Carousel --- Scalable and (nearly) complete collection of Information

Terry Lam
(with M. Mitzenmacher and G. Varghese)
Data deluge in Networks

- Millions of potentially interesting events
- How to get a coherent view despite bandwidth and memory limits?
- Standard solutions: sampling and summarizing

Denial of Service

Worm outbreak
What if you want **complete collection**?

- Need to collect infected stations for remediation
- Other examples of complete collection:
  - List all IPv6 stations
  - List all MAC addresses in a LAN
Example: worm outbreak

Intrusion Detection System (IDS)

Slammer C  Witty B  Slammer A

Management Station

A B C
Abstract model

Challenges:

- Small logging bandwidth: $L \ll B$
  - e.g., $L = 1 \text{ Mbps}$; $B = 10 \text{ Gbps}$
- Small memory: $M \ll N$
  - e.g., $M = 10,000$; $N = 1 \text{ Million}$

Opportunity:

- Persistent sources: sources will keep arriving at the logger
Our results

- **Carousel**: new scheme, with minimal memory can log *almost all* sources in close to *optimal time* \((N/L)\)

- Standard approach is *much worse*
  - \(\ln(N)\) times worse in an optimistic random model
  - Adding a *Bloom filter* does not help
  - Infinitely worse in a deterministic adversarial model
Why the logging problem is hard

- Sources 2 and 3 are never collected if pattern repeats
- 1 is logged many times
- In worst case, N – M (many!) sources can be missed
Why the problem is still hard with a Bloom filter

- Similar performance to a standard logger
  - Again, sources 2 and 3 are never collected because of timing

Bloom filter is necessarily small (M) compared to sources (N)
When input traffic exceeds capacity, standard solution is admission control: but it requires source cooperation.

What can a poor resource do to protect itself unilaterally without cooperation from senders?

Our approach: Randomized Admission Control.
- Break sources into random groups and “admit” one group at a time for logging.
Our solution: Carousel

Hash to color the sources say red and blue

IDS

sink

Carousel

Hash to color the sources say red and blue

Only red sources are logged in this phase
Rotating the Carousel

IDS

sink

Carousel

Change color!
How many colors in Carousel?

IDS

sink

Bloom filter

memory

Bloom filter full

Increase Carousel colors

Carousel
Summary of Carousel algorithm

- **Partition**
  - $H_k(X)$: lower $k$ bits of $H(S)$, a hash function of a source $S$
  - Divide the population into partitions with same hash value

- **Iterate**
  - $T = M / L$ (available memory divided by logging bandwidth)
  - Each phase last $T$ seconds, corresponds a distinct hash value
  - Bloom filter weeds out duplicates within a phase

- **Monitor** (to find right partition size)
  - Increase $k$ if Bloom filter is too full
  - Decrease $k$ if Bloom filter is too empty
Snort implementation

1. Linux PCAP
2. Snort Detection Engine

Packet of current color?

- Y: Packet in Bloom filter?
  - Y: Drop packet
  - N: Add packet to Bloom filter

- N: Bloom filter overflow?
  - Y: Increase colors
    - Y: Reset timer
    - N: Clear Bloom filter
  - N: Snort output module

Packet in Bloom filter?

- N: Add packet to Bloom filter

Bloom filter underflow?

- Y: Change color
  - Y: Reset timer
  - N: Clear Bloom filter
- N: Reduce colors

Timer expires?

- N: Snort output module
- Y: Change color
Carousel is “competitive” in that it can collect almost all sources within a factor of 2 from optimal time

- \( N = \) sources, \( L = \) logging speed, optimal time = \( N/L \)
- Collection time \( \approx 2 \, N/L \),

Example: \( N = 10,000 \), \( M = 500 \), \( L = 100 \)

![Graph showing number of logged sources vs. time](image)
Simulated worm outbreaks

Number of logged sources

Carousel is nearly ten times faster than naïve collector
Snort Experimental Setup

- **Scaled down** from real traffic: 10,000 sources, buffer of 500, input rate = 100 Mbps, logging rate = 1 Mbps
- Two cases: source S picked **randomly** on each packet or **periodically** (1, 2, 3 . . . 10,000, 1, 2, 3, . . )
Snort results

(a) Random traffic pattern

(b) Periodic traffic pattern

3 times faster with random and 100 times faster with periodic
Using 1 Mbit of memory, less than 5% of an ASIC

Can be easily added to hardware IDS/IPS chipsets
Related work

- High speed implementations of IPS devices
  - Fast reassembly, normalization and regular expression
  - No prior work on scalable logging

- Alto file system: dynamic and random partitioning
  - Fits big files into small memory to rebuild file index after crash
  - Memory is only scarce resource
  - Carousel handles both limited memory and logging speed
  - Carousel has a rigorous competitive analysis
Limitations of Carousel

- Carousel is probabilistic: sources can be missed with low probability \( \Rightarrow \) mitigate by changing hash function on each Carousel cycle.

- Carousel relies on a “persistent source assumption”
  - Does not guarantee logging of “one-time” events

- Carousel does not prevent duplicates at the sink but has fast collection time even in an adversarial model.
Conclusions

- **Carousel** is a scalable logger that
  - Collects nearly all *persistent* sources in nearly optimal time
  - Is easy to implement in hardware and software
  - Is a form of *randomized admission control*

- Applicable to a wide range of monitoring tasks with:
  - High line speed, low memory, and small logging speed
  - And where sources are *persistent*