Tracking Network Events with Write Optimized Data Structures

Justin Raizes*, Evan West*, Thomas M Kroeger*, Brian Wight*, Cindy Phillips*, Jon Berry*,
Michael Bender†, Rob Johnson‡
*Sandia National Labs {jraizes, ewest, tmkroeg, bjwrigh, caphill, jberry}@sandia.gov
†Stoney Brook University bender@cs.stonybrook.edu
‡VMWare Labs rob@cs.stonybrook.edu

***Abstract*—The basic action of two IP addresses communicating is still a critical part of most security investigations. Typically security tools focus on logging torrents of security events. Some more advanced environments will try to send the logs to a variety of databases. Unfortunately, when faced with indexing billions of events such databases are usually unable to keep up with the rate of network traffic. As a result, security monitors typically log with little to no indexing.**

Write-optimized data structures (WODS) provides a novel approach to traditional data structures. WODS use RAM to aggregate multiple insertions into a single write and as a result are able to ingest data 10 to 100 times faster while answering queries in a timely manner. Our Diventi tool uses a write optimized B-Tree known as a B^e-tree to index layer 3 network activity either from bro connection logs or netflow data. In 2017 our tool was able to track all bro-ids monitored traffic indexing at rates above 100,000 events per second, and typically answering queries in milliseconds.

This year diventi will connect directly with the SciNet security team's core tap and aggregation infrastructure, ingesting netflow records directly from the Ixia switch that will be doing the security monitoring. Working closely with the network security team, diventi will provide sub-second query results to help security responders identify which IPs were communicating at what times.

***Index Terms*—security, networking, indexing, write optimized, IDS**

I. Introduction

Advanced security monitoring must juggle two opposing efforts. Sensor teams focus on collection and recording data as fast as possible, while analytic teams focus on understanding and analysis, which requires access across large swaths of data. If these analytics are to provide on-line monitoring to protect systems as they operate then these systems need to perform their analytics in a timely manner. Ideally these analytics should be able to see and use a wide view of the data collected but this hinders their responsiveness. We can store one second's worth of data in one second. However, searching a year's worth of data in one second is much more challenging. If our analysts can't use our data much of the value is lost. To put this into perspective, imagine an Internet without search engines.

Our research seeks to fill the gap between sensors and analytics to find efficient ways in which sensors can still record one second of data in one second in addition to organizing the data to ensure that analytics can query one year's worth of data in one second. As the size of data scales beyond primary storage (RAM), systems are faced with one of two choices: expire data or move to slower secondary storage and fall behind. Traditionally, these challenges have been tackled by expanding the amount of primary storage available using clusters of computers with lots of RAM. However, the data eventually catches up, overwhelming the amount of primary storage available. Recent work with Write-Optimized Data Structures (WODS) has shown that it is possible to ingest torrential streams of data using larger, less expensive secondary storage, while still maintaining timely queries.

At SuperComputing 2017, Diventi used a B^e tree to track IP addresses across 600 gigabits per-second (Gbps) of monitored traffic on a single computer. In three days, our system indexed over 6 billion events, while maintaining point query response times of milliseconds. In contrast, the SciNet security team used a cluster of 30 nodes to monitor traffic using a commercial log management tool.

This year, at the Network Research Exhibition, we will aim to index, all IPs seen from netflow data received directly from the network infrastructure. We will additionally pay close attention to query performance during active ingestion, in order to understand how an analytic might interact with Diventi. With netflow data, we remove the need for security tools like bro and integrate directly with the core networking infrastructure to provide near real-time insights for the security team.

II. Background

Inserting data into a traditional B-tree typically requires $O(\log_B N)$ writes to secondary storage per each insert. While many of these can be cached in RAM, as N grows to years worth of data the number of writes on each insert trend towards this bound. As a result these traditional data structures don't represent an efficient way to balance primary and secondary storage for tracking torrents of security monitoring data. Because of this security monitoring systems have typically shied away from indexing or used RAM based data structures, limiting the scope of data tracked.

