Programmable Measurement and Monitoring for Software Defined Networks

Yang Guo (yang.guo@nist.gov)

https://www.nist.gov/software-defined-virtual-networks
Technical Approach

• Leverage open source networking and emerging AI to develop secure and resilient networks

• Develop novel network measurement technologies
• Automate network measurement and anomaly detection via network programmability
• Leverage AI for autonomous and secure networks
Instrumenting Open vSwitch with Monitoring Capabilities
Motivation

• Fine-grained and flexible network traffic monitoring is important for effective network management
  • Traffic engineering, anomaly detection, network diagnosis, traffic matrix estimation, DDoS
detection and mitigation, etc.

• Scalability has been the main challenge
  • High switching speed
  • Large number of flows
  • Solution: sampling, probabilistic based measurement, hardware enhanced measurement
  solutions, etc.

• Open vSwitch (OVS) is a popular software switch
  • Developed by Nicira as an edge switches for Data center
  • slower switching speed, smaller #flows, access to more CPU and memory resources
Motivation

• Our Idea: instrument software switch to provide *user-defined traffic monitoring*

• Why software switch?
  • Slower switching speed
  • Access to more resources (both CPU and memory)
  • Sitting at the edge
  • Open source

• What UMON aims to achieve?
  • Monitor arbitrary fields
  • Programmable monitoring
  • Allow to push other management functions, such as anomaly detection, to the switches
Establishing the Technical Basis for Trustworthy Networking

UMON Architecture

Challenge I: Minimal Implementation Complexity

Combining forwarding rule with monitoring rule

Monitoring APIs

Table

0

Table

1

Table

n

Upcall

Userspace

Kernel Space

Install Flow Rule

Packet Out

recv packet

Fine-grained Kernel Flow Cache

Kernel Flow Cache gets bloated during peak traffic + handlers and revalidators get heavily loaded

[DstPort=80, Counts('SrcIP')]

[SrcIP=C, DstIP=A, DstPort=80, Output(0)]

[SrcIP=C, DstIP=A, DstPort=22, Output(0)]

[SrcIP=D, DstIP=B, DstPort=80, Output(1)]

[SrcIP=D, DstIP=B, DstPort=22, Output(1)]

[SrcIP=C, DstPort=80, count]

[SrcIP=D, DstPort=80, count]
On-Path FCAP/SMON

Userspace
Kernel Space

recv packet

Kernel Flow
Cache

Monitoring
APIs

[DstPort=80]
Table

Subflow
Tables

Netlink

FCAP (FlowCapture)

SMON (SketchMon)

Challenge II:
Resource-Accuracy
Tradeoffs

n-tuple flow stats
FCAP

Flow XOR-ed n-tuple + stats

SMON IBLT
Off-Path FCAP/SMON

Userspace

Kernel Space

recv packet

Kernel Flow Cache

Monitoring APIs

Subflow Tables

Monitoring Table

ring buffer cache

Sample collector thread

PacketOut

Filter Table

Custom n-tuple Table

Challenge III: Minimal Interferences with Forwarding Path
Evaluation

• Testbed Setup
  • Intel Xeon 4-Core 3.20GHz CPU; 4GB memory
  • Host and OVS connected with 10Gbps cables
  • Ryu SDN controller

• Total CPU utilization of all related threads:
  • 2 handlers + 2 revalidators
  • collector thread in the userspace
  • Custom sample_collector thread in the kernel module

UMON incurs highest CPU utilization

FCAP incurs less CPU overhead than SMON
Establishing the Technical Basis for Trustworthy Networking

### Overall Comparison and Insights

<table>
<thead>
<tr>
<th>Designs</th>
<th>On-Path</th>
<th>Off-Path</th>
<th>UMON</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SMON</td>
<td>FCAP</td>
<td>SMON</td>
</tr>
<tr>
<td>CPU Overhead</td>
<td>moderate</td>
<td>low</td>
<td>moderate</td>
</tr>
<tr>
<td>Memory Consumption</td>
<td>low</td>
<td>low</td>
<td>moderate</td>
</tr>
<tr>
<td>Measurement Accuracy</td>
<td>high</td>
<td>precise</td>
<td>high</td>
</tr>
<tr>
<td>Forwarding Latency</td>
<td>high</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>Implementation Complexity</td>
<td>high</td>
<td>high</td>
<td>high</td>
</tr>
</tbody>
</table>

