Microsoft Research

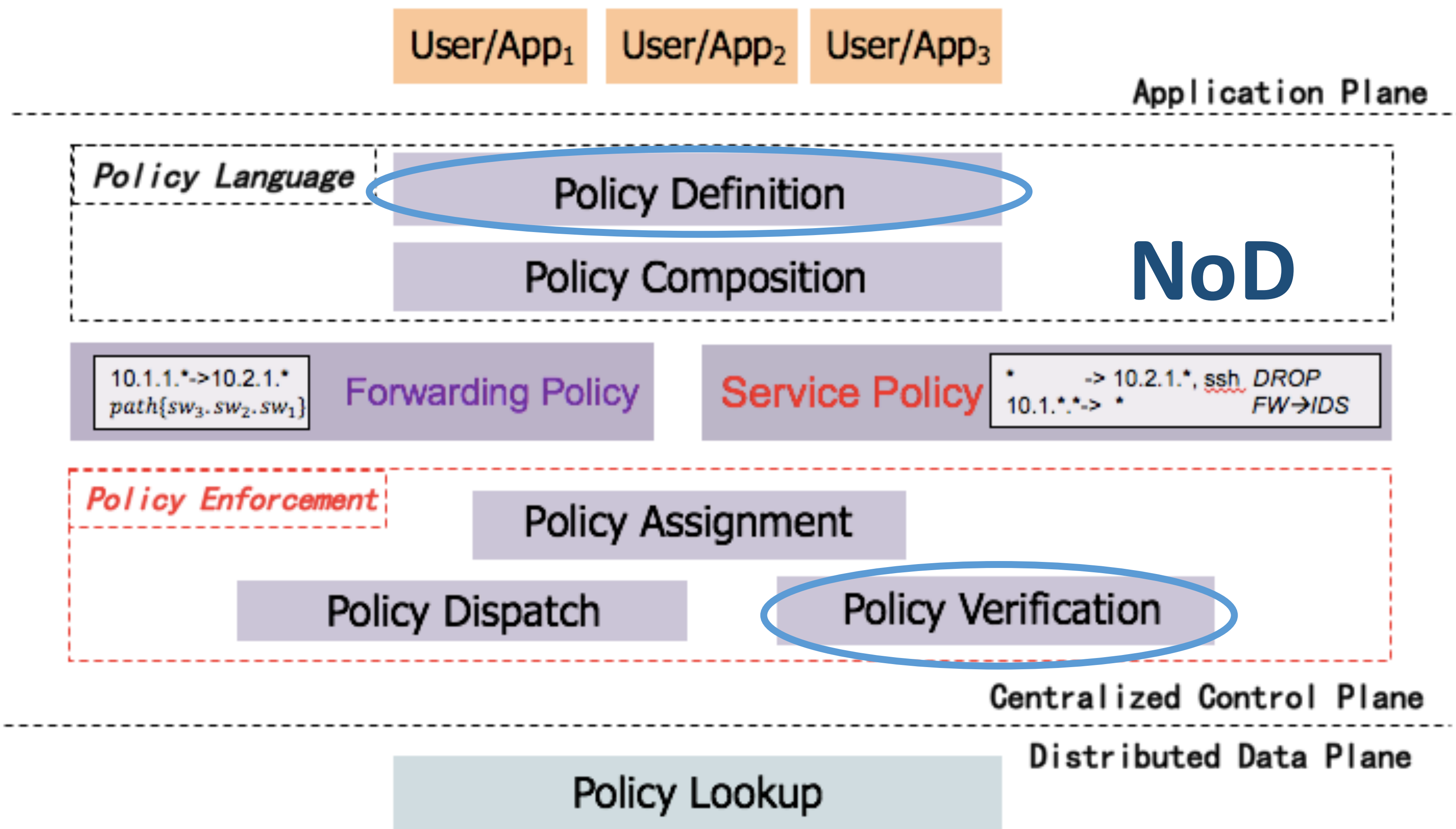
Microsoft

Checking Beliefs in Dynamic Networks

NSDI '15

Nuno Lopes, Nikolaj Bjørner, Patrice Godefroid, Karthick Jayaraman, George Varghese

