Hashdoop: A MapReduce Framework for Network Anomaly Detection

Romain Fontugne*†, Johan Mazel*†, Kensuke Fukuda*
*National Institute of Informatics
†Japanese - French Laboratory for Informatics
Tokyo, Japan

***Abstract*—Anomaly detection is essential for preventing network outages and maintaining the network resources available. However, to cope with the increasing growth of Internet traffic, network anomaly detectors are only exposed to sampled traffic, so harmful traffic may avoid detector examination. In this paper, we investigate the benefits of recent distributed computing approaches for real-time analysis of non-sampled Internet traffic. Focusing on the MapReduce model, our study uncovers a fundamental difficulty in order to detect network traffic anomalies by using Hadoop. Since MapReduce requires the dataset to be divided into small splits and anomaly detectors compute statistics from spatial and temporal traffic structures, special care should be taken when splitting traffic. We propose Hashdoop, a MapReduce framework that splits traffic with a hash function to preserve traffic structures and, hence, profits of distributed computing infrastructures to detect network anomalies. The benefits of Hashdoop are evaluated with two anomaly detectors and fifteen traces of Internet backbone traffic captured between 2001 and 2013. Using a 6-node cluster Hashdoop increased the throughput of the slowest detector with a speed-up of 15; thus, enabling real-time detection for the largest analyzed traces. Hashdoop also improved the overall detectors accuracy as splits emphasized anomalies by reducing the surrounding traffic.**

I. Introduction

Identifying attacks in a network infrastructure is a crucial task to prevent network outage and maintain network resources available to legitimate users. Overwhelmed by the increasing amount of IP traffic, network operators rely on anomaly detectors to automatically identify harmful events that occur on networks. Unsupervised detection techniques are especially attractive as they require no prior knowledge and are easier to deploy. However, to cope with global Internet growth and obtain detector results in a reasonable time frame, the analyzed traffic is usually sampled [3], which is inherently detrimental to anomaly detection as it deliberately discards traffic that may be anomalous.

Motivated by the need for analyzing large datasets, various research communities have developed efficient tools that use the MapReduce model [5]. In this paper, we investigate the benefits of MapReduce to achieve real time anomaly detection with non-sampled traffic. Furthermore, available implementation such as Hadoop MapReduce [15] provides scaling capabilities and fault tolerance that are crucial features for Internet security.

Nonetheless, we found fundamental contradictions between Hadoop data distribution and anomaly detectors requirements. In the MapReduce model, data is conceptually record-oriented; consequently, Hadoop divides datasets into splits, i.e., subsets of records, and distributes them in a cluster to be independently processed. Splits of a dataset have all the same size to ensure that the processing end time for all nodes coincides. In the case of network traffic, a split usually represents a subset of packets [10]. However, related packets may spread across different splits, thus dislocating traffic structures that are essential for anomaly detectors. For example, anomaly detectors can identify DoS attacks because of the numerous requests sent to the same service, but detecting this kind of attack with only a subset of the requests is notably more difficult.

To overcome this, we take advantage of a hash function to divide traffic into splits that preserve the spatial and temporal traffic structures. Therefore, we propose Hashdoop, a MapReduce framework that splits data with a hash function and processes numerous anomaly detectors in parallel. The proposed framework conserves all advantages of the MapReduce model and can virtually be used with any network traffic anomaly detector. We conduct experiments with traffic measured at a trans-Pacific link and two anomaly detectors, a packet count based detector, and a traffic stationarity based detector called "Astute" [14]. Using a local 6-node cluster Hashdoop achieves a maximum speed-up of 3.5 with the packet count based detector and 15 with Astute. In other words, the packet count based detector throughput was improved from 360k packets per second (pps) to 1.27 Mpps, and, for Astute, from 25 kpps to 375 kpps. Consequently, a 900-second trace that was analyzed in 1296 seconds with the classical version of Astute was processed in 216 seconds on Hashdoop, thus enabling real-time processing for this detector. Using Hashdoop, we also observed an overall improvement in detector accuracy as anomalies are filtered in splits with less background traffic.

