You Don't Need a Centralized Verifier: Toward Scalable Data Plane Checking via Distributed, On-Device Verification

Qiao Xiang¹, Chenyang Huang¹, Ridi Wen¹, Yuxin Wang¹, Xiwen Fan¹, Alan Liu², Linghe Kong³, Dennis Duan⁴, Franck Le⁵, Wei Sun⁶

¹Xiamen University, ²University of Maryland, ³Shanghai Jiao Tong University, ⁴Yale University, ⁵IBM T.J. Watson Research Center, ⁶UT Austin

June 2023, Hong Kong, China, ACM APNet'23
Background

• Modern networks are
  • increasingly more complex
  • outages increasingly more costly

Source: http://groups.geni.net/geni/chrome/site/thumbnails/wiki/TangoGENI/OF-VLAN3715_1000.jpg
Network Verification: Powerful Tool to Check Network Forwarding Behavior

Control Plane
- config
- execute

Data Plane
- FIB

Control plane verification
- Batfish [NSDI'15]
- Bagpipe [OOPSLA'16]
- Minesweeper [SIGCOMM'17]
- Bonsai [SIGCOMM'18]
- Hoyan [SIGCOMM'21]
- ...

Data plane verification
- HSA [NSDI'12]
- Veriflow [NSDI'13]
- Atomic Predicates [ICNP'13]
- APKeep [NSDI'20]
- Flash [SIGCOMM'22]
- ...
Why Study Data Plane Verification?

- Data plane is the ground truth of actual forwarding behavior

Google was hit with massive outage, including YouTube, Gmail and Google Classroom

By David Goldman, CNN Business
Updated 1723 GMT (0123 HKT) December 14, 2020

Q: Why CP verification fails to catch this?
A: ACL/routing configurations pass control plane verification, but with a different input from the actual network
How Does DPV Usually Work?

• A (cluster of) server(s) as a **centralized** verifier
  • Collect the data plane (e.g., FIB/ACL) from routers/switches through management network
  • Run **computation-intensive, memory-intensive computation** to verify the network data plane
Issues of Centralized DPV

1. Need a highly-available management network
   • Incomplete DP, inconclusive result

2. Verifier becomes performance bottleneck
   • Larger network, more computation power

3. Verifier becomes single PoF
   • Verifier fails, no result

4. Out-of-band verification
   • Slow reaction to errors
Our Proposal: Distributed, On-Device Verification

• **Basic idea**: offload verification to distributed computations on network devices

• **Benefits**:
  • Minimize the requirement of management network
    • Once configured, run automatically
  • No performance bottleneck
    • More network devices, more verifiers
  • No single PoF
    • Device malfunction equals a network error
  • Verification-as-a-service for vendors
    • Support flexible in-situ fast reroute
Distributed, On-Device Verification: Challenges

• **Challenge 1**: How to specify the invariants to check?
  • Most DPV tools only check a fixed set of invariants (e.g., reachability, loop-free and blackhole-free)

• **Challenge 2**: How to make the on-device tasks lightweight?
  • Switches/routers have low-end CPU and already run multiple protocols (e.g., SNMP, OSPF and BGP)

• **Challenge 3**: How to make devices exchange results correctly and efficiently?
  • Distributed computing has its own issues (e.g., safety, liveness and consistency)
Tulkun: A Generic, Distributed, On-Device DPV Framework

C1: Invariant specification
C2: lightweight tasks decomposition
C3: collaborative verification

- Declarative Invariant Specification Language
- Verification Planner
- On-Device Verifiers / DV Protocol

Verification Invariant + Network Topology

DVNet

On-Device Tasks

Verification Messages
A Running Example

Invariant: all packets entering the network from S with a destination IP in 10.0.0.0/23 must be delivered to D in a simple path waypointing W.

Topology:
Step 1: Invariant Specification

**Invariant:** all packets entering the network from $S$ with a destination IP in $10.0.0.0/23$ must be delivered to $D$ in a simple path waypointing $W$.

