XGT4: an Industrial Grade, Open Source Tester for Multi-Gigabit Networks

Leonardo R. Juracy[1], Felipe B. Lazzarotto[1], Daniel Pigatto[2], Ney L. V. Calazans[1], Fernando G. Moraes[1]
[1]PUCRS-FACIN - Ipiranga Av., 6681 - Porto Alegre - Brazil, 90619-900
[2]Teracom Telemática - Av. América 1000 - Eldorado do Sul - Brazil, 92990-000
daniel@datacom.ind.br, {ney.calazans,fernando.moraes}@pucrs.br

***Abstract*—Network test needs grow as fast as network performance. These must be reliable, remain available without interruption, and secure user data. Thus, they must continually be tested for service quality, based on measurements like throughput, latency, and jitter. Network and network equipment tests can rely on test software, dedicated test hardware, or in FPGA platforms. The latter offer a trade-off between software solutions limited bandwidth and high performance dedicated test equipment. This work proposes a new, industrial grade, low-cost and accurate network test solution, XGT4. XGT4 employs the NetFPGA SUME board and provides full support to the RFC2544 test benchmarks for 1G and 10G Ethernet equipment, a feature not available in other open source testers. Besides, XGT4 can be remotely operated through a dedicated web interface, and deals with multiple simultaneous 10GbE streams. Results of using XGT4 to test a commercial DUT report real-world results for all RFC2544 tests: throughput, latency, back-to-back, frame loss rate, system recovery and reset. An additional feature is timed throughput runs, to conduct bit error rate tests. The XGT4 hardware has a small area footprint, taking less than 20% of the platform FPGA.**

I. Introduction and Related Work

Current high-speed communication networks feed the ever growing demand of Internet-connected devices for fast data exchange. As the number and bandwidth of such devices increase without any foreseeable limit, network backbones are stressed and impose strong requirements on network performance and associated standards.

Network testing needs grow in the same scale as network performance. Since networks must be reliable, remain available without interruption and secure user data, to cite a few of their mandatory properties, they must constantly be tested to check their service quality, which can be measured in terms of throughput, latency and packet loss. Less than 20 years ago appeared the RFC2544 (https://www.ietf.org/rfc/rfc2544.txt), a standardized way to benchmark a range of network appliances. Previous works have proposed traffic generator and traffic monitor implementations, the main test benchmark modules, but few of these addressed support to this RFC until recently. Nonetheless, the RFC2544 is widely used in industry. It enables, for example, providing throughput rate, latency and loss-rate standard definitions and metrics.

Network and network equipment tests count with architectural solutions based on: (*i*) test software; (*ii*) dedicated test equipment hardware; (*iii*) fast prototyping (FPGA) hardware. Software-based approaches can be highly configurable, but limit tests' accuracy and range, due to intrinsic software speed limitations. Hardware-based solutions improve the accuracy compared to software, increase the support to high speed network devices, but reduce flexibility [1].

FPGA platforms currently benefit from open source initiatives like the NetFPGA [2]. Besides the availability of several related platforms (e. g. NetFPGA-1G, NetFPGA-10G and NetFPGA-SUME), this initiative provides free hardware intellectual property (IP) cores and associated free software.

Among recent works, however, no open source tester found in the literature provides full support to the RFC2544 or features like LFSR signature for Bit Error Rate Test (BERT) measurements, an interesting feature while testing e. g. switches. Also, just one approach cites a GUI availability [3], where a local Python-based interface improves usability, but which does not enable web access to the system. Table I summarizes the features of the works found in the literature. Ruiz et al. [4] and Wang et al. [5] approaches are similar to the XGT4 proposed herein. However, only the proposal of Ruiz et al. presents results for 10Gb/s networks.

TABLE I: Comparison among revised network test approaches. Legend: "*": supported, "X": unsupported , "P": partially supported , "NA": not addressed.

