Wormhole: a novel big data platform for 100 Gbit/s network monitoring and beyond

Rafael Leira*†, Lluís Gifre*, Iván González*†, Jorge E. López de Vergara*†, Javier Aracil*†
*Universidad Autónoma de Madrid, Spain.
†Naudit High Performance Computing and Networking, S.L., Spain.
{rafael.leira, lluis.gifre, ivan.gonzalez, jorge.lopez_vergara, javier.aracil}@uam.es

***Abstract*—Internet measurement and analysis is increasingly challenging as the Internet evolves, primarily due to changing-trends, speed increments or new protocols and ciphers. As such, ad-hoc monitoring equipment comes in handy, albeit cost-effectiveness impedes deployment at a very large scale. As an alternative, big data-based distributed architectures are being proposed for network monitoring and analysis. However, in light of the high throughput currently offered by 100 Gbit/s links, it turns out that state-of-the-art big data solutions fall short of capacity, unless a huge amount of computers are used. In order to effectively tackle that issue, we have created *Wormhole*: a streaming engine that circumvents existing limitations by distributing the input messages/packets coherently among different off-the-shelf analysis equipment, thus reducing costs and equipment. Should the incoming data rate be larger than the system throughput, a distributed file system can be used for temporary data storage, for subsequent filtering and in-depth analysis. The proposed solution provides on-line real-time monitoring metrics with the ability to gain further insights when required. The prototyped architecture is able to deal with 100 Gbit/s networks and can be easily scaled up to higher rates by just adding more computing nodes and/or by trimming encrypted packet payloads.**

I. Introduction

In the recent years, the share of services being deployed in large cloud data centers is increasingly growing. To convey such amounts of traffic at reasonable costs in terms of energy consumption and packet loss, high bandwidth networks are of paramount importance both to provide connectivity from the cloud data centers to the rest of the Internet and also for the internet data center high-speed network itself, *e.g.* 100 Gbit/s.[1] To ensure business continuity it is essential to tackle a number of issues such as "*Is the data center network safe?*", "*Are there performance problems?*", "*Which is the next step to scale up the equipment?*", or "*What really happened during an intrusion incident?*". To this end, network traffic monitoring capabilities are cornerstone.[2]

Two types of traffic monitoring techniques are usually considered: passive monitoring, which consists of capturing the traffic that is conveyed through the network; and active monitoring, which injects specific packets into the network with the aim of accounting some concrete metrics. In this paper, we are focused on passive monitoring techniques that, in turn, can be divided into two different data collection techniques: packet sampling, which consists of capturing and accounting some periodically and randomly selected packets to infer the traffic models and conditions; and full packet capture, where all the traffic is analyzed in a packet-per-packet fashion [1]. Clearly, a full capture analysis requires much more processing and storage capacity than a sampling-based solution.

In some network scenarios, like video distribution, on-line gaming or social networks, sampling techniques could be enough to detect attacks such as a Distributed Denial of Service (DDoS), where huge amounts of similar packets are received in short time periods. However, sampling is not always desirable [2], and the monitoring system should be able to capture every packet traversing the network. For example, in critical scenarios such as banking or govern applications, all packets entering the data center must be analyzed to prevent low-speed DDoS [3] among others.

The abovementioned requirement, in turn, poses an additional problem: the increase in data traffic, now reaching aggregated 100 Gbit/s in modern data centers, also requires processing the traffic at that line rates for real-time monitoring. However, such traffic processing is very hard to achieve in practice as in the worst-case scenario in a 100 Gigabit Ethernet (GbE) link, we can expect nearly 150 million of 64-byte packets per second, resulting in a processing time of less than 7 ns per packet. As a reference value, a single L2 cache hit requires near 5 ns to be resolved in a new-generation CPU[3]. Despite we can split the analysis in different cores, current memory bandwidth still limits the analysis speed. Moreover, high-end equipment is needed with this purpose, and resiliency aspects must also be taken into account. Actually, the whole data center monitoring is put at stake if a single monitoring node is employed and a hardware failure happens. In summary, the system should be dimensioned to support traffic peaks, to prevent attacks that could affect the monitoring system.Thus, the following problems have been identified in these 100 Gbit/s networks: (i) high monitoring cost, (ii) lack of scalability, and (iii) analysis complexity.

