OSNT: Open Source Network Tester

Gianni Antichi[†‡‡], Muhammad Shahbaz[‡‡‡], Yilong Geng[*‡‡], Noa Zilberman[†] Adam Covington[*],
Marc Bruyere[¶‖**], Nick McKeown[*], Nick Feamster[‡], Bob Felderman[††], Michaela Blott[§], Andrew W. Moore[†],
Philippe Owezarski[¶‖],
[*]Stanford University [†]University of Cambridge [‡]Georgia Tech [§]Xilinx [¶]Université de Toulouse [‖]CNRS,LAAS
[**]DELL [††]Google [‡‡]These authors contributed equally to this work

***Abstract*—Despite network monitoring and testing being critical for computer networks, current solutions are both extremely expensive and inflexible. Into this lacuna we launch the Open Source Network Tester (OSNT), a fully open-source traffic generator and capture system. Our prototype implementation on the NetFPGA-10G supports 4×10Gbps traffic generation across all packet sizes and traffic capture is supported up to 2×10Gbps with naïve host software. Our system implementation provides methods for scaling and coordinating multiple generator/capture systems and supports 6.25ns timestamp resolution with clock drift and phase coordination maintained by a GPS input. Additionally, our approach has demonstrated lower-cost than comparable commercial systems while achieving comparable levels of precision and accuracy; all within an open-source framework extensible with new features to support new applications, while permitting validation and review of the implementation.**

***Index Terms*—Open Source, Programmable Hardware, High Speed, NetFPGA, Monitoring, Traffic-generation, Open Source, Packet-Capture and Packet-Sniffing.**

I. Introduction

COMPUTER networks are the hallmark of 21st Century society and underpin virtually all infrastructure in the modern world. Consequently, society relies on the correct operation of these networks. To achieve compliant and functional equipment, effort is put into all parts of the network-equipment lifecycle. Testing validates new designs, equipment is tested throughout the production process and new deployments are rigorous testing for compliance and correctness. In addition, many owners of network equipment employ a relentless battery of testing and measurement to ensure the infrastructure operates correctly.

The continuous innovation that is such a desirable property of the Internet has also led to a dilemma for network testing. For a typical piece of new networking equipment there will be a multitude of related IEEE standards and standards-track IETF RFCs, each one requiring test cases to ensure correctness for network-equipment. This has led to a multi-billion dollar industry in network test equipment giving rise to companies such as Ixia, Spirent, Fluke, and Emulex/Endace among others.

However, such equipment has evolved with a number of undesirable characteristics: commonly closed and proprietary systems with limited flexibility well outside the reach of most universities and research laboratories. Even a modest two port 10GbE network tester capable of full line-rate costs upward of $25,000 and adding support for additional protocols, large numbers of TCP streams, and non-trivial traffic profiles quickly increases this price. This has been the case for two reasons. Firstly, network test equipment capable of full-line rate with high-precision timestamping is a significant engineering challenge, leading to state-of-the-art and specialist physical components. Secondly, test equipment is often developed simultaneously with early prototype network equipment. Thus, modest numbers of units sold mean an expensive and slow time to develop test hardware and software.

This slow development cycle and high expense opens an opportunity for an open-source network tester. It is no longer necessary to build network testers on top of specialized, proprietary hardware. There are multiple open-source hardware platforms with the potential for line-rate across many 10GbE ports, for example, the NetFPGA-10G[1], Xilinx VC709[2] and Terasic DE5-Net[3]. Each of these fully-reprogrammable cards purports being capable of running at line-rate. For example, the NetFPGA-10G has 4×10GbE interfaces, is based on a Xilinx FPGA, and is available to the research and teaching community for less than $2,000 including firmware and software.

We therefore present the Open-Source Network Tester (OSNT[4]), primarily for the research and teaching community. Such a tester needs to be able to achieve full line-rate, provide sufficiently accurate timestamping and be flexible enough to allow new protocol tests to be added to the system. We believe that, as an open-source community grows, a low-cost open-source network tester will also prove valuable to the networking industry. We also envisage the enabling of new testing and validation deployments that are simply financially impractical using commercial testers. Such deployments may see the use of hundreds or thousands of testers, offering previously unobtainable insights and understanding.

