Enabling Open-Source High Speed Network Monitoring on NetFPGA

Gianni Antichi, Stefano Giordano
Dept. of Information Engineering, University of Pisa, ITALY
Email: {gianni.antichi,stefano.giordano}@iet.unipi.it

David J. Miller, Andrew W. Moore
Computer Laboratory, University of Cambridge, UK
Email: {david.miller,andrew.moore}@cl.cam.ac.uk

***Abstract*—Network measurement both as diagnostic and within measurement-based techniques of traffic engineering and management, alongside network measurement for security has maintained the needs of researchers and network operators for the ongoing development of measurement tools for traffic monitoring/characterisation and to support Intrusion Detection Systems (IDSs). Many such tools capitalise on the pricing of commodity hardware by operating on general purpose architectures. Many are based on the well known `libpcap` API, a de facto standard in this area. Despite the many improvements that have been applied to packet capturing, packet-monitoring implementations still suffer from either: performance flaws on commodity hardware due mainly to unresolvable hardware bottlenecks, or costly and inflexible niche systems. To address such issues, the paper proposes a system architecture based on the cooperation of NetFPGA and a general purpose host PC. The NetFPGA is an open networking platform accelerator that enables rapid development of hardware-accelerated packet processing applications. The objective is to combine the high performance of a hardware-oriented solution with the flexibility of general purpose PCs.**

I. Introduction

Passive network measurement remains the best way to observe packets on a network without disturbing the nature or timing of the existing data. The diversity of flexible, easy-to-use and easy-to-customise network monitoring software suggests the PC as an ideal, cheap platform for network measurement and testing. Indeed, applications such as `tcpdump` [4], `wireshark` [10], and `ntop` [8] prove very effective tools for a large variety of monitoring tasks. As network link speeds increase, it is arguable that network measurement techniques which depend upon software timestamping and capture will not scale to the required standard of performance.

In particular, to sustain a high-packet rate, the PC must drive interface cards by using a polling scheme or interrupt driven I/O with interrupt mitigation enabled. Without hardware-assisted timestamping, both techniques result in poor timestamp accuracy.

In addition, packet loss is inevitable at high-speed if the host cannot allocate or release memory quickly enough, or if DMA bandwidth is insufficient for the volume and pattern of captured traffic. Moreover, CPU time used for timestamping captured packets cannot be used for on-line analysis [13] [12] with the result that such analysis must be done off-line.

Despite expense, hardware-assisted capture is made attractive by its ability to timestamp packet arrival at the earliest possible moment, with a high-resolution counter (thereby avoiding interrupt latency); to filter unwanted packets (thereby avoiding wasted memory allocation and disc storage); to record only the components of interest (typically just the header, thereby further saving wasted memory and disc utilisation); and to prove the integrity of a trace by showing that no packets were lost — or, if they were, how many and where in the trace they would have appeared (which not all software solutions can guarantee.)

We describe the development of the first hardware-assisted packet capture tool for the NetFPGA[6] platform. Our solution overcomes the inadequacies of software-based solutions at a fraction of the cost of other, well-established hardware solutions [3]. It is capable of accurate timestamps sustained at full-rate Gigabit Ethernet, while preserving the flexibility of other PC-based solutions.

II. Related Work

Several works on passive measurement systems have been proposed in the literature over the last few years. Our work originates from the need for a cheap, flexible architecture able to timestamp packets with high accuracy, for which reason, we propose a cooperative PC/NetFPGA architecture.

The Endace DAG card [3] is a widely respected benchmark for hardware measurement solutions. Newer DAG cards provide a limited filtering capability of up to eight bidirectional rules, where our solution provides for up to 32 one-way rules. As we shall show, timestamp accuracy in our solution is comparable with that of the DAG, yet a DAG card costs as much as three times that of a NetFPGA.

In [20], Wolf *et al.* propose Distributed Online Measurement Environment (DOME), a distributed system of passive measurement nodes equipped with Intel IXP2400 network processors. Their work includes header anonymization schemes with performance of flows up to 500 Mbit/s of minimum-sized packets (64 bytes), as compared with full line-rate capability of both Endace DAG cards and our solution.