Approach	Traffic Generation	Traffic Monitoring	RFC2544 Support	Open Source	10Gb/s Support	Web Interface
Ruiz (2016) [4]	*	*	X	NA	*	NA
Wang (2015) [5]	*	*	P	NA	X	NA
Antichi (2014) [3]	*	*	X	*	*	*
Groléat (2013) [6]	*	X	X	*	*	X
Tockhorn (2011) [1]	*	*	X	NA	X	NA
Covington (2009) [7]	*	*	P	NA	X	NA
Both (2009) [8]	*	*	P	X	X	X
This work	*	*	*	*	*	*

This work proposes a new, low-cost and accurate network test solution, the XGT4 tester (Ten(X) Gigabit(G) Tester(T) with Four(4) Streams). Industrial grade means XGT4 has been tested with commercial hardware operating in real world network environments. XGT4 employs the NetFPGA-SUME board. It provides full support to the RFC2544 test benchmark for 1G and 10G Ethernet equipment. XGT4 can be operated via a dedicated web interface and may simultaneously deal with multiple streams.

II. System Overview

The XGT4 system architecture comprises three main components, as Figure 1 details: (*i*) the client, that configures and runs tests remotely over the Internet; (*ii*) the host, with a web service, the middleware and the FPGA hardware (in the platform FPGA); (*iii*) the device(s) under test (DUTs). The tester supports rates up to 10Gb/s.

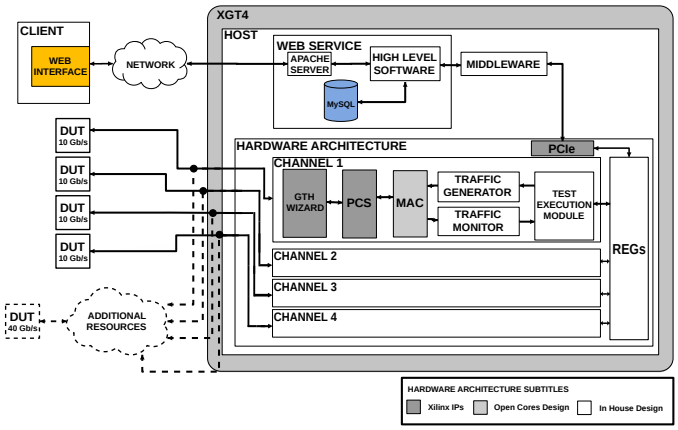

Fig. 1: The XGT4 system architecture block diagram.

The client may remotely run tests through a web interface. As the hardware infrastructure enables simultaneously testing several DUTs, the host supports multiple connections. The web service executing at the host provides communication with the FPGA board. A database (MySQL) queues the tests received from clients, which the web service manages. This software communicates with a middleware, which configures the hardware according to a parameter set sent by some client. The middleware is the software that launches each test. It communicates with the FPGA using a PCI Express (PCIe) bus, writing test parameters and reading test results. It also receives the raw results to process, and transmits these to the client(s). The user can also execute the middleware as a standalone program, running tests without the web software.

III. The Web and Middleware Softwares

The FPGA contains 4 identical, independent and configurable 10GbE testers, enabling the concurrent test of 4 networks. A register bank (REGs in Figure 1) interfaces hardware and software. The test execution module reads the bank and controls the traffic generator module, producing a traffic stream. This stream goes to a DUT, which returns it to the traffic monitor. The test execution module computes the latency, throughput and frame loss rate, among other performance figures. During the test, the middleware periodically reads the bank to collect partial results, storing them in its database. By the test end, the middleware transmits final results to the client.

