INTERNATIONAL JOURNAL OF NETWORK MANAGEMENT

Int. J. Network Mgmt 0000; **00**:2–22

Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/nem

Multi-granular, Multi-purpose and Multi-Gb/s Monitoring on Off-the-shelf Systems

Victor Moreno*, Pedro M. Santiago del Río, Javier Ramos, David Muelas, José Luis García-Dorado, Francisco J. Gomez-Arribas and Javier Aracil

High Performance Computing and Networking research group, Universidad Autónoma de Madrid, Spain

SUMMARY

As an attempt to make network managers' life easier, we present $\mathtt{M}^3\mathtt{Omon}$, a system architecture that helps to develop monitoring applications and perform network diagnosis. $\mathtt{M}^3\mathtt{Omon}$ behaves as an intermediate layer between the traffic and monitoring applications that provides advanced features, high performance and low cost. Such advanced features leverage a multi-granular and multi-purpose approach to the monitoring problem. Multi-granular monitoring gives answer to tasks that use traffic aggregates to identify an event, and requires either flow records or packet data or even both to understand it and, eventually, take the convenient countermeasures. $\mathtt{M}^3\mathtt{Omon}$ provides a simple API to access traffic simultaneously at several different granularities—i.e., packet-level, flow-level and aggregate statistics. The multi-purposed design of $\mathtt{M}^3\mathtt{Omon}$ allows not only performing tasks in parallel that are specifically targeted to different traffic-related purposes (e.g., traffic classification and intrusion detection) but also sharing granularities between applications—e.g., several concurrent applications fed from flow records that are provided by $\mathtt{M}^3\mathtt{Omon}$. Finally, the low-cost characteristic is brought by off-the-shelf systems (the combination of open-source software and commodity hardware) and the high performance is achieved thanks to modifications in the standard NIC driver, low-level hardware interaction, efficient memory management and programming optimization.

Received ...

KEY WORDS: Multi-granular monitoring; Off-the-self systems; Multi-Gb/s networks.

1. INTRODUCTION

Network managers' life has become progressively more laborious given the ever-increasing users' demands for both traffic volumes and further quality of experience, the arrival of novel and heterogeneous applications, and peaks in operational and capital expenditures [1]. To complicate matters, there is a lack of synergy between traffic capture engines and network management applications (e.g., network anomaly and intrusion detection systems, NIDS, or traffic classification tools). As a result, traffic capture devices do not incorporate the necessary flexibility to actually process the traffic, which is the ultimate goal of any network manager. To tackle this issue we present and make public $\mathtt{M}^3\mathtt{Omon}$, a monitoring framework that exploits the interaction between capturing and processing traffic, which provides: (i) a simplification of network monitoring tools, (ii) a significant performance increase and (iii) a CAPEX reduction thanks to the use of off-the-shelf systems—the combination of open-source software and commodity hardware [2].

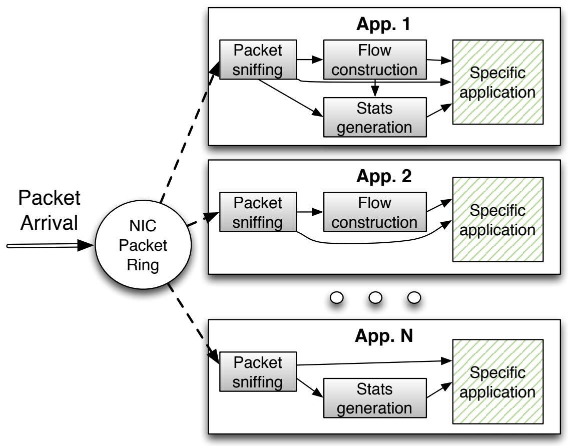
1.1. Novel features: multi-granular and multi-purpose approaches

Network trouble shooting typically entails the following three steps. First, traffic aggregates are used to spot an anomalous situation such as sudden traffic peak. Second, flow level traces are employed, for example, to identify the troublesome agents (IP address/ranges, port numbers). Third, inspection of a traffic trace is made in order to further diagnose the problem (for example, duplicated or lost packets). We refer to this kind of applications that leverage more than one granularity of data as multi-granular. However we find that the literature fails to provide practitioners with an intermediate software layer that actually provides such multi-granular data access through a unified API. We note that this is a very challenging problem because flow record logging must happen concurrently with trace storage and this is very demanding in terms of processing, parallelism and disk throughput. Our novel $\mathtt{M}^3\mathtt{Omon}$ software efficiently tackles this issue by providing an API whereby each application running over it may ask for data at different granularities. Remarkably, this is performed in commodity hardware at

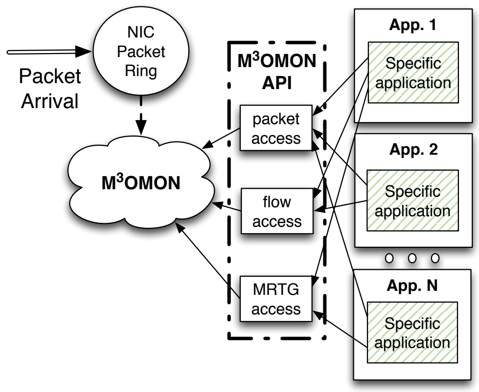
*Correspondence to: Victor Moreno, High Performance Computing and Networking research group, Universidad Autónoma de Madrid, Francisco Tomás y Valiente, 11, 28049, Madrid, Spain. E-mail: victor.moreno@uam.es very high speed. Therefore, from the point of view of a practitioner, there is not difference between asking for a packet or a flow-record. Specifically, $\mathtt{M^3Omon}$ defines in its current version three levels of granularity, namely time-series aggregates (e.g., MRTG-like series), flow records and packet traces. All of these granularities may be retrieved in real-time or after being stored in a hard drive. This is a clear departure from the current state of the art that has focused on packet capture, storage and flow record creation as separate processes. Clearly, doing all these activities in parallel is most challenging at high-speed, due to the very demanding parallel processing that must be achieved in a constrained general-purpose architecture.

