VeriFlow: Verifying Network-Wide Invariants in Real Time

Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, P. Brighten Godfrey
University of Illinois

Presented by Ofri Ziv
November 2013
Outline

• Motivation
• Design
• Evaluation
• Example
• Conclusion
Motivation

• Networks are complex
  • Ensure network’s correctness and security

• SDN increases software complexity
  • Multiple applications program the physical network simultaneously

• Check network-wide invariants as network evolves

• Prevent bugs as they arise
Bugs Effect

• Allow unauthorized packets to enter a secured zone in a network

• Make services and the infrastructure prone to attacks

• Make critical services unavailable

• Affect network performance
Configuration Verification (Offline)

• Problems:
  • Prediction is difficult
    • Various configuration languages
    • Dynamic distributed protocols
  • Miss control-plane implementation bugs
VeriFlow approach: Data-plane Verification

• Advantages:
  • Less prediction
  • Closer to actual network behavior
  • Unified analysis for multiple control-plane protocols
  • Catch control-plane implementation bugs
Challenges

• Obtaining real time view of the network
  • Interpose between controller and network elements
  • Utilize the centralized data-plane view available in an SDN (Software-Defined Network)

• Verification speed
The Tool: VeriFlow

• Checks network-wide invariants in real time using data-plane state
  • Absence of routing loops, black holes, access control violations, etc.

• Functions by
  • Monitoring dynamic changes in the network
  • Constructing a model of the network behavior
  • Using custom algorithms to automatically derive whether the network contains errors
VeriFlow Overview

Controller

New Flow

Generate Equivalence Classes

Generate Forwarding Graphs

Run Queries

VeriFlow

Report:
- network invariant violation
- Affected set of packets

"Good Rule"

"Bad Rule"
Limit the search space

- **Equivalence class**: Packets experiencing the same forwarding actions throughout the network

  - **Fw Rules**:
    - 0.0.0.0/1
    - 64.0.0.0/3

  - **Eq. classes**:
    - 1
    - 2
    - 3
    - 4
Computing Equivalence Classes

A = (Match =0.1, Action, device)
B = (Match =0.*, Action, device)

Eq. Classes – {0.0}, {0.1}
Represent Forwarding Rules

- **Forwarding graphs:**
  - Nodes representing network devices
  - Edges representing forwarding rules

- All the information to answer queries
Check Invariants

• **Queries:**
  • Black holes
  • Routing loops
  • VLANs Isolation
  • Access control policies

• **Response:**
  • **Good Rules** → Send flow to network element
  • **Bad Rules** → Report: invariant violated, affected set of packets
Evaluation #1 – Microbenchmarking VeriFlow run time

- **Goal**: Observe VeriFlow’s different phases contribution to the overall run time

- Simulated an IP network
  - 172 routers

- Replayed BGP traces
  - 5 million RIB entries
  - 90K BGP updates
Evaluation #2 –
Effect on TCP connection setup latency

• Experiment #2 – Impact of VeriFlow on TCP connection setup latency

• Mininet OpenFlow network
  • 10 switches arranged in chain-like topology
  • A host connected to every switch

• NOX controller running “learning switch” app

• TCP connections between random pairs of hosts
Future Work

• Handling packet transformations

• Deciding when to check (transactions)

• Handling queries other than reachability

• Dealing with multiple controllers
Demo application

hosts = {<ip: (device, port)>}
switches = {(sw1, sw2): port}

def packet_in(pkt, in_port, device):
    if (GARP == pkt.proto):
        if (hosts.has_key(pkt.src_ip)):
            (d,i) = hosts[pkt.src_ip]
            delete_flow(match=pkt.src_ip, d)
            hosts[pkt.src_ip] = (device, in_port)
            install_flow(match=pkt.src_ip, out=in_port, device)
    else if (hosts.has_key(pkt.dst_ip)):
        (d,i) = hosts[pkt.dst_ip]
        install_flow(match=pkt.dst_ip, out=switches[(device,d)], device)
        send_packet(pkt, switches[(device,d)], device)
Conclusion

• VeriFlow achieves real-time verification:
  • A layer between SDN controller & network elements
  • Find faulty flows issued by SDN applications
  • Verify network-wide invariants as each flow is inserted

• Can prevent a flow from reaching the network