Network Monitoring as a Streaming Analytics Problem

Arpit Gupta

Princeton University

Rüdiger Birkner, Marco Canini, Nick Feamster, Chris Mac-Stoker, Walter Willinger
Conventional Network Monitoring

Monitoring Queries → Compute → Store → Network Monitor

Monitoring Sensor

NetFlow, sFlow, SNMP, etc.
Big ("Internet") Data

Not suited for large networks & real-time monitoring applications
Network Monitoring as Streaming Analytics Problem

Is using state-of-art stream processor good enough solution?
Is it Good Enough?

• Use Case:
  – Reflection Attack Monitoring Query
    Detect hosts for which # of unique source IPs sending UDP response messages exceeds threshold
Is it Good Enough?

• Use Case:
  – Reflection Attack Monitoring Query
    Detect hosts for which # of unique source IPs sending UDP response messages exceeds threshold

```scala
victimIPs =
  pktStream.window(W).transform {
    wndPkts =>
      wndPkts.filter(p => p.proto == 17)
        .map(p => (p.dIP, p.sIP)).distinct
        .map((dIP, sIP) => (dIP, 1))
        .reduceByKey(sum)
        .filter((dIP, count) => count > T)
        .map((dIP, count) => dIP)
  }
```
Is it Good Enough?

• Use Case:
  – Reflection Attack Monitoring Query
    Detect hosts for which # of unique source IPs sending UDP response messages exceeds threshold
  – Use two hour IPFIX data trace from a large IXP

• Prohibitively Costly:
  – Packet Processing Cost: requires processing 220 M packets per second

How can we bring down these costs?
Idea 1: Iterative Query Refinement

• **Observation:**
  Small fraction of traffic satisfies monitoring queries, e.g. only 1% of the traffic satisfies reflection attack query

• **How it works:**
  – Augment operator’s query to observe at coarser level
  – Iteratively zoom-in to filter out uninteresting traffic

• **Trade-off:**
  – Reduces count bucket cost
  – Introduces additional detection delay cost
Iterative Query Refinement

Stream Processor’s output used by Run-time to refine queries
Iterative Refinement in Action

root - all traffic

Q(dIP/8)
Iterative Refinement in Action

root - all traffic

\[ Q(dIP/8) \]

\[ Q(dIP/16) \]
Iterative Refinement in Action

root - all traffic

Q(dIP/8)

Q(dIP/16)

Q(dIP/32)
Iterative Refinement in Action

D deteces hosts that satisfy the query in 3 window intervals
Idea 2: Query Partitioning

• **Observation:**
  Data Plane can process packets at line rate

• **How it works:**
  Delegate query processing operations that can be executed in the data plane, e.g. *filtering, sampling*

• **Trade-off:**
  - Reduces both the pkt processing & count bucket cost
  - Introduces additional state in the data plane
Query Partitioning

Iterative Refinement

Monitoring Queries

Run-time

Stream Processor

Data Plane Configurations

Programmable Data Plane

Runtime Partitions Monitoring Queries
## Performance Improvements

<table>
<thead>
<tr>
<th>Reflection Attack Monitoring (dIP/16 → dIP/32)</th>
<th>Rate (pps)</th>
<th># Buckets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream Processor Only</td>
<td>220 M</td>
<td>1.16 B</td>
</tr>
<tr>
<td>Iterative Refinement Only</td>
<td>220 M</td>
<td>12 K</td>
</tr>
<tr>
<td>Iterative Refinement + Query Partitioning</td>
<td>5.4K</td>
<td>12 K</td>
</tr>
</tbody>
</table>

Trades pkt processing & count bucket cost for additional detection delay
Network Monitoring Applications

• Reflection Attack Monitoring (Security)
  Detect hosts for which # of unique sIPs sending UDP response messages exceeds thresh

• Distributed Port Scan Detection (Security)
  – Detect hosts for which # of unique dIP exceeds thresh
  – Detect hosts for which # of unique dPorts exceeds thresh

• Distributed Jitter Monitoring (QoE)
  Detect user groups for which RTT exceeds thresh
Future Directions

• **Query Language**
  How to dynamically map high-level abstractions to packet tuples?

• **Iterative Refinement**
  How to automate generation of optimal refinement plans for a query?

• **Query Partitioning**
  How to execute more complex streaming operations like map, reduce, join etc. in the data plane?
Summary

• Big (“Internet”) Data motivates modulating network monitoring as a streaming analytics problem
  Using state-of-art stream processors is not enough

• Stream processors + programmable data planes raise new opportunities

• Iterative Query Refinement and Partitioning can reduce pkt processing and count buckets by 4 and 5 orders of magnitude, respectively
Backup Slides
Feast or Famine Dichotomy

• Feast:
  • Capture all traffic, e.g. pcap
  • Detect all interesting network events
  • Higher cost and slower detection

• Famine:
  – Capture subset of traffic, e.g. Netflow, SNMP etc.
  – Not useful for many monitoring applications

Current Trend:
“Capture all the packets, all the time”
Refinement Plan Search

Learn \( \{x\} \) and \( r \) that minimize a linear combination of cost metrics

Count buckets
Run-time

\[
\text{victims}_{16}(t+1) = \text{pktStream} . \text{window}(W) . \text{transform} \{ \text{wndPkts} \Rightarrow \\
\text{wndPkts} . \text{filter}(p \Rightarrow p.\text{proto} == 17) . \text{filter}(d\text{IP} \in \text{victims}_{8}(t)) . \text{sample}(r) . \text{map}((d\text{IP}) \Rightarrow (d\text{IP}/16)) . \text{map}(p \Rightarrow (p.d\text{IP}, p.s\text{IP})).\text{distinct} . \text{map}((d\text{IP}, s\text{IP}) \Rightarrow (d\text{IP}, 1)) . \text{reduceByKey}(\text{sum}) . \text{filter}((d\text{IP}, \text{count}) \Rightarrow \text{count} > T) . \text{map}((d\text{IP}, \text{count}) \Rightarrow d\text{IP})
\]
SONATA Architecture

Application Interface

Q₁  Q₂  Qₙ

Run-time

Refinement

Partition

Data Plane Configurations

Packets

Tuples

Programmable Data Pane

Stream Processor

Data Processing Pipeline

Training Data

Query Output