10 Gbit Line Rate Packet-to-Disk Using n2disk

Luca Deri*†, Alfredo Cardigliano*
IIT/CNR†, ntop*
Pisa, Italy
{deri, cardigliano}@ntop.org

Francesco Fusco
ETH Zurich
Zürich, Switzerland
fusco@tik.ee.ethz.ch

***Abstract*—Capturing packets to disk at line rate and with high precision packet timestamping is required whenever an evidence of network communications has to be provided. Typical applications of long-term network traffic repositories are network troubleshooting, analysis of security violations, and analysis of high-frequency trading communications. Appliances for 10 Gbit packet capture to disk are often based on dedicated network adapters, and therefore very expensive, making them usable only in specific domains.**

This paper covers the design and implementation of n2disk, a packet capture to disk application, capable of dumping 10 Gbit traffic to disk using commodity hardware and open-source software. In addition to packet capture, n2disk is able to index the traffic at line-rate during capture, enabling users to efficiently search specific packets in network traffic dump files.

***Index Terms*—Traffic Dump to Disk, Packet Capture, 10 Gbit Traffic Monitoring.**

I. Introduction and Background

Most network traffic monitoring and security applications such as IDS/IPS (Intrusion Detection/Prevention System) analyze the traffic as it passes by without storing it first. A persistent copy of the packet can be eventually stored on disk in sporadic conditions (e.g. in case a security flaw is detected). The copy can be triggered by specific network traffic, such as malformed packets to enable deeper analyses. In some cases it is however mandatory to store the entire network traffic persistently on disk in order to create a repository that allows network administrators to travel back in time to analyze past network traffic conditions. There are many domains where such a network "time machine" is required:

- In the contexts where the network traffic cannot be processed in real-time because the analysis processes are computationally intensive and do not allow the traffic to be processed in the given time boundaries.
- In fields such as lawful interception and network forensics, where it is compulsory to save raw, unmodified network packets as observed on the network in order to show the evidence of network communications.
- In security domains, security experts need to analyze the network traffic to detect and analyze new network threats that are not yet identified by existing tools.
- In high-frequency trading where network latencies and communications must satisfy specified constraints, it is required to keep a copy of network communications so that the recorded traffic traces can be used during troubleshooting or to settle a dispute.

Packet recording applications, often called packet-to-disk, or packet loggers are able to capture live traffic and dump unmodified packets persistently on disk. In order not to loose any information during capture, they have to operate at line-rate with any packet size and under every traffic mix. Depending on the environment where such applications are deployed, either all packets need to be recorded, or just those matching specified filters. In a 10 Gbit link the data volume to be stored for each second can exceed one Gigabyte. Loosing packets at capture times is not acceptable as it will make the recorded traces mostly useless, because they cannot be used anymore to provide the evidence of a fact.

In addition to packet filters used during traffic capture, network administrators and security officers need to extract packets matching specific criteria from recorded traces. The de-facto standard for packet filtering is the Berkeley Packet Filter (BPF) [1], used in the popular libpcap [2] library. BPF filters can be used both for filtering live traffic or to filter packets from recorded traces. Such traces are usually saved in *pcap* format, which is supported by the large majority of network monitoring and security applications. Supporting the BPF filter syntax does not necessarily imply that the same filtering engine provided by the libpcap has to be used as the BPF engine is known not to be very efficient [3]. As BPF is the de-facto format for packet filters, pcap is the de-facto format for file-based packet traces. A pcap file trace has a header that contains information including the interface type where packets have been captured, and the packet snaplen (i.e. which portion of the packet has been actually stored on disk).

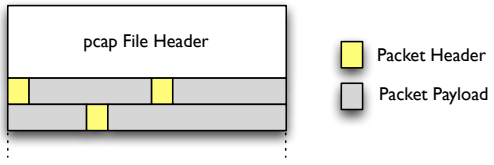

Fig. 1. Pcap File Dump Format

After this header, individual packets are dumped. Each packet has a fixed-size header that contains the timestamp corresponding to the time when the packet has been captured and two lengths: the length of the packet on the wire and the length of the data stored (i.e. the snaplen). The header does not contain any indexing metadata but just packet lengths. Therefore in order to read a specific packet it is necessary to

scan the file from the beginning until the required packet is reached.

