MIDeA: A Multi-Parallel Intrusion Detection Architecture

Giorgos Vasiliadis, FORTH-ICS, Greece
Michalis Polychronakis, Columbia U., USA
Sotiris Ioannidis, FORTH-ICS, Greece

CCS 2011, 19 October 2011
Network Intrusion Detection Systems

• Typically deployed at ingress/egress points
  – Inspect *all* network traffic
  – Look for suspicious activities
  – Alert on malicious actions
Challenges

• **Traffic rates** are increasing
  – 10 Gbit/s Ethernet speeds are common in metro/enterprise networks
  – Up to 40 Gbit/s at the core

• Keep needing to perform *more complex analysis* at *higher speeds*
  – Deep packet inspection
  – Stateful analysis
  – 1000s of attack signatures
Designing NIDS

• Fast
  – Need to handle many Gbit/s
  – Scalable
    • Moore’s law does not hold anymore

• Commodity hardware
  – Cheap
  – Easily programmable
Today: fast or commodity

• Fast “hardware” NIDS
  – FPGA/TCAM/ASIC based
  – Throughput: High

• Commodity “software” NIDS
  – Processing by general-purpose processors
  – Throughput: Low
MIDeA

- A NIDS out of *commodity* components
  - Single-box implementation
  - Easy programmability
  - Low price

*Can we build a 10 Gbit/s NIDS with commodity hardware?*
Outline

• Architecture
• Implementation
• Performance Evaluation
• Conclusions
Single-threaded performance

- Vanilla Snort: 0.2 Gbit/s
Problem #1: Scalability

• Single-threaded NIDS have limited performance
  – Do not scale with the number of CPU cores
Multi-threaded performance

- Vanilla Snort: 0.2 Gbit/s
- With multiple CPU-cores: 0.9 Gbit/s
Problem #2: How to split traffic

- Synchronization overheads
- Cache misses
- Receive-Side Scaling (RSS)
Multi-queue performance

- Vanilla Snort: 0.2 Gbit/s
- With multiple CPU-cores: 0.9 Gbit/s
- **With multiple Rx-queues: 1.1 Gbit/s**
Problem #3: Pattern matching is the bottleneck

Offload pattern matching on the GPU

> 75%

gvasil@ics.forth.gr
Why GPU?

• General-purpose computing
  – Flexible and programmable

• Powerful and ubiquitous
  – Constant innovation

• Data-parallel model
  – More transistors for data processing rather than data caching and flow control
Offloading pattern matching to the GPU

- Vanilla Snort: 0.2 Gbit/s
- With multiple CPU-cores: 0.9 Gbit/s
- With multiple Rx-queues: 1.1 Gbit/s
- **With GPU: 5.2 Gbit/s**
Outline

• Architecture
• Implementation
• Performance Evaluation
• Conclusions
Multiple data transfers

• Several data transfers between different devices

Are the data transfers worth the computational gains offered?
Capturing packets from NIC

- Packets are hashed in the NIC and distributed to different Rx-queues
- Memory-mapped ring buffers for each Rx-queue
CPU Processing

- Packet capturing is performed by different CPU-cores \textit{in parallel}:
  - Process affinity

- Each core \textit{normalizes} and \textit{reassembles} captured packets to streams:
  - Remove ambiguities
  - Detect attacks that span multiple packets

- Packets of the same connection \textit{always} end up to the same core:
  - No synchronization
  - Cache locality

- Reassembled packet streams are then \textit{transferred to the GPU} for pattern matching:
  - \textit{How to access the GPU?}
Accessing the GPU

• Solution #1: Master/Slave model

• Execution flow example

Transfer to GPU:

GPU execution:

Transfer from GPU:
Accessing the GPU

• Solution #2: Shared execution by multiple threads

Thread 1
Thread 2
Thread 3
Thread 4

• Execution flow example

Transfer to GPU:

GPU execution:

Transfer from GPU:

P1  P2  P3  P1  • • •
P1  P2  P3  P1  • • •
P1  P2  P3  P1  • • •
Transferring to GPU

- Small transfer results to PCIe throughput degradation
  ➔ Each core batches many reassembled packets into a single buffer

gvasil@ics.forth.gr
Pattern Matching on GPU

- Uniformly, *one GPU core for each reassembled packet stream*
Pipelining CPU and GPU

• Double-buffering
  – Each CPU core collects new reassembled packets, while the GPUs process the previous batch
  – Effectively hides GPU communication costs
Recap

GPUs:

CPUs:

NIC:

1-10Gbps

Data-parallel content matching

Per-flow protocol analysis

Demux

Packets

Reassembled packet streams

Packet streams

NIC:

CPUs:

GPUs:
Outline

• Architecture
• Implementation
• Performance Evaluation
• Conclusions
Setup: Hardware

- NUMA architecture, QuickPath Interconnect

<table>
<thead>
<tr>
<th>Model</th>
<th>Specs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 x CPU</td>
<td>Intel E5520, 2.27 GHz x 4 cores</td>
</tr>
<tr>
<td>2 x GPU</td>
<td>NVIDIA GTX480, 1.4 GHz x 480 cores</td>
</tr>
<tr>
<td>1 x NIC</td>
<td>82599EB, 10 GbE</td>
</tr>
</tbody>
</table>
The performance of a single GPU increases, as the number of CPU-cores increases.
Pattern Matching Performance

- The performance of a single GPU increases, as the number of CPU-cores increases.
Setup: Network

Traffic Generator/Replayer 10 GbE MIDeA
Synthetic traffic

- Randomly generated traffic

Packet size

- 200b: 1.1 Gbit/s (MIDeA), 1.5 Gbit/s (Snort 8x cores)
- 800b: 2.1 Gbit/s (MIDeA), 4.8 Gbit/s (Snort 8x cores)
- 1500b: 2.4 Gbit/s (MIDeA), 7.2 Gbit/s (Snort 8x cores)
• **5.2 Gbit/s with zero packet-loss**
  – Replayed trace captured at the gateway of a university campus
Summary

• MIDeA: A multi-parallel network intrusion detection architecture
  – Single-box implementation
  – Based on commodity hardware
  – Less than $1500

• Operate on 5.2 Gbit/s with zero packet loss
  – 70 Gbit/s pattern matching throughput
Thank you!

gvasil@ics.forth.gr