ECEN 689
Special Topics in Data Science for Communications Networks

Nick Duffield
Department of Electrical & Computer Engineering
Texas A&M University

Lecture 15
Finding Heavy Hitters

- Heavy hitters = frequent items by weight
- Set of keyed weights
  - \((k_i, x_i), i = 1, 2, \ldots, n;\)
  - Total weight \(X = \sum_{i=1, \ldots, n} x_i\)
  - Key aggregates \(X(k) = \sum_{i: k(i)=k} x_i\)
- \(\phi\)-Heavy Hitter (\(\phi\)-HH)
  - any key whose aggregate is at least a fraction \(\phi\) of total
  - \(\phi\)-HH = \(\{k: X(k) \geq \phi X\}\)
- Challenges
  - Fast, small space, close approximation
  - Example: using count-min sketch
Finding Heavy Hitters in Internet Traffic

• Traffic anomalies are common
  – DDoS attacks, Flash crowds, worms, failures

• Traffic anomalies are complicated
  – Multi-dimensional ➔ may involve multiple header fields
    • E.g. src IP 1.2.3.4 AND port 1214 (KaZaA)
    • Looking at individual fields separately is not enough!
  – Hierarchical ➔ Evident only at specific granularities
    • E.g. 1.2.3.4/32, 1.2.3.0/24, 1.2.0.0/16, 1.0.0.0/8
    • Looking at fixed aggregation levels is not enough!
Challenges for traffic anomaly detection

• Immense data volume (esp. during attacks)
  – Prohibitive to inspect all traffic in detail
• Multi-dimensional, hierarchical traffic anomalies
  – Prohibitive to monitor all possible combinations of
different aggregation levels on all header fields
• Sampling (packet level or flow level)
  – May wash out some details
• False alarms
  – Too many alarms = info “snow” → simply get ignored
• Root cause analysis
  – What do anomalies really mean?
Implementation Considerations

• Offline vs. streaming
  – If offline: can use multiple passes for better accuracy
  – Streaming: need to adaptive find HH prefixes

• Sampling vs. Exact
  – E.g. Sampled Netflow Records
  – Need to accommodate estimation inaccuracy in HH thresholding
Looking at traffic aggregates

- Aggregating on individual packet header fields gives useful results but:
  - Traffic reports are not always at the right granularity (e.g., individual IP address, subnet, etc.)
  - Cannot show aggregates defined over multiple fields (e.g., which network uses which application)
- The traffic analysis tool should automatically find aggregates over the right fields at the right granularity.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Destination IP</th>
<th>Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>jeff.dorm.bigU.edu</td>
<td>11.9%</td>
</tr>
<tr>
<td>2</td>
<td>tracy.dorm.bigU.edu</td>
<td>3.12%</td>
</tr>
<tr>
<td>3</td>
<td>risc.cs.bigU.edu</td>
<td>2.83%</td>
</tr>
</tbody>
</table>

Most traffic goes to the dorms ...

<table>
<thead>
<tr>
<th>Rank</th>
<th>Source port</th>
<th>Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Web</td>
<td>42.1%</td>
</tr>
<tr>
<td>2</td>
<td>Kazaa</td>
<td>6.7%</td>
</tr>
<tr>
<td>3</td>
<td>Ssh</td>
<td>6.3%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rank</th>
<th>Destination network</th>
<th>Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>library.bigU.edu</td>
<td>27.5%</td>
</tr>
<tr>
<td>2</td>
<td>cs.bigU.edu</td>
<td>18.1%</td>
</tr>
<tr>
<td>3</td>
<td>dorm.bigU.edu</td>
<td>17.8%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rank</th>
<th>Source port</th>
<th>Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Web</td>
<td>42.1%</td>
</tr>
<tr>
<td>2</td>
<td>Kazaa</td>
<td>6.7%</td>
</tr>
<tr>
<td>3</td>
<td>Ssh</td>
<td>6.3%</td>
</tr>
</tbody>
</table>

What apps are used?

Which network uses web and kazaa?
### Ideal traffic report

<table>
<thead>
<tr>
<th>Traffic aggregate</th>
<th>Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web traffic</td>
<td>42.1%</td>
</tr>
<tr>
<td>Web traffic to library.bigU.edu</td>
<td>26.7%</td>
</tr>
<tr>
<td>Web traffic from <a href="http://www.catsplayingpiano.com">www.catsplayingpiano.com</a></td>
<td>13.4%</td>
</tr>
<tr>
<td>ICMP traffic from sloppynet.badU.edu to jeff.dorm.bigU.edu</td>
<td>11.9%</td>
</tr>
</tbody>
</table>

Web is the **dominant** application. The library is a **heavy user** of web. This is a **Flash crowd**! Denial of Service attack **!!**
Hierarchical Heavy Hitter

