A Fast Compiler For NetKAT

Steffen Smolka, Spiridon Eliopoulos, Nate Foster, Arjun Guha

Presented By Yonathan Fisseha (Group 28)

Outline

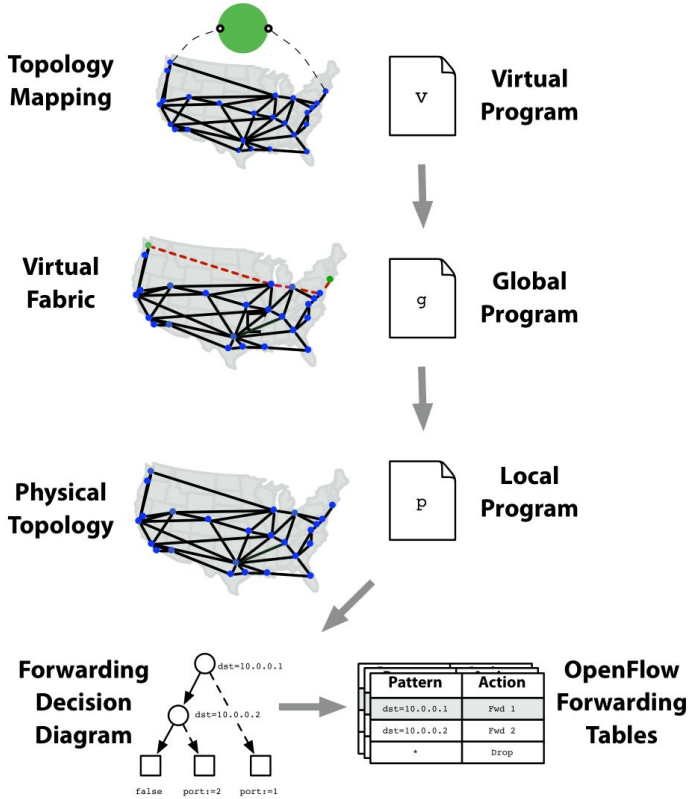
1. Network Programming
2. Introduction to NetKAT
3. Compiler Challenges for NetKAT
4. The Local Compiler
5. The Global Compiler
6. Virtualization
7. Experiments
8. Comments

Network Programming

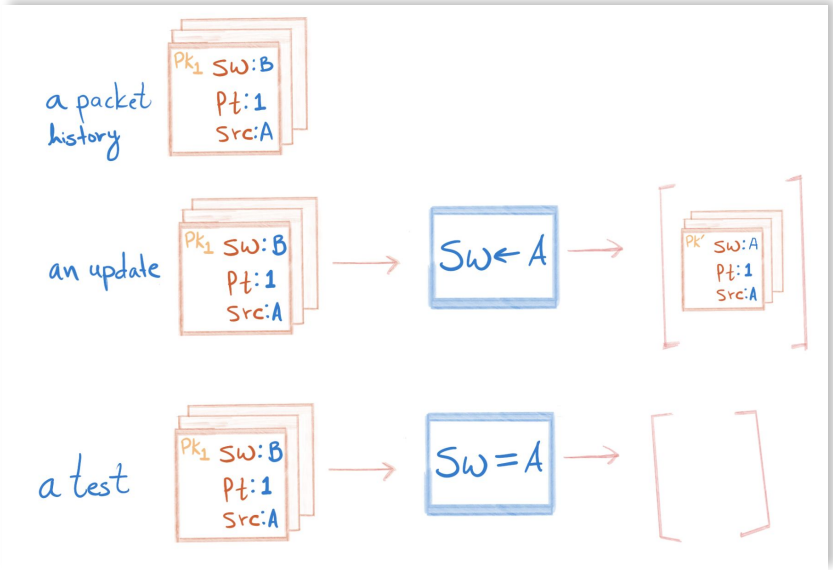
- Reprogrammable network switches enable software defined networking (SDN)
- High-level languages are used to express network design and properties
- The compiler translates the high-level language constructs to hardware-level table-based forwarding rules

