A Robust System for Accurate Real-time Summaries of Internet Traffic

Ken Keys, David Moore, Cristian Estan

CAIDA
University of California, San Diego
University of Wisconsin-Madison

SIGMETRICS – June 8, 2005
Big Picture

• Operators and researchers want to understand the traffic on Internet links.

• Generic measurement system goals:
  – Produce answers which match user questions
  – Be accurate in measurements
  – Scale to high network speeds (OC-768, 10GigE, …)
  – Be robust for all traffic mixes (DDoS, worm, flash crowd, …)
Typical Operational Measurement Questions

• What is the application breakdown in packets & bytes?

• How much traffic came from or went to a particular subnet?

• What are the best ISPs to peer with to decrease my costs based on the actual traffic of my customers?

• Where is the best place to deploy a new web cache?

• Which of my web servers has the most unique clients?

• Which of my hosts seem to be spam servers?
Flow Measurement

• How do we answer these questions?

• Current operational traffic measurement:
  – Typically collected on routers
  – Packet sampling employed on high-speed links
  – Flow-based
    • For each tuple of: protocol, source & destination IP addresses and ports
    • Count: # packets, # bytes

• Generally, people aggregate flows to make summaries keyed by specific fields (e.g. just source IP address)
## Flow aggregation (by source ip)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1.1.93</td>
<td>1.82.0.1</td>
<td>UDP</td>
<td>53</td>
<td>53</td>
<td>2</td>
<td>497</td>
<td></td>
</tr>
<tr>
<td>6.1.0.14</td>
<td>4.44.0.1</td>
<td>TCP</td>
<td>80</td>
<td>2223</td>
<td>4</td>
<td>646</td>
<td></td>
</tr>
<tr>
<td>6.3.0.27</td>
<td>1.95.0.1</td>
<td>TCP</td>
<td>1214</td>
<td>62772</td>
<td>125</td>
<td>187008</td>
<td></td>
</tr>
<tr>
<td>6.1.1.93</td>
<td>4.71.0.6</td>
<td>TCP</td>
<td>49200</td>
<td>80</td>
<td>3</td>
<td>565</td>
<td></td>
</tr>
<tr>
<td>6.1.0.28</td>
<td>1.82.0.1</td>
<td>TCP</td>
<td>49199</td>
<td>80</td>
<td>5</td>
<td>647</td>
<td></td>
</tr>
<tr>
<td>6.1.1.93</td>
<td>1.95.0.1</td>
<td>TCP</td>
<td>49198</td>
<td>80</td>
<td>5</td>
<td>647</td>
<td></td>
</tr>
<tr>
<td>6.1.1.93</td>
<td>4.71.0.6</td>
<td>TCP</td>
<td>49196</td>
<td>80</td>
<td>6</td>
<td>708</td>
<td></td>
</tr>
<tr>
<td>6.1.2.59</td>
<td>7.88.0.1</td>
<td>TCP</td>
<td>51643</td>
<td>80</td>
<td>6</td>
<td>817</td>
<td></td>
</tr>
</tbody>
</table>

|              |                |       |           |            | 16           | 2417       | 4          |
Large-Scale Malicious Traffic

• Denial-of-service attacks, worm spread and port scanning can overwhelm flow measurement systems

• Fields in the flow key take on a much larger range than in normal traffic:
  – Spoofed source DoS = random source IP address
  – Typical Internet worm = random destination IP address
  – Port scanning = walk of large # of ports and addresses

• In these situations every single packet may result in a separate flow
Outline

• Background

• Traffic Summaries

• System Overview and Algorithms:
  – Count flows
  – Identify important entries
  – Adapt sampling rates

• Conclusions
Traffic Summaries

• Most users have a well-defined set of reports and aggregations they normally want.

• Can we do better than Adaptive NetFlow with the Flow Counting Extension (or similar flow-based reporting) when the user specifies the desired aggregations in advance?

