wNetKAT: A Weighted SDN Programming and Verification Language

Kim G. Larsen, Stefan Schmid, Bingtian Xue

Aalborg University, DENMARK

OPODIS 2016 in Madrid, Spain
Computer Networks

- Computer networks (datacenter networks, enterprise networks, wide-area networks) have become a critical infrastructure of the information society.

- Concern: are today‘s networks dependable and flexible enough?
Even techsavvy companies struggle to provide reliable operations

We discovered a misconfiguration on this pair of switches that caused what's called a “bridge loop” in the network.

A network change was [...] executed incorrectly [...] more “stuck” volumes and added more requests to the re-mirroring storm

Service outage was due to a series of internal network events that corrupted router data tables

Experienced a network connectivity issue [...] interrupted the airline's flight departures, airport processing and reservations systems

Source: Talk by Nate Foster at DSDN Workshop
Another Anecdote: Wall–Street Bank

- Outage of a data center of a Wall Street investment bank
- Lost revenue measured in USD $10^6$ / min!
- Quickly, an emergency team was assembled with experts in compute, storage and networking:
  - **The compute team**: came armed with reams of logs, showing how and when the applications failed, and had already written experiments to reproduce and isolate the error, along with candidate prototype programs to workaround the failure.
  - **The storage team**: similarly equipped, showing which file system logs were affected, and already progressing with workaround programs.
  - And the **networking team**? Only had ping and traceroute.
Another Anecdote: Wall–Street Bank

“All the networking team had were two tools invented over twenty years ago to merely test end-to-end connectivity. Neither tool could reveal problems with the switches, the congestion experienced by individual packets, or provide any means to create experiments to identify, quarantine and resolve the problem. Whether or not the problem was in the network, the network team would be blamed since they were unable to demonstrate otherwise.”

Software-Defined Networks (SDNs) promise to introduce networking innovations, by decoupling the control plane from the data plane, and by making networks programmable and verifiable automatically.
Data plane:
Packet streaming

- Forward
- Filter
- Buffer
- Mark
- Rate-limit
- Measure packets
Control plane: Distributed
Distributed algorithms

Track topology changes
Compute routes
Install forwarding rules
Software Defined Networks (SDN)

Logically-centralized controller

API to the data plane (e.g., OpenFlow)
SDN/OpenFlow: Match–Action Devices

Logically–centralized controller

API to the data plane (e.g., OpenFlow)

OpenFlow Rule

Match   Action   Stats

Packet + byte counters

1. Forward packet to port(s)
2. Update header fields
1. Drop packet
......

Switch Port | MAC src | MAC dst | Eth type | VLAN ID | IP Src | IP Dst | IP Prot | TCP sport | TCP dport
---|---|---|---|---|---|---|---|---|---

OPODIS 2016
SDN/OpenFlow: Match–Action Devices

Logically-centralized controller

Matching Layer-2, Layer-3, Layer-4 **header fields** (e.g., IP destination, TCP port, etc.)

OpenFlow Rule

Match | Action | Stats

Packet + byte counters

1. Forward packet to port(s)
2. Update header fields
1. Drop packet

Switch Port | MAC src | MAC dst | Eth type | VLAN ID | IP Src | IP Dst | IP Prot | TCP sport | TCP dport

1. Forward packet to port(s)
2. Update header fields
1. Drop packet

Packet + byte counters

Switch Port | MAC src | MAC dst | Eth type | VLAN ID | IP Src | IP Dst | IP Prot | TCP sport | TCP dport

1. Forward packet to port(s)
2. Update header fields
1. Drop packet

Packet + byte counters
**SDN/OpenFlow: Match–Action Devices**

Logically-centralized controller

E.g., update header field (new MAC destination), forward, drop...

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
<th>Stats</th>
</tr>
</thead>
</table>

Packet + byte counters

1. Forward packet to port(s)
2. Update header fields
1. Drop packet
......

Switch Port | MAC src | MAC dst | Eth type | VLAN ID | IP Src | IP Dst | IP Prot | TCP sport | TCP dport |
-------------|---------|---------|----------|---------|--------|--------|---------|----------|----------|
Logically-centralized controller

Not important right now. But trend: more stateful switches (e.g., load-balance, “stateful” firewall...)