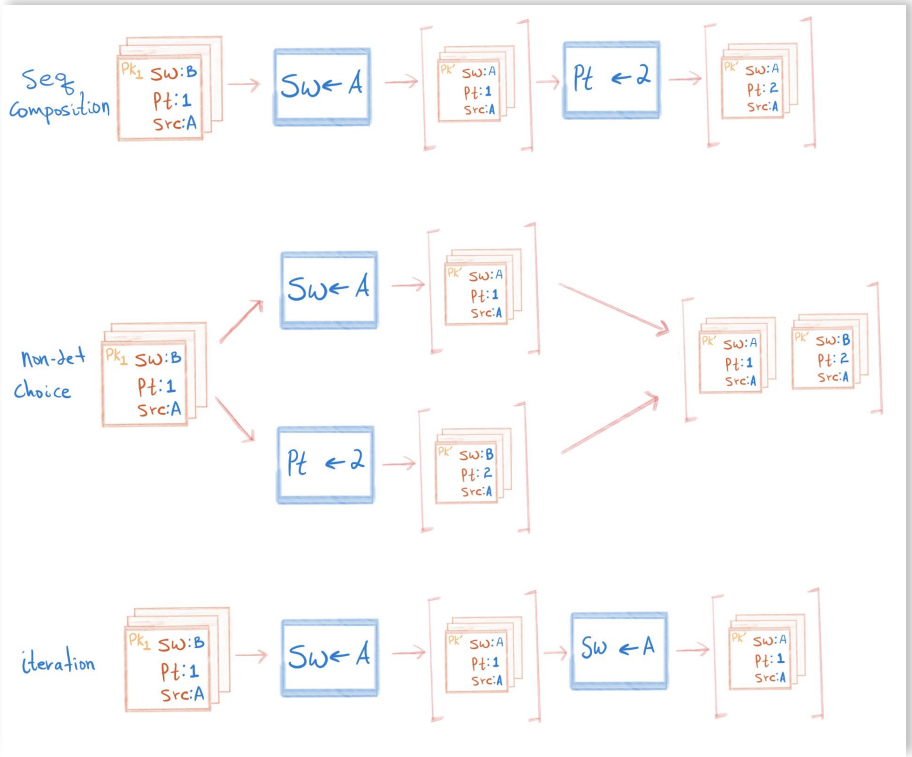
NetKAT: Yet Another Programming Language

- A Verification First language where formal verification is built into the language from the beginning
- Provides formal semantics for network programming based on Kleene Algebra with Tests (KAT)
- Verification is performed via a semi-automatic equational reasoning in the algebraic framework
- Can express both local (switch level) and global (across switches/topological) behaviors



NetKAT Semantics





Compiling NetKAT

- Global programs are hard to compile since they break switch boundaries
- The compiler must insert and maintain some state for global programs
- Local programs could be converted to forwarding tables easily but this approach is inefficient

Pattern	Action
⋆	pt←2

$pol_A \triangleq \mathsf{pt}{\leftarrow}2$

(a) An atomic modification

Pattern	Action
dst=A	*true*
⋆	*false*

$pol_B \triangleq \mathsf{dst}{=}\mathsf{A}$

(b) An atomic predicate

Pattern	Action
dst=A	pt←2
⋆	*false*

$pol_B \cdot pol_A$

(c) Forwarding to a single host

Pattern	Action
dst=A	pt←1
dst=B	pt←2
⋆	*false*

$pol_D \triangleq \mathsf{dst}{=}\mathsf{A} \cdot \mathsf{pt}{\leftarrow}1 + \mathsf{dst}{=}\mathsf{B} \cdot \mathsf{pt}{\leftarrow}2$

(d) Forwarding traffic to two hosts

Pattern	Action
dst=A	pt←3
proto=ssh	pt←3
⋆	*false*

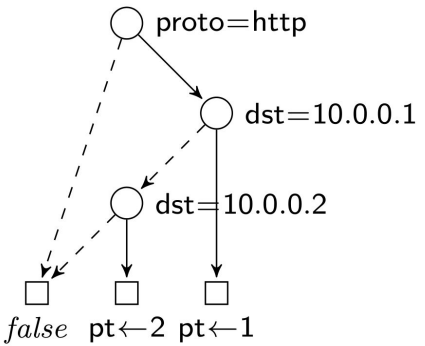
$pol_E \triangleq \begin{pmatrix} \mathsf{proto}{=}\mathsf{ssh} + \\ \mathsf{dst}{=}\mathsf{A} \end{pmatrix} \cdot \mathsf{pt}{\leftarrow}3$

(e) Monitoring SSH traffic and traffic to host A

The Local Compiler

- Forwarding Decision Diagrams (FDDs) are binary trees where the nodes are tests and the leaves are assignments
- Translate NetKAT programs to FDDs. This is sound (Theorem 1)
- The FDDs are in ordered and reduced form
- The order of the fields is reflected in the order of the rules in the table

proto = http * (dst = 10.0.0.1 * pt ← 1 + dst = 10.0.0.2 * pt ← 2)



