FPGA-based load test accelerator

Anton Borodin[1]

[1] **Department of Computer Science, Moscow State Forest University, Mytischi, 141005, Russia**

Abstract

The information systems play the main role at a current time. These systems collect, control and give information for many users. Information by itself becomes the most valuable resource on a whole world. That is why reliability of computer systems is important. The main way to check, if the system works properly during development process, is testing. This paper represents results of our work on improvement of load testing. We propose new method of creating load, which is based on using a module with FPGA. On this paper we provided results of experiments for that module evaluation. Additionally to it, for compare, here there are represented results of experiments for estimation of a load capability of current computer systems. These results show how much time is needed for network stack processing on modern operation systems launched on computers with adequate hardware characteristics.

Keywords: *load testing, FPGA, load creation, load capability, OS network stack evaluation.*

1. Introduction

Computer becomes an important part of our live. We already can't imagine modern world without smartphones, laptops, Internet, social networks, online shops and etc. Computer now is a source of information, way of communication, and a tool for controlling different complex processes. Life of a human being, work of a small company or the whole country now is depending on information technologies. That's why quality of it becomes very significant. If quality of hardware is ensured by industrial production standards, quality of software is relying on developers. Testing is the most important tool that helps developers to maintain high quality of software product. There is a wide range of different types of tests, from functional to security testing.

At current time most of software systems are oriented at work with large amount of concurrent users. To guaranteeing high quality of those systems, developers need to know:

- Does system work correctly under load caused by expected amount of concurrent users?
- Does system performance violate service-level agreement under operating load?
- Does selected hardware is adequate to operating load?
- On what components they should focus their attention for future system expansion?
- How many users can be processed by system until performance will degrade?

A subtype of software performance testing called load-testing helps to answer on these and many more other questions. Load testing consist many stages [1] that can be tentatively merged in three big steps: preparing to testing, launching of a tests and results analysis. Studying of works related to load testing shows that attention of most researchers attracted to first and third step described above. There many different works focused on generating realistic tests [2-6], developing methods for testing of distributed systems [7-8], creating load models [9] and effective result analysis [10-11]. At the same time, stage of test launching doesn't get much attention of researchers. Effectiveness of that stage is depending on quality of load generator realization. Under effectiveness, in this case, we meant how much load can produce one computer with concurrent gathering performance metrics. Constant progress of information systems makes them complicated and high-powered, so load generator on a single computer can't create enough load to reach operating condition. That's why most of load testing tools offer different methods of increasing produced load. Current methods presented by cloud computing, distributed computer clusters and distributed computing based on computers connected by local network. Using cloud computing or distributed clusters require information system to be connected to global network which can be not acceptable for systems that still under construction and are not properly tested. Cloud computing and distributed clusters mostly used for finished software systems to model realistic load. On the other cases widespread method of load creation using distributed computing based on local network. On our work we propose an other method of creating load and gathering performance metrics. That method is based on using FPGA-based hardware module for load creation. This article represents results of our work.

2. Overview of load creation process

So-called load injectors are used for load creation at current time. These load injectors are a part of load testing tools. During work these injectors reads testing scripts, prepared at previous stages of load testing, and according to them make a lot of operating system API function calls. Each function call is forcing operating system (OS) to start network stack routines for sending a form by injector request. Network stack routines consist of many stages which request must pass. Additionally to it each stage of network stack have different protocols for various tasks. On most common ways process of request sending goes as follow. Depending on a size of request its divided on few pieces and each piece is put in a packet with additional service information necessary for its transmission. Then each packet goes through stage of route determination and

at the end of network routines execution it gets on network interface adapter (NIC). The NIC is implementing physical
packet transmission between two or more connected devices. Each request formed by load injector passes all these stages. Additional to it, every formed request can be checked by security components of OS. All these routines are handled by central processing unit. The more requests formed by load injector, the more processing resources are spent on network stack routines for its transmission. Additionally to it during test run stage is launched special programs for gathering performance metrics. These programs are analyzing incoming traffic from software system under load test. Moreover, because for load testing are used common computers with standard OS on the background are always launched different applications that can consume processing power. These factors additionally to hardware peculiarities are causing that a single computer doesn't have enough resources to produce enough load.