In order to overcome these problems, this paper contributes with the first, to the best of our knowledge, big data-based architecture designed to capture and analyze passively monitored traffic in 100 Gbit/s networks. The rest of this article is organized as follows. In Section II, we briefly discuss current solutions using passive full-packet capture monitoring techniques. Next, in Section III, we compare existing streaming engines and present our own solution, together with our distributed monitoring architecture, while in Section IV we present our preliminary results. Finally, in Section V, we draw the main conclusions and future work.

[1]https://www.geekwire.com/2017/amazon-web-services-secret-weapon-custom-made-hardware-network/ [15 Dec. 2018]

[2]http://www.datacenterknowledge.com/archives/2016/12/08/network-monitoring-first-line-defense [15 Dec. 2018]

[3]https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf [15 Dec. 2018]

II. Related work

High-speed network monitoring is not a recent problem, and it has been dealt with from different viewpoints along the years. If we focus on 100 Gbit/s network monitoring, the first monitoring prototype was designed in 2012 [4], where the authors proposed to split the traffic into different 10 Gbit/s links, which could be analyzed at that time by IDS working with PF_RING [5]. Although functional, this architecture needed one physical equipment for every effective 10 Gbit/s stream to be analyzed. This seemed to be the only way to monitor network equipment, until the arrival of Flowscope [6] and Cento [7]. Flowscope can only analyze the traffic looking for preconfigured events and store small fragments of traffic around that event. On the other hand, Cento has much more capabilities: It allows real-time flow generation, IPFIX exports, DPI with selective packets stored to disk, and Cassandra database insertions. Cento needs many high frequency cores in the same NUMA node to work and may have a 20%-40% packet loss when all its functionality is activated.

In other words, it is difficult to monitor and analyze a 100 Gbit/s network using a single probe without information loss. Consequently, a *divide and conquer design pattern* has been used to build many scalable architectures, being Lambda [8] the most popular one. This architecture poses that data path can be forked in two ways: on-line *streaming* and off-line *batch* processing. In this way, light and critical processing can be done in real or near-real time, whereas detailed in-depth analysis is performed in programmed batches or on demand. An example of Lambda architecture applied to Netflow processing is presented in [9], where authors using a 10 node cluster can compute up to 315 Kflows/s in stream processing and 2.6 million flows/s in off-line analysis. In [10] and [11], authors proposed a big data off-line and multi-computer analysis system over previously captured traces. Both papers provide solutions to process PCAP files using Hadoop MapReduce and Hive SQL-like queries. Finally, Crail [12] appears to be a possible solution for 100 Gbit/s processing, but it is based on RDMA technology and thus focused exclusively on storage.

III. Proposed Solution

The design of a big-data systems based monitoring system is rather involved, which implies that an adequate choice of the design principles largely conditions the final outcome. Given that the final objective is to capture and monitor traffic at very high speed and our minimal data entity is a packet conceptually encapsulated in a message, our system has to sustain high rates through all its components. Then, the initial filtering and the decision of what packets and data have to be stored is a prerequisite to ensure feasibility in terms of hardware and costs. As it turns out, it is sensible to use a derivate of the Lambda architecture, where a streaming flow performs a pre-filtering before data is stored for off-line processing.

