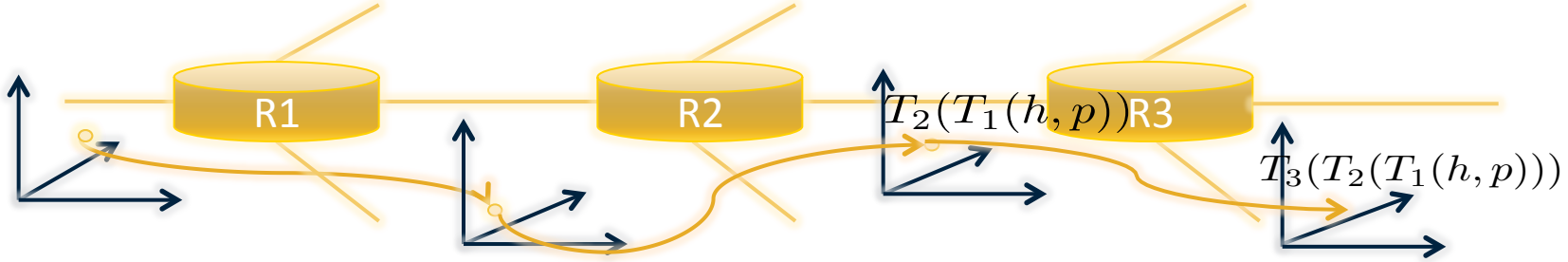
$$T(h, p) = \begin{cases} (\text{rw\_mac}(\text{dec\_ttl}(h), \text{next\_mac}), 1) & \text{if dst\_ip}(h) = 172.24.74.x \\ (\text{rw\_mac}(\text{dec\_ttl}(h), \text{next\_mac}), 2) & \text{if dst\_ip}(h) = 172.24.128.x \\ (\text{rw\_mac}(\text{dec\_ttl}(h), \text{next\_mac}), 3) & \text{if dst\_ip}(h) = 171.67.x.x \end{cases}$$

Example Actions

- Rewrite: rewrite bits 0-2 with value 101
  - (h & 000111…) | 101000…
- Encapsulation: encap packet in a 1010 header.
  - (h >> 4) | 1010….
- Decapsulation: decap 1010xxx… packets
  - (h << 4) | 000…xxxx
- TTL Decrement:
  - if ttl(h) == 0: Drop
  - if ttl(h) > 0: h – 0…000000010…0
- Load Balancing:
  - LB(h,p) = {(h,P_1),…(h,P_n)}

Composing Transfer Functions

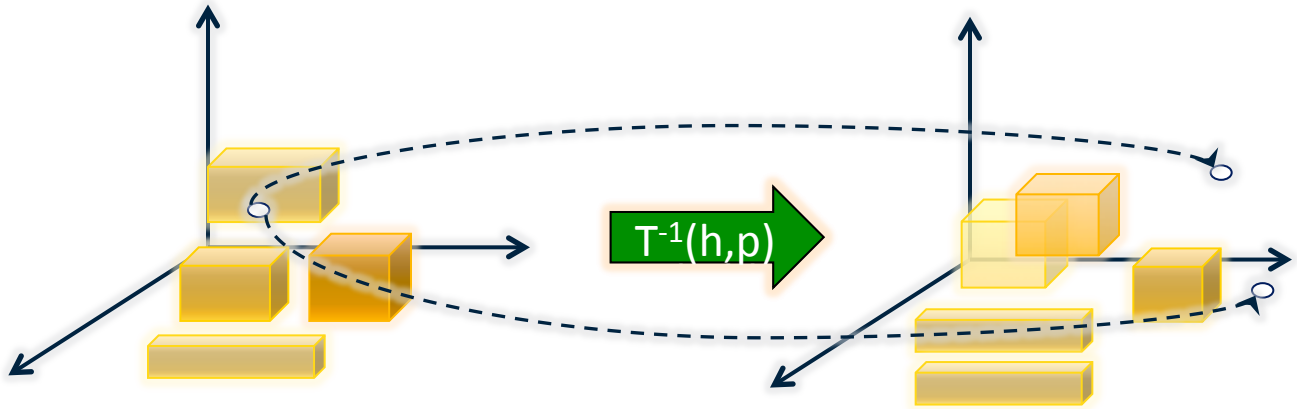
- We can determine end to end behavior by composing transfer functions,



$$T_3(T_2(T_1(h,p)))$$

Inverting Transfer Functions

- Tell us all possible input packets that can generate an output packet.