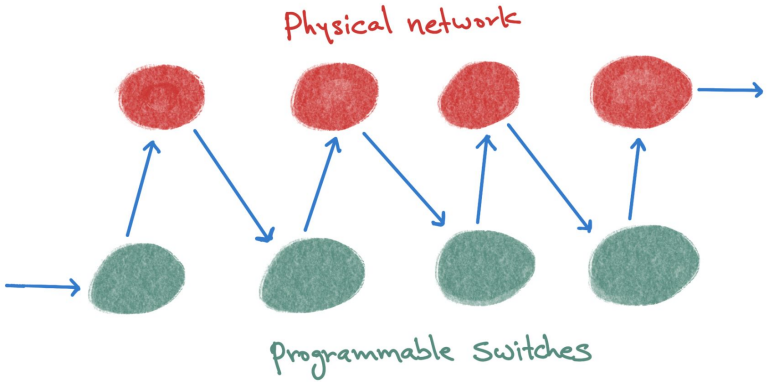
Pattern	Action
proto=http, dst=10.0.0.1	pt←1
proto=http, dst=10.0.0.2	pt←2
proto=http	*false*
⋆	*false*

The Global Compiler

- The NetKAT automaton is defined on guarded strings of form

$$pk_{in} * pk_1 * dup * pk_2 \ldots * pk_{out}$$

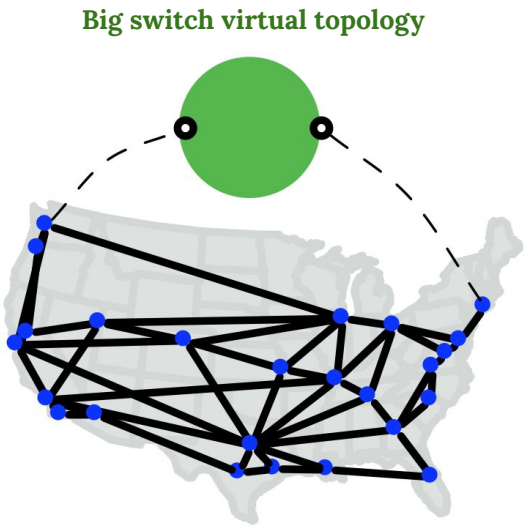
- A new PC field tracks the state of the automaton
- All NetKAT programs can be converted to a NetKAT automata. This is sound (Theorem 2)
- Textbook determinization and optimization techniques can be applied
- Take union of the FDDs of the local programs (green blobs in the NA) and treat it as a local program!



Extract the local programs from the global program and let the physical network provide the links between the switches

Virtualization

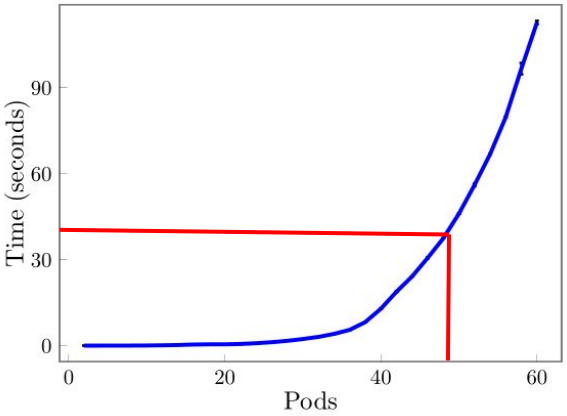
- Virtualization is an abstraction that allows programmers to write more concise programs
- Virtual programs are defined against a virtual topology decoupling the program from the physical topology
- Can target various physical topologies
- Can provide a simple form of traffic isolation at the program level
- The compiler now needs to lower the virtual program to a specific physical topology



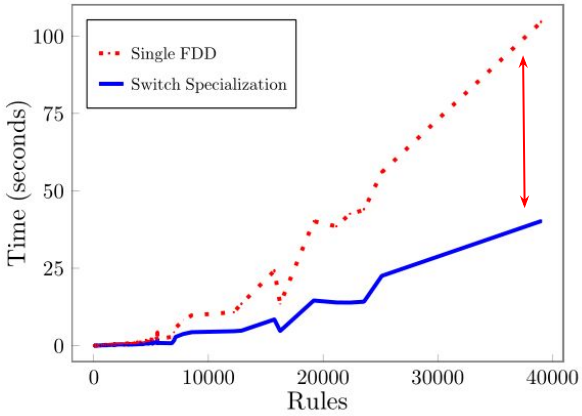
Virtualization

- The compiler gets
  - a relation R that maps Virtual Ports to Physical ports
  - a virtual topology
  - sets of in and out ports
- Packets are tracked across the virtual topology and the physical topology
- The fabric is a NetKAT program that updates the location of the packets in the physical topology when its location changes in the virtual topology (and vice versa)
- Challenge: keeping the two locations consistent!
- Solution: a game between V (the program) and F (the fabric). F attempts to keep up with V; if it can't, the physical topology can't implement the program
- There could be multiple fabrics, each optimizing for different variables (e.g. # of hops, # of links)