3. Load capability estimation

Maximum quantity of requests per second computer system can produce is limited by architecture of NIC [12]. Can computer reach that throughput depend on OS realization and hardware. In our work we took few experiments to measure time needed by OS for processing one request and to measure approximate load capability. These experiments were made on computers with technical specification represented at table 1.

For our experiments we used program "tcpdump" which is embedded on UNIX-like operating systems. That program is using low level library "pcap" for capturing packets on driver level of OS [13]. Each captured packet have timestamp that represent value of system time at moment of capturing. So to measure time needed by OS for processing one request we created experimental program that do following:

- Read system time.
- Put it on a packet.
- Send packet through network stack.

System time experimental program is inserting in UDP packet. That transport protocol has been chosen because of its simplicity and speed of work. Using it we can measure load capability on best case. Besides, during work, experimental program gradually increase amount of formed packet per second. Experiment consists of next steps:

- Launch the packet capture "tcpdump" program.
- Launch the experimental program.
- Wait until experimental program will finish it work.
- Stop execution of packet capture program.
- Result gathering and analysis.

First experiments showed that single threaded program couldn't form packets with big intensity. Experimental program was making about 100000 calling of

API functions OS to send system time, but nevertheless quantity of sent packets were much lesser. The more copies of experimental program were run, the more packets were sent per second. That's why during experiment few copies of experimental program were launched. Furthermore with number growth of experimental programs, packet loss is increasing during capturing due to lack of resources. To determine optimal experimental programs amount, we estimated dependence between packet loss and quantity. Estimated dependence for experimental platforms is represented on figure 1.

Table 1:Platforms description

Name	Tech. specs.	OS
Experimental platform № 1	Desktop Intel Core i7 920 2.67 GHz, DDR3-1066 9GB	CentOS 6.4
Experimental platform № 2	Laptop Samsung R580-JS03 Intel Core i5 430M 2.26 GHz, DDR3-1066 3 GB	Ubuntu 12.10
Experimental platform № 3	Laptop MacBook Air Intel Core i7 3667U 2 GHz, DDR3L-1600MHz 8 GB	OS X 10.8.5

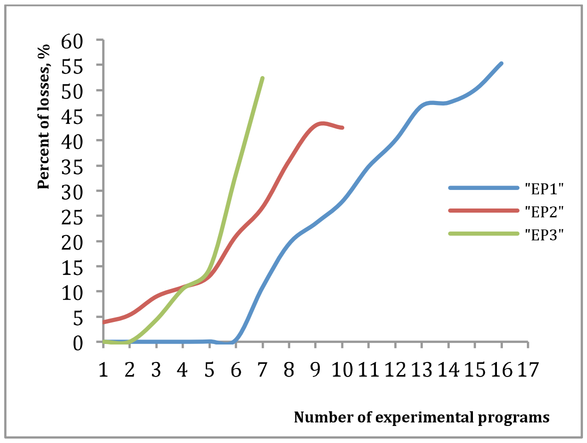


Figure 1. Dependence between percent of losses during capturing from number of launched experimental programs