In this paper we present an architecture for OSNT, describe our first prototype based upon the NetFPGA open-source hardware platform, and present early-day benchmarks illustrating the tester in operation. OSNT is portable across a number of hardware platforms, maximizing reuse and minimizing reimplementation costs as new hardware, physical interfaces and networks become available. By providing an open-source solution we invite everyone from the community to audit (and improve) our implementation as well as adapt it to their needs.

[1]http://www.netfpga.org
[2]http://www.xilinx.com/products/boards-and-kits/EK-V7-VC709-CES-G.htm
[3]http://www.de5-net.terasic.com
[4]http://www.osnt.org

II. Related Work

Network testers, and open-source network testers are not new; uniquely, OSNT brings the incorporation of designs that operate intimately with the hardware. Our efforts ride the established tradition of network measurement and testing that exists in the network research and academic communities.

A small sample of open-source and community projects include: Iperf [1] and later Netperf [2], developed to provide performance tests of throughput and end-to-end latency. Traffic loads from previously captured *pcap* files could be transmitted using Tcpreplay [3]. Netalyzer [4] uses bespoke server and client infrastructure to measure many aspects of Internet performance and behaviour. Swing [5] provided a closed-loop traffic generator: first monitoring and characterizing, and then regenerating system load replicating the measured characteristics. Early attempts at both flexible and feature-rich traffic generation led to the Ostinato [6] traffic generator. The netmap [7] achieves near-optimal host throughput but is still restricted by the underlying hardware for timestamps, traffic-shaping and maximum-rate capacity. A final example, Bonelli *et al.* [8] describe a near-line-rate traffic on a 10Gbps link that uses multi-core multi-queue commodity hardware, albeit without the flexibility or guarantee of full line-rate throughput, precise traffic replay timing and sufficient packet capture timestamp accuracy and precision.

Commercial network testers are provided by a number of companies: Ixia and Spirent dominate, but other test equipment manufacturers also have network-test offerings. Despite their ability to perform at high line-rate, a criticism common to all these systems is the cost and inflexibility. Supporting newly-designed protocols is often expensive while supporting newly-designed physical line standard can result in an entirely new system.

In the measurement community the ubiquitous *pcap* program, *tcpdump*, has been the tool of choice for network capture. However, capture-system performance (and rates of loss) are dictated by the underlying host: a combination of hardware, operating-system, device-drivers and software. Additionally, it is rare for these software systems to provide any common clock across the captures, making end-to-end latency measurements complicated and inaccurate. There have been software/hardware efforts in the past that incorporate GPS-coordinated high-precision hardware timestamps and use device-driver designs intended to mitigate loss under load [9]. However, this work was limited to 1GbE and serves now only to provide a motivating example. NTP is a mature time synchronization method; however, it can only achieve an accuracy better than 1ms under limited conditions [10]; making it unsuitable for high precision traffic characterization.

In contrast to the large range of commercial offerings available to generate traffic; the high-precision capture market has few commercial systems and is dominated by the Endace[5] DAG card.

Several previous NetFPGA-based projects using the previous generation NetFPGA 4×1GbE platform have also provided traffic-generation [11] and traffic-monitoring [12]. The architecture of OSNT has been heavily informed by the designs, limitations and experience with these systems.

[5]Endace became a division of Emulex following acquisition in 2012-13.

III. The OSNT Architecture

The OSNT architecture is motivated by limitations in past work: closed-source/proprietary solutions, high costs, lack of flexibility, and the omission of important features such as timestamping and precise packet transmission. Alongside flexibility there is a need for scalability; while our prototype work has focused on single-card solutions, our desire to reproduce real operating conditions means we must have a system that can test beyond single network elements; a production network needs be tested as close as possible to its real operating conditions — this means the OSNT system must also be able to recreate such real operating conditions.

