

Editor-in-Chief
Dr. Kouroush Jenab

INTERNATIONAL JOURNAL OF
ENGINEERING (IJE)

ISSN : 1985-2312

Volume 5 ▪ Issue 4 ▪ October 2011
Publication Frequency: 6 Issues / Year



CSC PUBLISHERS
<http://www.cscjournals.org>



Copyrights © 2011 Computer Science Journals. All rights reserved.

INTERNATIONAL JOURNAL OF ENGINEERING (IJE)

VOLUME 5, ISSUE 4 2011

**EDITED BY
DR. NABEEL TAHIR**

ISSN (Online): 1985-2312

International Journal of Engineering is published both in traditional paper form and in Internet. This journal is published at the website <http://www.cscjournals.org>, maintained by Computer Science Journals (CSC Journals), Malaysia.

IJE Journal is a part of CSC Publishers

Computer Science Journals

<http://www.cscjournals.org>

INTERNATIONAL JOURNAL OF ENGINEERING (IJE)

Book: Volume 5, Issue 4, October 2011

Publishing Date: 05-10-2011

ISSN (Online): 1985-2312

This work is subjected to copyright. All rights are reserved whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication of parts thereof is permitted only under the provision of the copyright law 1965, in its current version, and permission of use must always be obtained from CSC Publishers.

IJE Journal is a part of CSC Publishers

<http://www.cscjournals.org>

© IJE Journal

Published in Malaysia

Typesetting: Camera-ready by author, data conversion by CSC Publishing Services – CSC Journals, Malaysia

CSC Publishers, 2011

EDITORIAL PREFACE

This is the third issue of volume five of International Journal of Engineering (IJE). The Journal is published bi-monthly, with papers being peer reviewed to high international standards. The International Journal of Engineering is not limited to a specific aspect of engineering but it is devoted to the publication of high quality papers on all division of engineering in general. IJE intends to disseminate knowledge in the various disciplines of the engineering field from theoretical, practical and analytical research to physical implications and theoretical or quantitative discussion intended for academic and industrial progress. In order to position IJE as one of the good journal on engineering sciences, a group of highly valuable scholars are serving on the editorial board. The International Editorial Board ensures that significant developments in engineering from around the world are reflected in the Journal. Some important topics covers by journal are nuclear engineering, mechanical engineering, computer engineering, electrical engineering, civil & structural engineering etc.

The initial efforts helped to shape the editorial policy and to sharpen the focus of the journal. Starting with volume 5, 2011, IJE appears in more focused issues. Besides normal publications, IJE intend to organized special issues on more focused topics. Each special issue will have a designated editor (editors) – either member of the editorial board or another recognized specialist in the respective field.

The coverage of the journal includes all new theoretical and experimental findings in the fields of engineering which enhance the knowledge of scientist, industrials, researchers and all those persons who are coupled with engineering field. IJE objective is to publish articles that are not only technically proficient but also contains information and ideas of fresh interest for International readership. IJE aims to handle submissions courteously and promptly. IJE objectives are to promote and extend the use of all methods in the principal disciplines of Engineering.

IJE editors understand that how much it is important for authors and researchers to have their work published with a minimum delay after submission of their papers. They also strongly believe that the direct communication between the editors and authors are important for the welfare, quality and wellbeing of the Journal and its readers. Therefore, all activities from paper submission to paper publication are controlled through electronic systems that include electronic submission, editorial panel and review system that ensures rapid decision with least delays in the publication processes.

To build its international reputation, we are disseminating the publication information through Google Books, Google Scholar, Directory of Open Access Journals (DOAJ), Open J Gate, ScientificCommons, Docstoc and many more. Our International Editors are working on establishing ISI listing and a good impact factor for IJE. We would like to remind you that the success of our journal depends directly on the number of quality articles submitted for review. Accordingly, we would like to request your participation by submitting quality manuscripts for review and encouraging your colleagues to submit quality manuscripts for review. One of the great benefits we can provide to our prospective authors is the mentoring nature of our review process. IJE provides authors with high quality, helpful reviews that are shaped to assist authors in improving their manuscripts.

