Header Space Analysis: Static Checking For Networks

Peyman Kazemian (Stanford University)
George Varghese (UCSD, Yahoo Labs)
Nick McKeown (Stanford University)

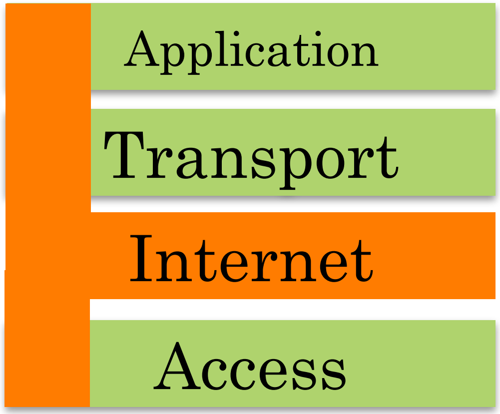
November 7th, 2012
IRTF

Motivation

- It is hard to understand and reason about end-to-end behavior of networks:
  - Can host A talk to host B?
  - What are all the packet headers from A that can reach B?
  - Are there any loops or black holes in the network?
  - Is Slice X isolated totally from Slice Y?
  - What will happen if I remove an entry from a router?

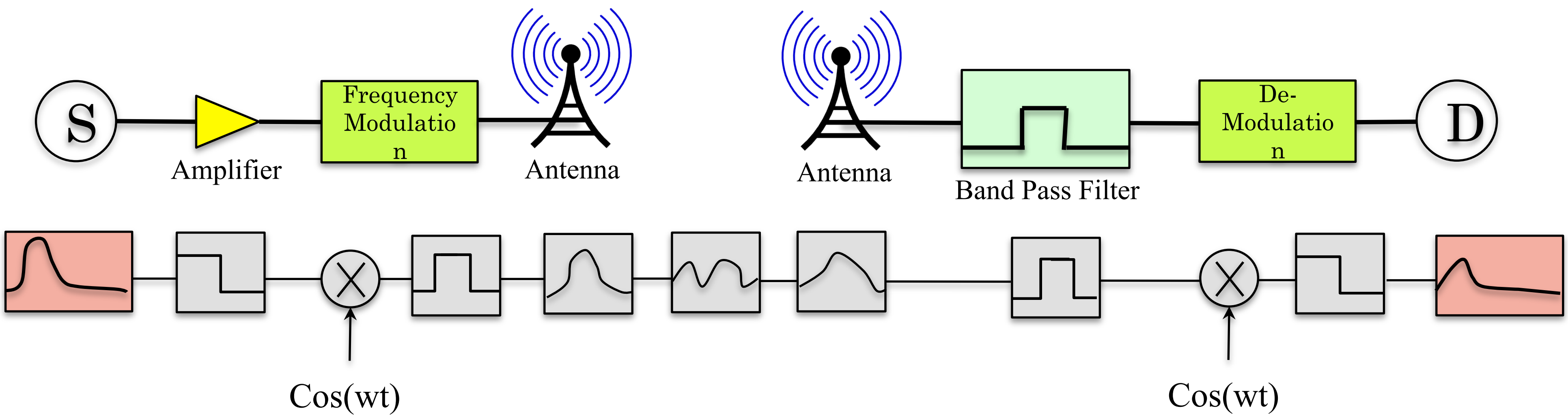
MOTIVATION

- There are two reason for this complexity:
  - Networks are getting larger.
  - Network functionality becoming more complex.
    - Firewalls, ACLs and deep packet inspection MBs.
    - VLAN and inter-VLAN routing.
    - Encapsulation (MPLS, GRE).
    - ToS-based routing.
    - nondeterministic routing.



Looking at the other fields

Communication Systems:



Header Space Analysis

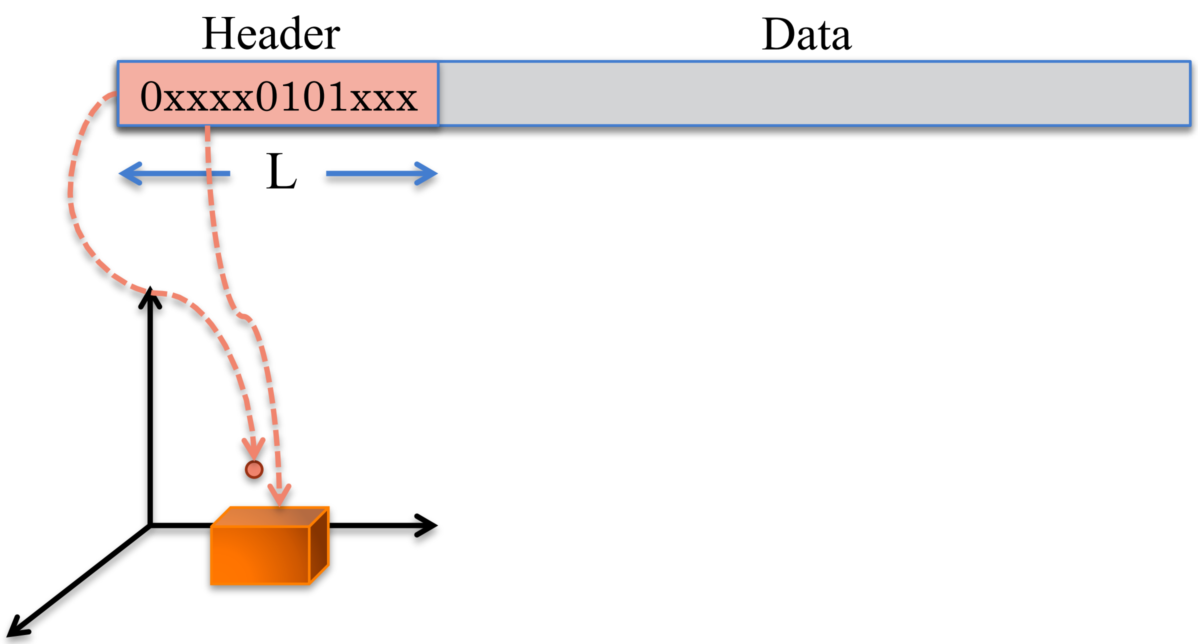
A simple abstraction to model all kinds of forwarding functionalities regardless of specific protocols and implementations.