- **UMON**: *least* implementation efforts; *highest* CPU overhead; *highest* memory consumption.
- **Off-path designs**: outperform on-path designs in terms of switching performance; higher memory usage.
- Hash table: *more efficient* than sketch, *lower* computational cost.
vPROM: vSwitch Enhanced Programmable Measurement
Software Defined Network Programmability

- Program the network with perception that underlying network is a single device
  - High-level languages
    - e.g., Frenetic, Pyretic, Ox
    - High-level, unified abstractions
    - Compositional semantics
- Run-time system
  - Handles module interactions
  - Deals with asynchronous behavior
- Controller client
  - Shim between runtime system and controller

![Diagram showing Data Plane, OFA server, and controller client interactions](image-url)
SDN based Programmable Measurement

- Network measurement controlled and managed by a program written in networking program language

- Benefits:
  - Automate the measurement process
  - Utilize software switches as measurement points across the networks
  - Acquire only necessary statistics
    - dynamically adjust what/where to measure
    - minimize resource usage
vPROM: vSwitch enhanced Programmable Measurement

• Issues with vanilla SDN based programmable measurement
  • Interaction between forwarding, monitoring, and other applications is complex
  • SDN controller is involved too frequently
  • Limited measurement resources, e.g., TCAM, at physical SDN switches
  • Packet and byte counts associated with flow forwarding entries are neither flexible nor sufficient

• Key Ideas: leverage instrumented UMON switch at network edge
  • Extend OpenFlow, run-time system, and network programming language to have a unified system
vPROM Architecture

1. UMON: instrumented Open vSwitch
2. OpenFlow+: extended OpenFlow protocol support UMON
3. Ryu client
4. Pyretic+: extended Pyretic runtime system
5. vPROM-GUARD: DDoS and port detection vPROM application
vPROM Example

\[ Q = \text{count_packets}(\text{interval}=t, \text{group_by}=[\text{`srcip', `dstip'}]) \]

\[ \text{match(ethtype=0X0800)} \& \text{match(protocol = 6)} \Rightarrow Q \]

- **Routing**
- **Monitoring**
- **Run-time System**
- **controller client**
- **Controller**

in_port=1,priority=60000,actions=output:2
priority=59999,actions=drop
tcp,actions=subflow_collection:nw_src=0.0.0.0/32,nw_dst=0.0.0.0/32
Pyretic Run-time System

match(inport=1) >> if_(match(protocol=6), Q, identity) >> fwd(2)
Q = count_packets(interval=t, group_by=['srcip', 'dstip'])
Establishing the Technical Basis for Trustworthy Networking

Pyretic+ Run-time System

Abstract syntax tree (AST)

Goal: using the same set of Pyretic abstractions/policies
• Three query policies are defined to collect statistics of packets of each group

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Packets(limit=n, group_by=[f1,f2,...])</code></td>
<td>Callbacks on every packet received for up to n packets identical on fields f1,f2, …</td>
</tr>
<tr>
<td><code>Count_packets(interval=t, group_by=[f1,f2,...])</code></td>
<td>Count every packet received. Callback every t seconds to provide count for each group</td>
</tr>
<tr>
<td><code>Count_bytes(interval=t, group_by=[f1,f2,...])</code></td>
<td>Counts every byte received. Callback every t seconds to provide count for each group</td>
</tr>
</tbody>
</table>

• `group_by` defines the granularity of subsets of flows; To support TCP flagged packets monitoring, we introduce ‘tcpflag’ to the `group_by` parameter

• new policy ‘prtscan_detection’ could activate/deactivate local port-scan detector
OpenFlow+ Protocol

• Monitoring Table Management

<table>
<thead>
<tr>
<th>OpenFlow message type</th>
<th>OpenFlow commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFPT_MONITOR_MOD</td>
<td>OFPMMC_ADD, OFPMMC_MODIFY, OFPMMC_DELETE, OFPMMC_MODIFY_STRICT, OFPMMC_DELETE_STRICT</td>
</tr>
</tbody>
</table>

• Stats Collection
  • Define a new multi-part message OFPMP_MONITOR_STATS with two types: OFPMR_ALL and OFPMR_EXACT

