Wire-Speed Statistical Classification of Network Traffic on Commodity Hardware

Pedro M. Santiago del Rio
UAM, Madrid, Spain
pedro.santiago@uam.es

Dario Rossi
Telecom ParisTech, France
dario.rossi@enst.fr

Francesco Gringoli
UNIBS, Brescia, Italy
francesco.gringoli@ing.unibs.it

Lorenzo Nava
UNIBS, Brescia, Italy
lorenzo.nava@ing.unibs.it

Luca Salgarelli
UNIBS, Brescia, Italy
luca.salgarelli@ing.unibs.it

Javier Aracil
UAM, Madrid, Spain
javier.aracil@uam.es

ABSTRACT

In this paper we present a software-based traffic classification engine running on commodity multi-core hardware, able to process in real-time aggregates of up to 14.2 Mpps over a single 10 Gbps interface – i.e., the maximum possible packet rate over a 10 Gbps Ethernet links given the minimum frame size of 64 Bytes.

This significant advance with respect to the current state of the art in terms of achieved classification rates are made possible by: (i) the use of an improved network driver, PacketShader, to efficiently move batches of packets from the NIC to the main CPU; (ii) the use of lightweight statistical classification techniques exploiting the size of the first few packets of every observed flow; (iii) a careful tuning of critical parameters of the hardware environment and the software application itself.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network Monitoring

Keywords

Statistical Identification, Commodity Hardware, Traffic Monitoring

1. INTRODUCTION AND MOTIVATION

The ability to identify which application is generating every single traffic session is recognized as a crucial building block of today IP networks and unavoidable requisite for their evolution [6]. Effective techniques could open new possibilities for actual deployment of QoS, for enforcing user traffic to comply with policies, for legal interception and intrusion detection [14].

Classic techniques based on Deep Packet Inspection (DPI) have been thoroughly analyzed during the years: though specialized hardware based on Network Processor [17] and FPGAs [18] have been considered, the emergence of multi-core commodity hardware has gained increasing attention and exhaustive performance analyses have been reported also for advanced systems using off-the-shelf Graphics Processing Units (GPUs) [28, 27]. Unfortunately, despite the powerfulness of the underlying hardware none of aforementioned approaches is able to actually sustain a 10 Gbps throughput.

The latest years have also seen a flurry of proposals exploiting different "features" (in machine learning terms) to perform the classification [20]. Statistical techniques based on the size and directions of the first few packets of a flow [1, 5] emerged as especially appealing due to their low complexity if compared to current state of the art DPI approaches. Furthermore, such techniques can be used when traffic is encrypted, while DPI approaches simply cannot. However, most of the previous work on statistical classification focused on assessing the *accuracy* of the different techniques (that we take for granted given results in [1, 5, 14, 16]) without measuring their *achievable classification rates*.

In this work, we argue that commodity multi-core hardware offers intrinsic scalability at low cost, while providing the unbeatable flexibility of software-only solutions. Hence, we take a different twist with respect to works employing specialized hardware based on Network Processor or FPGAs: furthermore, our software based solution is the first to achieve two important milestones. First, using both real Tier-1 traces and synthetic traffic, we demonstrate that multi-Gbps statistical traffic classification is feasible with open-source software on off-the-shelf hardware. Second, our solution is able to sustain higher classification rates than previous work [27, 17, 28, 16] with a sizeable gain in terms of the maximum amount of classification actions per second and manageable packet rates.

In more detail our software can easily handle a real Tier-1 traffic aggregate (i.e., a CAIDA OC192 trace [29]) replayed at 10 Gbps, corresponding to 1.6 million packets per second (Mpps) and 58 thousand flow classifications per second (Kfps). Using two interfaces, our system sustains classification rates of 20 Gbps, 3.2 Mpps, 116 Kfps. Yet, the upper bound of the system performance is much higher, as we manage to handle classification rates up to 14.2 Mpps and 2.8 Mfps without any losses (benchmark with synthetic

IMC'12, November 14–16, 2012, Boston, Massachusetts, USA.

worst-case traffic scenario with trains of 64B packets, 5 packets per flow, over a 10 Gbps link).

Such astonishing performance follows the use of a lightweight classification algorithm (which bases its decisions upon the size of the first 4 packets of every unidirectional flow [1, 5]), as well as of the most recent advances in terms of packet processing techniques [11]. Yet, as we will see in the following, engineering the system so that it sustains wire-speed classification in the worst-case traffic scenario required investigation of several delicate architectural trade-offs of the software, as well as the careful tuning of hardware parameters. We believe the removal of all software bottlenecks in the workflow to be another major contribution of this paper.

