Real-Time Classification of Multimedia Traffic using FPGA

Weirong Jiang\textsuperscript{1} and Maya Gokhale\textsuperscript{2}
\textsuperscript{1}University of Southern California
\textsuperscript{2}Lawrence Livermore National Laboratory
August 31, 2010

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.
Background

- Internet Traffic Classification
  - Identify the application type of a packet / flow
    - e.g. WWW, Email, FTP, Instant Messaging, Skype, IPTV, ...
  - Traffic Engineering, QoS, Security, …

- Multimedia Applications
  - Difficult to identify
    - Random port (P2P), encrypted payload (Skype), etc.
Existing Methods

- **Port-based**
  - Traditional method, e.g. WWW: 80
  - **Fail** for applications using random port numbers

- **Deep Packet Inspection**
  - Pattern matching in packet payload
  - **Fail** for encrypted payload

- **Host Behavior**
  - Connection patterns between hosts
  - Topology / Traffic –dependent
  - May not capture packets from both sides of connection

- **Machine Learning**
  - active research in ML approaches to traffic classification
  - few ML hardware acceleration approaches
    - C4.5 decision trees: 8Mpackets/s on NetFPGA
Statistical Traffic Classification

- Existing efforts
  - Feature selection: packet fields, flow duration
  - Machine learning algorithms
  - Focus on accuracy and robustness

- Challenge
  - Computation complexity in both training and testing
  - State size (memory requirements)
  - Achieve real-time classification

- Our goal
  - High accuracy: fraction of packets correctly classified
  - Low memory requirements
  - High classification throughput
    - Packets per second
  - Dynamic update
    - Online training
    - Intermix training and data packets
Data sets

- Data sets contain IP traffic for three categories of multimedia application
  - VoIP, IM, IPTV from http://tstat.tlc.polito.it/traces.shtm

<table>
<thead>
<tr>
<th>Application</th>
<th>Traces</th>
<th>Start date &amp; time</th>
<th>Duration</th>
<th>IP protocol</th>
<th># packets</th>
<th>Data size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skype</td>
<td>Skype1</td>
<td>2006-05-29 02:18:41</td>
<td>95 hour 26 min</td>
<td>TCP</td>
<td>2357997</td>
<td>338.5</td>
</tr>
<tr>
<td></td>
<td>Skype2</td>
<td>2006-05-29 02:01:25</td>
<td>95 hour 45 min</td>
<td>UDP</td>
<td>39627543</td>
<td>8396.8</td>
</tr>
<tr>
<td></td>
<td>Skype3</td>
<td>2006-05-29 02:49:20</td>
<td>79 hour 3 min</td>
<td>UDP</td>
<td>3049284</td>
<td>231.3</td>
</tr>
<tr>
<td>Instant Mess.</td>
<td>MSN</td>
<td>2006-05-29 02:01:25</td>
<td>95 hour 45 min</td>
<td>TCP</td>
<td>15434573</td>
<td>2234.3</td>
</tr>
<tr>
<td>(IM)</td>
<td>YMSG</td>
<td>2006-05-29 02:01:26</td>
<td>95 hour 45 min</td>
<td>TCP &amp; UDP</td>
<td>841221</td>
<td>79.1</td>
</tr>
<tr>
<td></td>
<td>XMPP</td>
<td>2006-05-29 02:01:25</td>
<td>95 hour 45 min</td>
<td>TCP</td>
<td>214636</td>
<td>34.8</td>
</tr>
<tr>
<td>IPTV</td>
<td>IPTV</td>
<td>2008-05-06 06:19:42</td>
<td>5 min 32 sec</td>
<td>UDP</td>
<td>13513514</td>
<td>18633.8</td>
</tr>
<tr>
<td>Legacy</td>
<td>WIDE-2000</td>
<td>2000-01-01 13:59:00</td>
<td>1 hour 35 min</td>
<td>TCP &amp; UDP</td>
<td>2095192</td>
<td>1052.5</td>
</tr>
</tbody>
</table>

- We use packet level features only
- IP protocol, packet size, TCP/UDP ports, TCP flags
Machine Learning Algorithm

- **k Nearest Neighbors (k-NN)**
  - Supervised learning
  - Assigns to a test instance the majority class type of its k nearest neighbors.