The *web software* comprises the web interface and the web service. It enables remote access to the DUTs. This feature allows testing the performance of networks without an operator physical presence. Also, it provides a short learning curve, with a user-friendly interface to users. Figure 2 presents a screenshot of the web interface for a test configuration. Here, the user first selects the tester to employ. Each tester is a *channel*. After selecting the channel, he chooses the RFC2544 tests to execute, the frame sizes, the flow rate, the acceptable loss rate and the MAC addresses. These parameters are transmitted to the web service (running on the host), which queues the test. Each test to execute is called a *job*. Besides the RFC2544, the interface enables the (*i*) loopback and (*ii*) timed throughput services. It is also possible to let an LFSR generate the payload. The web service contains four FIFO execution queues, one for each channel. This enables executing one job per channel. If more than one job is scheduled in the same channel, latter jobs execute once the channel becomes available. The web interface displays a job list, signaling the status of each one as *waiting*, *running*, *canceled* or *done*. Done jobs present links to access results stored in the database.

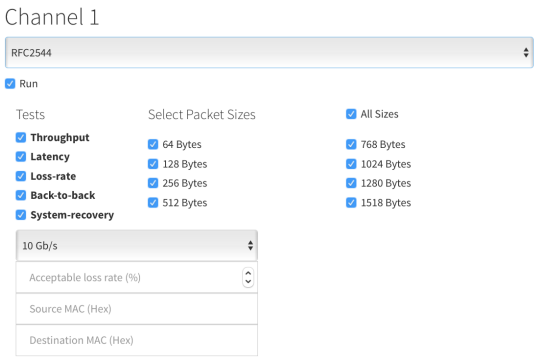

Fig. 2: Web interface screenshot forming the RFC2544 test configurations.

The *middleware* is a software that computes test results: RFC2544 tests and custom services. It can execute in a host or in an embedded processor. A set of input parameters defines the tests to execute, which are transmitted to the NetFPGA-SUME through the PCIe interface. Data received from this interface determine the results computation.

RFC2544 tests. The RFC2544 is a benchmarking methodology for network devices. All tests run in a closed-loop form, including a DUT and a channel. The RFC2544 tests cover:

- **Throughput**: measures the maximum rate at which none of the offered frames are dropped by the DUT, defined as Th_{max}. The hardware transmits multiple data streams, with different rates. To discover the DUT maximum supported rate, it employs binary search algorithms, optimizing test performance.
- **Latency**: measures the round-trip time taken by a test frame to travel through the DUT and back to a test port in some channel, using Th_{max}. The test inserts an identifier in a frame after generating data for one minute. The latency is measured as the delay to receive back the frame with the specific identifier.
- **Back-to-back bursts**: measures the longest frame burst at Th_{max} that the DUT handles without losing frames. This measurement indicates the DUT buffering capacity.
- **Frame loss rate**: detects the loss-rate values for throughput rates above Th_{max}. This is useful to report the DUT performance in overloaded states.
- **System recovery**: measures the speed at which a DUT recovers from an overload condition. The test applies a rate 10% above Th_{max} for 60 sec. After this time, throughput reduces to 55% of Th_{max}, measuring the time the DUT takes to accept all frames.
- **Reset response**: measures the speed at which a DUT recovers from a hardware or software reset.

Custom Services. The XGT4 also implements additional tests:

- **Timed throughput**: XGT4 can generate and apply a data stream during a specific time period, using a fixed frame size, with the goal of performing a DUT BERT. The user configures the payload using one LFSR, or using either an all zeros or an all ones pattern. The web interface can be used to define the payload and choose an LFSR polynomial and seed. These can also be directly furnished as parameters to the middleware.
- **Loopback**: XGT4 can loopback a given channel. A loopback is defined in hardware, after the MAC module. This test sets the channel in promiscuous or filtering modes.

IV. Hardware Architecture

XGT4 uses the NetFPGA SUME platform [2], which contains a Xilinx Virtex-7 XC7VX690T device. The board provides four external SFP+ ports and an x8 PCIe Gen3 interface to communicate with a host. Figure 1 presents the hardware modules inside the *Hardware Architecture* rectangle.

