APKeep: Realtime Verification for Real Networks

Peng Zhang*, Xu Liu*, Hongkun Yang+, Ning Kang*, Zhengchang Gu*, Hao Li*

*Xi’an Jiaotong University, +Google
Background

Network outages are common
human misconfiguration, software bugs, etc.

Post-effect troubleshooting is slow
manually find the root cause after outages using simple tools

The cost can be quite expensive
service down for hours/days, heavy loss of revenue

Network Verification
automatically check network correctness with formal methods
Realtime Network Verification

Realtime Network verification
VeriFlow [NSDI’13], NetPlumber [NSDI’13], AP Verifier [ICNP’13], Delta-net [NSDI’17]

Check correctness of data plane state after each update in real time

correctness: reachability, no loops, no blackholes, isolation, …
real time: verify each update in small amount of time (<1ms)
**Equivalence Class (EC):** a set of packets with the same forwarding behavior

- **IP header space:**
  - EC1
  - EC2

- **Data plane state:**
  - 0.0.0.0/1: fwd B
  - 0.0.0.0/1: received
  - 0.0.0.0/1: fwd D

- **Network model:**
  - ECs allowed by the edge
Preliminary to Realtime Network Verification

**Incremental update and verification** [VeriFlow, NSDI’13] [AP Verifier, ICNP’13] [Delta-net, NSDI’17]

Update the ECs

Data plane state

Network model
**Preliminary to Realtime Network Verification**

**Incremental update and verification** [VeriFlow, NSDI’13] [AP Verifier, ICNP’13] [Delta-net, NSDI’17]

**Update the ECs** >> **Update the model**

**IP header space**

0.0.0.0/1

EC1  EC3  EC4  EC2

64.0.0.0/3

**Data plane state**

0.0.0.0/1: fwd B  64.0.0.0/3: fwd C
0.0.0.0/1: received

**Network model**

A  3  B  1,3,4  D

A  1,3,4  C  1,3,4

Update the ECs >> Update the model
Incremental update and verification [VeriFlow, NSDI’13] [AP Verifier, ICNP’13] [Delta-net, NSDI’17]
Realtime Verification for “Real” Networks

FW rules:
- dstIP=192.168.0.0/16  port5
- dstIP=192.168.10.0/24  VLAN10
- ...
Realtime Verification for “Real” Networks

Various functionalities beyond forwarding
- filtering (ACL), rewriting (NAT), traffic policy, ...

Requirement 1: Network model should be expressive of common functionalities

FW rules:
- dstIP=192.168.0.0/16   port5
- dstIP=192.168.10.0/24  VLAN10
- ...

ACL1 → ACL2
ACL2 → FW
FW → NAT

Vlan10

port1 → dstIP=192.168.0.0/16   port5
port4 → dstIP=192.168.10.0/24  VLAN10 ...

port2 → ...
port3 → ...
port5 → ...
Realtime Verification for “Real” Networks

Various functionalities beyond forwarding
- filtering (ACL), rewriting (NAT), traffic policy, ...

Requirement 1: Network model should be expressive of common functionalities

Multiple fields other than IP prefix
- 5-tuples used by ACL, traffic policy, NAT, etc.

Requirement 2: Update of ECs should be scalable for multi-field rules

FW rules:
- $\text{dstIP}=192.168.0.0/16$ $\text{port5}$
- $\text{dstIP}=192.168.10.0/24$ $\text{VLAN10}$
- ...

ACL1 rules:
- $\text{dstIP}=10.0.0.0/16$ $\text{dstPort}=22$ permit
- $\text{dstIP}=10.0.1.0/24$ $\text{srcIP}=10.0.2.0/24$ $\text{dstPort}=80$ deny
- ...

Scalability Issue due to Multi-Field Rules

(1) ECs based on Ranges: fast for single-dimensional forwarding rules

**Forwarding rules**

R1. **dstIP**=00: forward port2  
R2. **dstIP**=10: forward port2

**ACL rules**

R3. **dstIP**=0*, **dstPort**=0: deny  
R4. **dstIP**=**, **dstPort**<2: permit

<table>
<thead>
<tr>
<th>Network</th>
<th>#fw rules</th>
<th>#acl rules</th>
<th># of ECs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stanford</td>
<td>$3.84 \times 10^3$</td>
<td>686</td>
<td>15,100,968</td>
</tr>
<tr>
<td>Purdue</td>
<td>$3.52 \times 10^6$</td>
<td>2707</td>
<td>&gt;104,743,229</td>
</tr>
</tbody>
</table>

**Explosion of ECs**

- Memory overflow
- Long verification time
Scalability Issue due to Multi-Field Rules

(2) ECs based on Atomic Predicates [AP Verifier, ICNP’13]: minimum # of ECs

Forwarding rules

R1. $\text{dstIP} = 00$: forward port2
R2. $\text{dstIP} = 10$: forward port2

ACL rules

R3. $\text{dstIP} = 0^*$, $\text{dstPort} = 0$: deny
R4. $\text{dstIP} = 0^*$, $\text{dstPort} < 2$: permit