- Keep sorted list of k nearest neighbors
- Distance measure can be
  - Euclidean
  - Manhattan
  - Hamming
k-NN characteristics

- Desirable Properties
  - Proved high accuracy in traffic classification
  - Simple implementation
  - Low training cost
    - No training needed for the basic k-NN!
    - Fast update

- Challenges
  - High computation complexity (Low throughput)
    - Given $N$ training samples,
    - It takes $O(N)$ time to classify each test instance
    - e.g. $\sim 1.6 \text{ sec}$ for classifying a test instance against 100K training samples in our experiments
  - Potentially high memory requirements
K-nn algorithm

- R = training set, r in R, q is a test instance

1. Initialize a k-entry list L = {} to hold the k training samples with smallest distance to q
2. For each r in R
   1. Compute d(q,r) distance between q and r
   2. Insert r into L in sorted order; truncate L to length k
3. Assign q the label of the majority among the k training samples in L
Locality-Sensitive Hashing (LSH)

- Proposed in 1998
  - Approximate k-NN in high dimensional spaces

- Basic idea
  - Hash the data points using multiple hash functions;
  - for each function $h(.)$: if $d(p, q) < d(p', q')$,
    \[ \Pr(h(p) = h(q)) > \Pr(h(p') = h(q')) \]

- Reduced computation complexity
  - # of hash tables = $H$
  - $O(N) \Rightarrow O(H)$ time for classifying each test instance
LSH Function

- Distance function: Hamming distance

Example
- \( p = 0101, q = 1101 \)
- \( p' = 0001, q' = 1111 \)
- Distance
  - \( d(p, q) = 1 \)
  - \( d(p', q') = 3 \)

- Hash function \( h \): random single bit selection
  - \( \Pr(h(p) = h(q)) = 1 - \frac{d(p, q)}{4} = \frac{3}{4} \)
  - \( \Pr(h(p') = h(q')) = 1 - \frac{d(p', q')}{4} = \frac{1}{4} \)

- So with hamming distance and random bit selection,
  - \( \Pr(h(p) = h(q)) > \Pr(h(p') = h(q')) \)
K-nn with LSH algorithm

Training
1. Choose H LSH functions $g_i$, $i = 1, \ldots, H$
2. Construct H hash tables $h_i$
3. For each $r$ in $R$
   1. For $i = 1$ to $H$
      1. $h_i[g_i(r)] = r$

Testing
1. Initialize a k-entry list $L = \{}$ to hold the k training samples with smallest distance to $q$
2. For $i = 1$ to $H$
   1. $r = h_i[g_i(q)]$
   2. compute $d(r, q)$
   3. Insert $r$ into $L$ in sorted order; truncate $L$ to length $k$
3. Assign $q$ the label of the majority among the k training samples in L
Performance Evaluation

- Data Set
  - Skype
  - Instant messaging (IM): MSN, Yahoo, Gtalk, ...
  - IPTV
  - Legacy applications: WWW, Email, DNS, ...

- Features
  - IP protocol (8 bits)
  - packet size (16 bits)
  - TCP/UDP ports (16 bits)
  - TCP flags (8 bits)

- Performance metrics
  - Accuracy
    - # of correctly classified packets / # of all packets
  - Throughput
    - # of packets per second (PPS)
Software Implementation

<table>
<thead>
<tr>
<th># of Training samples</th>
<th>Classification accuracy</th>
<th>Classification time per test instance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>k-NN</td>
<td>LSH</td>
</tr>
<tr>
<td>100</td>
<td>85.48%</td>
<td>81.24%</td>
</tr>
<tr>
<td></td>
<td>2.11 msec</td>
<td>0.023625 msec</td>
</tr>
<tr>
<td>1K</td>
<td>87.23%</td>
<td>87.24%</td>
</tr>
<tr>
<td></td>
<td>15.15 msec</td>
<td>0.024625 msec</td>
</tr>
<tr>
<td>10K</td>
<td>99.90%</td>
<td>99.79%</td>
</tr>
<tr>
<td></td>
<td>164.04 msec</td>
<td>0.025500 msec</td>
</tr>
<tr>
<td>100K</td>
<td>100%</td>
<td>99.97%</td>
</tr>
<tr>
<td></td>
<td>1568.11 msec</td>
<td>0.025750 msec</td>
</tr>
</tbody>
</table>

