Deep Packet Inspection Using GPUs

Qian Gong, Wenji Wu, Phil DeMar
GPU Technology Conference 2017
May 2017
• Main uses for network traffic analysis
  – Operations & management
  – Capacity planning
  – Performance troubleshooting

• Levels of network traffic analysis
  – Device counter level (snmp data)
  – Traffic flow level (flow data)
  – **Packet level** (The focus of this work)
    • Network securities
    • Application performance analysis
    • Traffic characterization studies
Background (cont.)

Characteristics of packet-based network traffic analysis applications

• Time constraints on packet processing

• Computing and I/O throughput-intensive

• High levels of data parallelism
  • Packet parallelism. Each packet can be processed independently
  • Flow parallelism. Each flow can be processed independently

• Extremely poor temporal locality for data
  • Typically, data processed once in sequence; rarely reused
Packet-based traffic analysis tools face performance & scalability challenges within high-performance networks.

- High-performance networks:
  - 40GE/100GE link technologies
  - Servers are 10GE-connected by default
  - 400GE backbone links & 100GE host connections loom on the horizon

- Millions of packets generated & transmitted per sec
Packet-based Traffic Analysis Tool Platform (I)

- Requirements on computing platform for high performance network traffic analysis applications
  - High compute power
  - Ample memory & IO bandwidth
  - Capability of handling data parallelism inherent with network data
  - Easy programmability
Three types of computing platforms:
- NPU/ASIC, CPU, GPU

---

<table>
<thead>
<tr>
<th>Features</th>
<th>NPU/ASIC</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>High compute power</td>
<td>Varies</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>High memory bandwidth</td>
<td>Varies</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Easy programmability</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Data-parallel execution model</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

---

Architecture Comparison

<table>
<thead>
<tr>
<th>Features</th>
<th>cores</th>
<th>Bandwidth</th>
<th>DP</th>
<th>SP</th>
<th>Power</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVidia K80</td>
<td>4992</td>
<td>480 GB/s</td>
<td>2.91 TF</td>
<td>8.73 TF</td>
<td>300W</td>
<td>$4,349</td>
</tr>
<tr>
<td>Intel E7-8890</td>
<td>18</td>
<td>102 GB/s</td>
<td>0.72 TF</td>
<td>1.44 TF</td>
<td>165W</td>
<td>$7,174</td>
</tr>
</tbody>
</table>

NVidia K80 vs. Intel E7-8890
Our Solution

Network Traffic Analysis using GPUs

Highlights of our work:

• Demonstrated GPUs can significantly accelerate network traffic analysis

• Designed/Implemented a generic I/O architecture to capture and move network traffics from wire into GPU domain

• Implemented a GPU-accelerated library for network traffic analysis
GPU-based Network Traffic Analysis Framework

Running modes
- Online analysis
  - Traffic capture
- Offline analysis

GPU-based analysis
- Header analysis
- Payload analysis

Applications
- Network Monitoring
- IPS/IDS
- Traffic Engineering
- And more

Configuration

Network Traffic Source
- Network
- WireCAP Packet Capture Engine
- Storage
- Libpcap library

Online traffic
Offline traffic

GPU Domain
- Filter (BPF)
- Header Parser
- Flow Table (SrcIP, DstIP, SrcPort, DstPort, Proto)
- Traffic Summarization

Payload Analysis
- Pattern Matching
- Header Parsing and/or Assembly (Suspicious packets)
- Abnormal Warning

Applications
- Network Monitoring
- IPS/IDS
- Traffic Engineering

…
System Architecture – Online Analysis

Four types of logical Entities:

- Traffic Capture
- Preprocessing
- GPU-based Analysis
- Output (in JSON format)
WireCAP Packet Capture Engine

• An advanced packet capture engine for commodity network interface cards (NICs) in high-speed networks
  – Lossless zero-copy packet capture and delivery
  – Zero-copy packet forwarding
  – A Libpcap-compatible interface for low-level network access

• WireCAP project website
  – http://wirecap.fnal.gov (Note: source code is available)
GPU-based Network Traffic Analysis

• A GPU-accelerated library for network traffic analysis
  – Dozens of CUDA kernels
  – Can be combined in a variety of ways to perform intended analysis operations

• Two types of GPU-based network traffic analysis
  – Header analysis (see our GTC’13 talk)
  – Packet payload analysis
    • Deep packet analysis (TCP streams)
Challenges in Stream Reassembly (I) --- Parallelism

Why stream reassembly?

- Payload of packet affiliated to the same TCP stream need to be assembled before matching against pre-defined patterns

However...