Current streaming engines, such as Storm, Spark Streaming or Flink [13], have several advantages, such as standardization, a relative ease of use and, overall, proven stability and scalability. Nevertheless, to meet these objectives, such approaches have partially obliterated the way in which they use network resources, preventing from using all available bandwidth in an Ethernet link. Authors in [13] presented a comparison among the above-mentioned streaming engines, but, even though the paper provides useful insights regarding latency and internal batch management, it does not give any details on their effective bandwidth usage. In a previous paper [14], authors did a performance comparison, where they proved that the number of nodes incremented linearly the cluster performance and the stream processing capacity. However, they needed 60 nodes to ingest data at a line rate of 20 Gbit/s. This effect was also observed in [15], where an alternative to current streaming engines was proposed, taking advantage of the SDN technology. This prototype showed several improvements in the effective bandwidth and latency with respect to basic solutions. Nevertheless, its applicability is limited to SDN environments, and the rate in number of messages (tuples) per second (about 140 thousands) is far away from the number of messages to be processed in a 100 Gbit/s network As an example, for an average packet size of 900 Bytes, 13.3 Million of packets would be transmitted per second. After assessing some of the most popular streaming engines and verifying that the state of the art works reflects the actual performance of those frameworks, we decided to implement our own stream engine, which we have named *Wormhole*.

A. Wormhole design and dynamic behavior compilation

In this section, we briefly introduce the *Wormhole* architecture. It is composed of four differentiated elements, which functions are similar to other streaming systems:

1) **Holes**: We refer by *hole* to a thread or process that uses the *Wormhole* library. A physical element can run an arbitrary number of *holes*, independently of their interconnection. It would be equivalent to a *Bolt* or a *Spout* if we use Storm terminology.
2) **Worm**: It is the term used for every messages, because they travel between different *holes* in the *Universe*.
3) **Universe**: Is the term used to define the *hole* topology, and how *holes* interact among them. It would be similar to the term *Topology* using Storm terminology.
4) **Zeus**: It is the master in charge of coordinating and deploying the *holes* defined in the *universe* in the corresponding nodes. It is equivalent to *Nimbus* in Storm.

Creating *Wormhole* faces a complex scenario, because we had to identify the problems in prior solutions and sacrifice part of their functionality to achieve a better performance. The first key point in the development is the programming language and the network stack. Works like [16] show that using Java and its network stack have a significant overhead, worsening the performance. Thus, it is crucial to use a low-level language (such as C). With respect to the operating system network stack, a context switch between the user and kernel spaces is needed to send a message. To minimize this overhead, we mimic the Spark Streaming batching approach, i.e., packing

the received messages and periodically processing them [13]. The last point to take into account is the decision about sending a message. Although this can seem trivial, it is necessary to let every computing node choose on demand which other node data will be sent to, and, potentially, do it several million times per second. Right now, state-of-the-art systems must check their transmission rules for every message to send, or even worse, send the message to a hub node. The ideal option is that each computing node software is compiled just in time and programmed minimizing their number of instructions. In that regard, each *Hole* thread compiles its own path as a native dynamic library, and links them in runtime. This decision permits to change the path in real time, while maintaining the less possible computational cost per message. In the best case, e.g., when the destination is always the same, the code will be notably minimized and optimized by the compiler.

B. Monitoring Architecture

Once we have presented *Wormhole* as a streaming engine, we can define our monitoring architecture. It is shown in Figure 1 and it is divided in 6 steps:

1) Link aggregation: The first step consists of extracting the packets conveyed through the data center's input/output links. This step is highly dependent on the interconnection topology used by the data center to reach external networks. It is clear that different interconnection topologies will motivate different traffic collection procedures. However, to have coherent monitored traffic, it is mandatory to aggregate traffic, at least, at a per-flow granularity, whereby a separate flow is a set of packets sharing a specific meaning, such as the same tuple <origin, destination, protocol>, the same target service, the same subnetwork, etc. This step is presented for completeness, but for simplicity, in this paper we will focus on the most complex scenario: a single capture point at 100 Gbit/s that aggregates the traffic of a core switch/router in a data center. However, thanks to the proposed architecture, several vantage points and aggregation processes could also be considered. We plan to study advanced aggregation processes in future works.

2) Useless Payload Reduction: Part of the passive monitoring process consist of identifying the traffic and obtaining relevant information about it. Currently, most part of the network traffic is encrypted, providing useless information above its network and transport headers. Then, to scale the system without adding analysis nodes, we have considered adding an optional bandwidth thinner implemented in an FPGA, as proposed in [17]. This approach allows a reduction in the monitored bandwidth from 100 Gbit/s to less than 10 Gbit/s by truncating the packet payload. Thus, it would potentially reach 1 Tbit/s monitoring if we aggregate the output of 10 traffic thinners, which is planned as future work.