In [18], Ficara *et al.* propose an architecture which combines the flexibility of general purpose PCs (equipped with `libpcap` based applications) with the power of the Intel IXP2XXX network processor (NP) family. In this scenario, the NP applies early filtering techniques before forwarding traffic to other sensors, according to “locality buffers” (hash-based load balancing). This system provides a timestamp accuracy

978-1-4673-0269-2/12/$31.00 © 2012 IEEE

of microseconds where, in contrast, our solution competes with the DAG resolution of ten nanoseconds. A related project, SCAMPI [9], developed a framework for high speed traffic monitoring and filtering which relies on the COMBO6 card [2], an FPGA-based network adaptor developed by CESNET. SCAMPI implements traffic filtering in the FPGA, thus off-loading packet classification and filtration from host software.

Finally, nCap [14] and related works provide software-based measurement techniques that work well, but are at the mercy of kernel-based timestamping. These software solutions are both inexpensive and flexible, but traffic load and timestamp quality are both limited by NIC hardware and kernel performance.

Hardware solutions typically provide very good timestamp quality, but hardware is typically expensive and offer limited flexibility — especially in the case of proprietary offerings. A NetFPGA-based solution offers the accuracy of hardware timestamping on a platform which, thanks to support from Xilinx (*inter alios*), is very affordable, with the flexibility of open firmware and the associated, rapidly growing community of developers and academics.

III. NetFPGA Platform

NetFPGA [6] is a low-cost platform, developed by the High Performance Networking Group at Stanford University, primarily as a tool for teaching networking hardware and router design. It is a standard PCI card that plugs into a standard PC. The card contains a Xilinx Virtex-II Pro Field-Programmable Gate Array (FPGA) which is programmed with user-defined logic and has a clock of 125 MHz. The PCI interface connecting the host PC to the NetFPGA is managed by a small Xilinx Spartan II FPGA. Four Gigabit Ethernet ports, 4.5 MiB of static RAM (in 2 banks) and 64MiB of DDR2 dynamic RAM are also on-board.

A reference package containing Verilog source for the FPGA, C code for the host PC and Java code for the graphical interface, which can be downloaded from the NetFPGA website, provides basic networking functions including a Network Interface Card (NIC), a PW-OSPF IPv4 Router, and a Layer 2 switch. Together, they constitute a hardware-accelerated network research platform which complements the flexibility of a host PC, and provide the framework on which we build our measurement solution.

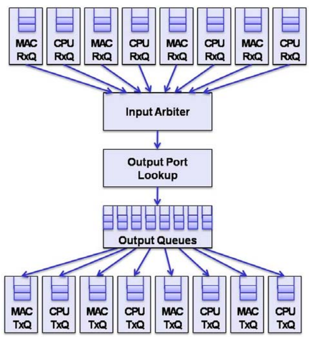

Fig. 1. The NetFPGA Reference Pipeline[16].

The platform can support the implementation of the control and forwarding plane of an IPv4 router. Thanks to its modularity, NetFPGA allows researchers to explore new ideas for next generation networks. The reference pipeline, Figure 1 [16], presents eight receive queues, eight transmit queues and a user data path in which custom modules (such as an output port lookup module) are inserted. Each "MAC" (Media Access Controller) is a physical network port with an associated queue, and each MAC queue has an associate CPU queue used for communication between the NetFPGA and the host PC. The input arbiter services the set of eight input queues in a round robin fashion to supply a 64-bit wide packet pipeline.

IV. Architecture

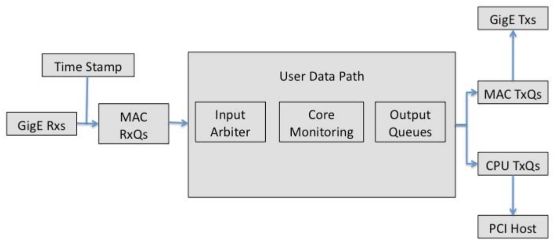

Fig. 2. The overall monitoring scheme.