II. Background

A. Anomaly Detection

Despite the numerous detection methods proposed during the last decade, the approach to identify network traffic anomalies is generally common to all detectors and consists of three main steps: traffic discretization, normal traffic modeling, and anomaly detection.

1) Spatial and temporal discretization of traffic consists in aggregating traffic in flows and partitioning time in nominal intervals. Various definitions of flows are available in the literature, ranging from the traditional 5-tuple flow (protocol, source host, destination host, source port, and destination port) to a coarser view of

978-1-4799-3088-3/14/$31.00 ©2014 IEEE

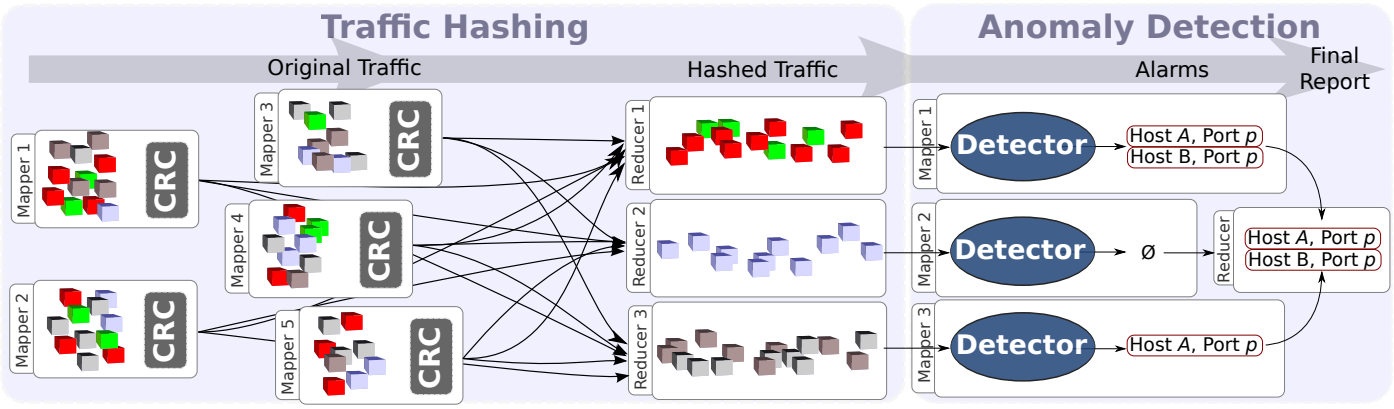


Fig. 1: Overview of Hashdoop. In this example, the original trace contains 5 splits that are hashed in $N = 3$ buckets. Each bucket is independently analyzed by a detector and all detectors results are summarized in the anomaly report. Note that here traffic is hashed only once for clarity but in fact Hashdoop produces $2N$ buckets for source and destination hashes.

traffic using only port number [2], [12], host address [6], or origin-destination flow [9].
Usually, time is partitioned by using a sliding window or time bins.

2) Normal traffic modeling is the statistical analysis of discretized traffic aimed at uncovering the usual behavior of traffic and computing a normal reference. Being the hardest step of the three, the research community has experimented with numerous approaches to model normal traffic, for example, using principal component analysis [9], entropy [12], gamma modeling [6], Kullback-Leibler divergence [2], image processing [8], or equilibrium model [14].Note that traffic is hashed only once for clarity. We stress that these statistical tests are usually relevant only if there are enough flows per bin and the number of bins is sufficient.
3) The last step compares a sample flow with the computed normal reference and classifies it as normal or anomalous. This is usually achieved with simple, yet robust, thresholding techniques [2], [6], [8], [9], [12], [14].

Implementing this general approach with MapReduce is not trivial as the traffic discretization and MapReduce data splitting might interfere.

B. MapReduce

MapReduce is a programming model designed to process large a dataset by using a cluster of computers. A MapReduce program is composed of two key procedures, *map* and *reduce*, that are executed on a cluster where each node carries splits of the dataset. Thereby, all splits are independently processed by numerous concurrent mappers; then, the mappers results are shuffled, sorted, and passed to the reducers that digest them into the final results. A MapReduce program is therefore inherently parallel and can greatly decrease the computational time when processing huge datasets.