Network & #fw rules & #acl rules & # of ECs
\hline
Stanford & $3.84 \times 10^3$ & 686 & 515 \\
Purdue & $3.52 \times 10^6$ & 2707 & 4160 \\

challenging to update atomic predicate fast
- An update potentially affects all atomic predicates
- Checking all atomic predicates is expensive (~10ms)
APKeep

- Modular Network Model
- Scalable Update of ECs
The Modular Network Model

A Network Device

ACL1 → ACL2
ACL2
FW
port1
port2

vlan10
port3
don't care
port4
don't care
port5

NAT
The Modular Network Model

A Network Device

ACL1

ACL2

FW

NAT

elements

permit in deny

permit in deny

permit in deny

rule1 in id

port1

port2

port3

port4

port5

default

vlan10

The Modular Network Model

in permit deny

in permit deny

in permit deny

in permit deny

in permit deny

in permit deny
APKeep

- Modular Network Model
- Scalable Update of ECs
Equivalence Class in Modular Network Model

The model supports general representation of EC
Equivalence Class in Modular Network Model

The model supports general representation of EC

predicate a: dstIP=10.0.0.0/16 \land dstPort \neq 22
Fast Update of Minimum Number of ECs

APKeep fast updates the minimum number of ECs (atomic predicates) with three operations:

**Split** a predicate*

![split diagram]

<table>
<thead>
<tr>
<th>port1</th>
<th>port2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \land b, a \land \neg b$</td>
<td></td>
</tr>
</tbody>
</table>

**Transfer** a predicate

![transfer diagram]

<table>
<thead>
<tr>
<th>port1</th>
<th>port2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td></td>
</tr>
</tbody>
</table>

**Merge** predicates

![merge diagram]

<table>
<thead>
<tr>
<th>port1</th>
<th>port2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \land b, a \lor b$</td>
<td></td>
</tr>
</tbody>
</table>

*Inspired by AP Verifier to compute atomic predicates
Example

ACL rules
R1. dstIP=0*, dstPort=0: deny
R2. dstIP=**, dstPort<2: permit

Forwarding rules
R3. dstIP=00: forward port2
R4. dstIP=10: forward port2

Device
Initial State without Rules

ACL
permit
deny
FW
port1
port2
default

\begin{array}{c}
\text{dstPort: } y_1 y_2 \\
\text{dstIP: } x_1 x_2 \\
a \leftarrow \text{True}
\end{array}
Initial State without Rules

ACL rules

R1. dstIP=0*, dstPort=0: deny

No behavior change:
o no need to update
Splitting and Transferring Predicates

ACL rules

R1. \( \text{dstIP}=0\ast, \text{dstPort}=0 \): deny
R2. \( \text{dstIP}=** \), \( \text{dstPort} < 2 \): permit

\[
\begin{array}{c|c}
\text{dstIP}: x_1x_2 & \text{dstPort}: y_1y_2 \\
00 & 00 \\
01 & 01 \\
10 & 10 \\
11 & 11 \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{ACL} & \text{FW} \\
\text{permit} & \text{port1} \\
\text{deny} & \text{port2} \\
\text{default} & a \\
\end{array}
\]
Splitting and Transferring Predicates

ACL rules

R1. dstIP=0*, dstPort=0: deny
R2. dstIP=**, dstPort<2: permit

Part of a changes behavior from deny to permit
Splitting and Transferring Predicates

ACL rules

R1. dstIP=0*, dstPort=0: deny
R2. dstIP=**, dstPort<2: permit

Transfer part of a from deny to permit
Splitting and Transferring Predicates

ACL rules

R1. dstIP=0*, dstPort=0: deny
R2. dstIP=**, dstPort<2: permit

Split a to b and a-b

Transfer part of a from deny to permit
Splitting and Transferring Predicates

ACL rules
R1. dstIP=0*, dstPort=0: deny
R2. dstIP=**, dstPort<2: permit

Transfer part of a from deny to permit
Merging Predicates

ACL rules

R1. dstIP=0*, dstPort=0: deny
R2. dstIP=**, dstPort<2: permit

Forwarding rules

R3. dstIP=00: forward port2
R4. dstIP=10: forward port2

ACL
dstIP: $x_1 x_2$
dstPort: $y_1 y_2$

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>c</th>
<th>a</th>
<th>e</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>c</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>01</td>
<td>b</td>
<td>b</td>
<td>f</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FW

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>port1</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>port2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>default</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

c and e have the same forwarding behavior
Merging Predicates

ACL rules
R1. dstIP=0*, dstPort=0: deny
R2. dstIP=**, dstPort<2: permit

Forwarding rules
R3. dstIP=00: forward port2
R4. dstIP=10: forward port2

ACL
permit : b d
deny : a c

FW
port1 : a b
donport2: c d
default:
System Implementation

Verification applications running on the model (policy checker still under development)