Traffic Generator - Produces UDP Echo request datagrams, with different headers (source and destination IP and MAC addresses) and payloads (frame ID, all ones, all zeroes or 64-bit LFSR signature). The user configures the traffic through either the web client or through direct middleware commands. This module sends frames to the MAC, signaling frame start and end.

Traffic Monitor - Receives frames from the MAC and verifies the frame payloads; filters frames by IP or MAC addresses (if requested); counts the number of received frames and can determine the current test end. Each test may have different end conditions and results to evaluate.

For throughput, back-to-back and frame loss rate tests, the traffic generator sends a fixed amount of frames. The traffic monitor counts the number of received frames, being this the test result. The test execution module controls the end of these tests using a timeout mechanism.

For the latency test, the traffic generator sends frames during one minute. After this, the middleware configures the traffic generator to send one frame with an identifier tag and a timestamp (ts_A). When the traffic monitor detects the identifier tag, it computes the latency, subtracting the current timestamp from ts_A.

For the system-recovery test, the traffic generator sends frames with a rate 10% higher than Th_{max} during one minute, inducing frames losses. Next, the traffic generator reduces the rate to 55% of Th_{max}, storing the current timestamp in ts_A. The traffic monitor detects a frame loss when the frame sequence number is not ordered. Upon a frame loss detection, ts_B stores the current timestamp. The test ends after the traffic monitor receives 10,000 frames in order, i.e. without frame losses. The system-recovery result is determined by subtracting ts_B from ts_A.

For a reset test, the traffic generator continuously sends frames at Th_{max}. All frames contain a sequence ID in the payload. The traffic monitor saves the last received frame ID and identifies when an out of sequence ID appears, which means that the DUT has recovered from a reset (a 64-bit sequence counter guarantees this condition is always fulfilled). The timestamp difference between these IDs is the test result.

Test Execution Module - Controls all test executions. It reads the test configuration from REGS. Next, it signals to the traffic generator module when to send a frame, counts the number of sent and received frames, and detects the test end. The test result is written back in REGS. The functionality corresponds to a 7-state finite state machine (FSM). From the initial state, when the middleware sets the status register, the FSM changes state, entering a 4-state loop where the traffic generator sends frames. In this loop, the FSM sends each frame individually, respecting the inter-frame gap (IFG). For the latency, system-recovery and reset tests the FSM goes straight to the initial state, after transmitting the frames specified in the test. For the remaining tests, the FSM transmits frames until a timer (set with a user-defined value, typically 5 seconds) expires.

Registers - Provides communication between the middleware and the hardware. They control all parameters related to test execution, including: (1) test type; (2) source and destination IP and MAC addresses; (3) size of the frame (in bytes); (4) inter-frame gap, in clock cycles (reference clock: 156.25 MHz); (5) number of frames to send; (6) status registers, to signal the start of a test; (7) MAC filtering, used in loopback tests. The test execution module also writes results to registers, including: (1) the number of received frames; (2) the number of bits with error, for timed throughput tests; (3) test results such as latency, recovery time, and maximum throughput.

Open Source and Xilinx IP Modules. Previous paragraphs described some hardware modules, all written in VHDL. Besides these, other IP Cores described here were employed (see Figure 1). The MAC Ethernet module is a 10Gb/s IP available from Opencores [9]. Its internal bus width is 64 bits, and the reference clock is 156.25 MHz, which allows handling 10Gb/s flows. This core provides communication between the traffic generator/monitor and the physical coding sublayer (PCS) IP. The PCS IP performs data encoding/decoding, scrambling/descrambling and has two FIFOs for synchronizing with higher speed interface clock domains. This is a modification of the 10GbE PCS Reference Design provided by Xilinx [10]. The SFP+ ports are connected to GTH transceivers. Xilinx provides a wizard to configure its FPGA on-chip transceivers [11], making available a set of protocols and encoding options. This design employs the 10GBASE-R protocol with 64b/66b encoding. Communication with the host employs a PCIe bus. Xilinx provides an IP [12], a generic solution for Virtex-7 FPGAs and an example design named Programmed Input/Output (PIO). This design was modified to read/write from REGS.

