Testing and verifying SDN applications with NetKAT

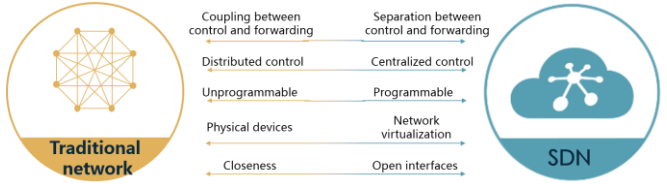
Czentye János, czentye@tmit.bme.hu

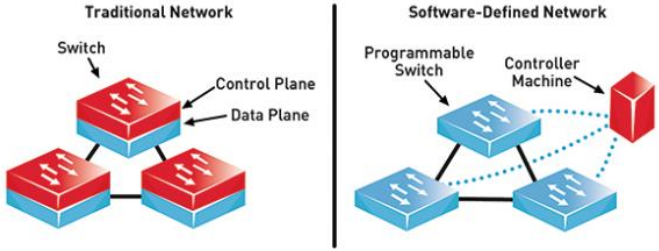
SINCE 1992

HSN Lab

Software-Defined Networking (SDN)

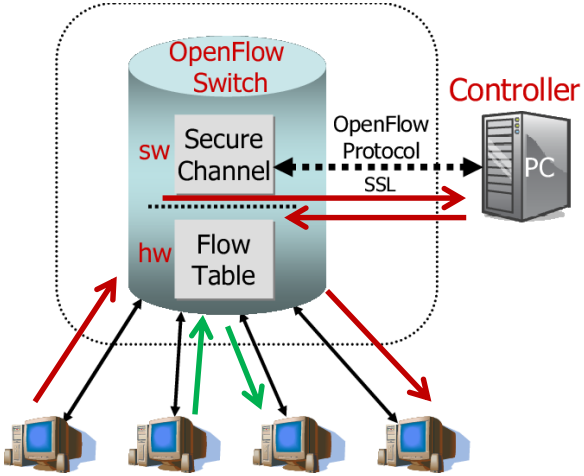
- Traditional networks
  - Decentralized
  - Complex cfg. (routing protocols)
- SDN networks
  - Centrally managed → Controller
  - Directly programmable
  - Flexible and agile
  - Programmatically configured
  - Open standards-based
  - Vendor-neutral





OpenFlow

OpenFlow 1.0 Flow table basics



Reactive vs. Proactive

Examples

Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	*	*	*	*	*	*	port6

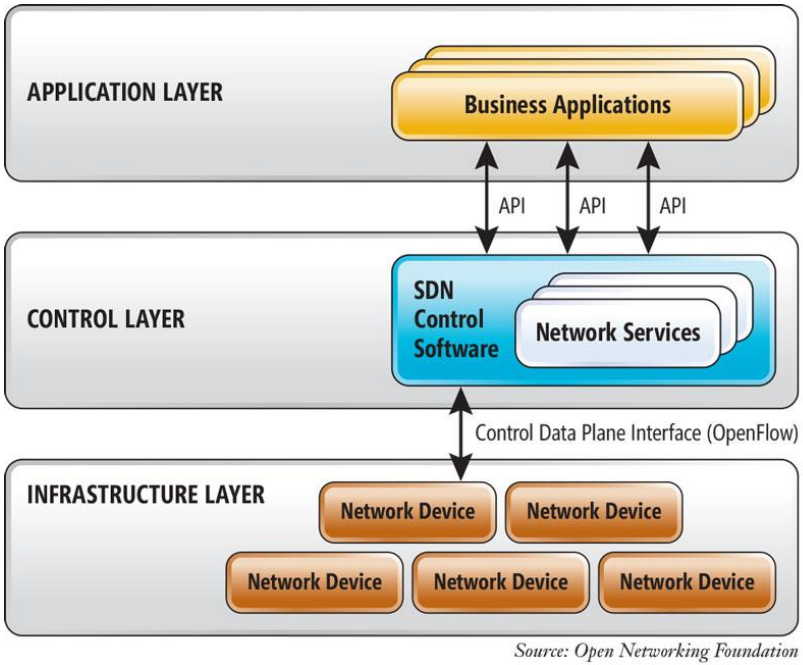
Flow Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
port3	00:20..	00:1f..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6

Firewall

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	*	*	*	22	drop

Open Network Foundation: SND Architecture



Source: Open Networking Foundation

Northbound Interface, NBI

Southbound Interface, SBI

SDN concept overview

- Advantages
  - General-purpose controller machine (no dedicated HW)
  - Programmable switches with standard API (OpenFlow)
  - Centralized & global view of network!
  - No hardware middle-boxes
  - Network logic is realized by SW written in any language
- Disadvantages
  - Software bugs → new kind of network errors
  - (controller = SPOF)
- → Software V&V methods + network knowledge

