A Passive Network Appliance for Real-Time Network Monitoring

Michael J Schultz, Ben Wun, and Patrick Crowley
Applied Research Laboratory
Washington University in Saint Louis

This material is based upon work supported by the National Science Foundation under Grant No. CNS 096420
Million Mile View

Network Operators want to know about their network.
An Example

- End host suddenly opens many connections
  » What happened? Not sure, didn’t capture it – something bad?
Getting Data from a Network

- Capture complete packet traces? Nope.

- Sample packets? Nope.

- Special purpose equipment is costly/hard to maintain

Disk error: Out of space
The Passive Network Appliance

- Re-evaluate what modern commodity hardware can do
- First kernel-space network monitor (that we know of)

Specifically
  » Present our kernel-space network monitor
  » Explain our API that allows monitors to enforce policy at network frame granularity
  » Quantitative comparison between user-space and kernel-space monitors
PNA Design

Log Generation

PNA System

User-space

Packet Monitor

Decode

Active Logging

Kernel-space

Alert System Hooks

Alert Subsystem

Connection Monitor

Local IP Monitor

Real-Time Monitors

Packet In

Logging

Real-Time Monitoring

Packet Out
Logging and Log Generation

■ First and Foremost: Summarize the packet
  » Where is the packet from?
  » Gather up summary statistics (bytes, time, etc.)

■ Flush records every 10 seconds to capture state of network

■ Creates a file that can be aggregated to form continuous view of network
Real-Time Monitoring

- Allows network administrators enforce policy *as network frames arrive*

- Chain arbitrary number of monitors together
  - Has no direct effect on summary logging
  - Indirect effect of slowing down the system

- Alerts can be generated *at the moment* malicious activity is detected
Implementation Details

- Linux Kernel Module
  » Implies that it will have less overhead than any user-space monitor*

- Runs on commodity hardware
  » Servers are relatively low-cost (<$3000)
  » Un-patched Linux Kernel

* We’ll get into that a bit later.
Decoding

- Must be quick (every frame is logged)

Step 1

Decoding

\[ \text{flowkey} = <\text{IP, 192.168.53.7, 128.252.165.4, TCP, 63130, 80}> \]
Logging

- Must be quick (every frame is logged)

Step 2

Flow Table

- hash(flowkey)
- No match
- Match

Update entry

Flow table: ~8 million entries

Packet

Real-Time Monitors
Real-Time Monitors

- Every frame passes through monitors
- Enforce network policy at *per frame* granularity

Example: Connection Monitor

\[
\times 1000 \text{ conversations} = \text{probably bad}
\]
Extending the System

- Example: Find all HTTP traffic (on non-standard ports)
  - Write a hook() function
  - Look at payload for request method/response status
  - If found use pna_alert() to alert network operator

- Other functions
  - init() and release() prepare/destroy global resources
  - clean() runs every 10 seconds and can perform data maintenance

Your Monitor(s)!
Evaluation

- Tested with worst-case and real-world conditions
- PNA System
  - 2.27 GHz “Nehalem” with 12 GiB memory
  - Allows about 8 million flow table entries

LKPG = Linux Kernel Packet Generator
Ran with “base,” “flow,”, and “real-time” monitors

<table>
<thead>
<tr>
<th></th>
<th>Minimum sized packets</th>
<th>Maximum sized packets</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single flow</strong></td>
<td>Min table insertions</td>
<td>Min table insertions</td>
</tr>
<tr>
<td></td>
<td>Max packets/second</td>
<td>Max packets/second</td>
</tr>
<tr>
<td><strong>Many flows</strong></td>
<td>Max table insertions</td>
<td>Max table insertions</td>
</tr>
<tr>
<td></td>
<td>Min packets/second</td>
<td>Min packets/second</td>
</tr>
</tbody>
</table>
Min-sized Packet Throughput

Throughput (kilopackets per second)

one base
one flow
one real-time
many
many

Throughput (kilopackets per second)
Throughput at Various Packet Sizes

Throughput (% of peak rate at packet size)

Packet Size (in bytes)

64 128 256 512 1024 1514
Packet Entries/Drops (per second)

Packets Inserted or Dropped (thousands)

- inserts
- soft flow
- max-nic
- inserts
- soft real-time
- max-nic
Back in Reality

- Real networks don’t see 1.48 Million packets per second
  » Average packet size PNA sees is about 1000 bytes

- Graph of insertions (blue)/misses (orange)/drops (red)
  » Per 10 second period
Kernel-space v. User-space

- Known that syscall overheads hurt performance
  - Prior work minimizes syscall overheads (Deri [7], Braun [5])
  - What if we avoid syscalls altogether?

- Measure single-core performance: capture, count, drop

<table>
<thead>
<tr>
<th></th>
<th>Linux Default</th>
<th>PF_RING</th>
<th>Kernel Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput (Mbps)</td>
<td>495.89 ± 1.01</td>
<td>747.72 ± 7.38</td>
<td>951.75 ± 1.23</td>
</tr>
</tbody>
</table>
Summary

- PNA kernel module gives complete snapshots
- API for real-time monitors to enforce policy as frames arrive
- Evaluation under worst-case and real-world conditions
  - PNA logs at worst 43% of traffic
  - Typically captures all the traffic @ 1 Gbps
- Comparison of Linux default/PF_RING/kernel module

Code available at www.github.com/pcrowley/PNA