Flow-based Packet Process Framework on DPDK and VPP

Hongjun Ni, Qi Zhang @ Intel
Acknowledgement

VPP Community
John DiGiglio @ Intel
Ray Kinsella @ Intel
Jokul Li @ Intel
Xiang Wang @ Intel

Jerome Tollet @ Cisco
Dave Barach @ Cisco
Damjan Marion @ Cisco
Andreas Schultz @ Travelping
Mathias Gumz @ Travelping

*Other names and brands may be claimed as the property of others.
Agenda

• Flow-based Solution
• HW Offload and SW Flow Process
• TCP Segments Reassembly
• Hyperscan Block and Stream Mode
• Identifying Layer 7 Applications
• Key Takeaway

*Other names and brands may be claimed as the property of others.*
Flow-based Solution

Data Plane

- **NIC1**
  - Flow Director
- **NIC2**
  - RSS
  - SW Flow
  - Flow table
- **NIC3**
  - NIC
- **NIC4**
  - SW Flow
  - Flow table

Flow Director

- Flow table

Flow id

- HW flow

- Tunnel (VxLAN, GTPU, etc.)

Host Stack, QUIC

- Deep Packet Inspection

- Network Intelligence

- Qos
  - Rate Limiting
  - Routing
  - SDWAN

Routing

- SDWAN
Flow Configuration

• Configures static flows with 5-tuple and VRF-aware.

• Supports both ipv4 and ipv6 flows.

• First try HW offload to NIC based on DPDK rte_flow mechanism.

• If failed, then will create SW flow mappings.

• Each flow creates two HW or SW flow mappings, i.e. bi-directional traffic.

• Both flow mappings will be mapped to the same flow.
HW Flow Offload

- Leverages `rte_flow` mechanism from DPDK:
  - Supports ipv4-n-tuple, ipv6-n-tuple, ipv4-vxlan, ipv6-vxlan, etc.
  - Supports `rte_flow_item`, `rte_flow_action`, etc.
  - Using `rte_flow_create` to create a flow.
  - If failed, then create a SW flow session.

- HW flow matching:
  - If one packet matches a flow, then flow ID is marked by HW.
  - DPDK drivers sets flow ID to metadata of packet descriptor.
  - Subsequent features could retrieve flow ID from packet descriptor.
SW Flow Process

• Leverages RSS mechanism:

• Supports ipv4-n-tuple, ipv6-n-tuple, etc.

• Calculate hash value and look up SW flow table.

• If matched, set flow ID to metadata of packet descriptor.

• Subsequent features could retrieve flow ID from packet descriptor.
NICs & Flows

One flow is mapping to many NICs.
  • Same flow from all NICs are sent to same core to handle.

One NIC contains many flows.
  • Different flows from one NICs are sent to different core to handle.
If having same 5-tuple packet matching on 2 different VLANs or even on 2 different interfaces, they should be treated as different flows.
DPI Flow Process

• When HW flow offload matched, packets will be redirected to DPI plugin with dpi flow id in packet descriptor.

• If not, packets will be bypassed to DPI plugin from ip-input, and then lookup SW flow mapping table.
TCP Connection Track

• Tracks TCP three-way handshakes.

• Identify TCP traffic direction.

• Tracks TCP send sequence and ack sequence.
typedef struct dpi_app_match_rule_
{
    char *host;
    char *pattern;
    char *app_name;
    u32 app_id;
} dpi_app_match_rule;

dpi_app_match_rule app_match_rules[] = {

    {"www.cisco.com", NULL, "Cisco", DPI_APP_CISCO}
,
    ...
,
    {"www.intel.com", NULL, "Intel", DPI_APP_INTEL}
,
    ...
};
How Hyperscan Works

Phase 1: Compilation at system initialization phase

*Rulesets*
- User Defined

1. `/foo.*bar/s`
2. `/[a-f]{6,12}/i`
3. `/^GET\s.*HTTP/m`

**Mode (Block, Stream) Predefined Flags**

**Hyperscan Compiler**
- `hs_compile_multi`

**Bytecode (database)**

Phase 2: Data Searching & Match

**Runtime Init (Once)**
- Allocate Scratch
  - `hs_alloc_scratch`
- Scratch space
  - `hs_scratch_t`
- Hyperscan Runtime Scan (Block mode)
  - `hs_scan`

**Data Blocks**

**Match callback**
- Match for ID n at Offset i
Block Mode and TCP Segments Reassembly

• Block mode can scan rules only in a complete payload.