From the outset it has been obvious that flexibility must be a key part of the OSNT approach. This flexibility is needed to accommodate the variety of different uses for OSNT. Four distinct modes of use have become clear.

- *OSNT Traffic Generator*: a single card, capable of generating and receiving packets on four 10GbE interfaces. By incorporating timestamps into each outbound packet, information on end-to-end delay and loss can be computed. Such a system can be used to test a single networking element, *e.g.*, switch or router, or a network encompassed within a sufficiently small area that different inputs and outputs from the network can be connected to the same card.
- *OSNT Traffic Monitor*: a single card, capable of capturing packets arriving through four 10GbE ports, transferring them to the host software for analysis and further processing. Alongside a range of techniques utilized to reduce the bottleneck of PCIe bandwidth (packet-batching, ring-receivers and pre-allocated host system memory), packets are optionally hashed and truncated in hardware. The card is intended to provide a loss-limited capture system with both high-resolution and high-precision timestamping of events in a live network.
- *Hybrid OSNT system*: our architecture allows the combination of Traffic Generator and Traffic Monitor into single FPGA device and single card. Using high-precision timestamping of departing and arriving packets, we can perform full line-rate, per-flow characterization of a network (device) under test.
- *Scalable OSNT system* is our approach for coordinating large numbers of multiple traffic generator and traffic monitors synchronized by a common time-base to provide the resources and port-count to test larger network systems. While still largely untested, such a coordinated system has been a design objective from the outset.

The OSNT architecture is designed to support these needs for network testing using a scalable architecture that can utilize multiple OSNT cards. Using one or more synchronized OSNT cards, our architecture enables a user to perform measurements throughout the network, characterizing aspects such as end-to-end latency and jitter, packet-loss, congestion events and more.

It is clear our approach must be capable of full line-rate operation. To this end we built our prototype upon the

NetFPGA-10G platform — an open-source hardware platform designed to be capable of full line-rate. We describe our prototype implementation in section VI.

While there is a clear need that one or both of the traffic-capture and traffic-generator cores in our OSNT system be present in each use case; these two subsystems have orthogonal design goals: the capture system is intended to provide high-precision inbound timestamping with a loss-limited path that gets (a subset of) captured packets into the host for further processing, whereas the traffic-generator requires precision transmission of packets according to a generator function that may include close-loop control, (*e.g.*, TCP) and even (partial) application protocol implementation.

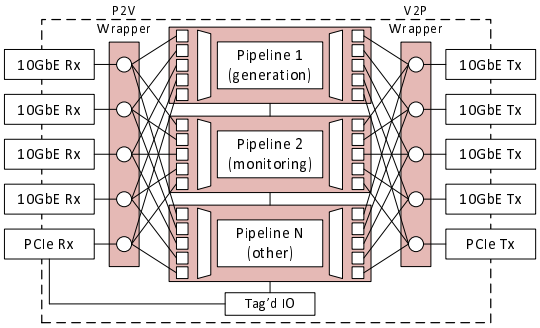

Fig. 1: NetV - an approach for NetFPGA Virtualization.

Given we already had a proven starting design for both generator and capture engines [11], [12], along with a keen desire to employ component reuse, we were led to develop the *NetV* approach that virtualizes the underlying hardware platform[6]. The approach, shown in Figure 1, extends a hardware platform such as the NetFPGA, using *P2V*: Physical to Virtual and *V2P*: Virtual to Physical wrappers. The V2P hardware-wrapper is a per-port arbiter that shares access among each of the 10GbE and PCIe interface-pipelines. This permits multiple NetFPGA pipelines within a single FPGA fabric on a single board. In turn providing support for seamless integration of existing pipelines with strong isolation characteristics. For example, a traffic generator can co-exist with a high-precision capture engine. Each pipeline is tagged with a unique ID to ensure register accesses can be distinguished among different pipelines. In this manner, traffic generation and monitoring can be implemented either as standalone units or as a combined system on a single card. Using multiple pipelines in the same design does not affect the overall performances as long as they do not share data structures. The only limitation is given by the available FPGA resources.

