EXPERIMENTAL ANALYSIS OF CLIENT/SERVER APPLICATIONS IN EMBEDDED SYSTEMS

Nikolay R. Kakanakov

Department of Computer Systems and Technologies, Technical University of Sofia, branch Plovdiv, 61 "St. Peterburg" Blvd. , 4000 Plovdiv, Bulgaria, phone: +359 32 659758, e-mail: kakanak@tu-plovdiv.bg , web: http://temperature.tu-plovdiv.bg/

Test-bed experimental analysis of the client/server applications in embedded systems is presented. The experiments are executed on different embedded platforms and with different packet sizes. The evaluation of client/server application includes measuring the speed of network hardware and protocol stack processing and the OS speed in creating a socket and reading/writing to it.

The experiments include UDP and TCP client/server applications. They are made using a software tool for test-bed experiments in experimental network [5].

The paper includes a comparison of the embedded systems based on the received results. It examines the dependency of the communication latency on packet sizes for small packets (1-500 bytes).

Using two modes of TCP communication (single transport stream for multiple transfers and new transport stream for every packet) enables distinguishing the socket creation time from socket access time.

Keywords: Distributed Embedded Systems, Web Services, Client/Server model.

1. INTRODUCTION

The embedded systems today undergo a remarkable progress. According to Moore's law the cost of computing and memory continue to go down and it is possible to integrate more functionality in a single chip. This makes using Networked Embedded Systems Technology (NEST) very reasonable and popular [1, 3].

The advance of NEST relies on the integrated networking capabilities of the embedded systems. These capabilities include physical interface, driver programs and integrated TCP/IP protocol stack. There is a big variety networked embedded systems on the market today which differ in communication hardware, protocol stack and software implementation [1, 3].

When developing large systems of distributed nodes it is necessary to examine their communication behavior and programming abilities. The most popular programming model for distributed applications is the Client/Server model. Distributed embedded systems programming use the technology of the general purpose distributed programming. This motivates experiments for evaluation and analysis of the TCP/IP stack performance and Client/Server programming on embedded devices [1, 3].

Networked embedded systems can implement the Client/Server model in different ways. These ways differ on the presence of operating system or virtual machine (like Java virtual machine), programming language used for application development, programming model of the embedded system, implementation of the TCP/IP stack.

In the current paper experiments are made on three different target devices that correspond to three different classes of networked embedded systems. These embedded devices are: Motorola Freescale, Beck IPC@Chip, Dallas DSTINIm400 [2, 4, 7].

Presented experiments are part of framework for evaluating abilities of embedded systems for creating distributed automation applications based on Web Services Architecture. This framework is part of National Science Fund of Bulgaria project – "BY-906", 2005, entitled "Web Services and Data Integration in Distributed Automation and Information Systems in Internet Environment".

2. EXPERIMENTAL DESIGN

The experiments take place in a experimental network built specially for them and using software tool for test-bed experiments of networking in embedded systems.

2.1. Experimental network

The Experimental network consists of Monitoring station, Internet cloud and Local network of embedded devices (Fig. 1).

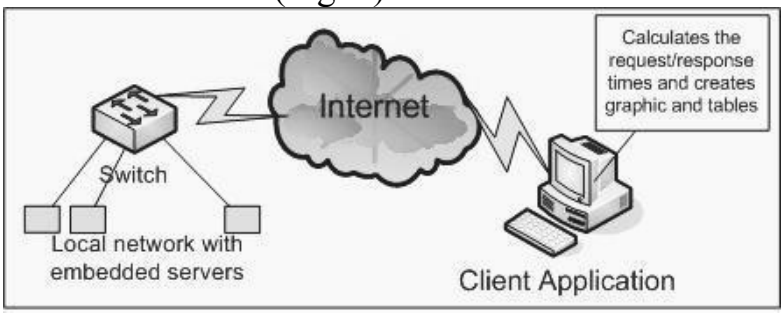

Fig. 1: Architecture of experimental network.

Monitoring station is a PC running Windows 98 with NE2000 network interface card (10/100), Pentium II 333MHz processor. On this station runs the monitoring part of the software tool and packet capturing software.