We define a packet recording application as a tool that is able to i) dump packets to disk at line-rate in pcap format, and ii) allow users to specify BPF capture filtering expressions. The application has to come with some companion tools for enabling network administrators to extract selected packets from file traces in a reasonable amount of time, without the need to sequentially scan the entire packet repository.

In this paper we present a novel packet recorder application named *n2disk*, that we have developed from scratch by exploiting our recent research work in the field of high-speed packet capture analysis. The scope of this work is to demonstrate that expensive packet recording appliances often based on proprietary hardware (multi-1 Gbit recorders usually cost 50-75k $, and 10 Gbit recorders over 100k $) can now be built at a fraction of the cost, using the libraries and tools developed by the authors on top of commodity hardware. In contrast with most commercial packet recorders that offer as default file format a proprietary format with pcap as second option (often after proprietary-to-pcap offline file conversion), n2disk natively supports BPF filters and pcap file format so that the recorded traces can be used with a rich family of pcap-based network applications.

The original contributions of this work are manyfold:

- n2disk is the first 10 Gbit packet recorder application based on commodity hardware and open source software capable to achieve line-rate.
- n2disk supports both live packet filtering at line rate and packet filtering on traces using BPF filtering expressions.
- n2disk supports only open standards, such as the pcap file format and BPF filters, without using any vendor-specific format as main dump format from which to export pcap.
- n2disk is able to index packets at 10 Gbit line rate during packet capture outperforming state of the art [4] indexing techniques that can be used to post-process existing recorder pcap files but are not suitable to indexing in real-time at 10 Gbit rates.
- This work is royalty-free, not being based on any patented technique for packet indexing and filtering, paving the way to open, low-cost packet recording.

II. N2DISK ARCHITECTURE

n2disk comes in two flavors: single-threaded (ST) and multi-thread (MT) packet consumers. The first version is suitable for multi-Gbit networks, whereas the latter can be used for 10 Gbit networks. As in n2disk every CPU cycle matters, we decided to create two optimized application versions that can be used depending on the capacity of the monitored network links. Even if the multi-threaded version can be used as drop-in replacement for the single-threaded version, we believe that it is a good practice to avoid using too many threads when possible, so that dual-core CPUs could be sufficient for monitoring multi-Gbit links, while many-cores for 10 Gbit links.

A. Single-Threaded n2disk Packet Processing Architecture

In the single-threaded version of n2disk, there is only one packet consumer thread. The packet reader thread captures packets from a single network interface, discards packets that do not match the configured packet filter, if there is a filter configured, and copies the remaining packets into memory buffers allocated once when the application is started.

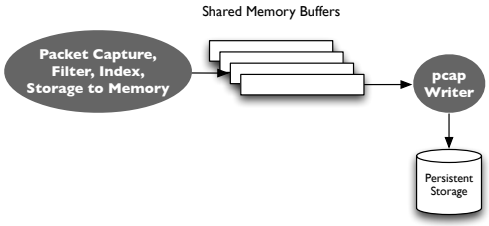

Fig. 2. Single-Threaded n2disk Packet Processing Architecture

These buffers are pre-filled with pcap file headers (see fig. 1) and captured packets are copied into the buffer together with the corresponding pcap packet header. When a memory buffer is full, the reader starts filling the next buffer and so on. The idea is that the packet reader writes packets on a format that is immediately ready to be dumped on disk. The task of writing the memory buffers to disk is carried on by a second thread, that is responsible to dump as fast as possible the available memory buffers to disk. In order to limit the overhead of data buffering introduced by the operating system primitives such as *write()*, we have taken advantage of Direct I/O on operating systems supporting it. This feature allows us to totally avoid the overhead introduced by the write storage system, at the price of using memory locations that are aligned at the page size. This can be accomplished by allocating the shared memory buffers with *posix_memalign()* instead of *malloc()*. In order to have a fully lock-less design, the writer and reader threads are fully decoupled. In fact instead of using *wait()/ signal()* for notifying the writer when a new buffer to dump is available, the writer performs an active poll with 1 usec sleep across checks, that grants low CPU load while avoiding semaphores. Packets are either captured with the libpcap API on non-Linux platforms (e.g. Apple OSX), or with PF_RING [7] on Linux.