The paper sheds light on traffic monitoring and measurement, giving advices and guidelines for designing high performance traffic analysis tools which may be helpful for researchers and network program designers. In this light, we have released the code of our implementation to the community as open-software[1] which may enlighten researchers and designers about how to implement their own network monitoring tools (or modify the existing ones) to cope with current high speeds, i.e. 10Gbps and beyond.

Finally, we point out that to stress-test our system, we also had to develop highly efficient traffic injection engines able to either generate infinite synthetic traffic or replay huge traces thus saturating 10 Gbps links during very long experiments. As final contribution of this paper, we make these latter tools, whose scope of applicability goes beyond that of the Internet traffic classification, available as open-source software[2] as a research byproduct.

2. RELATED WORK

We present recent advances in (i) packet capturing engines, (ii) flow management and (iii) traffic classification techniques. We summarize performance for each category in Tab. 1, further providing a comparison of our results with respect to the current state of the art.

Packet capturing engines. Although modern Network Interface Cards (NICs) hardware can handle high packet rates, standard operating system network stacks are still affected by a few software bottlenecks (e.g., per-packet operations like buffer allocation and transfer to user-space). To overcome such issues, several approaches bypass standard stacks by (i) processing multiple packets in batch to limit IRQs and DMA transactions; (ii) exposing memory of packet buffers to the user-space for zero-copy access; (iii) tying every capture thread with its own ring buffer to a fixed CPU to increase cache memory hits (Non-Uniform Memory Access, NUMA) and (iv) using Receive Side Scaling (RSS) to split incoming flows among different input queues/capture threads. We report in Tab. 1 packet capture performance for a number of systems including: PF_RING with Threaded NAPI [10] and variants [4]; Netmap [24, 23]; PacketShader [11]; and PFQ [2]. As can be seen from the table, performance at 10 Gbps are impressive and this demonstrates that off-the-shelf hardware can now be used in place of costly Network Processors. Finally, since these systems achieve comparable performance, *the choice of a specific technique is not critical for our purposes, and falls on PacketShader [11]* (see Sec. 3).

[1] `http://www.eps.uam.es/~psantiago/hptrac.html`
[2] `http://www.eps.uam.es/~psantiago/hpcap.html`

Flow matching. Monitoring at the flow level requires to match each packet to the correct flow bin. In software based solutions such as Tstat [25] or YAF [12] this is usually accomplished using hash-based structures over the flow 5-tuple. To the best of our knowledge, however, performance of flow matching code in complex monitoring systems is rarely evaluated alone and extrapolating such data from overall measurements can be tough or even misleading. For instance, [12] describes a flow management module in detail, explaining how to optimize flow management using slab allocator [3] for fast recycling of expired flow records, but benchmarks of the system performance are not publicly available. Otherwise, the performance analysis for flow matching modules has been done either monitoring real ISP deployments [9] or over offline traces [25, 30]. However, as real 10 Gbps traffic is not by itself a stress-test scenario, this calls for synthetic benchmarks.

Explicit performance are reported instead in [8] where a dual Xeon box hosts a dedicate Endace DAG card which achieves matching of up to 6 Mpps. In [22] an Intel IXP2850 Network Processor is shown matching 10 million concurrent flows at 10 Gbps at full packet rate. Switching to off-the-shelf setup, an application note from Intel [15] reports flow matching of trains of 64 bytes packets at 17 Mpps out of 24 Mpps received over 16× 1 Gbps interfaces, where each NIC is tied to a different core of an Intel multi-core CPU system (unfortunately the study does not report the number of concurrent flows). A similar architecture [7] matches up to 11 Mpps for 1 million concurrent flows at 10 Gbps using "FastFlow" algorithms spawned over 6 cores. *For comparison, our system is able to handle aggregate flow rates up tp 2.8 Mfps using just two cores.*

Statistical traffic classification. Several techniques have been proposed for the classification of Internet traffic. Traditional ones are based either on the analysis of the transport layer port numbers as in CoralReef [19], or on Deep Packet Inspection of the packet payload as most commercial tools do. Statistical techniques, instead, observe basic properties like packet lengths and interarrival times to classify traffic and are celebrated for their accuracy and speed [20]. However, while the former has been experimentally demonstrated [14, 16] on real traces, the latter is far from being assessed. As a result, the classification rates and scalability of these new algorithms are either unknown or really far from those needed for real world deployment [6]: e.g., [1, 5] merely discuss the complexity of the classification technique, while the most recent performance analysis in [16] reports as few as $30 \cdot 10^3$ classification per seconds with Naïve Bayes. *On a off-the-shelf setup similar to that used in [16], we achieve* $2.8 \cdot 10^6$ *classification per seconds running custom implementation of Naïve Bayes, thus boosting performance of a factor of 100.*