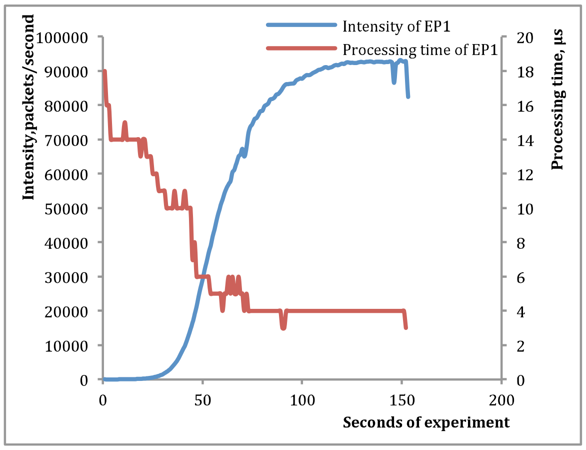


Figure 2. Curves of EP1 intensity and processing time during experiment

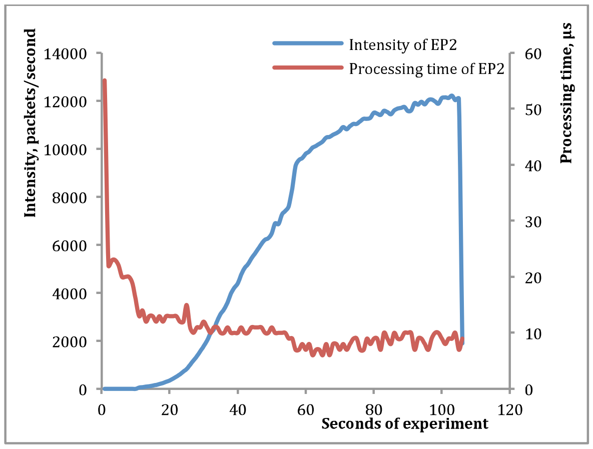


Figure 3. Curves of EP2 intensity and processing time during experiment

According to retrieved data for experiment was selected conditions when packet loss not exceeded more than 5 percent. The results of our experiment are presented on table 2. Figures 2-4 shows change of intensity and average processing time of one packet during experiment for each experimental platform.

Represented graphs show that with growth of intensity, average time of one packet processing is decreasing. Because of that, we separately collected data at time when experimental platforms reached maximal packet generation. These data are represented at table 2.

Time needed OS to process one packet is affected by many factors: launched applications, settings and realization of network stack, schemes of power management and etc. According to that our experiments were made at conditions when experimental platforms didn't do any other tasks. Even in that case during two identical experiments under identical conditions, the results had small differences because of processes inside of computer system and OS.

Table 2: Results of experiments

Platform	Number of trials	$t_{average}$	t_{best}	t_{worst}	Inaccuracy of average
General measures					
EP № 1	10	9,8 µs	2 µs	3279 µs	± 2,8497 µs
EP № 2		10,27 µs	2 µs	3770 µs	± 0,5950 µs
EP № 3		60,53 µs	< 0 µs	2090 µs	± 0,9771 µs
Peak measures					
EP № 1	10	3,83 µs	2 µs	2527 µs	± 0,0721 µs
EP № 2		8,20 µs	2 µs	3770 µs	± 0,0779 µs
EP № 3		17,97 µs	< 0 µs	2030 µs	± 0,2056 µs

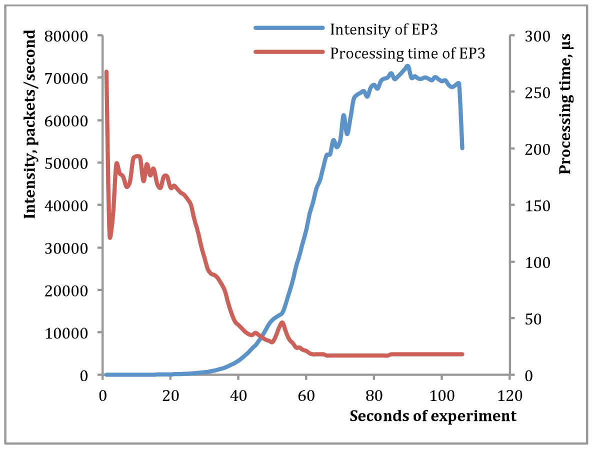
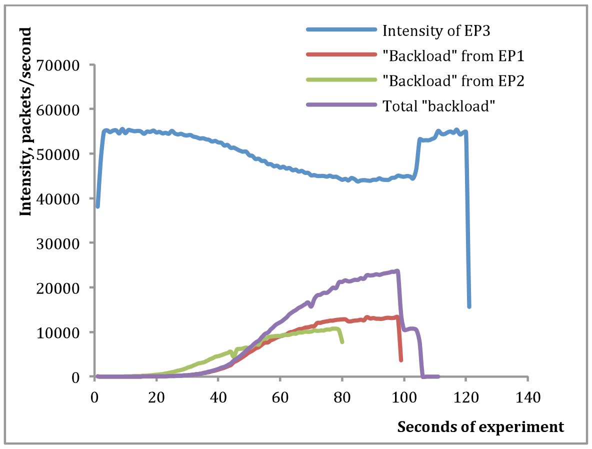