• Input stream $S$ of keyed weights $\{(k_i, x_i)\}$, total weight $X$
• Keys $k_i$ drawn from a hierarchical domain $D$ of height $h$
• Let $p$ denote a prefix in the domain hierarchy
• $D(p)$ be the set of weights in $D$ that are descendants of $p$
• $X(p) =$ total weight in $S$ with keys in $D(p)$
  - $X(p) = \sum_{i: k(i) \in D(p)} x_i$

• $\phi$-Hierarchical Heavy Hitter ($\phi$-HHH)
  - Any prefix $p$ for which $D(p)$ is at least a fraction $\phi$ of total weight
  - $\phi$-HHH = $\{p \in D: X(p) \geq \phi X\}$
One-dimensional example

Hierarchy

Threshold=100

CS Dept

2nd floor

AI Lab

10.0.0.0/28

10.0.0.8/29

10.0.0.12/30

10.0.0.14/31
One-dimensional example

Threshold=100

10.0.0.0/28

10.0.0.0/29
120

10.0.0.0/30
50

10.0.0.2/31
15

10.0.0.3
35

10.0.0.4
30

10.0.0.5
40

10.0.0.8/31
160

10.0.0.0/29
70

10.0.0.4/30
70

10.0.0.0/30

10.0.0.8/30
305

10.0.0.8/31
270

10.0.0.0/28

10.0.0.0/29
380

10.0.0.8/31
75

10.0.0.10
35

10.0.0.12/30
75

10.0.0.14/31

10.0.0.14/31
75

10.0.0.0/29

10.0.0.8/31
35

10.0.0.0/28

10.0.0.0/30

10.0.0.10
380

10.0.0.12/30
75

10.0.0.14/31

10.0.0.0/30

10.0.0.10
35

10.0.0.14/31

10.0.0.0/28

10.0.0.0/29
380

10.0.0.8/30
35

10.0.0.12/30
75

10.0.0.14/31

10.0.0.8/31
270
(Compressed) Hierarchical Heavy Hitter

• Previous setup
  – Input stream $S$ of keyed weights $\{(k_i, x_i)\}$, total weight $X$
  – Keys $k_i$ drawn from a hierarchical domain $D$ of height $h$.
  – Prefix $p$; descendants $D(p)$; $X(p) =$ total weight under $p$

• Define $\phi$-Hierarchical Heavy Hitter ($\phi$-HHH’) inductively
  – At lowest level in hierarchy, prefix = key
    • $\phi$-HHH’ = $\phi$-HH on key set
  – At any higher level
    • $Y(p) = X(p) - \sum \{ Y(q) : q$ a child of $p$ that is a $\phi$-HHH}$
  – Is $\phi$-HHH’ if $Y(p) \geq \phi X$
Unidimensional report example

Rule: omit clusters with traffic within error $T$ of more specific clusters in the report.

<table>
<thead>
<tr>
<th>Source IP</th>
<th>Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.0/29</td>
<td>120</td>
</tr>
<tr>
<td>10.0.0.8/29</td>
<td>380</td>
</tr>
<tr>
<td>10.0.0.0/29</td>
<td>120</td>
</tr>
<tr>
<td>10.0.0.8/29</td>
<td>380</td>
</tr>
<tr>
<td>10.0.0.9</td>
<td>110</td>
</tr>
<tr>
<td>10.0.0.9</td>
<td>110</td>
</tr>
</tbody>
</table>
Multidimensional structure

Nodes (clusters) have **multiple** parents
Nodes (clusters) overlap

**US Web**
Frequent Itemset Mining in Hierarchies

- Consider flow records = basket
- Key fields = items
  - Basket = { SrcIP, DstIP, SrcPrt, DstPrt, Proto, ….}
- Vanilla FIM has no recognition of hierarchies
  - HHH distributed over addresses under a prefix is not recognized
- Challenge: how to make FIM hierarchy-aware?
Hierarchical expansion

• Expand IP addresses as fill set of prefixes
• 32-bit IP address \(\rightarrow\) 25 prefixes of length 8 through 32
  – Prefixes shorter than 8 bit have not been assigned
• Full Expansion
  – Replace IP addresses in each “basket” by list of 25 prefixes
  – Apply frequent itemset mining to find frequent prefixes
• Drawbacks
  – Full expansion rarely necessary in practice
  – Most heavy hitters are not 32 bit addresses
  – If a prefix is known to not be heavy hitter, no need to expand it further
    • Downward closure property
Progressive Expansion

• Don’t expand all prefixes initially
• Prefix of length k is explored only if parent k-1 is frequent
  – Based on downward closure property
• Drawback
  – Computation cost: Need to expand prefixed as many as 25 times
k-by-k Progressive Expansion