Header Space Framework

- Step 3: Header Space Set Algebra
  - Intersection
  - Complementation
  - Difference
  - Check subset and equality condition.
- Every region of Header Space, can be described by union of Wildcard Expressions. (example: 10xx U 011x)
- **Goal**: do set operation on wildcard expressions.

HS Set Algebra: Intersection

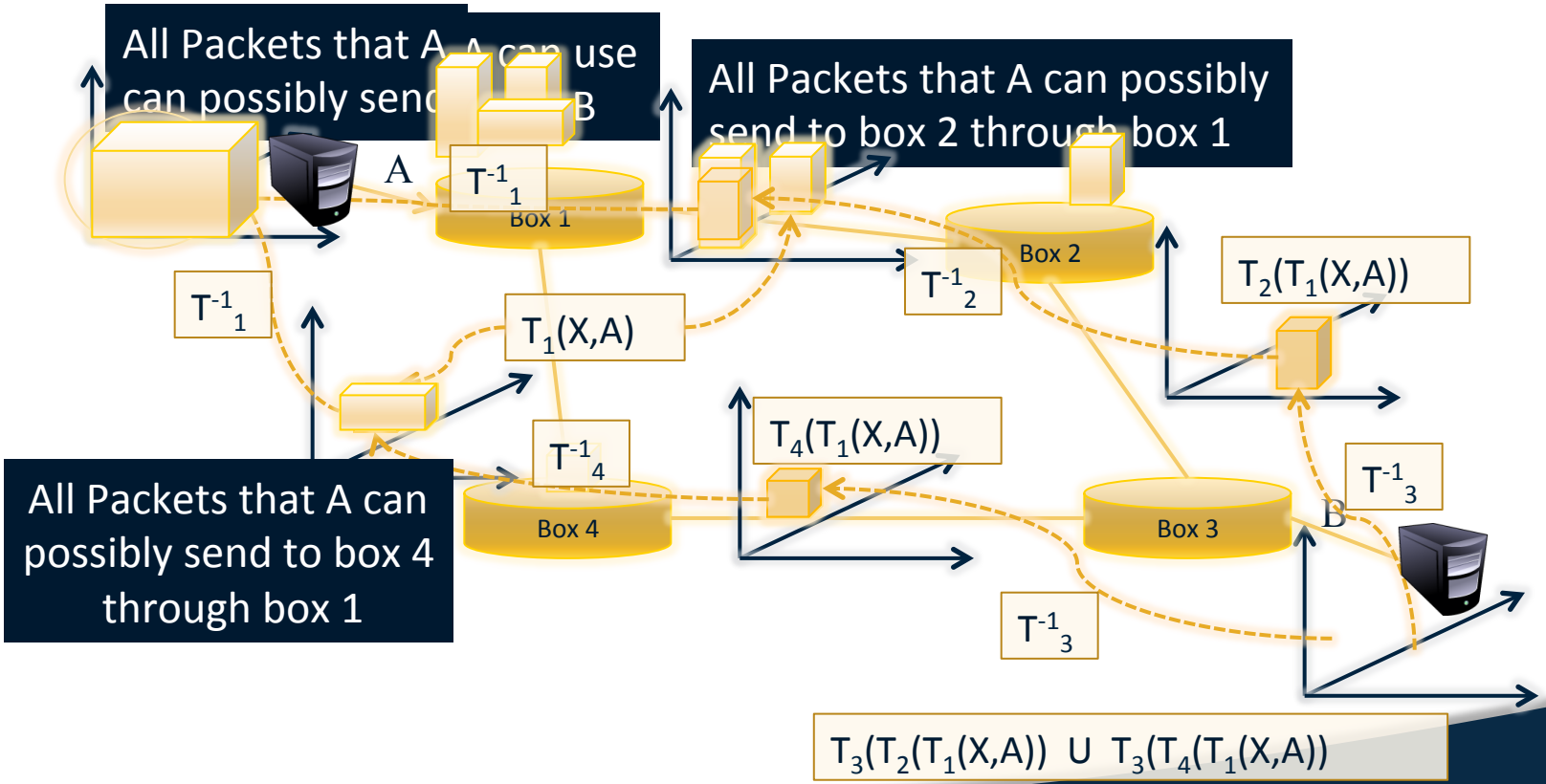
- Bit by bit intersect using intersection table:
  - Example: $10xx \cap 1xx0 = 10x0$
  - If result has any ‘z’, then intersection is empty:
  - Example: $10xx \cap 0xx0 = z0x0 = \phi$