Summary in the beginning



Summary in the beginning

- 用"belief"的概念代替"policy"的概念；
- 给出了belief的定义及建模方式(policy language – policy definition)；并借助已有工具(Datalog)解决规则校验(policy verification)的问题；
- 与HSA将router建模为function的思路不同，NoD将router建模为input header与output header之间的关系；——可用现有工具Datalog再稍加改进，方便的实现求解；

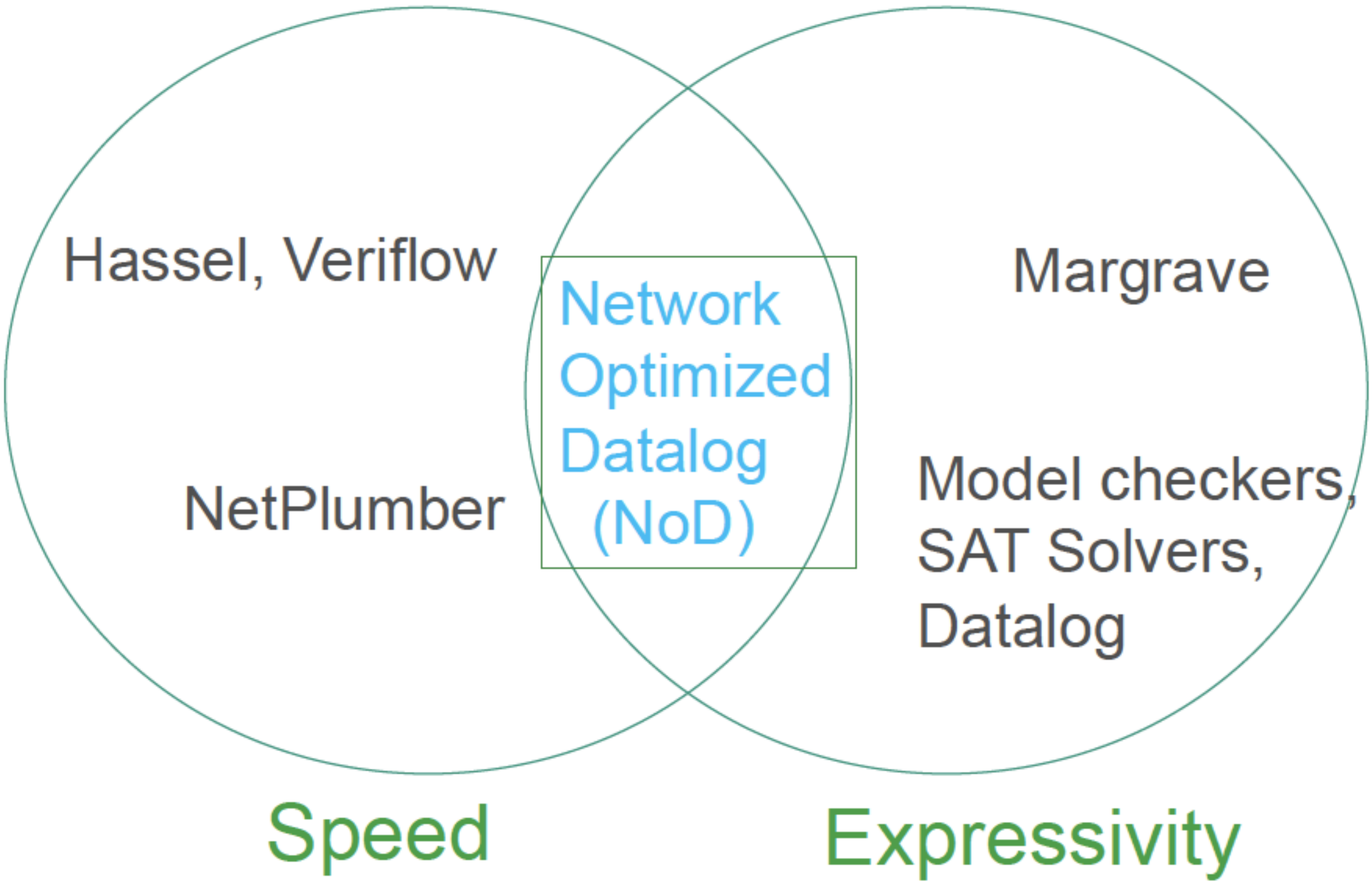
Networks

- Business critical and complex
  - Expensive bugs
- Fast protocol deployment in datacenters
  - Frequent protocol changes
- A lot of legacy to maintain
  - Operators don’t have the full picture