Our design has focussed upon one particular architectural approach; this direction was selected to maximize code reuse at the expense of potential redundant gate-logic. Other OSNT architectures may be appropriate but are not explored here for sake of brevity.

[6]Our reference prototype is the NetFPGA, but we believe that the architecture including approaches such as *NetV* will be generic across a range of hardware platforms.

IV. Traffic Generation

The OSNT traffic generator both generates packets and analyzes return statistics. It is designed to generate full line-rate per card interface, and is scalable in a manner that allows for multiple traffic generators to work in parallel within a single OSNT environment. Traffic generation features include:

- support large number of different traffic flows
- flexible packet header specification over multiple headers
- support several standard protocols
- sufficient flexibility to test future protocols
- simulate multiple networking devices/end-systems (e.g. routers running BGP)
- allow timestamping of in and out-bound packets
- allow per-packet traffic-shaping
- statistics gathered per-flow or flow-aggregate
- support for negative testing through malformed packets

In addition to the above features, OSNT can be customized to support different protocols, numbers of flows and many other features in each given application context.

Figure 2 illustrates the high-level architecture of the traffic generation pipeline. The center of the pipeline is a set of micro-engines, each used to support one or more protocols at network and transport-layers such as Ethernet, TCP or UDP and application-protocols such as BGP. Each micro-engine either generates synthetic or replays captured traffic for one or more of the selected egress interfaces. A basic micro-engine is a simple packet replay: a set of pre-defined packets are sent out a given number of times as configured by the software. Each micro-engine contains three building blocks: Traffic Model (TM), Flow Table (FT) and Data Pattern (DP). The Traffic Model contains information about the network characteristics of the generated traffic, such as packets' size and Inter-Packet Delay (IPD). It is a compiled list of these characteristics, extracted by the host software and installed into the hardware. Each parameter is software defined, permitting arbitrary rate distribution patterns: *e.g.*, Constant Bit Rate (CBR) or Poisson distribution. The Flow Table contains a list of header template values used by the micro-engine when generating a packet. Each packet-header is defined by the Flow Table. In this manner, multiple flows with different header characteristics can be generated by a single micro-engine. The micro-engine takes each header-field and manipulates it in one of several ways before setting it: a field may remain constant, incrementally increase, interleave, be set randomly or set algorithmically. The number of flows supported by the Flow Table depends on the trade-off between trace complexity and the number of fields to be manipulated. The Data Pattern module sets the payload of a generated packet. The payload can be set to a random pattern, or a pre-specified pattern. A pre-specified pattern allows a user to set the payload of packets to a unique pattern so that the user can execute specific network tests such as continuous-jitter measurement. It also provides in-payload timestamping of departing packets and capabilities for debugging/validating received packets.

Packets generated by the micro-engine are sent to a per-port Arbiter. The arbiter selects among all the packets destined for a port from each micro-engine. Ordering is based upon

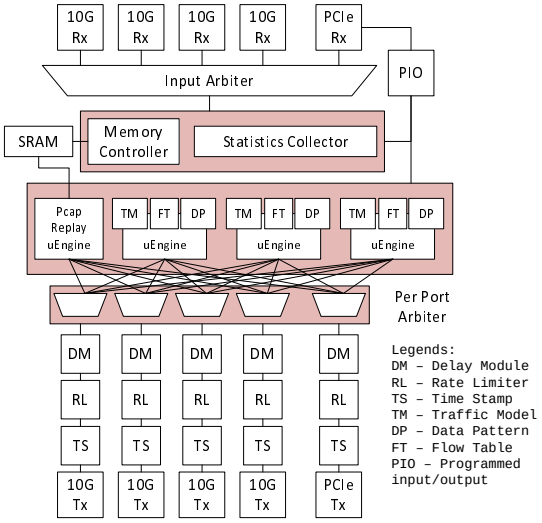

Fig. 2: The architecture for OSNT traffic generation system.

the required packet departure time. A Delay Module (DM) located after the arbiter will delay packets by each flow's Inter-Packet Delay. A Rate Limiter (RL) guarantees that no flow exceeds the rate assigned to it at each port. Lastly, the packet goes to the (10GbE) MAC, from which it is transmitted to its destination.