• If defining a rule "abcde", then for Block Mode, "abcde“ should be in a complete PDU payload.

• Requirements for TCP Segments process:
  • Reassembly TCP segments first to a complete PDU payload.
  • Scan PDU payload through Block mode.
  • Fragment TCP segments again.

• This degrades the performance.

• Most DPI open source projects leveraging Hyperscan performs in above way.
Stream Mode and TCP Segments Reassembly

- Stream mode can scan rules straddling into different TCP segments.

- If defining a rule "abcde", then for Stream Mode, then "abc" can be reside in packet 1, and "de" can be in packet 2.

- Requirements for TCP Segments process:
  - Reassemble TCP segments reassembly on the fly.
  - Can handle out-of-order tcp segments.
  - Can handle overlapping segments.

- This helps to improve the performance.

- VPP DPI plugin is implemented in this way.
Identifying Layer 7 Applications

• Identify SSL/TLS certificate message and subsequent segments.

• Scan SSL/TLS certificate message through hyperscan, and get application ID if hit.

• If maximum packets for this flow are scanned and not matched, the detection will end up.
<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.006000</td>
<td>10.67.118.188</td>
<td>10.240.252.16</td>
<td>TCP</td>
<td>74</td>
<td>53282 → 912 [SYN] Seq=0 Win=29208 Len=0 MSS=1460 SACK PERM=1 TSecr=0 WS=128</td>
</tr>
<tr>
<td>2</td>
<td>0.02603</td>
<td>10.240.252.16</td>
<td>10.67.118.188</td>
<td>TCP</td>
<td>78</td>
<td>912 → 53282 [SYN, ACK] Seq=8 Ack=1 Win=65535 Len=0 MSS=1360 SACK PERM=1 TSecr=2110335135 WS=64</td>
</tr>
<tr>
<td>3</td>
<td>0.02659</td>
<td>10.67.118.188</td>
<td>10.240.252.16</td>
<td>TCP</td>
<td>66</td>
<td>53282 → 912 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSecr=226840611 TSecr=2110335135</td>
</tr>
<tr>
<td>4</td>
<td>0.02907</td>
<td>10.240.252.16</td>
<td>10.67.118.188</td>
<td>HTTP</td>
<td>271</td>
<td>CONNECT <a href="http://www.intel.cn:443">www.intel.cn:443</a> HTTP/1.1</td>
</tr>
<tr>
<td>5</td>
<td>0.05291</td>
<td>10.240.252.16</td>
<td>10.67.118.188</td>
<td>TCP</td>
<td>66</td>
<td>[TCP Window Update] 912 → 53282 [ACK] Seq=1 Ack=1 Win=262848 Len=0 TSecr=226840611 TSecr=2110335138</td>
</tr>
<tr>
<td>6</td>
<td>0.06498</td>
<td>10.240.252.16</td>
<td>10.67.118.188</td>
<td>TCP</td>
<td>165</td>
<td>HTTP/1.1 200 Connection established</td>
</tr>
<tr>
<td>7</td>
<td>0.06539</td>
<td>10.67.118.188</td>
<td>10.240.252.16</td>
<td>TCP</td>
<td>66</td>
<td>53282 → 912 [ACK] Seq=206 Ack=40 Win=29312 Len=0 TSecr=226840612 TSecr=2110335139</td>
</tr>
<tr>
<td>8</td>
<td>0.06935</td>
<td>10.240.252.16</td>
<td>10.67.118.188</td>
<td>TLSv1.2</td>
<td>583</td>
<td>Client Hello</td>
</tr>
<tr>
<td>9</td>
<td>0.08239</td>
<td>10.240.252.16</td>
<td>10.67.118.188</td>
<td>TLSv1.2</td>
<td>1414</td>
<td>Server Hello</td>
</tr>
<tr>
<td>10</td>
<td>0.08277</td>
<td>10.240.252.16</td>