A network monitor may either be installed in-series with the link to be monitored, or connected by means of a network tap. Optical network links make the choice easy: passive optical splitters are inexpensive, and other than during initial installation, offer no possibility of interruption of the link.

Copper network links, on the other hand, are more challenging. Some protocols, such as 10/100 Ethernet can be tapped using a passive resistive network, but others (including Gigabit Ethernet) either require an expensive active tap (such as the NetOptics TP-CU3 [7]), or installation of the monitor in-line. In-line monitoring is cheap, and offers the possibility of building an Intrusion Prevention System, but comes at the cost of significant extra latency and the risk of interruption of the link, should the monitor lose power, be misconfigured, or otherwise fail.

Since we aim for a low cost solution, we choose to install the NetFPGA in-series with a link (using the card's extra ports as an integrated, full-duplex copper tap), however end-users certainly may opt for an external active copper tap where reliability is essential. Where cost is especially sensitive, our solution could be modified to monitor two full-duplex links with ease.

Our solution is cooperative, meaning that the NetFPGA takes care of filtering and timestamping traffic of interest, while existing `libpcap`-based software provides a familiar user interface, and flow selection is accomplished by commands to a simple, shell-like CLI (Command Line Interface).

While all received traffic is relayed via the NetFPGA's network ports, only traffic that matches a filter rule is passed to host software (with associated timestamp). Packets that don't match are counted and passed on.

A. Hardware Plane

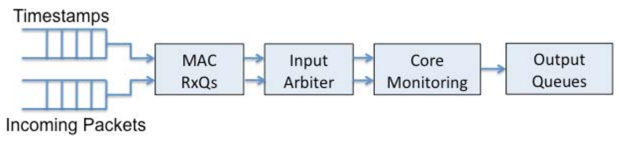

Fig. 3. The Timestamp and Packet data streaming are passed in parallel.

Figure 2 is a high-level block diagram of our monitoring solution. The Output Port Lookup module found in the reference pipeline is replaced by our Core Monitoring module and, for the sake of minimising FIFO-induced jitter, the Timestamping module is inserted at the RGMII interface. Figure 3 shows how timestamps are carried in a side-band channel (parallel with packet data) to minimise latency, and also to avoid packet bandwidth inflation, which could lead to dropped packets.

We take careful precautions to ensure that timestamps are discarded when packets are discarded, to prevent the mismatch of packets with timestamps. Should a packet be dropped (*e.g.* due to congestion or a bad CRC), the timestamp of that packet is available to the host via a pair of PCI registers.

1) Timestamping — a naïve solution: Packets are timestamped at the very earliest possible moment (*i.e.* at the RGMII interface) to minimise jitter caused by FIFOs. In our first implementation, timestamps were generated from a 64-bit counter driven by the 125 MHz free-running system clock, giving one increment every 8 ns (in contrast with the 96 ns inter-frame gap of Gigabit Ethernet). Using the system clock means the timestamp counter is synchronous with the data path, obviating the need for clock domain crossing logic which would add jitter.

Since Ethernet preamble can be variable length, we sample the timestamp counter when the Ethernet start-of-frame delimiter arrives — again, to minimise jitter.

2) Timestamping — a more accurate solution: While this naïve implementation was simple, it provided no means by which to correct oscillator frequency drift. Moreover, it produced timestamps expressed in units of 8 ns, where they would more usefully be a fixed-point representation of time in seconds (*i.e.* units of 2^{-32}) — and such conversion requires a division operation difficult to implement in hardware.

Both problems can be solved by means of Direct Digital Synthesis (DDS) [19], a technique by which arbitrarily variable frequencies can be generated using FPGA-friendly, purely synchronous, digital logic and indeed, this is how the DAG cards generate their timestamps [15].