Furthermore we can compare our results to that achieved by non-commercial Deep Packet Inspection (DPI) systems [17, 28, 27], that run pattern matching algorithms on multicore GPUs. It is worth noting that while the amount of GPUs power is already enough to process up to 40 Gbps traffic, bottlenecks in the communication subsystem crushes the actual performance down to a mere 5.2 Gbps corresponding to 1 Mpps and 5 Kfps [28]. Similarly, [17] and [27] achieve 3.5 Gbps and 6 Gbps of aggregated traffic rate, corresponding to less than 2 Mpps. *Our technique not only sustains*

Table 1: Maximum processing rate in the state of the art, and advances offered by this work.

Category	*Ref.*	*Rates* MFlow/s	MPkts/s	Gb/s	*Comments*
Packet capture	[10, 4]	-	14.8	10	60B frames
	[24, 23]	-	14.2	10	64B frames
	[11]	-	14.2	10	64B frames
	[2]	-	12	10	Sender limitation
	this work[11]	-	*14.2*	*10*	*64B frames*
Flow handling	[8]	-	6	10	Using Endace DAG cards
	[15]	-	17	-	Using 16 cores (16x1Gbps interfaces)
	[7]	1	10	10	Using 6 cores
	this work	*2.8*	*14.2*	*10*	*Using 2 cores*
Traffic classification	[27]	-	1.8	6.7	Using a GPU, synthetic traffic
	[17]	-	-	3.5	Using real traffic
	[28]	0.005	1	5.2	Using a GPU, real traffic
	[16]	0.03	-	-	Offline experiments on real traffic (Weka)
	this work	*0.116*	*3.2*	*20*	*Real Tier-1 OC192 traffic over 2×10Gbps NIC*
		2.8	*14.2*	*10*	*Synthetic traffic (64B frames, 5pkts/flow)*

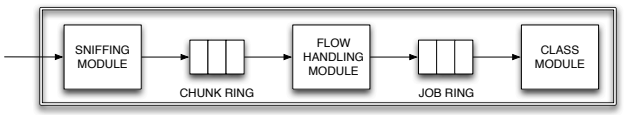

Figure 1: System modules.

10 Gbps of aggregated traffic, but potentially much more as we handle $2.8 \cdot 10^6$ *classifications per second.*

3. SYSTEM MODULES

We report in Fig. 1 the three main blocks that compose our classification system, the two data ring structures for packet/flow queuing and the logical connections that push information from left to right.

Sniffing module. We capture incoming packets making use of PacketShader [11], a customized version of the Intel ixgbe driver that can fetch chunks of multiple frames from the NIC using one single DMA data transfer, greatly reducing the I/O overhead and the per-packet buffer allocation cost. Thanks to a native feature of the Intel 82599EB 10 Gbps Ethernet controller[13], incoming frames are partitioned in RSS queues according to a hash function: one sniffing module (a thread running in user space that has direct access to the kernel-space buffers) can then be set up to fetch frames only from a given RSS queue. Following Intel paradigm, we also tie every capture thread to a specific CPU core (thread affinity) so as to keep the data locally in that CPU cache (hence limiting cache thrashing between processor sockets). This same feature extends parallelism from the NIC to the user layer, as RSS queues feed different cores with multiple chunks at the same time, pushing them through multiple lanes of the PCIe bus and hence increasing the overall throughput with respect to a single core solution. Packets are then organized in a circular ring and made available to the user space with zero-copy technology. Here a thread running on the same CPU copies the chunks from the kernel ring and enqueues resulting data to a *Chunk Ring* of Fig. 1: if the ring is full, a chunk might be lost. We set the chunk size to 128 packets.

Flow Handling module. A thread then dequeues packets from the Chunk Ring and perform lookup into a *Flow Table*. A hash over the packet 5-tuple is used as a primary key to access a hash table, while collisions are handled by chaining (a data structure based on linked list of flow buckets). Once the bucket is found (or a new one is appended if the flow was not already known), a new feature is added to the flow structure, namely the length of the corresponding packet (read from the IP header). Each flow is considered active within a timeout (default 15 sec) after the reception of the last packet. When the timeout expires, its position in the linked list can be reused by a new flow (no deallocation overhead). Once a configurable number of packets for a given flow has been seen (4 in this work), a new classification Job is fired to the *Job Ring* of Fig. 1 if a position is available (otherwise, the Job will likely be inserted the next time a packet from that flow will be analyzed). We set the flow hash table size[3] to 50 millions.