Editorial Board Members

International Journal of Engineering (IJE)

EDITORIAL BOARD

Editor-in-Chief (EiC)

Dr. Kouroush Jenab
Ryerson University (Canada)

ASSOCIATE EDITORS (AEiCs)

Professor. Ernest Baafi
University of Wollongong
Australia

Dr. Tarek M. Sobh
University of Bridgeport
United States of America

Professor. Ziad Saghir
Ryerson University
Canada

Professor. Ridha Gharbi
Kuwait University
Kuwait

Professor. Mojtaba Azhari
Isfahan University of Technology
Iran

Dr. Cheng-Xian (Charlie) Lin
University of Tennessee
United States of America

EDITORIAL BOARD MEMBERS (EBMs)

Dr. Dhanapal Durai Dominic P
Universiti Teknologi Petronas
Malaysia

Professor. Jing Zhang
University of Alaska Fairbanks
United States of America

Dr. Tao Chen
Nanyang Technological University
Singapore

Dr. Oscar Hui

University of Hong Kong
Hong Kong

Professor. Sasikumaran Sreedharan

King Khalid University
Saudi Arabia

Assistant Professor. Javad Nematian

University of Tabriz Iran

Dr. Bonny Banerjee

Senior Scientist at Audigence
United States of America

AssociateProfessor. Khalifa Saif Al-Jabri

Sultan Qaboos University
Oman

Dr. Alireza Bahadori

Curtin University
Australia

Dr Guoxiang Liu

University of North Dakota
United States of America

Dr Rosli

Universiti Tun Hussein Onn
Malaysia

Professor Dr. Pukhraj Vaya

Amrita Vishwa Vidyapeetham
India

Associate Professor Aidy Ali

Universiti Putra Malaysia
Malaysia

TABLE OF CONTENTS

Volume 5, Issue 4, October 2011

Pages

- 277 - 291 Analysis of Practicality and Performance Evaluation for Monolithic Kernel
and Micro-Kernel Operating Systems
Hui Miao
- 292- 301 Mutual Authentication Between base and Subscriber Station Can Improve the
Security of IEEE 802.16 Wimax Network.
Mohammad Zavid Parvez, Mohammad Hossain, Mohammad Hamidul Islam
- 302 - 312 Design Model-free Fuzzy Sliding Mode Control of Internal Combustion Engine
Farzin Piltan, N. Sulaiman, Payman Ferdosali, Iraj Assadi Talooki

Analysis of Practicality and Performance Evaluation for Monolithic Kernel and Micro-Kernel Operating Systems

Hui Miao

hui.miao@microchip.com

Microchip Australia Design Centre

Microchip Technology Inc.

Brisbane, 4108, Australia

Abstract

The microkernel system (as opposite to monolithic systems) has been developed for several years, with the hope that microkernels could solve the problems of other operating systems. However, the evolution of the microkernel systems did not go as many people expected. Because of faultinesses of the design in system structure, the performance of the first generation of microkernel operating systems was disappointing. The overhead of the system was too high to bear for users. However, the second-generation microkernel system uses an improved design architecture that could substantially reduce the overhead in previous microkernel systems.

This project evaluates the system performance of the MINIX3.1.2a and compares the results with the performance of Linux by using Unixbench system evaluating tool. By this way, it could testify whether the microkernel systems could be more flexible, portable and secure than monolithic operating systems. Unixbench could give sufficient statistics on different capacities of MINIX3 and Linux, such as system call overhead, pipe throughput, arithmetic test and so on. The result illustrates MINIX3 has better performance on Shell Scripts running and Arithmetic test and Linux has better performance on other aspects such as system call overhead, process creation and so on. Furthermore, we provide a more detailed analyse on the microkernel Minix 3 system and propose a method that could improve the performance of the MINIX3 system.

Keywords: Monolithic System, Microkernel System, Operating System

1. INTRODUCTION

Kernel controls the critical parts of operating systems. Nowadays, many current operating systems are monolithic kernel operating systems (e.g. Linux). Monolithic kernel operating systems implement most system functionalities such as file management, device drivers, process management and I/O management in kernel mode. Although monolithic kernel operating systems are very popular, they may have some disadvantages. First, the kernel is intensively complex. A kernel with thousands lines of code could be hard and difficult to maintain. Updating one part of the system may result in needing to recompile the whole kernel. Second, a large amount of code means that the operating system could not be ported to different hardware, especially for embedded systems. Third, the monolithic operating system is not reliable; since the kernel's complexity, the possibility of a system crash could be high. A single tiny error in the kernel could lead the whole system to crash. So microkernel operating systems are designed to overcome the disadvantages of the monolithic systems.