Additionally, monitoring applications may actually be distributed in several physical machines, which carry out the most diverse monitoring tasks—e.g. intrusion detection and traffic classification applications. In most cases no synergies between monitoring applications are exploited at all, which implies a severe performance loss. For example, a firewall and a traffic classification application (say, for billing purposes), that both perform flow records creation in separate machines. We strongly believe that there are many synergies between monitoring applications, which can be exploited to decrease the CAPEX and increase the efficiency of monitoring systems. Indeed, $\mathtt{M^3Omon}$ performs data pre-processing at several granularities, which are made available for all the applications running on top of it. As an example, $\mathtt{M^3Omon}$ provides flow records to all the applications that require such flow-level data, thus saving the extra effort of flow record creation on a per-application basis. This novel characteristic entails a significant advantage because nowadays more and more monitoring applications use flow records [3]. Several $\mathtt{M^3Omon}$ threads must access the packet-level data at the same time —i.e., packet capture and storage, flow construction and statistic generation. This is very challenging in terms of parallel processing due to the high-speed. We note that synchronization at high-speed is very hard to achieve as it strongly penalizes performance. Therefore, even the low-level kernel interaction must be carefully planned in the traffic capture and storage chain, which drove the development of an ad-hoc driver called `HPCAP`, that we also describe in this paper.

Figure 1 presents a workflow description that portrays the conventional decoupled monitoring applications scenario compared to our current proposal. On the one hand, Figure 1a depicts the



(a) Conventional approach



(b) M^{3}Omon

Figure 1. Contrast between our approach and a conventional one

conventional approach for a monitoring probe, where several applications individually retrieve traffic from the NIC, separately pre-process the data on their own and finally execute a specific monitoring task. On the other hand, Figure 1b highlights our novel approach, where the proposed framework provides a common pre-processed data source for all the specific applications. This data source is the output of M^{3}Omon, which is the only module capturing packets and pre-processing them. After that, each specific application addresses its respective final task in the same way than the conventional approach. As a consequence, repeated efforts that were previously carried out by each specific monitoring task are now moved into M^{3}Omon, resulting in a decreased complexity and higher efficiency of multi-granular monitoring applications.

1.2. High-performance in off-the-self systems

With the previous characteristics in mind, we note that current monitoring probes render useless if their capacity is not in the multi-Gb/s range [4]. Indeed, line-rate monitoring in 10GbE links involves processing at up to tens of million packets per second (Mp/s) and millions of flows per second (Mf/s).

In order to provide the multi-granular and multi-purpose features, involving both packets and flow records, several novel optimization techniques have been applied at three different levels. First, each module of M^{3}Omon and their corresponding client monitoring applications are bound to a different CPU core, thus, exploding low-level hardware affinities [5]. Second, HPCAP driver follows the recently-proposed modifications for both standard NIC driver and default network stack optimization [5], and, additionally, the number of write operations to hard drive is minimized to optimize storage throughput. Third, we have performed software optimization, efficient memory management, and tailored data structures in the M^3Omon layer itself.

We have thoroughly evaluated the performance of HPCAP and M^3Omon on a general-purpose server. The results show that the system is able to deal with 10GbE links even in challenging scenarios with small packet size. Moreover, to show the applicability of our system, we present a network traffic monitoring tool, named DetectPro, implemented over the proposed framework, which is able to provide monitoring statistics, report alarms and afterwards perform forensic analysis based on packet-level traces, flow-level records and aggregate statistic logs. DetectPro has been put in production in commercial networks from several banks and ISPs [6]. We believe that such experiences highlight the importance of the interaction between traffic measurements and statistics at different granularities, especially traffic aggregates and flow records.

As an additional contribution, we release the code of the proposed driver, HPCAP, and intermediate pre-processing software layer, M^3Omon, under an open-source license [7], which may be useful for the research community for comparison purposes and moving forward in the development of high-performance tasks on off-the-self systems.

1.3. Contributions

Finally, as a summary, the contributions of M^3Omon are manifold: (i) it features an API to facilitate the development of multi-granular applications (first M in its name); (ii) it provides a novel mechanism to construct and share data at different granularities between applications thus saving duplicated efforts (multi-purpose, second M); (iii) it works at multi-Gb/s rates after a carefully low-level hardware interaction, a new NIC driver design, and software optimization (third M); and, (iv) for the first time, all of this runs on off-the-self systems (that is the O) thus providing low-cost and additional flexibility and scalability, available under an open-source license. The rest of this paper details all these M^{3}Omon functionalities and their implementation.

2. SYSTEM OVERVIEW

Let us detail our architecture, which is made up of the three different blocks shown in Figure 2: HPCAP, M^{3}Omon and an end-user API.

The HPCAP block consists of one kernel-level module, which implements a *traffic sniffer*, responsible for capturing incoming packets at line-rate. This module instantiates a kernel-level thread in charge of polling the NIC for new packets, and copies them into an intermediate packet buffer.

M^{3}Omon runs on top of HPCAP. It consists of a set of user-level processing modules are simultaneously fed from the traffic sniffed by HPCAP. Such user-level modules are in charge of delivering packet-level, flow-level and MRTG-like data accessible by any end-user application—according to the multi -granular and -purpose features.

Finally, M^{3}Omon provides an API that allows monitoring applications to access the different granularity data both in a real-time and offline fashion. This means that applications running on top of M^{3}Omon can focus on final monitoring tasks such as DPI, statistical classification or security analysis starting from a common data base. This architecture allows users to instantiate a variable number of modules, referred as *application layer*, each of them responsible for a specific monitoring task.

Note that the different tasks and modules (sniffing, flow handling, statistics collecting, multiple level traces dumping, specific monitoring) are simultaneously running on different CPU cores. Performance is not degraded due to a careful scheduling and to the use of CPU affinity techniques—i.e., the execution of each thread bound to a different core. Such schedule allows the system to make the most of contemporary computer architectures.

2.1. HPCAP

HPCAP [8] and other capture engines such as the ones described in [9–11] can handle line-rate traffic in high-speed networks. The use of HPCAP is additionally motivated by the specific needs of the