Header Space Framework

Simple Observation: a packet is a point in the space of possible headers and a box is a transformer on that space.

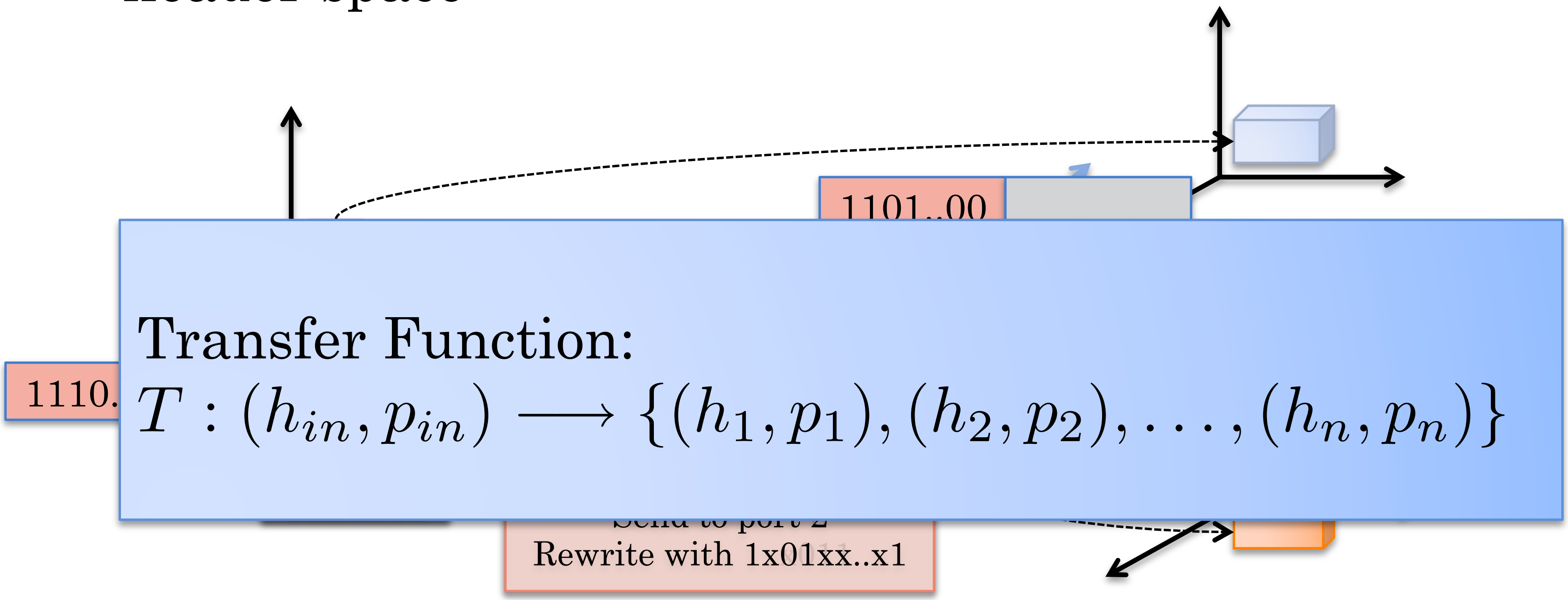
Header Space Framework

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



HEADER SPACE FRAMEWORK

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

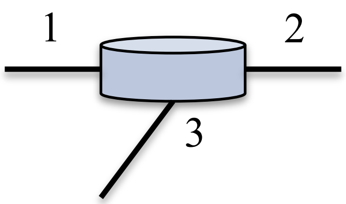


Transfer Function:

$$T : (h_{in}, p_{in}) \longrightarrow \{(h_1, p_1), (h_2, p_2), \ldots, (h_n, p_n)\}$$