Internet cloud is represented by a Linux-based router – Pentium 200MHz, 256MB RAM with two NE2000 interface cards, running SmoothWall operating system.

The Local network of embedded devices is built with a 6-port switch and the three target devices. The switch works on 10Mbps because network interface speed of the three target devices is 10Mbps.

2.2. Target embedded devices

Freescale MC9S12NE64 is evaluation platform from Motorola. It has 16-bit HCS12 processor working at 25MHz with integrated Ethernet 10/100 MAC and physical layer. It has not a built-in TCP/IP stack and operating system. Applications are written in C and they use OpenTCP protocol stack. Client/Server programming is based on event listener for events coming on network interface [7].

IPC@Chip DK40 is evaluation platform from Beck Microcontroller. It has 16-bit i80186 compatible processor working at 20MHz. It comes with RTOS, Flash-based file system and TCP/IP stack built in memory. Network interface is 10baseT based.

Programs are written in C for 16-bit DOS (every IDE for DOS programming is acceptable) [4].

DS TINIm400 is Dallas platform for networked embedded solutions based on DS80C400 (8051 like) processor working at 75MHz. It has built-in TCP/IP stack and operating system. The abilities of the platform are divided between TINI OS and TINI java virtual machine (and its classes). Application programs are written in Java and class files are then converted for the TINI JVM [2].

2.3. Software tool for Test-bed experiments of networking in embedded systems

The tool consists of two parts – client and server. The client side is a console application that gets options from command line and outputs results in comma separated file (.csv). The call look like this:

G:\Tool.exe <IP_addr> <port> <repeats> <interval> <data_size> <out> <mode>

First two parameters define the endpoint of the target. The following two define the number of experiments and the interval between the. The <data_size> is the size of the data part in the testing packet (tcp.len).The parameter <out> defines the output file. The <mode> chooses one of three possible modes of experiments. Mode 1 uses single TCP stream two send and receive all the packets exchanged with the target. Mode 2 creates new TCP stream for every send/receive couple. Mode 3 uses UDP datagrams for the exchange [5].

The server side works on the target system and is a simple echo server. It writes back every received packet without any processing on it. For every target system a specific echo server must be developed [5].

The tool calculates the time between the request and the response of the echo server. It starts timer immediately before sending and stops it immediately after receiving the echo [5, 9].

The output file includes the sequence number of experiment, the request/response delta time, the size of data sent, the size of data received and the received data. The last two are controls for checking the correctness of transfer [5].

3. EXPERIMENTAL RESULTS AND ANALYSIS

For each of the target systems the tests are made with an echo server with simple configuration without any further optimizations. The TCP/IP stacks are configured with Nagle algorithm and delayed Acknowledgement turned off [6, 8].

Test-bed experiments include the three possible modes provided by the software tool – single TCP stream communication, multiple TCP streams communication and UDP communication. For the second and third modes experiments are made with different data lengths of the request.

3.1 Comparison of latency of the target systems for three modes of communication.

The results for communication latency estimation for single TCP stream connection for ten consecutive request/response couples with 300 seconds interval between them is shown on Fig.2. For the first couple the latency for DS TINI and

IPC@Chip is higher then the latency for the next couples. The reason is the time needed for the OS to create and initialize the socket. For the Freescale this time is not needed because there is actually no socket creation, only event generation. It can be seen also in the other two modes (Fig.3 and Fig.4).