b_i \ b_j	0	1	x
0	0	z	0
1	z	1	1
x	0	1	x

Header Space Framework

- Simple abstraction that gives us:
  - Common model for all packets
    - Header Space.
  - Common model for forwarding functionality of all networking boxes.
    - Transfer Function.
  - Mathematical foundation to check end-to-end properties about networks.
    - $T(h,p)$ and $T^{-1}(h,p)$.
    - Set operations on Header Space.

Finding Reachability

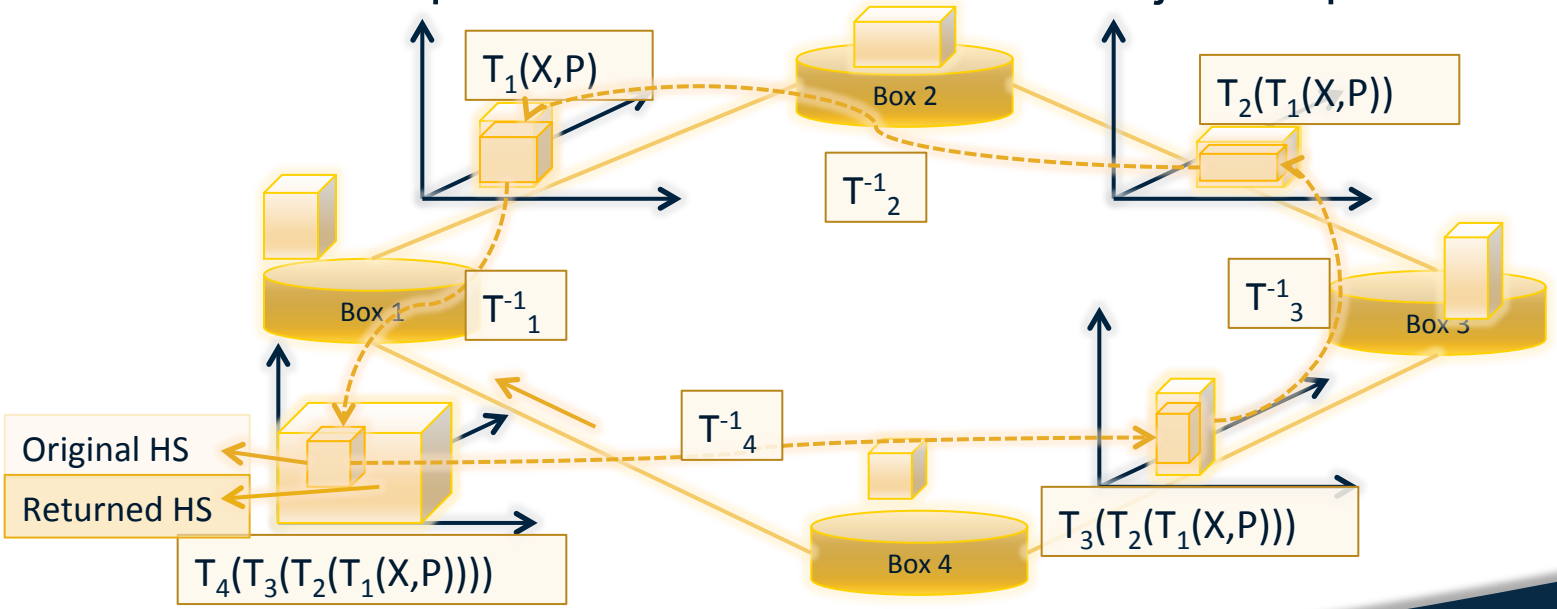


Predicates on Paths: Policies

- Can generalize to check *path predicates:*
  - Blackhole freedom (A → B and notice unexpected drop)
  - Communication via middle box. (A→B packets must pass through C)
  - Maximum hop count (length of path from A → B never exceeds L)
  - Isolation of paths (http and https traffic from A→B don't share the same path)