<td>10.67.118.188</td>
<td>TCP</td>
<td>1414</td>
<td>912 → 53282 [ACK] Seq=1388 Ack=723 Win=262848 Len=1348 TSecr=2110335125 TSecr=226840613 [TCP segment of a reassemble]</td>
</tr>
<tr>
<td>11</td>
<td>0.08279</td>
<td>10.67.118.188</td>
<td>10.240.252.16</td>
<td>TCP</td>
<td>66</td>
<td>53282 → 912 [ACK] Seq=723 Ack=2736 Win=34688 Len=0 TSecr=226840631 TSecr=2110335215</td>
</tr>
<tr>
<td>12</td>
<td>0.08282</td>
<td>10.240.252.16</td>
<td>10.67.118.188</td>
<td>TCP</td>
<td>1466</td>
<td>912 → 53282 [PSH, ACK] Seq=2736 Ack=723 Win=262848 Len=1400 TSecr=2110335215 TSecr=226840613 [TCP segment of a reassemble]</td>
</tr>
<tr>
<td>13</td>
<td>0.08281</td>
<td>10.67.118.188</td>
<td>10.240.252.16</td>
<td>TCP</td>
<td>66</td>
<td>53282 → 912 [ACK] Seq=4136 Ack=4356 Win=37964 Len=0 TSecr=226840631 TSecr=2110335215</td>
</tr>
<tr>
<td>14</td>
<td>0.08282</td>
<td>10.240.252.16</td>
<td>10.67.118.188</td>
<td>TLSv1.2</td>
<td>1159</td>
<td>Certificate, Certificate Status, Server Key Exchange, Server Hello Done</td>
</tr>
<tr>
<td>15</td>
<td>0.089214</td>
<td>10.67.118.188</td>
<td>10.240.252.16</td>
<td>TLSv1.2</td>
<td>192</td>
<td>Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message</td>
</tr>
<tr>
<td>16</td>
<td>0.089795</td>
<td>10.67.118.188</td>
<td>10.240.252.16</td>
<td>TLSv1.2</td>
<td>243</td>
<td>Application Data</td>
</tr>
<tr>
<td>17</td>
<td>0.089667</td>
<td>10.67.118.188</td>
<td>10.240.252.16</td>
<td>TLSv1.2</td>
<td>1172</td>
<td>Application Data</td>
</tr>
<tr>
<td>18</td>
<td>0.09239</td>
<td>10.240.252.16</td>
<td>10.67.118.188</td>
<td>TCP</td>
<td>66</td>
<td>912 → 53282 [ACK] Seq=5229 Ack=1026 Win=262056 Len=0 TSecr=2110335225 TSecr=226840633</td>
</tr>
<tr>
<td>19</td>
<td>0.099791</td>
<td>10.240.252.16</td>
<td>10.67.118.188</td>
<td>TLSv1.2</td>
<td>308</td>
<td>New Session Ticket, Change Cipher Spec, Encrypted Handshake Message</td>
</tr>
<tr>
<td>20</td>
<td>0.099262</td>
<td>10.240.252.16</td>
<td>10.67.118.188</td>
<td>TLSv1.2</td>
<td>143</td>
<td>Application Data</td>
</tr>
<tr>
<td>21</td>
<td>0.110345</td>
<td>10.67.118.188</td>
<td>10.240.252.16</td>
<td>TCP</td>
<td>66</td>
<td>53282 → 912 [ACK] Seq=2132 Ack=5548 Win=42880 Len=0 TSecr=226840638 TSecr=2110335242</td>
</tr>
<tr>
<td>22</td>
<td>0.116396</td>
<td>10.67.118.188</td>
<td>10.240.252.16</td>
<td>TLSv1.2</td>
<td>104</td>
<td>Application Data</td>
</tr>
<tr>
<td>23</td>
<td>0.123406</td>
<td>10.67.118.188</td>
<td>10.240.252.16</td>
<td>TLSv1.2</td>
<td>1119</td>
<td>Application Data</td>
</tr>
<tr>
<td>24</td>
<td>0.123444</td>
<td>10.67.118.188</td>
<td>10.240.252.16</td>
<td>TLSv1.2</td>
<td>1697</td>
<td>Application Data</td>
</tr>
<tr>
<td>25</td>
<td>0.123466</td>
<td>10.67.118.188</td>
<td>10.240.252.16</td>
<td>TCP</td>
<td>66</td>
<td>53282 → 912 [ACK] Seq=2170 Ack=8232 Win=48768 Len=0 TSecr=226840641 TSecr=2110335256</td>
</tr>
</tbody>
</table>
Test Result