• Yes!
  – Smaller, more specific reports.
  – More precise estimates, including tight lower-bounds.
  – Isolation of damage from DoS, worms and scanning.
Traffic Summaries

- Provided reports are of “hogs” (or “heavy-hitters”)
  - All aggregates contributing significant numbers of packets, bytes or flows are reported

- Operator configures desired aggregations

- For example:
  - Source IP addresses – top sources by pkts, bytes or flows
  - Protocol/Ports – for determining top applications
Traffic Summaries – “Hog” Reports

Key type:
• source IP address
• destination IP address
• src. Port and Protocol
• dst. Port and Protocol
• AS matrix
• dst. network prefix
• …

Counter type:
• Packet hogs
• Byte hogs
• Flow hogs
• Out-degree hogs
• In-degree hogs
• …
Robustness & Isolation

• Robustness – system should degrade gracefully in the face of unexpected load, continuing to provide accurate answers with bounded error.

• Isolation – results of each separate summary should be similar to what it would be if that summary was the only one being computed.
  
  – i.e. traffic mixes which cause one table to rapidly fill should not interfere with the accuracy of the other tables
Outline

• Background
• Traffic Summaries
• System Overview and Algorithms:
  – Count flows
  – Identify important entries
  – Adapt sampling rates
• Conclusions
Why is this hard?

- The most straightforward way of generating a hog report would be to keep a table indexed by the each key in the traffic with a simple counter for the measured value.
  - Tables can easily get very large, holding entries which will never be part of the final hog report.
  - Counting flows (or in-/out- degree) cannot be done with simple counters alone, since more state is required to track the unique members of a set.
• Basic logical control flow is replicated for each desired summary table.
• The actual design shares computation and information to improve both system efficiency and the accuracy of counters.
Algorithms – Flow Counting

• Generic goal: given a stream of items, count the number of unique items.

• Our goal: given a stream of packets belonging to specific flows, count the number of unique flows.

• Caveat: we need to do this operation for many entries (100,000s) in parallel
  – e.g. for each different source IP address that will be reported, we must track how many flows had that address
**Algorithms – Flow Counting**

• Three general approaches:
  – Keep a table of all flows and aggregate by key-type when needed
    • Exact answers, but infeasible for most real-time applications
    • Generally used in existing deployments
  – Individual flow counting data structure per key-type table entry
  – Global data structure and simple counter per key-type table entry
Per Entry Flow Counting Algorithms

• Multiresolution bitmaps (MRB)
  – Memory requirements are logarithmic in maximum number of flows
  – A couple kilobytes is sufficient to give 3% average error for hundreds of millions of flows

• Triggered bitmaps (TRB)
  – Most entries will have a small number of flows
    • Start with a small direct bitmap.
    • When it fills, dynamically switch to a MRB.
  – Saves memory in the typical case but the accuracy is less than using MRB from the beginning
Per Entry Flow Counting Algorithms

• List-triggered bitmaps (LTRB)
  – Again, most entries will have a small number of flows
    • For each entry, maintain a small list (2 – 4) of the actual flow ids seen
    • When the list fills, switch to MRB. Populate the MRB with the exact set of seen flow ids.

  – Accuracy is the same as MRB, while achieving space savings similar to TRB.
Global Flow Counting Algorithms

• Tuple set membership:
  – Maintain a set of all flows previously counted.
  – On all packets, check its flow in the set:
    • If present, the flow has already been counted
    • If not present, the flow is new, update the counts for all table entries involving this packet
  – Provides exact counts, but memory usage explodes

• Bloom filter tuple set:
  – Use a bloom filter to approximate set membership with fixed false-positive rates
  – Significantly smaller memory requirements
Global Flow Counting Algorithms

• Bloom filter:
  – False-positives can lead to under-estimation of flow counts
    • Every reported count is a lower-bound
  – Some false-positives can be detected by combining information from multiple summary tables
    • e.g., if the bloom filter says that we have already seen the flow associated with a packet, but there is no entry in the destination IP address table when there should be, then we know this is a new flow
Experimental Setup
(For all data in this presentation)