Classification Module. Classification threads run a custom implementation of Naïve Bayes with Gaussian density estimation. Given the first four packets of a flow have been received, the algorithm associates the flow to the protocol whose model scores the maximum likelihood for the generation of the flow. For each protocol model the algorithm uses the size of the packets as indexes into the four lookup tables that have been associated to that protocol during the training phase: the values extracted are then summed together, we optimize, in fact, the algorithm by storing the logarithms of the table values to avoid products as reported in [26]. By comparing the values obtained for each of the protocols for which a model is available, the algorithm chooses the application and the classification of the flow terminates: for more details please refer to [5]. Although classification accuracy, in terms of packets *correctly* classified, is a key issue, it is not the goal of this paper because it has been already

[3]For reason of space, we are unable to report a detailed sensitivity analysis of the hash table size; here, we merely stress that we set a hash size large enough to significantly mitigate the occurrence of chaining.

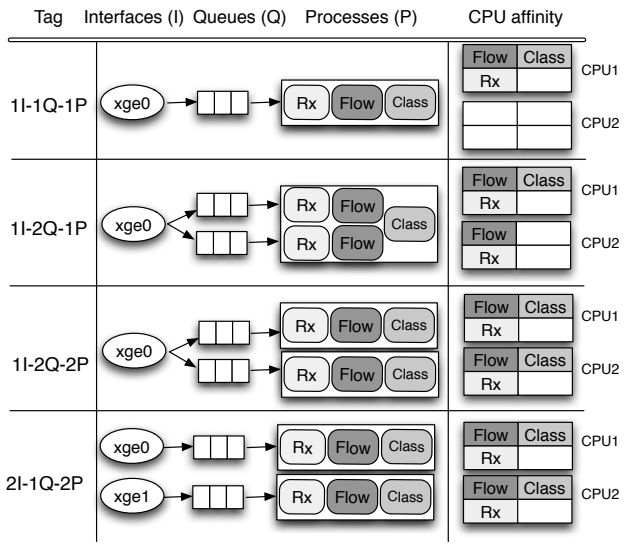

Figure 2: Different architectural configurations of our system.

analyzed. Thus, we have chosen a representative classification technique, such as Naïve-Bayes, whose accuracy has been previously showed as enough for traffic classification purposes [20]. Once chosen the classification technique, we evaluated its performance in terms of computational cost and the feasibility of its implementation on a real system (based on commodity hardware and open software). Note that the computational complexity, given a model, is not a function of accuracy.

To increase the speed at which Jobs are extracted from the Job Ring, multiple threads can be spawned according also to the complexity of the algorithm. Each thread extracts one Job from the ring, processes the data, and writes the classification verdict in the flow bucket inside the Flow Table. New packet for that flow will be marked with this verdict in their Type Of Service (TOS) field of the IPv4 header[4]. By default, we set only one thread to classify: as we will see in the next section, this is sufficient to classify all Jobs generated by flow handling module.

4. SYSTEM CONFIGURATION

Contrary to traditional network applications, new capture engines extend the capture data rates by several orders of magnitude. This implies that probably current networking software is not able to work adequately at this speed. In addition to this, multi-core hardware opens a new opportunity to develop applications that take advantage of the parallelism that such engines allow, whereas current applications were developed in the pre-multicore area. Thus, careful engineering of the system (e.g., CPU and memory affinity, threads/process choice) is needed to achieve the highest classification rates of Tab. 1 under worst-case traffic. Configurations explored in this paper are sketched in Fig. 2.

1I-1Q-1P. This is the simplest configuration: traffic received at the same RSS queue of a single interface is captured by a thread which in turn pushes packets to one chunk ring in the user space. All packets are matched in the flow table sequentially: one classification thread works on the single Job ring. *In this case, though, some CPU cores are not utilized.* Note that using a single RSS queue may be preferable to multi-queue in some cases —whenever the performance is enough to cope with line-rate. For instance: to get smaller CPU usage (and therefore less power consumption), not to re-implement monitoring tools which have not been designed for parallel processing, or to avoid packet reordering issues due to multi-queue[31].

1I-2Q-1P. This configuration holds the single process model of the previous one but two RSS queues are used: this means two threads for fetching packets chunks and two separate flow matching modules. Each capturing flow is executed on a different CPU, but only one hash table is used. Since a single Job ring is used, data coming from two different CPUs is merged again in a single data flow. *In this case, locking is used to enable concurrent accesses to the flow table, and this might decrease the overall performance.*

1I-2Q-2P. Though similar to the previous configuration (two RSS queues, two threads for capturing, bound to different CPUs), the two threads for flow-handling *reside in different processes*, each on the same CPU of the corresponding sniffer. This means that two separated flow tables are maintained and no locking is required thanks to the hash function used at the NIC for dividing packets into the two queues: each single flow lives on a single queue only (no mixing). Moreover, the classification code (threads) and data structures (Job rings) are duplicated and fairly spread among the two CPUs with no data flow merge. *In this case, locking is solved at the price of doubling the amount of memory.*

