wNetKAT: Programming and Verifying Software–Defined Networks

Bingtian Xue
Aalborg University, DENMARK

Joint work with
Kim G. Larsen, Stefan Schmid

NWPT, Nov 1, 2016
Computer Networks

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

- For the future: dependable and flexible enough?
Traditional Networks
Traditional Networks: Data Plane

Data plane:
Packet streaming

- Forward
- Filter
- Buffer
- Mark
- Rate-limit
- Measure packets
Traditional Networks: (Distributed) Control Plane

**Control plane:**
Distributed algorithms

- Track topology changes
- Compute routes
- Install forwarding rules
Management plane:
Humans

Collect measurements
Configure the equipment
Traditional Networks

- Dependable and flexible enough?

**NO!**

Example: a link failure!
Solution: *Software-Defined Networks (SDN)*!

- “A Purpose-Built Global Network: Google‘s Move to SDN”
  - One of the key reasons for Google’s move to SDN: *faster and more efficient* /fine-grained traffic engineering, improving WAN network utilization.
  - A second reason: *more predictable behavior* under failures (no unpredictable and slow distributed reconvergence).
Software Defined Networks (SDN)

Logically-centralized controller

API to the data plane (e.g., OpenFlow)
Software Defined Networks (SDN)

Logically-centralized controller

Benefit 1: Decoupling! Control plane can evolve independently of data plane: innovation at speed of software development.

Benefit 2: Simpler network management through logically centralized view.

Benefit 3: It is all about generalization!
Software Defined Networks (SDN)

Benefit 2: Simpler network management through logically centralized view: many network management and operational tasks are inherently non-local.
**Software Defined Networks (SDN)**

**Logically-centralized controller**

**Benefit 3: OpenFlow is about generalization!**
- Generalize devices (L2-L4: switches, routers, middleboxes)
- Generalize routing and traffic engineering (not only destination-based)
- Generalize flow-installation: coarse-grained rules and wildcards okay, proactive vs reactive installation
- Provide general and logical network views to the application / tenant
Software Defined Networks (SDN)

Logically-centralized controller

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

Benefit 4: It supports **formal verification!**
Programming SDN

- Programming languages for networks lie at the heart of SDN
Programming SDN

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. Encapsulate and forward to controller
3. Drop packet
4. Send to normal processing pipeline

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

- Programming languages for networks lie at the heart of SDN

- OpenFlow is very low-level: inconvenient for programmers

- Researchers have started developing more high-level languages

- NetKAT: state-of-the-art framework for programming and reasoning about networks
NetKAT

- Kleene Algebra (KA) \((K, +, \cdot, *, 0, 1)\)
  - +, \cdot binary operators
  - * unary operator
  - 0,1 constants
- KAT \((K, B, +, \cdot, *, \bar{}, 0, 1), B \subseteq K\)
  - \((K, +, \cdot, *, 0, 1)\) KA
  - \((B, +, \cdot, \bar{}, 0, 1)\) Boolean Algebra
  - \((B, +, \cdot, 0, 1)\) subalgebra of \((K, +, \cdot, 0, 1)\)
- NetKAT: KAT with the following atoms
  - \(f \leftarrow w\) assignment
  - \(f = w\) test
  - \textit{dup} duplication
NetKAT

Syntax

Fields \( f ::= f_1 \mid \cdots \mid f_k \)

Packets \( pk ::= \{f_1 = v_1, \cdots, f_k = v_k\} \)

Histories \( h ::= \emptyset | pk::h \)

Predicates \( a, b ::= 0 \quad \text{Identity} \)
\[ \begin{align*}
0 & \quad \text{Drop} \\
\neg a & \quad \text{Negation} \\
a + b & \quad \text{Disjunction} \\
a \cdot b & \quad \text{Conjunction} \\
\end{align*} \]

Policies \( p, q ::= a \quad \text{Filter} \)
\[ \begin{align*}
f & \leftarrow n \quad \text{Modification} \\
p + q & \quad \text{Union} \\
p \cdot q & \quad \text{Sequential composition} \\
p^* & \quad \text{Kleene star} \\
dup & \quad \text{Duplication} \\
\end{align*} \]