• Packet traces allow testing with different algorithms and seeds

• OC-48 trace (5 minute portion):
  – 22.5M packets (75 kpps), 12.8GB (342 Mbps), 1.21M flows

• Simulated DDoS:
  – Fixed destination IP address and port, 44 byte TCP SYN packets
  – Random source IP addresses and source ports
  – 10M packets (33 kpps), 400MB (12 Mbps), 10M flows

• Additional datasets used in paper
• Same software runs in real-time on monitor boxes
Picking a Counting Algorithm

- MRB (multiresolution bitmap) memory usage is 1062 MB.
- TRB (triggered bitmap), LTRB (list-triggered bitmap), Bloom (bloom filter)
System Overview

- Does packet have entry?
  - Yes: Identify Important Entries
  - No: Adapt Sampling Rate
- Selected: Create entry
- Increment Counters
Algorithms – Identifying Important Entries

• Packet sample and hold (PSH) ensures there are table entries for anything with a large number of packets
  – For each packet, if there is already an entry in the table, increment the packet count
  – If there is not an entry, probabilistically sample this packet and create an entry in the table when sampled

• Flow sample and hold (FSH) ensures there are table entries for anything which will have a large number of flows
  – For each packet, if there is already an entry in the table, update the associated flow count using previous techniques
  – If there is not an entry, sample this flow by checking if Hash(flowID) < f (sampling fraction) and create an entry in the table
Why both PSH & FSH (& DSH...)?

- Full tuple set table, so all error due to sampling
- PSH can use lower sampling rates and keep packet count errors low
- FSH can use lower sampling rates and keep flow count errors low
- Use both to keep total sampling low with low error
System Overview

- Does packet have entry?
  - Yes: Identify Important Entries, Adapt Sampling Rate
  - No: Increment Counters

- Create entry if selected.
Algorithms – Adapting Sampling Rate

• Observe which tables and samplers (PSH, FSH) are contributing to memory consumption and dynamically adjust the sampling rates for each.

• Details in paper/technical report.
Adaptivity – Robustness/Isolation during DDoS Attack

- Recall DDoS attack is spoofing source addresses and not the destination addresses.
- We expect such an attack to over-fill the source IP address table, reducing accuracy.
- However we wish to ensure that other tables are not adversely affected.
Adaptivity – Robustness/Isolation during DDoS Attack

Backbone Traffic with DDoS adaptive sampling rate over time

sampling rate

0.001

0.01

0.1

1

time (seconds)

0

50

100

150

200

250

300

fsh
dst IP

dst IP

fsh src Proto Port

psh src Proto Port

fsh

src IP

psh

src IP
Conclusions

• Producing traffic summaries, rather than collecting all flows and forcing the user to aggregate:
  – significantly decreases memory usage and reporting bandwidth
  – significantly increases accuracy of results

• Novel algorithms:
  – Flow sample and hold (FSH) allows online streaming identification of “flow hogs”
  – Bloom filter tuple set counting and list-triggered bitmaps (LTRB) efficiently solve “flow counting” for 100,000s of counters

• Adaptive controls:
  – provide robustness against malicious/unexpected traffic
  – allow isolation between independent reports
Questions?
Extended Technical Report

Traffic Summaries – Global Traffic Counters

- # of packets
- # of bytes
- # of active flows (5-tuples)
- # of active source IP addresses
- # of active destination IP addresses
- ...
**Traffic Summary Isolation**

- We would prefer that the separate aggregation reports were independent and isolated:
  - Traffic which causes one table to rapidly fill should not interfere with the accuracy of the other tables

- To solve this, we:
  - Adjust the sampling rates independently for each report
  - Dynamically adapt memory consumption for each separate table to ensure high fidelity for all
Bottlenecks
Picking a Counting Algorithm

Flow estimates for src IP by rank

true
LTRB
Bloom

true rank

flow count

0 5 10 15 20 25 30 35 40 45 50

7000
6500
6000
5500
5000
4500
4000
3500
3000
2500
2000