The traffic generator implementation can also receive incoming packets and provide statistics on them at either port or flow level. This allows use of the traffic generation subsystem as a standalone unit without an additional external capture subsystem. To this end, packets entering the card through a physical interface are measured, the statistics gathered and the received packets discarded. The gathered statistics are relayed to host software using the programmed input/output (PIO) interface.

The traffic generator has an accurate timestamping mechanism, located just before the transmit 10GbE MAC. The mechanism, identical to the one used in the traffic monitoring unit and described in section V, is used for timing-related measurements of the network, permitting characterization of measurements such as latency and jitter. The timestamp is embedded within the packet at a preconfigured location and can be extracted at the receiver as required.

As for the software side, we provide an extensible GUI to interact with the HW (*e.g.*, load a PCAP trace to replay in HW, define the per-packet inter-departure time, etc.).

V. Traffic Monitoring

The OSNT traffic monitor provides four functions:

- packet capture at full line-rate
- packet filtering permitting selection of traffic-of-interest
- high precision, accurate, packet timestamping
- statistics gathering

Figure 3 illustrates the architecture of the monitoring pipeline that provides the functionality enumerated above. The 5-tuple (protocol, IP address pair and layer four port pair) extraction is performed using an extensible packet parser able to recognize both VLAN and MPLS headers along with IP in IP encapsulation. Further flexibility is enabled by extending the parser implementation-code as required.

A module positioned immediately after the Physical interfaces and before the receive queues timestamps incoming packets as they are received by hardware. Our design is an architecture that implicitly copes with a workload of full line-rate per port of minimum sized packets. However this will often exceed the capacity of the host-processing, storage, etc., or may contain traffic of no practical interest. To this end we implement two traffic-thinning approaches. The first of these is to utilize the 5-tuple filter implemented in the "Core Monitoring" module. Only packets that are matched to a rule are sent to the software, while all other packets are dropped. The second mechanism is to record a fixed-length part of each packet (sometimes called a *snap-length*) along with a hash of the entire original packet. The challenge here is that if a user is interested in all packets on all interfaces it is possible to exhaust the host resources. We quantify the PCIe bandwidth and the tradeoff for snap-length selection in section VII.

As for the software side, we provide a python-based GUI that allows the user to interact with the HW components (*e.g.* enable cut/hash, set filtering rules, check statistics). A C-based application that comes with it records the received traffic in both PCAP or PCAPNG format. This allows offline use of common libpcap-based tools (*e.g.* TCPDump, Wireshark.) These tools do not work directly with OSNT: the device driver secures performance by bypassing the Linux TCP/IP stack. We refer the reader to the OSNT website for further information about the software API.

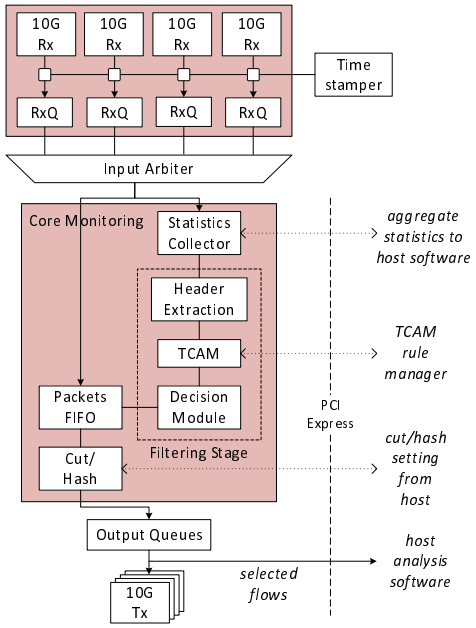

Fig. 3: The architecture for OSNT traffic monitoring system.