The microkernel system excludes several services out of the kernel. One service can run as a user level application out of the kernel. For example, Mach [1] uses an external pager and the file system can also be running out of the kernel. Minix3 is another microkernel-based operating system which is an earlier version of OS inspired the invention of Linux. The kernel of the latest version of Minix3 only has 4000 lines of code [6], much smaller than Linux. The L4 kernel is a developing microkernel system; the latest version of L4 kernel is L4ka::Pistachio 0.4, which can run on a wide variety of hardware [6].

Compared with macrokernel systems, the microkernel approach has following advantages: First, a microkernel system can provide higher reliability than a monolithic system. A microkernel system has much less chance to crash than a system with a huge kernel. Reducing the size of a kernel is a strategy to reduce the problems in the system. Second, a microkernel system with less kernel code could be maintained more easily. Recompiling the kernel is not a huge task for microkernel systems. Last, a microkernel system could be easily ported to simple hardware, especially in embedded systems.

2. RELATED WORKS AND MOTIVATIONS

2.1 First Generation Microkernels

The first pioneering microkernel conception was in Carnegie-Mellon University, the microkernel Mach operating system. Mach minimizes the kernel into a very small module. The kernel of Mach only provides process management; thread management, IPC and I/O service. The file management, which traditionally is in the kernel, is placed out of the kernel. Mach's external pager [1] was the first conceptual breakthrough toward real microkernel. The conceptual foundation of the external pager is that the kernel manages physical and virtual memory, yet the pager is outside of the kernel. As Fig. 1 shows: if a page fault occurs in user applications it will forward the faults to the pager by message passing. The message is handled by the kernel. This technique permits the mapping of files and databases into user address spaces without having to integrate the file/database systems into the kernel [1].

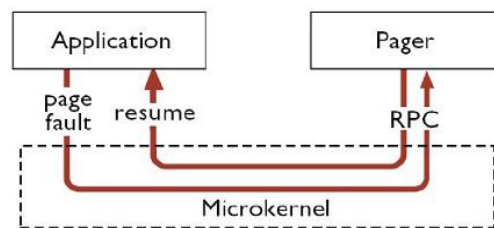


FIGURE 1: Page Fault Processing [1]

The prospect of Mach seems splendid. But it was not as ideal as people expected. The Mach system makes the file system run as a user processes on top of the kernel and uses interprocess communication (IPC) to control this module. IPC contributes a huge overhead to the whole operating system. System calls of traditional operating systems use traps, which are much faster than IPC. Mach needs to create messages, send and switch between processes. As Fig. 2 shows, the overhead is excessive. Chen and Bershada [2] compared applications under Ultrix (a Unix based operating system) and Mach on a DECStation 5-200/200 and found peak degradations of up to 66% on Mach (compared to Ultrix); 66% is really unbearable for user. 75% of the low efficiency is related to IPC. So this first generation microkernel system failed. The Mach system project was abandoned by Carnegie-Mellon University team in 1994.

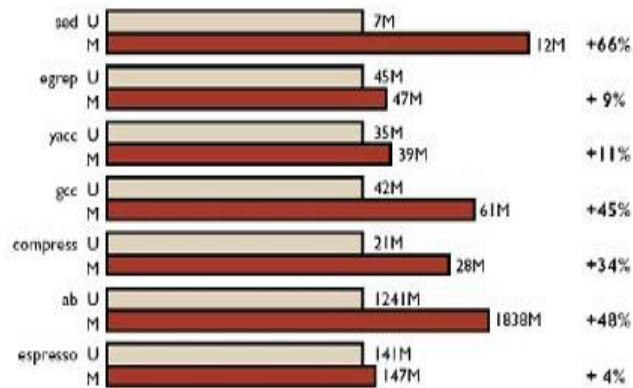


FIGURE 2: Non-idle cycles under Ultrix and Mach [1]

In the case of the failure of first generation, people put forward a compromised way of designing the microkernel system. The main idea is to expand the kernel, put the some file services and device drivers back into the kernel again. In this way, they could reduce the switch time between the user space and kernel. The Chorus operating system [1] uses this design idea. However, the idea of expanding the kernel impairs the original intention of building a small high integrity microkernel. It reduces the extensibility, flexibility, portability and reliability of the operating system. So L4ka appeared.