In contrast, a write-optimized B-tree such as the B^e-tree uses buffers at each level in the tree to aggregate multiple inserts into each write. By buffering writes at internal nodes we improve write latency as much as 10 to 100 times while

Fig. 1: B^e tree stores data points at each node. When a node fills up the cached entries are flushed to the lower nodes.

losing only a small constant factor in query performance [1]. Each node, of size B, is divided into two sections: pivots and cache. A tunable parameter, ϵ, takes values between 0 and 1, to determines the ratio between the two components. Pivots and child pointers take up B^ϵ space and the cache takes up the remaining space $(B - B^\epsilon)$. As elements are inserted, they are added to the buffer of the root node. When a buffer fills up, its contents are flushed to lower nodes, where the insertions are again buffered. Figure 1 shows how a typical B^e-tree works.

This behavior results in an insertion performance of $O(\frac{\log_B N}{B^{1-\epsilon}})$, as compared to a standard B-tree's $O(\log_B N)$. For perspective, parameters of $B = 1024$ and $\epsilon = \frac{1}{\ln 1024} \approx .14$ provide a speedup of $\approx 54x$ insertion rate over a standard B-tree.

In trade, the B^ϵ tree pays $O(\frac{\log_B N}{\epsilon})$ for queries instead of a standard B-tree's $O(\log_B N)$. Continuing with the example of $\epsilon = \frac{1}{\ln 1024} \approx .14$, a query takes only $\approx 6.93x$ longer for a B^ϵ tree. In our experiences last year at SuperComputing, queries typically took under 500 ms. From the practical perspective of a security monitoring system, this trade off is frequently the difference between being able to index events in near real-time or having no index at all. In this light the slower queries that use an existing index are far better than the many systems we have seen which run grep in parallel and take minutes to answer the same question [2].

Alternative solutions involving distributing workload over clusters of computers using tools such as MongoDb, Hadoop, or ElasticSearch have also been evaluated. On a cluster of 5 machines, researchers found that none of the tools were able to exceed 200 transactions per second after $100,000,000$ records (average record size 2866 bytes) had been ingested in a write-heavy workload [3].

III. Methods and Engineering

During the Network Research Exhibition, we will receive a copy of the security team's tap of netflow data. Diventi will attempt to index this data in near real-time, tracking the rates at which it is able to do so. The hardware used is a basic Dell server, specifically a Dell 730 server equipped with 128 GB of RAM, one 16 core processor, and 5 SSDs aggregated into an 8 TB data store. Diventi will be used to answer queries about specific IP addresses from the security team as they work to investigate security alerts.

A. Data Schema

Our security monitoring events typically contain a notice about two IP addresses communicating to include timestamp, ports, protocols, and some sense for the number of packets and bytes in each direction. Our B^ϵ-tree stores these events in a key-value pairing. Since each event represents two IP addresses (IP A and IP B) communicating, two separate key-value pairs are inserted per event, to enable quick lookup of either IP.

We begin each key with an IP and timestamp to enable efficient searching for a given IP, possibly over a given time frame or possibly a given subnet of IPs. For example this index can quickly find activity from 1.2.3.4 for the last month, or all activity from sub-net 1.2.3.X. On the other hand searching for all activity from 1.2.3.X from last month would require doing the indexed search of 1.2.3.x and then manually filtering the entries that are in that time frame.

One common limitation of a write optimized data structure is the inability to detect collisions on inserts. This is because detecting if that key already exists would require a full traversal down the tree to verify that it didn't exist at any level. This would require significantly more IOs per insertion and would reduce the ingestion rate. We work around this limitation by filling out our key with the rest of the unique details from this event. This ensures that each unique security event is kept in a unique key and there are no collisions. Table I shows a summary of the data we have in our key.