2I-1Q-2P. The last configuration is exactly as the first one, but two NICs are used: for each NIC a complete capture and classification chain is instantiated, each complete chain lives on a separated CPU. *Given the number of cores in our system, we cannot explore other configurations when 2 interfaces are in use.*

5. PERFORMANCE EVALUATION

First, we describe our experimental testbed, covering hardware, software and traffic details. Then, we benchmark system performance in two scenarios: (i) we locate and solve system bottlenecks using synthetic traffic in a worst-case scenario (64B packets at maximum rate); (ii) we assess our system in a real scenario, replaying traces from a Tier-1 link over one or two 10 Gbps interfaces.

5.1 Experimental testbed

Hardware setup. Our setup consists of two general-purpose servers: one acts as traffic generator, the other receives and classifies the traffic. Both are equipped with 24 GB of DDR3 memory and feature two Intel Xeon E5620 processors, counting four cores each (with hyper-threading capabilities disabled to obtain actual parallelism among cores) working at 2.40 GHz. Concerning connectivity, each server is equipped with one dual port 10Gbps Intel X520-SR2 NIC, and servers are directly connected with a fiber link. This NIC model is based on 82599 chipset, that enables multi-queue techniques, up to 16 RSS queues per interface and direction.

[4]For reason of space, we do not assess packet forwarding after classification in this paper.

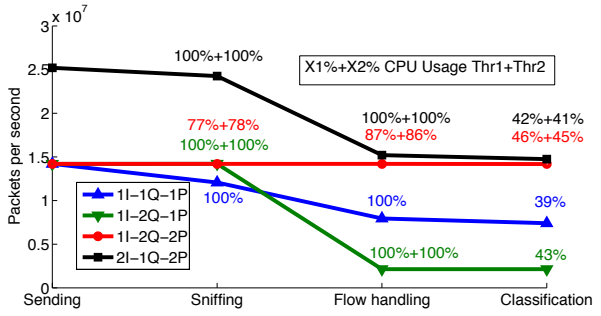

Figure 3: System Performance. Worst-case scenario: synthetic traffic 64B packets, 5 packet/flow.

Software. Ubuntu 10.04 server 64-bit is installed on both servers with a 2.6.35 Linux kernel. In order to inject traffic, we have developed a tool on top of PacketShader API to send traffic at maximum rate: generic tools such as tcpreplay[5] can not, in fact, saturate 10Gbps links. Our tool[6] instead is able either to inject infinite synthetic traffic or to replay very long packet-level traces at maximum speed.

Traffic. We utilized both synthetic traffic and real traces. *Synthetic traffic* consists of TCP segments encapsulated into 64B-size Ethernet frames, forged with incremental IP addresses and TCP ports. We stress once more that, though all packets have 64B-size, the packet length features are extracted from the IPv4 header: hence, our benchmark methodology does not affect the relevance of the classification results. For each flow (5-tuple combination), we send 5 packets, since the 5-th packet will be the first to have the chance to be classified (on the basis of the packet-length features of the previous 4 packets). At a maximum rate of 14.2 Mpps for 64B frames, this translate into 2.8 Mfps. *Real traffic* consists of a packet-level trace sniffed in 2009 at an OC192 (9953 Mbps) backbone link of a Tier-1 ISP located between San Jose and Los Angeles, available from CAIDA [29]. All the packets in the trace are anonymized and captured without payload, the average of the original packet size in the trace is 744 bytes and the average number of packets per flow is 49.

5.2 Stress test: finding the bottlenecks

We first stress test the system in a worst-case scenario, i.e., using synthetically generated 5-packets long flows, sending 64B frames at maximum rate, i.e., 14.2 Mpps or 2.8 Mfps per interface. We measure the amount of packets processed by each module during 60-second experiments (but we also tested the system on 24-hr long experiments on the best configuration without observing losses). Note that using two interfaces we were only able to send at ≈25 Mpps (instead of the theoretic maximum 28.4 Mpps) due to limitations in the sender.

Fig. 3 shows the performance of each module (sniffing, flow handling and classification) for the different configurations. In the parallel coordinates plot, a negative slope in the curves reflects a bottleneck: i.e., a module is not able to process all packets generated by the previous one. Curves in Fig. 3 are annotated with the CPU usage of each module: if a module runs two threads, CPU usage is expressed summing the two corresponding values.CPU usage is computed using `sysstat` utilities[7]. We obtained the CPU load per thread every 5 seconds and then we averaged. Background CPU usage (when there is no classification systems running) and experimental variance are negligible.