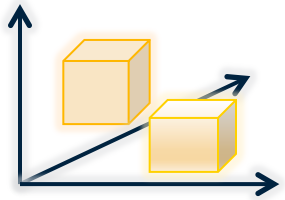
Finding Loops

- Is there a loop in the network?
  - Inject an all-x test packet from every switch-port
  - Follow the packet until it comes back to injection port

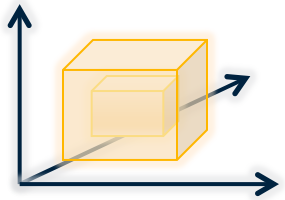


Finding Loops

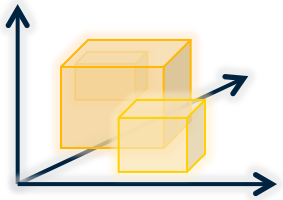
- Is the loop infinite?



Finite Loop



Infinite Loop



?

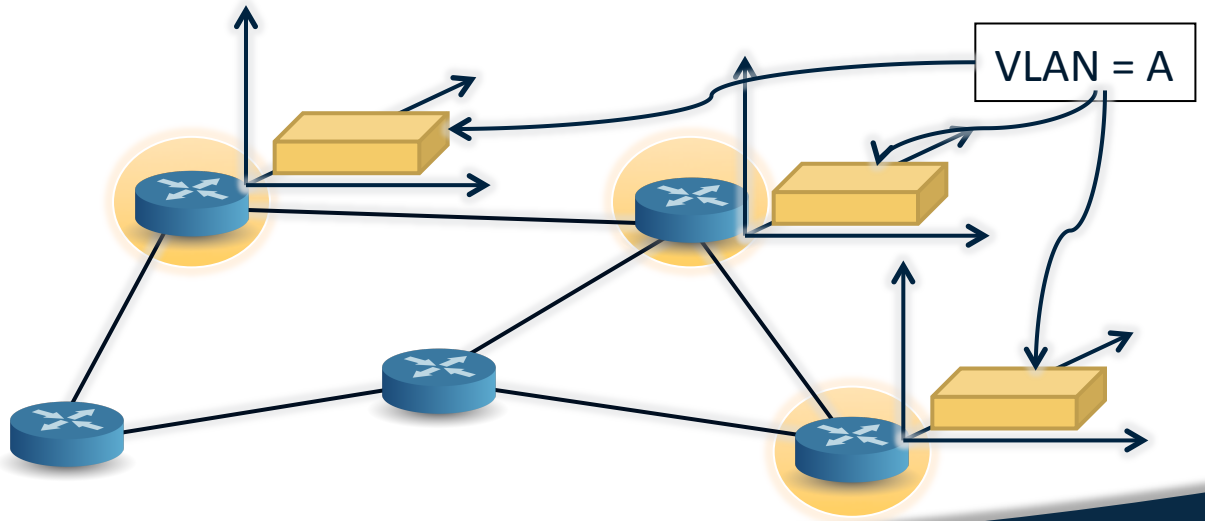
Network Slices

- By slicing network we can share network resources. (e.g. Bank of America and Citi share the same infrastructure in a financial center).
- Like VM, we need to ensure no interaction between slices. (security, independence of slices).

→ We need to check isolation of slices.