Figure 4. Curves of EP3 intensity and processing time during experiment



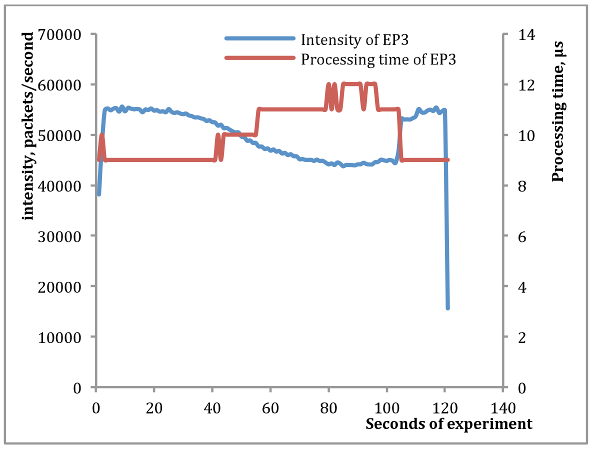
Figures 5. Intensity change during experiment

During load testing, software system under test is forming responses stream. That stream on our work we call a backload. Operating system is processing every response from that stream before it can be analyzed by load testing application. Additionally some network protocols to maintain connections require exchanging of service information, which also consume resources. Centralized design of conventional computer is causing handling of all these tasks by central processor. Therefore backload can affect on load capability of a computer system. We made additional experiments to check this assumption.

For experiment were used same program as before and all platforms were connected to local network. During experiment one of the platform was acting like load injector and other platforms were emulating backload. Experiment consists from next steps:

- Launching experimental program on "load injector" platform.
- Waiting until platform reach maximal intensity.
- Launching experimental program on platforms acting as backload for "load injector" platform.
- When backload reach it maximal intensity, execution of experimental program on "backload" platforms is being stopped.
- Stopping execution of experimental program on "load injector" platform.
- Results analysis.

During the experiment the only one copy of experimental program on each platform was used. On the role of "load injector" was selected experimental platform №3, because program here can produce packets with greater intensity in comparison with other platforms. Results of an experiment represented on figures 5-6.



Figures 6. Processing time change during experiment

Graph shows that with growing of backload, the intensity of "load injector" platform starts to decrease and when backload disappear, intensity restores its previous level. Average time of one packet processing is changing too during backload. On our work the influence of incoming traffic on load capability, we call the backload effect.

Load capability of conventional computer is limited by many internal and external factors. These factors are a reason why resources of a single computer are not enough to produce enough load. To overcome these disadvantages we created other approach.

4. Our approach

At current time FPGA is a flexible and powerful tool for wide range of tasks – from digital signal processing to avionics control [14]. Works of different researchers show that using FPGA for network tasks is also effective[15-17]. That is why instead of additional computers for load creating we propose using of FPGA-based hardware module. Main goals of a module are to create load on software system and gather performance metrics. Test requests created at preparing stages of load test are transmitting on a module. According to requests and additional testing information, module is creating load on software system and gathering performance metrics. Then results of load testing are transmitting back on a computer for further analysis. Our key requirements to that module are:

- High effectiveness of load creation process.
- Load capability shouldn't be affected by incoming traffic.
- Independence from software being tested.
- Gathered performance metrics should be represented in a convenient way for further analysis.

To achieve goals the module should consist of such components as:

- Load creation component.
- Light-weighted network stack to communicate with software system.
- Connections control component.
- Incoming traffic analysis component.
- Components needed for downloading testing preferences and uploading load-testing results.

4.1 Implementation and evaluation.

Based on goals and key requirements we created prototype of load creation hardware module. Prototype was built at board "Altera DE2-115 Development and Education Board" with FPGA Altera Cyclone IV EP4CE115. All key components were implemented on hardware using VHDL language. Prepared requests and testing preferences are transmitting on a board using JTAG UART. Database "Redis" located on dedicated computer are used to store results of load testing. Board and database are connected by separate Ethernet link. Additionally, communication with database required database-client, which was implemented on hardware too.

During load creation process, prototype create great amount of TCP (Transmission Control Protocol) connection with software system and send requests. Each request and responding with time information are transmitting on a database. Packet generation process consists of session information selection from memory, header generation, data preparing and transfer generated packets to Ethernet controller. Thanks to architecture of FPGA, most of packet generation stages can be processed concurrently. Time needed to generate one packet by prototype can be described as:

$$t_{gen} = t_{ss} + t_{hgdp} \qquad (1)$$

where t_{ss} is time of session selection and where is time of header generation and data preparing.

General time for packet generation and its transmission on Ethernet controller can be described as:

$$t_{general} = t_{gen} + \frac{N}{D}T \qquad (2)$$

where N is length of a packet, D is data bus width and T is clock period.