Briefly, DDS involves an accumulator register to which the supplied rate constant is added every clock cycle. The output is an enable signal (suitable for a counter) generated from the overflow of this accumulator. For example, a counter controlled by DDS with a 32-bit rate constant of 0x8000_0000 and a 125 MHz base clock would increment at 62.5 MHz because the accumulator would overflow only every other cycle.

The DDS constant of width W_{DDS} bits can be determined for synthesised frequency, f_s, and base clock frequency, f_c, by Equation 1:

$$DDS_{const} = 2^{W_{DDS}} \times \frac{f_s}{f_c} \tag{1}$$

By choosing a DDS constant of approximately 0x8970_5F41, the 125 MHz system clock will increment the timestamp counter once every 2^{-26} seconds (or 14.9 ns). When shifted left by 6 bits, each timestamp becomes a fixed-point, 64-bit representation of time in seconds in the upper 32 bits, and fractions of seconds in the lower 32 bits.

By reference to an external time-base (such as GPS pulse-per-second), software can adjust the DDS rate constant to reach the target frequency of 2^{26} Hz, and local oscillator drift is thus corrected.

This control loop is best implemented in a kernel device driver, as it is in the DAG. The results obtained by using an external reference in this way are well documented in [15]. We note that, although the current NetFPGA has no easy GPS input, one could be added without too much difficulty; the (new) NetFPGA 10Gb/s has a serial interface making GPS interface trivial.

We also note that the logistics of installation of GPS equipment can be quite difficult, and so we wished to explore other options for drift correction. We describe our approach in detail in Section V.

3) Core Monitoring: While the internal NetFPGA datapath (by its nature) can cope with full-rate, minimum-sized packets (potentially each with a new flow), the NetFPGA PCI interface lacks the bandwidth to record all traffic, so we provide a 5-tuple (protocol, IP address pair, and port pair) filter of up to 32 entries, implemented in the "Core Monitoring" module.

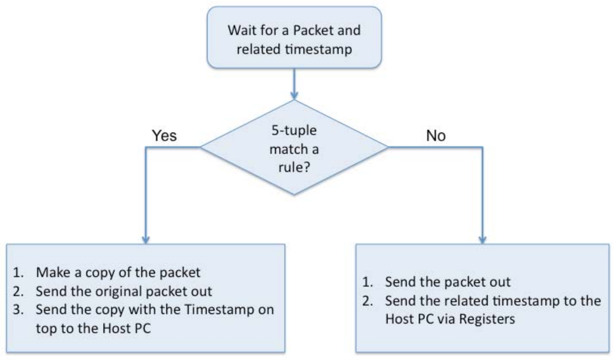
Wait for a Packet and related timestamp
5-tuple match a rule?
Yes
No
1. Make a copy of the packet
2. Send the original packet out
3. Send the copy with the Timestamp on top to the Host PC
1. Send the packet out
2. Send the related timestamp to the Host PC via Registers

Fig. 4. Flow Diagram of the Core Monitoring.

We implement this filter using a pair of 16-entry, SRL16E-based TCAMs as provided by the Xilinx CoreGen tool [11]. As shown in Figure 4, the 5-tuple is the search pattern and, one cycle later, the match result is returned. For matching packets, we prepend the packet's timestamp in Intel byte-order (for the sake of convenience for host software) and direct the result to a CPU queue. The format of these packets is illustrated in Figure 5.

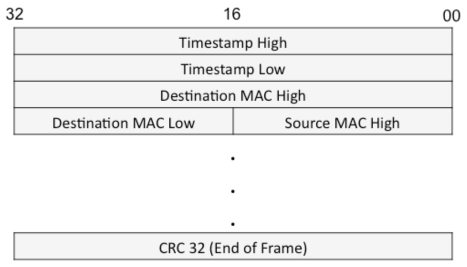

Fig. 5. Format of the Packet sent to the CPU.

As with matching packets, non-matching packets are re-transmitted via a network port. In addition, the timestamp for non-matching packets is made available to the host via PCI registers. Our present implementation only records the timestamp of the most recent non-matching packet, but we intend to provide a mechanism for sending the timestamp and 5-tuple of non-matching packets to the host so the user can fine-tune the filter rule-set.