2.2 Second Generation Microkernels

After the failure of the Mach operating system, scientists began to redesign the structure of the kernel. Prof Dr. Jochen Liedtke invented his first microkernel with low overhead in passing messages, L3. The L3 kernel directly passes the message between processes leaving the process security and authentication to user space servers. This design method greatly reduced massive IPC overhead which occurred in Mach system. On the same system where Mach required 114 microseconds for even the smallest of messages, L3 could send the same message for less than 10. The overall time for a system call was less than half the time on Unix, as opposed to Mach where the same system call took five times longer than that of Unix [3].

After successfully implementing L3, Liedtke designed L3 more comprehensively. The result was a more flexible kernel, L4. A basic idea of L4 is to support recursive construction of address spaces by user-level servers outside the kernel. The kernel only does three address space operations: Grant, Mapping and Unmapping. Liedtke's design was as follows [3]: the owner of an address space can assign any of its pages to another space, provided that the recipient agrees. Similarly, the owner of an address space can map any of its pages into another address space, provided the recipient agrees. The owner of an address space can also flush any of its pages. The flushed page remains accessible in the flusher's address space, but is removed from all other address spaces which had received the page directly or indirectly from the flusher.

The Mach microkernel had a limitation of implementing the external pager policy outside the kernel. And now, this limitation is largely removed by L4's address space concept. This mechanism implements some protection schemes and physical memory management on the top of the kernel. Grant and map operations need IPC, since they require an agreement between granter/mapper and recipient of the mapping. So cross-address-space communication, also called inter-process communication (IPC), must be supported by the microkernel, which gives extra overhead to Mach system. L4 uses many methods and techniques to reduce the IPC overhead. Liedtke improved the performance of the system and reduced the overhead of IPC by redesigning the kernel. The result is positive. One RPC cost L4 only 10 μ s, in contrast to 230 μ s in

Mach and 20 μ s in Unix. IPC is not a burden to L4 any more. Tests of a Linux kernel ported to run on top of L4 and another ported to run on Mach (MkLinux) and the basic Linux system itself showed clear performance gains with L4. Even in the best case MkLinux was 15% slower than the monolithic kernel, whereas L4 was about 5-10% slower [16].

Minix3 is another second generation microkernel system which is developed by Andrew Tanenbaum. Minix3 was redesigned to be a pure microkernel system from its early version Minix2. Like other microkernel systems, device drivers, process manager, file system and memory manager are all implemented outside the kernel. Recently, there is another radical idea of designing a kernel. In the exokernel concept [7], the operating system only provides manipulating the raw hardware. The kernel only takes charge of securing the hardware and controls it. The application-level libraries and servers can directly implement traditional operating system abstractions. There are already some exokernel operating systems in experimental stage, such as XOK that is implemented by MIT research group and also Nemesis, written by University of Cambridge, University of Glasgow, Citrix Systems, and the Swedish Institute of Computer Science.

2.3 Motivation and Project Aims

The Mach operating system is considered to have poor performance by many people, because of the overheads in IPC. There are many kinds of other microkernel systems that are claimed to boast better performance such as L4ka and Minix3. The project's topic is to use several user application tools such as Unixbench which run on different operating systems to evaluate their performance and to discuss the practicality of new generation microkernel operating systems. The project also uses monolithic operating systems for comparison, because most current operating systems are based on monolithic kernel. Linux is a typical monolithic operating system we can use. Comparing the test results of microkernel-based systems to the test results of Linux could be a good source of discussing the practicality of microkernel operating system.

The comparison has been done by using the benchmark tool Unixbench. Minix3, a pure microkernel operating systems has been used for testing. Minix3 is a developing microkernel system, which could be a good option, and also Minix3 supports POSIX [6]. It is well developed and it can be installed easily. Minix3 has C compiler and Shell like Unix. Therefore, it is easier to implement and run application test programs on Minix3 than implementing test programs on L4. Furthermore Minix3 is written by C and Minix3 is a clearly structured microkernel system, so we could clearly know how microkernel system works after reading the source code of Minix3.