As discussed later on this paper, PF_RING enables line-rate packet capture performance but it also allows specific features of selected network adapters to be exploited:

- On 10 Gbit Intel 82599-based network adapters or Silicom Director cards, it is possible to offload packets filtering to hardware so that incoming packets that do not match the specified criteria are dropped in hardware by the NIC itself without reaching the PF_RING framework.
- On Intel 1 Gbit 82576/82580/i350 and Silicom 10 Gbit precise time-stamping NICs, packet timestamps can be provided with high-accuracy (~80 nanoseconds) by the network adapter instead of using the system clock.

Users can specify packet filters at startup time using the BPF format. Internally, n2disk represents the string-based filter with an efficient structure that can be mapped to hardware if a capable hardware-based filtering adapter is used, or evaluated in software otherwise. During the design of our filtering engine we have decided not to support the full BPF syntax as some statements are rarely used in practice and also not usable to filter at line-rate due to performance limitations. However, users can decide to use full libpcap-powered BPF filters at the

cost of a higher filter processing overhead. The restrictions we introduced in our reduced BPF implementation are:

- Each filtering sub expression enclosed by '( )' must contain homogeneous operators and up to two nested levels are supported. Example "(X and Y and (Z or W))" is supported whereas "(X or (Y and (Z or W)))" is not.
- Some keywords such as gateway, frag, greater/less, decnet, byte ranges in packet payload are not supported.

B. Multi-Threaded n2disk Packet Processing Architecture

At 10 Gbit, the single-threaded version of n2disk can sustain wire-rate only on high-frequency CPUs (3.0 GHz and above) and only with a limited amount of software-based packet filtering rules. The reason is that the number of CPU cycles required to handle every single packet is higher than the clock cycles provided by a single core. Timestamping, for example, is an expensive operation to be done in software. In fact, computing timestamps in software requires about 80 clock ticks that is an high value compared with 134 ticks available on a 2 GHz core when processing traffic from a 10 Gbit interface at line rate. Therefore, to enable high-speed packet capture on platforms not supporting hardware timestamping, we have created a multi-threaded n2disk version that overcomes this bottleneck.

Fig. 3. Multi-Threaded n2disk Packet Processing Architecture

In this n2disk version, we delegates computationally intensive activities to various threads, so that the overall computation can be completed within the specified time constraints. This design has been possible due to PF_RING's *libzero* that is a library built over PF_RING able to dispatch captured packets in zero-copy to consumer threads while enabling threads to defer their processing (i.e. threads must not process packets in the same sequence as they have been received from the network as it happens with most packet capture libraries). By exploiting *libzero*, the packet capture thread passes incoming packets to consumer threads in packet batches to improve memory temporal and spatial locality (e.g. up to 32 packets to the first consumer, then up to 32 packets to the second and so on). In fact, policies such as round-robin that assign packets to threads in circular order, would result in increased pressure on the memory subsystem both in terms of page faults and cache misses. If capture filters are not specified, the capture thread queues pointers of captured packets to the consumer thread queues without inspecting the packet itself. In contrast, if a packet filter is defined, the capture thread must also parse the packet and match it against the specified filter before passing it to consumer threads. It is worth to remark that even though modern NICs allow incoming packets to be balanced in hardware to multiple RX queues via the RSS (Receive-Side Scale) mechanism, we decided not to use this feature in n2disk. This is because RSS would cause the packets to be saved in an order that is different from the order in which packets are received by the network adapter.

Consumer threads are responsible to copy packets into the shared memory buffers, optionally index them by exploiting the parse metadata produced by the capture thread. When there are no capture filters configured, as no metadata has been made available by the capture thread, the consumer thread must first parse the packet before updating the index. Instead, when a capture filter is defined, the capture thread is responsible to both filter the packet and update the index.

C. Indexing PCAP Files

In addition to packet capture and dump to disk, n2disk can be configured to create in real-time a packet index. The index accelerates the extraction of selected packets matching a specified filter on dumped files. For each .pcap file created by n2disk, a companion index file is created. Its format is depicted in fig. 4. During packet capture, indexes are stored on some extra memory buffers allocated during application startup. When indexes are used, for each packet memory buffer, n2disk allocates a companion index memory buffer. The main index design goals are:

- Ability to process traffic at 10 Gbit while dumping packets on disk (i.e. no post-processing is required).
- Uncompressed index size independent of captured packet types: we want to avoid that the index complexity and size depends on captured packet type so that we can set an upper-bound on the number of CPU cycles we need in order to create the index. This is mandatory to guarantee line rate packet capture under every traffic condition.
- Ability to use the index for immediately making a decision if the companion .pcap file is likely to contain (unfortunately bloom filters have limited false positive rates) the searched packets.
- Have a full 5-tuple packet digest for each packet, so that we can support complex filters and not just equality filters as it happens with hash-based indexes such as blooms.
- Ability to extract matching packets by jumping to matching-packet file offsets without sequentially scanning the whole .pcap file.