Header Space Framework

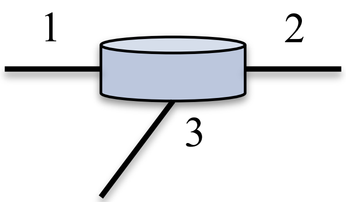
- Example: Transfer Function of an IPv4 Router
  - 172.24.74.0 255.255.255.0 Port1
  - 172.24.128.0 255.255.255.0 Port2
  - 171.67.0.0 255.255.0.0 Port3



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

Header Space Framework

- Example: Transfer Function of an IPv4 Router
  - 172.24.74.0 255.255.255.0 Port1
  - 172.24.128.0 255.255.255.0 Port2
  - 171.67.0.0 255.255.0.0 Port3

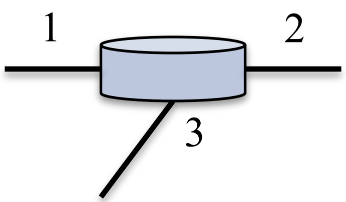


$$T(h, p) = \begin{cases} (\text{dec\_ttl}(h),1) & \text{if dst\_ip}(h) = 172.24.74.x \\ (\text{dec\_ttl}(h),2) & \text{if dst\_ip}(h) = 172.24.128.x \\ (\text{dec\_ttl}(h),3) & \text{if dst\_ip}(h) = 171.67.x.x \end{cases}$$

Header Space Framework

- Example: Transfer Function of an IPv4 Router

  - 172.24.74.0 255.255.255.0 Port1
  - 172.24.128.0 255.255.255.0 Port2
  - 171.67.0.0 255.255.0.0 Port3



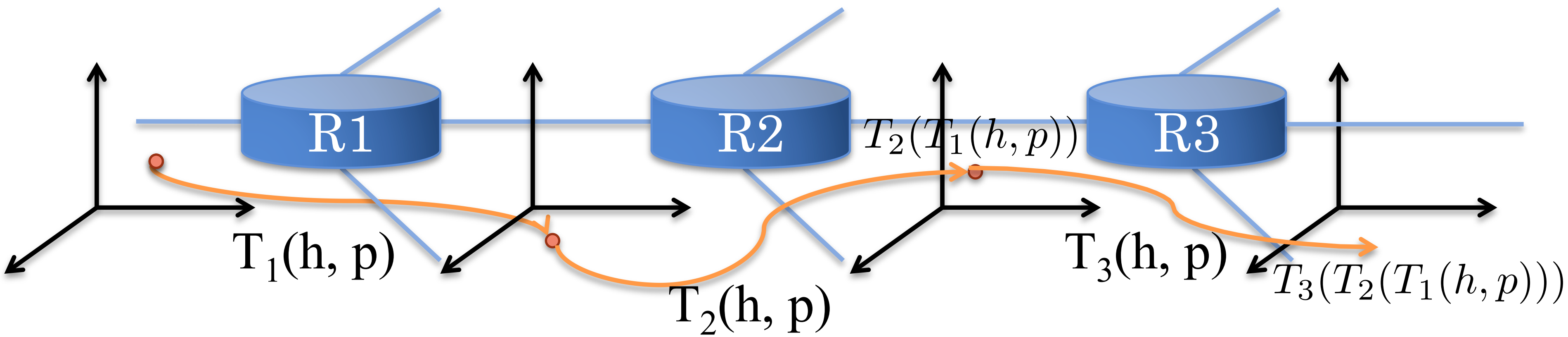
$$T(h, p) = \begin{cases} (rw\_mac(dec\_ttl(h), next\_mac), 1) & \text{if } dst\_ip(h) = 172.24.74.x \\ (rw\_mac(dec\_ttl(h), next\_mac), 2) & \text{if } dst\_ip(h) = 172.24.128.x \\ (rw\_mac(dec\_ttl(h), next\_mac), 3) & \text{if } dst\_ip(h) = 171.67.x.x \end{cases}$$

Example Rules:

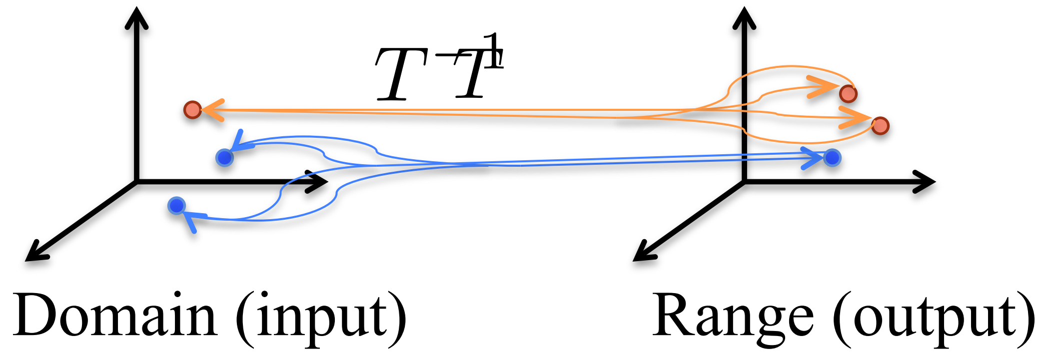
- FWD & RW: rewrite bits 0-2 with value 101
  - (h & 000111…) | 101000…
- Encapsulation: encap packet in a 1010 header.
  - (h >> 4) | 1010….
- Decapsulation: decap 1010xxx… packets
  - (h << 4) | 000…xxxx
- Load Balancing:
  - $LB(h,p) = \{(h,P_1),\ldots(h,P_n)\}$

