Performance Evaluation of PDM Implementation using eBPF in TC versus Traditional Kernel Methods

Amogh Umesh
Chinmaya Sharma

IEPG, IETF 119 @ Brisbane AU
Overview

- eBPF Concepts
- Extension header (PDM) implementation in eBPF
- Performance analysis of the eBPF implementation and kernel implementation of PDM
Why eBPF?

Why eBPF over a kernel implementation?
- Quicker development times and lesser maintenance
- More robust
- Better portability
- BPF verifier ensures safer implementation
- Accuracy of timestamp captured

Why eBPF over raw sockets?
- Adding extension header made easier by just making space in fully crafted packet
- Existing userspace applications need not be modified
tc-BPF

- Subset of eBPF programs attached at qdisc level
- Can be attached to both ingress and egress compared to only ingress in XDP
- Better packet mangling capability
- Executed after sk_buff is created
- Not good for complete packet rewrites
Implementation of PDM using tc-BPF

- PDM - [RFC8250](https://tools.ietf.org/html/rfc8250) is a destination options header used for measuring packet processing and network delays
- Using tc-BPF, so that we can attach to both ingress and egress of a interface
- Using bpf helpers for packet mangling
- eBPF maps to store the 5-tuple state
Benchmarking against Kernel Implementation of PDM

- CPU Cycles
- Network Throughput
- Packet Processing Latency
## CPU Cycles

<table>
<thead>
<tr>
<th>CPU Usage(cycles)</th>
<th>Mean</th>
<th>Median</th>
<th>St. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>eBPF Egress</td>
<td>8.60e10 cyc.</td>
<td>8.54e10 cyc.</td>
<td>9.08e9 cyc.</td>
</tr>
<tr>
<td>eBPF Ingress</td>
<td>1.53e10 cyc.</td>
<td>1.57e10 cyc.</td>
<td>8.71e9 cyc.</td>
</tr>
<tr>
<td>PDM Kernel</td>
<td>2.29e9 cyc.</td>
<td>2.13e9 cyc.</td>
<td>6.49e8 cyc.</td>
</tr>
</tbody>
</table>
### Network Throughput

<table>
<thead>
<tr>
<th>Network Throughput</th>
<th>Mean</th>
<th>Median</th>
<th>St. Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without PDM</td>
<td>18.80 Gbps</td>
<td>18.58 Gbps</td>
<td>2.19 Gbps</td>
</tr>
<tr>
<td>PDM Kernel Implementation</td>
<td>18.52 Gbps</td>
<td>18.33 Gbps</td>
<td>2.21 Gbps</td>
</tr>
<tr>
<td>eBPF Implementation</td>
<td>18.03 Gbps</td>
<td>17.22 Gbps</td>
<td>2.51 Gbps</td>
</tr>
</tbody>
</table>
# Packet Processing Latency (Per Packet)

<table>
<thead>
<tr>
<th>Packet Processing Latency</th>
<th>Mean</th>
<th>Median</th>
<th>St. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDM Kernel Implementation</td>
<td>0.707 µs</td>
<td>0.641 µs</td>
<td>0.414 µs</td>
</tr>
<tr>
<td>With eBPF Egress</td>
<td>5.808 µs</td>
<td>6.142 µs</td>
<td>0.986 µs</td>
</tr>
<tr>
<td>Without eBPF Egress</td>
<td>4.528 µs</td>
<td>4.668 µs</td>
<td>0.785 µs</td>
</tr>
<tr>
<td>With eBPF Ingress</td>
<td>3.634 µs</td>
<td>3.977 µs</td>
<td>0.906 µs</td>
</tr>
<tr>
<td>Without eBPF Ingress</td>
<td>3.082 µs</td>
<td>3.321 µs</td>
<td>1.246 µs</td>
</tr>
</tbody>
</table>

\[
eBPF\text{ Egress Mean Packet Processing Latency} - (5.808 - 4.528) \mu s = 1.28 \mu s
\]

\[
eBPF\text{ Ingress Mean Packet Processing Latency} - (3.634 - 3.082) \mu s = 0.552 \mu s
\]
Future Work

- Optimization of the eBPF program to find out the limits of how well an eBPF based extension header insertion program would work
- Performance Analysis of the eBPF program in high performance computing environments
- Implementation and analysis of other extension headers in eBPF
References

ebpf.io

RFC8250

tc-BPF

PDM-in-eBPF-draft