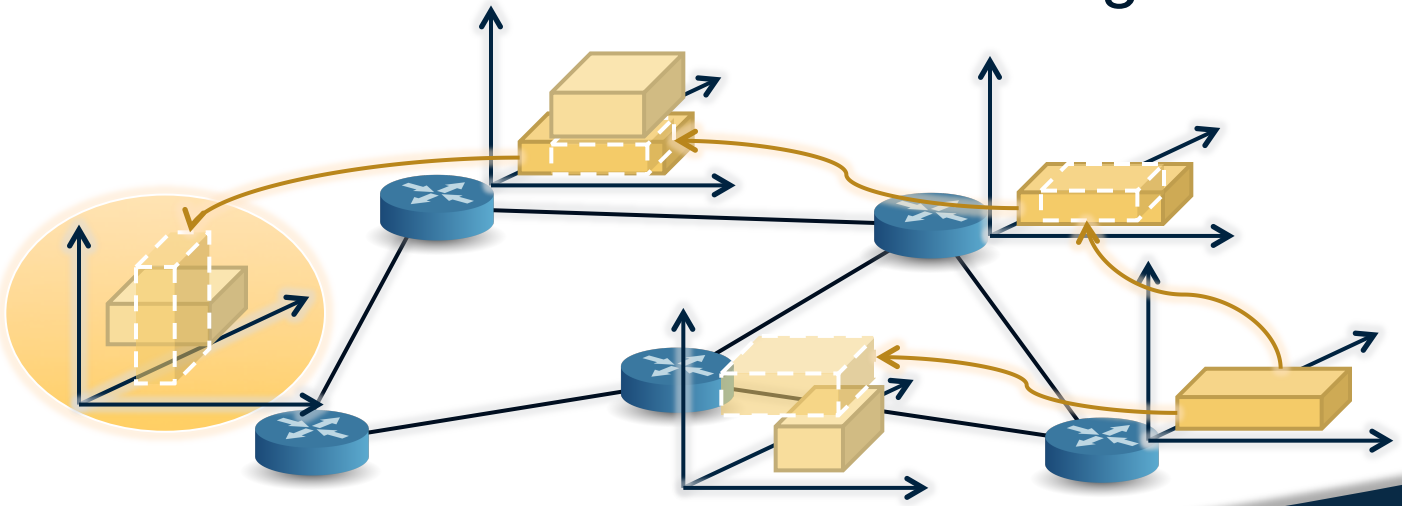
Definition of Slice in HSA

- Network slice is a piece of network resources defined by
  - A topology consisting of switches and ports.
  - A set of predicates on packet headers.

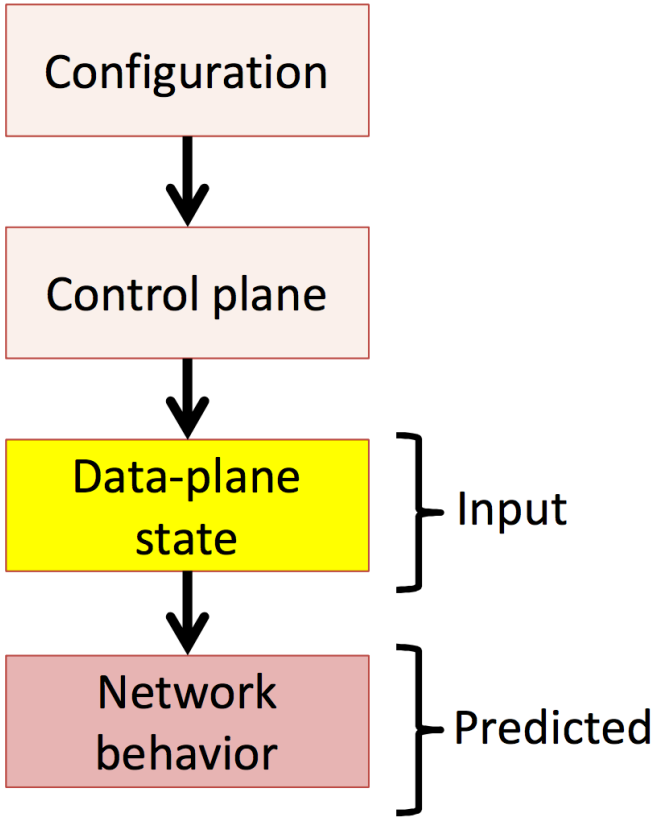


Checking Isolation of Slices

- How to check if two slices are isolated?
  - Slice definitions don't intersect.
  - Packets don't leak after forwarding.