Network Programming Languages

- Domain-specific languages for SDN
  - Raise the level of abstraction above OpenFlow
  - Increase reliability & expressiveness
  - Target: NBI or SBI
  - Clear semantics → to analyze and verify SDN behavior
- Initial results on NPL: The Frenetic Project
  - 1) generates static forwarding policies
  - 2) policies are compiled to OpenFlow

frenetic ≫

- → More sophisticated foundational model: **NetKAT**

NetKAT: intro

- New network programming language (framework) for specifying, programming and reasoning about networks
- Based on solid mathematical foundation
  - *Kleene algebra with tests (KAT)*
  - Sound and complete equational theory
- Practical applications
  - Checking reachability in network (*connectivity*)
  - Proving non-interference properties (traffic *isolation*)
  - Establishing reliable (SDN) applications (*correctness*)

NetKAT: semantic foundations 1/2

- Network = automaton
  - Moves packets from node to node along the links in the network
- Global network behavior → *Kleene algebra (KA)*
  - Path = concatenation of steps, $p \cdot q \cdot$ …
  - Set of path = union of paths, $p + q +$ …
  - Iterated processing = Kleene star operator, $(p \cdot q)^*$
- Switch-processing functionality
  - Predicate: to match packets → *Boolean algebra*
  - Action: to transform or forward packets (policy)
- Kleene + Boolean → Kleene algebra with tests (KAT)

NetKAT: semantic foundations 2/2

- NetKAT = KAT applied on the network domain
- Atomic NetKAT axioms & policies related to packets:
  - Packet = record of header fields + location fields: switch, port
  - Filter: yield the packet if field is equal to n, $(f = n)$
  - Modify: yield a packet with overwritten field to n, $(f \leftarrow n)$
    - Transmitting: modifying location fields, $(sw \leftarrow n)$ or $(pt \leftarrow n)$
- Policy combinators:
  - Union: apply policies on a packet, $(p + q)$
  - Sequential composition: apply policies in order: $(p \cdot q)$
- History: non-empty sequence of packets

NetKAT: full syntax, semantics & axioms

Syntax

Fields	$f ::=$	$f_1 \mid \cdots \mid f_k$	
Packets	$pk ::=$	$\{f_1 = v_1, \cdots, f_k = v_k\}$	
Histories	$h ::=$	$pk::\langle\rangle \mid pk::h$	
Predicates	$a, b ::=$	1	*Identity*
		0	*Drop*
		$f = n$	*Test*
		$a + b$	*Disjunction*
		$a \cdot b$	*Conjunction*
		$\neg a$	*Negation*
Policies	$p, q ::=$	a	*Filter*
		$f \leftarrow n$	*Modification*
		$p + q$	*Union*
		$p \cdot q$	*Sequential composition*
		p^*	*Kleene star*
		dup	*Duplication*

Semantics

$$[\![p]\!] \in \mathsf{H} \to \mathcal{P}(\mathsf{H})$$

$$[\![1]\!]\ h \triangleq \{h\}$$

$$[\![0]\!]\ h \triangleq \{\}$$

$$[\![f = n]\!]\ (pk::h) \triangleq \begin{cases} \{pk::h\} & \text{if } pk.f = n \\ \{\} & \text{otherwise} \end{cases}$$

$$[\![\neg a]\!]\ h \triangleq \{h\} \setminus ([\![a]\!]\ h)$$

$$[\![f \leftarrow n]\!]\ (pk::h) \triangleq \{pk[f := n]::h\}$$

$$[\![p + q]\!]\ h \triangleq [\![p]\!]\ h \cup [\![q]\!]\ h$$

$$[\![p \cdot q]\!]\ h \triangleq ([\![p]\!] \bullet [\![q]\!])\ h$$

$$[\![p^*]\!]\ h \triangleq \bigcup_{i \in \mathbb{N}} F^i\ h$$

where $F^0\ h \triangleq \{h\}$ and $F^{i+1}\ h \triangleq ([\![p]\!] \bullet F^i)\ h$