These TCAMs are optimised for lookup performance. Where lookup operations complete in just one cycle, write operations take 16 cycles to complete. Rule updates may take place during an active capture session without causing loss of forwarded packets, however some packets may miss the filter until the update is complete. We consider this an acceptable compromise given that rule updates are unlikely to happen frequently.

For the purposes of proof-of-concept, we elected not to try automatically both combinations of source and destination address and port, meaning separate rules are required to match each direction of a flow. This feature could be added without difficulty, but we feel that acceptable results (and certainly higher densities) might be obtained by making use of a Bloom filter instead of TCAMs. We feel that the possibility of a few false positives is acceptable, and could be handled by additional filtering on the host (if required), provided enough unwanted packets are eliminated by the filter. A counting Bloom filter would make possible on-line filter updates, if required, or else if update frequency is low, the entire Bloom filter could be rewritten on each update.

B. *Software Plane*

We modified the NetFPGA kernel device driver to strip off the prepended timestamp and to store this timestamp in the packet's `struct sk_buff`, before the packet is passed on. A recent 2.6 kernel that contains a `ktime_t` timestamp in `struct sk_buff` must be used. Provided this modified driver is in use, our solution is backwards compatible with all standard `libpcap` applications.

In order to access nanosecond granularity timestamps, we modified `libpcap` (and `tcpdump` along with it) to return `struct timespecs` instead of `struct timevals`, and any applications that wish to get nanosecond timestamps must be likewise modified.

It is our ambition to support the live `libtrace` [5] capture interface (and thereby Endace [3] ERF format).

1) Filter management: Beyond these modifications, we provide an auxiliary command-line tool for TCAM rule management, which also initialises the hardware timestamp counter with the current date and time. This tool can list the rules set in hardware, insert and delete individual rules, and load a rule-set from a file. Each rule is a 5-tuple with an associated don't-care mask for each field.

2) Statistics collector: We also provide a statistics daemon (with an interface for Nagios) which tracks captured traffic (*i.e.*, only traffic that matches the currently loaded rule-set), and optionally records this traffic with nanosecond timestamps to disc in a pcap-compatible file.

The daemon keeps aggregate counts for total packets, bytes, IP and non-IP, TCP and UDP, TCP SYN/FIN/RST and options, as well as mean inter-arrival time and mean bit rate.

V. IRCT: Independent Rate-Controlled Timestamper

Choosing a DDS constant to give a timestamp in the desired units is a matter of calculation, but correcting for oscillator drift without reference to an external time-base is more challenging.

Since no two oscillators will drift at the same rate, we attempt to use two local, independent oscillators to estimate drift. While such an approach can never outperform that of a GPS-corrected DAG, we thought it worthwhile to see how close to that standard this technique could get us.

A side benefit is that the algorithm can be implemented entirely inside the FPGA, without even kernel support. We note that NTP might provide better correction, but anything involving the host must somehow cope with PCI latency [17]. This is a matter for future work.

In essence, we use a second oscillator to derive a pulse-per-second strobe. At each second, the timestamp counter is sampled and compared with the last second's timestamp. The difference, minus one second, represents the dynamic drift, δ_{drift} of one oscillator relative to the other. Since we have no means of estimating absolute drift, we include a fixed constant, κ, that accounts for absolute drift and which must be determined empirically.

We then add a proportion, $2^{-\alpha}$, of this error term into the current DDS constant:

$$DDS_{const,n} = DDS_{const,n-1} - (\delta_{drift} \gg \alpha + \kappa) \qquad (2)$$

Small α results in undamped oscillation of DDS_{const}, and large α limits the minimum amount of error that can be corrected. Choosing a suitable value for α is crucial to the success of this algorithm. Empirical evidence suggests that 10 provides a good balance (remembering that the six LSbs of the timestamp are 0s), and results in an error floor of $\approx$200 ns.