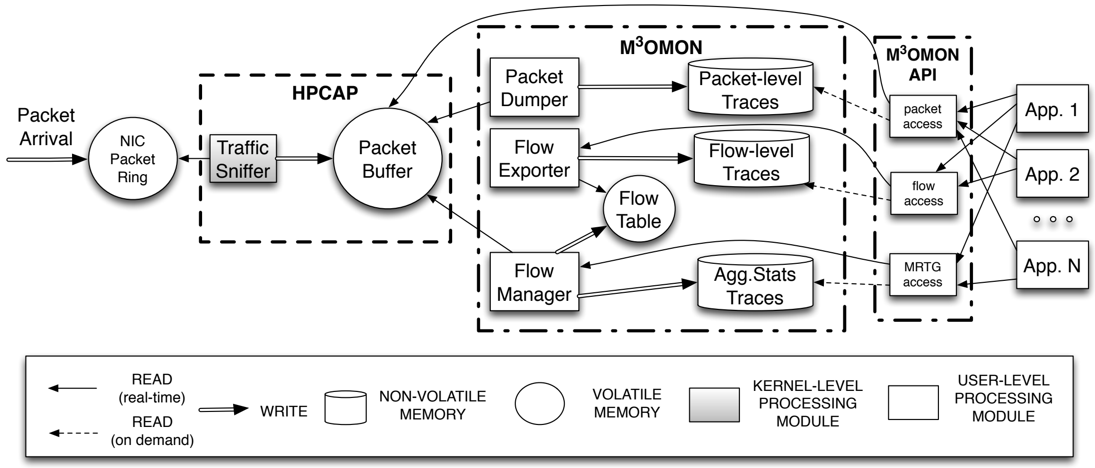


Figure 2. System Architecture.

multi-granular and multi-purpose monitoring tasks—multiple applications accessing different data and optimized persistent packet storage.

In order to maximize the amount of available CPUs in the final system, HPCAP has been designed to process a fully saturated 10GbE link using just one receive queue. Thus, for each NIC to be monitored, a kernel-level thread is instantiated and assigned to its receive queue. For optimal performance, those threads must be executed in the processor the corresponding NIC has been connected [5, 11] to. Such assignment process depends on the hardware architecture and must be applied by the user.

Once they are launched, such kernel-level threads will constantly poll their corresponding NIC's packet ring for incoming packets. If a new packet is detected, the thread makes a copy to a kernel-level packet buffer together with a header containing the packet's timestamp (second and nanosecond), capture length and actual length. Both the packet buffer's memory alignment and the byte-stream oriented data format are crucial in order to maximize performance when storing the data into a non-volatile volume. A detailed discussion on the timestamping accuracy of HPCAP can be found in [12].

The kernel-level packet buffer behaves as a circular queue. Its memory is allocated before the capture begins, avoiding per-packet dynamic allocations. Its size may be configured, with a default value of 1 GB, which is the maximum size for a kernel-level static buffer in our system. Such size is convenient for providing robustness against traffic burstiness that may appear either at wire-level or when processing data in upper-layers.

The packet buffer is mapped by any number of user level applications, all of them capable of accessing the data on it avoiding additional intermediate copies. Packet data is accessed in a single-producer/multiple-consumer basis. That is, the kernel-level updates a write pointer while each upper-layer application updates its read pointer. In order to keep data consistency, the packet read throughput corresponds to that of the slowest listener. Note that this policy isolates the reception of packets from the packet processing tasks, allowing their parallelization and thus improving the overall performance. `HPCAP` provides an API for the listeners to manage devices and access packet data. Such API allows to easily reading packets through a read packet operation similar to the PCAP library counterpart.

2.2. M^3Omon

M^3`Omon` in turns consists of three different modules in charge of generating triple-grain monitoring information, namely: Packet Dumper, Flow Manager and Flow Exporter.

Packet dumper: The Packet dumper module is responsible for generating packet-level traces in disk. The module is implemented as a `HPCAP` listener that consumes packets from the intermediate packet buffer through the driver's API. It reads fixed-size blocks of bytes (e.g., 1 MB) from the buffer and writes them into disk. Note that working with byte-blocks instead of packets minimizes the amount of write operations. This fact combined with page-alignment of those byte blocks allows the module to obtain the maximum write performance. It is worth remarking that the destination volume has to be capable of consuming the desired data throughput. In order to avoid concurrent write accesses to disk leading to a performance degradation, it is advisable that the storage device for packet traces is independent from the devices that any other processes may use.

To give more manageability and compatibility, packet traces are split in fixed-size files (e.g., 2 GB). The captured traces may be processed offline—e.g., with forensic analysis purposes. As the non-volatile storage space in the system is limited, an independent periodic process (e.g. a script invoked through the `cron` API) is in charge of deleting old capture files when the volume is nearly-full. This deletion process only implies removing filesystem's i-nodes and it has proven not to interfere in the system's performance. Note that, using the multi-granular approach, such traces may be infrequently accessed and may be bounded to specific time intervals as flow-level and MRTG statistics provide the most relevant information needed for monitoring.

Flow Manager: Concurrently, the Flow Manager module is in charge of two tasks: flow reconstruction and statistic collection. Again, the module acts as another `HPCAP` listener, reading packets from the buffer one-by-one. For each packet, its corresponding flow information and the aggregate counters are updated. This process is computationally heavy and its implementation has been tailored in several ways to achieve line-rate throughput.

To store flows, this module uses a table indexed with a hash over the 5-tuple, handling collisions with linked lists. A flow is marked as expired when it does not present packets during a given time interval (e.g., 30 seconds) or when it has been explicitly finished with TCP FIN/RST flags. Note that the timeout expiration process requires a garbage-collector mechanism, but scanning the whole hash table is computationally unaffordable. For this reason, we keep a list of active flows with each node containing a pointer to the flow record in the hash table. This active list is sorted by the last packet's timestamp in decreasing order. When a flow is updated with the information of a new packet, the corresponding node in the active list is moved to the end, keeping the list sorted with negligible computational cost. Thus, the garbage-collector only checks the first active flows in order to expire inactive flows. Expired flows (for both timeout and flags) are queued to be exported for the next module.

All the memory used during the process (structures, nodes, lists and hash-table) is pre-allocated in a memory pool in order to reduce insertion/deletion times. Once a data structure is no longer required, the memory is returned to the pool but not de-allocated. Thus, such data structure can be reused without a new allocation process. This policy significantly increases the performance of the system.

Importantly, the Flow Manager module periodically (e.g., every second) generates the MRTG statistics both writing them to a file and sending them through a multicast socket. The multicast socket allows several applications to concurrently access the exported real-time data with no additional cost, which perfectly fits the multi-purpose philosophy. Note that the use of multicast sockets allows that the applications making use of the exported data to be located on different machines, thus distributing the monitoring process. The MRTG statistics are also written to disk in case offline access to the data is needed. Such statistics are generated every second for three metrics: packets, bytes and active flows.