1. Forward packet to port(s)
2. Update header fields
1. Drop packet
   ......

OpenFlow Rule

Packet + byte counters

Switch Port | MAC src | MAC dst | Eth type | VLAN ID | IP Src | IP Dst | IP Prot | TCP sport | TCP dport
--- | --- | --- | --- | --- | --- | --- | --- | --- | ---
Benefits of SDN

**Benefit 1:** Decoupling! Control plane can evolve independently of data plane: innovation at speed of software development. And simpler network management through logically centralized view: many network management and operational tasks are inherently non-local.

**Benefit 2:** Simple match-action devices: supports formal verification! Switches/ routers are “simple” and “passive” match-action devices. (Unlike in, e.g., active networks.) Can do, e.g., header space analysis.
Solution: *Software-Defined Networks (SDN)*!

- SDNs are popular: deployments in enterprises, datacenters, WAN, IXPs
- E.g., Google: “A Purpose-Built Global Network: Google’s Move to SDN”
- SDN is about programming the networks

- But OpenFlow is very low-level: inconvenient for programmers

- Hence, over the last years, many network specific programming languages have been developed

- Researchers have started developing more high-level languages

- **NetKAT**: state-of-the-art framework for programming and reasoning about networks
NetKAT: Kleene Algebra with Tests (KAT) with atoms like:

- \( f \leftarrow w \) assignment
- \( f = w \) test

Assigning values to header fields

Testing values in header fields
NetKAT: No need to understand all details now 😊

**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 \( a, b ::= 1 \quad \text{Identity} \)

\( 0 \quad \text{Drop} \)

\( f = n \quad \text{Test} \)

\( a + b \quad \text{Disjunction} \)

\( a \cdot b \quad \text{Conjunction} \)

\( \neg a \quad \text{Negation} \)

Policies \( p, q ::= a \quad \text{Filter} \)

\( f \leftarrow n \quad \text{Modification} \)

\( p + q \quad \text{Union} \)

\( p \cdot q \quad \text{Sequential composition} \)

\( p^* \quad \text{Kleene star} \)

\( \text{dup} \quad \text{Duplication} \)

**Semantics**

\([p] \in H \rightarrow \mathcal{P}(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\} \cdot 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] \cdot [q]) h\]

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

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

\([\text{dup}] (pk::h) \triangleq \{pk::(pk::h)\}\]
**Important concept 1:**
Packets with a set of fields