The simplest configuration. The simplest configuration 1I-1Q-1P sniffs traffic from one interface, uses one RSS queue and one process. In this case, it can be observed that not all packets can be sniffed using a single RSS queue (a single core). Particularly, only 12.1 Mpps are sniffed out of the 14.2 Mpps sent: notice that the CPU usage of the sniffing module is 100% which pinpoints a processing bottleneck. Similarly, flow module is only able to process 7.9 Mpps out of 12.1 Mpps sniffed packets. As CPU utilization of the flow handling module is also 100%, we have strong indication of a second processing bottleneck. Finally, a slight negative slope can be observed in the classification module. This is not due to a CPU bottleneck (39%), but rather to the fact that only flows with 5 packets can be classified and there have been packet losses in previous modules. The classification rates sustained by 1I-1Q-1P configuration are thus 7.9 Mpps and 1.5 Mfps.

Using 2 RSS queues. In order to remove bottlenecks observed in the first configuration, we increment the number of RSS queues and the number of cores dedicated to packet sniffing and flow management. Thus, we test two configurations, namely 1I-2Q-1P and 1I-2Q-2P. As shown in Fig. 2 and explained in Sec. 4, the former configuration uses one process with two threads for sniffing, one per queue, and two threads for flow handling, having a unique hash table and a unique classification thread. Conversely, 1I-2Q-2P configuration uses two processes, one per queue, which do not share neither data structures nor processing cores. We can observe that the bottleneck in the sniffing module is removed in both cases: i.e., at 14.2 Mpps, two RSS queues are enough to receive and process the traffic. Besides, the 1I-2Q-2P configuration with two processes consumes less CPU power.

Locking issue. The behavior of the flow handling module is different for 1I-2Q-1P and 1I-2Q-2P. Indeed, 1I-2Q-1P is not able to process all packets received by the sniffing module, and performance are even worse than in the case with only one RSS queue. The bottleneck in this configuration is tied to the contention in the access to shared data structures, such as the hash table and the Job ring. To arbitrate concurrent access to shared memory, synchronization and locking mechanisms are necessary but they are detrimental to overall performance. As in the first configuration, all flows generated by the flow handling module can be classified. With 1I-2Q-1P configuration, the performance of the whole system fall to 2.1 Mpps and 0.4 Mfps.

Wire-speed classification. Using two RSS queues and two independent processes, we remove both sniffing bottleneck and flow management locking issues. With 1I-2Q-2P, the system sniffs, processes and classifies all packets sent at wire-speed without losses. Notice further that not even a single CPU core is saturated (sniffing 77%+78%, flow management 87%+86%, classification 45%+46%), so that the remaining processing power could be useful to perform other

[5] `http://tcpreplay.synfin.net/`
[6] `http://www.eps.uam.es/~psantiago/hpcap.html`
[7] `http://sebastien.godard.pagesperso-orange.fr/documentation.html`

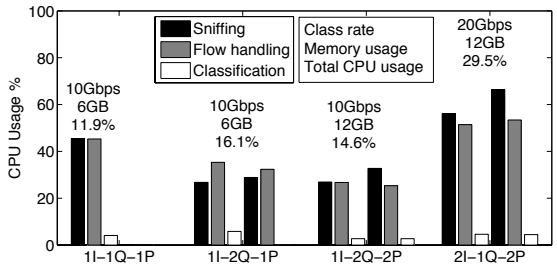

Figure 4: Real scenario: CAIDA trace with original packet length.

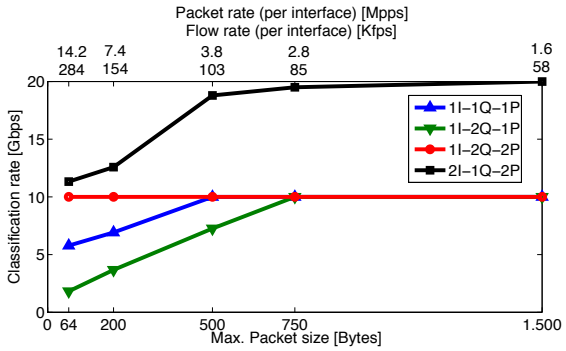

Figure 5: CAIDA trace with capped packet length.

tasks (such as packet forwarding or statistics collection). This also means that the processing capabilities of 1I-2Q-2P configuration exceed the maximum data rate at 10 Gbps., i.e., 14.2 Mpps and 2.8 Mfps.