Veriflow: Data-Plane Verification



- Less prediction
- Closer to actual network behavior
- Unified analysis for multiple control-plane protocols
- Can catch control-plane implementation bugs

Challenges with Real-Time Verification

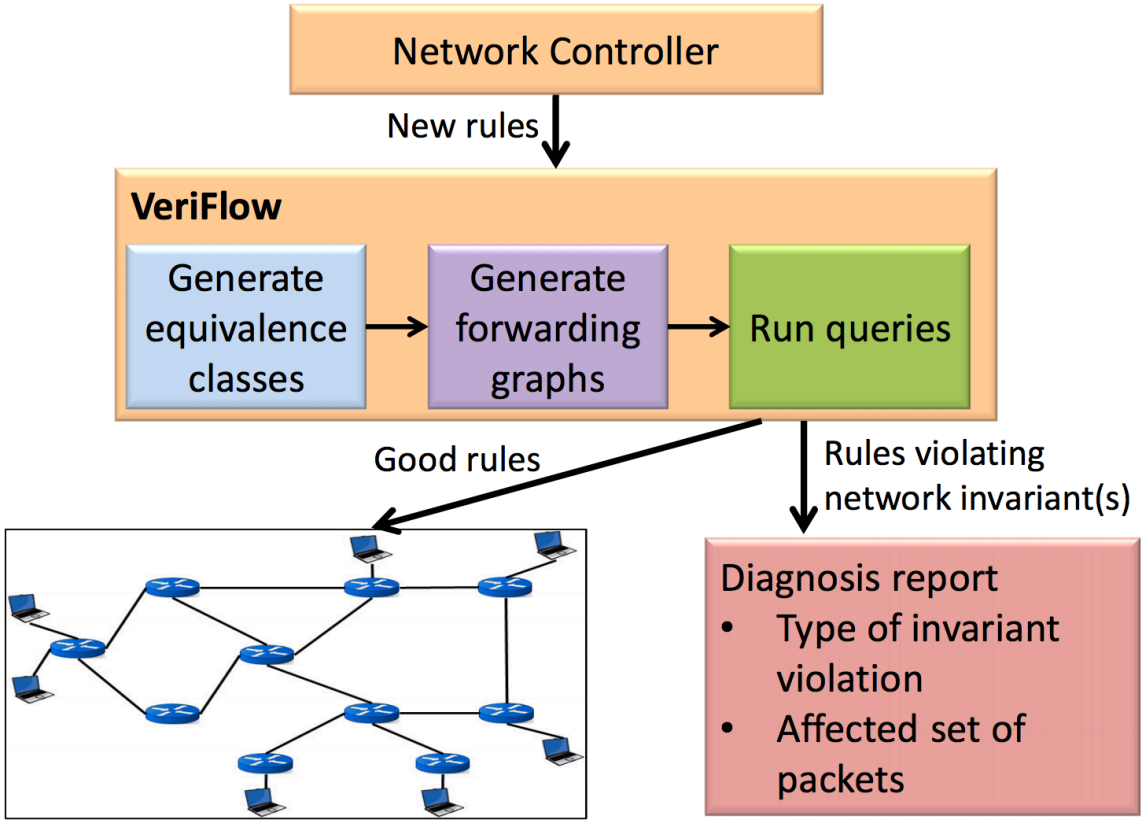
- Challenge 1: Obtaining real-time view of network
  - Solution: Utilize the **centralized** data-plane view available in an **SDN (Software-Defined Network)**
- Challenge 2: Verification speed
  - Solution: Off-the-shelf techniques?

No, too slow!

Veriflow: Check Data-Plane State in Real-time

- VeriFlow checks network-wide invariants in **real time** using data-plane state
  - Absence of routing loops and black holes, access control violations, etc.
- VeriFlow functions by
  - Monitoring **dynamic changes** in the network
  - Constructing a **model** of the **network behavior**
  - Using **custom algorithms** to automatically derive whether the network contains errors

VeriFlow Operation



Summary

- Can perform data-plane verification by performing symbolic execution on packets
  - Can tell you if data plane state is correct
- Limitations
  - Scaling can be challenging
  - The errors themselves may be in configuration or control logic
  - Dynamics are not handled