**Tulkun program:**

\[(\text{dstIP} = 10.0.0.0/23, [S], S .* W .* D \text{ and loop_free, } \text{"exist } \geq 1\text{"})\]
Step 2: From Invariant and Topology to DVNet

**Invariant:**

\[
\text{dstIP} = 10.0.0.0/23, \ [S], S \cdot W \cdot D \text{ and loop-free, } "\text{exist } \geq 1"
\]

\(X\) (DFA multiplication)

**Topology:**

**DVNet:** a DAG representing all valid paths in the network
Step 3: From Verification in Network to Counting in DVNet

- Each node counts # of downstream paths provided by data plane in reverse topological order

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.0/23</td>
<td>fwd(ALL, {A})</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.0/24</td>
<td>fwd(ANY, {B, W})</td>
</tr>
<tr>
<td>10.0.1.0/24</td>
<td>fwd(ALL, {W})</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.0/23</td>
<td>fwd(ALL, {C})</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.0/23</td>
<td>fwd(ALL, {C})</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.0/23</td>
<td>fwd(ALL, {D})</td>
</tr>
</tbody>
</table>

S

A

B

W

C

D

S: $[(P_1, 0)]$

A: $[(P_1, 0)]$

B: $[(P_1, 0)]$

W: $[(P_1, 0)]$

C: $[(P_1, 0)]$

D: $[(P_1, 0)]$

Error found!

\[(P_2, [0, 1]), (P_3, 1)\]

$P_1$: dstIP=10.0.0.0/23

$P_2$: dstIP=10.0.0.0/24

$P_3$: dstIP=10.0.1.0/24
Step 4: Distributed, Event-Driven, On-Device Counting

- Counting in DVNet can be naturally decomposed and executed at corresponding devices

<table>
<thead>
<tr>
<th>S</th>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10.0.0.0/23</td>
<td>fwd(ALL, {A})</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10.0.0.0/24</td>
<td>fwd(ANY, {B, W})</td>
</tr>
<tr>
<td></td>
<td>10.0.1.0/24</td>
<td>fwd(ALL, {W})</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10.0.0.0/23</td>
<td>fwd(ALL, {W})</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W</th>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10.0.0.0/23</td>
<td>fwd(ALL, {C})</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10.0.0.0/23</td>
<td>fwd(ALL, {D})</td>
</tr>
</tbody>
</table>
Tulkun: Extensions

1. Data plane with packet transformation
   • **Sub-pub mechanism** added to on-device tasks

2. Message-free local verification
   • **All-path-availability**: any task finding error equals network error (Microsoft RCDC [SIGCOMM'19] becomes a special case)

3. Fault-tolerant invariant (e.g., shortest path under k-link-failure)
   • **Precomputing** fault-tolerant DVNet

4. Incremental deployment
   • device/vm interconnect + **partition**
Implementation

- A total ~8K lines of Java code
- On-device verifier runs on SONiC and ONL
  - Requirement: JRE
- SONiC HLD proposal
  - [https://github.com/sonic-net/SONiC/pull/948](https://github.com/sonic-net/SONiC/pull/948)
  - To be open-sourced upon publication
Experiment 1: Functionality Demonstration

- 6-switch testbed: 4 Mellanox, 1 UfiSpace, 1 Edgecore
- Demo scenarios:
  - Waypoint reachability (running example)
  - Multicast
  - Anycast
  - Different-ingress consistent reachability
  - All-shortest-path availability
- Demo videos: [http://distributeddpvdemo.tech/](http://distributeddpvdemo.tech/)
- Interactive demos: [https://distributeddpchecking.com:8443](https://distributeddpchecking.com:8443)
Experiment 2: Verifying Internet2

• 9-switch testbed with Internet2 topology: 4 Mellanox, 4 P4, 1 Edgecore whitebox
• Inject propagation latency among switches based on device location in topology
• Public Internet2 data plane dataset
• Snapshot, all-rule verification: finish verifying all-pair loop-free, blackhole-free, <=x+2-hop reachability in 0.99 second
• 10,000 incremental, per-rule-update verification: 80% finish in <=5.42ms
Experiment 3: Large-Scale Simulations

- 13 datasets spanning campus networks, WAN and DCN
- Compare with 5 centralized DPV tools: Veriflow [NSDI'13], Deltanet [NSDI'17], AP [ICNP'13], APKeep [NSDI'20], Flash [SIGCOMM'22]