• Application Thread Management
  • Define new action OFPAT_PRTSCAN_DETECTION for vertical and horizontal scanning detections
vPROM-GUARD: DDoS and Port Scan Detection

• Anomaly detection often requires low level feature, e.g., packet-level or micro-flow, measurement at line rate – challenging

• vPROM-GUARD
  • monitor attack cues at coarse level when in normal operations
    • Monitoring TCP signaling packets – TCP {SYN, SYN/ACK} and {SYN, FIN} are request-response pairs that should be balanced
    • Using Cumulative Sum Method to detect the deviation
  • when suspicious activities are detected, switch to a full-blown fine grained network monitoring and start DDoS and port-scan detection at both edge UMON vSwitches and at the central vPROM application

• Benefits:
  • Only alerted hosts conduct fine grained measurement
  • Local detection at edge mitigates the burden at central detection
  • False alarms are more tolerable
vPROM-GUARD

<table>
<thead>
<tr>
<th>Flag Indicators</th>
<th>Potential Attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Big Flow" /> + <img src="image2" alt="CUSU M" /></td>
<td>TCP SYN flooding attack</td>
</tr>
<tr>
<td><img src="image3" alt="Big Flow" /></td>
<td>other types of DDoS attacks</td>
</tr>
<tr>
<td><img src="image4" alt="CUSU M" /></td>
<td>vPROM-GUARD starts collecting subflows and detecting port scanning attacks</td>
</tr>
</tbody>
</table>
vPROM-GUARD SYN Flood Attack Detection

- Vertical line: change-point monitoring issues a potential attack warning
- Dot: vPROM-GUARD actually detects the attack.
- ~10 seconds (2 polling periods)
Conclusions

• Propose, design and prototype vSwitch Enhanced Programmable Measurement framework
  • Instrument the edge software switch for measurement and anomaly detection
  • Automate the measurement process
  • Acquire only necessary statistics: minimize resource usage

• Related work
  • Network programmability (run-time system and network programming language) has been studied extensively
    • Frenetic, NetKAT, SDX, Kinetic, etc.
  • Flow-rule based measurement using physical SDN switches
    • Limited TCAM
  • Programmed measurement
    • Path query, intentional monitoring
    • Constant controller involvement
Accomplishment

• Publications:

• Open Source Code
  • Instrumented Open vSwitch source code, https://github.com/iOVS/iOVS
Ongoing Work and Future Direction (I)

• Machine Learning Based Network Anomaly Detection
  • Available AI based network anomaly detection suffers from multiple scaling issues
  • Aim to develop a Distributed ML based anomaly detection framework by leveraging vPROM framework
  • Conduct distributed monitoring and distributed ML based anomaly detection at the network edge as well as at a central location
Establishing the Technical Basis for Trustworthy Networking

**Userspace**
- Kernel Flow
- Cache

**Kernel Space**
- ring buffer cache
- Filter

**Data Plane**
- OVS
- Custom Thread
  - Flow Features
  - HTTP Detection
  - Suspicious Hosts

**Control Plane**
- Remote Controller
  - Correlation Module
    - Whitelist
    - Blacklist
    - Greylist
  - Offline Training
- Load TensorFlow Model

**Server**
- Packet Out
- Packet In

**C&C Detection Module**
- IRC
- P2P

**Load TensorFlow Model**
- Offline Training

**Remote Controller**
- TensorFlow Model
- Shared Memory
- Attack Detector

**Correlation Module**
- Offline Training

Ongoing Work and Future Direction (II)

• High-Speed Data Plane Measurement using Programmable Switches
  • Programmable switch is designed to be programmable using high-level domain specific language, e.g. P4
    • Implement new functions at line speed
    • A uniform pipeline of programmable stages to process packet headers in rapid succession
    • Fast rollout of new network protocols
  • Challenges:
    • Stringent time budget per pipeline stage (around 1ns)
    • Limited amount of memory per pipeline stage
• Coincidence Counting based Large Flow Detection in Data Plane
Questions and Discussion

• For more information:
  • Software Defined Virtual Networks
    • https://www.nist.gov/software-defined-virtual-networks
  • Advanced Network Technologies Division.
    • https://www.nist.gov/itl/antd
  • Information Technology Laboratory
    • https://www.nist.gov/itl