Header Space Framework

- Properties of transfer functions
  - Composable: $T_3(T_2(T_1(h, p)))$



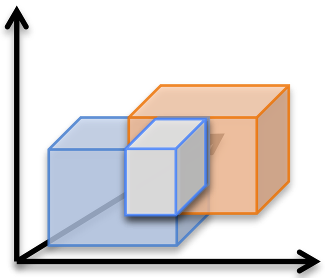
  - Invertible:



Header Space Framework

- Step 3 - Develop an algebra to work on these spaces.
- Every object in Header Space, can be described by union of Wildcard Expressions.

- We want to perform the following set operations on wildcard expressions:
  - Intersection
  - Complementation
  - Difference



Header Space Framework

- Finding Intersection:
  - Bit by bit intersect using intersection table:
    - Example: $10xx \cap 1xx0 = 10x0$
    - If result has any 'z', then intersection is empty:
    - Example: $10xx \cap 0xx1 = z0x1 = \phi$

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

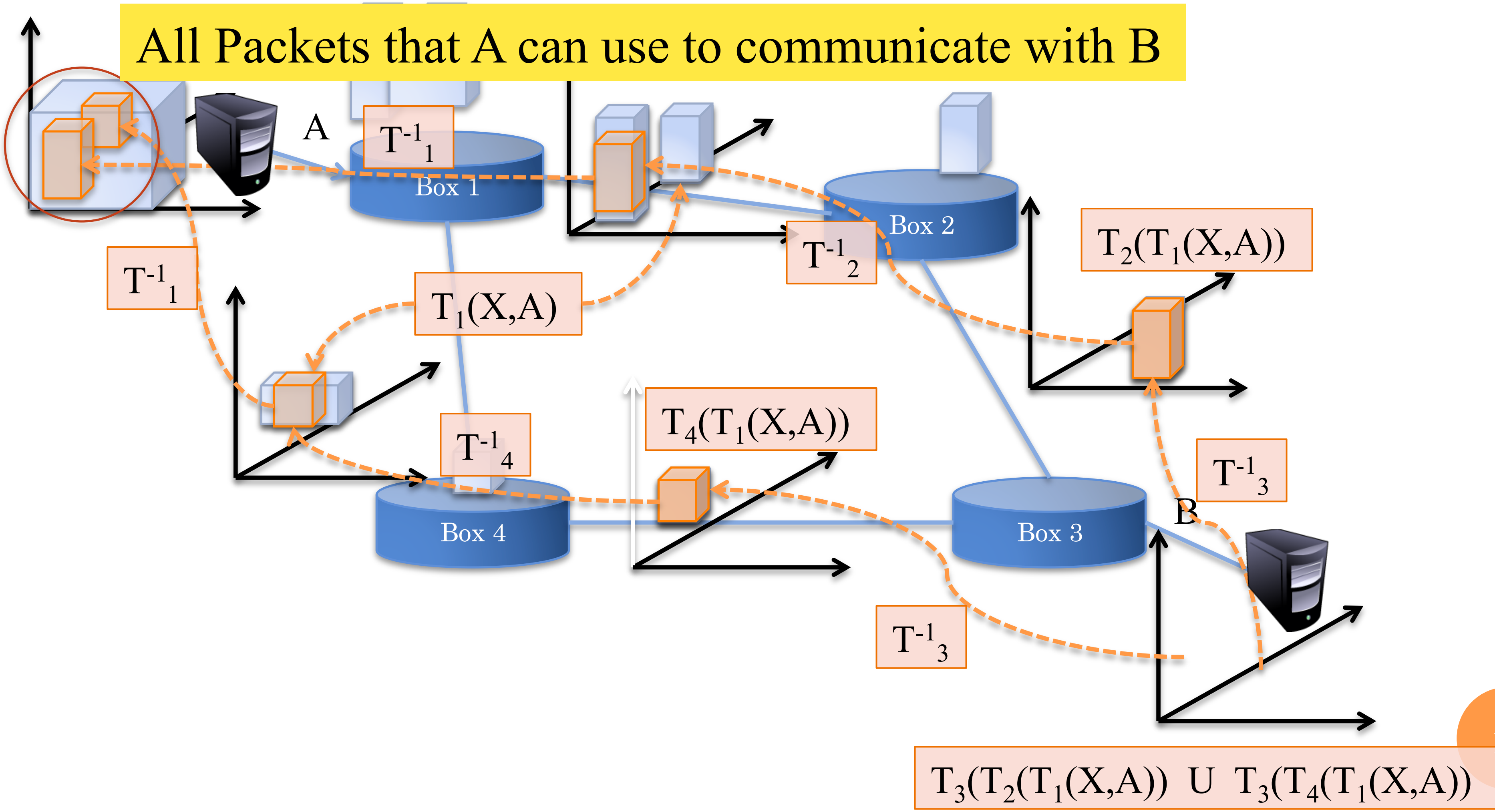
- See the paper for how to find complement and difference.