The MapReduce implementation of a network traffic anomaly detector is fairly straightforward. The map procedure detects anomalies in a split, and the reduce procedure summarizes all anomalies detected. Nevertheless, particular care should be taken when dividing traffic into smaller splits. Usually, the dataset is sliced into consecutive fixed-size-splits, but in the case of packet traces, this simple slicing creates splits, i.e., sets of packets, with two undesirable properties. First, the time duration of the traffic is dramatically decreased and varys along with the packet rate of the captured traffic. Second, related packets, e.g., packets from the same flow, can span over several splits.

As explained above, anomaly detectors perform temporal discretization of the traffic by using binning or sliding windows. However, fixed-size slicing significantly shortens the time duration of the traffic analyzed by a mapper; thus, it dramatically reduces the number of times bins are used for the statistical analysis and prevents detectors from properly modeling the normal traffic.

Furthermore, distributing traffic sent or received by the same host into several splits can weaken distinctive features of anomalies and make them unidentifiable. For example a network scan is a sequence of tentative connections characterized by a typical spatial distribution; it is particularly difficult to detect if the sequence is broken up into smaller parts.

III. Methodology

We propose a novel MapReduce framework that consists of two steps: First, using a hash function, the traffic is divided into splits where both the spatial and temporal structures of the traffic are preserved. Second, detectors identify anomalies in each split of data. The anomalies are then collected and reported to network operators. As shown in Figure 1, these two steps are implemented as two distinct MapReduce programs.

A. Traffic Hashing

To overcome the two problems stated in the previous section, we longitudinally slice traffic traces with the hash function. Using IP address as the key for the hash function allows us to divide traffic in splits (hereafter called “buckets”) while preserving the time duration and flow consistency. In fact, the traffic is hashed twice, once with the source IP address

as the key and once with the destination IP address as the key, thus assuring that the traffic sent or received by a certain host falls in a single bucket. The hash function we use is the cyclic redundancy check (CRC) algorithm, which is commonly used for traffic load balancing [4].

Suppose that a packet trace is distributed on several nodes by using the classical fixed-size-split and N is the number of buckets per hash function. Then, the traffic is hashed by using the following map and reduce procedures.

1) Map: For each packet p in a split, the map computes two hashes, $h_s = CRC(srcIP) \mod N$ and $h_d = (CRC(dstIP) \mod N) + N$, where $srcIP$ and $dstIP$ are respectively the source and destination IP address for the packet p. Thereby, the map outputs two key-value pairs, where the key is composed of a hash (h_s or h_d) and the timestamp of p, and the value is the packet p.

2) Shuffle and Reduce: Using the keys of all mappers outputs, the shuffle separates packets into $2N$ buckets, and within each bucket, sorts packets chronologically.

The reduce procedure reads a single bucket, and, regardless its size, writes it to the file system as a single split. Hence, reducers on different nodes can concurrently process all buckets, store them evenly on the cluster, and ensure that a bucket is not divided into smaller splits.

B. Anomaly Detection

Using traffic hashing makes the MapReduce implementation of a network traffic anomaly detector fairly straightforward. The map procedure implements the anomaly detection for a single bucket, and results for all buckets are summarized by the reduce procedure. The key advantage of Hashdoop is that virtually any classical anomaly detector can be used for the map procedure.

1) Map: We experimented with two anomaly detectors for the map procedure, a simple packet count based detector and Astute [14], an anomaly detector monitoring traffic stationarity.

The packet count based detector reports hosts that are sending or receiving significantly more packets than other hosts. Let $\bar{c}$ be the mean number of packets sent or received by hosts in a bucket and s the corresponding standard deviation; then, hosts with a packet count c, such that

$$c > \bar{c} + \tau s$$

, are reported as anomalous. τ is an arbitrary threshold set at 3 in our experiments.