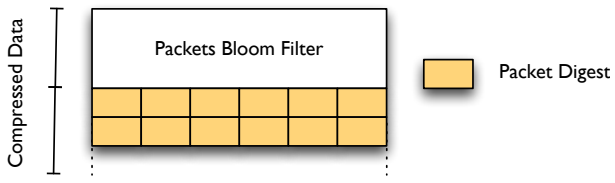

Fig. 4. n2disk Packet Index File Format

To achieve these goals we have followed a pragmatic approach. Instead of building real-indexes, such as the one proposed in [4], which cannot be built at the data rates observed in 10 Gbit networks due to computational constraints, we have decided to combine a probabilistic index with an index-less data-format to accelerate the data scanning. The bloom filters are used to avoid scanning packet traces that do not contains the result set. In contrast, the digest is used to

accelerate the linear scanning by reducing the data volumes to be scanned. In fact, the query processor has to linearly scan the data digest instead of the entire pcap trace to find packet matches. In practice, this two-layers indexing infrastructure can be created at run-time more efficiently than real-indexes, provides significant benefits at query time and does not poses excessive requirements in terms of disk utilization.

To further reduce the index space, the index is stored in compressed format using the LZF library [8] that in our tests has shown an average compression ratio of 4:1 with peaks of 10:1. We decided to use this library because of its license (BSD), because it is patent-free, and because offers a good compression/decompression speed. The index is divided in two parts that are compressed separately so that they can be decompressed individually. The bloom filter [9] is made of four statically sized different blooms:

- 2^16 bits long MAC addresses bitmap.
- 2^16 bits long TCP/UDP/SCTP ports bitmap.
- 2^16 bits long IPv4 address rounded to /24 bit-mask bitmap. This bitmap is used to speed up searches on /24 bit-masked IP addresses.
- 2^32 bits long IPv4/v6 address bitmap.

For each incoming packet, n2disk:

- Parses the packet header up to layer-4.
- Sets the corresponding bits into each individual bloom.
- Fills a 5-tuple slot in the packet digest section of the index. Packet digest slots have a fixed size regardless of the IP version and port, and in addition to the 5-tuple they contain the byte offset with respect to file begin of the corresponding packet into the companion .pcap file.

D. Filtering Packets on Disk Using Indexes

During packet dumping, n2disk concurrently creates both packet dumps and indexes on a user specified directory. The directory contains X directories named with sequential indexes (e.g. 1/, 2/, 3/...), each containing up to Y files whose name is y.pcap and y.idx. Users configure the maximum size/number of packets of each .pcap file, as well the values of X and Y. n2disk sequentially dumps files on the specified directories starting from the lowest up to the highest index. When the file X/Y.pcap has been fully dumped, n2disk starts again from the beginning by overwriting the first .pcap file. This file layout is simple and easy for setting an upper limit on storage size, but it has the drawback that it is not possible to search dumps based on packet capture time. For this reason n2disk also maintains a *dump timeline* that is basically a time-ordered directory tree whose format is year/month/day/hour/tent-of-minutes. Whenever a pcap/index file is created, n2disk adds a symbolic link on the timeline, and if a pcap/index file is overwritten, whose corresponding symbolic link is deleted from the timeline. This simple solution allows tools to search for selected packets by first selecting the set of directories contained in the specified search time boundaries, and then analyzing dumps.

n2disk comes with *findPacketsWithTimeline,* a companion tool able to extract selected packets from timeline-ordered dump files, and another tool named *findPacketsWithIndex,* for indexing existing .pcap files that lack of an index file (e.g. those that have not been produced by n2disk). The first tool takes as input the timeline directory path, the begin/end packet dump time, and a filtering expression defined on the n2disk subset of BPF (e.g. “host 192.168.1.2 and port 80”). Such filter is interpreted and the set of corresponding bits in the bloom filter computed. Once the set of files and directories have been selected under the timeline directory, the tool sequentially scans such files to dump matching packets on a time-ordered output .pcap file. For each index file, the tool decompresses only the bloom filter section, and in case the computed filter matches the bloom, the packet digest is decompressed and sequentially scanned for match. Once a match is found, the offset of the matching packet stored on the digest is used to extract the packet from the corresponding .pcap without sequentially reading the .pcap as libpcap does. In essence the bloom is a simple solution for checking if a .pcap has potential matching packets without sequentially scanning the packet digest section and thus minimizing the search time.

