Packet Classification and TCAMs

Jon Turner
Computer Science & Engineering
Washington University

www.arl.wustl.edu/~jst
Packet Classification

- General router mechanism with several uses
  - security firewalls, intrusion detection systems
  - network address translation
  - load balancing for large web sites
  - special handling of selected flows or groups of flows

- Common form of packet filter based on IP 5-tuple
  - source address, destination address prefixes
  - protocol field - may be specified as wildcard
  - src port number, dest port number range (for TCP, UDP)

- No ideal solution for general case
  - exhaustive search - limited to slow links, few filters
  - ternary content-addressable memory (TCAM)
    - only effective choice in practice, for high speed links
    - drawbacks include cost, power, port range handling
### Packet Classification in IPv4

- **Example header**
  - 010011, 010100, udp, 11, 3
  - matches 3 and 5
  - first match is 3

- **Each filter has associated result field**
  - e.g. default, discard, send to output 5 using queue 37

- **General case useful mainly for security apps**
  - number of filters fairly limited (hundreds to thousands)
  - flow classification involves exact match on all fields
    - can be many filters, but can implement using hashing

<table>
<thead>
<tr>
<th>src addr</th>
<th>dest addr</th>
<th>proto</th>
<th>src ports</th>
<th>dest ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001*</td>
<td>101*</td>
<td>udp</td>
<td>2-6</td>
<td>*</td>
</tr>
<tr>
<td>10*</td>
<td>1*</td>
<td>tcp</td>
<td>*</td>
<td>3-10</td>
</tr>
<tr>
<td>*</td>
<td>0101*</td>
<td>*</td>
<td>5-12</td>
<td>2-3</td>
</tr>
<tr>
<td>0100*</td>
<td>001*</td>
<td>icmp</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>010*</td>
<td>*</td>
<td>udp</td>
<td>10-20</td>
<td>*</td>
</tr>
<tr>
<td>1011*</td>
<td>1100*</td>
<td>tcp</td>
<td>15-30</td>
<td>0-8</td>
</tr>
<tr>
<td>*</td>
<td>10*</td>
<td>*</td>
<td>3</td>
<td>*</td>
</tr>
</tbody>
</table>
Geometric Perspective

- Each filter is a rectangular subspace of $R^d$
- Objective is to find the highest priority filter containing the query point
TCAM

- Maintains set of (key,value) pairs
  - keys may have wildcards (*) in any bit position
    - implemented using "mask bits"
  - comparison circuits in each word check that all bits match
  - priority encoder selects first match
- CMOS bit uses 16 transistors vs. 6 for SRAM
  - but price per bit can be 5-10x
- Relatively high power consumption
  - all storage bits active during a lookup
  - in SRAM, only one row of storage array active

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0<em>001</em>10</td>
<td>drop</td>
</tr>
<tr>
<td>01<em>01</em>10</td>
<td>q6</td>
</tr>
<tr>
<td>***01110</td>
<td>q2</td>
</tr>
<tr>
<td>1101****</td>
<td>q8</td>
</tr>
<tr>
<td>0<strong>10</strong>1</td>
<td>default</td>
</tr>
<tr>
<td>*1001**0</td>
<td>default</td>
</tr>
</tbody>
</table>

query: 01001110
matches rows 1, 2, 5
result: q6
Using TCAMS for Prefix Matching

- Store prefixes by length
  » so first match is longest match
- To add new prefix, shift others
  » $O(n)$ time to shift all
- Sufficient to move one prefix of each length
  » since equal length prefixes can be in any order
  » so 32 steps to add/remove IPv4 prefix
- TCAM expensive option for prefix matching
  » makes sense only if TCAM is already available
# TCAMs for Packet Classification

## General approach
- Store in priority order, first match is result
- Insertion requires moving filters
- Need not move all

## Range Conversion
- Encode ranges as multiple filters
- Worst-case 30 filters per range, 900 per range pair
- Limited use of ranges means typical impact much smaller