Using 2 interfaces. The latest configuration, 2I-1Q-2P uses two interfaces to receive traffic, but a single RSS queue per interface due to the limit in the number of cores. As expected, behavior is similar to 1I-1Q-1P, with bottlenecks in both sniffing (24.2 Mpps received out of 25.2 Mpps sent) and flow handling (15.2 Mpps processed out of 24.2 Mpps received).

5.3 Real scenario: classifying at 20 Gbps

Original packet size. We now test performance on real Tier-1 traffic. We replay traces by sending packets back-to-back at 10 Gbps by filling payload with zeros. Using full packet size in Fig. 4, all system configurations sustain maximum rate: i.e., 1.6 Mpps and 58 Kfps on a single 10Gbps interface, or 3.2 Mpps and 116 Kfps on two interfaces. CPU usage and memory occupancy report that cores are far from being saturated: this proves that our system could classify more than 20 Gbps traffic in a realistic scenario. Notice that the simplest configuration (only 3 threads) is enough to classify all traffic with the smallest CPU load (hence the lowest carbon footprint).

Capped packet size. Finally, in Fig. 5 for each frame of size S_i we control the maximum amount of bytes sent on the wire as $\max(S_i, L)$ with L the maximum frame size, that we vary in the range $[64, 1500]$B to tune the flow and packets arrival rates for a fixed datarate of 10 Gbps – hence finding the packet and flow processing rate bottleneck of each configuration. From the figure, we gather that the simplest configuration 1I-1Q-1P can sustain up to 3.8 Mpps and 103 Kfps using only 3 cores.

6. CONCLUSIONS

We propose an all software solution for statistical traffic classification on commodity hardware. Our system achieves a significant advance with respect to the state of the art in several ways. First, it demonstrates the feasibility of online statistical traffic classification, that was so far confined on offline analysis published in the literature. Second, it significantly outperforms state of the art classification techniques. Indeed, while the raw classification throughput on real traffic aggregates is about 3× higher than [27] and 4× higher than [28], however our system is able to sustain flow classification rates 93× higher than [16] and 560× higher than [28]. Additionally, this paper sheds light on traffic monitoring and measurement, giving advices and guidelines for designing high performance traffic analysis tools which may be helpful for researchers and designers. In this light, we released the code of our implementation to the community as open-software which may be useful for researchers and designers in order to implement their own network monitoring tools (or modify the existing ones) to cope with current high speeds (10Gbps and beyond).

Notice that the above performance gap between our work and the previous ones, is hard to remove. Indeed, on the one hand, while GPUs in [28] could in principle process 40 Gbps equivalent of traffic, this is forbidden by a bottleneck in the path from the NIC to the GPU. That is, current approaches force to pass through main memory, and waste processing time, to transfer data between the NIC and the GPU creating a bottleneck — although there are preliminary results which may avoid such limitation, they can be only used with Infiniband technology yet[21]. On the other hand, the statistical technique is anyway much more lightweight than DPI (only process packet headers), so that it would benefit more from a GPU. This gap is intrinsic to the nature of the statistical classification process, that avoid to transfer packet payload from the NIC to GPUs unlike DPI. Thus, statistical approaches are unachievable for DPI, even making use of different hardware, such as GPUs.

While this work constitutes a significant advance, we believe that further optimization are possible, which are part of our current ongoing work. First, we aim at wire-speed performance on two interfaces, which should be achieved with a 2I-2Q-4P configuration on a 12 core server. Second, we plan to implement C4.5 trees, due (i) their known discriminative power, and (ii) the fact that they can be efficiently implemented as if-then-else branches. Finally, we want to optimize flow management, avoiding to compute hash in software, by exporting the hash computed by the NIC to map packets to RSS queues, which should requires only simple modification to the NIC driver.

Acknowledgements

This work has been carried out when Pedro was at LINCS[8], with the support of the Spanish FPU scholarship, and has been partly supported by the FP7 IP mPlane project. This work was supported in part by a grant from the Italian MIUR, under the PRIN project IMPRESA.

[8] `http://www.lincs.fr`

7. REFERENCES

[1] L. Bernaille, R. Teixeira, and K. Salamatian. Early application identification. In *ACM CoNEXT 2006*.

[2] N. Bonelli, A. Di Pietro, S. Giordano, and G. Procissi. On multi-gigabit packet capturing with multi-core commodity hardware. In *Passive and Active Measurement (PAM) 2012*.

[3] J. Bonwick. The slab allocator: An object-caching kernel memory allocator. In *USENIX Summer Technical Conference 1994*.

[4] A. Cardigliano, J. Gasparakis, and F. Fusco. vPF_RING: Towards wire-speed network monitoring using virtual machines. In *ACM IMC 2011*.

[5] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Comput. Commun. Rev.*, 37(1):5–16, 2007.