Flow Exporter: Finally, the Flow Exporter module instantiates a different thread which is in charge of exporting the expired flow records both writing them to disk and using a multicast socket as previously described. The multicast sockets give the same advantages for flow processing as aforementioned. Note that different multicast groups are used for the MRTG and flow data and for each network interface. This way, upper-layer applications can subscribe only to the data they desire. Flows may be exported in either an extended NetFlow or standard IPFIX formats. Each flow record contains: 5-tuple, MAC addresses, first/last packet timestamps, counters of bytes and packets, average/standard deviation/minimum/maximum for both packet length and interarrival times, TCP statistics (e.g., counters of flags or number of packets with TCP zero-window advertisements), the first 10 packet lengths and interarrivals and, if required, the first N bytes of payload, which is configurable.

2.3. M^{3}Omon's API

M^{3}Omon provides a simple and efficient API to access the multi-granular data from monitoring applications. The API provides real-time and offline access to the data gathered by the system, namely: raw packets (PCAP format), MRTG statistics and flow records. It has been designed taking as a reference the de-facto standard PCAP library.

To access real-time packet-level data, M^{3}Omon allows applications to hook as HPCAP listeners and read packets using a packet loop function similar to pcap_loop implemented in the PCAP library. Additionally, monitoring applications may need to process captured traffic on demand (e.g forensic analysis). In this case, the API offers a similar packet loop mechanism that accesses the packet data once users define the time interval they wish to analyze. Note that offline access to the packet traces is scheduled as low I/O priority process, so that write performance is minimally affected, at the expense of a higher access latency.

To access the exported flow records in a real-time fashion, the API provides a method to loop over the flow records subscribing to the corresponding multicast group. This API allows the gathering of flow records either in the same machine and in a distributed way. Additionally, the stored flow records may be accessed on demand through a method that loops over the flow records in a given time interval. Note that the flow table shown in Figure 2 and its state is not accessible through the API due to security, consistency and mutual exclusion policies.

The $\mathtt{M^3Omon}$ API provides similar methods to access MRTG information both real-time and on demand. To access MRTG in a real-time fashion, a method to loop over the MRTG registers given a multicast MRTG group is provided. To access MRTG data on demand, a time interval must be specified to loop over the data.

This approach provides great flexibility and allows monitoring applications to obtain data at any of the three granularities in an efficient and if possible distributed way. Due to the very own nature of the API implementation several applications may access any of the offered data with minimum processing overhead, in full compliance with both the multi-granular and multi-purpose approaches.

3. PERFORMANCE EVALUATION RESULTS

Our experimental setup consists of two servers (one receiver and one sender) directly connected with an optical fiber link. Both servers are based on two Intel Xeon E52630 equipped with six cores per processor running at 2.30 GHz and with 96 GB of DDR3 RAM at 1333 MHz. The motherboard model is Supermicro X9DR3-F with two processor sockets (or NUMA, Non-Uniform Memory Access, nodes) and three PCIe 3.0 slots directly connected to each processor. The NIC (10 GbE Intel NIC based on 82599 chip) is connected to a slot assigned to the first processor. Regarding the system storage, 12 Serial ATA-III disks that made up a RAID-0 are controlled by a LSI Logic MegaRAID SAS 2208 card have. These disks are Hitachi HUA723030ALA640 with SATA-3 interface and 3 TB of capacity. On the other hand, the operating system is an Ubuntu server 64-bit version with a 3.2.16 Linux kernel, with XFS filesystem—as preliminary experiments showed that is the best choice in terms of performance and scalability. The total cost of the receiver system is around $6000.

In order to inject traffic, we have developed a tool on top of PacketShader's [11] API capable of: (i) generating tunable-size Ethernet packets at maximum speed, and, (ii) replaying PCAP traces at variable

Table I. Packet sniffer and dumper modules performance when sending synthetic line-rate traffic (D=packet dumper, S=packet sniffer)

		Packet size (bytes, CRC excluded)						
		60	64	128	256	512	1024	1514
Throughput (Gb/s) $\bar{x} \pm SE_{\bar{x}}$	Theoretical Max.	7.14	7.27	8.42	9.14	9.55	9.77	9.84
	S	4.92 ± 0.02	7.27 ± 0	8.42 ± 0	9.14 ± 0	9.55 ± 0	9.77 ± 0	9.84 ± 0
	D	4.81 ± 0.02	7.27 ± 0	8.42 ± 0	9.14 ± 0	9.55 ± 0	9.77 ± 0	9.84 ± 0

rates. For our experiments, we have used both synthetic and real traffic. Synthetic traffic consists of TCP segments encapsulated into fixed-size Ethernet frames, forged with incremental IP addresses and TCP ports. Note that synthetic traffic allows us to test worst-case scenarios in terms of byte and packet throughput, but they are not useful for testing the flow-related modules. The real traffic trace was sniffed at an OC192 backbone link of a Tier-1 ISP located between San Jose and Los Angeles (both directions), available from CAIDA [13]. In order to evaluate the performance of the flow-related modules, a key metric is the number of concurrent flows rather than the throughput in packets or bytes. Replaying the backbone trace at line-rate leads to a throughput of 9.59 Gb/s[†], 1.65 Mp/s, with a maximum of 2.25 million concurrent. All the results shown in this section have been obtained by replaying the corresponding traffic along a 10 minutes period.

First, we have assessed the performance of a simple packet sniffer application (as provided by M^{3}Omon's API) and the packet dumper thread. Table I shows the mean throughput and standard error of the mean when repeating the 10-minutes experiments 50 times, for both applications and for fixed-size line-rate synthetic traffic. The table shows that both applications only loose packets in the worst-case scenario (i.e., 60-bytes packets, as CRC is deleted at NIC level). It is also shown that above this packet size all packets can be successfully captured stored into disk in a full-saturated 10 Gb/s link.

The effect of processor affinity on each module of M^{3}Omon has been studied as well. In our case, although the NIC is plugged into a PCIe slot attached to NUMA node 0, the intermediate packet buffer is allocated in node 1's memory. Thus, it seems reasonable that applications making use of that buffer benefit from being executed in node 1. Table II empirically shows this effect. The table shows the

[†]This is the maximum achievable speed due to the preamble and inter-frame gaps that the Ethernet protocol requires.