Network Verification to the Rescue

- Identify bugs
- Identify misbeliefs
- Increase confidence

Existing Work versus Ours



Why Expressiveness Matters

- Network level
  - Enables modeling dynamic network behaviors such as new packet headers, new forwarding behaviors, failures…
- Specification level
  - Enables higher-level verification queries, e.g.
    - Customer VMs cannot reach fabric controller
    - All backup routers are equivalent

Twofold approach to do verification

- General modeling language to model networks
- General specification language to specify beliefs

-> A verification engine that can specify beliefs and has the ability to model dynamism such as new packet headers or failures.

Solution

What is ‘Belief’ in networking?

- higher-level abstract policy specifications
  - Management stations should not be reachable from customer VMs or external Internet addresses
- A Boolean combination of reachability predicates expressed using Datalog definitions.

Example Beliefs

Policy Template	Example
Protection Sets	Customer VMs cannot access controllers
Reachable Sets	Customer VMs can access other VMs
Consistency	ECMP/Backup routes should have identical reachability
Middlebox	Forward path connections through middlebox should reverse
Locality	Packets between two hosts in the same cluster should stay within the cluster

Network-Optimized Datalog

- Datalog for the specification of
  - Data-plane/control plane
  - Verification propetries
- Tool for efficient verification
  - Available in open-source Z3

What is ‘Datalog’ ?

- 一种基于逻辑的编程语言，其语句由事实和规则组成；可以实现对知识库的演绎推理。
- 数据查询语言，专门设计与大型关系数据库交互。

- A declarative logic language in which each formula is a function-free Horn clause, and every variable in the head of a clause must appear in the body of the clause.
- A lightweight deductive database system where queries and database updates are expressed in the logic language.

What is ‘Datalog’ ?

- 一条Datalog的规则包括如下三部分的内容：

  1. 规则头P

  2. 蕴含符号 :-

  3. 规则体，即一个或多个子目标P1,P2,…,Pn，各子目标之间相当于AND连接。

规则的含义描述为：检查规则中变量的所有可能的取值，当这些变量使规则体中所有子目标均为真时，规则头为真。

Datalog Example

- A simple Datalog program:
- Facts:

```
parent(bill, mary).
parent(mary, john).
```

- Rules:

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z),ancestor(Z,Y).
```

- Query:

```
?- ancestor(bill,X).
```

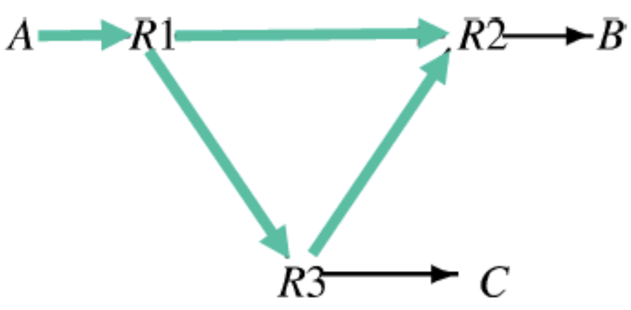
Why Datalog?

- An ideal language/tool for network verification should possess five features:
  - All Solutions
  - Packet Rewrites
  - Large Header Spaces ?
  - General Specification Language
  - General Modeling Language

Why Datalog?

- Good expressiveness / efficiency tradeoff
- Supports packet rewriting & load balancing (Datalog support non-deterministic rule)
- Provides all solutions for "free"
  - Unlike SAT solvers or model checkers

Networks as Datalog Programs



in	dst	src	rewrite	out
$R1$	$10\star$	$01\star$		$R2$
$R1$	$1\star\star$	$\star\star\star$		$R3$
$R2$	$10\star$	$\star\star\star$		B
$R3$	$\star\star\star$	$1\star\star$		C
$R3$	$1\star\star$	$\star\star\star$	$dst[1] := 0$	$R2$

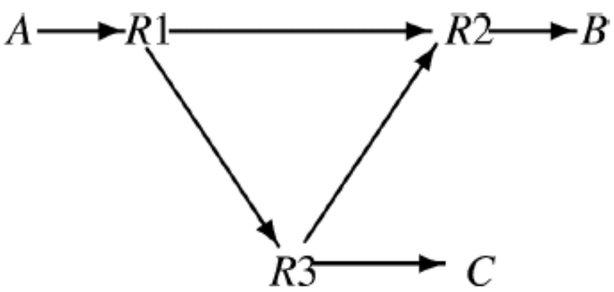
Dataplane

$$R1(dst, src) \quad :- \quad A(dst, src)$$

Guards

$$\begin{aligned}
G_{12} &:= dst = 10\star \wedge src = 01\star \\
G_{13} &:= \neg G_{12} \wedge dst = 1\star\star \\
G_{2B} &:= dst = 10\star \\
G_{3C} &:= src = 1\star\star \\
G_{32} &:= \neg G_{3C} \wedge dst = 1\star\star \\
Id &:= src' = src \wedge dst' = dst \\
Set0 &:= src' = src \wedge dst' = dst[2]\, 0\, dst[0]
\end{aligned}$$

Example of Reachability



in	dst	src	rewrite	out
$R1$	$10\star$	$01\star$		$R2$
$R1$	$1\star\star$	$\star\star\star$		$R3$
$R2$	$10\star$	$\star\star\star$		B
$R3$	$\star\star\star$	$1\star\star$		C
$R3$	$1\star\star$	$\star\star\star$	$dst[1] := 0$	$R2$

Compute all packets sent by A that reach B

$$\boxed{A(dst, src)}$$

$$\begin{array}{rcl}
R1(dst, src) & :- & A(dst, src) \\
R2(dst', src') & :- & R1(dst, src) \wedge G_{12} \wedge Id \\
R2(dst', src') & :- & R3(dst, src) \wedge G_{32} \wedge Set0 \\
R3(dst', src') & :- & R1(dst, src) \wedge G_{13} \wedge Id \\
B(dst', src') & :- & R2(dst, src) \wedge G_{2B} \wedge Id \\
C(dst', src') & :- & R3(dst, src) \wedge G_{3C} \wedge Id
\end{array}$$

$$\boxed{? \quad B(dst, src)}$$

Result:

$$10\star01\star \cup (10\star\star\star\star \setminus (10\star01\star \cup \star\star\star1\star\star))$$

$$= 10\star0\star\star$$

Beliefs and Dynamism in NoD

Policy Template	Example
Protection Sets	Customer VMs cannot access controllers
Reachable Sets	Customer VMs can access other VMs
Consistency	ECMP/Backup routes should have identical reachability
Middlebox	Forward path connections through middlebox should reverse