After transmission of a packet, new one is being generated according to TCP connection status. Thus, characteristics of a prototype are depend on performance of software system being tested. For example, before data transmission it is necessary to send connection request and then wait for response. While prototype is waiting for response, t_{ss} is increasing.

To evaluate prototype characteristics we conducted experiments. Additional resources were used to create components for $t_{general}$ measuring of a prototype. These components are calculating number of clocks needed for packet processing and save that information in a memory for each sent packet. Additionally, that memory is storing data about size of each sent packet. Using all gathered data we can calculate t_{gen}. During experiment prototype was sending requests to connected computer approximately 10 seconds. After that, data from prototype memory was read and analyzed.

At the time of experiment prototype generated 108050 packets. Work of a prototype can be divided into 2 stages. At the first stage it is sending a lot connection request packets and at the second prototype is transmitting data packets. Hence first stage is not depending on remote computer and that is why it can show performance of a prototype. Results of that stage is next:

$$t_{gen\,worst} = 340\,ns$$

$$t_{gen\,best} = 220\,ns$$

$$\overline{t_{gen\,average}} = \frac{\sum_{i=1}^{n} t_i}{n} = 220{,}60\,ns$$

Difference between results is caused by waiting for access to sessions memory. While prototype is creating connections, remote computer is responding with confirmation. When prototype is receiving confirmation, it updates information in a session memory. Because of that load creation components in some cases need to wait for access to sessions memory.

Using collected data we can calculate throughput of created prototype:

$$\overline{C} = \frac{1}{\overline{t_{gen\,average}} + \frac{N}{D}T} = 2\,773\,152{,}16 \text{ packets/second}$$

$$C_{worst} = \frac{1}{t_{gen\,worse} + \frac{N}{D}T} = 2\,083\,333,\,33 \text{ packets/s}$$

$$C_{best} = \frac{1}{t_{gen\ best} + \frac{N}{D}T} = 2\ 777\ 777{,}77 \text{ packets/s}$$

These data shows throughput for packets with length 56 bytes, which consist of TCP service information. Results for packets with length 232 bytes consisting of simplest GET requests to webserver is next:

$\overline{C}$ = 1 249 062,45 packets/s

C_{worst}= 1 086 956, 52 packets/s

C_{best}= 1 250 000 packets/s

Dependence between intensity and size of packet is represented on figures 7-8. These results show maximal load capability of a created prototype.

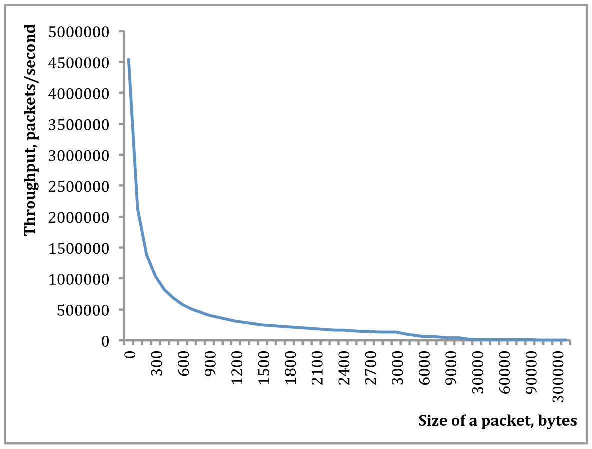


Figure 7. Curve of dependency between throughput and size of a packet expressed in packets per second.

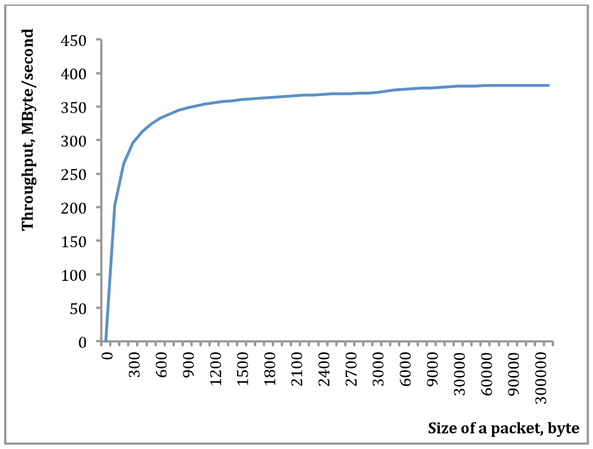


Figure 8. Curve of dependency between throughput and size of a packet expressed in MByte per second.