- Stream reassembly via parallel hash-table requires an atomic lock with each hash key (TCP 4-tuple)
- Limited data parallelism when less simultaneous TCP connections are present
Challenges in Stream Reassembly (II)  
--- Denial of Service Attack

- To address the problem of out-of-order packets, one widely adopted approach is *packet buffering* and *stream reassembly*, i.e., buffer all packets following a missing one, until they become in-sequence again.

- This approach is intuitive but vulnerable to denial-of-service (DoS) attacks, whereby attackers exhaust the packet buffer capacity by sending long segments of out-of-order packets.
GPU-based Deep Packets Analysis Pipeline

Hybrid Pattern Matching Pipeline

- Intra-batch TCP packets reordering & assembly
- Inter-batch split detection

*Pattern matching wo/ buffering or dropping out-of-order packets*
Observation 1

- According to previous internet traffic analysis report, only 2%-5% packets are affected by re-ordering
- When processing packets in batch (~1e6 packets), 0.1%-0.5% TCP streams spread across batches

Mechanism 1 --- **intra-batch stream reassembly**

+ Load packets from network to GPUs in batch
+ In-batch packet reordering and reassembly via parallel sorting
GPU-based TCP Stream Reassembly

Packet Reordering

raw packet

packets in flow and sequence order

flow identifier

next packet array

bytes of overlapping data (prefer new data)

Stream Normalization

sort by (4-tuple|seq #)

filter + scan

scan seq #
Key Mechanisms (II)

Observation 2

- If a string $S$ is matched across a list of packets $P_1P_2...P_N$, the suffix of $P_1$ must match a prefix of $S$, the prefix of $P_N$ must match a suffix of $S$, and $P_2...P_{N-1}$ must match the prefixes of a suffix-$S$.

Mechanism 2 --- **inter-batch split detection**

+ Combine the Aho-Corasick (AC) and suffix-AC automatons to detect signatures spread over different batches
Intra-batch: AC automaton

State transition automaton

Parallel execution mode

Keywords: $X = \{\text{he, his, she, hers}\}$

- One thread per packet
- Each thread scans extra $N$ bytes towards its consecutive packet
Inter-batch: AC automaton & Suffix-AC automaton

Suffix Pattern Tree (PST)

Suffix set of X: \{e,is,s,he,ers,rs\}

```c
struct {
    nextState[256];
    preState;
    preChar;
} PST;
```

Out-of-order Packets

- **case 1**
- **case 2**
- **case 3**
Performance Evaluation

Traffic Statistics
- Traffic source: real traffics mirrored from the Fermilab gateway
- Traffic pattern (average per batch)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># of packets</td>
<td>1 million</td>
</tr>
<tr>
<td># of data packets</td>
<td>776,207</td>
</tr>
<tr>
<td>mean packet length</td>
<td>1415-byte</td>
</tr>
<tr>
<td># of connections</td>
<td>15,500</td>
</tr>
</tbody>
</table>

Base Systems:
- Intel Xeon CPU E5-2650 @ 2.30 GHz, NVIDIA K40

Throughput (wo/ memory transfer)
- TCP reassembly: **72.96** Mpps (**×**192 speedup comparing to libnids on CPU)
- TCP state management: 286.85 Mpps
- Pattern matching (AC & Suffix-AC): 5.83 Mpps
## Comparison to Existing Tools

- Comparison to Snort\(^1\), Split-Detect\(^2\), and GASPP\(^3\)

<table>
<thead>
<tr>
<th></th>
<th>Snort</th>
<th>Split-Detect</th>
<th>GASPP</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Computing platform</strong></td>
<td>CPU</td>
<td>CPU</td>
<td>GPU</td>
<td>GPU</td>
</tr>
<tr>
<td><strong>Methods</strong></td>
<td>Stream Reassembly</td>
<td>Split detection</td>
<td>Intra-batch stream reassembly</td>
<td>In-batch stream reassembly + inter-batch split detection</td>
</tr>
<tr>
<td><strong>Detection over OOO packets</strong></td>
<td>✔</td>
<td>✔</td>
<td>limited</td>
<td>✔</td>
</tr>
<tr>
<td><strong>Resistance to fragmentation flood</strong></td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Throughput</strong></td>
<td>Low</td>
<td>Low</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>

Functionality Evaluation in the Presence of Adversaries

- Robust stream reassembly in facing of out-of-order packets
- Immune to SYN flood and ‘cold start’ in doing normalization
- Exempt from attacks on available buffer memory with timeout and connection evasion mechanisms
Future Works

• Extended the GPU-based deep packet analysis framework to work with regular expressions

• Complement the header info w/ with the payload detection results for a thorough inspection/analysis

• Optimize and evolve the GPU-based network traffic analysis framework for 40GE/100GE networks
Questions?

qgong@fnal.gov, wenji@fnal.gov, demar@fnal.gov