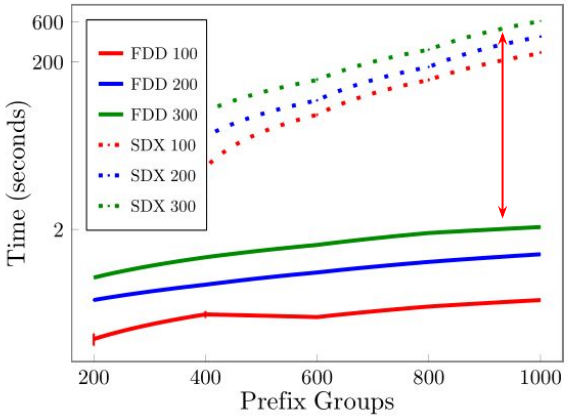
Experiments: Compilation Time for Local Compiler



(a) Routing on k-pod fat-trees.



(b) Destination-based routing on topology zoo.



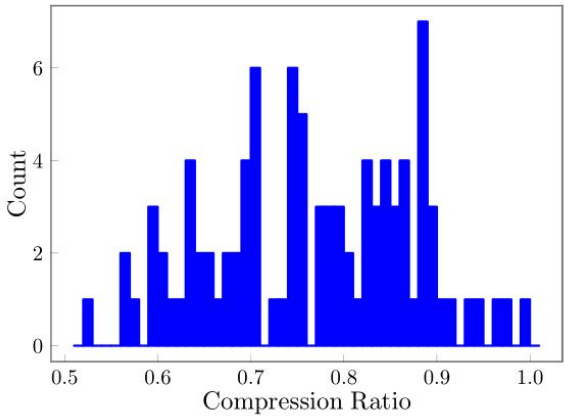
(c) Time needed to compile SDX benchmarks.

a) compilation time for a data-center size network is okay

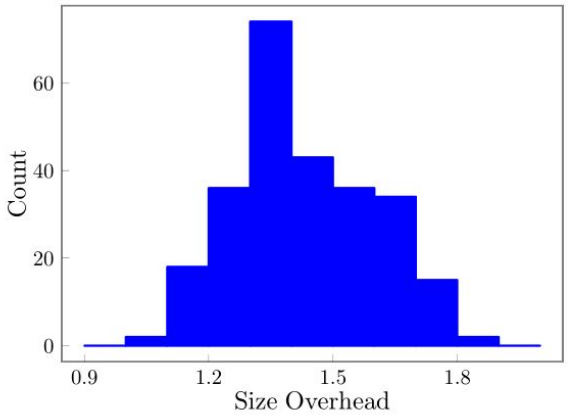
b) generating multiple small FDDs first helps

c) FDD ~10x better for the largest group (1k) and participants (300)

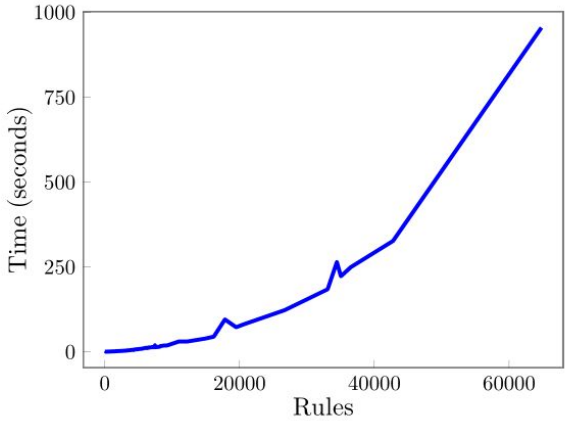
Experiments: Table Size and Time for Global Programs



(a) Compressing Classbench ACLs.



(b) Table size overhead for global programs.



(c) Compilation time for global programs.

a) 30% fewer rules on average (with simple optimizations)

b) Within 2x range of local tables (without minimizing the NetKAT automata)

c) Much slower than local compiler

Conclusion and Comments

- A compiler for the full NetKAT language with acceptable compilation time and table size overhead
- The concepts used (BDDs and finite automata) are well known and offer many optimizations/improvements directions for the compiler
- The formalism and syntax is awkward for FDDs since they need to inductively respect the ordering. Compare this to vanilla BDDs.
- Getting the ordering right is important for BDDs!
- GKAT (POPL'20) brings decision complexity to near-linear (compared to NetKAT's PSPACE) for a fragment of KAT. The automaton optimizations used here might not be needed if NetKAT moves to GKAT instead of full KAT.

Thank You!

Questions?

Yonathan Fisseha
yonathan@umich.edu