Traffic generation and reply of existing .pcap files is necessary for testing and troubleshooting applications, reproducing network field conditions, and evaluating the performance of network applications [16]. Hardware-based traffic generators can reproduce traffic at line rate but they are often unable to replay long .pcap files (often the limit is set to 256 packets). PF_RING DNA comes with a packet replay tool named *pfsend* that allows .pcap traces to be replayed at the original speed, at a specified speed, or at 10 Gbit line rate. This tool first reads the .pcap file containing packets to send and caches them in memory as reading them from disk during reply would limit the transmission rate. Due to this design, the maximum .pcap file size that can be replayed must not exceed the available memory that corresponds to tenth GB on modern systems, several order of magnitude better than what hardware-based traffic generators can do.

III. N2DISK PERFORMANCE EVALUATION

We have evaluated the n2disk dump and packet filtering performance using two different servers:

- A single processor low-end system (motherboard based on chipset Intel 3420 using a 2.5 GHz quad-core with Hyper Threading Xeon X3440). Each core provides around 168 CPU cycles/packet. This number can be obtained by dividing the CPU clock by the number of packets per second at 10 Gbit line rate (14.881 Mpps).
- A NUMA high-end system (motherboard based on chipset Intel C602 powered by a dual 2.0 GHz eight core with HT Xeon E5-2650, ~134 CPU cycles/packet). The system has seven Intel 520 series SSDs in RAID 0 using a LSI 9260-16i controller configured with a XFS filesystem. With rotating disks we can dump 10 Gbit line rate with at least 8 x 10k RPM disks, although in our experience it is better to use at least 10 disks.

TABLE I. N2DISK TOTAL CLOCK TICKS: SINGLE VS MULTITHREADED

	Single Thread n2disk	***Multi-Thread n2disk***
Dump w/o Indexing	149	153 (capture thread) 90 (each consumer thread)
Dump+Indexing	264	153 (capture thread) 210 (each consumer thread)

Table I highlights the number of CPU ticks required by the single-threaded and multi-threaded versions of n2disk to capture 64 bytes packets (worst case with respect to larger packets that result in fewer pps), without considering the time spent on file dump as it is a concurrent activity. This test provides an optimistic estimate of the number of threads required to dump at 10 Gbit line rate on both test systems. In essence in order to both dump and index packets, the multi-threaded n2disk has to be used because the amount of cycles-per-packet required to index and capture is higher than the cycles provided by a single core of both server platforms. For pure packet dumping without indexing, even if the high-end system has much more expensive CPUs than the low-end, the number of CPU cycles available is lower and prevents it from handling traffic at line rate on a 10 Gbit link. Table II shows how the overall application performance is influenced by packets filtering, indexing, and memory allocation.

TABLE II. STANDARD VS HUGE-PAGES MEMORY ALLOCATION ON HIGH-END SYSTEM

	Standard Memory Allocation	***2 MB Huge Pages***
Dump w/o Indexing on ST n2disk	5.83 Gbit 8.67 Mpps	6.30 Gbit 9.37 Mpps
Dump with Indexing on ST n2disk with capture BPF filter	5.04 Gbit 7.49 Mpps	5.12 Gbit 7.62 Mpps
Dump w/o Indexing on MT n2disk (2 consumer threads)	10.00 Gbit 14.88 Mpps	10.00 Gbit 14.88 Mpps
Dump with Indexing on MT n2disk (5 consumer threads)	9.48 Gbit 14.10 Mpps	9.57 Gbit 14.23 Mpps

Note that when using indexing, although we use SSDs, we need to create two files (the .pcap and the companion index) instead of one, putting extra pressure on storage that eventually causes drops as the writer thread has to spin. In order to improve the performance we have experimented with page sizes. In Linux in addition to the standard 4 KB memory pages, it is possible to use something called huge-pages whose default size is 2 MB. Their adoption offers some advantages compared to the standard size:

- Large amounts of physical memory can be reserved for memory allocation, that otherwise would fail especially when physically contiguous memory is required.
- Reduced overhead: as the TLB (Translation Look aside Buffer) contains per-page virtual to physical address mappings, using a large amount of memory with the default page size introduces a processing overhead for managing the TLB entries.