Locality	Packets between two hosts in the same cluster should stay within the cluster

Protection Sets

- Fabric managers are not reachable from guest virtual machines.
- With guest VMs size of 5000, manager size of 12, naïve way to express this query will explode the query to 60000 separate queries.
- But in NoD:

$$
\begin{array}{rcl}
VM(dst, src) & :- & AddrOfVM(src), AddrOfFM(dst). \\
 & ? & FM(dst, src).
\end{array}
$$

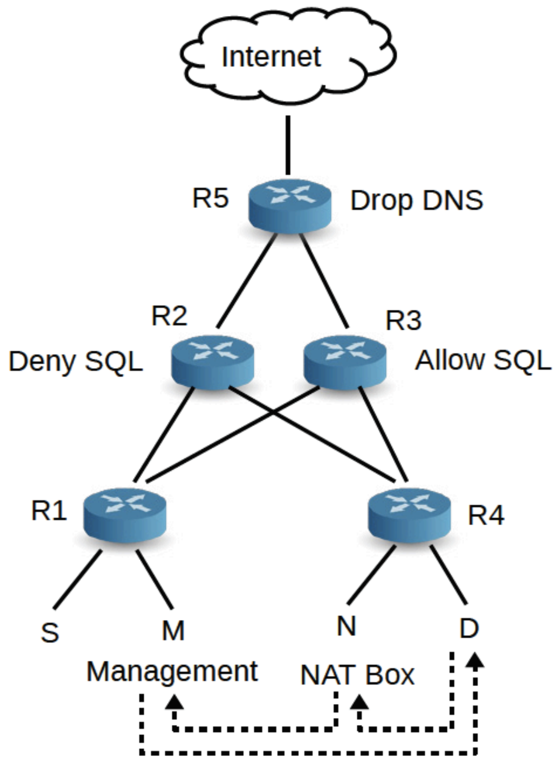
Reachability Sets

- All Fabric managers are reachable from jump boxes (internal management devices).
- In NoD:
  - query for addresses injected from jump boxes J , destined for fabric manager FM that nevertheless do not reach FM.

$$\begin{array}{rcl} J(dst, src) & :- & AddrOfJ(src), AddrOfFM(dst). \\ & ? & J(dst, src) \land \neg FM(dst, src). \end{array}$$

Equivalence of Load Balanced Paths

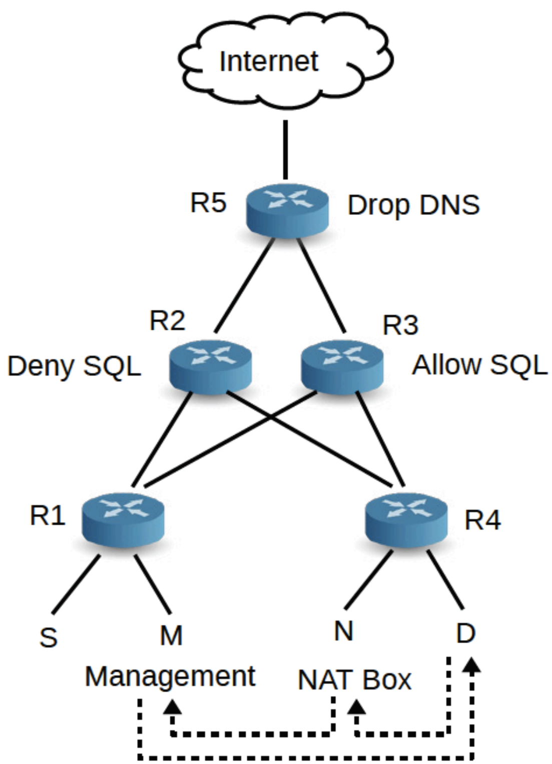
- Reachability across load balanced paths must be identical regardless of other variables such as hash functions.



$$
\begin{aligned}
R2(dst,h) &:- \quad G_{12} \wedge R1(dst,h) \wedge Select(h,dst). \\
R3(dst,h) &:- \quad G_{13} \wedge R1(dst,h) \wedge Select(h,dst).
\end{aligned}
$$

$$? \quad A(dst,h_1) \wedge \neg A(dst,h_2).$$

Locality



$$
\begin{array}{rcl}
DSP(dst) & :- & R2(dst). \\
DSP(dst) & :- & R3(dst). \\
DSP(dst) & :- & R5(dst). \\
L_{R1}(dst) & :- & dst = 125.55.10.0/24. \\
S(dst) & :- & L_{R1}(dst). \\
 & ? & DSP(dst).
\end{array}
$$

$$
\begin{array}{rcl}
L_{R4}(dst) & :- & dst = 125.75.10.0/24. \\
D(dst) & :- & L_{R4}(dst). \\
N(dst) & :- & L_{R4}(dst).
\end{array}
$$

Middleboxes and Backup Routers

- Incorrect Middlebox traversal
  - packets should go through the same set of middleboxes in the forward and reverse path.
  - add a fictitious bit to packets that is set when the packet passes through a middlebox.
- Backup Non-equivalence:
  - all paths between a source and destination pair passing through any one of a set of backup routers should have the same number of hops.
  - Encode path lengths in Datalog as a small set of control bits in a packet, and query whether a destination is reached from the same source across one of the set of backup routers, but using two different path lengths

Dynamic Packet Headers

- Datalog one does not require a priori definitions of all needed protocols headers before starting an analysis. One can easily define new headers post facto as part of a query.
