LEoNIDS: a Low-latency and Energy-efficient Intrusion Detection System

Nikos Tsikoudis

Thesis Supervisor: Evangelos Markatos

June 2013 Heraklion, Greece
Low-Power Design

• Low-power systems receive significant attention

• Energy efficiency in
  – Datacenters
  – Battery-operated devices
  – Computer networks
Network Intrusion Detection Systems (NIDS)

• Detect security violations
• Secure operation of computer networks
• NIDS utilize multi-core systems or cluster of servers
  – Increased network traffic volumes
  – Heavy computationally operations
Energy versus Performance

• Low-power techniques lead to performance degradation
  – Dynamic Voltage and Frequency Scaling (DVFS)

• NIDS performance factor
  – Detection latency

• Energy-latency tradeoff
Motivation

• NIDS is not often overloaded

• Why power consumption matters
  – Significant concern in data centers
    • Limited power capacity
  – NIDS on battery-operated devices

• Why detection latency matters
  – Fast active reaction and protection
Our Proposed Approach

• Identify the most important packets for attack detection

• Process most important packets with lower latency
  – Priority queue scheduling
  – Dedicated cores for these packets
Environment (1/2)

- 2 x Intel Xeon E5-2620
  - Six core processors

- Intel 82599EB 10GbE NIC
  - RSS feature: splits the traffic across cores

- Every core is assigned with a queue for packet queuing
Environment (2/2)

• Watts up? PRO ES

• Snort IDS
  – One detection process on each core

• Anonymized real traffic
  – 40GB trace, 59M packets, 1.5M flows
  – 1938 alerts, 90 attack signatures
Towards a Power Proportional NIDS
Power consumption

• CPU consumes the larger portion of the energy

<table>
<thead>
<tr>
<th>NIDS utilization</th>
<th>Total power consumption</th>
<th>CPU power consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>109.9 W</td>
<td>63.5 W</td>
</tr>
<tr>
<td>Moderate</td>
<td>129.6 W</td>
<td>78.0 W</td>
</tr>
<tr>
<td>High</td>
<td>145.7 W</td>
<td>90.6 W</td>
</tr>
</tbody>
</table>

• CPU-based low-power techniques
  – Dynamic Voltage and Frequency Scaling (DVFS)
  – Core sleep states (C-states)
Exploring the design space

• Operate at lower frequency with no idle time or utilize sleep states?

• More cores on lower frequency or less cores at higher frequency?
Exploring the design space

![Power consumption vs CPU frequency graph](chart.png)

- 4 active cores
- 6 active cores
- 8 active cores
- 10 active cores
- 12 active cores

0.6 Gbit/sec
Lower frequency or utilize sleep states?
Impact of core utilization

- Power consumption decreases as the core utilization increases
Lower frequency or utilize sleep states?

Operate at the lowest possible frequency with no idle time.
More cores on lower frequency or less cores at higher frequency?

<table>
<thead>
<tr>
<th>Active cores</th>
<th>Frequency</th>
<th>Power consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>2.0 GHz</td>
<td>107.0 W</td>
</tr>
<tr>
<td>8</td>
<td>1.5 GHz</td>
<td>104.2 W</td>
</tr>
<tr>
<td>10</td>
<td>1.2 GHz</td>
<td>100.2 W</td>
</tr>
</tbody>
</table>

1.5GBit/sec
More cores on lower frequency or less cores at higher frequency?

<table>
<thead>
<tr>
<th>Active cores</th>
<th>Frequency</th>
<th>Power consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td></td>
<td>W</td>
</tr>
<tr>
<td>8</td>
<td>1.2 GHz</td>
<td>W</td>
</tr>
<tr>
<td>10</td>
<td>100.2 W</td>
<td>W</td>
</tr>
</tbody>
</table>

1.5GBit/sec

More cores on lower frequency
A straight-forward power-proportional NIDS

• Utilize the smallest number of cores able to sustain the traffic at the lowest possible frequency
  – monitors the queues' utilization
  – adapts the number of cores and the frequency based on thresholds
Adapt to the traffic load

![Graph showing power consumption vs. traffic rate for Original Snort and Power-proportional Snort.](image)
Adapt to the traffic load

![Graph showing power consumption vs traffic rate. The graph compares power consumption for Original Snort and Power-proportional Snort. There is a 23% difference between the two lines.]
NIDS Performance
Detection latency

• Alert trigger timestamp – packet capture timestamp
  – Queuing delay
  – Processing time

• A high detection latency makes the NIDS reaction pointless
Detection latency

- Exponential increase when core utilization exceeds 70%

0.6 Gbit/sec
Energy – Latency tradeoff

Up to 7x increase for power lower than 100W

0.6 Gbit/sec
Deconstructing Detection Latency

![Graph showing the deconstruction of detection latency with bars representing processing time and queuing delay by traffic rate and frequency.]
Deconstructing Detection Latency

Queuing delay is the main factor of the increased detection latency
Detection latency of power-proportional system

- Power proportional system has close to 100% utilization at every core
Solving the Energy-Latency Tradeoff
Key Idea

• Identifying the most important packets

• Ensure low latency for them

• Small percentage of packets with higher probability to contain an attack
  – First few packets of each connection
    • Brute force attacks, port scanning, code-injection attacks
Identifying the Most Important Packets