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fields</strong></td>
<td>[ p \in H \rightarrow \mathcal{P}(H) ]</td>
</tr>
<tr>
<td>( f ::= f_1</td>
<td>\cdots</td>
</tr>
<tr>
<td><strong>Packets</strong></td>
<td>([0] \ h \triangleq { } )</td>
</tr>
<tr>
<td>( pk ::= { f_1 = v_1, \cdots, f_k = v_k } )</td>
<td>([f = n] (pk::h) \triangleq { pk[f := n]::h } ) if ( pk.f = n ) otherwise</td>
</tr>
<tr>
<td><strong>Histories</strong></td>
<td>([-a] \ h \triangleq { h } \setminus ([a] \ h) )</td>
</tr>
<tr>
<td>( h ::= pk::h )</td>
<td>([f \leftarrow n] (pk::h) \triangleq { pk[f := n]::h } )</td>
</tr>
<tr>
<td><strong>Predicates</strong></td>
<td>([p + q] \ h \triangleq [p] \ h \cup [q] \ h )</td>
</tr>
<tr>
<td>( a, b ::= 1 )</td>
<td>([p \cdot q] \ h \triangleq ([p] \bullet [q]) \ h )</td>
</tr>
<tr>
<td>Identity</td>
<td>([p^*] \ h \triangleq \bigcup_{i \in \mathbb{N}} F^i \ h ) \quad \text{where} ( F^0 \ h \triangleq { h } ) and ( F^{i+1} \ h \triangleq ([p] \bullet F^i) \ h )</td>
</tr>
<tr>
<td>Drop</td>
<td>([p\cdot q] \ h \triangleq ([p] \bullet [q]) \ h )</td>
</tr>
<tr>
<td>Test</td>
<td>([p\cdot q] \ h \triangleq ([p] \bullet [q]) \ h )</td>
</tr>
<tr>
<td>Disjunction</td>
<td>([p\cdot q] \ h \triangleq ([p] \bullet [q]) \ h )</td>
</tr>
<tr>
<td>Conjunction</td>
<td>([p\cdot q] \ h \triangleq ([p] \bullet [q]) \ h )</td>
</tr>
<tr>
<td>Negation</td>
<td>([p\cdot q] \ h \triangleq ([p] \bullet [q]) \ h )</td>
</tr>
<tr>
<td><strong>Policies</strong></td>
<td>([p\cdot q] \ h \triangleq ([p] \bullet [q]) \ h )</td>
</tr>
<tr>
<td>( p, q ::= a )</td>
<td>([p\cdot q] \ h \triangleq ([p] \bullet [q]) \ h )</td>
</tr>
<tr>
<td>Filter</td>
<td>([p\cdot q] \ h \triangleq ([p] \bullet [q]) \ h )</td>
</tr>
<tr>
<td>Modification</td>
<td>([p\cdot q] \ h \triangleq ([p] \bullet [q]) \ h )</td>
</tr>
<tr>
<td>Union</td>
<td>([p\cdot q] \ h \triangleq ([p] \bullet [q]) \ h )</td>
</tr>
<tr>
<td>Sequential composition</td>
<td>([p\cdot q] \ h \triangleq ([p] \bullet [q]) \ h )</td>
</tr>
<tr>
<td>Kleene star</td>
<td>([p\cdot q] \ h \triangleq ([p] \bullet [q]) \ h )</td>
</tr>
<tr>
<td>Duplication</td>
<td>([p\cdot q] \ h \triangleq ([p] \bullet [q]) \ h )</td>
</tr>
</tbody>
</table>
NetKAT: No need to understand all details now 😊

Important concept 2: history. We maintain packet history that records the state of each packet as it travels from switch to switch.
NetKAT: No need to understand all details now 😊

Policies includes, e.g., modification of header field.

We always work on the current packet, at front of history. (History: just to keep track of trajectory.)
NetKAT: No need to understand all details now 😊

Predicates includes, e.g., test of header field.

Producing an empty history = dropping the packet.
Producing a singleton = forwarding to a single port.
NetKAT: Example

Can model the topology as the union of smaller policies that encode the behavior of each link.
Can model the topology as the union of smaller policies that encode the behavior of each link.

\[ t ::= \text{sw} = s; (\text{sw} \leftarrow F_1 + \text{sw} \leftarrow v) \]
\[ +\text{sw} = F_1; (\text{sw} \leftarrow F_2^{(1)} + \text{sw} \leftarrow F_2^{(2)}) \]
\[ +\text{sw} = v; (\text{sw} \leftarrow F_1^{(1)} + \text{sw} \leftarrow F_2^{(2)}) \]
\[ +\text{sw} = F_2^{(1)}; \text{sw} \leftarrow t \]
\[ +\text{sw} = F_2^{(2)}; \text{sw} \leftarrow t \]
NetKAT: Use Cases

NetKAT allows to answer many important questions:

- “Can X connect to Y?”
- “Is traffic from A to B routed through Z?”
- “Is there a loop involving S?”
- “Are non-SSH packets forwarded?”
- Etc.
NetKAT: Use Cases

NetKAT allows to answer many important questions:

- “Can X connect to Y?”
- “Is traffic from A to B routed through Z?”
- “Is there a loop involving S?”
- “Are non-SSH packets forwarded?”
- Etc.