<table>
<thead>
<tr>
<th>Network</th>
<th>#Devices</th>
<th>#Links</th>
<th>#Rules</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet2 [56]</td>
<td>9</td>
<td>28</td>
<td>$7.74 \times 10^4$</td>
<td>WAN</td>
</tr>
<tr>
<td>Stanford [42]</td>
<td>16</td>
<td>74</td>
<td>$3.84 \times 10^3$</td>
<td>LAN</td>
</tr>
<tr>
<td>Airtel1-1 [35]</td>
<td>16</td>
<td>26</td>
<td>$2.83 \times 10^4$</td>
<td>WAN</td>
</tr>
<tr>
<td>Airtel2-1 [35]</td>
<td>68</td>
<td>158</td>
<td>$3.81 \times 10^4$</td>
<td>WAN</td>
</tr>
<tr>
<td>Airtel1-2</td>
<td>16</td>
<td>26</td>
<td>$9.60 \times 10^4$</td>
<td>WAN</td>
</tr>
<tr>
<td>Airtel2-2</td>
<td>68</td>
<td>158</td>
<td>$4.56 \times 10^5$</td>
<td>WAN</td>
</tr>
<tr>
<td>B4-2013</td>
<td>12</td>
<td>18</td>
<td>$7.92 \times 10^4$</td>
<td>WAN</td>
</tr>
<tr>
<td>B4-2018</td>
<td>33</td>
<td>53</td>
<td>$1.37 \times 10^5$</td>
<td>WAN</td>
</tr>
<tr>
<td>BT North America</td>
<td>36</td>
<td>76</td>
<td>$2.52 \times 10^5$</td>
<td>WAN</td>
</tr>
<tr>
<td>NTT</td>
<td>47</td>
<td>63</td>
<td>$1.98 \times 10^5$</td>
<td>WAN</td>
</tr>
<tr>
<td>OTEGlobe</td>
<td>93</td>
<td>103</td>
<td>$7.22 \times 10^5$</td>
<td>WAN</td>
</tr>
<tr>
<td>Fattree ($k = 48$)</td>
<td>2,880</td>
<td>55,296</td>
<td>$3.31 \times 10^6$</td>
<td>DC</td>
</tr>
<tr>
<td>NGClos</td>
<td>6,016</td>
<td>43,008</td>
<td>$3.23 \times 10^7$</td>
<td>DC</td>
</tr>
</tbody>
</table>
Simulation: Snapshot Verification

Tulkun accelerate snapshot verification by up to 1250x
Simulation: 10,000 Incremental Per-Rule-Update Verification

Tulkun verifies at least 73% of rule updates in less than 10 ms and up to 2355x faster than SOTA in 80% quantile.
Experiment 4: Microbenchmark on Commodity Switches

- Initialization overhead (computing local equivalence classes)
  - Memory $\leq 20$ MB, CPU load $\leq 0.46$, initialization in 1.75 s
Experiment 4: Microbenchmark on Commodity Switches

- Message processing overhead
  - Memory $\leq 45$ MB, CPU load $\leq 0.2$, 90% messages processed in 4 ms
Summary

• Advantages of distributed DPV over centralized DPV:
  • better scalability, no single PoF, verification-as-a-service for vendors

• Tulkun design elements:
  • declarative invariant specification language; planner via DVNet; on-device verifier and messaging protocol

• Extensive evaluation using demo/testbed/simulation/microbenchmark: real-time, scalable in-network verification

• Publications: ACM HotNets'22, ACM SIGCOMM'23

• Open-source/demos: https://github.com/sonic-net/SONiC/pull/948
  https://distributeddpchecking.com:8443/
  http://distributeddpvdemo.tech/

For more information about Tulkun, reach out by email (qiaoxiang@xmu.edu.cn) or WeChat
What Happens Next: Automating the Full Cycle of Network Configuration Management

CPV
A-Graph [INFOCOM'21]
CCoral [WIP]
DPV
Coral [HotNets'22]
[SIGCOMM'23]
Flash [SIGCOMM'22]

Verification

Interdomain Configuration

Interdomain Synthesis
SDI [INFOCOM'20]

Interdomain Verification
IVeri [arXiv'21]
ICoral [WIP]
InCV [APNet'23]

Interdomain Repair
IRepair [WIP]

Repair

Diagnosis / Repair
Scalpel [APNet'23]

Diagnosis / Localization
XMU SNGroup

SNGroup: sngroup.tech
Contact us: sngroup.xmu@outlook.com

Welcome to Visit Us!

Young team
- 6 PhD students
- 20 graduate students
- 4 undergraduates

Fun research
- Network and formal methods
- Programmable networks
- High-performance networks
- Interpretable intelligent networks