Framework

WE DEVELOPED SO FAR

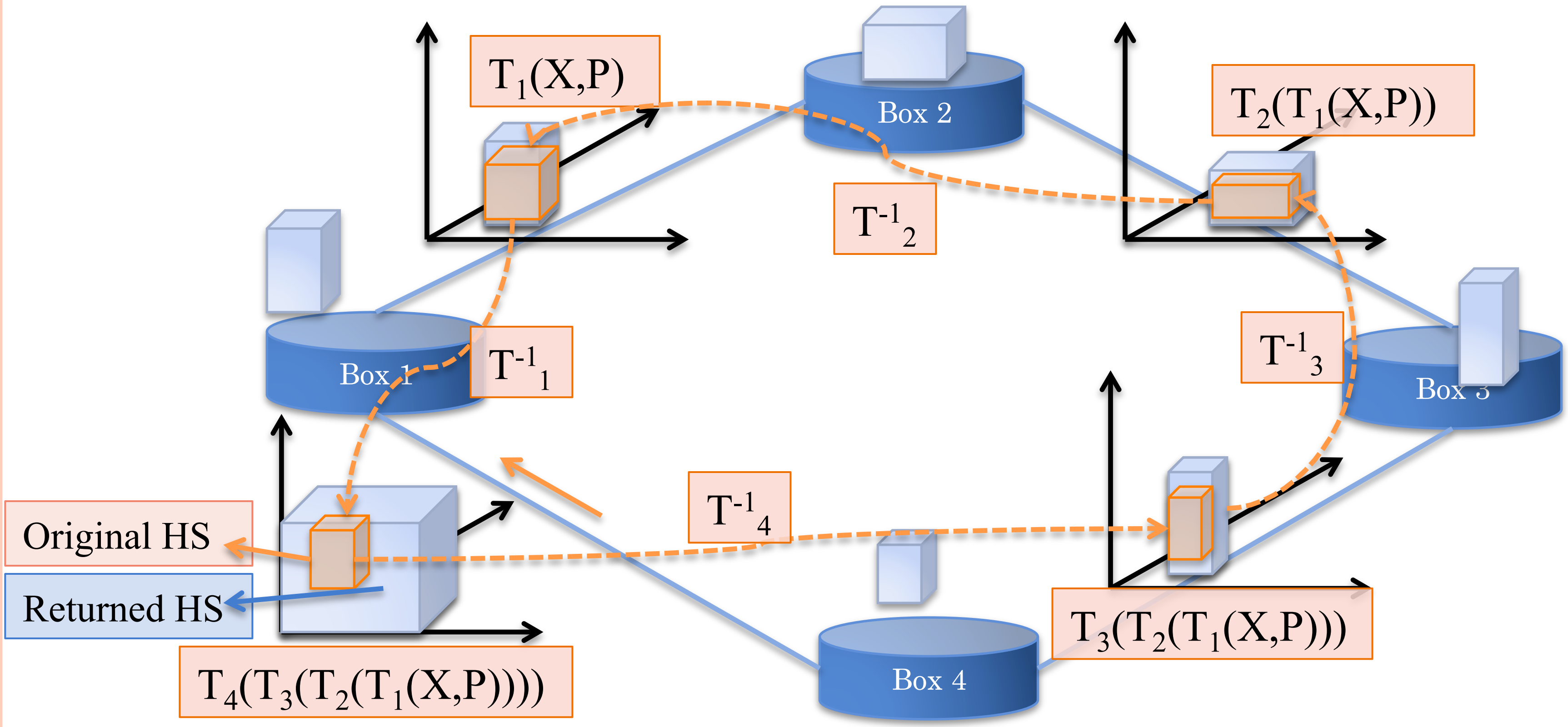
Use Cases

- Can host A talk to B?



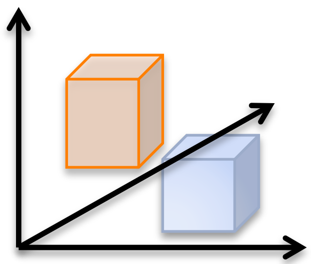
Use Cases

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

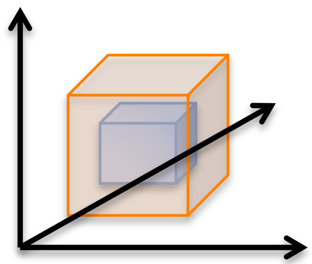


Use Cases

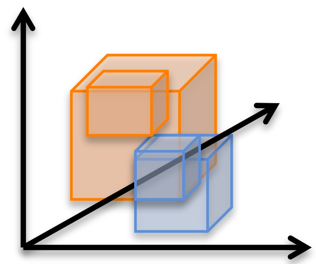
- Is the loop infinite?