On the basis of an equilibrium model, Astute seeks flows that violate the traffic stationarity. Astute inspects the traffic by using time bins and six different levels of flow aggregation and reports a significant shift in the number of packets or bytes for two consecutive time bins. The shift between the time bins i and $i+1$ is measured with the following equation.

$$K'_i = \frac{\hat{\delta}_i}{\hat{\sigma}_i}\sqrt{F}$$

, where $\hat{\delta}_i$ and $\hat{\sigma}_i$ are respectively the mean and the standard deviation of the volume changes for the F flows present in the time bins i and $i+1$. The time bin i is said to be anomalous if its shift exceeds a certain threshold K, $|K'_i| > K$. This threshold controls the false positive rate and is set to 3 in our experiments.

Both map procedures report alarms in the form of a key composed of six values, the source and destination IPs, source and destination ports, and starting and ending time of the anomalous traffic, and can contain wildcards for unspecified values. This unified output allows us to implement the same reduce procedure for both detectors.

2) Reduce: The goal of the reduce procedure is to compile and report detected anomalies to the network operators. Thereby, the reducer collects all alarms reported by the mappers, digests alarms reporting the same traffic, and produces the final anomaly report.

IV. Evaluation

We now evaluate the advantages and shortcoming of Hashdoop to provide real-time network traffic anomaly detection. We investigate two key elements that are, first, the overheads of traffic hashing and MapReduce, as compared with the benefits of parallel computing (Section IV-B), and second, the impact of traffic hashing on the detection performance of anomaly detectors (Section IV-C).

A. Hadoop Cluster and Dataset

The proposed framework was evaluated using a local Hadoop testbed of six nodes connected with 1-Gigabit Ethernet. Nodes ranged from commodity workstations with a single quad-core processor and 8 GB of RAM to higher specification servers with two 8-core processors and 64 GB of RAM. This 6-node cluster ran Hadoop 2.0.0 with a split size of 64 MB, a replication factor of 3, and was configured to concurrently process up to 128 mappers or 92 reducers.

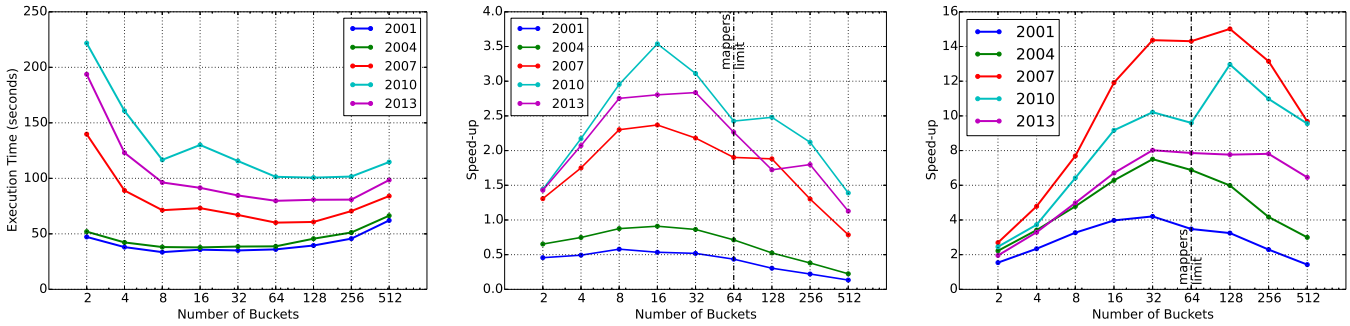
The traffic traces analyzed in these experiments are all from the MAWI traffic archive[1] and, more precisely, traffic captured at a trans-Pacific link between Japan and U.S. (samplepoint-B and samplepoint-F). This was originally an 18-Mbps link (CAR on 100 Mbps) and was updated to 150 Mbps in July 2006. In our experiments, we analyzed traffic traces captured from 14:00 to 14:15 JST on the 15^{th} of January, February, and March in 2001, 2004, 2007, 2010, and 2013[2]. We converted the pcap MAWI trace in a textual format by using Ipsumdump to ease the manipulation of these files on Hadoop. These traces contain only an IP header and port information, and no payload is kept in the MAWI archive. As shown in Table I the volume of captured traffic has considerably increased over the past 13 years; hence, the number of packets captured in 2010 is about 10 times higher than in 2001. This traffic growth allows us to inspect the scalability of the proposed method.