Packet 8
00:01:16:174624: pg-input
  stream cap, 563 bytes, 1 sw_if_index
  current data 0, length 563, buffer-pool 0, ref-count 1
  IP4: 00:1e:67:e6:fe:18 -> 00:00:5e:00:01:01
  TCP: 10.67.118.188 -> 10.240.252.16
    tos 0x00, ttl 64, length 569, checksum 0xa22f
    fragment id 0x0e90, flags DCNT_FRAGMENT
  TCP: 53282 -> 912
    seq. 0x0e375f7c ack 0x2dcbd00a
    flags 0x18 PSH ACK, tcp header: 32 bytes
    window 229, checksum 0x8a2b
00:01:16:174665: ethernet-input
  frame: flags 0x1, hw-if-index 3, sw-if-index 3
  IP4: 00:1e:67:e6:fe:18 -> 00:00:5e:00:01:01
00:01:16:174669: ip4-input
  TCP: 10.67.118.188 -> 10.240.252.16
    tos 0x00, ttl 64, length 569, checksum 0xa22f
    fragment id 0x0e90, flags DCNT_FRAGMENT
  TCP: 53282 -> 912
    seq. 0x0e375f7c ack 0x2dcbd00a
    flags 0x18 PSH ACK, tcp header: 32 bytes
    window 229, checksum 0x8a2b
00:01:16:174679: dpi4-input
  DPI from flow 0 app_id -1 next 0 error 0

Packet 9
00:01:16:174624: pg-input
  stream cap, 1414 bytes, 1 sw_if_index
  current data 0, length 1414, buffer-pool 0, ref-count 1
  TCP: 10.240.252.16 -> 10.67.118.188
    tos 0x00, ttl 56, length 1400, checksum 0x2f5f
    fragment id 0x6621
  TCP: 912 -> 53282
    seq. 0x2dcbd00a ack 0xe376181
    flags 0x10 ACK, tcp header: 32 bytes
    window 4107, checksum 0xa32c
00:01:16:174669: ip4-input
  frame: flags 0x1, hw-if-index 3, sw-if-index 3
00:01:16:174669: ip4-input
  TCP: 10.240.252.16 -> 10.67.118.188
    tos 0x00, ttl 56, length 1400, checksum 0x2f5f
    fragment id 0x6621
  TCP: 912 -> 53282
    seq. 0x2dcbd00a ack 0xe376181
    flags 0x10 ACK, tcp header: 32 bytes
    window 4107, checksum 0xa32c
00:01:16:174679: dpi4-input
  DPI from flow 0 app id 7 next 1 error 0
Key Takeaway

• Provides a flow-based framework for advanced packet processing.

• Supports HW flow offloading and SW flow process.

• Supports Hyperscan Stream Mode.

• Supports TCP segments reassembly on the fly.
Thank you!

Q & A

Email: hongjun.ni@intel.com
      qi.z.zhang@intel.com