VI. Performance evaluation

We evaluated the actual performance of our monitoring system through a variety of experimental tests. All tests are taken by means of Endace DAG 4.3ge SX card or Spirent AX4000 traffic analyser [1], which is an ASIC-based tool. We test the latency through the NetFPGA card, the quality of the timestamps returned to the host and the maximum number of packets that we are able to send from the card to the host PC without losing data.

A. Latency

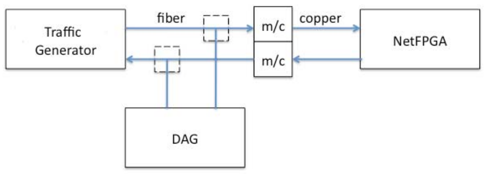

Fig. 6. System setup for latency estimation.

In the first experiment, we characterised the latency through the NetFPGA with an Endace DAG 4.3ge SX, as shown in Figure 6. Being optical, we were obliged to use a pair of media converters (Allied-Telesyn AT-MC-1004), and we did not have the means at our disposal to calibrate out the latency contributed by these devices. We measured latency through the two converters and the NetFPGA card at a constant 2.4 μs, irrespective of whether the test packets matched a filter rule, or how many rules were programmed into the filter.

B. Timestamp accuracy

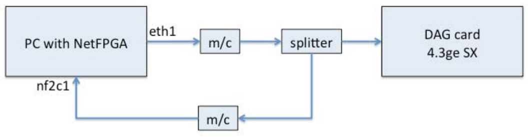

Fig. 7. System setup for timestamp accuracy.

We tested also the quality of timestamps returned to the host against the DAG card, using as timestamping module both the "naïve solution" and the one with the IRCT implementation. Note that although our first plots included error bars, we felt that they added little to the picture, and have omitted them from the following figures.

We used `tcpreplay` with a real traffic trace as "software traffic generator". In the first set of experiments we used the naïve solution and we compared the two absolute drift (Figure 8) and the relative drift between the two oscillator

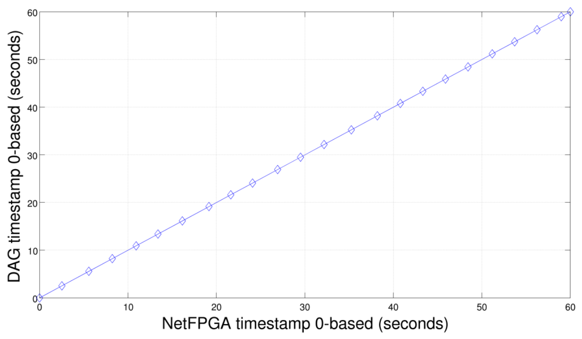

Fig. 8. Comparison of the two absolute drift with the naïve timestamping module.

(Figure 9) using the system setup shown in Figure 7. "nf2c1" stands for one of the four physical ports of NetFPGA board (where our design is loaded), while "eth1" is one of the ports of a NIC connected to the PC where NetFPGA is hosted. We sent 1000 packets (one-minute trace) and as we can see in Figure 8 the inter-arrival times of packets recorded by our solution are exactly the same as those achieved with the DAG card. Figure 9 however shows the relative drift of our solution with respect the DAG card. We lose approximately 1.7 milliseconds in 60 seconds. The relative drift continues to increment very fast with the passing of the time because of the used timestamping module do not provide means of correcting for oscillator drift.

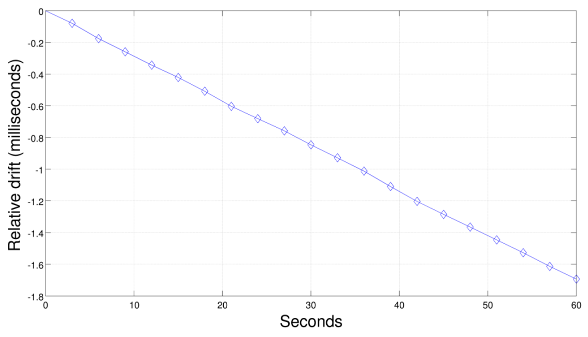

Fig. 9. Comparison between the two oscillator with the naïve timestamping module.