Semantics

\[ [p] \in H \rightarrow \mathcal{P}(H) \]
\[ [0] h \triangleq \emptyset \]
\[ [1] h \triangleq \{h\} \]
\[ [\neg a] h \triangleq \{h\} \setminus ([a] h) \]
\[ [a \cdot b] h \triangleq \{pk::h\} \text{ if } pk.f = n \]
\[ [a + b] h \triangleq \{pk::h\} \text{ if } pk.f = n \]
\[ [f = n] (pk::h) \triangleq \begin{cases} \{pk::h\} & \text{if } pk.f = n \\ \{\} & \text{otherwise} \end{cases} \]
\[ [f \leftarrow n] (pk::h) \triangleq \{pk[f := n]::h\} \]
\[ [p + q] h \triangleq [p] h \cup [q] h \]
\[ [p \cdot q] h \triangleq ([p] \bullet [q]) h \]
\[ [p^*] h \triangleq \bigcup_{i \in \mathbb{N}} F_i h \]

where \( F^0 h \triangleq \{h\} \) and \( F^{i+1} h \triangleq ([p] \bullet F^i) h \)
\[ [\text{dup}] (pk::h) \triangleq \{pk::(pk::h)\} \]

‘;’ instead of ‘.’ in policies!
\[ t \quad ::= \quad sw = s; (sw \leftarrow F_1 + sw \leftarrow v) \\
+ sw = F_1; (sw \leftarrow F_2^{(1)} + sw \leftarrow F_2^{(2)}) \\
+ sw = v; (sw \leftarrow F_1^{(1)} + sw \leftarrow F_2^{(2)}) \\
+ sw = F_2^{(1)}; sw \leftarrow t \\
+ sw = F_2^{(2)}; sw \leftarrow t \]
NetKAT: Example

Weighted NetKAT!
wNetKAT

- Add quantitative variables!
- Add switch variables (quantitative, non-quantitative)
- Quantitative Assignment
  \[ x \leftarrow (\sum_{x' \in \mathcal{V}'x'} + \delta) \]
- Quantitative Test
  \[ x \bowtie (\sum_{x' \in \mathcal{V}'x'} + \delta) \]
  where: \( x \in \mathcal{V}_q, \mathcal{V}' \subseteq \mathcal{V}_q, \bowtie \in \{>, <, \geq, \leq, =\}, \delta \in \mathbb{N} \) (or \( \mathbb{Q} \))

\( x \) might be from \( \mathcal{V}' \)

\( \mathcal{V}_q \) (\( \mathcal{V}_n \)) can be either packet or switch!

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

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

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

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

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

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

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

\[ x \in \mathcal{V}_n, \, y \in \mathcal{V}_q, \]
Other arithmetic operations

- min or max can be easily defined:
  \[ x \leftarrow \text{min}\{y, z\} \overset{\text{def}}{=} y \leq z; x \leftarrow y \& y > z; x \leftarrow z. \]
Example

\[ t ::= \begin{align*}
&sw = s; (sw \leftarrow F_1; co \leftarrow co + 1; ca \leftarrow \min\{ca, 8\} \\
&\quad \& sw \leftarrow v; co \leftarrow co + 5; ca \leftarrow \min\{ca, 2\}) \\
&\quad \& sw = F_1; \\
&\quad \quad (sw \leftarrow F_2^{(1)}; co \leftarrow co + 3; ca \leftarrow \min\{ca, 1\}) \\
&\quad \quad \& sw \leftarrow F_2^{(2)}; co \leftarrow co + 2; ca \leftarrow \min\{ca, 10\}) \\
&\quad \& sw = v; (sw \leftarrow F_2^{(1)}; co \leftarrow co + 3; ca \leftarrow \min\{ca, 3\}) \\
&\quad \quad \& sw \leftarrow F_2^{(2)}; co \leftarrow co + 2; ca \leftarrow \min\{ca, 1\}) \\
&\quad \& sw = F_2^{(1)}; sw \leftarrow t; co \leftarrow co + 6; ca \leftarrow \min\{ca, 1\} \\
&\quad \& sw = F_2^{(2)}; sw \leftarrow t; co \leftarrow co + 1; ca \leftarrow \min\{ca, 4\} 
\end{align*} \]
Example