The BERT Hardware Architecture. The BERT is embedded in the traffic generator and traffic monitor modules. Its function is to detect payload errors. It contains two 64-bit parallel LFSRs, a bit error counter and some control logic. The same seed initializes LFSRs in the traffic generator and traffic monitor. The LFSR generates pseudo-random words to define the frame payload. An XOR gate detects mismatches between the received 64-bit words and the value generated by its own LSFR. If a mismatch is detected, the number of distinct bits found defines the bit error counter increment. If a desynchronization occurs between the LFSRs (e. g. due to a frame loss), the BERT infrastructure detects it and resynchronizes the modules using the received payload as an LFSR seed. It then waits for a specific number of frames without errors to ensure the modules are resynchronized.

V. XGT4 Validation and Evaluation Experiments

Figure 3 presents the XGT4 evaluation setup. It includes a DATACOM DM4100 10Gb/s Switch (http://www.datacom.ind.br) as the DUT, a client, and a host. The interconnection between the DUT and the XGT4 (NetFPGA-SUME) employs 10Gb/s optical Ethernet links. Clients access the host through a local network or through an Internet connection. This Figure shows a two-channel scenario: ch#1 generates test data and receives streams from the DUT; ch#2 is in loopback mode. The generated stream enters a DUT optical port, traverses a first loopback to another port of the DUT, goes to the XGT4 ch#2, returns to the DUT, and through a third loopback returns to XGT4 ch#1. With this configuration, it is possible to validate the traffic generation with third-party components, i.e. the DUT. For reset tests, the user must manually reset the DUT. For 1Gb/s tests, the optical ports are replaced by appropriate ports.

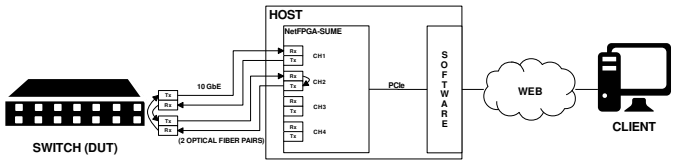

Fig. 3: XGT4 validation setup.

A. Test Results

1) Throughput: Figure 4 depicts the throughput test results using a graph, where the horizontal axis represents frame size and the vertical axis represents measured throughput. The following Equation furnishes throughput values: $Eff_{rate} = \frac{P_{size}}{P_{size}+UDP_{frame}+Gap} * 10$, where: Eff_{rate} is the effective throughput in Gb/s, P_{size} is the frame size, UDP_{frame} is the UDP protocol frame size (with fields Source Port, Destination Port, Length and Checksum) and Gap is the interframe gap size. The UDP protocol frame has 8 Bytes, and the XGT4 uses an interframe gap of 12 Bytes. Using $P_{size} = 64\ bytes$, the throughput is 7.62Gb/s, while for $P_{size} = 1518\ bytes$ the achieved throughput is 9.86Gb/s, as Figure 4 shows.

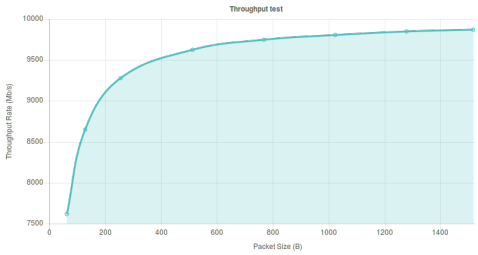

Fig. 4: Throughput test results.

2) Latency: Latency tests run after the throughput tests. These measure the time for a frame sent by the traffic generator module to traverse the DUT and return to the same tester channel. The result is a three-column table: (*i*) frame size.; (*ii*) throughput for the frame size (obtained from the throughput tests); (*iii*) latency result. Table II presents the latency results for the scenario of Figure 3.

TABLE II: Latency test results.