• Expand prefixes in length blocks of k
• 8,9,…8+k-1, 8+k,8+k+1,8+2k -1 etc
• Each block
  – FIM mining for heavy hitters
• Trade-off
  – Exploration of non-HHH prefixes
    • Descendants of non-HHH prefixes in same block
  – Reduced number of passes
    • FIM done jointly on all levels within block
Multidimensional Case

- Key fields: SrcIP, DstIP, SrcPrt, DstPrt, Proto
- Expand SrcIP, DstIP as before
- SrcPrt, DstPrt
  - Aggregate by range
    - well known: 0-1023
    - registered: 1024-49151
    - dynamic: > 49151
  - Aggregate by application type
    - web, p2p, real-time, gaming,…
- Previous FIM notions apply
  - Is a rule interesting?
    - Conditional vs. unconditional probabilities for 2-dimensional and highers sets of key field values
    - Compare Pr(Prt A | Prefix B) with Pr(Port A)
Limitations of FIM approach

- Multiple passes over the data
- Online detection of HHH?

- Can we dynamically attribute traffic to HH cluster?
- Difficulty: can’t use multiple passes to refine prefixes
HHH / Cluster detection

• Input
  – `<src_ip, dst_ip, src_port, dst_port, proto>`
  – Bytes (we can also use other metrics)

• Output
  – All traffic clusters with volume above 
    `(epsilon * total_volume)`
    • (cluster ID, estimated volume)
  – Traffic clusters: defined using combinations of IP prefixes, port ranges, and protocol
Standard Tries

• The standard trie for a set of strings $S$ is an ordered tree such that:
  – Each node but the root is labeled with a character
  – The children of a node are alphabetically ordered
  – The paths from the external nodes to the root yield the strings of $S$
• Example: standard trie for the set of strings $S = \{ \text{bear, bell, bid, bull, buy, sell, stock, stop} \}$
Analysis of Standard Tries

- A standard trie uses $O(n)$ space and supports searches, insertions and deletions in time $O(dm)$, where:
  - $n$ total size of the strings in S
  - $m$ size of the string parameter of the operation
  - $d$ size of the alphabet
Word Matching with a Trie

• We insert the words of the text into a trie
• Each leaf stores the occurrences of the associated word in the text
Compressed Tries

- A compressed trie has internal nodes of degree at least two.
- It is obtained from standard trie by compressing chains of “redundant” nodes.
Using Tries for HHH detection

• Node = prefix
• Count bytes in short prefixes
• If prefix becomes a HH
  – Breakout into descendant longer prefixes
  – Count only bytes in descendant prefixes
• Repeat as necessary
• Some post processing required to allocate initial byte counts in short prefixes
Dynamic Drilldown via 1-DTrie

- At most 1 update per flow
- Split level when adding new bytes causes bucket $\geq T_{split}$
- Invariant: traffic trapped at any interior node $< T_{split}$
1-D Trie Data Structure

- Reconstruct interior nodes (aggregates) by summing up the children
- Reconstruct missed value by summing up traffic trapped at ancestors
- Amortize the update cost
1-D Trie Performance

- **Update cost**
  - 1 lookup + 1 update

- **Memory**
  - At most $1/T_{\text{split}}$ internal nodes at each level

- **Accuracy: For any given $T > d*T_{\text{split}}$**
  - Captures all flows with metric $\geq T$
  - Captures no flow with metric $< T-d*T_{\text{split}}$
Extending 1-D Trie to 2-D: Products

Update(k1, k2, value)

- In each dimension, find the deepest interior node (prefix): (p1, p2)
  - Can be done using longest prefix matching (LPM)
- Update a hash table using key (p1, p2)

\[ \text{totalBytes}\{ p1, p2 \} += \text{value} \]
Cross-Producting Performance

• Update cost:
  – 2 X (1-D update cost) + 1 hash table update.

• Memory
  – Hash table size bounded by \((d/T_{\text{split}})^2\)
  – In practice, generally much smaller

• Accuracy: For any given \(T > d*T_{\text{split}}\)
  – Captures all flows with metric \(\geq T\)
  – Captures no flow with metric \(< T - d*T_{\text{split}}\)
Attribution of traffic

- Bin traffic in shorter prefixes
- After threshold T reached, bin traffic into longer prefixes
- How to allocate unsplit traffic in shorter prefix
- Two methods
  1. Lower bound: ignore unsplit traffic
  2. Proportional splitting of T amongst longer prefixes
A Sampling of the extensive literature

- **Automatically Inferring Patterns of Resource Consumption in Network Traffic**, Cristian Estan, Stefan Savage, George Varghese, SIGCOMM 2003
- **Finding hierarchical heavy hitters in data streams**, Graham Cormode, Flip Korn, S Muthukrishnan, Divesh Srivastava VLDB 2003
- (Credit to the various authors for some slides)