Table II. Throughput and packet loss of the different modules in the system while receiving the CAIDA-trace sent at line-rate on a 10 Gb/s link (K=kernel sniffer, D=packet dumper, P=flow process, E=flow export, S=packet sniffer app., F=flow reader app., O$_P$=offline packet reader, O$_F$=offline flow reader)

Active	Core schedule												Throughput	Packet
	NUMA node 0						NUMA node 1						(Gb/s)	loss (%)
Modules	0	1	2	3	4	5	6	7	8	9	10	11	$\bar{x} \pm SE_{\bar{x}}$	$\bar{x} \pm SE_{\bar{x}}$
D	K						D						9.59 ± 0	0 ± 0
	K	D											9.41 ± 0.04	0 ± 0.1
P+E	K						P	E					9.59 ± 0	0 ± 0
	K	E					P						6.23 ± 0.05	34.2 ± 0.4
	K	P					E						4.99 ± 0.04	47.1 ± 0.3
	K	P	E										5.53 ± 0.04	41.6 ± 0.3
M^{3}Omon	K						P	E	D				9.48 ± 0.05	1.1 ± 0.2
M^{3}Omon + apps.	K	O$_P$	O$_F$	F	F	F	P	E	D	F	F	F	9.15 ± 0.06	4.7 ± 0.1
	K	O$_P$	O$_F$	F	F	F	P	E	D	S	F	F	8.97 ± 0.05	6.1 ± 0.3
	K	O$_P$	O$_F$	F	F	F	P	E	D	S	S	F	8.97 ± 0.07	6.2 ± 0.3
	K	O$_P$	O$_F$	F	F	F	P	E	D	S	S	S	8.87 ± 0.07	6.5 ± 0.4
	K	O$_P$	O$_F$	S	S	S	P	E	D	S	S	S	7.98 ± 0.05	15.3 ± 0.4

mean and standard error of the mean (through 50 experiment repetitions) for both system's throughput and packet loss when receiving the CAIDA trace at link-speed. Note that, as the dumper application accesses the packets in a byte-block fashion, it experiences a much lower performance decrease if scheduled in the wrong node as it minimizes memory accesses: 1% compared to the 35%-45% of packet lost when flow-related module's threads are not properly scheduled. Those results show the relevance of a proper processor scheduling policy in terms of system's performance. Table II also shows the performance obtained by the complete M^{3}Omon system, when all of the modules are scheduled in the optimal slot. In such case, the system experiences a packet loss of 1.1%, leading to a global throughput of 9.48 Gb/s.

Once M^{3}Omon's performance has been assessed, we proceed to analyze the effect of instantiating additional end-user applications on top. Table II shows the overall performance when instantiating two forensic (offline) applications –one for packets and one for flows– and using all of the available cores for real-time flow record processing. In such case, a mean packet loss of 4.7% is experienced with a mean throughput of 9.15 Gb/s. This packet loss is due to the extensive usage of the non-volatile volume of this scenario: the packet dumper writes packets, and the offline packet reader simultaneously accesses the same volume. Note that packet-oriented applications can only be executed in the same machine we are sniffing traffic from. This way, we have tested the system when adding a different number of packet sniffing applications, while keeping the slot for the two forensic applications, and instantiating real-time flow processing applications in the free cores. Table II shows that adding additional packet sniffling applications may degrade system's mean throughput down to 7.98 Gb/s. Such results show that end-users may instantiate several listeners, although we note that a scalable monitoring policy should prioritize MRTG and flow-based monitoring rather than packet-based.

In terms of CPU usage, the kernel-sniffing thread (K in the table) uses 99.9% of its core. This is also true for the flow processing thread (P) and the real-time packet sniffing threads (S). On the other hand, the packet dumper thread (D) uses 60% of a core, and flow exportation threads (E) use 32% of their core. Regarding the offline data access application (O_P and O_F), they are both executed with the lowest I/O priority, leading to a core CPU usage of 42% and 23% respectively. Both real-time and offline MRTG data access applications use less than 1% of one core, so they can be executed at any core not being fully occupied –and this is the reason why they do not appear in the table. With respect to memory consumption, the flow-pool elements used by the flow manager and exporter threads has been set so that the system supports up to 450K concurrent connections entailing a use of 8GB of memory. The rest of elements of $\texttt{M}^3\texttt{Omon}$ use a negligible amount of memory compared to the previous one.

It is worth remarking that, due to the multicast export policy, the use of flow-based or MRTG-based applications does not affect system performance –regardless the NUMA node of choice. Interestingly, such applications can be executed on external machines if needed. In our test scenario, the total throughput for flow and MRTG exportation employs nearly 1 Gb/s, namely a considerable bandwidth is necessary for sending the flow/MRTG data through a distributed environment.

4. AN APPLICATION SAMPLE: DETECTPRO

Let us illustrate the functionality and applicability of the above introduced framework $\texttt{M}^3\texttt{Omon}$ with an example of a monitoring tool deployed in real bank networks. `DetectPro` leverages $\texttt{M}^3\texttt{Omon}$ to monitor network traffic without being concerned about lower level tasks (packet capturing, flow building and statistic aggregation), focusing only in network analysis. That is, `DetectPro` exploits three-level grained traces to detect anomalous events (aggregated statistics), locate hosts/network segments/services involved in the anomaly (flow records) and discover the root cause behind the anomaly (packet traces), minimizing the human intervention and coping with multi-Gb/s rates.

`DetectPro` reads aggregate statistics to diagnose both short-term and long-term changes [14] and reports the corresponding alarms, including information of detected anomalies such as start and end times of the alarm as well as the values of bit/packet/flow rates that caused the alarm. When an alarm is triggered, `DetectPro` automatically starts inspecting flow records to extract information about the network activity during the alarm period. Hence, it generates several reports containing, for instance, the distribution of hosts/network segments/services with the highest byte, packet, flow counts as well as other metrics (e.g., TCP flags, retransmissions), and the largest flows in terms of packets and/or bytes. Finally, the application selects and inspects packet traces corresponding to the alarm period. Note that such traces are automatically selected with the information previously obtained —e.g., filtering packets belonging to the busiest host.

In the following, we describe two case studies requiring the exploitation of multi-granular features by `DetectPro` to fully characterize anomalies detected in real network traffic.

First, we analyze an anomalous event observed in the traffic from a large commercial network. On June 3rd, 2013 `DetectPro` automatically launched an alarm, that indicates an increment in the number of concurrent flows from 200K up to 300K in a couple of minutes, returning to the initial steady state. Figure 3 shows some of the outputs (concurrent flows, bytes and packets time series) during the anomalous event. We can observe that neither bytes nor packets time series show anomalies, unlike concurrent flow series. This suggests that the number of connections has increased in this time interval but the increment in the involved bytes and packets is not relevant.