Finite Loop



Infinite Loop



?

Use Cases

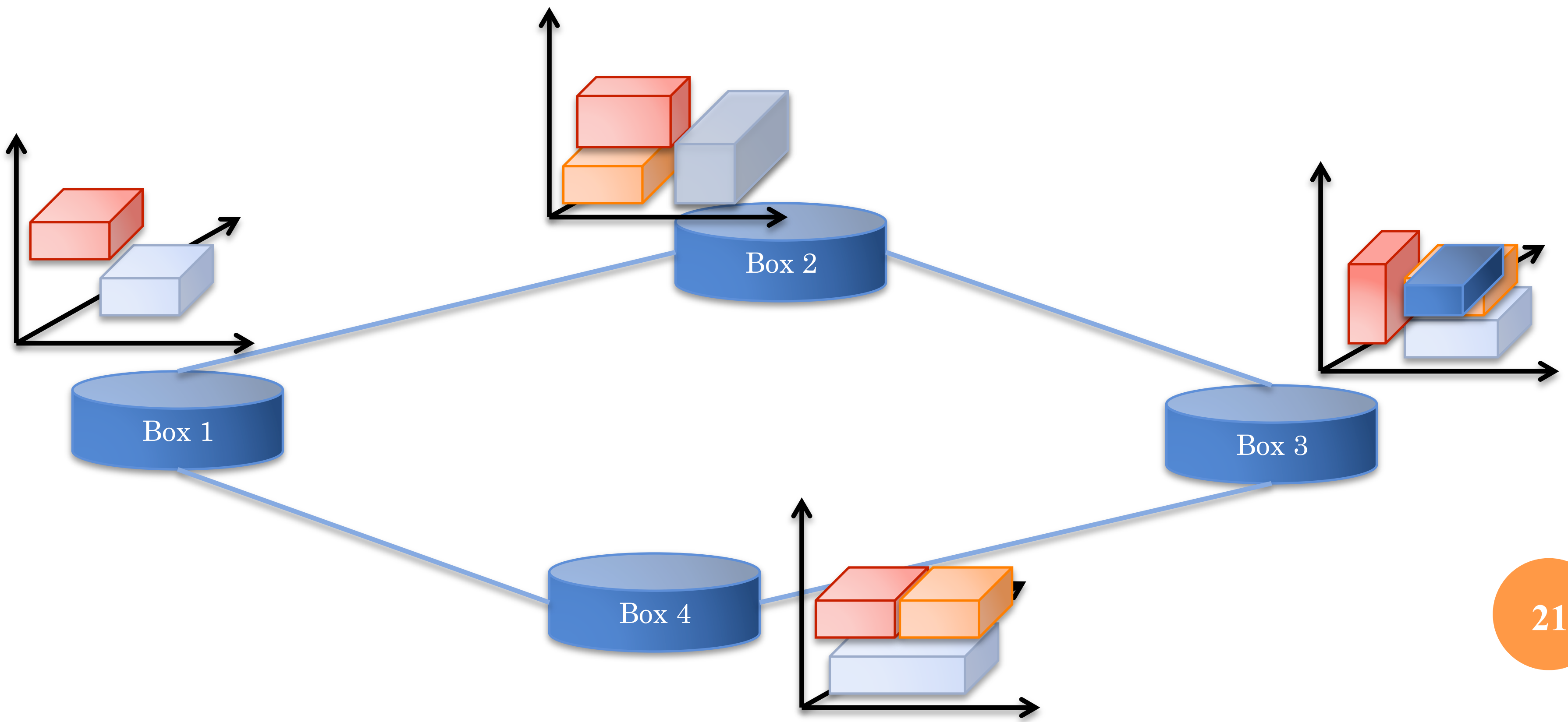
- Are two slices isolated?

- What do we mean by slice?
  - Fixed Slices: VLAN slices
  - Programmable Slices: slices created by FlowVisor

- Why do we care about isolation?
  - Banks: for added security.
  - Healthcare: to comply with HIPAA.
  - GENI: to isolate different experiments running on the same network.

USE CASES

- Are two slices isolated?
  - 1) slice definitions don't intersect.
  - 2) packets do not leak.