- one can also define new forwarding behaviors as part of the query.

Bounded label stacking example

$$
\begin{aligned}
R2^1(dst, src, 2016) &\quad :- \quad G, R5^0(dst, src).\\
R2^2(dst, src, l_1, 2016) &\quad :- \quad G, R5^1(dst, src, l_1).\\
R2^3(dst, src, l_1, l_2, 2016) &\quad :- \quad G, R5^2(dst, src, l_1, l_2).\\
Ovfl(dst, src, l_1, l_2, l_3) &\quad :- \quad G, R5^3(dst, src, l_1, l_2, l_3).
\end{aligned}
$$

So what’s wrong with Datalog?

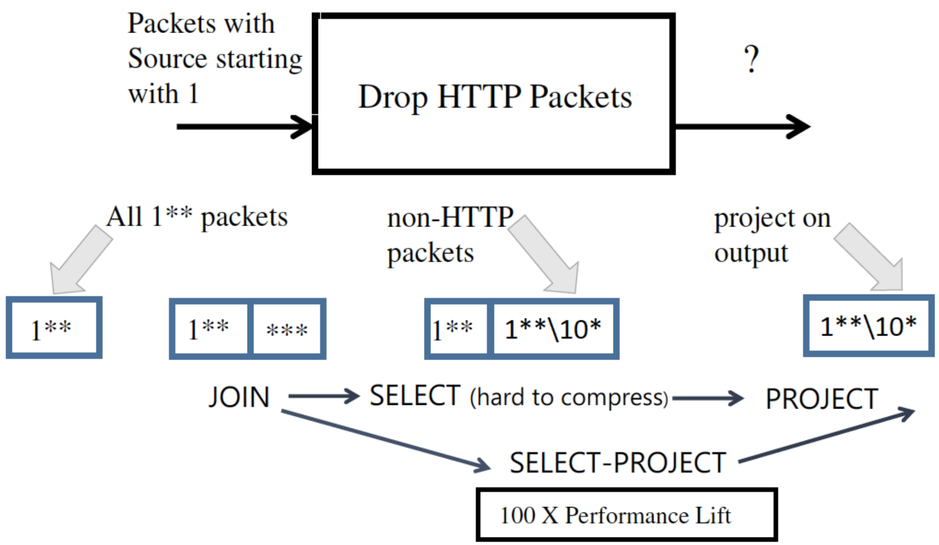
- Out-of-the-box implementations are slow
  - Work with a packet a time

- Our contributions:
  - Symbolic representation (dealing with sets of packets)
  - Efficient propagation of packets across routers

Network-Optimized Datalog

- Based on μZ (Microsoft Datalog framework)
  - A standard suite of database operators such as select, project, join to manipulate tables representing sets of packet headers.
  - The router relation models the forwarding behavior of the router including all forwarding rules and ACLs.
  - Pose reachability queries to μZ to compute the set of packets that flow from A to B.
- Optimization:
  - Compact Data Structure
  - Combining Select and Project

Compact Data Structure



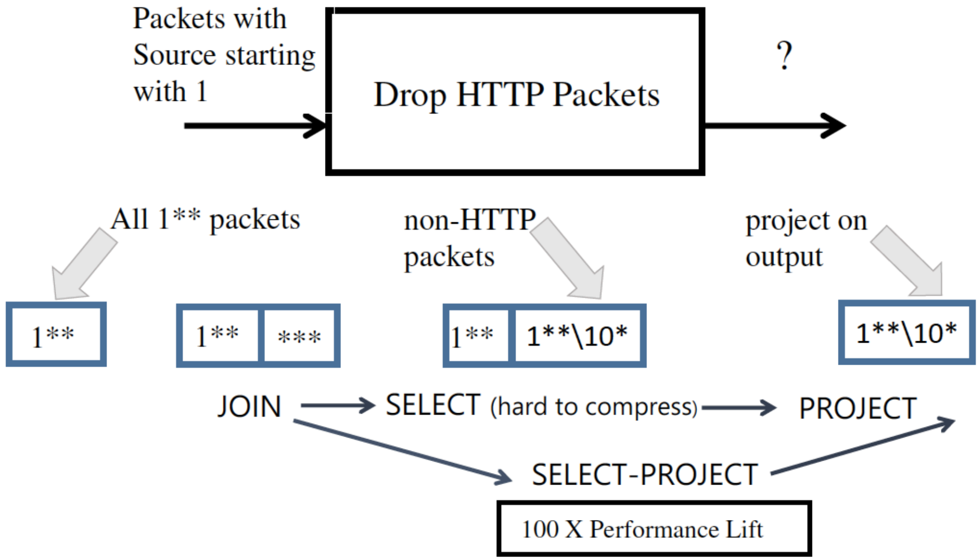
- The natural way for Datalog to represent a set of packet headers is as a table. -> 128-bit headers takes 2^{128} rows.
- Two table backends:
  - BDDs (a classic data structure to compactly represent a Boolean function)
  - DoCs (difference of cubes)

Compact Data Structure

- The natural way for Datalog to represent a set of packet headers is as a table. -> 128-bit headers takes 2^{128} rows.
- Two table backends:
  - BDDs (a classic data structure to compactly represent a Boolean function)
  - DoCs (difference of cubes)

- DoC:
  - Based on ternary bit vectors (similar to HSA)
  - 1**\10* $\bigcup_i \left(v_i \setminus \bigcup_j v_j \right)$
  - Particularly efficient at representing router rules that have dependencies.

Combining Select and Project



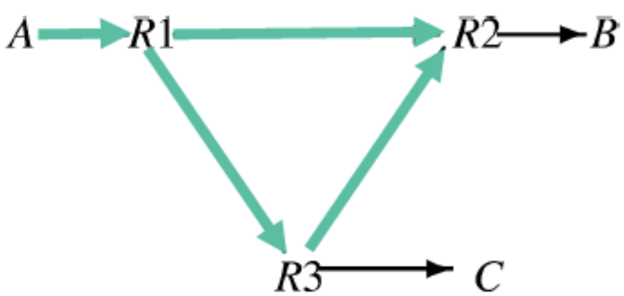
- μZ computes the set of output packets by finding a relation between input packets and corresponding output packets.
  - Joins the set of input packets I to the set of all possible output packets A to create a relation (I,A).
  - Selects the output packets (rows) that meet the matching and rewrite conditions to create a pruned relation (I, O).