Once the alarm was triggered, `DetectPro` accessed flow records obtaining several metrics and statistics which helped network manager to discover the traffic involved in the event. For instance, it reported the most active IP addresses and TCP/UDP ports in terms of concurrent flows during the time interval of the anomalous event. From such reports, it was observed that DNS connections to/from two given hosts represented the 90% of the total flows during the anomalous event. Then,

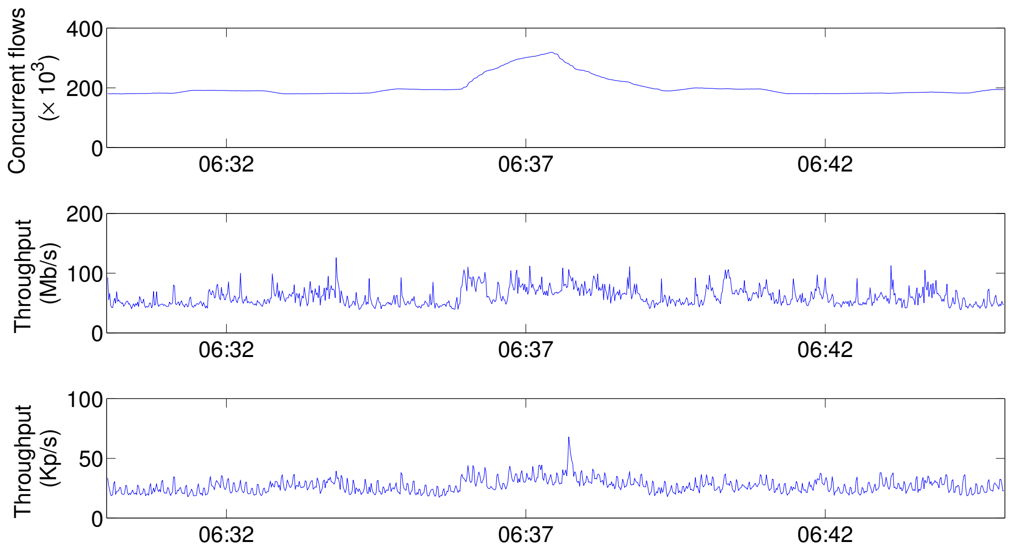


Figure 3. Time series for concurrent flows, bytes and packets during an anomalous event.

network managers may use the packet inspection feature of `DetectPro` to select some packets of the anomalous connections and infer the root causes behind the problem. In particular, the problem was related to Mozilla Firefox browser's DNS queries (updates, markups, popup blocking) whose responses were not properly answered.

Secondly, we describe the use of multi-granular monitoring for the characterization of a phenomenon detected in CAIDA San Jose datasets [13]. Figure 4 shows three time series representing concurrent flows, bytes and packets by replaying the traces of October 18th and November 15th of 2012, in both traffic directions (A and B). While replaying the trace corresponding to November 15th, an alarm was triggered. In direction A, flow concurrence reaches peaks of more than 15 million flows per second, which is more than ten times the expected value. Similarly, in direction B, flow concurrence doubles the expected value.

As stated before, `DetectPro` uses flow records to identify the hosts involved in this increase of flow concurrence. The results of this flow-level analysis reveal that hosts in the subnets represented as 40.10.0.0/16 and 238.138.39.0/24, in directions A and B respectively, generated a number of SYN-flag activated packets that exceeded in more than an order of magnitude those coming from the rest of the hosts discovered in these traces. Finally, network managers may use `DetectPro` to select

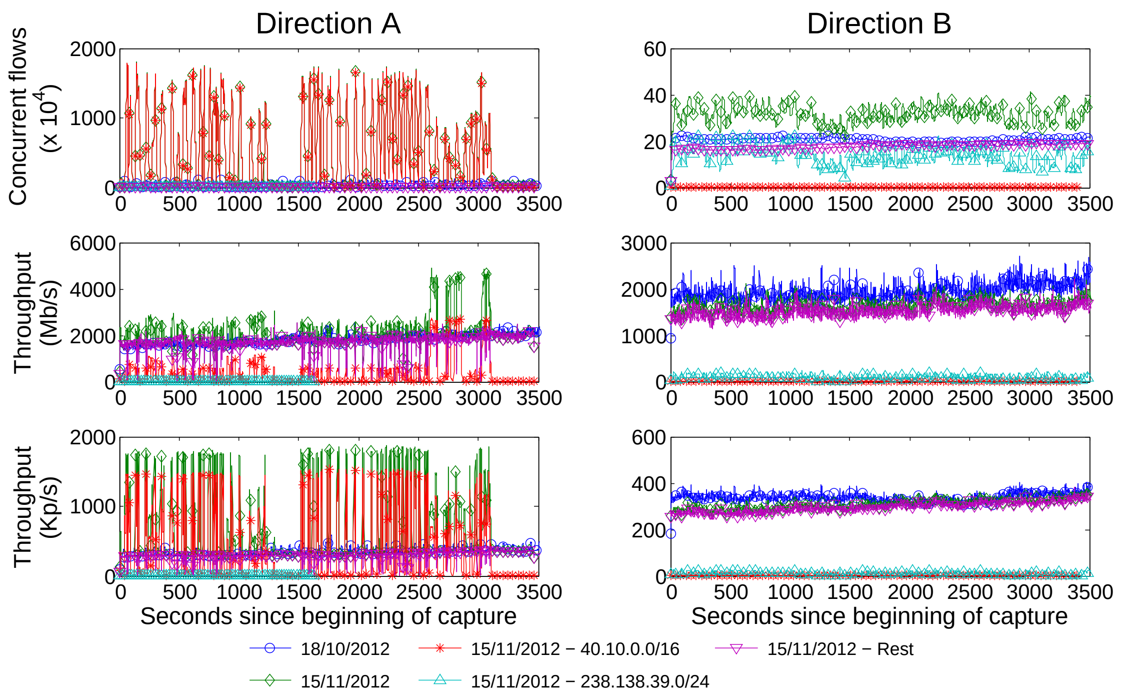


Figure 4. Flow concurrence and throughput in terms of Mb/s and Kp/s for both directions.

some packets of such anomalous connections and finally assess that the packet pattern for such hosts corresponds to two SYN flood DoS attacks against web servers.

5. RELATED WORK

In this section we present those systems that share with $\mathtt{M^3Omon}$ its target of facilitating the labor of network monitoring. Let us highlight in each case the offered flexibility, scalability, performance and kind of hardware required.