B. Processing Time

The total execution time for Hashdoop is the sum of the time spent hashing the traffic and processing the anomaly detection.

[1] http://mawi.wide.ad.jp/mawi/

[2] Due to intensive experiments polluting the traffic in February 15^{th} 2013, traffic from February 16^{th} 2013 was analyzed instead.



(a) Mean execution time for traffic hashing (b) Speed-up for packet count based detector (c) Speed-up for Astute

Fig. 2: Execution time for traffic hashing and anomaly detector speed-up with Hashdoop

		Traces				
		2001	2004	2007	2010	2013
Avg. nb. pkt.		2.9M	4.3M	17.7M	32.1M	25.5M
Relative Std. Dev.	4 buckets	27%	30%	13%	26%	28%
	8 buckets	26%	17%	12%	24%	23%
	16 buckets	33%	24%	15%	30%	31%
	32 buckets	41%	32%	21%	38%	40%
	64 buckets	51%	43%	28%	49%	52%
	128 buckets	65%	56%	37%	62%	68%
	256 buckets	83%	73%	48%	79%	87%
	512 buckets	106%	96%	63%	102%	112%

TABLE I: Average number of packets for original 15-minute traces and relative standard deviation of number of packets in buckets

1) Traffic Hashing: For traffic hashing, the time required to execute the map and reduce procedures is mainly bounded to three values: the size of the trace, Hadoop's split size, and the number of buckets per hash function. The number of mappers hashing the traffic is proportional to the size of the trace and Hadoop's split size, i.e., 64 MB in these experiments. For example, the smaller trace of our dataset (15/02/2001) accounts for 177.5 MB or 3 Hadoop's splits thus processed with 3 mappers, whereas the largest one (15/02/2010, 2.0 GB) occupies 33 splits processed by 33 concurrent mappers. The number of reducers writing buckets to the filesystem, however, is proportional to the number of buckets N. Because Hashdoop hashes the traffic twice, i.e. using the source IP addresses and the destination IP addresses, the number of required reducers to process them is $2N$.

Figure 2a depicts the average time measured to hash the 15 analyzed traces by using various values for the number of buckets, $N \in [2, 512]$. With $N = 2$ the average processing time for traces from 2010 to 2013 was about 4 times higher than that for traces from 2001 to 2004. However, with higher values of N, the processing time for traces from 2010 to 2013 rapidly decreased and accounts for 2 times that of traces from 2001 to 2004 with $N = 64$. Overall, the fastest processing time was achieved with $N = 8$ for traces from 2001 to 2004 and $N = 64$ for traces from 2007 to 2013, meaning that large traces can benefit from more mappers and reducers to quickly hash traffic.

2) Anomaly Detection: We measured the execution times of the anomaly detectors to entirely process each trace on a standalone computer (two 6-core processors and 12 GB of RAM) and compared them with the processing times of the detectors by using Hashdoop on our cluster. The execution times with the standalone computer reveal that the packet count based detector is an order of magnitude faster than Astute. Namely, the packet count based detector has an average throughput of 360k packets per second (pps), whereas Astute processes about 25 kpps. Figures 2b and 2c depict speed-up done using Hashdoop with the two detectors and different values of N. Here, the number of concurrent mappers, M, is directly derived from the total number of buckets, that is, $M = 2N$. Therefore, we expect better speed-up with higher values of N, keeping in mind that the maximum number of concurrent mappers for our cluster, $M = 128$, was reached for $N \geq 64$.