This research is designed to compare and evaluate the performance of two different operating systems by using OS benchmark tool Unixbench. There are many benchmarks Unixbench can provide, such as system call overheads, context switch overhead, file read/write throughput, pipe throughput, arithmetic performance and so on. Even more accurate evaluations were done by writing test programs which could test single IPC time or single system call time. We could analyze the result and discuss the practicality of microkernel from the evaluation results.

Linux and Minix3 were used in the testing. Minix3 is a mature microkernel based operating system. It is well structured and could easily be installed. Also Minix3 supports POSIX. It is Unix-like operating system. Linux is currently one of the most popular monolithic operating systems; it is open source and also supports POSIX. Linux is used widely in many fields. So Linux is a good option for testing. L4Linux is an operating system which runs with the L4 microkernel on the bottom level, and with Unix-like application runs on top of L4Linux. The reason not to use L4Linux is there are few research documents available for L4Linux. It is hard to get started and installed. Besides that, because the Linux kernel is in the middle between application level and L4 microkernel in L4Linux architecture, the result may not be accurate. The user application program is not implemented directly on L4 microkernel. So accurate results required implementing Unix-

like user application level directly on the L4 microkernel. It is required to work at all levels of abstraction from the bare machine to the application layer, which is a big challenge for researchers. Also, the exokernel is now in the experimental stage, and there are not sufficient documents and resources on the exokernel, and therefore the exokernel is not included in testing.

Second generation microkernels like L4ka had proved that the microkernel system could perform as well as monolithic kernel system. L4Linux on AIM benchmarks report a maximum throughput which is only 5% lower than that of native Linux. However, it is hard to compare pure L4 system with Linux, because it is difficult to implement user level application on L4 kernel. For native Linux, AIM measures a maximum load of 130 jobs per minute. L4Linux achieves 123 jobs per minute, 95% of native Linux. The corresponding numbers for user-mode L4Linux are 81 jobs per minute, 62% of native Linux, and 95 (73%) for the inkernel version. Averaged over all loads, L4Linux is 8.3% slower than native Linux, and 6.8% slower at the maximum load. This is consistent with the 6-7% we measured for recompiling Linux [4]. L4 is designed for optimizing the IPC overhead and context switch between processes, so the user level application implementation is poor. The performance of operating system is not only the kernel performance, but also the application layer performance, which is directly to users. So evaluate the performance of a relatively mature microkernel system is meaningful to microkernel system.

The project designed does not only to make evaluation benchmarks for microkernel system, but also would like to analyze the benchmarks of microkernel system and compare it to monolithic kernel systems. By that, we would like to outline a much clearer performance figure of microkernel system such as Minix3. From previous papers, IPC overhead was complained most in microkernel system, which was seen a biggest flaw affecting microkernel system performance. However, we believe not all the performance differences in microkernel system are due to the heavy IPC overhead. There are many different ways in implementing kernel and user layer between Minix3 and Linux, therefore it is important to find out which part of benchmark differences are due to the different system implementation. From analyzing the benchmarks, we try to separate performance results that are caused by different system implementation from the results that are inherited due to the heavy IPC overhead. The discuss on performance evaluating results are meaningful, because it makes a scrutiny figure on Minix3 microkernel performance and could give advices that which part of system could be improved by tuning the microkernel system.

3. Evaluation Environment and Equipments

3.1 Hardware Environment

The result of the test has to be accurate and correct. The selection of hardware is important. The entire test has to work on the same hardware, thus the hardware selected must be a common one which is supported by all the system kernels (Minix3, Linux). Minix3 supports many kinds of hardware: 386, 486 and Pentium and so on. To install Minix3 requires: Intel 386 or higher with 4 MB of RAM, an IDE hard disk with 100 MB of free disk space and an IDE CD-ROM for booting [5]. Linux also could support IA32 (Pentium). So a computer with Pentium or higher is a good choice. The RAM has to be 256MB or higher. PC must have IDE CD-ROM, VESA compatible VGA, PS/2 keyboard and PS/2 Mouse.