From down to top, the research community has recently paid attention to the development of open-source packet capture engines, with the aim of surpassing the traditional performance limits of the standards NIC drivers and default network stack. Some examples are `PacketShader` [11], `PF_RING` [9], `netmap` [10], `PFQ` [15], and `DPDK` [16] which grant high-performance packet capture on commodity hardware. Such capture engines modify such driver and stack to boost performance up by applying all, or some, of the following techniques: pre-allocation and re-use of memory resources, parallel processing, memory mapping, batch processing, affinity scheduling and prefetching [5]. Those approaches may be compared to our proposed driver HPCAP, which is in charge of packet capture as a subtask included in the M^{3}Omon's duties. HPCAP and the reviewed engines attain equivalent ratios of capture packets but HPCAP presents additional characteristics to assist in the development of monitoring applications. First, it allows that several processes (both applications and processes in charge of generating the different set of data grains) simultaneously access the traffic which in turn enables to offer multi -granular and -purpose capacities. And second, HPCAP cuts the number of writing operations required to store traffic in hard drive, which in turn enables to offer packet traces as the finest grain in M^{3}Omon framework.

We now turn our interest out to systems and applications that provide a more elaborated set of measurements. We acknowledge Tstat [17], an open-source monitoring tool with more than a decade of development. Tstat captures traffic, classifies it with remarkable accuracy, and generates a number of statistics aggregates at different granularities. Similarly, the authors in [18] also state the importance of forensic analysis of traffic and highlight that is enormously helpful to store packet traces to inspect them in case any problem arises. They proposed a system, TM, that collects only the first packets of each connection of the traffic under study. The rationale behind this approach is that connections follow a long-tail distribution whereby a small set of connections accounts for most of the traffic and conversely there are many connections with very little traffic. Thus, by capturing only a few packets per connection the throughput and storage requirements are dramatically reduced. On the downside, many packets are ruled out, however the authors claim that the most useful packets for monitoring purposes are the first ones, therefore the savings in space and computational burden pay off the accuracy lost. In this regard, it is worth remarking that M^{3}Omon is compatible with the operation of TM, by simply modifying the store module of M^{3}Omon to follow the packet discarding rules that TM proposes.

By comparing these proposals to M^{3}Omon, we note that neither Tstat nor TM are a framework or architecture for multi-granular and multi-purpose monitoring, and their performance are far from multi-Gb/s rates. Although Tstat highlighted the importance of multi-granular monitoring, it does not give any support to the applications running over. It leaves data available to be accessed later in an off-line fashion, being the developer who accesses the data and builds such application from the ground up. Although TM does provide user with traffic packets by simple mechanisms (specifically a tuned database), there are neither references to any richer set of data granularities nor multi-purpose deployments. By paying attention to performance, both Tstat and TM may work over commodity hardware, but they lack of optimization at packet reception level and their performance would be limited to 1 Gb/s rates.

Following a completely different approach to all the previous presented works (including ours), the authors in the technical report presented in [19] entrusted dedicated hardware with the task of high-performance monitoring. In particular, they developed a FPGA-based design name HAMOC. Similarly to Tstat and TM, HAMOC does not focus on exploiting the multi-purpose capacities of a monitoring system as M^3Omon does. Remarkably, HAMOC offers a rich set of applications modified to work over their FPGA-design, but such applications' works are isolated without sharing efforts to pre-process data. In contrast, M^3Omon features a driver that allows several threads to access to the same packet distribution queue when typically drivers permit the opposite, distributing traffic to different queues accessed each of them by a unique thread. Certainly, HAMOC pays attention to the generation of flow records at high-speeds, but there is no reference to packet storage, any other broader statistics or framework to access such data. Turning to similarities, HAMOC does share with M^3Omon multi-Gb/s performance rates. HAMOC showed rates close to 10 Gb/s both in the subtask of packet capture and application layer.

Regarding the process of coding monitoring applications, we emphasize Blockmon [20]. Blockmon allows building a monitoring application out of blocks, where each of them represents a different subtask in a given application. For example, reading a PCAP trace or filtering traffic at 4-layer may be subtasks for an application that inspects DNS traffic. Thus, Blockmon may ease the development of multi-granular applications over our proposal. That is, an application that correlates data at different granularities may be written as a sequence of Blockmon's blocks with all the support that Blockmon's library offers. The application examples provided with Blockmon use only packet-level information and run over PFQ capture engine. Using multiple queues, the system is able to process about 5 Gb/s in the worst-case scenario of small packet size—about 15% less compared to PFQ's capture process [20].

As examples of successful implementations of final applications, we remark some related to traffic classification [21, 22] and NIDS [23, 24], which are able to process the incoming traffic at 10 Gb/s rates. All of these proposals and those forthcoming may benefit from the flexibility and scalability that $\mathtt{M}^3\mathtt{Omon}$ offers to the applications running on top.

6. CONCLUSIONS

We have proposed a novel monitoring system architecture that provides network managers and network analysts with mechanisms to deploy more flexible, scalable, and affordable monitoring applications over high-speed networks.

The design is based in five key aspects which in turn comprise the contributions of this paper: (i) a novel and optimized layer between the traffic sniffer and the monitoring application themselves that provides advanced characteristics ($\mathtt{M}^3\mathtt{Omon}$), (ii) a framework to access multi-granular data, (iii) an improved NIC driver (HPCAP) for monitoring purposes, and all of them (iv) running as an off-the-self system at (v) high-speed.

$\mathtt{M}^3\mathtt{Omon}$ allows applications to run in parallel reading traffic at different granularities—time-series aggregates, flow records and packet traces. Such granularities have been constructed only once, to be shared by the set of applications running over $\mathtt{M}^3\mathtt{Omon}$. We have termed this as multi-granular and multi-purpose features, and they provide outstanding flexibility and scalability over the state-of-the-art.

Such features are easily exploited thanks to the provided framework, and viable thanks to an improved NIC driver as well as low-level hardware interactions, efficient memory management tuning, and programming optimization. This paper has comprehensively explained all these implementation details so that the paper results useful for both network managers and analysts in their duty to develop novel multi-granular monitoring tools. In this regard, we have shown a real monitoring application, DetectPro, which illustrates its successful exploitation in production environments in real networks.

To conclude, we remark that our system runs over commodity hardware as open-source software available under an open-source license [7], which gives additional adaptability, scalability, and cost-aware developments. That is, commodity hardware is easily upgraded over time and replicated over the infrastructure, and its cost is lower than dedicated solutions. High-speed is the other characteristic of our proposal, we note that the performance evaluation has shown that it reaches multi-Gb/s rates. We believe that our proposal paves the way for future migration of high-performance tasks to off-the-self systems.