The graph shows the cumulative distribution function (CDF) of packet ranks and the fraction of packets. The red line indicates attacks detected, while the green dashed line represents the fraction of packets. The x-axis represents packet rank, and the y-axis represents CDF. The graph includes an inset that zooms into the lower portion of the main graph.
Identifying the Most Important Packets

50% attacks within the first 10 packets of each flow
Identifying the Most Important Packets

90% attacks within the first 100 packets of each flow
Identifying the Most Important Packets

1% beyond the first 800 packets of a flow
Identifying the Most Important Packets
Identifying the Most Important Packets

10% of the total packets contain 90% of the total attacks
Resolving Energy-Latency Tradeoff

• We propose two alternative techniques
  – Time sharing
    • Priority queue scheduling
  – Space sharing
    • Dedicated cores with lower utilization
Implementation

- Techniques are implemented within the capturing subsystem as kernel module
Time Sharing

- Classifies packet into transport-layer flows
- Assign low and high priority according to a flow cutoff value
- Uses the strategy described for power-proportional system
Space Sharing

- Separate cores for each priority
  - based on a *flow cutoff value*
- Flow migration
- Adaptive core management
- Keep high-priority cores less utilized
Adapting the Number of Active Cores
Experimental Evaluation

• Evaluate the alternative approaches to find out and optimal cutoff
  – Low cutoff values result in more attacks in low-priority packets
  – High cutoff values result in increased fraction of high-priority packets with more benign packets

• Compare all approaches
Time Sharing

![Graph showing detection latency over cutoff (packets per flow)]
Time Sharing

![Graph showing latency vs. cutoff (packets per flow)]

- Queuing delay – high-priority packets
- Processing time – high-priority packets
- Queuing delay – low-priority packets
- Processing time – low-priority packets

1.0 Gbit/sec
Time Sharing

1.0 Gbit/sec

Latency (ms)

Cutoff (packets per flow)

Queueing delay – high-priority packets
Processing time – high-priority packets
Queueing delay – low-priority packets
Processing time – low-priority packets

49x
Time Sharing

1.0 Gbit/sec
Time Sharing

1.0 Gbit/sec
Space Sharing

![Graph showing detection latency vs. cutoff (packets per flow) with a 1.0 Gbit/sec marker.](image)
Space Sharing
Space Sharing Performs Better

- Low- and high-priority packets are processed in parallel
- We keep high-priority cores less utilized
Comparison of all Approaches

Graph showing the comparison of detection latency and power consumption of different approaches with varying traffic rates. The approaches include Original Snort, Power-proportional Snort, LEO-NIDS with time sharing, and LEO-NIDS with space sharing.
Comparison of all Approaches

More than 40%
Comparison of all Approaches

![Comparison Graph]

- Original Snort
- Power-proportional Snort
- LEO NIDS with time sharing
- LEO NIDS with space sharing

22% reduction in power consumption
Conclusions

• Energy-efficiency in NIDS
• Energy-latency tradeoff
• Identify most important packet
• LEoNIDS: Low-latency, Energy-Efficient NIDS
  – Space sharing
Thank you
Back up slides
More cores on lower frequency or less cores at higher frequency?
More cores on lower frequency or less cores at higher frequency?

![Graph showing power consumption vs CPU frequency for different numbers of active cores.](image)

- 6 cores, 1.2 GHz, 95.9W
- 4 cores, 1.8 GHz, 98.5W
Adapt to the traffic load

![Graph showing power consumption vs traffic rate. The graph compares Original Snort and Power-proportional Snort. The red line represents Original Snort, and the dashed green line represents Power-proportional Snort. A red arrow labeled 24% points to the difference in power consumption between the two lines at a certain traffic rate. The x-axis represents traffic rate (Gbit/sec), and the y-axis represents power consumption (W).]
Adapt to the traffic load
Time Sharing

1. The system starts with a single active core at the minimum frequency while the rest cores are in C6 state.

2. It continuously monitors the queues’ usage.

   2.1. If queues are filled by more than a high threshold:
   
      2.1.1. If there are inactive cores, it wakes up one more core uses one more queue.
      
      2.1.2. Else, it increases the frequency of all cores to the next step.

    2.2. If queues are filled by less than a low threshold:

      2.2.1. If the system uses the lowest frequency, it deactivates one core.
      
      2.2.2. Else, it decreases the frequency.
1. The system starts with one high-priority core and one low-priority core.

2. It continuously monitors the queues’ usage.

   2.1. If high-priority queues are filled by more than a high-priority up threshold:
       2.1.1. If there are inactive cores, activate another high-priority core.
       2.1.2. Else increase the CPU frequency.
       2.1.3. If maximum frequency is used, reduce flow cutoff (so less packets will be considered as high priority) until it reaches a certain limit.

   2.2. If high-priority queues are filled by less than a high-priority down threshold:
       2.2.3. Increase cutoff up to a certain limit.
       2.2.1. Else reduce the CPU frequency.
       2.2.2. If the lowest frequency is used, deactivate a high-priority core.

2.3. If low-priority queues are filled by more than a low-priority up threshold:
    2.3.1. If there are inactive cores, activate another low-priority core.
    2.3.2. Else increase the CPU frequency.

2.4. If low-priority queues are filled by less than a low-priority down threshold:
    2.4.1. Reduce the CPU frequency.
    2.4.2. In case of the lowest frequency, deactivate a low-priority core.
Space Sharing Performs Better

![Graph showing the comparison between different packet processing metrics and cutoffs. The graph compares queuing delay and processing time for high and low priority packets across various cutoffs.]