HEADER SPACE FRAMEWORK

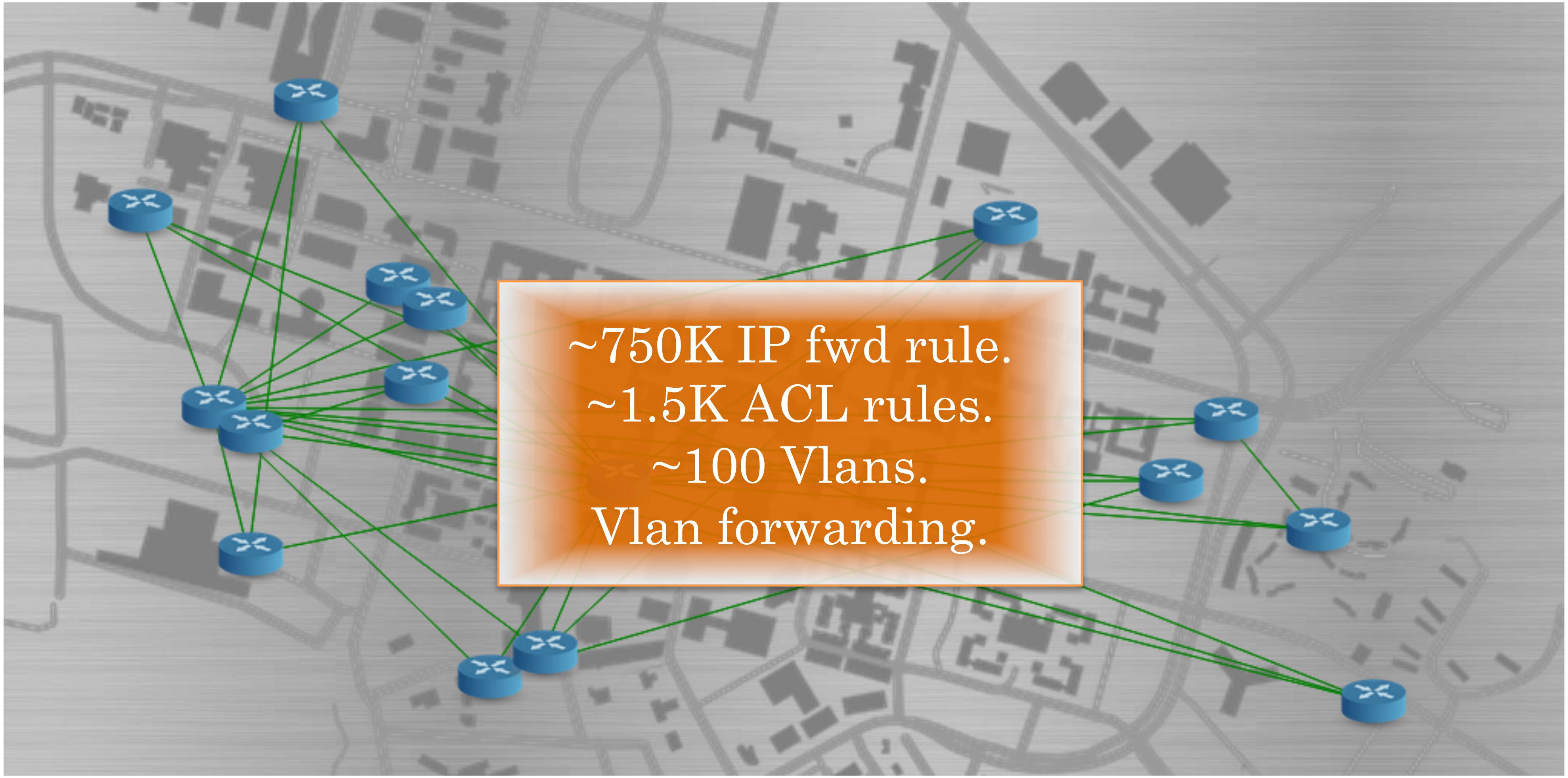
- A Powerful General Foundation that gives us
  - A common model for all packets
    - Header Space.
  - A unified view of almost all type of boxes.
    - Transfer Function.
  - A powerful interface for answering different questions about the network.
    - $T(h,p)$ and $T^{-1}(h,p)$
    - Set operations on Header Space

