ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

A Survey of Benchmarking Techniques for Network Performance

A.Selvakumar[*1], Surender Dhanasekaran[2]
[*1,2] Department of Electrical and Electronics, Vel-Tech Dr.RR & Dr.SR Technical University, Chennai-62, India
selva.shivayanama@gmail.com

Abstract

Ethernet implemented on PCI is now the dominant LAN technology in the world. Which is used for communication between computers, networks and industrial applications etc., the protocol is implemented on hardware platforms, operating systems and Ethernet cable. The benchmark testing comprises nodes and data sources, for data exchange among nodes, which measures the performance of PC control system. Data sources and consumers include time synchronization, hardware and software events broadcasting. A single network cable is used for exchange of status and control data among nodes. There is a requirement for analyzing and optimizing the data transfer rate with implementation of the protocol stack on hardware platforms with distances between nodes. To find transfer rate, bandwidth, jitter, Stack latency and round trip time delay. In this project, we are going to find the inter node round trip time bandwidth, stack latency and jitter on both real time and non real time environment and comparing the inter node bandwidth, round trip time, stack latency and jitter of real time environment by using various algorithms and methodologies.

Keywords: Round Trip Time, Latency, Jitter, Bandwidth.

Introduction

PCs were basically designed for applications with non-real time responses. Operating Systems (OS) for PC do not exhibit deterministic response to real time events and data [1]. Most of the common OS are not pre-emptive. However, as per [1], Xenomai kernel for Linux OS is a better preemptive kernel, which provides deterministic real time response. Xenomai is an alternative to the proprietary real time operating system, because it extends GNU/Linux with real time performance [2]. The pre-emptive kernel needs a better support from the system hardware, so that hard-real time interrupts are serviced within a given time-frame. PC based controllers require networked nodes for data acquisition, event monitoring and control.

However, Xenomai is being updated by the open source community. The comparative study [3] of VxWorks, RTAI and Xenomai indicates that performance of Xenomai is acceptable for most applications. It is necessary to apply ADEOS patch for Xenomai. We have used standard open source patches available for Xenomai kernel.

The main difference between RTOS and Non-RTOS is on their scheduling algorithm. Scheduling algorithm in Non- RTOS is based on tasks priority, but tends to take fairness as the priority to reach a high throughput [3]. RTOS utilizes scheduling algorithm which is based on the task priority strictly. No lower-priority task will be proceeding if there's a higher-priority task in queue.

Xenomai. The Xenomai project also uses Adeos as the micro-kernel, however in its standard version. The real-time core of the Xenomai is called Nucleus. It communicates directly with the Hardware Abstraction Layer2 [4]. The real- time tasks — similary to the RTAI — can be developed and executed in the kernel and user spaces. Version 2.3.3 is currently available and it was used for testing purposes. Xenomai supports following hardware architectures:

- x86 and x86 64
- PowerPC and PowerPC 64
- ARM (some subarchitectures)
- IA64
- Analog Devices Blackfin
- MIPS (not distributed yet)
- SH-4/SH-3 (not distributed yet)

Xenomai implements a powerfull and convenient native interface [5]. This interface provides Nucleus functionality to the developer. Its big advantage is that it is used the same way in both

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

kernel and user spaces. New, different interfaces can be easily developed. Many of them have been implemented to simplify a migration from the other real-time operating systems. These different interfaces are called skins. The implemented and supported skins are:

- native
- POSIX
- pSOS+R
- VxWorksR
- VRTX R
- uITRON
- RTAI (only in the kernel space)

The C and C++ (the user space only) languages can be used during development process. Xenomai and RTAI have been once developed as two different projects. Since then they have been merged into a one and then separated again. Nowadays they differ a lot despite common roots and similar architecture. The biggest adventage of Xenomai is the very simple and well defined interface. It is also published with the GPL license. More inforamtion about Xenomai can be found at the developer website [5].

Background

The aim of the reported work has been the evaluation of realtime Linux solutions for a possible replacement of VxWorks, currently in use at RFX-mod. Based on the reported performance measures, we observe the following facts:

The performance of the current Linux 2.6 kernel is very good and may be acceptable in small, dedicated systems. This is however not the case for the feedback control system of RFXmod, where the involved control units need to handle high data throughput in I/0 and network communication.