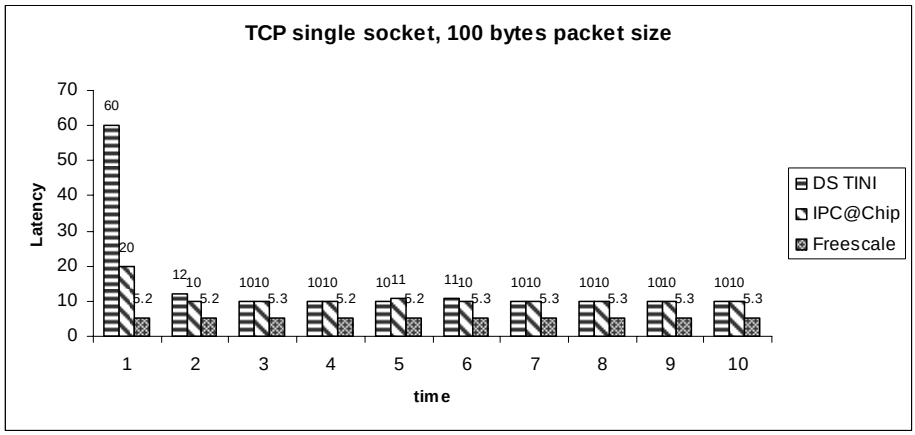

Fig.2

The results of estimation of the latencies in mode 2 are shown on Fig.3. For every request/response couple there is socket creation and it reflects on the latency. It is most notable for the DS TINI platform because of the time needed for switching between TINI JVM and Native Socket Interface.

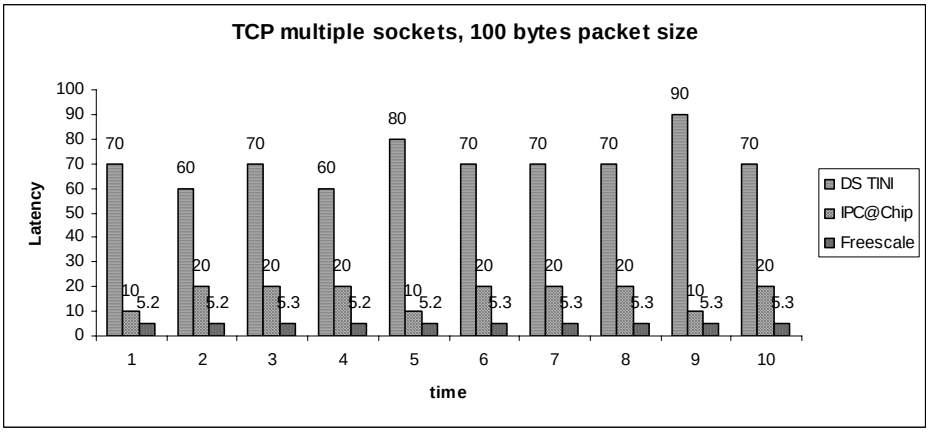

Fig.3

The results for the latency in UDP communication are presented on Fig.4. The time for socket creation and initialization can be seen again for the first couple. More stable latencies can be seen in this mode in comparison with mode 2.

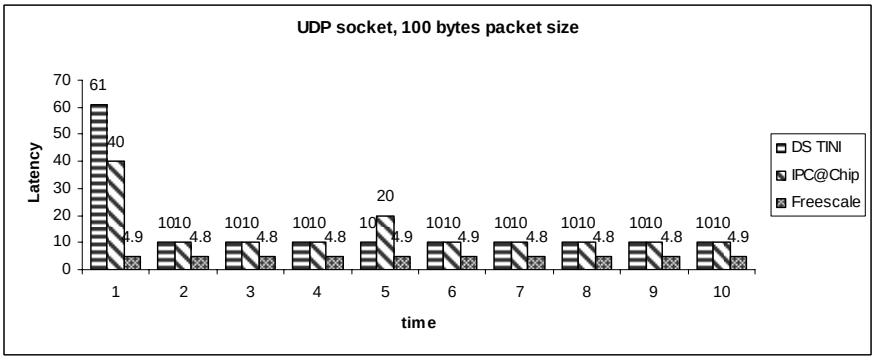

Fig.4

3.2. Examination of the dependency of the latency from data size.

The experiments are made separately for the TCP (Fig.5) and UDP (Fig.6) communication. Data size change in following order 1, 10, 20, 50, 100, 200, 500 bytes per request. For each data size fifty repetitions are made and the average value is presented.