However, NetKAT is limited to binary contexts. What is missing today is a framework to reason about the inherent weighted aspects of networking: wNetKAT.
Real networks are **weighted**: links have costs (latency, energy, peering costs, etc.) and are capacitated (e.g., bandwidth). But also **nodes** may have capacity constraints or entail costs. Moreover, nodes may transform the traffic volume (e.g., add or remove encapsulation headers or compress packets).
Real networks are **weighted** links have costs (latency, energy, peering costs, etc.) and are capacitated (e.g., bandwidth). But also **nodes** may have capacity constraints or entail costs. Moreover, nodes may transform the traffic volume (e.g., add or remove encapsulation headers or compress packets).

**The Case for Weighted NetKAT!**
wNetKAT: Challenges

The weighted extension of NetKAT is non-trivial:

- capacity constraints introduce dependencies between flows (e.g., packets compete for bandwidth)
- we need arithmetic operations such as addition (e.g., in case of latency to compute the end-to-end delay) or minimum (e.g., in case of bandwidth)

Therefore, we extend the syntax of NetKAT toward weighted packet- and switch-variables, as well as queues, and provide a semantics accordingly.
Paper Contributions

- We show for which weighted aspects and use cases which language extensions are required.

- We show the relation between WNetKAT expressions and weighted finite automata: an important operational model for weighted programs.
  - This also leads to the undeciability of WNetKAT equivalence problem.

- We explore the complexity of verification more generally and for subsets of the language
  - We prove the decidability of whether an expression equals 0 (emptiness testing): for many practical scenarios a sufficient and relevant problem (e.g., reachability)
We add two types of variables to NetKAT:

- quantitative packet variables!
- (quantitative, non-quantitative) switch variables

Accordingly, we generalize assignment and test, to also include arithmetic operations (namely addition):

- Quantitative Assignment
- Quantitative Test
wNetKAT: Additions to NetKAT

We add two types of variables to NetKAT:

- quantitative packet variables!
- (quantitative, non-quantitative) switch variables

Accordingly, we generalize assignment and test, to also include arithmetic operations (namely addition):

- Quantitative Assignment
- Quantitative Test

Also allows us to model more stateful switches (as they are currently underway)
wNetKAT: Another no–go slide! 😊

\[
\begin{align*}
\llbracket x \leftarrow \omega \rrbracket(\rho, pk :: h) &= \begin{cases} 
\{\rho, pk[\omega/x] :: h\} & \text{if } x \in \mathcal{V}_p \\
\{\rho(v)[\omega/x], pk :: h\} & \text{if } x \in \mathcal{V}_s \text{ and } pk(sw) = v
\end{cases} \\
\llbracket x = \omega \rrbracket(\rho, pk :: h) &= \begin{cases} 
\{\rho, pk :: h\} & \text{if } x \in \mathcal{V}_p \text{ and } pk(x) = \omega \\
\emptyset & \text{or if } x \in \mathcal{V}_s, pk(sw) = v \text{ and } \rho(v, x) = \omega \\
& \text{otherwise}
\end{cases}
\end{align*}
\]

\[
\begin{align*}
\llbracket y \leftarrow (\Sigma_{y' \in \mathcal{V}} y' + r) \rrbracket(\rho, pk :: h) &= \begin{cases} 
\{\rho, pk[r'/x] :: h\} & \text{if } x \in \mathcal{V}_p \\
\{\rho(v)[r'/x], pk :: h\} & \text{if } x \in \mathcal{V}_s \text{ and } pk(sw) = v
\end{cases}
\end{align*}
\]

where \( r' = \Sigma_{y_p \in \mathcal{V}_p} pk(y_p) + \Sigma_{y_s \in \mathcal{V}_s} \rho(v, y_s) + r \)

\[
\begin{align*}
\llbracket y = (\Sigma_{y' \in \mathcal{V}} y' + r) \rrbracket(\rho, pk :: h) &= \begin{cases} 
\{\rho, pk :: h\} & \text{if } x \in \mathcal{V}_p \text{ and } pk(x) = r' \\
\emptyset & \text{or } x \in \mathcal{V}_s, pk(sw) = v \text{ and } \rho(v, x) = r' \\
& \text{otherwise}
\end{cases}
\end{align*}
\]

where \( r' = \Sigma_{y_p \in \mathcal{V}_p} pk(y_p) + \Sigma_{y_s \in \mathcal{V}_s} \rho(v, y_s) + r \)

\[x \in \mathcal{V}_n, y \in \mathcal{V}_q\]
Quantitative update: update the corresponding header field if $x$ is a packet-variable, or update the corresponding switch information of the current switch if $x$ is a switch-variable.