3) Distribution: The third step is divided in two parts: (i) Traffic capture, and (ii) Distribution and balancing. The most efficient way to implement them is to use DPDK [18] for the capture and use the Receive Side Scaling (RSS) queues to distribute the captured traffic as efficiently as possible [19]. This distribution is used to encapsulate the traffic into different TCP streams. In this way, we need a single *Wormhole+DPDK*

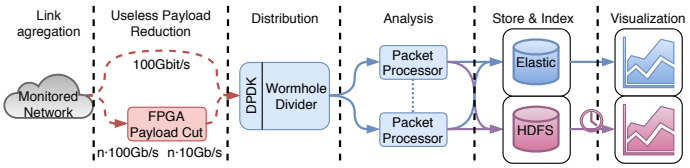

Figure 1: 100 Gbit/s monitoring architecture

application, with as much threads and queues as analysis systems we have in the analysis step, where each thread reads a packet with DPDK and sends it through the corresponding *Hole*. Because the traffic capture and balancing have to be executed by the same process, this step could be a bottleneck if balancing is performed suboptimally.

4) Analysis: The fourth step is the packet-per-packet real time analysis. This step receives the *worms* sent by the prior step in different *holes*, whose tasks are: (i) to reduce packets to a more compact element (for instance, a flow) that is stored for further detailed analysis by an on-demand/off-line tool, and (ii) to extract useful information for a real time in-line analysis. To validate our architecture we have used a flow analysis software (*Naudit DetectPro*[4]) with an estimated processing capacity of 15 Gbit/s per instance. Anyways, this analysis can be replaced by any application that a network manager needs, provided that it can be integrated in the *Wormhole* message passing system.

5) Storage and Indexing: The fifth step is the storage and indexing of preprocessed data. It is the starting point of a Lambda architecture, where real-time data (in our case flows statistics) are indexed by ElasticSearch, InfluxDB or a similar indexing engine, for subsequent visualization in real time.[5] On the contrary, bulk data (in our case flows and packets) is stored in Hadoop File System (HDFS) or similar distributed file system[5] for later execution of more time consuming tasks, like database queries, machine learning algorithms, etc.

6) Visualization: The final step is data visualization, namely displaying indexed data using Grafana or similar visualization suites[5], whereas HDFS data is readily available for future processing and visualization, e.g., using Hive[5] queries.

IV. Experimental Evaluation

One of the requirements to evaluate and analyze a 100 Gbit/s network is to have access to such infrastructure, not only to test the proposed architecture, but also to be able to capture real traffic. Unfortunately, without access to real 100 Gbit/s traffic, our approach was to use traffic from a lower speed network, for example CAIDA 2016 trace [20], and transmit it at 100 Gbit/s. In addition, we have also generated synthetic traffic with perfectly balanced IP flows and different packet sizes, in order to measure the performance of the proposed architecture in realistic (CAIDA traffic) and extreme (synthetic traffic) cases.

The transmission from pcap files at high-speed is another challenging task: storage bandwidth, memory bandwidth and computing capabilities could be a limitation. For this reason, we have developed a custom traffic generator, that is freely

[4]http://www.naudit.es/en/detectpro/ [15 Dec. 2018]
[5]http://bigdata.andreamostosi.name/ [15 Dec. 2018]

