Introduction to Intrusion Detection Systems

Firewall/Intrusion Detection System

Protected Intranet

Unprotected Internet

Quarantined Packets
What is Intrusion Detection?

• All incoming packets are filtered for specific characteristics or content
• Databases have thousands of patterns requiring string matching
  – FPGA allows fine-grained parallelism and computational reuse
• 10 Gb/s and higher rates desired
  – Provided by pipelined, streaming architectures
Other Approaches

- Objective: find all occurrences of a pattern in an input
- Naïve approach: $O((n-m+1)m)$
- Shift-and-compare: $O(n)$, large hardware requirements, $O(nm)$ work
- Hashing: $O(n)$, hashing can be complex, $O(nm)$ work
- KMP: $O(n)$: other algorithms may be faster in practice, but do not provide low precise upper bound $(2n - m)$, $O(n+m)$ work
High-Performance Shift-and-Compare Architectures

Various contributions to shift-and-compare architectures:

- Pre-decoded architecture provides significant area and routing improvements over encoded data

- Graph-based partitioning of patterns allows for reduced routing complexity and increased frequency performance through multiple pipelines

  - Average of 15% decrease in area, 5% decrease in clock period over unpartitioned unary
Reduction of Resource Usage

• Trie-based prefix grouping allows for reduced area consumption through lower redundant comparisons
  • 4-byte prefixes turn out to be very appropriate for intrusion detection:
    /cgi-bin/bigconf.cgi
    /cgi-bin/common/listrec.pl
    /cgi-sys/addalink.cgi
    /cgi-sys/entropysearch.cgi
• Replication of hardware and delays allow for multi-byte per cycle throughput at high clock rates
  • Pipeline is not increased in size – large source of slice consumption
  • Front end decoders increases in size by $k$
  • Back end matchers increase in size by $k$
<table>
<thead>
<tr>
<th></th>
<th>1 way</th>
<th>4 way</th>
<th>8 way</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of Slices</strong></td>
<td>299</td>
<td>721</td>
<td>1338</td>
</tr>
<tr>
<td><strong>Clock Period</strong></td>
<td>4.2ns</td>
<td>4.6ns</td>
<td>5.3ns</td>
</tr>
<tr>
<td><strong>Throughput</strong></td>
<td>1.9Gb/s</td>
<td>6.9Gb/s</td>
<td>12.1Gb/s</td>
</tr>
<tr>
<td><strong>Efficiency</strong></td>
<td>1</td>
<td>1.51</td>
<td>1.41</td>
</tr>
</tbody>
</table>

*Efficiency in throughput/area, normalized to 1-way (~100 rules)*
Customized Performance

• Variations in tool flow provide customizable performance:
  – Tool Options
    • Small: partitioned and pre-decoded architecture
      – Prefix trees
    • Fast: $k$-way architecture
    • Fast reconfiguration, minimum complexity
      – KMP architecture
### Comparison of Related Architectures

<table>
<thead>
<tr>
<th>Design</th>
<th>Throughput</th>
<th>Unit Size</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>USC Unary</td>
<td>2.1 Gb/s</td>
<td>7.3</td>
<td>283</td>
</tr>
<tr>
<td>USC Unary (1 byte)</td>
<td>1.8 Gb/s</td>
<td>5.7</td>
<td>315</td>
</tr>
<tr>
<td>USC Unary (4 byte)</td>
<td>6.1 Gb/s</td>
<td>22.3</td>
<td>271</td>
</tr>
<tr>
<td>USC Unary (8 byte)</td>
<td>10.3 Gb/s</td>
<td>32</td>
<td>322</td>
</tr>
<tr>
<td>USC Unary (Prefilter)</td>
<td>6.4 Gb/s</td>
<td>9.4</td>
<td>682</td>
</tr>
<tr>
<td>USC Unary (Tree)</td>
<td>2.0 Gb/s</td>
<td>6.6</td>
<td>303</td>
</tr>
<tr>
<td>Los Alamos (FPL '03)</td>
<td>2.2 Gb/s</td>
<td>243</td>
<td>9.1</td>
</tr>
<tr>
<td>UCLA (FPL '02)</td>
<td>2.9 Gb/s</td>
<td>160</td>
<td>18</td>
</tr>
<tr>
<td>UCLA w/Reuse (FCCM '04)</td>
<td>3.2 Gb/s</td>
<td>11.4</td>
<td>280</td>
</tr>
<tr>
<td>U/Crete (FPL '03)</td>
<td>10.8 Gb/s</td>
<td>269</td>
<td>40.1</td>
</tr>
<tr>
<td>U/Crete (FCCM '04)</td>
<td>9.7 Gb/s</td>
<td>57</td>
<td>170</td>
</tr>
<tr>
<td>GATech (FCCM '04)</td>
<td>7.0 Gb/s</td>
<td>50</td>
<td>140</td>
</tr>
</tbody>
</table>

* Throughput is assumed to be constant over variations in pattern size. Unit size is the average unit size for a 16 character pattern (in logic cells; one slice is two logic cells), and performance is defined as Mb/s/cell.)
Incremental Architecture Synthesis

- Goal: Reduce place and route costs
- Cost for changing rules in one of $k$ partitions: $\text{overhead} + 1/k$
- Key: Predefinition of area constraints
Determining the Optimal Partition

\[ \delta_i = \left( S_{p^*} \setminus P_i \right) \]

find \( j \) such that \( |\delta_j| = \min_{i=0}^{P} |\delta_i| \)

characters to add to partition \( j \) are in \( \delta_j \)

Definitions:

- \( S_{p^*} \) is the set of characters required to represent the new pattern \( p^* \).
- The set difference between the characters currently represented in \( P_i \) and the characters that are present in \( S_{p^*} \) is \( \delta_j \).
- The partition which will require the addition of the minimum number of new characters is the optimal partition \( P_j \).
- The optimal partition is selected from the set of partitions \( P \).
Relevant Publications


Additional publications: http://ceng.usc.edu/~prasanna