Then we tested our solution with the IRCT module, instead of the naïve one, against the DAG card. We used the same system setup of the previous experiments (Figure 7) and we used different values of absolute drift parameter, κ, in the correction module (Equation 2).

The main purpose of this test was to understand how the correction system of IRCT reacts with the passing of the time and the role of κ. As no particular speed was required for this test, we sent 18000 packets in about three hours in order to have a good overview of the benefit introduced by the IRCT system with respect the naïve solution. As we can see in

Figure 10 introducing the κ parameter allowed us to improve the performances up to 3125% compared the solution with a control loop without the κ parameter. In fact, using a value of 118 the system lose approximately 0.8 milliseconds in three hours against the 25 milliseconds lost by the one with κ equals to zero. As stated before, using as PPS source something with its own drift led to the need of a constant term in Equation 2 that try to compensate this drift during the passing of the time.

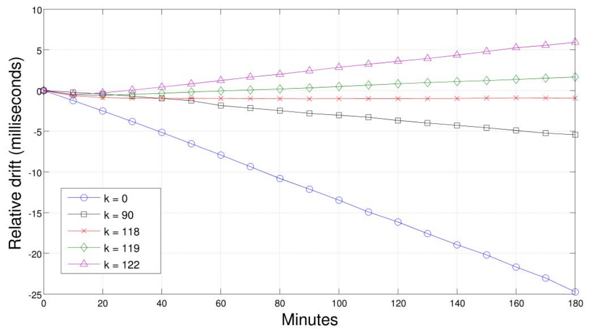

Fig. 10. Comparison between the two oscillator with the IRCT module.

C. From the card to the host PC: maximum throughput

In the last set of experiments we tested the maximum number of packets that we are able to send from the card to the host PC without losing data. The processing of the packet in hardware does not represent a bottleneck for the system. We are able to process up to 4 Gb/s without losing data because of during the life of the packet in hardware our modules do not access at any external memory (*i.e.* SRAM or DRAM) avoiding in this way possible hardware bottlenecks. The problems could arise when a lot of packets are pushed to the host PC through the PCI bus. In this case the performances strictly depends on the way the transfer from HW to NetFPGA kernel driver is done. For this reason we connected the Spirent AX4000 to the NetFPGA board and we exploited it to generate high-rate traffic. We disabled our filter module (on the NetFPGA board) in order to push all the received packets to the host PC through the bus PCI. We used the “Statistic collector” daemon in order to check how many packets have been received in user-space. We created CBR traffic with 64 B length packets in order to test the system in the worst case scenario.

Figure 11 shows the obtained throughput. As we can see in the worst case scenario it is possible to pass to the host PC up to 90,000 pkt/s. This is due in particular, because of:

- Hardware does not implement any kind of interrupt mitigation.
- NetFPGA kernel driver is an old only interrupt-driven one.

Since, in the current NetFPGA1G driver, no interrupt mitigation is implemented, an interrupt per packet is created thus leading to a significant loss of performance. For these

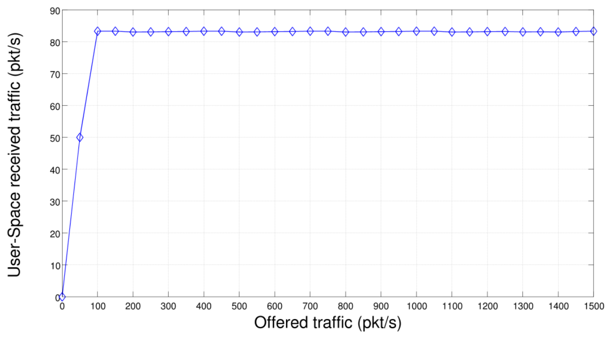

Fig. 11. Throughput from the board to the host PC with growing rates of the traffic.