The configurations of the hardware machines are listed as the following:

Central Process: Pentium □ with 800MHz speed;

Random Access Memory: 256MB RAM;

Hard Disk: IBM DTLA-307020 20GB ATA hard disk;

CD-ROM: 24X IDE CD-ROM

Accessories: VESA compatible VGA, PS/2 keyboard and PS/2 Mouse

3.2 Minix3 Environment

The version of Minix3 used for evaluating is 3.1.2a, which is claimed to be a stable version of the Minix3 operating system [9]. To install, download the compressed CD image of Minix3.1.2a from official server in Minix3 home website, then decompress CD image and burn it into a writable CD. After that, boot computer from CD-ROM. Set Minix3 boot in regular sequence, because we have more than 16MB RAM. The following are the configurations of Minix3 in installation:

Keyboard Standard: US-Keyboard Standard;

Ethernet Chip: None;

Full distribution: Yes (requires 1GB space);

Size of "/home" directory: 2GB;

Data Block Size: 4-KB per block;

For easier implementing testing programs and benchmark tools on Minix3, extra software packages should be installed on Minix3 system. These software packages can be downloaded from the Minix3 home website or installed directly from Minix3.1.2a installation CD. Because Linux uses the compiler GCC to compile test programs and the benchmark tool, the GCC software package was installed in Minix3. Moreover, because Linux uses the Bash shell to execute programs, the shell Minix3 used should be identical with Bash. Consequently, the Bash 3.0 software package was also installed in Minix3 system. If Minix3 used its compiler CC compiler to compile programs and the default shell ash to execute test programs, it will make an inaccurate performance benchmarks. Linux uses compiler GCC to compile programs and uses Bash shell to execute test programs. There are sufficient hard disk spaces for storing, so both the package software binary distributions and their source codes are install in Minix3.

Minix3 has a version of the X window software package (X11 R6.8.2) which provides a window display for Minix3. However, the Minix3 X window system was not installed in Minix3 in the testing, because the X windows software is not a crucial part for system performance evaluating. Furthermore, due to the way Minix3 memory management works, running X window could lead a program to fail because it runs out of memory. "chmem" command should be used to provide sufficient stack space for the program. The memory of X window binary usually set to a very large number, which often could result in X window not starting. The hardware has 256MB memory which is not sufficient for running the X windows as default setting. So "chmem" should be used for giving the sufficient stack spaces. The higher memory X window consumed the less free memory will be available for other application programs [10]. That means the system performance of Minix3 could be greatly deteriorated if running X window software on the system.

3.3 Linux Environment

Fedora core 6.0 and FreeBSD 6.0 were used in performance evaluation at the initial stage of the research. The ISO images of Fedora core and FreeBSD could be downloaded from AARNet. Fedora core is an RPM-based Linux distribution. It is well developed and widely used around world, which is a typical monolithic kernel operating system and POSIX-compatible with 7000 software packages. Therefore, Fedora core 6.0 is a suitable Linux distribution system to be used for system evaluation. The Fedora core involved in research was installed with X window software.

FreeBSD is also an Unix-like operating system. It is similar to Linux and also is a typical monolithic kernel operating system. Many software packages are identical with those of Linux. FreeBSD is totally free for the user, and it also provides binary compatibility with other Unix-like operating systems, including Linux, which means programs running on Linux could also run well on FreeBSD without any modifications. It is as reliable and robust as Linux. In this research, FreeBSD 6 was installed for testing.

At the initial stage of the research, Linux, FreeBSD and Minix3.1.2a were used in the system performance evaluation. As the research went along, we found the performance results were very similar between Linux and FreeBSD. There was only 5%-10% difference between Linux and FreeBSD in system call overhead. There is about a 5% difference between Linux and FreeBSD on pipe throughput overhead. The shell and software packages are almost the same in Linux and FreeBSD. At the middle stage of the project, we decided to stop evaluate the performance of FreeBSD. The following reasons are why we did that:

1. The purpose of the project is to obtain benchmarks of microkernel system and monolithic kernel system and discuss the practicality of microkernel system to see whether it could be more reliable and sophisticate as well as current monolithic kernel system. Performance comparison between Linux and FreeBSD does not make any sense for the intention of the project. Furthermore, there are already many benchmarks and benchmark tools for the performance evaluation between those Unix-like monolithic kernel systems. Many documents about benchmarks and performance evaluations could be found in the Internet.