After evaluation of a prototype we used it to create load on web-server and study how it will work in practice. As systems under load were selected experimental platforms 1 and 2 with installed web-server "nginx". All preferences and program modules of nginx was stayed by default. When server was receiving request it forms standard HTML page. For comparing, the same web-servers were being tested using load-testing tool "Apache JMeter". Empirically "JMeter" parameters were set to create load as big as possible. Server access journal was the source of information about created load. Web-server is putting information about each received request on that journal. Results of prototype assaying represented at figures 9-10.

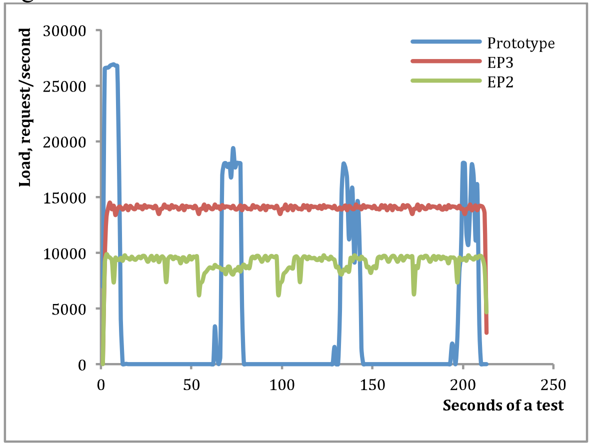


Figure 9. Results of using prototype for load creation on EP1.

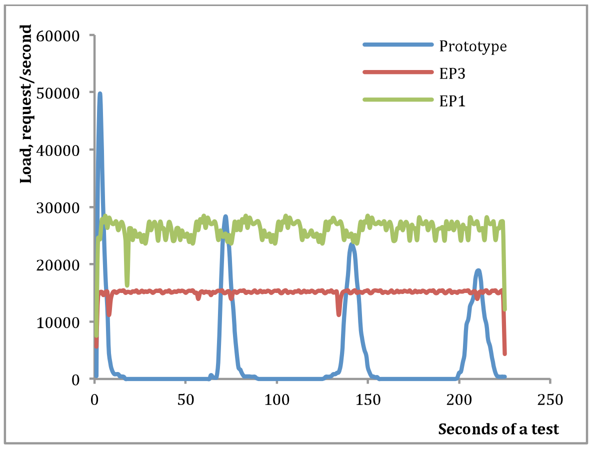


Figure 10. Results of using prototype for load creation on EP2.

Depictured graphs show that prototype during assaying create high load on a short period of time with some periodicity. The following analyzing of data gathered by packet capture program revealed that web-server is not responding on requests. Prototype during load creation process is sending request on a server and waiting for response. Packet capturer data show that computer get request, but web-server is not responding on it. Reason of it is that prototype is trying to create maximum possible load at once without smooth increasing of load. Web-server cannot process that amount of request at time.

Period of time between load spikes is caused by time-out period. Prototype is waiting for responding from server, which is not happening. After time-out period, server is trying to close connection. Prototype react to that, it is reconnecting to server and send request again. Additionally to that server is not accepting all connection because of high intensity. Hence to hold load testing more accurate, load should be created smoothly. Current realization of prototype doesn't have mechanism for that, feature of load controlling will be added in future.

5. Conclusion

This paper presents results of our work on acceleration of load creation process using hardware module with FPGA. Created prototype has high load capability with latency about 220 ns. Process of load creation is fully independent from income traffic, so prototype is not being influenced by back-load effect. Implementation of it requires 17,098 LUTs and 174 memory blocks that utilize 15 percent of Cyclone IV E resources. Despite high load capability, prototype requires additional work before it can be used as full-blown load testing tool. It is necessary to create flexible mechanism for management of load creation process, easy way for uploading test requests on device and additional work on network stack components. Nevertheless created hardware loader shows that using FPGA for load testing task is highly effective.