$$[\![\mathsf{dup}]\!]\ (pk::h) \triangleq \{pk::(pk::h)\}$$

Kleene Algebra Axioms

$p + (q + r) \equiv (p + q) + r$	KA-PLUS-ASSOC
$p + q \equiv q + p$	KA-PLUS-COMM
$p + 0 \equiv p$	KA-PLUS-ZERO
$p + p \equiv p$	KA-PLUS-IDEM
$p \cdot (q \cdot r) \equiv (p \cdot q) \cdot r$	KA-SEQ-ASSOC
$1 \cdot p \equiv p$	KA-ONE-SEQ
$p \cdot 1 \equiv p$	KA-SEQ-ONE
$p \cdot (q + r) \equiv p \cdot q + p \cdot r$	KA-SEQ-DIST-L
$(p + q) \cdot r \equiv p \cdot r + q \cdot r$	KA-SEQ-DIST-R
$0 \cdot p \equiv 0$	KA-ZERO-SEQ
$p \cdot 0 \equiv 0$	KA-SEQ-ZERO
$1 + p \cdot p^* \equiv p^*$	KA-UNROLL-L
$q + p \cdot r \leq r \Rightarrow p^* \cdot q \leq r$	KA-LFP-L
$1 + p^* \cdot p \equiv p^*$	KA-UNROLL-R
$p + q \cdot r \leq q \Rightarrow p \cdot r^* \leq q$	KA-LFP-R

Additional Boolean Algebra Axioms

$a + (b \cdot c) \equiv (a + b) \cdot (a + c)$	BA-PLUS-DIST
$a + 1 \equiv 1$	BA-PLUS-ONE
$a + \neg a \equiv 1$	BA-EXCL-MID
$a \cdot b \equiv b \cdot a$	BA-SEQ-COMM
$a \cdot \neg a \equiv 0$	BA-CONTRA
$a \cdot a \equiv a$	BA-SEQ-IDEM

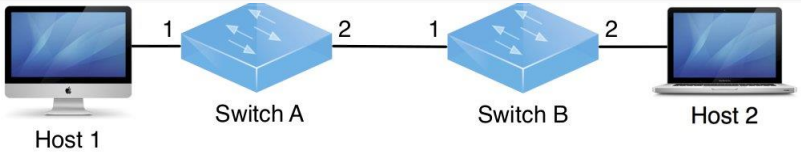
Packet Algebra Axioms

$f \leftarrow n \cdot f' \leftarrow n' \equiv f' \leftarrow n' \cdot f \leftarrow n$, if $f \neq f'$	PA-MOD-MOD-COMM
$f \leftarrow n \cdot f' = n' \equiv f' = n' \cdot f \leftarrow n$, if $f \neq f'$	PA-MOD-FILTER-COMM
$\mathsf{dup} \cdot f = n \equiv f = n \cdot \mathsf{dup}$	PA-DUP-FILTER-COMM
$f \leftarrow n \cdot f = n \equiv f \leftarrow n$	PA-MOD-FILTER
$f = n \cdot f \leftarrow n \equiv f = n$	PA-FILTER-MOD
$f \leftarrow n \cdot f \leftarrow n' \equiv f \leftarrow n'$	PA-MOD-MOD
$f = n \cdot f = n' \equiv 0$, if $n \neq n'$	PA-CONTRA
$\sum_i f = i \equiv 1$	PA-MATCH-ALL

Surface syntax:

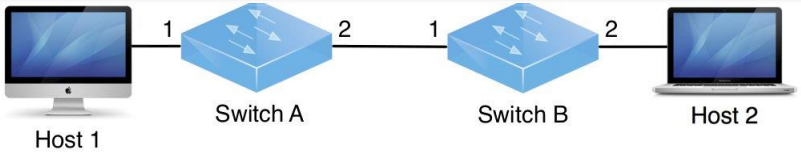
$a, b ::= 1$	`a,b ::= true`
$\mid 0$	`| false`
$\mid f = n$	`| f = n`
$\mid a + b$	`| a or b`
$\mid a \cdot b$	`| a and b`
$\mid \neg a$	`| not a`
$p, q ::= a$	`p,q ::= filter a`
$\mid f \leftarrow n$	`| f := n`
$\mid p + q$	`| p | q`
$\mid p \cdot q$	`| p ; q`
$\mid p^*$	`| p*`
$\mid$ dup	`| dup`

NetKAT: hands-on example



- Network elements:
  - 2 switch: A, B
  - Each switch has 2 ports: $1, 2$
  - 2 host: H_1, H_2
- Expected network behavior:
  - *Forwarding*: transfer packets between hosts
  - *Access control*: block SSH packets

NetKAT: network element description



- Switch-processing behavior

$$p \triangleq (dst = H_1 \cdot pt \leftarrow 1) + (dst = H_2 \cdot pt \leftarrow 2)$$

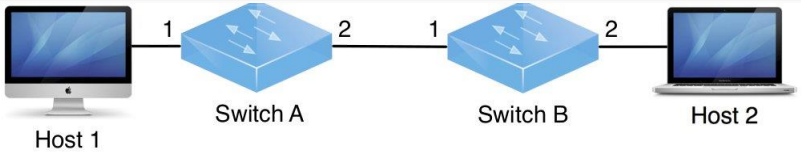
- Access control

$$p_{AC} \triangleq \neg(typ = SSH) \cdot p$$

- Split AC policies (no need to test all packets!)