Frame Size (bytes)	Rate (Mb/s)	Latency(μs)
64	7619.04	1.2458
128	8648.65	1.2966
256	9275.36	1.3984
512	9624.06	1.6032
768	9746.19	1.8086
1024	9808.43	2.0125
1280	9846.15	2.2170
1518	9869.96	2.3744

TABLE III: System recovery test results.

Frame Size (Bytes)	110% rate(Mb/s)	55% rate(Mb/s)	Recovery Time (us)
64	818.60	451.28	29,048.72
128	959.13	528.53	172,544.59
256	1,012.95	549.14	184,326.53
512	1,053.10	549.57	230,364.80
768	1,073.72	549.71	629,043.19
1024	1,071.95	549.78	230,360.92
1280	1,071.54	549.83	1,917,479.25
1518	1,070.80	549.78	1,858,799.38

TABLE IV: Timed throughput test results.

Frame Size (Bytes)	Rate (Mb/s)
64	7,619.05
128	8,648.65
256	9,275.36
512	9,624.06
768	9,746.20
1,024	9,808.43
1,280	9,846.15
1,518	9,869.96
9,000	9,977.83

3) Back-to-back: The hardware sends a data stream, increasing the number of frames until some loss occurs. Bursts are always sent at Th_{max}, measured in the throughput test. The middleware interface allows selecting the frame number step size and maximum test frame burst. This test result relates frame size and the number of counted packets. The hardware stops the test after counting 1 million packets. If this number is reached, the test finished correctly.

4) Frame loss rate: This test starts with a throughput 10% higher than Th_{max}, decreasing it in steps of 10%, measuring the frame loss for each rate. Figure 5 plots the frame loss rate test results, with the DUT configured for 1Gb/s. With the DUT configured to 10Gb/s all frames are received (frame loss rate equals 0). The horizontal axis presents the stream rate (percentage of 1Gb/s), the vertical axis shows the loss rate. There is a curve for each specific frame size. As the packet size increases, the frame loss rate decreases up to 1.30% with a frame size equal to 1,518 bytes. The overhead due to the UDP protocol frame (8 bytes) and the interframe gap (12 bytes) leads to higher loss rates with smaller frame sizes.

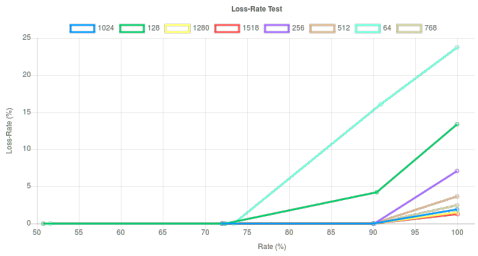

Fig. 5: Frame loss rate test results.

5) System Recovery: The traffic generator computes the time the DUT takes to recover from an overload condition. Table III presents the test results in four columns: (*i*) frame size; (*ii*) 110% of Th_{max}; (*iii*) 55% of Th_{max}; (*iv*) the recovery time. The latter is proportional to the frame size, because the tester must receive 10,000 frames in sequence without errors.

6) Reset: This test presents a simple result, the time it takes for the DUT to restart, establish connection and receive back 1,000 packets. The DM4100 switch showed a 50-second average time to recover.

7) Timed Throughput: In the the timed throughput test, the XGT4 sends frames for a period to execute a BERT. Inputs to this test are the frame size and the time to generate the stream. This test also supports jumbo frames. The following Equation computes the number of frames to generate: $N_{pkts} = \frac{Time}{I_{avg_{cycles}} * P_{size} * C_{per}}$, where: N_{pkts} is the number of frames to generate, $Time$ is the specified time, $I_{avg_{cycles}}$ is the average number of idle cycles used for each frame, P_{size} is the frame size and C_{per} is the clock period (6.4 ns). Table IV shows the measured rate for a 10-second time throughput test, for all RFC2544 frame sizes and a jumbo frame of 9,000 bytes. The BER is 0, since the test is executed in a controlled environment.