2. The performance evaluation results are very similar in Linux and FreeBSD. There is only 5%-10% difference between Linux and FreeBSD in system call overhead. It is about 5% difference between Linux and FreeBSD on pipe throughput overhead. The structure of the kernels in Linux and FreeBSD are similar. Both of the FreeBSD and Linux are designed as monolithic kernel system, so performance measurement between two monolithic kernel systems certainly will give a similar result.

3.4 Benchmark Tool

The selection criteria of benchmark tool were not complex. The benchmark tool should be able to run correctly on Minix3 and Linux, and gives accurate benchmarks for the system. After carefully reviewing through benchmark tools such as LMBench, Unixbench and Ubench, we finally decided to use Unixbench 4.0.1 [11] to test the system performance of Minix3 and Linux. The reason not use LMBench is that LMBench requires specific header files at time of configuration. The header files only could be found in Linux system or other mature Unix-like system. Minix3 does not have these header files; therefore it will cause compiling failure during install LMBench on Minix3.

Unixbench is another system benchmark tool like LMBench. Unixbench gives performance benchmarks on many aspects of operating systems. Unixbench is a simple portable and POSIX microbenchmarks tool. Unixbench can give operating system benchmarks such as Dhrystone, system call overhead, file system performance on Write/Read/Copy, pipe throughput, context switch, shell script running, arithmetic test, compiler performance and so on. Thus, Unixbench can give a comprehensive figure of system performance on Minix3 and Linux. The main idea Unixbench use to evaluate performance of operating system as follows: On each system, a constant running time is given. Then, an infinite loop running test program is started, and a global variable is used to record the number of loops. When the time is up, a signal interrupts the loop and records how many times the test program runs. Obviously, an operating system which has higher efficiency could run more loops than that with lower performance. The more loops run the better system performed. For example, we use Unixbench to evaluate process creation on Linux:

1. At start, the test program is set to run 10 seconds. "signal(SIGALRM, func)" is used for setting an interrupt function handler. "alarm(10)" will sign a signal after 10 seconds.
2. Use "while (1)" to run an infinite loop. In the loop, "fork()" and "wait(&status)" are used for creating process, and "iter++" counts the times program run during 10 seconds.
3. After 10 seconds, the infinite loop is interrupted by a signal sent by the "alarm(10)" system call. In the end, record the value of variable "iter". The bigger "iter" is the better system performs in process creation.

The Unixbench only gives the number of loops run to present the performance of the system. What if people want more direct performance benchmarks? For example: microsecond time values on each test program run are wanted. Therefore, in this project we also used another way to measure the performance of Minix3 and Linux. The system call “gettimeofday()” returns the time in seconds and microseconds since epoch in GMT. So we could run “gettimeofday()” system call at the beginning and the end of test program and subtract the returned values in order to get the microseconds used in running test program. Also, taking the process creation test as an example, the following code determines the execution time of one process creation:

```
gettimeofday(time, tzzone); /*get the time befor running */
    if ((slave = fork()) == 0) {
        exit(0);
    } else if (slave < 0) {
        exit(2);
    } else
        wait(&status);
    if (status != 0) {
        exit(2);
    }
gettimeofday(time1, tzzone1); /*record finish time */
```

From the value of structure “time1” and “time” we could calculate the execution time of process creation for once.

4. Evaluation Results and Analysis

4.1 Unixbench Evaluation Benchmarks

At the beginning of the evaluation, we installed Unixbench on Linux and Minix3 separately first. First problem was how to transfer data into Minix3. Minix3 was not developed as well as Windows and Linux. Linux could automatically mount and unmount many file devices such as USB and CD-ROM. With X window, Linux could easily transfer Unixbench program from mobile devices (USB, CD) to hard disk. However, Minix3 could not obtain data sophisticatedly from outside devices. MINIX's primary purpose is to illustrate operating system principles. Keeping MINIX small enough to fit into a student's head during a semester- or year-long course has required keeping it simple. In particular, the MINIX file system supports mounting only media containing MINIX file systems [9]. In this research, we used command “isoread” to read the content of Unixbench from ISO-9660 CD-ROM and copied the content to local hard disk. We wrote Unixbench install program into a writable CD and use “isoread /dev/c0d2/ Unixbench.zip > /home/Unixbench/Unixbench.zip” command to copy Unixbench.zip into /home/unixbench/ directory. After that, to use GCC, we have to change the PATH to add “/usr/gnu/bin” into PATH of Minix3 system. At last, using “make” command to compile and link all the programs of Unixbench then we used “./run” to run Unixbench to get benchmarks.