$$[[y \leftarrow (\Sigma_{y \in \mathcal{V}} y' + r)]](\rho, pk :: h) = \begin{cases} \{\rho, pk[r'/x] :: h\} & \text{if } x \in \mathcal{V}_p \\ \{\rho(v)[r'/x], pk :: h\} & \text{if } x \in \mathcal{V}_s \text{ and } pk(sw) = v \end{cases}$$

where $r' = \Sigma_{y_p \in \mathcal{V}_p \cap \mathcal{V}_p} pk(y_p) + \Sigma_{y_s \in \mathcal{V}_s \cap \mathcal{V}_q} \rho(v, y_s) + r$

$$[[y = (\Sigma_{y \in \mathcal{V}} y' + r)]](\rho, pk :: h) = \begin{cases} \{\rho, pk :: h\} & \text{if } x \in \mathcal{V}_p \text{ and } pk(x) = r' \\ \emptyset & \text{or } x \in \mathcal{V}_s, pk(sw) = v \text{ and } \rho(v, x) = r' \\ \emptyset & \text{otherwise} \end{cases}$$

where $r' = \Sigma_{y_p \in \mathcal{V}_p \cap \mathcal{V}_p} pk(y_p) + \Sigma_{y_s \in \mathcal{V}_s \cap \mathcal{V}_q} \rho(v, y_s) + r$

$$x \in \mathcal{V}_n, y \in \mathcal{V}_q$$
wNetKAT: Another no–go slide! 😊

\[
\llbracket x \leftarrow \omega \rrbracket (\rho, \ pk :: h) = \begin{cases} 
\{\rho, \ pk[\omega/x] :: h\} & \text{if } x \in \mathcal{V}_p \\
\{\rho(v)[\omega/x], \ pk :: h\} & \text{if } x \in \mathcal{V}_s \text{ and } pk(sw) = v 
\end{cases}
\]

\[
\llbracket x = \omega \rrbracket (\rho, \ pk :: h) = \begin{cases} 
\{\rho, \ pk :: h\} & \text{if } x \in \mathcal{V}_p \text{ and } pk(x) = \omega \\
\text{or if } x \in \mathcal{V}_s, \ pk(sw) = v \text{ and } \rho(v, x) = \omega & \text{otherwise} 
\end{cases}
\]

\[
\llbracket y \leftarrow (\sum_{y' \in \mathcal{V}_p} y' + r) \rrbracket (\rho, \ pk :: h) = \begin{cases} 
\{\rho, \ pk[r'/x] :: h\} & \text{if } x \in \mathcal{V}_p \\
\{\rho(v)[r'/x], \ pk :: h\} & \text{if } x \in \mathcal{V}_s \text{ and } pk(sw) = v 
\end{cases}
\]

where \( r' = \sum_{y_p \in \mathcal{V}_p} pk(y_p) + \sum_{y_s \in \mathcal{V}_s} \rho(v, y_s) + r \)

\[
\llbracket y = (\sum_{y' \in \mathcal{V}_p} y' + r) \rrbracket (\rho, \ pk :: h) = \begin{cases} 
\{\rho, \ pk :: h\} & \text{if } x \in \mathcal{V}_p \text{ and } pk(x) = r' \\
\text{or } x \in \mathcal{V}_s, \ pk(sw) = v \text{ and } \rho(v, x) = r' & \text{otherwise} 
\end{cases}
\]

where \( r' = \sum_{y_p \in \mathcal{V}_p} pk(y_p) + \sum_{y_s \in \mathcal{V}_s} \rho(v, y_s) + r \)

Test the quantitative variables using the current packet– and switch–variables.
- We only support addition