However, for the packet count based detector and traces from 2001 (Fig. 2b) the execution time was at best 2 times slower than with the standalone computer. This underachievement highlights the MapReduce overhead, mainly due to the propagation of the jobs over the cluster [5], and demonstrates that there is no benefit in analyzing small traces with Hashdoop. Nevertheless, for larger traces, Hashdoop performs up to 3.5 times faster than the standalone version, namely, it processes the 2010 traces with an average throughput of 1.27 Mpps. Interestingly, the best performances were obtained with $N = 16$, but for higher values of N, the MapReduce overhead became more apparent as there were less packets in the buckets and the mappers quickly completed their tasks.

Since Astute throughput is significantly lower than the packet count based detector, it requires more resources to compute the traces and consequently takes better advantage of MapReduce (Fig. 2c). For example, the analysis of the 2001 traces with the MapReduce version of Astute was about 4 times faster ($N \in [16, 32]$) than the standalone version; thus, the throughput was raised from 25 kpps to 100 kpps. With the traces from 2007 and $N = 128$, we obtained a maximum speed-up of 15, boosting the Astute throughput to 375 kpps. This result is interesting for two reasons. One, the maximum speed-up was reached with $N = 128$, thus $M = 2 * 128$

mappers, which was above the physical limit of our cluster, and second, this was done with the traces from 2007 but not with the traces from 2010 to 2013, which contained more traffic. The first observation exhibited the limits of our cluster and suggests that better speed-up for traces from 2007 to 2013 could be attained with more nodes. We investigated the causes for the second observation and found that traffic in 2007 was better spread in the buckets than for any other trace. The traffic distribution in buckets is depicted in Table I with the relative standard deviation RSD, defined as

$$RSD = \frac{s}{\bar{x}}$$

, where $\bar{x}$ is the mean number of packets per bucket and s is the corresponding standard deviation. We observed that the RSD of traffic captured in 2007 was about 2 times lower than that for traces from 2010 to 2013, meaning that the size of buckets was more even for traffic in 2007, whereas, for traffic from 2010 to 2013, a few buckets contained considerably more traffic than did the others. Since the processing time of a mapper is proportional to the size of the analyzed bucket, traffic unevenly distributed in buckets induces a few mappers to consume more processing time, thus delaying the whole MapReduce process.

Overall, Hashdoop drastically decreased the processing time for large traffic traces that cannot be analyzed in real-time with the Astute standalone detector. For example, the detection with the Astute standalone detector and the 900-second long traffic traces from 2010 took on average 1296 seconds, whereas, with the MapReduce version and $N = 128$ buckets, the traffic hashing and the anomaly detection took on average 216 seconds (100 seconds for traffic hashing and 116 seconds for the detection).

C. Detection Performance

We investigated the impact of the proposed MapReduce framework on detection performance. Our main interest here was to determine if traffic hashing rather eases the detection by isolating anomalies in buckets with less legitimate traffic, or on the contrary, it inadvertently misleads detectors to report legitimate traffic as anomalous. We measured the accuracy of the detectors with Hashdoop and their standalone version by using MAWILab anomaly reports [7] as ground truth data and the traditional F-score, defined as:

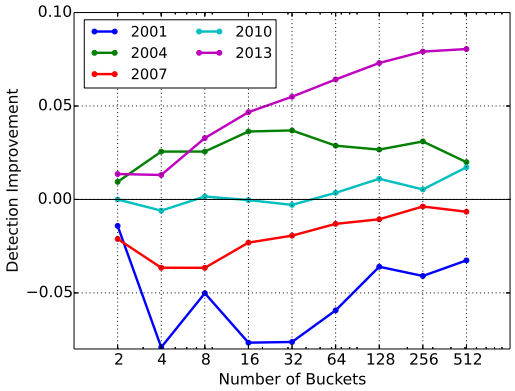
$$F = \frac{2TP}{2TP + FN + FP},$$

where TP, FN, and FP are respectively the number of True Positive, False Negative, and False Positive alarms. The detection improvement DI of Hashdoop against the standalone one is then the difference between their respective F-score, F_{MR} and F_S:

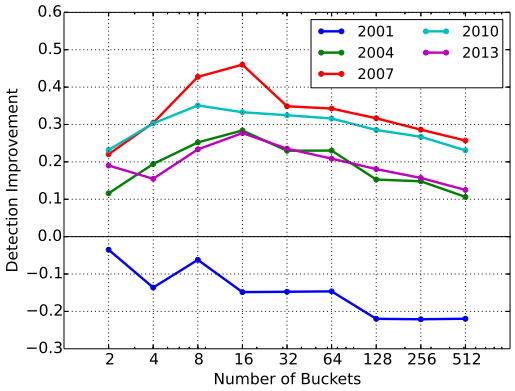
$$DI = F_{MR} - F_S.$$

Therefore, positive (resp. negative) values of DI indicate that Hashdoop performs better (resp. worst) than the standalone version.

Figures 3a and 3b depict the average detection improvement for the packet count based detector and Astute. For traces in 2001, the proposed framework with either detector achieved inferior F-scores as compared with those from the standalone version, and globally, the detection improvement decreased as



(a) Packet count based detector



(b) Astute

Fig. 3: Detection improvement (DI) for the detectors with Hashdoop and various numbers of buckets. Detection improvement is measured as difference between F-measure of classical detector and that with Hashdoop.

N increased. These poor performances were mainly due to the small number of packets contained in each bucket, which broke the detectors statistical assumptions. For traces captured from 2004 to 2013, the detection improvement for both detectors and almost all traces was greater than zero; nevertheless, both detectors exhibited different behavior when varying the number of buckets N.

The detection improvement for the packet count based detector and traces from 2007 to 2013 gradually improved as N increased, meaning that splitting these large traces in buckets emphasizes anomalies hidden in a large traffic volume. The exception being the traces in 2007, as their average detection improvement was negative. Inspecting these results revealed that the standalone detector performed extremely well for the trace from January 15^{th}, 2007 ($F_S = 0.88$, the highest score in our experiments) and globally lowered the average DI for 2007. Results for the 2004 traces were yet different; the detection improvement reached a maximum for $N \in [16, 32]$ and then decreased as N increased.

Similarly, the detection improvement of Astute for 2004 to 2013 reached a maximum around $N = 16$. The detection improvement for 2010 traces stayed reasonably constant for

higher value of N; thus, for $N = 128$, the average detection improvement for the 2010 traces was 0.28. The best detection improvement for Astute was obtained with the 2007 traces; however, we found that the standalone version of Astute was performing poorly with these traces, thus emphasizing the performance of the proposed framework. Overall, we observed a particularly low detection rate for the standalone version of Astute, and tuning Astute parameters somewhat improved its F-score, although the shape of the detection improvement curves were left unchanged, hence yielding similar results.

V. Related Work

Applications of hash functions have been researched extensively in the networking community. In the area of anomaly detection, researchers have applied independent hash functions to create multiple random projections of traffic traces called "sketches." Sketches represent different views of the traffic that assist detectors to pinpoint anomalous flows [2], [6]. Schweller et al. [13] proposed a reverse hashing method to identify IP addresses responsible for heavy changes in the traffic. Diverse sketching techniques have recently enabled efficient measurements on software-defined networks (SDN) [16]. Our utilization of the hash function more resembles load balancing [4], but like any anomaly detector, sketch-based detectors can certainly be used with Hashdoop.

MapReduce has recently received a lot of attention in the literature including a few studies on network traffic. Lee et al. implemented a scalable platform for traffic analysis that uses Hadoop [10]. They implemented a specific wrapper to read pcap files on Hadoop and showed its benefits with several traffic analysis tools. Our proof of concept implementation is currently using traffic in textual format, but we are planning in future work to use this wrapper or a similar one [1] to improve the performance of Hashdoop. In addition, Logothetis et al. proposed a MapReduce platform [11] to distribute processing tasks close to the nodes that capture data. This in-situ approach reduces the cost of data transfer and is also part of the future plans for Hashdoop.