Using AP Verifier and APT to encode match fields and packet rewriting action with BDD (BDD library: JDD)

Parsing config files and rule updates into vendor-neutral format

AP Verifier and APT are open source, available at:
8 Datasets from Stanford, Internet2, Purdue, and Delta-net

- 6 datasets with only IP forwarding rules
- 2 datasets with both IP forwarding rules and ACL rules

<table>
<thead>
<tr>
<th>Network</th>
<th>Nodes</th>
<th>Links</th>
<th>Forwarding rules</th>
<th>ACL rules</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airtel1</td>
<td>68</td>
<td>260</td>
<td>$6.89 \times 10^4$</td>
<td>0</td>
<td>$1.42 \times 10^7$</td>
</tr>
<tr>
<td>Airtel2</td>
<td>68</td>
<td>260</td>
<td>$9.84 \times 10^4$</td>
<td>0</td>
<td>$5.05 \times 10^8$</td>
</tr>
<tr>
<td>4Switch</td>
<td>12</td>
<td>16</td>
<td>$1.12 \times 10^6$</td>
<td>0</td>
<td>$1.12 \times 10^6$</td>
</tr>
<tr>
<td>Internet2</td>
<td>9</td>
<td>56</td>
<td>$1.26 \times 10^5$</td>
<td>0</td>
<td>$2.52 \times 10^5$</td>
</tr>
<tr>
<td>Stanford*</td>
<td>16</td>
<td>74</td>
<td>$3.84 \times 10^3$</td>
<td>0</td>
<td>$7.68 \times 10^3$</td>
</tr>
<tr>
<td>Purdue*</td>
<td>1,646</td>
<td>3,094</td>
<td>$3.52 \times 10^6$</td>
<td>0</td>
<td>$7.04 \times 10^6$</td>
</tr>
<tr>
<td>Stanford</td>
<td>124</td>
<td>182</td>
<td>$3.84 \times 10^3$</td>
<td>686</td>
<td>$9.05 \times 10^3$</td>
</tr>
<tr>
<td>Purdue</td>
<td>2,159</td>
<td>3,607</td>
<td>$3.52 \times 10^6$</td>
<td>2,707</td>
<td>$7.05 \times 10^6$</td>
</tr>
</tbody>
</table>
Evaluation – Verification Speed

Verification: checking loops after each update. Setting: Linux desktop with 3.0GHz Intel Core i5 CPU and 32GB RAM
**Evaluation – Verification Speed**

- **IP forwarding only**: 90%: <50μs
- **+ ACL**: 90%: <250μs

**Verification**: checking loops after each update. **Setting**: Linux desktop with 3.0GHz Intel Core i5 CPU and 32GB RAM
### Evaluation – Verification Speed

<table>
<thead>
<tr>
<th>Network</th>
<th>AP Verifier</th>
<th>VeriFlow</th>
<th>Average time (µs)</th>
<th>Delta-net$_{MF}$</th>
<th>APKeep$^-$</th>
<th>APKeep</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airtel1</td>
<td>80</td>
<td>59</td>
<td>3,804</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Airtel2</td>
<td>135</td>
<td>48</td>
<td>TO</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>4Switch</td>
<td>5,316</td>
<td>2,706</td>
<td>19,678</td>
<td>4</td>
<td>2,190</td>
<td>21</td>
</tr>
<tr>
<td>Internet2</td>
<td>1,660</td>
<td>144</td>
<td>2,123</td>
<td>3</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>Stanford*</td>
<td>1,953</td>
<td>468</td>
<td>8,700</td>
<td>9</td>
<td>98</td>
<td>94</td>
</tr>
<tr>
<td>Purdue*</td>
<td>777</td>
<td>648</td>
<td>MO</td>
<td>15</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>Stanford</td>
<td>2,072</td>
<td>$4.8 \times 10^6$</td>
<td>9,532</td>
<td>MO</td>
<td>$3.1 \times 10^5$</td>
<td>127</td>
</tr>
<tr>
<td>Purdue</td>
<td>TO</td>
<td>TO</td>
<td>MO</td>
<td>MO</td>
<td>MO</td>
<td>13</td>
</tr>
</tbody>
</table>

- **Verification**: checking loops after each update.
- **Setting**: Linux desktop with 3.0GHz Intel Core i5 CPU and 32GB RAM.

- **Timeout**: >24 hours
- **Memory overflow**: >32GB

Our multi-field extension of Delta-net
Conclusion

APKeep: checking correctness of data plane with real devices in real time

- Modular network model: expressive and extensible for real network devices
- Scalable update of ECs: fast updating the minimum number of ECs (<1ms)

Future work

- Checking operator intent beyond reachability
- Parallelizing the update of predicates
Thanks for your attention

Peng Zhang
p-zhang@xjtu.edu.cn