Timestamping

Providing an accurate timestamp to (incoming) packets is a critical objective of the traffic monitoring unit. Packets are timestamped as close to the physical Ethernet device as

possible so as to minimize FIFO-generated jitter and permit accurate latency measurement. A dedicated timestamping unit stamps packets as they arrive from the physical (MAC) interfaces. Each packet is appended with a 64-bit timestamp.

Motivated by the need to have minimal overhead while also providing sufficient resolution and long-term stability, we have chosen to use a 64-bit timestamp divided into two parts, the upper 32-bits count seconds, while the lower 32-bits provide a fraction of a second with a maximum resolution of approximately 233ps; the practical prototype resolution is 6.25ns. Integral to accurate timekeeping is the need to correct the frequency drift of an oscillator. To this end, we use Direct Digital Synthesis (DDS), a technique by which arbitrary variable-frequencies can be generated using synchronous digital logic[13]. The addition of a stable pulse-per-second (PPS) signal such as that derived from a GPS receiver permits both high long-term accuracy and the synchronization of multiple OSNT elements. The selection of a timestamp with this precision was a conscious effort on our part to ensure the abilities of the OSNT design are at least as good as the currently available commercial offerings.

VI. OSNT NetFPGA-10G Prototype

Our prototype implementation of the OSNT platform has been on the NetFPGA-10G open-source hardware platform. The NetFPGA system provides an ideal rapid prototyping target for the work of OSNT. Since its original inception as an open-source high speed networking platform for the research and education community [14] and, through its second-generation platform [15], the NetFPGA has proven to be an easy-to-use platform. The NetFPGA platform supplies users with both basic infrastructure and a number of pre-worked open-source designs intended to dramatically simplify a users' design experience.

The NetFPGA-10G card, as shown in figure 4, is a 4 port 10GbE PCIe adapter card incorporating a large FPGA fabric. At the core of the board is a Xilinx Virtex-5 FPGA: XC5VTX240T-2 device. There are five peripheral subsystems that complement the FPGA. The first one is the network interface subsystem, with four 10Gbps SFP+ Ethernet interfaces, connected through PHY devices to the FPGA. It is through these interfaces that network-traffic enters and leaves the FPGA. A Gen1 PCIe subsystem is the host-bus adapter for interfacing NetFPGA and PC. This allows both register-access as well as packets transfer between the platform and the motherboard. The third subsystem is memory consisting a combination of both SRAM and DRAM devices. The memories were selected to provide minimal latency and maximal bandwidth over the available FPGA I/Os, and can be configured for a variety of purposes. Example uses include: lookup tables, packet buffering, and counters. The fourth subsystem is an expansion interface, terminating twenty SerDes links, intended to host a daughter card or to communicate with another board. The last subsystem is concerned with the configuration of the FPGA; the FPGA can be configured through either a standard JTAG interface or from either of two FLASH devices connected to the FPGA through a CPLD. The board is implemented as a three-quarter length PCIe adapter, but can also operate as a standalone unit outside the server environment.

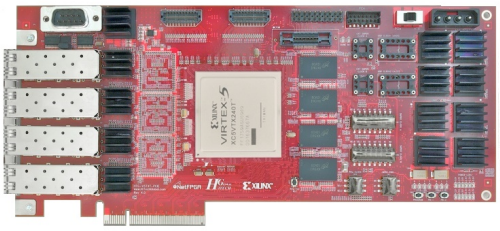

Fig. 4: The NetFPGA-10G board.

VII. Experiences with our prototype

By building our prototype on the NetFPGA-10G platform we have inherited several platform constraints. Despite having a large FPGA device, design decisions must trade resources. One example of this is in the sizing of TCAM tables for filtering. Table size is traded directly against overall design size. In our prototype implementation, the tuple-based filtering tables is limited to 16 entries.