Byte Range	Length	Field
0–3	4	IP A
4–11	8	Timestamp
12–14	2	Port A
15–18	4	IP B
19–21	2	Port B
22–22	1	Misc flags

TABLE I: Data fields for key

The value holds many of the other relevant components of a network event. These include the duration of interaction, amount of data transferred, and any relevant tcp flags. Typically, security analysts care about the scale of data and packets transferred (e.g. 2 GB 'versus 2 bytes). Tools such as Bro record the exact values, with each of the 4 fields using 4 bytes. Since the complete data is kept in the original logs, our goal is simply to help the security analyst quickly assess whether a connection is of note. With this in mind, we record the magnitude of data transferred instead of its full value, using only a single byte per field, rather than 4. The reduction in value size reduces the amount of writing done per event, thus improving ingestion speed.

Finally we note that while our focus has been on indexing IP address, our code design has focused on clear abstractions between events, keys and values. This has made it easy to adapt our tool to ingesting both bro connection logs and netflow data. We believe this design will also enable easy adaption to indexing other security events such as URLs and e-mail addresses.

B. Contributions and impact

In recent years, much effort has been focused on analysis of big data and network inspection. This work typically has the goal of creating a security system which, in an automated fashion: actively monitors the network, recognizes threats, and takes action to stop those threats. However, considerably less effort has been focused on providing that data in the context of network events in actionable times to these analytics. Without timely query responses and comprehensive data, these systems must either limit to scope of the data they consider or reduce their approach to a post event alert instead of active responses.

Berry and Porter [4] showed that state-of-the-art hashing and expiration methods can quickly degrade in performance as the data set needed to correctly find patterns of interest exceeds the size of available memory. They used the (non-write-optimized) reference implementation for the Firehose benchmark. When the working space could exactly hold the set of active keys, the reference implementation reported $2/3$ of the reportable keys (prematurely expiring the rest). When the working space could hold half the active keys, about $1/3$ of the reportable keys were reported. When only $1/4$ the active keys fit in memory, essentially nothing was reported. This motivates the need for tools that can efficiently use both RAM and secondary storage to increase storage capacity while still providing timely query responses.

Diventi fills this need by indexing data quickly for storage across both primary (RAM) and secondary (SSD) storage while still allowing for timely query responses. Secondary storage is both less expensive and available in larger quantities than primary storage, reducing costs significantly. The increased speed and storage capacity make more efficient use of the resources available and increase the scope of data analytics can consider. It is tools like Diventi that will help bridge the gap between sensors that can record torrents of data in one second and analytics that wish to consider years of that data in one second.

Acknowledgments Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

References

[1] M. A. Bender, M. Farach-Colton, R. Johnson, R. Kraner, B. C. Kuszmaul, D. Medjedovic, P. Montes, P. Shetty, R. P. Spillane, and E. Zadok, "Don't thrash: How to cache your hash on flash," *PVLDB*, vol. 5, no. 11, pp. 1627–1637, 2012.

[2] A. Sharma and V. Stoffer, "p0wnage and detection with bro," in *Proceedings of BroConn 2015*, 2015.

[3] R. P. Ritchey, "Accumulo/hadoop, mongodb, and elasticsearch performance for semi structured intrusion detection (ids) data," ICF, Inc. Columbia United States, Tech. Rep., 2016.

[4] J. W. Berry and A. M. Porter, "Stateful streaming in distributed memory supercomputers," https://www.osti.gov/servlets/purl/1406959, slides from an invited talk at the Chesapeake Large-Scale Analytics Conference (CLSAC) in October 2016. Slides archived at OSTI. Accessed: 2018-4-3.

[5] M. A. Bender, M. Farach-Colton, W. Jannen, R. Johnson, B. C. Kuszmaul, D. E. Porter, J. Yuan, and Y. Zhan, "An introduction to b$^\epsilon$-trees and write-optimization," *:login; magazine*, vol. 40, no. 5, pp. 22–28, October 2015.