<table>
<thead>
<tr>
<th>src addr</th>
<th>dest addr</th>
<th>proto</th>
<th>src ports</th>
<th>dest ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001*</td>
<td>101*</td>
<td>udp</td>
<td>2-6</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0001*</td>
<td>0010*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0010*</td>
<td>0010*</td>
</tr>
<tr>
<td>10*</td>
<td>1*</td>
<td>tcp</td>
<td>*</td>
<td>3-10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>00011</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0011*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0100*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>01010</td>
</tr>
<tr>
<td>*</td>
<td>0101*</td>
<td>*</td>
<td>5-12</td>
<td>2-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4 prefixes</td>
<td>1 prefix</td>
</tr>
<tr>
<td>0100*</td>
<td>001*</td>
<td>icmp</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110*</td>
<td>*</td>
<td>udp</td>
<td>10-20</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4 prefixes</td>
<td></td>
</tr>
<tr>
<td>1011*</td>
<td>1100*</td>
<td>tcp</td>
<td>15-30</td>
<td>0-8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5 prefixes</td>
<td>x2 prefixes</td>
</tr>
<tr>
<td>*</td>
<td>10*</td>
<td>*</td>
<td>3</td>
<td>*</td>
</tr>
</tbody>
</table>
Reducing Ranges

- Range sets define set of disjoint primitive ranges
  - minimal set with a subset for each original range
  - e.g. primitive ranges for range set
    - 0-10, 4-10, 7-13, 7-15, 14-20
    - 0-3, 4-6, 7-10, 11-13, 14-15, 16-20
  - rewrite original ranges as primitive ranges
    - \( r_0-r_2, r_1-r_2, r_2-r_3, r_2-r_4, r_4-r_5 \)
  - encode ranges of primitive-range indices as prefixes
    - e.g. \( r_0-r_2 \) is encoded as pair of prefixes \( 00\star,010 \)
    - usually need fewer prefixes for new ranges (15 vs. 8)

- Primitive range table needed to classify packets
  - lookup port numbers in table to get their primitive range
  - use primitive range numbers in TCAM lookup
Better Range Encoding

- Can reduce required number of prefixes by encoding integers with non-standard bit patterns

  e.g. \[
  \begin{array}{cccccccc}
  0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
  0000 & 0001 & 0101 & 0111 & 1110 & 1010 & 1000 & \\
  \end{array}
  \]

  yields range encodings

  \[
  \begin{array}{cccccc}
  0-1 & 0-2 & 0-3 & 1-2 & 1-3 & 1-4 & 2-3 & 2-4 & 2-5 & \ldots \\
  000* & 0*0* & 0*** & 0*01 & 0***1 & ***1 & 01*1 & *1*1 & *1** & \\
  \end{array}
  \]

- based on grey-scale encoding construct using Karnaugh map

- longer ranges can be constructed using pairs of ranges
  - so get 1-6 using pair 0**1,11**
Other Issues with TCAMs

- Adding and removing filters
  - Position of filter in table determines selection
  - Adding new filter may require moving overlapping filters

<table>
<thead>
<tr>
<th>src addr</th>
<th>dest addr</th>
<th>proto</th>
<th>src ports</th>
<th>dest ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 10*</td>
<td>1*</td>
<td>tcp</td>
<td>*</td>
<td>3-10</td>
</tr>
<tr>
<td>6 1011*</td>
<td>1100*</td>
<td>tcp</td>
<td>*</td>
<td>15-30</td>
</tr>
<tr>
<td>7 *</td>
<td>*</td>
<td>tcp</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

  - e.g. 2 overlaps 6, 7 but 6 does not overlap 7

  - some TCAMs allow wildcards in queries, and multiple matches, enabling fast identification of overlaps

- Partitioning TCAMS to save power
  - some TCAMs allow searches in sections to save power
  - requires knowing which section to search
    - can use one section as index to guide second-level search