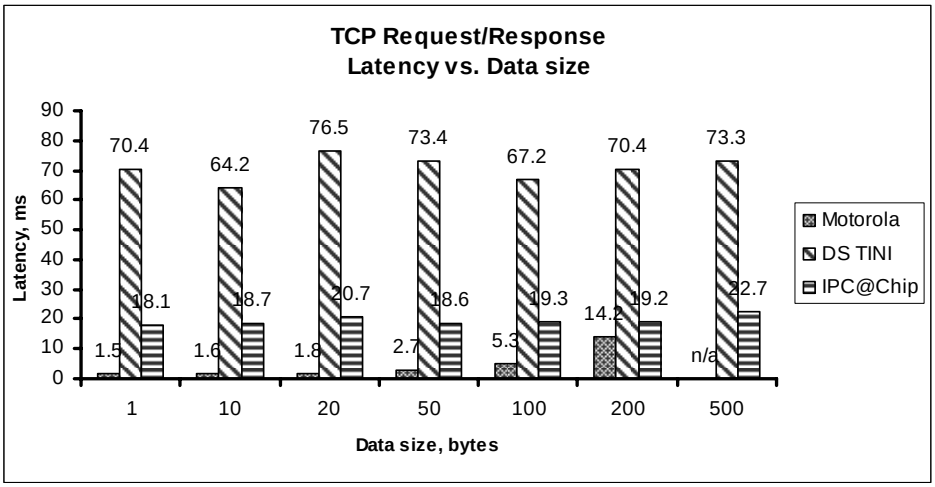

Fig.5

The result for 500 bytes data size for Freescale is not available because it need further configuration of the evaluation board which is not part of the experiments.

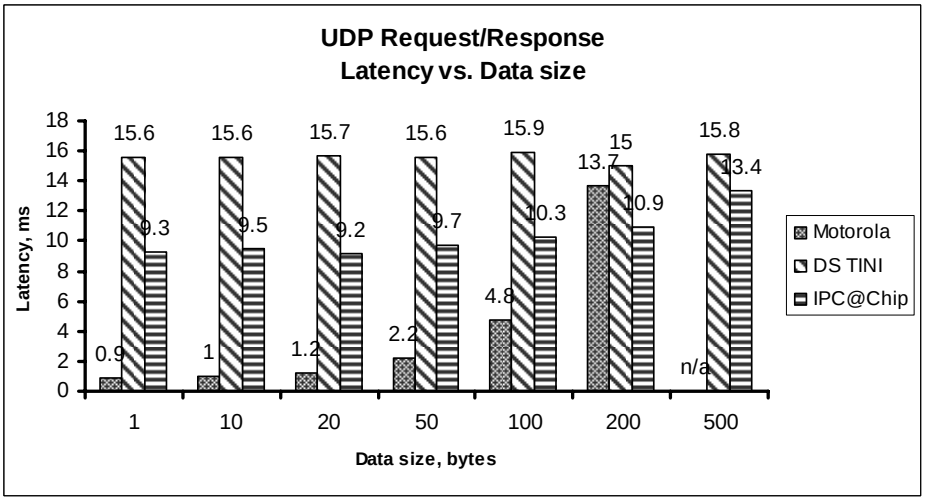

Fig.6

On both presented graphs (Fig.5 and Fig.6) results are similar. For IPC@Chip and Freescale latency grows when data size grow. This growing is more notable for Freescale because for IPC@Chip the most important part of the latency is made up by the socket access time not by data access (OS provided buffered data access). For DS TINI platform latencies are bigger than those for the other two platforms and do not depend on data size. That is probably due to the lack of constant data access time for TINI JVM. This inconstant delay hides the dependence of the latency on data size.

4. DISCUSSION

The presented experiments provide Client/Server communication performance estimation. They show predictability in the communication latency for the Freescale platform and clear border for the variation of the latency for IPC@Chip. On the other side DS TINI platform shows solid instability in this latency. This makes the first platform a good choice when real-time communication is needed, the second is applicable for soft real-time communication and the third is applicable for non real-time (instability and relatively high response times).

The difference in latencies for the three platforms shows the disadvantages of using Java VM and OS in embedded devices. It slows up the applications that are closely related to hardware, like network communication. On the other side the advantages of Java for embedded systems are the ease of application development and abstraction of the platform. The advantages of the RTOS presence can be seen when multiple tasks are running and their synchronization is needed.

5. CONCLUSION AND FUTURE WORK

Presented results provide a view over some communication parameters for embedded devices that work on TCP Client/Server model. They represent the difference of these parameters in different classes of embedded devices – with or without RTOS or JVM. The future work include hardware performance analysis of the target systems and dividing the latencies in connection time, network access time and real transfer time. This division will allow deeper analysis of the communication and will provide clues for optimization and preparation of the systems for real-time communication – a key direction for distributed embedded systems.