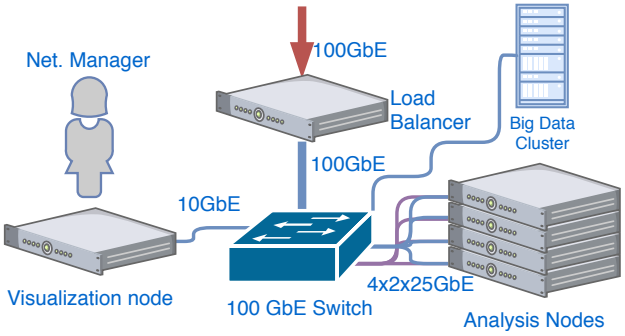

Figure 2: Experimental architecture

available[6]. The key point of the generator, at hardware level, is the use of six 800 GB *Intel P3600* NVMe drives configured as software RAID 0 together with an XFS file system optimized for managing large files. Based on the data sheet, each NVMe drive should be capable of reading at 20.8 Gbit/s, although in our experiments the bandwidth was around 20.6 Gbit/s. Using a software RAID 0, the total achieved bandwidth is 110 Gbit/s, i.e., which is enough for our purposes. The traffic generator is our equivalent of the *Link aggregation* (step 1) in the proposed architecture. The complete hardware equipment used for the design and evaluation of the proposed architecture is shown in Figure 2. All computing systems, except the Big Data Cluster, are running a *Gentoo OS* with *Kernel Linux 4.14.7*.

The *Useless Payload Reduction* step (step 2) is omitted in our tests, because the selected traffic, CAIDA and synthetic, does not include real payload. Anyway, we have implemented this functionality in a Xilinx VCU118 card with two 100 Gbit/s interfaces, and it is able to reduce CAIDA bandwidth from 100 Gbit/s to 8 Gbit/s, because the FPGA cuts all the packets due to the binary content of the payload (all 0's). For this reason, none of the measurements presented below makes use of the FPGA in step 2.

The *Load Balancer* node is in charge of all tasks related to step 3, and it includes two *Intel Xeon Gold 6126 @2.60GHz* processors and two *Mellanox Connect-X 5* network interface cards (NICs), one of them for reception of the 100 Gbit/s traffic and the other to balance and send the data to *Analysis nodes* using *Wormhole*. Due to the use of one interface to send all data to the *Analysis nodes*, a switch is required to manage the connectivity between all nodes. The model we are using is a *Huawei CE8800*, capable of processing up to 1 Tbit/s. The switch includes eight 25 Gbit/s ports connected to four *Analysis nodes* in charge of executing the analysis explained in step 4 of the proposed architecture. The reasons why we are using 2 interfaces per node are the following: (i) *Wormhole* inserts the data in a simple and custom packet format, which is transmitted using TCP. Therefore, the transmission and reception bandwidth is increased. (ii) The load balancing mechanism is not ideal and it depends on the input data, in this case, network packets. Then, executing at least two monitoring tasks per node helps reducing the problem. (iii) To transmit data back to the visualization node in real time.

[6]https://github.com/hpcn-uam/iDPDK-PcapSender [15 Dec. 2018]

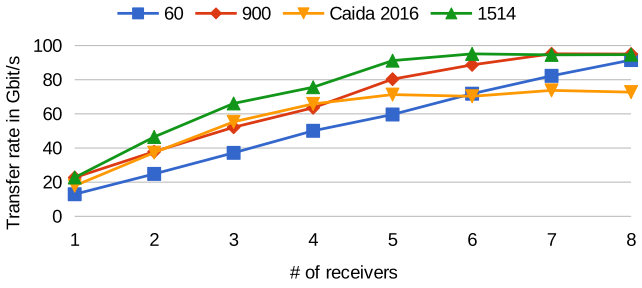

Figure 3: Wormhole performance with CAIDA traces

Analysis nodes include a double port 25 GbE Mellanox Connect-X4 NIC, two *Intel Xeon E5-2623 v4 @2.60GHz* processors and four 800 GB *Intel P3600* NVMe drives. The NVMe drives are used as local storage and they are also part of the HDFS storage system of the *Big Data Cluster*, to store the bulk data, as explained in step 5 of the proposed architecture. Notwithstanding, real-time data are transmitted using *Wormhole* to the *Visualization node* for the real-time indexing and visualization of the data (step 5 and 6). Given that the computing task of this node is not heavy, the *Visualization node* includes only one *Intel Xeon E5-1620 v2 @3.70GHz* processor, and one 10 GbE *Intel 82599* NIC for receiving data from the *Analysis nodes*. It also includes a hardware RAID 0 of four 4TB hard drives to store both the real-time data and indexed data.