  - Projects away input packets and produces the set of output packets O.

Combining Select and Project

- The output of the select which is extremely inefficient to represent. But the output of the select is merely a way station on the path to the output;

-> We do not need to explicitly materialize this intermediate result.

Packet Rewriting Example



in	dst	src	rewrite	out
$R1$	$10\star$	$01\star$		$R2$
$R1$	$1\star\star$	$\star\star\star$		$R3$
$R2$	$10\star$	$\star\star\star$		B
$R3$	$\star\star\star$	$1\star\star$		C
$R3$	$1\star\star$	$\star\star\star$	$dst[1] := 0$	$R2$

- Input packet 1** *** at R3
- Join input and output: 1** ***, *** ***
- Apply guard and rewrite formulas and handle rule's dependency: 1** ***, 10* 0**
- But this is not enough: the copying relationship is incorrectly represented! 1*1 ***, 100 0** is also allowed!
- So: $$1\star\star\star\star\star10\star0\star\star \setminus (\star\star0\star\star\star\star\star1\star\star\star \cup \star\star1\star\star\star\star\star0\star\star\star \cup \star\star\star\star0\star\star\star\star\star1\star \cup \star\star\star\star1\star\star\star\star\star0\star \cup \star\star\star\star\star0\star\star\star\star\star1 \cup \star\star\star\star\star1\star\star\star\star\star0)$$ However, after projection: $10\star0\star\star$

Combining Select and Project

- Compute the projection implicitly without explicitly materializing intermediate results;
- Using a standard union-find data structure to represent equivalence classes (copying) between columns;

Evaluation

Benchmarks

- Stanford
  - A snapshot of the routing tables of Stanford backbone;
  - 16 routers, 12978 rules, extensive NAT and VLAN support.
- Generic Cloud Provider
  - A parameterizable model of a cloud provider network.
  - Fat tree topology.
- Production Cloud
  - Live data center located in HK and Singapore of Microsoft.
  - 3 clusters and thousands of machines, each ~2000 ECMP forwarding rules (200K rules).
- Experimental Backbone
  - SDN backbone based on the SWAN design.

Evaluation questions

- Do beliefs help?
- How hard is it to add a new forwarding protocol?
- How does NoD performs compared with existing verification tools?
- Is this useful in practice?

Protection sets test

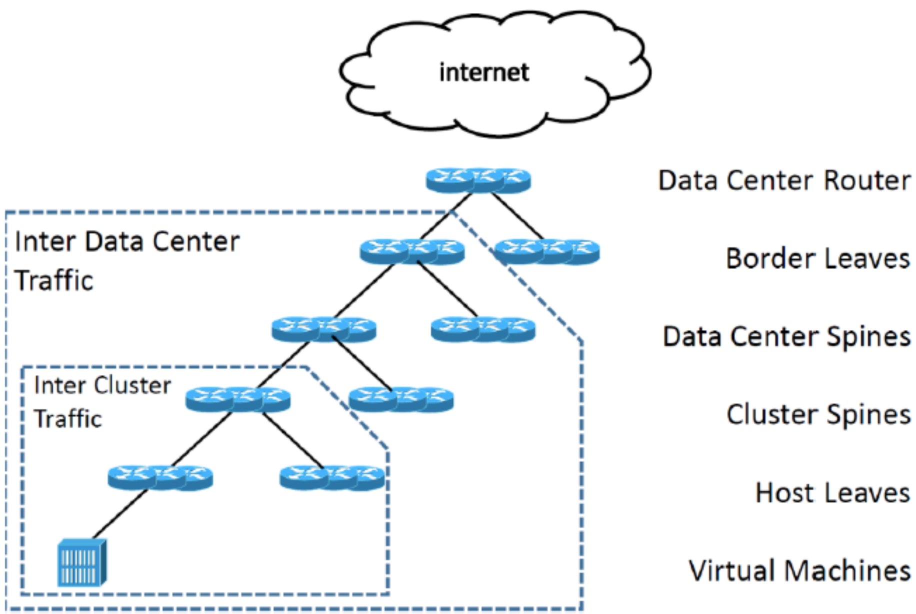
- Protection: checked whether two policies based on the Protection sets template hold in the Singapore data center.