reasons we are planning to change the NetFPGA kernel driver implementing NAPI (New API). NAPI is an interface to use interrupt mitigation techniques for networking devices in the Linux kernel. Such an approach is intended to reduce the overhead of packet receiving. The idea is to defer incoming message handling until there is a sufficient amount of them so that it is worth handling them all at once. Also, in the Linux kernel 2.6.35 onwards, it is possible to take advantage of RPS (Receive Packet Steering) feature if the host CPU count is more than one. Usually, NAPI + RPS on a multicore CPU is done just because RPS helps to distribute interrupts across CPUs in the multi-processor system. Finally, one more bottleneck in Linux kernel network stack, in receive direction, happens when socket buffers are copied from kernel space to user-space. The ”Zero Copy” feature available for packet sockets allows to share the buffer between kernel-space and user-space and thereby avoids copying of data.

All these features will be very useful in the design of the new NetFPGA board that will allow the user the process up to 40 Gb/s of traffic with four 10G Ethernet ports.

VII. Conclusions and Future Work

The paper introduces a flexible yet cost-effective passive monitoring system based on a cooperative PC/NetFPGA architecture. Our implementation shows promising results, and compares favourably with a widely-recognised commercial system for traffic while having a significantly lower cost — particularly for academic institutions. The NetFPGA is a promising platform on which to develop low-cost and open-source instrumentation devices. The release of the new board (NetFPGA10G: a NetFPGA card capable of both 1 and 10Gb/s) provides a high-performance monitoring system supporting 10Gb/s per port based upon the designs outlined in this paper.

We plan to add support for an external time-base (where available) and expand considerably the available pool of selection filters by replacing the current TCAM-based filter with a Bloom filter. Work is already underway to adapt our solution for the AXI-Stream environment found in the NetFPGA 10G.

REFERENCES

[1] *Ax4000, http://www.spirent.com.*

[2] *Combo6, http://www.liberouter.org.*

[3] *Endace, http://www.endace.com.*

[4] *Lawrence Berkeley National Labs, tcpdump/libpcap, Network Research Group, http://www.tcpdump.org.*

[5] *Libtrace, http://www.wand.net.nz/trac/libtrace.*

[6] *NetFPGA, http://www.netfpga.org.*

[7] *NetOptics, http://www.netoptics.com.*

[8] *Ntop network traffic probe, http://www.ntop.org.*

[9] *SCAMPI project, http://www.ist-scampi.org.*

[10] *Wireshark protocol Analyzer (was Ethereal), http://www.wireshark.org.*

[11] *Xilinx, http://www.xilinx.com.*

[12] L. Deri. Passively monitoring networks at gigabit speeds using commodity hardware and open source software. In *PAM*, San Diego, CA, 2003.

[13] L. Deri. Improving passive packet capture: Beyond device polling. In *SANE*, Amsterdam, NL, 2004.

[14] L. Deri. ncap: Wire-speed packet capture and transmission. In *End-to-End Monitoring*, Nice, France, 2005.

[15] S. F. Donnely. *High Precision Timing in Passive Measurements of Data Networks*. PhD thesis, Waikato University, 2002.

[16] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo. NetFPGA–an open platform for gigabit-rate network switching and routing. *IEEE Int'l Conf. on/Multimedia Software Engineering*, 2007.

[17] D. J. Miller, P. M. Watts, and A. W. Moore. Motivating future interconnects: a differential measurement analysis of pci latency. In *Proc. 5th ACM/IEEE Symp. on Architectures for Networking and Communications Systems*, 2009.

[18] A. D. Pietro, D. Ficara, S. Giordano, F. Oppedisano, G. Procissi, and F. Vitucci. A network processor based architecture for multi gigabit traffic analysis. *International Journal of Communication Systems*, 22(11), 2009.

[19] P. Saul. Direct digital synthesis. In *Circuits and systems tutorials*, 1996.

[20] T. Wolf, R. Ramaswamy, S. Bunga, and N. Yang. An architecture for distribuited real-time passive network measurement. In *IEEE MASCOTS*, 2006.