- Corresponding maximum throughput: 40 KPPS
Hardware Accelerator

- Parallel & pipelined architecture
- Pipelined bitonic sorting network

- Parameterized architecture
  - $k$, # hash tables $H$, hash table size $M$
Number of hash tables

- More hash tables gives better accuracy and correspondingly larger resource utilization

### Impact of the number of hash tables

<table>
<thead>
<tr>
<th># of Hash tables</th>
<th>Overall accuracy</th>
<th>Slice usage</th>
<th>BRAM usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>73.58%</td>
<td>1%</td>
<td>3%</td>
</tr>
<tr>
<td>4</td>
<td>97.56%</td>
<td>5%</td>
<td>9%</td>
</tr>
<tr>
<td>8</td>
<td>99.56%</td>
<td>13%</td>
<td>20%</td>
</tr>
<tr>
<td>12</td>
<td>99.83%</td>
<td>22%</td>
<td>30%</td>
</tr>
<tr>
<td>16</td>
<td>99.97%</td>
<td>27%</td>
<td>40%</td>
</tr>
</tbody>
</table>
Hash table size

- Larger hash table increases accuracy
- No impact on BRAM usage until 2K due to fixed BRAM size allocations

### Impact of Hash Table Size

<table>
<thead>
<tr>
<th>Hash table size</th>
<th>Overall accuracy</th>
<th>Slice usage</th>
<th>BRAM usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>99.34%</td>
<td>28%</td>
<td>40%</td>
</tr>
<tr>
<td>512</td>
<td>99.83%</td>
<td>29%</td>
<td>40%</td>
</tr>
<tr>
<td>1K</td>
<td>99.97%</td>
<td>27%</td>
<td>40%</td>
</tr>
<tr>
<td>2K</td>
<td>99.98%</td>
<td>29%</td>
<td>80%</td>
</tr>
</tbody>
</table>
FPGA Implementation

- Parameters used: $k=2$, $H=16$, $M=1K$
- Xilinx Virtex 5 xc5vlx50t with -1 speed grade on a Xilinx ML555 development board

<table>
<thead>
<tr>
<th></th>
<th>Available</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Slices</td>
<td>7200</td>
<td>27%</td>
</tr>
<tr>
<td># of IOs</td>
<td>480</td>
<td>25%</td>
</tr>
<tr>
<td># of BRAMs</td>
<td>60</td>
<td>43%</td>
</tr>
<tr>
<td>Total memory</td>
<td>2160 Kb</td>
<td>40%</td>
</tr>
<tr>
<td>Clock rate</td>
<td></td>
<td>125 MHz</td>
</tr>
</tbody>
</table>

- Throughput (w/ dual-port RAM):
  - Classification: 250 MPPS
    - **80 Gbps** for minimum size (40 bytes) packets
  - Training (Update): 125 MPPS = 40Gbps
Performance Summary

- Comparison

<table>
<thead>
<tr>
<th></th>
<th>k-NN (SW)</th>
<th>LSH (SW)</th>
<th>LSH (FPGA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>100 %</td>
<td>99.97%</td>
<td>99.97%</td>
</tr>
<tr>
<td>Throughput</td>
<td>0.6 PPS</td>
<td>40K PPS</td>
<td>250M PPS</td>
</tr>
<tr>
<td>Speedup</td>
<td>1</td>
<td>6.7×10⁴</td>
<td>4.2×10⁸</td>
</tr>
</tbody>
</table>

- PPS: # of packets per second
- Latency improvement
  - 6×10⁴
  - 1.5×10⁷
Concluding Remarks

- Statistical traffic classification is promising for classifying multimedia traffic
  - Attracted a lot of research interests recently
- Real-time performance demanded (*Yet little work has been done!*)
  - Algorithm-level optimization
  - Hardware accelerators
- Our contributions
  - Propose using LSH for high-speed traffic classification
  - First 10+ Gbps FPGA design for traffic classification
  - Achieving high accuracy and high throughput
- Open Problems
  - Other machine learning algorithms
  - Other packet/flow/network-level features
  - Other hardware accelerators (multi-core, many-core, ...)

USC Viterbi School of Engineering
Q & A

- Thanks