- But we can also support other arithmetic operations
  - min or max can be easily defined (see paper):

\[
x \leftarrow \min\{y, z\} \overset{\text{def}}{=} y \leq z; x \leftarrow y \land y > z; x \leftarrow z.
\]
Example: Characterizing Weighted Topologies

Can model the topology as the union of smaller policies that encode the behavior of each link.
Can model the topology as the union of smaller policies that encode the behavior of each link.

t ::= sw = s; (sw ← F₁; co ← co + 1; ca ← min{ca, 8} & sw ← v; co ← co + 5; ca ← min{ca, 2}) & sw = F₁;
    (sw ← F₁; co ← co + 3; ca ← min{ca, 1} & sw ← F₂; co ← co + 2; ca ← min{ca, 10}) & sw = v; (sw ← F₁; co ← co + 3; ca ← min{ca, 3} & sw ← F₂; co ← co + 2; ca ← min{ca, 1}) & sw = F₂; sw ← t; co ← co + 6; ca ← min{ca, 1} & sw = F₂; sw ← t; co ← co + 1; ca ← min{ca, 4}
Example: Characterizing Weighted Topologies

Can model the topology as the union of smaller policies that encode the behavior of each link.

At s, can either go to F1 or v. Update cost and min bandwidth accordingly.

\[ t ::= \begin{align*}
sw &= s; (sw \leftarrow F_1; co \leftarrow co + 1; ca \leftarrow \min\{ca, 8\}) \\
&\quad \& sw \leftarrow v; (co \leftarrow co + 5; ca \leftarrow \min\{ca, 2\}) \\
&\quad \& sw = F_1; \\
&\quad \quad (sw \leftarrow F_1^{(1)}; co \leftarrow co + 3; ca \leftarrow \min\{ca, 1\}) \\
&\quad \quad \& sw \leftarrow F_1^{(2)}; (co \leftarrow co + 2; ca \leftarrow \min\{ca, 10\}) \\
&\quad \& sw = v; (sw \leftarrow F_2^{(1)}; co \leftarrow co + 3; ca \leftarrow \min\{ca, 3\}) \\
&\quad \quad \& sw \leftarrow F_2^{(2)}; (co \leftarrow co + 2; ca \leftarrow \min\{ca, 1\}) \\
&\quad \& sw = F_2^{(1)}; sw \leftarrow t; (co \leftarrow co + 6; ca \leftarrow \min\{ca, 1\}) \\
&\quad \& sw = F_2^{(2)}; sw \leftarrow t; (co \leftarrow co + 1; ca \leftarrow \min\{ca, 4\})
\end{align*} \]
Function $F_2$ increases the flow rate by an additive constant

\[ p_{F_2} := (sw = F_2^{(1)} \& sw = F_2^{(2)}); ca \leftarrow ca + \gamma \]
Applications: Cost Reachability

- “Can node B be reached from A at cost at most c?”

Topology: Google B4 Wide-Area Inter-datacenter connect:
Applications: Capacitated Reachability

- "Can node A communicate with B at rate at least r?"

- Unsplittable

- Splittable
Applications: Service Chain

- “Can node A reach B at cost/latency at most $l$ and/or at rate/bandwidth at least $r$, via $F1\rightarrow F2$?”

- Check whether the following is equal to "drop"

\[
\text{src} \leftarrow s; \text{dst} \leftarrow t; \text{co} \leftarrow 0; \text{ca} \leftarrow r; \text{sw} \leftarrow s; \text{pt}(pt)^*; \text{sw} = F_1; p_{F_1}; t(pt)^*; \text{sw} = F_2; p_{F_2}; t(pt)^*; \text{sw} = t; \text{co} \leq l; \text{ca} \geq r.
\]
Applications: Fairness

- “Does the current flow allocation satisfy max-min fairness requirements?”
(Un)Decidability

- Theorem: (undecidability)

Deciding equivalence of two WNetKAT expressions is equal to deciding the equivalence of the two corresponding weighted WNetKAT automata.

- Theorem:

Deciding whether a WNetKAT expression is equal to "drop" is equal to deciding the emptiness of the corresponding weighted automaton.
Questions?

Thank you for your attention!