$p_{F_2} ::= (sw = F_2^{(1)} \& sw = F_2^{(2)}); \text{ca} \leftarrow \text{ca} + \gamma$

Only packet variables in this example
Applications: Cost Reachability

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

- **Topology 't'**
  
  e.g., $dc_1 \rightarrow dc_2$
  
  $sw = dc_1; pt = 1; sw \leftarrow dc_2; pt \leftarrow 4; l \leftarrow l + 4$

- **Policy 'p'**
  
  e.g., $dc_2$
  
  $src = (pt) \leftarrow (p)$

- Check whether the following equal to "drop"
  
  $scr \leftarrow A; dst \leftarrow B; l \leftarrow 0; sw \leftarrow A;
  
  $pt(pt)^*;
  
  $sw = B; l \leq c.$

Also works for multi-weights!
Applications: Capacitated Reachability

- “Can node A communicate with B at rate at least r?”
  - Unsplittable
  - Splittable
Applications: Capacitated Reachability

- “Can node A communicate with B at rate at least r?”
- **Unsplittable**
  - Topology 't'
    e.g., $dc_1 \rightarrow dc_2$
    $sw = dc_1; pt = 1$
    $sw \leftarrow dc_2; pt \leftarrow 4; c \leftarrow \min\{c, 4\}$
  - Policy 'p'
    e.g., $dc_2$
    $src = dc_1; dst = dc_5; sw = dc_2; pt = 4$;
    ($pt \leftarrow 1 \& pt \leftarrow 3$)
  - Check whether the following equal to "drop"
    $scr \leftarrow A; dst \leftarrow B; c \leftarrow r; sw \leftarrow A$
    $pt(pt)^*$;
    $sw = B; c \geq r$. 

NWPT, Nov 1, 2016
“Can node A communicate with B at rate at least r?”

- **Splittable**
  - Topology 't'
    - e.g., \( dc_1 \rightarrow dc_2 \)
    - \( sw = dc_1; pt = 1; \)
    - \( sw \leftarrow dc_2; pt \leftarrow 4; c \leftarrow \min\{c, 4\} \)
Applications: Capacitated Reachability

- “Can node A communicate with B at rate at least r?”
- Splittable
  - Policy ‘p’
  - Split: e.g., $dc_2$

$sr = dc_1; dst = dc_5; sw = dc_2; pt = 4; c \leq 5$;

$$((pt \leftarrow 1; c \leftarrow \min\{c, 3\})$$

$$& (pt \leftarrow 3; c \leftarrow \max\{0, c - 3\}))$$
Applications: Capacitated Reachability

- “Can node A communicate with B at rate at least r?”
- Splittable
  - Policy 'p'
  - Split
    - Merge: e.g., $dc_4$
      
      
      \[
      sw = dc_4; src = dc_1; dst = dc_5;
      (pt = 1 & pt = 2);
      C = C + c; X ← X − 1;
      (X ≠ 0; drop
      & X = 0; c ← C; pt ← 3).
      \]

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
<th>in</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>dc_1</td>
<td>dc_5</td>
<td>1, 2</td>
<td>3</td>
</tr>
<tr>
<td>dc_5</td>
<td>dc_2</td>
<td>3, 4</td>
<td>1, 2</td>
</tr>
</tbody>
</table>

NWPT, Nov 1, 2016
Applications: Capacitated Reachability

- “Can node A communicate with B at rate at least r?”
- Splittable
  - Policy 'p'
    \[ sw = dc_4; src = dc_5; dst = dc_2; \]
    \[ (pt = 3 & pt = 4); \]
    \[ C = C + c; X \leftarrow X - 1; \]
    \[ (X \neq 0; drop) \]
    & \[ X = 0; c \leftarrow C; c \leq 8 \]
    \[ ((pt \leftarrow 1; c \leftarrow \min\{6, c\}) \]
    & \[ (pt \leftarrow 2; c \leftarrow \max\{0, c - 6\})). \]