Implementation And Evaluation

IMPLEMENTATION

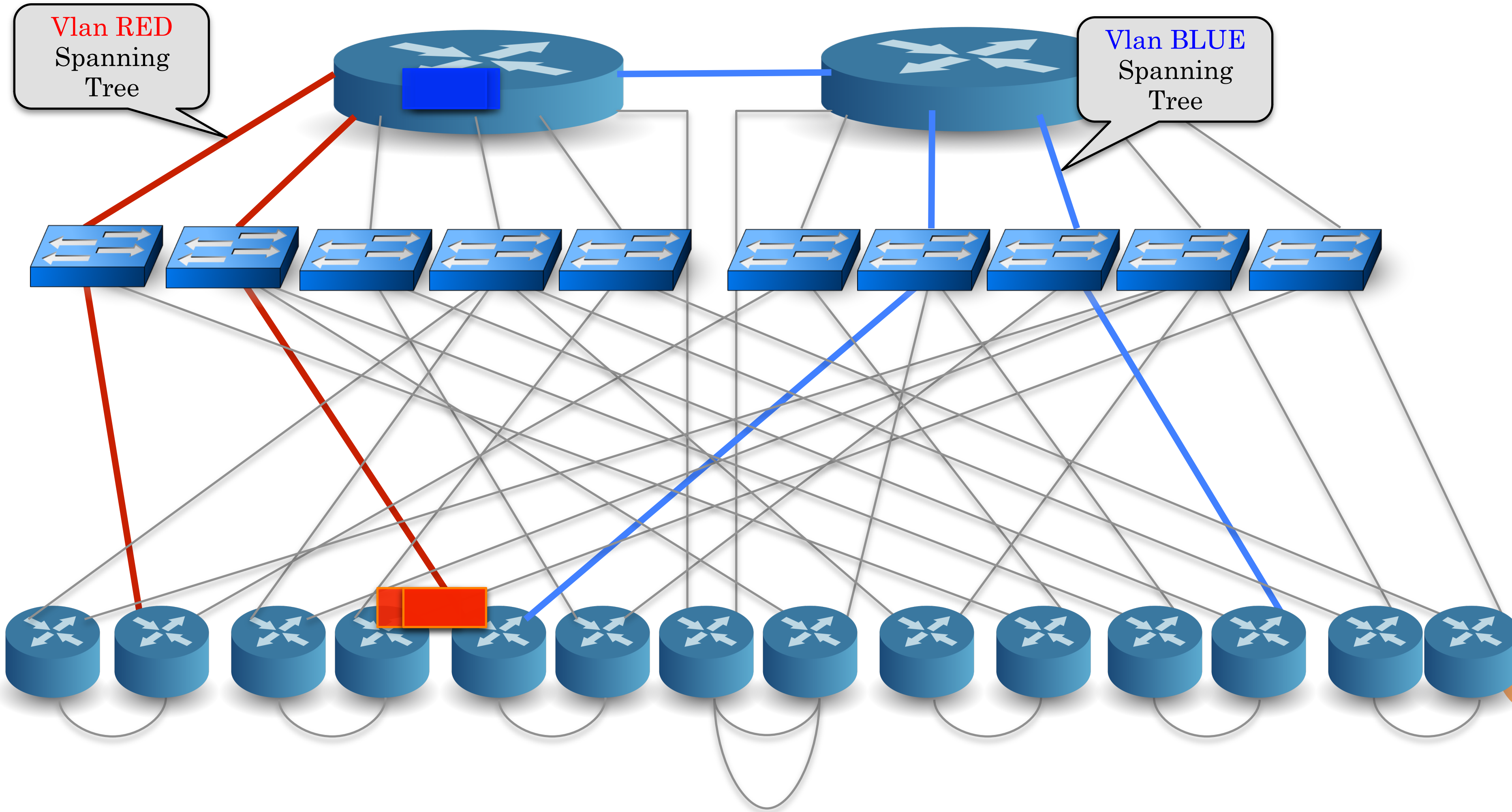
- Header Space Library (Hassel)
  - Written in Python and C.
  - Implements Header Space Class
    - Set operations
  - Implements Transfer Function Class
    - T and T^{-1}
  - Implements Reachability, Loop Detection and Slice Isolation checks.
    - < 50 lines of code
  - Includes a Cisco IOS parser, Juniper Junos Parser and OpenFlow table dump parser.
    - Generates transfer function from CLI output.
    - Keeps the mapping from Transfer function rule to line number in the CLI output.
  - Publicly available: git clone https://bitbucket.org/peymank/hassel-public.git

STANFORD BACKBONE NETWORK



Stanford backbone network

- Loop detection test – run time < 10 minutes on a single laptop.



Performance

Performance result for Stanford Backbone Network on a single machine: 4 core, 4GB RAM.

	Python	C
Generating TF Rules	~150 sec	-
Loop Detection Test (30 ports)	~560 sec	~5 sec
Average Per Port	~18 sec	~40ms
Min Per Port	~8 sec	~2ms
Max Per Port	~135 sec	~1sec
Reachability Test (Avg)	~13 sec	~40ms

Next Steps

- Automatic Test Packet Generation (To appear in CoNEXT 2012).
  - Uses HSA model to Generate minimum number of test packets to maximally cover all the "rules" in the network. (Data Plane Testing)
  - One error detected, find the location of error in data plane.

- NetPlumber: Real Time Network Policy Checker.
  - A tool to run HSA-style checks in real time by incrementally updating results as network changes.
  - Achieve on average, sub-ms run time per update for checking more than 2500 pairwise reachability checks on Google WAN.

SUMMARY

- Introduced Header Space Analysis As
  - A common model for all packets (Header Space).
  - A unified view of almost all type of boxes. (Transfer Function.)
  - A powerful interface for answering different questions about the network. (T, T^{-1}, Header Space Set Algebra)

- Showed that direct implementation of HSA algorithms scales well to enterprise-size networks.

Thank You!

Questions?

COMPLEXITY

- Run time

  Reachability: $O(dR^2)$

  Loop Detection: $O(dPR^2)$

  - R: maximum number of rules per box.
  - d: diameter of network.
  - P: number of ports to be tested

  Slice Isolation Test: $O(NW^2)$

  - W: number of wildcard expressions in definition of a slice.
  - N: number of slices in the network.

  See paper for more details.

COMPLEXITY OF REACHABILITY AND LOOP DETECTION TESTS

- Run time

  Reachability: $O(dR^2)$

  Loop Detection: $O(dPR^2)$

  - R: maximum number of rules per box.
  - d: diameter of network.
  - P: number of ports to be tested

  Assumption: Linear Fragmentation