Both RTAI and Xenomai are worth of consideration. Xenomai proved to be slightly less performing than RTAI, mainly because of its layered approach, which introduces some overhead in interrupt management. On the other hand, Xenomai is better structured and is available for a larger number of platforms. Moreover, Xenomai provides a set of emulation layers which may prove useful when porting large systems.

Compared to VxWorks, both RTAI and Xenomai can be less user friendly for software developers. Since real-time tasks are to be executed in kernel mode in order to achieve best performance, the programmer cannot rely on the system services normally available in user space and debugging becomes very difficult. It is however possible, for both Xenomai and RTAI, to let user processes become real time. Allowing the development of user processes for real-time applications simplifies the development of real-time systems and permits also IPC with standard Linux processes. Real-time user processes are managed by a dedicated scheduler, which works in conjunction with the Linux scheduler by stealing user processes when they request to become realtime. Unlike kernel processes, context switching for user processes requires the remapping of the Page Table, a potentially time consuming operation. For this reason we plan further tests in order to quantify the impact of the MMU remapping in context switching.

The network performance represents a strong point in favour of the migration towards RTAI or Xenomai. UDP has been successfully used for real-time network communication, and RTnet proved to be a very performing solution, especially compared with the poor performance we experienced with the latest version of the VxWorks IP stack for the considered board. The best performance in RTnet is achieved without enabling the TDMA access discipline, which appears to be best suited to systems with a large number of access points (and therefore higher probability of access conflicts) but less stringent timing requirements. This is not the case of the RFX-mod experiment which involves less than 10 control units.

Benchmarking Techniques

There are three metrics selected for benchmarking. Here are the three metrics and how to calculate value of each metric.

A. Processing Time / Latency

Latency in this case more appropriately referred as processing time, which means the time required to process a package since the package is read from client socket buffer to be sent through the socket receiver buffer. Processing time is calculated on the order of nanosecond. Processing time calculation is performed for 10,000 packets per connection.

B. Jitter

Jitter is a variation of the time required to process each received packet. Jitter is obtained by calculating the difference of processing time between current and previous packet. Jitter is calculated on the order of nanosecond. Similar to the calculation of processing time, jitter calculation is also performed for 10,000 packets per connection.

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

C. Throughput

Throughput is calculated by running the program for a period of 10 seconds and then recorded amount of packets successfully processed during that period. To get size of packet/second, then the amount of packets that were successfully processed divided by 10. In the calculation of throughput, we must consider steady state issue. Steady state is the state when program is not affected by initialization and closing program Calculation of throughput is performed by turning on alarm seven times every 10 seconds. Each time the alarm was on, it will record packet_counter value into array of throughput and then reset the value packet_counter. From all of seven values, first and last values were not considered because it is not a steady state. Second to sixth value is recorded as the value of throughput.

D. Round Time Trip

Round-trip time (RTT) is the length of time it takes for a signal to be sent plus the length of time it takes for an acknowledgment of that signal to be received. This time delay therefore consists of the propagation times between the two points of a signal. The round trip time varies from packet to packet

E. Bandwidth

Bandwidth is the data rate supported by a network connection or interface. One most commonly expresses bandwidth in terms of bits per second (bps). The term comes from the field of electrical engineering, where bandwidth represents the total distance or range between the highest and lowest signals on the communication channel (band).