$$p_A \triangleq (sw = A \cdot \neg(typ = SSH) \cdot p) + (sw = B \cdot p)$$
$$p_B \triangleq (sw = A \cdot p) + (sw = B \cdot \neg(typ = SSH) \cdot p)$$

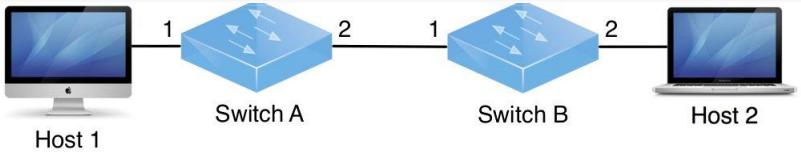
NetKAT: topology description



- Global network behavior (directed graph)
  - Links are uni-directional
- Network = union of behavior of each link
- Internal links

$$t = (sw = A \cdot pt = 2 \cdot sw \leftarrow B \cdot pt \leftarrow 1) +$$
$$(\mathrm{sw} = \mathrm{B} \cdot pt = 1 \cdot sw \leftarrow A \cdot pt \leftarrow 2) +$$
$$(sw = A \cdot pt = 1) +$$
$$(sw = B \cdot pt = 2)$$

NetKAT: overall network behavior



- Overall network behavior (switches meet topology)

$$p_{AC} \cdot t \cdot p_{AC}$$

- Arbitrary number of steps(even 0) ➔ Kleene star operator

$$(p_{AC} \cdot t)^*$$

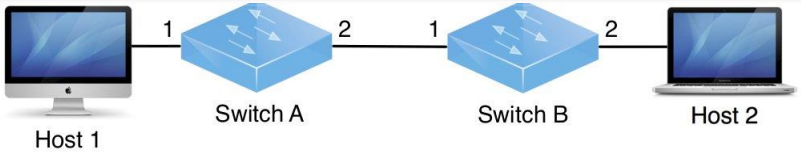
- Restrict packets that enter or exit the network

$$e \triangleq (sw = A \cdot pt = 1) + (sw = B \cdot pt = 2)$$

$$p_{net} \triangleq e \cdot (p_{AC} \cdot t)^* \cdot e$$

- In general: $in \cdot (p \cdot t)^* \cdot out$ (logical-crossbar / big-switch?)

NetKAT: formal reasoning



- "Are SSH packets dropped?"

  e.g. $(typ = SSH \cdot sw = A \cdot pt = 1 \cdot (p_{AC} \cdot t)^* \cdot sw = B \cdot pt = 2) \equiv 0$

- "Are non-SSH packets forwarded?"

  e.g. $(\neg(typ = SSH) \cdot sw = A \cdot pt = 1 \cdot sw \leftarrow B \cdot pt \leftarrow 2) \leq (p_{AC} \cdot t)^*$

- "Are p_{AC}, p_A and p_B equivalent?"
- "Is traffic form H_1 to H_2 routed through A?"

NetKAT: Summary

- Language & foundation model for SDN app programming
- Applications
  - SDN controller app verification & testing (compiler correctness)
  - Reasoning switch functionality (traffic isolation, reachability)
- SDN NBI verification with compilation
  - NetKAT → OpenFlow Normal Form (ONF) → OF rules
  - Proved compilation steps in NetKAT
- Implemented in *Ocaml* language
- Integrated into the Frenetic SDN controller framework

References

[1] Nick McKeown , Tom Anderson , Hari Balakrishnan , Guru Parulkar , Larry Peterson , Jennifer Rexford , Scott Shenker , Jonathan Turner, **OpenFlow: enabling innovation in campus networks**, *ACM SIGCOMM Computer Communication Review*, v.38 n.2, April 2008

[2] ONF SDN specification, https://www.opennetworking.org/software-defined-standards/specifications/

[3] Nate Foster , Rob Harrison , Michael J. Freedman , Christopher Monsanto , Jennifer Rexford , Alec Story , David Walker, **Frenetic: a network programming language**, *Proceedings of the 16th ACM SIGPLAN international conference on Functional programming*, September 19-21,2011, Tokyo, Japan

[4] The Frenetic family of languages, http://frenetic-lang.org/

[5] Carolyn Jane Anderson , Nate Foster , Arjun Guha , Jean-Baptiste Jeannin , Dexter Kozen , Cole Schlesinger , David Walker, **NetKAT: semantic foundations for networks**, *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, January 22-24, 2014, San Diego, California, USA

[6] Steffen Smolka , Spiridon Eliopoulos , Nate Foster , Arjun Guha, **A fast compiler for NetKAT**, *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming*, August 31-September 02, 2015, Vancouver, BC, Canada

Thank you for your kind attention!