Enabling Event-Triggered Data Plane Monitoring

Jan Kučera
jan.kucera@cesnet.cz
CESNET, a.l.e.

In collaboration with:
Diana Andreea Popescu, Andrew Moore (University of Cambridge)
Han Wang (Barefoot Networks), Jan Kořenek (Brno University of Technology)
and Gianni Antichi (Queen Mary University of London)
High-volume traffic clusters

- The importance of finding high-volume traffic clusters
- Real-time detection is beneficial to many network applications

<table>
<thead>
<tr>
<th>Network event</th>
<th>Management task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heavy Hitters</td>
<td>accounting, traffic engineering</td>
</tr>
<tr>
<td>Superspreaders</td>
<td>worm, scan, DDoS detection</td>
</tr>
<tr>
<td>Changes in traffic patterns</td>
<td>anomaly detection</td>
</tr>
</tbody>
</table>
High-volume traffic cluster definition

- **Traffic cluster** (or aggregate)
  - Object exceeding a pre-determined threshold in a time window

- **For IP address** as a key
  - IP prefixes that contribute with a traffic volume, in terms of bytes, packets or flows, larger than a **threshold T** during a **time interval t**

![Diagram showing IP prefixes and traffic clusters]
Traffic clusters events

- **Heavy hitter (HH)**
  - A host that sends or receives at least a given number of packets (or bytes)
  - A traffic cluster in terms of packets or bytes per second

- **Superspreader (SS)**
  - A source host that contacts at least a given number of distinct destinations
  - A traffic cluster in terms of unique flows per second
  - If applied to distinct sources also known as DDoS victim detection

- **Change detection**
  - Identifying changes in the traffic patterns over two consecutive intervals
  - Identifying the traffic that contribute the most for the change
  - A change of traffic clusters in terms of packets, bytes or flows
Dataplane programmability

- In the past, the **detection performed outside the dataplane**
  - In **software** collectors, **packet sampling** employed, using **NetFlow** or **sFlow**

- Today, we can leverage dataplane programmability!

  - **HashPipe** [1]
    - Exports the **top-k heavy flows counters** at **fixed time** intervals
    - **Pipeline of hash tables** to retain counters of heavy flows

  - **UnivMon** [2], **Elastic Sketch** [3]
    - Export smart **representation of aggregated statistics** at **fixed time** intervals
    - **Sketch-based data structures** to record network traffic statistics

---


Motivating a new solution (1)

What do these solutions have in common?
- The dataplane *only aggregates statistics* -- *only assists* in the detection
- Poll-based model -- the controller *polls the structures* at *fixed time* intervals
- The actual detection (processing the structure) *performed in the control plane*

Is this a problem?

**Reporting time** of heavy hitters detection
- CAIDA packet traces, *reporting time* 20 s
- Which HHs *could have been detected* earlier?
- > 60% could have been detected *within 1 second*

➢ *Reporting time should be as low as possible*
Motivating a new solution (2)

- **Is it possible?**
- **The cost of statistics collection**
  - At least 60k - 150k counters required
  - At least 2.5 - 5 seconds needed to retrieve 80k counters when the switch in idle state
  - Retrieving the structures is time consuming
- **Would a push-based sketch work?**
- **The limited memory access**
  - RMT architecture restrictions to guarantee high throughput
  - Only a few addresses in a memory block can be read or written

![Time to retrieve HW counters](image-url)
Our questions (?)

■ Is it possible to design a data structure, well-suited for
  ■ push-based design, that
  ■ would access only a small memory block and
  ■ expose a single entry upon the detection of a network event?

■ Enabling true in-network event-triggered detection?
  ■ Event-based controller does not have to receive a lot of useless data
  ■ As soon as detected, take pre-defined actions → better reactivity

■ Is it possible? Yes, it is!
  ■ We did it. We designed such a data structure & algorithm
  ■ We call it Elastic Trie
Elastic Trie data structure in a nutshell

- Prefix tree that **grows** or **collapses**
  - Focus on prefixes that account for a large share of the traffic

- Each node consists of **three elements**
  - (1) left child **counter**, (2) right child **counter**, (3) node **timestamp**
  - **Starting condition:** a single node for the **zero-length prefix** *

- For every incoming packet (5 possible cases)
  - Find the **most specific node** (LPM) and use **timeouts** to detect clusters
  - **Compare** packet and node **timestamps**, node **counters** and defined **threshold**
    - (1) **expand** the node, (2) **collapse** the node, (3) **keep** the node
    - (4) **invalidate** the node, or (5) **update** the node counter
Updating the node counters

Starting condition
Root node

Starting condition
Root node

On the incoming packets basis

$\begin{align*}
&\text{Root node} \\
&c_1^{**} += 1 \\
&c_0^{**} += 1 \\
&t_N^{**} = t_p
\end{align*}$
Elastic Trie in action | How does it work? (2)

- **Expanding the node**

  - Adds a child, resets a counter, generates a report

  - Expanding the node

    - $c_1^{**} >= \text{threshold } T$
      - $c_1^{**} = 0$
      - $t_N^{1*} = t_P$
      - $c_0^{1*} = c_1^{1*} = 0$
      - ... packet reception counters updates ...

    - $c_0^{**} >= \text{threshold } T$
      - $c_0^{**} = 0$
      - $t_N^{0*} = t_P$
      - $c_0^{0*} = c_1^{0*} = 0$
      - ... packet reception counters updates ...