6. REFERENCES

[1] Borriello, G., Want, R., (2000) "*Embedded Computation Meets the World Wide Web*", Communications of ACM, Vol. 43 №5, May 2000, pp. 59-66.

[2] Dallas DSTINIm400 homepage – http://www.dalsemi.com/.

[3] Estrin, D., G. Borriello, R. Colwell, J. Fiddler, M. Horowitz, W. Kaiser, N. Leveson, B. Liskov, P. Lucas, D. Maher, P. Mankiewich, R. Taylor, J. Waldo, *Embedded, Everywhere. A Research Agenda for Networked Systems of Embedded Computers*, NAP Washinton, D.C. 2001, ISBN 0-309-07568-8.

[4] IPC@Chip® microcontroller homepage, http://www.beck-ipc.com/.

[5] Kakanakov, N., Spasov, G. *A System for Test-bed Experiments of Networking in Embedded Systems.* Will take part in Proceedings of Computer Science'2005, Technical University of Sofia, Bulgaria, 30 Sept. – 2 Oct, 2005.

[6] Krishnamurthy B., Rexford J., *Web Protocols and Practice. HTTP/1.1, Networking Protocols, Caching and Traffic Measurement*, ADDISON-WESLEY, 2001, ISBN 0-201-71088-9.

[7] Motorola Freescale homepage – http://www.freescale.com/.

[8] Rodriguez, A., Gatrell, J., Karas, J., Peschke, R., *TCP/IP Tutorial and Technical Overview*, IBM Redbooks, www.redbooks.ibm.com/ .

[9] Sridhar, T., *Designing Embedded Communications Software*, CMP Books © 2003, ISBN: 157820125X.

An embedded system is a small or large non-computer device with integrated software based on microcontrollers and microprocessors for performing a dedicated function or a limited set of functions. It may or may not have a screen and a keyboard, be either programmable or non-programmable, perform a single function in isolation, or work as a part of a large system. A TV remote control, a microwave oven, a network of sensors and control systems in automobiles and complicated manufacturing robotic equipment â€" all these devices and electronic systems operate due to embedded software. Common Features of Embedded Systems. Designed to perform specific repeated functions on certain single-purpose devices. The metadata server cluster can expand or contract, and it can rebalance the file system dynamically to distribute data evenly among cluster hosts. This ensures high performance and prevents heavy loads on specific hosts within the cluster. You can integrate Ceph with Hadoop using the Java CephFS Hadoop plugin.Â You run a client program on your computer, which talks to one or more storage servers on other computers. When you tell your client to store a file, it will encrypt that file, encode it into multiple pieces, then spread those pieces out among multiple servers. The pieces are all encrypted and protected against modifications. Experimental analysis of the implementation After a particular protocol is designed and implemented it must be validated. Validation can be done by simulation, test-bed experiments or real-word deployment. Initial validation chosen for CNDEP is test-bed experiments and estimation of request-response times of some of the CNDEP commands.Â [3] Kakanakov, N., Experimental Analysis of Client/Server Applications in Embedded Systems, proceedings of ELECTRONICS'05, 21-23 Sept.2005, Sozopol, Bulgaria, book 4, pp 97-102, ISBN:954-438-520-7. [4] Stallings, W., High-Speed Networks and Internets, 2nd Ed, Prentice Hall, 2002, ISBN: 0-13-032221-0. [5] Topp Topp, U., P. MÃ¼ller. Testing the application and analyzing the results led to the validation of the proposed SCADA system. Keywordsâ€"SCADA systems; middleware; data acquisition; data stream; distributed systems.Â The server module was developed in C++, and the MCIP client was developed as an application in C # (using the .NET platform).Â The second test architecture consists of 6 embedded eBox 2300 SX systems, 1 embedded PDX 089T system with Windows CE real-time operating system, a Master desktop PC (the same as in the first test architecture) and a Super Stack II Baseline 10/100 Switch.Â The analysis of the experimental results chart in Fig. 4 also shows a tendency to decrease the data transfer speed once with the increase of the update rate. 350 300 250 200 150 100.