REFERENCES

1. J. L. García-Dorado, A. Finamore, M. Mellia, M. Meo, and M. Munafò. Characterization of ISP traffic: Trends, user habits, and access technology impact. *IEEE Transactions on Network and Service Management*, 9(2):142–155, 2012, doi:10.1109/TNSM.2012.022412.110184.
2. L. Braun, A. Didebulidze, N. Kammenhuber, and G. Carle. Comparing and improving current packet capturing solutions based on commodity hardware. In *Proceedings of ACM Internet Measurement Conference*, 2010, doi:10.1145/1879141.1879168.
3. B. Li, J. Springer, G. Bebis, and M. H. Gunes. A survey of network flow applications. *Journal of Network and Computer Applications*, 36(2):567 – 581, 2013, doi:10.1016/j.jnca.2012.12.020.
4. Cisco Systems. Cisco network convergence system, 2013. `http://www.cisco.com/en/US/products/ps13132/index.html`, [15 March 2014].
5. J. L. García-Dorado, F. Mata, J. Ramos, P. M. Santiago del Río, V. Moreno, and J. Aracil. High-performance network traffic processing systems using commodity hardware. In *Data Traffic Monitoring and Analysis*, chapter 1, pages 3–27. Springer Berlin Heidelberg, 2013, doi:10.1007/978-3-642-36784-7_1.
6. naudit. Detect-Pro, 2013. `http://www.naudit.es/index.php?s=3&p=5&l=1`, [15 March 2014].
7. High Performance Computing and Networking Group, Universidad Autónoma de Madrid (HPCN-UAM). HPCAP and $\mathtt{M^3Omon}$, 2013. `https://github.com/hpcn-uam/`, [15 March 2014].
8. V. Moreno. Development and evaluation of a low-cost scalable architecture for network traffic capture and storage for 10Gbps networks. Master's thesis, Universidad Autónoma de Madrid, 2012. `http://www.ii.uam.es/~vmoreno/Publications/morenoTFM2012.pdf`, [15 March 2014].
9. F. Fusco and L. Deri. High speed network traffic analysis with commodity multi-core systems. In *Proceedings of ACM Internet Measurement Conference*, 2010, doi:10.1145/1879141.1879169.
10. L. Rizzo. Revisiting network I/O APIs: the netmap framework. *Communications of the ACM*, 55(3):45–51, 2012, doi:10.1145/2090147.2103536.

11. S. Han, K. Jang, K. Park, and S. Moon. PacketShader: a GPU-accelerated software router. *SIGCOMM Comput. Commun. Rev.*, 40(4):195–206, 2010, doi:10.1145/1851275.1851207.

12. V. Moreno, P. M. Santiago del Río, J. Ramos, J. J. Garnica, and J. L. García-Dorado. Batch to the future: Analyzing timestamp accuracy of high-performance packet I/O engines. *IEEE Communications Letters*, 16(11):1888–1891, 2012, doi:10.1109/LCOMM.092812.121433.

13. C. Walsworth, E. Aben, k.c. claffy, and D. Andersen. The CAIDA anonymized 2009 and 2012 Internet traces. `http://www.caida.org/data/passive/passive_2009_dataset.xml`; `http://www.caida.org/data/passive/passive_2012_dataset.xml`, [15 March 2014].

14. F. Mata, J. L. García-Dorado, and J. Aracil. Detection of traffic changes in large-scale backbone networks: The case of the Spanish academic network. *Computer Networks*, 56(2):686 – 702, 2012, doi:10.1016/j.comnet.2011.10.017.

15. N. Bonelli, A. Di Pietro, S. Giordano, and G. Procissi. On multi-gigabit packet capturing with multi-core commodity hardware. In *Proceedings of Conference on Passive and Active Network Measurement*, 2012, doi:10.1007/978-3-642-28537-0_7.

16. Intel. Intel Data Plane Development Kit (Intel DPDK), 2014. `http://www.intel.com/content/dam/www/public/us/en/documents/release-notes/intel-dpdk-release-notes.pdf`, [15 March 2014].

17. A. Finamore, M. Mellia, M. Meo, M.M. Munafò, and D. Rossi. Experiences of Internet traffic monitoring with Tstat. *IEEE Network*, 25(3):8 –14, 2011, doi:10.1109/MNET.2011.5772055.

18. G. Maier, R. Sommer, H. Dreger, A. Feldmann, V. Paxson, and F. Schneider. Enriching network security analysis with time travel. In *ACM SIGCOMM*, pages 183–194, 2008, doi:10.1145/1402958.1402980.

19. P. Čeleda, R. Krejčí, J. Barienčík, M. Elich, and V. Krmíček. HAMOC—hardware-accelerated monitoring center. Technical Report 9/2010, CESNET, 2010. `http://http://archiv.cesnet.cz/doc/techzpravy/2010/hamoc/`, [15 March 2014].

20. A. di Pietro, F. Huici, N. Bonelli, B. Trammell, P. Kastovsky, T. Groleat, S. Vaton, and M. Dusi. Toward composable network traffic measurement. In *Procedings of IEEE INFOCOM*, 2013, doi:10.1109/INFCOM.2013.6566737.

21. F. Gringoli, A. Este, and L. Salgarelli. MTCLASS: Traffic classification on high-speed links with commodity hardware. In *Proceedings of IEEE Conference on Communications*, 2012, doi:10.1109/ICC.2012.6363806.

22. P. M. Santiago del Rio, D. Rossi, F. Gringoli, L. Nava, L. Salgarelli, and J. Aracil. Wire-speed statistical classification of network traffic on commodity hardware. In *Proceedings of ACM Internet Measurement Conference*, 2012, doi:10.1145/2398776.2398784.

23. M. Jamshed, J. Lee, S. Moon, I. Yun, D. Kim, S. Lee, Y. Yi, and K.S. Park. Kargus: a highly-scalable software-based intrusion detection system. In *Proceedings of ACM Conference on Computer and Communications Security*, 2012, doi:10.1145/2382196.2382232.

24. G. Vasiliadis, M. Polychronakis, and S. Ioannidis. MIDeA: a multi-parallel intrusion detection architecture. In *Proceedings of ACM Conference on Computer and Communications Security*, 2011, doi:10.1145/2046707.2046741.