A complete review of benchmarking literature is well beyond the scope of this paper. Here we discuss some directly relevant prior work evaluating Linux and/or Xenomai systems. [5] compares RTAI, VxWorks, Xenomai, and stock Linux performance for use in a nuclear fusion application. They perform three different benchmarks. The first is directly comparable to our responsivity benchmark. The test system is required to change an output DAC line when an input ADC line changes. The metrics are latency from input to output, and latency jitter. The measurements are taken by oscilloscope applied to the lines. All implementations are kernel mode. All code runs on a Motorola MVME5500 (PowerPC architecture). The reported latencies range from 69.2µs (Vx- Works) to 73.2µs (Xenomai); jitter is sub-1µs in all cases. Xenomai is outperformed by the stock Linux kernel (which is, in turn, outperformed by RTAI and VxWorks.) The paper does not report how many measurements were taken for each configuration. The system is unloaded for all reported numerical measurements, although the authors comment that Linux performance measures "hold only for a system which is not loaded, and soon decrease when the workload increases." While the paper does not report how latency and jitter are calculated, the sub-1µs jitter values seem qualitatively different from the variations we observed in testing the BeagleBoard. As shown in Table 4, in our testing, the Xenomai kernel response implementation showed nearly a factor of 4 difference between median response (9µs) and slowest response (37µs.) Furthermore, note that while the Linux kernel does outperform the Xenomai kernel on a 95% basis in our results, the converse is true on a 100%-basis. Based on these distinctions, we suspect that the measurement methodology and sampling duration used in [6] have limited validity in deciding whether any of the measured systems can be used for 100%-hard nuclear fusion control. The second benchmark described in [7] is a similar latency test; however, the input thread notifies a second thread to perform the output write. The additional latency, compared with the first experiment, is determined to be the scheduling overhead, with a maximum of under 6µs on stock Linux. The third experiment involves separating the input and output functions onto separate computers; the input system sends a UDP packet to the output system over gigabit Ethernet. The latency reported ranges from 101µs on RTAI+RTnet to to 157µs on VxWorks. [8] extends and deepens the real-time networking comparisons.This paper compares a Xenomai userspace and an RTLinux Pro kernelspace query/response implementations. It reports that for a 4-byte request/response,

Xenomai has a 61µs latency while RTLinux Pro has a 58µs response. Jitter is not reported. [9] describes common sources of latency in Linux x86 systems. It also includes a number of (selfmeasured) latency results based on outputs to one parallel port pin which is wired to another parallel port pin used for input. Latency is the time from when the system stimulates the output pin to when it handles an incoming interrupt from the input pin. All measurements are strictly reported against stock Linux; however, the computational load and hardware are varied, resulting in dramatically different responsivity histograms. [10], performed at the same institution two years later, is in some ways similar to our present work. It reports (self-measured) responsivity experiments run using a parallel-port loopback, as well as periodic activity tests with

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

internal measurement. It reports results across a kernel and userspace implementations for Linux with various preemption patches, for RTAI, and for Xenomai. In general the trends are as expected. It is noteworthy, however, that CONFIG PREEMPT increases the average latency not just of stock Linux results, but also of Xenomai results; the authors discuss some possible causes. The measurements are taken over periods of 1 minute each, which the authors note is a brief enough period to put measured maximum values into question.

Conclusion and Remarks

The advanced real-time systems will possess capabilities for high-speed data processing and communication which will require very high time bound processing than what is available in state-of-the-art systems. This necessitates the need of improving the real-time Networking mechanisms in RTOS for our future need. To cope with these challenges, Very high performance is necessary at all levels of implementation application level and system level. In this paper we reviewed several Benchmarking techniques for measuring the Network performance of RTOS. It is hoped that by providing insights into the Benchmarking techniques, this paper would help the researches in addressing the implementation challenges of Networking in RTOS for efficient Networking real-time systems of tomorrow.

References

[1] *Barbalace, et al., "Performance Comparison of VxWorks, Linux, RTAI and Xenomai in a Hard Real-Time Application", IEEE TRANS.ON NUCLEAR SCIENCE, VOL.55, NO. 1, FEB.2008*

[2] *Byoung Wook Choi,et al., "Real-time control architecture using Xenomai for intelligent service robots in USN environments", Intel Serv Robotics(2009) 2:139-151 DOI 10.1007/s 11370-009-0040-0*

[3] *F. Leroux et al., "New Developments on Tore Supra Data Acquisition Units", Proceedings of ICALEPCS, Grenoble, France, pp. 922-925, 2011.*

[4] *Adeos Home Page, [Online]. htt://www.adeos.or*

[5] *Xenomai Home Page, [Online].http://www.xenomai.org*

[6] *Xenomai: Real-time framework for linux. http://www.xenomai.org*

[7] *Real-time linux frequently asked questions.https://rt.wiki.kernel.org/index.php/Frequently_Asked_Questions.*

[8] *The real-time linux wiki. https://rt.wiki.kernel.org/index.php/Main_Page.*

[9] *Philippe Gerum. Xenomai – Implementing a RTOS emulation framework on GNU/Linux, April 2004*

[10] *N. Vun, H. F. Hor, and J. W. Chao. Real-time enhancements for embedded linux. In ICPADS '08: Proceedings of the 2008 14th IEEE International Conference on Parallel and Distributed Systems, pages 737–740,Washington, DC, USA, 2008. IEEE Computer Society*