- neither Internet addresses or customer VMs can access the protected fabric controllers for security reasons. (The set of addresses are very large;)
- Takes ~12 mins;
- NoD has the power of exploring a very general beliefs.

Reachable sets test

- Reachable: checked whether two policies from the reachable sets template hold in the same Singapore data center.
- If all of “utility boxes” can reach all “fabric controllers”:
  - Takes ~4 mins.
- Checks whether “service boxes” can reach “fabric controllers:
  - Takes ~6 mins.

Locality test

- Found multiple violations of traffic locality



Query	Cluster 1	Cluster 2	Cluster 3
C2C	12 (2)	13 (2)	11 (2)
B2DSP	11 (2)	11 (2)	11 (2)
B$\bar{2}$DSP	3 (1)	4 (1)	4 (1)
B2CSP	11 (2)	11 (2)	11 (2)
B$\bar{2}$CSP	11 (2)	12 (2)	11 (2)

Verification time in seconds

Dynamism Test

- Experimental MPLS-like backbone with custom forwarding.
- Took a few hours to model without any tool change.
- Loop detection in <1 second.
- Identified 56 flows as black holes in 5 seconds.

Differential Reachability

- Use middle size synthetic cloud benchmark.
- Change the ALCs at one of the core routers such that one of the links in a set of load balanced paths allowed VLAN 3 and blocked VLAN 1, while all other links blocked VLAN 1 and allowed VLAN 3.
- Check the difference in reachability across all load-balanced paths between the Internet and a host in the data center.
- Take ~1.9s

Comparison with existing tools

Test	Model Checkers		SMT		Datalog		Hassel C
	BMC	PDR	Reach.	All sols.	BDDs	DoC	
Small Cloud	0.3	0.3	0.1	–	0.2	0.2	–
Medium Cloud	T/O	10.0	0.2	–	1.8	1.7	–
Medium Cloud Long	M/O	M/O	4.8	–	7.4	7.2	–
Cloud More Services	7.2	8.5	12.5	–	5.3	4.8	–
Large Cloud	T/O	M/O	2.8	–	16.1	15.7	–
Large Cloud Unreach.	T/O	M/O	1.1	n/a	16.1	15.7	–
Stanford	56.2	13.7	11.5	1,121	6.6	5.9	0.9
Stanford Unreach.	T/O	12.2	0.1	n/a	2.6	2.1	0.1
Stanford Loop	20.4	11.7	11.2	290.2	6.1	3.9	0.2

Network Verification in Production

- Simplified version of NoD: SecGuru
  - Local checks on each router
- Deployed in Azure
- Finds ~1 problem per day

Conclusion

- NoD is expressive:
  - Protocol specification -> Dynamism
  - Verification properties -> Beliefs
- More expressive than previous network verification tools, while competitive in speed
- Network operator's beliefs are fragile
- Code and benchmarks available online