Once the hardware equipment has been presented, the next step is to evaluate the performance of the architecture. Although there are several network bandwidth measurement points, we are considering the output of step 3 as effective bandwidth. This is due to the fact that all previous communication tasks have been performed correctly because they make use of the TCP protocol or other reliable upper-layer protocol, which ensures that no data loss happens. Therefore, the performance of this architecture is limited by that of the slowest component (or step).

Figure 3 shows performance results for a number of receivers with synthetic traces made of single-sized packets: 60 Bytes (minimum Ethernet frame size excluding CRC), 900 Bytes (mean packet size in CAIDA trace), the real CAIDA trace, and 1514 Bytes (maximum Ethernet frame size without VLAN nor CRC). The number of receivers in this test is the number of analysis instances distributed among the 4 nodes using a simple round-robin. This is, with 2 instances only 2 nodes are receiving; with 4 instances all nodes are receiving; with 6 instances 2 nodes are receiving in 2 interfaces, and the other 2 in only 1; finally, with 8 instances, all nodes are receiving in both interfaces.

Thus, Figure 3 shows the performance in the worst, middle and best cases,when using our available hardware. It is worth noting the trend. When we reach effective 94 Gbit/s, system TCP overload and *Wormhole* encapsulation limit the overall performance, even if we increase the number of receivers and analysis threads. On the other hand, until this point, the raise in the processing speed is linear with the number of consumers. This behavior indicates that, with a larger number of NICs

the system could grow up to 200 Gbit/s or more, even using the same analysis equipment, because many of their cores were idle. In the CAIDA trace case, we can see that up to 4 analysis threads, their behavior is very similar compared to the synthetic ones. However, the analysis throughput of the CAIDA trace reaches stationarity at 70 Gbit/s, where the system does not improve its performance although we increase the number of analysis threads.

Identifying the root cause for this behavior is rather complex, due to the interplay of the many elements involved. However, we have some hints that are currently in our research agenda for further assessment and possible mitigation. As a first approximation, we believe that micro-batches of packets with a same destination node may be responsible for such behavior. This can be a problem, because if a node is saturated, it can have a harmful effect in the TCP state machine at the emitter and the receivers, producing small TCP pauses and finally stalls in the complete system. However, other options can also be possible, such as the very nature of the CAIDA trace, or the effects in the packet size variation when processing them in memory. Therefore, we note that several research avenues may be pursued at this point, which constitute the focus of our current research.

V. Conclusions and Future Work

In this article, we have presented two novel elements: (i) *Wormhole* streaming framework and (ii) a complete network monitoring Big Data Architecture for 100 Gbit/s. *Wormhole* library has been released open source under a MIT License for both academic and industrial use[7]. As we have shown in Figure 3, the framework scales linearly with the processing nodes involved, until it reaches almost 100 Gbit/s, which is a clear limitation due to the load balancer in the 100 Gbit/s NIC as well as TCP and *Wormhole* overheads. Anyway, we believe this is a great result that promises performance above 100 Gbit/s and paves the way for breaking the Terabit/s analysis frontiers.

In addition, the performance of *Wormhole* clearly outperforms state of the art streaming frameworks, as it provided as much as 130 million messages per second in the 60 Bytes per message test, with 8 receiving threads. We are currently working on exploring and analyzing in detail the *Wormhole* architecture, to assess its performance in comparison to other solutions in the state of the art, but focusing on other metrics apart from bandwidth, such as latency, messages per thread and per core. Generally, speaking, we seek to push the limits of the *Wormhole* architecture in the quest for unprecedented performance in high-speed traffic analysis.

In terms of the monitoring architecture, we believe that our proposal is the first one considering a full path from a network aggregation point to the network manager eyes at 100 Gbit/s. However, we are still working on identifying the root causes of the performance degradation observed with real traffic.As of today, our preliminary investigations show promising results, even in case of flow misbalancing, that will be reported in the sequel.

[7]https://github.com/hpcn-uam/wormhole [15 Dec. 2018]

Acknowledgment

This work has been partially supported by the projects TRÁFICA (MINECO/FEDER TEC2015-69417-C2-1-R), and H2020 METRO-HAUL (EC Project ID: 761727).