The following graph shows how the number of threads affects the performance of n2disk on the high-end system when it is capturing and indexing packets. As shown in the table I, packet indexing significantly affects the application performance due to higher per packet processing costs.

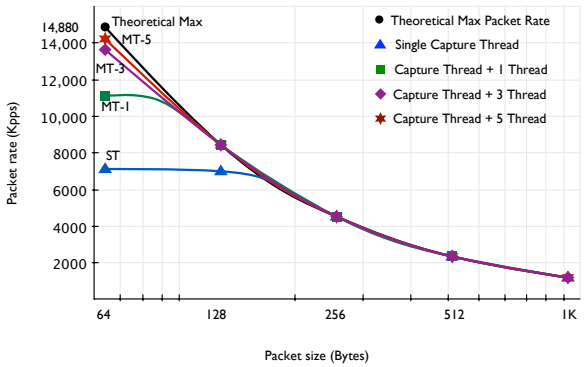

Fig. 5. Scalability on High-End System: Dump With Packet Indexing

We have also evaluated the performance of n2disk BPF-like packet filtering when compared to the BPF implementation built into libpcap. The test confirms that n2disk implementation is much more efficient with both simple and complex filtering expressions. We believe this is a good result compared to the restrictions introduced by BPF-like filters that are basically limited to the inability to filter packets based on payload content. Figure 6 shows the efficiency of packet filtering with several examples of increasing complexity, ranging from a simple "host 10.10.10.10 and port 80" filter (complexity 1) to "(host 10.10.10.10 or host 10.10.10.20 or host 10.10.10.30 or host 10.10.10.40) and (port 80 or port 3128 or port 443 or 8080)" (complexity 7).

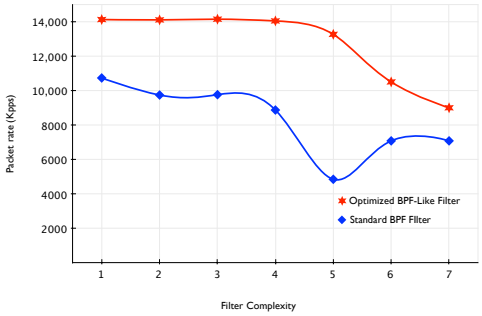

Fig. 6. Standard BFP vs. n2disk BPF-like Implementation

In order to evaluate the efficiency of the extraction tools when searching for packets we have dumped full packets on files of 1 GB each. This size roughly corresponds to one second of traffic at 10 Gbit line rate. Figure 7 reports the time required to look for packets that are not present or present in every .pcap file in a 1 to 16 TB-sized packet repository. In the first case, the search only requires the bloom filters to be accessed and it only takes around 20 seconds for a 16 Tb packet archive. Instead searching for packets that exist on every .pcap dump, which represents the worst case as we have to open all indexes and .pcap files, can be completed within the 2 hours.

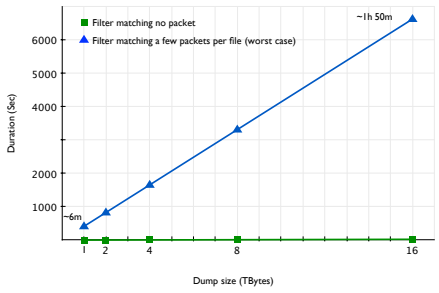

Fig. 7. Packet Extraction Performance

These figures are significantly better than the results that can be obtained when there are not indexes in place, as in case of libpcap, which requires the sequential scanning of all .pcap files. In conclusion, during packet capture/indexing, the parallelism offered by NUMA systems is not easy to exploit when packets are spread across nodes, as they have to cross the processors interconnect (QPI on our platform) that has a negative impact on the overall performance. High-end multi-core processors, such as the 16 cores CPUs we have used, do not represent a good option as well, because the amount of CPU cycles per second provided is not sufficient to achieve line rate on the single-threaded n2disk version. Considered that with less than 8 cores n2disk can dump at line rate, we recommend to use a CPU with fewer cores but higher clock rate. As described in tests documented in our website [12], our advice to obtain line rate packet to disk is to use a single processor system (non NUMA) equipped with a high-frequency CPU (3 GHz or more) that is almost one order of magnitude cheaper than the high-end multi-core CPU used in our tests.