While the internal NetFPGA datapath has been designed to accommodate full line-rate, minimum-sized packets, the PCIe interface lacks the bandwidth to transmit all traffic to or from the host. The NetFPGA-10G provides a first generation, 8-lane PCIe implementation. This interface uses an MTU of 128 bytes and without careful packing a naïve implementation of DMA and device driver may achieve as low as 33.5% utiliazation (for transactions of 129 byte packets). Furthermore, even for an ideal scenario this interface imposes a limit of around 13.1 Mpps for an MTU of 128 bytes or a little over 15 Gb/s. It is clear that capture-to-host of all four interfaces when operating at 10Gb/s into the host is not practical. Alongside flow-filtering the traffic-thinning technique of selecting a snap-length places a known limit on the maximum amount of data that needs to be transferred over the PCIe to the host.

The option to add a hash of the original packet, along with a fixed snap-length, means that we can reduce the potential number of bytes per packet to a known upper boundary. Although the hash adds an overhead of 128 bits per packet, it permits practical packet identification which in turn means we can perform end-to-end latency measurements as well as identifying specific loss-events. The ability to do bandwidth limiting in this way allows us to achieve a maximum rate of approximately 21.7 Mpps provided we use non-naïve DMA and device-driver mechanisms.

Fortunately, there has been considerable progress in non-naïve DMA and device-driver mechanisms to reduce the bottleneck of PCIe bandwidth; packet-batching, ring-receivers and pre-allocated host system memory have all seen use in past dedicated capture systems [9]. Recent efforts such as netmap achieve rates of 14.8 Mpps into user-space for single port commodity 10GbE interface cards. Our architecture is not limited to a current hardware-implementation; the OSNT

system when running on more advanced hardware such as the Xilinx VC709, using the third generation PCIe, has sufficient bandwidth to support full size payloads for all four 10GbE ports. In fact, the open-source nature of OSNT means that having this system operate effectively on any future NetFPGA platform, other platforms from Xilinx or indeed from other FPGA vendors is no more complicated than the porting of any open-source project.

We validated the actual OSNT performance against the IXIA 400T[7] equipped with 2 ports 10G each. IXIA provides the capability of both generating full line rate traffic and full line rate monitoring; permitting validation of both capture and generation capabilities.

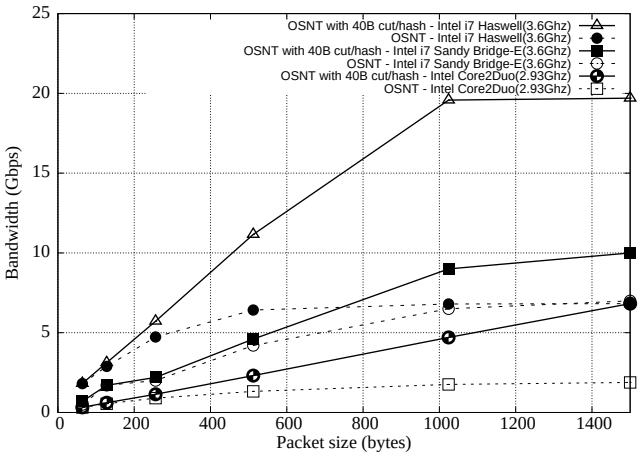

Fig. 5: The OSNT loss-less per-packet capture engine performance for various presented traffic loads.

Figure 5 shows capture engine performance results of a pure-polling device driver on several combinations of motherboard and CPU. Any non-naïve device driver is quickly able to improve on these results; these provide merely a baseline noting the considerable reliance on CPU clock rate. One results is the significant skew between old systems (*e.g.* Intel Core2Duo) and newer ones (*e.g.* Intel Haswell). For this experiment the bandwidth is defined as the maximum traffic presented to the system without loss; the impact of the cut/hash feature at reducing traffic across the PCIe is clear.

Testing of the traffic-generator we were able to confirm to our satisfaction that the OSNT Traffic Generator is able to generate full line rate over two ports independently of the packet length. Tests were conducted over a range of packet-sizes with results compared directly against IXIA-based generators. In all experiments data was generated (and measured) on all four NetFPGA ports with a combination of IXIA and Endace packet-capture and measurement.