References

[1] A. Pescapé, D. Rossi, D. Tammaro, and S. Valenti, "On the impact of sampling on traffic monitoring and analysis," in *Teletraffic Congress (ITC), 2010 22nd International*. IEEE, 2010, pp. 1–8.

[2] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina, "Impact of packet sampling on anomaly detection metrics," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '06. New York, NY, USA: ACM, 2006, pp. 159–164.

[3] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Information metrics for low-rate ddos attack detection: A comparative evaluation," in *2014 Seventh International Conference on Contemporary Computing (IC3)*, Aug 2014, pp. 80–84.

[4] S. Campbell and J. Lee, "Prototyping a 100g monitoring system," in *2012 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, Feb, 2012, pp. 293–297.

[5] L. Deri, "ncap: Wire-speed packet capture and transmission," in *End-to-End Monitoring Techniques and Services, 2005. Workshop on*. IEEE, 2005, pp. 47–55.

[6] P. Emmerich, M. Pudelko, S. Gallenmüller, and G. Carle, "Flowscope: Efficient packet capture and storage in 100 gbit/s networks," in *Proc. 16th International IFIP TC6 Networking Conference, IEEE*, 2017.

[7] L. Deri and A. Cardigliano, "Towards 100 gbit flow-based network monitoring," in *FloCon 2016*, Jan. 2016.

[8] M. Kiran, P. Murphy, I. Monga, J. Dugan, and S. S. Baveja, "Lambda architecture for cost-effective batch and speed big data processing," in *2015 IEEE Int. Conf. on Big Data (Big Data)*, Oct 2015, pp. 2785–2792.

[9] V. K. Bumgardner and V. W. Marek, "Scalable hybrid stream and hadoop network analysis system," in *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '14. New York, NY, USA: ACM, 2014, pp. 219–224.

[10] Y. Lee and Y. Lee, "Toward scalable internet traffic measurement and analysis with hadoop," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 1, pp. 5–13, 2013.

[11] W. Nagele, "Large-scale PCAP Data Analysis Using Apache Hadoop," 2011. [Online]. Available: https://labs.ripe.net/Members/wnagele/large-scale-pcap-data-analysis-using-apache-hadoop

[12] P. Stuedi, A. Trivedi, J. Pfefferle, R. Stoica, B. Metzler, N. Ioannou, and I. Koltsidas, "Crail: A high-performance i/o architecture for distributed data processing." *IEEE Data Eng. Bull.*, vol. 40, no. 1, pp. 38–49, 2017.

[13] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, and P. Poulosky, "Benchmarking streaming computation engines: Storm, flink and spark streaming," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2016, pp. 1789–1792.

[14] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, "Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters." *HotCloud*, vol. 12, pp. 10–10, 2012.

[15] J. Cho, H. Chang, S. Mukherjee, T. V. Lakshman, and J. Van der Merwe, "Typhoon: An sdn enhanced real-time big data streaming framework," in *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '17. New York, NY, USA: ACM, 2017, pp. 310–322.

[16] Y. Wang, C. Xu, X. Li, and W. Yu, "Jvm-bypass for efficient hadoop shuffling," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, May 2013, pp. 569–578.

[17] M. Ruiz, G. Sutter, S. López-Buedo, and J. E. López de Vergara, "Fpga-based encrypted network traffic identification at 100 gbit/s," in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Nov 2016, pp. 1–6.

[18] *DPDK: Programmer's Guide*, Apr. 2018, https://fast.dpdk.org/doc/pdf-guides-18.05/.

[19] S. Woo and K. Park, "Scalable tcp session monitoring with symmetric receive-side scaling," *KAIST, Daejeon, Korea, Tech. Rep*, 2012.

[20] C. Walsworth, E. Aben, k. Claffy, and D. Andersen, "The CAIDA anonymized Internet traces 2016 dataset," http://www.caida.org/data/passive/passive_2016_dataset.xml, [06 April 2016].