IV. Related Work

Packet recorder appliances have been available on the market for long time [6]. However over the last few years the emerging requirements of security and data retention markets [13] ignited the development of efficient tools for packet retrieval such as the Time Machine [11], Hyperion [14] and pcapIndex [4]. Researchers have always focused on packet indexing and retrieval [10] without tackling packet-to-disk performance issues. Our work unique as it combines these two aspects into a single application. This fact is very important, as at 10 Gbit capturing packets and then indexing them off-line is not efficient, introduces latency as indexed searches cannot be performed until the index is built, and unnecessarily stresses the storage system, which, in fact, would have to simultaneously write new packets while reading the ones that have to be indexed.

V. Future Work Items

We are currently experimenting with Graphical Processing Units to create real indexes and not just packet digests [15]. From our preliminary results, indexing packets in real-time seems to be feasible with modern GPUs by passing them the packet parsing metadata computed on the CPU. However, before migrating to a heterogeneous architecture made of CPUs and GPUs, we need to evaluate if the additional complexity and costs introduced by the adoption of GPU lead to significant packet search performance benefits compared to the current indexing infrastructure.

VI. Final Remarks

This paper has presented the design and implementation of a 10 Gbit packet-to-disk application able to operate at line rate with any packet size on commodity hardware. The evaluation has demonstrated that it is nowadays possible to both capture and index packet at line rate without experiencing packet losses by using high-frequency CPUs (3 GHz or more). Additionally, the evaluation has shown that packet indexes allow multi-TB archives to be searched within a limited amount of time. The implementation of BPF-like filters in *n2disk* demonstrated that both capture and dump packet filters can conceal efficiently with BPF expressiveness. To the best of our knowledge, our work is the first combining 10 Gbit packet dump, indexing and packet retrieval demonstrating that it is now possible to perform all these activities simultaneously, at line rate in Gbit links, and using just commodity hardware.

References

[1] McCanne and V. Jacobson. The BSD Packet Filter: a New Architecture for User-level Packet Capture, Proc. of the USENIX Winter Conference, 1993.

[2] S. McCanne, C. Leres, and V. Jacobson, libpcap, Lawrence Berkeley National Labs, http://www.tcpdump.com/, 1994.

[3] S. Ioannidis, xPF: packet filtering for low-cost network monitoring, Proc. of High Performance Switching and Routing Workshop, 2002.

[4] F. Fusco, X. Dimitropoulos, M. Vlachos, and L. Deri, pcapIndex: An Index for Network Packet Traces with Legacy Compatibility, ACM SIGCOMM Computer Communication Review, 2012.

[5] S. Donnelly, DAG Packet Capture Performance, White Paper, 2006.

[6] Napatech Inc, A Guide to Building High Performance Capture and Replay Appliances, White Paper, 2010.

[7] F. Fusco and L. Deri, High Speed Network Traffic Analysis with Commodity Multi-Core Systems, Proc. of IMC 2010, 2010.

[8] M. Lehmann, LZF: An Extremely Fast/Free Compression/ Decompression-Method, http://liblzf.plan9.de/, 2008.

[9] B. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of the ACM, July 1970.

[10] F. Fusco, M. P. Stoecklin, and M. Vlachos, Net-fli: on-the-fly compression, archiving and indexing of streaming network traffic, Proc. of VLDB Endow., 2010.

[11] G. Maier et al., Enriching network security analysis with time travel, Proc. of ACM SIGCOMM 2008

[12] ntop, BYO10GPR: Build Your Own 10 Gbit Packet Recorder, Technical Report, http://blog.ntop.org, 2012.

[13] C. R. Kalmanek et al., Darkstar: Using exploratory data mining to raise the bar on network reliability and performance, Proc. of 7th Int. DRCN Workshop, 2009.

[14] P. J. Desnoyers and P. Shenoy, Hyperion: high volume stream archival for retrospective querying, Proc. of the USENIX Annual Technical Conference, 2007.

[15] A. Nottingham and B. Irwin, Parallel packet classification using GPU co-processors, Proc. of SAICSIT '10, 2010.

[16] Napatech, Traffic Generation for the Mainstream Ethernet Market, White Paper, April 2011.