VIII. Conclusions

In this paper we introduced OSNT, an open source network tester. We described the OSNT architecture which permits a flexible combination of multiple packet-processing pipelines using a new virtualization technique, NetV. While the NetV virtualization approach was designed with the NetFPGA in mind, this technique is not bound to that hardware and should be able to provide flexibility and versatility across a range of uses. Using the NetV approach we showed how the OSNT system can implement both traffic-generator and network monitor functions. We also described our prototype implementation using the rapid-prototyping NetFPGA platform and characterized aspects of that implementation.

The OSNT platform provides a network tester that is able to combine desirable software flexibility with the advantages of being built upon an open-source hardware platform. The versatility of OSNT is in its suitability for a range of applications, from the testing of single items of networking equipment to the characterizing of large distributed networks.

The OSNT system will be available to the research community through the NetFPGA project. Any user who owns a NetFPGA card can simply use it, with no additional hardware expense. We envisage the project being extended and enhanced by the research community, and users are encouraged to contribute further features and capabilities, as well as to share their own experience using OSNT.

The promise of OSNT is an exciting one. In the field of network measurement alone, high-precision, loss-limited capture has led to remarkable progress in the characterization and understanding of the modern Internet. The OSNT traffic monitor overcomes the two biggest issue with these capture deployments to date — the cost and lack of flexibility — while also, by virtue of being open-source, providing an auditable test system that encourages repeatability in network science.

Thanks

We thank the NetFPGA community, David Fermor, Scott Whyte and Richard Hay for inspiring and assisting with this work. The language and form of this paper has been improved immeasurably by feedback from Jon Crowcroft and the anonymous reviewers.

References

[1] *iperf, TCP and UDP bandwidth performance measurement tool, http://code.google.com/p/iperf.*
[2] *Netperf, http://www.netperf.org.*
[3] *Tcpreplay, https://github.com/synfinatic/tcpreplay.*
[4] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson, "Netalyzr: Illuminating The Edge Network," in *Internet Measurement Conference (IMC)*.
[5] K. V. Vishwanath and A. Vahdat, "Swing: Realistic and responsive network traffic generation," *IEEE/ACM Transactions on Networking*, vol. 17, no. 3, pp. 712–725, 2009.
[6] P. Srivats, "OSTINATO: An open, scalable packet/traffic generator," in *FOSS.IN*, 2010.
[7] L. Rizzo, "Netmap: a novel framework for fast packet i/o," in *USENIX Annual Technical Conference (ATC), 2012*. USENIX, 2012.
[8] N. Bonelli, A. Di Pietro, S. Giordano, and G. Procissi, "Flexible high performance traffic generation on commodity multi–core platforms," in *Traffic Monitoring and Analysis*. Springer, 2012, pp. 157–170.
[9] A. Moore, J. Hall, C. Kreibich, E. Harris, and I. Pratt, "Architecture of a network monitor," in *Passive & Active Measurement Workshop*, 2003.
[10] M. Ussoli and G. Prytz, "Sntp time synchronization accuray measurements," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2013*. IEEE, 2013.
[11] A. Covington, G. Gibb, J. W. Lockwood, and N. Mckeown, "A packet generator on the NetFPGA platform," in *IEEE Symposium on Field Programmable Custom Computing Machines (FCCM), 2009.* IEEE, 2009, pp. 235–238.

[7] www.ixiacom.com/products/display?skey=ch_400t

[12] G. Antichi, S. Giordano, D. J. Miller, and A. W. Moore, "Enabling open-source high speed network monitoring on NetFPGA," in *IEEE/IFIP Network Operations and Management Symposium (NOMS), 2012*. IEEE/IFIP, 2012, pp. 1029–1035.

[13] P. Saul, "Direct digital synthesis," in *Circuits and Systems Tutorials*, 1996.

[14] 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," in *IEEE International Conference on Microelectronic Systems Education (MSE), 2007*. IEEE, 2007, pp. 160–161.

[15] M. Blott, J. Ellithorpe, N. McKeown, K. Vissers, and H. Zeng, "FPGA research design platform fuels network advances," *Xilinx Xcell Journal*, no. 73, 2010.