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
<th>in</th>
<th>out</th>
<th>C</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>dc_1</td>
<td>dc_5</td>
<td>1,2</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>dc_5</td>
<td>dc_2</td>
<td>3,4</td>
<td>1,2</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
Applications: Capacitated Reachability

- “Can node A communicate with B at rate at least r?”
- **Splittable**
  - Check whether the following equal to "drop"
  
  \[
  scr \leftarrow A; \quad dst \leftarrow B; \quad c \leftarrow r; \quad sw \leftarrow A; \\
  pt(pt)^*; \\
  sw = B; \quad c \geq r.
  \]
Applications: Service Chain

- “Can node A reach B at cost/latency at most l and/or at rate/bandwidth at least r, via the service chain?”

- Check whether the following equal to "drop"

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

- “Does the current flow allocation satisfy max–min fairness requirements?”
Applications: Fairness

“Does the current flow allocation satisfy max-min fairness requirements?”

\[
f_3 = sw \leftarrow s_2; scr \leftarrow s_2; dst \leftarrow d_2; a \leftarrow 10;
\]
\[
x_3 \leftarrow 1; x_1 \leftarrow 0; x_2 \leftarrow 0; tp(tp)^*; sw = d_2; x_3 = a
\]

\[
f_1 = sw \leftarrow s_1; scr \leftarrow s_1; dst \leftarrow d_1; a \leftarrow 10;
\]
\[
x_1 \leftarrow 2; x_3 \leftarrow 1; x_2 \leftarrow 0; tp(tp)^*; sw = d_1; x_1 = a
\]

\[
f_2 = sw \leftarrow s_1; scr \leftarrow s_1; dst \leftarrow d_2; a \leftarrow 10;
\]
\[
x_2 \leftarrow 3; x_1 \leftarrow 2; x_3 \leftarrow 1; tp(tp)^*; sw = d_1; x_2 = a
\]
Applications: Quality of Service

- E.g., prioritize a certain flow (e.g., a VoIP call) over another (e.g., a Dropbox synchronization).

\[
sw = r; (pt = 1 \& pt = 2);
(x = high; C_h < 8; pt \leftarrow 3; C_h \leftarrow C_h + 1; \& x = low; C_l < 2; pt \leftarrow 3; C_l \leftarrow C_l + 1);
C_h = 8; C_l = 2; C_h \leftarrow 0; C_l \leftarrow 0
\]
Further Extension: Queues

\[
\llbracket EQ \; Q \rrbracket(\rho, \; pk :: h) = \begin{cases} 
\llbracket 1^* \rrbracket(\rho, \; pk :: h) & \text{if } Q \neq FULL, \\
\emptyset & \text{otherwise}
\end{cases}
\]

\[
\llbracket DQ \; Q \rrbracket(\rho, \; pk :: h) = \begin{cases} 
\{\rho, \; pk :: h\} & \text{if } HEAD(Q) = pk :: h \\
\llbracket 1^* \rrbracket(\rho, \; pk :: h) & \text{otherwise}
\end{cases}
\]

NWPT, Nov 1, 2016
Applications: Quality of Service

- E.g., prioritize a certain flow (e.g., a VoIP call) over another (e.g., a Dropbox synchronization).

\[
\begin{align*}
sw &= r; x = low; EQ Q_l & \& sw = r; x = high; \\
& (x_h < 8; DQ Q_h; pt \leftarrow 3; x_h \leftarrow x_h + 1 \\
& \& x_h = 8; Q_l = \emptyset; DQ Q_h; pt \leftarrow 3 \\
& \& x_h = 8; Q_l \neq \emptyset; skip) \\
x_h &= 8; x_l = 2; x_h \leftarrow 0; x_l \leftarrow 0
\end{align*}
\]
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!