VI. Conclusion

In this paper, we studied the applicability of the MapReduce model to detect Internet backbone traffic anomalies in real time. We found that the classical data slicing used for textual documents breaks spatial and temporal traffic structures, which dramatically deteriorates anomaly detector performance. Therefore, we proposed Hashdoop, a framework that preserves spatial and temporal traffic structures by splitting the traffic with a hash function, thus permitting anomaly detectors to be processed with the MapReduce model. This framework was evaluated with a 6-node Hadoop cluster, fifteen backbone traces captured between 2001-2013, and two anomaly detectors, a packet count based detector and Astute. Our experiments revealed that the proposed framework improved the packet count based detector throughput from 360 kpps to 1.27 Mpps and from 25 kpps to 375 kpps for Astute, thus dividing its processing time by 15. In addition, we observed better detection performance with Hashdoop as hashing projects anomalies in buckets with less background traffic. However, we observed that dividing traffic into too many splits may cause adverse results because each split may contain insufficient traffic for the statistical tests of anomaly detectors. This trade-off is dependent on the anomaly detector used and will be studied in future work.

The implemented proof of concept benefits from Hadoop scalability, scheduling, and fault tolerance, which are particularly useful for anomaly detection. In future work, we are planning to investigate these benefits in order to run different detectors in parallel and combine them as it is done in MAWILab [7].

Acknowledgments

This research has been supported by the Strategic International Collaborative R&D Promotion Project of the Ministry of Internal Affairs and Communication in Japan and by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 608533 (NECOMA). The opinions expressed in this paper are those of the authors and do not necessarily reflect the views of the Ministry of Internal Affairs and Communications, Japan, or of the European Commission.

References

[1] Ripe hadoop pcap. *https://labs.ripe.net/Members/wnagele/large-scale-pcap-data-analysis-using-apache-hadoop*, 2011.

[2] D. Brauckhoff, X. Dimitropoulos, A. Wagner, and K. Salamatian. Anomaly extraction in backbone networks using association rules. *IMC '09*, pages 28–34, 2009.

[3] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina. Impact of packet sampling on anomaly detection metrics. In *IMC'06*, pages 159–164, 2006.

[4] Z. Cao, Z. Wang, and E. Zegura. Performance of hashing-based schemes for internet load balancing. In *INFOCOM'00*, volume 1, pages 332–341, 2000.

[5] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[6] G. Dewaele, K. Fukuda, P. Borgnat, P. Abry, and K. Cho. Extracting hidden anomalies using sketch and non gaussian multiresolution statistical detection procedures. *SIGCOMM LSAD '07*, pages 145–152, 2007.

[7] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda. MAWILab : Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. *CoNEXT '10*, 2010.

[8] R. Fontugne and K. Fukuda. A hough-transform-based anomaly detector with an adaptive time interval. *ACM SIGAPP Applied Computing Review*, 11(3):41–51, 2011.

[9] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. *SIGCOMM '05*, pages 217–228, 2005.

[10] Y. Lee and Y. Lee. Toward scalable internet traffic measurement and analysis with hadoop. *SIGCOMM Comput. Commun. Rev.*, 43(1):5–13, Jan. 2012.

[11] D. Logothetis, C. Trezzo, K. C. Webb, and K. Yocum. In-situ mapreduce for log processing. In *2011 USENIX Annual Technical Conference (USENIX ATC11)*, page 115, 2011.

[12] G. Nychis, V. Sekar, D. G. Andersen, H. Kim, and H. Zhang. An empirical evaluation of entropy-based traffic anomaly detection. *IMC '08*, pages 151–156, 2008.

[13] R. Schweller, A. Gupta, E. Parsons, and Y. Chen. Reversible sketches for efficient and accurate change detection over network data streams. In *IMC'04*, pages 207–212, 2004.

[14] F. Silveira, C. Diot, N. Taft, and R. Govindan. Astute: Detecting a different class of traffic anomalies. In *SIGCOMM'10*, pages 267–278, 2010.

[15] T. White. *Hadoop: the definitive guide*. O'Reilly, 2012.

[16] M. Yu, L. Jose, and R. Miao. Software defined traffic measurement with opensketch. In *NSDI'13*, pages 29–42, 2013.