At current time more papers emerge devoted to using FPGA for application-specific tasks. Tasks that traditionally have been solved using only software. For example, emulation of java runtime machine, DBMS tasks, simulation of different hardware architectures for its evaluation, processing of market data and etc. Constant progress of FPGA technology is causing increasing number of device features, decreasing its costs and simplifying development tools. This, in its turn, leads to the growth of FPGA field of application. For example in future, FPGA can be used for solving operation system tasks thereby leave central processing unit only for applications.

References

[1] J.D. Meier, C. Farre, P. Bansode, S. Barber, D. Rea, Performance Testing Guidance for Web Applications, *Microsoft patterns & practices*. Retrieve from http://msdn.microsoft.com/en-us/library/bb924375.aspx, 2007.

[2] P. Zhang, S. Elbaum, M. B. Dwyer, "Automatic Generation of Load Tests", in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, 2011, pp. 43-52.

[3] P. Zhang, S. Elbaum, M. B. Dwyer, "Compositional load test generation for software pipelines", in *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, 2012, pp. 89-99.

[4] K. Morrison, H. M. Haddad, "Converting users to testers: an alternative approach to load test script creation, parameterization and data correlation", Journal of Computing Sciences in Colleges, 28(2), 2012, pp. 188-196.

[5] D.V. Silakov, "Avtomatizacija testirovanija web-prilozhenij, osnovannyh na skriptovyh jazykah" [Test automation of the web applications based on scripting languages]. *Trudy Instituta sistemnogo programmirovanija RAN*, no. 2, 2008, pp. 159-178.

[6] A. Sortov, A. Horoshilov, "Funkcional'noe testirovanie Web-prilozhenij na osnove tehnologii UniTesK" [Functional testing of web-application using UniTesK technologies]. *Trudy Instituta sistemnogo programmirovanija RAN*, 2004, no. 8, pp. 77-97.

[7] P.N. Jakovenko, A.V. Sapozhnikov, "Infrastruktura testirovanija web-servisov na baze tehnologii TTCN-3 i platformy.NET" [Testing infrastructure of web-services on a base TTCN-3 technology and platform .net]. *Trudy Instituta sistemnogo programmirovanija RAN,* 2009, vol. 17, pp. 63-74.

[8] A.A. Ermykin, "Razrabotka metoda postroenija kompleksa nagruzochnogo testirovanija raspredelennoj informacionnoj sistemy" [Creating of method for developing load testing complex of distributed information system], Ph.D. thesis (Technical Sciences), department of Mechanics and Optics, St. Petersburg National Research University of Information Technologies, St. Petersburg, Russia, 2005.

[9] Y. Cai, J. Grundy, J. Hosking, "Synthesizing client load models for performance engineering via web crawling", in *ASE '07 Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, 2007, pp. 353-362.

[10] Z. M. Jiang, "Automated analysis of load testing results", in *ISSTA '10 Proceedings of the 19th international symposium on Software testing and analysis*, 2010, pp. 143-146.

[11] H. Malik, "A Methodology to Support Load Test Analysis", in *ICSE '10 Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, 2010, pp. 421-424.

[12] D. Serpanos, and T. Wolf, *Architecture of Network Systems*, Burlington : Morgan Kaufmann Publishers, 2011.

[13] *Pcap*, http://en.wikipedia.org/wiki/Pcap, 2014.
[14] *Field-programmable gate array*, http://en.wikipedia.org/wiki/Field-programmable_gate_array, 2014.
[15] G.W. Morris, D.B. Thomas, W. Luk, "FPGA Accelerated Low-Latency Market Data Feed Processing", in *HOTI '09 Proceedings of the 2009 17th IEEE Symposium on High Performance Interconnects*, 2009, pp. 83-89.
[16] N. Weaver, V. Paxson, J.M. Gonzalez, "The shunt: an FPGA-based accelerator for network intrusion prevention", in *Proceeding FPGA '07 Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays*, 2007, pp. 199 – 206.
[17] S. Mühlbach, A. Koch, "A Scalable Multi-FPGA Platform for Complex Networking Applications", in *Proceeding FCCM '11 Proceedings of the 2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines,* 2011, pp. 81-84.

Borodin Anton A. was born in the 19th of September 1988 in Tashkent (Uzbekistan). He received his BS degree and engineer's degree on computer science from Moscow State Forest University (Russia) in 2009 and 2010. At current time he is an assistant at Department of Computer Science on Moscow State Forest University and is working on his Ph.D. thesis. His research interests include embedded systems, FPGA, load testing and computer networks.