[6] A. Dainotti, A. Pescape, and K. Claffy. Issues and future directions in traffic classification. *Network, IEEE*, 26(1):35 –40, 2012.

[7] M. Danelutto, L. Deri, and D. De Sensi. Network monitoring on multicores with algorithmic skeletons. In *International Conference on Parallel Computing (PARCO) 2011*.

[8] L. Deri. IP traffic monitoring at 10 Gbit and above. `http://www.terena.org/activities/ngn-ws/ws2/deri-10g.pdf`.

[9] A. Finamore, M. Mellia, M. Meo, M. Munafo, and D. Rossi. Experiences of Internet traffic monitoring with Tstat. *Network, IEEE*, 25(3):8–14, 2011.

[10] F. Fusco and L. Deri. High speed network traffic analysis with commodity multi-core systems. In *ACM IMC 2010*.

[11] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: a GPU-accelerated software router. In *ACM SIGCOMM Comput. Commun. Rev.*, volume 40, pages 195–206, 2010.

[12] C. Inacio and B. Trammell. YAF: yet another flowmeter. In *International conference on Large installation system administration (LISA) 2010*.

[13] Intel. Intel Í 82599 10 GbE Controller Datasheet. *October*, (December), 2010.

[14] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee. Internet traffic classification demystified: myths, caveats, and the best practices. In *ACM CoNEXT 2008*.

[15] A. Lim and R. Kinsella. Data plane packet processing on embedded intel architecture platforms. `http://download.intel.com/design/intarch/papers/322516.pdf`.

[16] Y. Lim, H. Kim, J. Jeong, C. Kim, T. Kwon, and Y. Choi. Internet traffic classification demystified: on the sources of the discriminative power. In *ACM CoNEXT 2010*.

[17] Y. Liu, D. Xu, L. Sun, and D. Liu. Accurate traffic classification with multi-threaded processors. In *IEEE International Symposium on Knowledge Acquisition and Modeling Workshop (KAM) 2008*.

[18] A. Mitra, W. Najjar, and L. Bhuyan. Compiling PCRE to FPGA for accelerating SNORT IDS. In *ACM/IEEE Symposium on Architecture for networking and communications systems (ANCS) 2007*.

[19] D. Moore, K. Keys, R. Koga, E. Lagache, and K. C. Claffy. The CoralReef software suite as a tool for system and network administrators. In *USENIX conference on System administration 2001*.

[20] T. Nguyen and G. Armitage. A survey of techniques for Internet traffic classification using machine learning. *Communications Surveys & Tutorials, IEEE*, 10(4):56–76, 2008.

[21] NVIDIA Corporation. NVIDIA GPUDirect Technology. `http://developer.download.nvidia.com/devzone//devcenter/cuda/docs/GPUDirect_Technology_Overview.pdf`.

[22] Y. Qi, B. Xu, F. He, B. Yang, J. Yu, and J. Li. Towards high-performance flow-level packet processing on multi-core network processors. In *ACM/IEEE Symposium on Architecture for networking and communications systems (ANCS) 2007*.

[23] L. Rizzo. netmap: a novel framework for fast packet I/O. In *USENIX Annual Technical Conference 2012*.

[24] L. Rizzo, M. Carbone, and G. Catalli. Transparent acceleration of software packet forwarding using netmap. In *IEEE INFOCOM 2012*.

[25] D. Rossi and M. Mellia. Real-time TCP/IP analysis with common hardware. In *IEEE ICC 2006*.

[26] D. Rossi, S. Valenti, P. Veglia, D. Bonfiglio, M. Mellia, and M. Meo. Pictures from the Skype. *ACM Performance Evaluation Review (PER)*, 36(2):83–86, 2008.

[27] G. Szabó, I. Gódor, A. Veres, S. Malomsoky, and S. Molnár. Traffic classification over Gbit speed with commodity hardware. *IEEE J. Communications Software and Systems*, 5, 2010.

[28] G. Vasiliadis, M. Polychronakis, and S. Ioannidis. MIDeA: a multi-parallel intrusion detection architecture. In *ACM conference on Computer and communications security (CSS) 2011*.

[29] C. Walsworth, E. Aben, k. claffy, and D. Andersen. The CAIDA anonymized 2009 Internet traces. `http://www.caida.org/data/passive/passive_2009_dataset.xml`.

[30] D. Wang, Y. Xue, and Y. D. Memory-efficient hypercube flow table for packet processing on multi-cores. In *IEEE GLOBECOM 2011*.

[31] W. Wu, P. DeMar, and M. Crawford. Why can some advanced Ethernet NICs cause packet reordering? *IEEE Communications Letters*, 15(2):253–255, 2011.