B. FPGA Resource Utilization

The XGT4 hardware was prototyped in the NetFPGA SUME FPGA, a XC7VX690T Virtex-7 device. The employed synthesis tool was *Xilinx Vivado*, version 2015-2. Table V shows resource occupation data for the XGT4 design. The design uses roughly 13% of the available FPGA resources.

TABLE V: FPGA resources taken by XGT4.

	Available	Overall Design	Single Channel	PCIe Core
Slice LUTs	433,200	49,774	10,993	4,349
Slice Registers	866,400	32,894	6,917	3,958
LUT FF Pairs	433,200	54,683	12,337	5,144
BlockRAMs	1,470	33	6	9
GTHE2 Channels	36	12	1	8
BUFGs	32	23	4	5

VI. Conclusions and Further Work

To the knowledge of the Authors, this article presents the most encompassing industrial grade, open source network tester proposal to date. XGT4 fully implements the RFC2544 test methodology and extends it for other measurement types. Besides, the proposed tester supports web-based operation. The XGT4 web interface enables remote access to client equipment for test operation. The included custom test services allow a more thorough network test, using techniques such as timed throughput. Compared to the available literature, few works implement the RFC 2455 methodology fully or allow remote access to execute tests as the XGT4 does.

Ongoing work includes modifying the XGT4 tester to operate with 40Gb/s networks, as well as investigations about other network test methodologies to include in XGT4. Extensions such as those proposed by Raumer et al. [13] to extend the functionality of the RFC2544 are under consideration.

References

[1] A. Tockhorn *et al.*, "A Configurable FPGA-based Traffic Generator for High-Performance Tests of Packet Processing Systems," in *ICIMP*, 2011, pp. 14–19.

[2] N. Zilberman *et al.*, "NetFPGA SUME: Toward 100 Gbps as Research Commodity," *IEEE Micro*, vol. 34, no. 5, pp. 32–41, 2014.

[3] G. Antichi *et al.*, "OSNT: Open Source Network Tester," *IEEE Network*, vol. 28, no. 5, pp. 6–12, 2014.

[4] M. Ruiz *et al.*, "Accurate and Affordable Packet-train Testing Systems for Multi-Gb/s Networks," *IEEE Communications Magazine*, vol. 54, no. 3, pp. 80–87, 2016.

[5] Y. Wang *et al.*, "An FPGA-based High-speed Network Performance Measurement for RFC 2544," *EURASIP Journal on Wireless Communications and Networking*, vol. 2015, no. 1, pp. 1–10, 2015.

[6] T. Groléat *et al.*, "Flexible, extensible, open-source and affordable FPGA-based traffic generator," in *HPPN*. ACM, 2013, pp. 23–30.

[7] A. Covington *et al.*, "A Packet Generator on the NetFPGA Platform," in *FCCM*, vol. 9, 2009, pp. 235–238.

[8] C. B. Both *et al.*, "FPGA Implementation and Performance Evaluation of an RFC 2544 Compliant Ethernet Test Set," *Int. Journal of High Performance Systems Architecture*, vol. 2, no. 2, pp. 107–115, 2009.

[9] I. Mohor, *Ethernet IP Core Specification*, 2002. [Online]. Available: http://www.opencores.org/projects/ethmac/

[10] J. Gaither and M. Cimadevilla, *10 Gigabit Ethernet/FibreChannel PCS Reference Design*, 2004, Xilinx, Inc., XAPP775, 7 p., Aug 2004.

[11] Xilinx, Inc., *LogiCORE IP 7 Series FPGAs Transceivers Wizard v3.3*, 2014.

[12] ——, *Virtex-7 FPGA Gen3 Integrated Block for PCI Express Product Guide*, 2012.

[13] D. Raumer *et al.*, "Revisiting Benchmark Methodology for Interconnect Devices," in *ANRW*, 2016, pp. 59–62.