    - $c_0^{0*} >= \text{threshold } T$
      - $c_0^{0*} = 0$
      - $t_N^{00} = t_P$
      - $c_0^{00} = c_1^{00} = 0$
Elastic Trie in action | How does it work? (3)

- **Keeping** the node
  - Resets counters, sends a report

- **Collapsing** the node
  - Removes the child, resets counters

\[ t_N^{1*} << t_p \text{ and } c_0^{1*} + c_1^{1*} \geq \text{threshold } T \]

- Packet reception counters updates...

\[ t_N^{0*} << t_p \text{ and } c_0^{0*} + c_1^{0*} < \text{threshold } T \]

- Packet reception counters updates....
Elastic Trie implications | Other events

- The dataplane iteratively **refines the responsible IP prefixes**
  - The controller can receive **flexible granularity information**

- Each prefix **tree layer** can have a **different timeout**
  - Trade-off between tree **building process** and **memory consumption**

- **Superspreaders** (not at the same time, **either HH or SS detection**)
  - Bloom filter to identify **unique flows**
  - Node counters for **distinct destinations count** of source prefixes

- **Traffic pattern changes** (independently, **on top of HH or SS detection tree**)
  - Identified by looking at the **growing rate** of the tree
  - Tracking the **difference in number of expanded and collapsed nodes**
Elastic Trie implementation

- **LPM classification**
  - The prefix tree structure

- **Bloom filter (SS only)**
  - To test if packet belongs to a new unique flow or not

- **Main memory**
  - Where all the per-node information are stored

- **Control logic**
  - The brain of the algorithm
Elastic Trie implementation in P4 (1)

- **LPM match-action tables**
  - We cannot use them
  - We cannot modify entries directly from the dataplane

- **Custom LPM implementation**
  - Hash table for each prefix length
  - Hash extern API with CRC32
  - Each hash table implemented as a register array
Elastic Trie implementation in P4 (2)

- **Bloom filter**
  - To support **superspreaders**
  - Register-based **bit array**
  - Set of **hash functions**

- **Main memory**
  - Register **array**
  - The **hash value** of the LPM is the **address** to access a register that stores the node information
  - **Two node counters** (2x 32-bit)
  - Node **timestamp** (48-bit)
Elastic Trie implementation in P4 (3)

**Control logic**
- Compares the node timestamp and the packet timestamp
- Compares the node counters and the threshold
- Decides what to do:
  1. Update the node counter
  2. Expand / 3. Collapse the node
  4. Keep / 5. Invalidate the node
- Implements the structure update logic
- Implements the push-based mechanic with a digest message
Experimental evaluation

- We tested the original **P4 implementation** running in **BMv2**
- We created **FPGA implementation** to quantify **HW resources**
  - Two **Xilinx FPGAs**
  - **Virtex 7, UltraScale+**

- We further created **C++ model** for packet **traces simulations**
  - Simulation of heavy hitter, superspreader and change detection on
  - **Four one-hour packet traces** from **CAIDA** (San Jose 2009 and Chicago 2016)
  - Comparison with other solutions (**UnivMon, HashPipe, ElasticSketch**) in terms of memory occupancy, detection **accuracy** and **speed** and **bandwidth** utilization

<table>
<thead>
<tr>
<th>Chip</th>
<th>LUTs</th>
<th>Regs</th>
<th>Frequency</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Logic</td>
<td>Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Virtex 7</td>
<td>11 088</td>
<td>2 880</td>
<td>14 104</td>
<td>172.4 MHz</td>
</tr>
<tr>
<td>Virtex US+</td>
<td>9 135</td>
<td>2 641</td>
<td>14 103</td>
<td>307.9 MHz</td>
</tr>
</tbody>
</table>
Experimental results (1)

- **Heavy hitters detection**
  - Reporting **time interval**: 20 s
  - **Threshold**: 5% (of total traffic amount)
  - **Accuracy** defined using **F1 score**
    \[
    F_1 = \frac{2T_P}{2T_P + F_P + F_N}
    \]
    
    \(T_P\) ... true positives, \(T_N\) ... true negatives
    \(F_P\) ... false positives, \(F_N\) ... false negatives
  - **Average** over all the CAIDA traces
  - **ElasticTrie outperforms others**
    - ElasticTrie > ~20 kB
    - UnivMon > ~800 kB
    - HashPipe > ~100 kB
    - ElasticSketch > ~140 kB
**Experimental results (2)**

- **Change detection**
  - **Scan attack** and **DoS attack** injected into the real traffic trace (at $t = 2500$ s)

**On top of HH detection tree**

**On top of SS detection tree**
Experimental results (3)

- **Controller-dataplane communication**
  - Reporting **time interval** / active **timeout**: 20 s, **ElasticTrie**: 12B per report
  - **UnivMon**: 800 KB, **Elastic Sketch**: 140 KB, **HashPipe**: 100 KB

*Detection speed*

*Bandwidth utilization*
Conclusions

- **Enabling Event- Triggered Data Plane Monitoring**
- **Elastic Trie** enables **in-network detection** of traffic aggregates
- Suitable for **heavy hitter**, **superspreader** and **change detection**
- **Push-based** monitoring approach
  - Faster detection
  - Adaptive refinement
  - Smaller bandwidth utilization
  - Low memory footprint

Questions?  
jan.kucera@cesnet.cz
Outline

- High-volume traffic clusters
- State-of-the-art solutions, motivating a new solution
- Elastic Trie, data structure & algorithm
- Experimental results
- Conclusion
Hierarchical high-volume traffic clusters

- **Hierarchical traffic cluster**
  - Special case of traffic cluster
  - It must exceed the threshold after excluding the contribution of all its cluster descendants

- **Minimum overhead**
  - Hierarchical aggregates provides all the necessary information
  - Pure aggregates are always only prefixes of more specific hierarchical aggregates