The total evaluation procedure lasted for 52 minutes. Unixbench gave system evaluation benchmarks on Dhrystone, system call overhead, file system performance on Write/Read/Copy, pipe throughput, context switch, shell script running (with 8 and 16 concurrent users), arithmetic test, C compiler throughput and process creation. Because Linux uses Bash shell, which is different from Minix3's ash shell, so we installed Bash 3.0 shell on Minix3 and run Unixbench on Minix3 with Bash 3.0. The following table Table 1 in next page are benchmarks of Unixbench on Minix3 with Bash 3.0, Minix3 and Linux. Here are the terminologies for the table:

lps: loops per second
lmp: loops per minute

KBps: Kbytes per second

From table.1, we could find that performance of Minix3 with Bash shell is similar to Minix3 with the default shell. Therefore, we only compare performance of Minix3 with performance of Linux instead of comparing those three benchmarks between Minix3, Minix3 with Bash and Linux. From the table, we could see Minix3 behaves little worse than Linux in some benchmarks such as in Dhrystone, File Write, Shell scripts and Recursion Test. In some benchmarks Minix3 could perform as well as Linux did, such as Arithmetic Test in short integer. In some cases like System Call overhead, Pipe throughput, Pipe-based context switching, Process Creation, Excel throughput and Arithmetic test in float and double type. Minix3 is far slower than Linux. For example, Minix3 is about 182 times slower than Linux. Minix3 could give better performance on Shell Script running with 8 or 16 concurrent users and Arithmetic test in integer and long integer. We used bar charts (Fig 3, Fig 4, Fig. 5 and Fig. 6) to illustrate the benchmarks. The bar charts of the performance benchmarks of Minix3 and Linux are after Table. 1 (Linux as 100 marks).

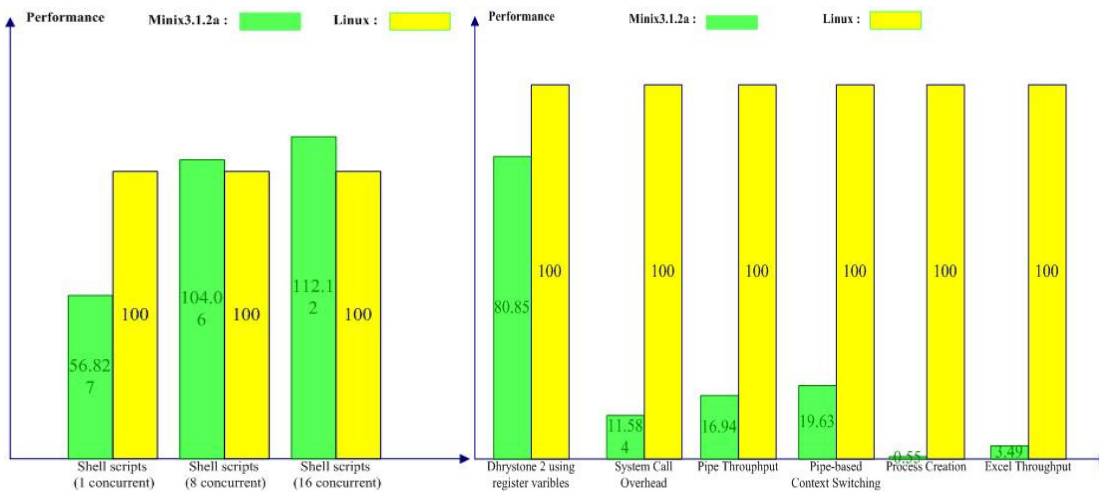


FIGURE 3: and FIGURE 4: Benchmarks in Shell Script Running

Benchmarks	Minix 3.1.2a running Bash-3.0	Minix 3.1.2a	Linux
Dhrystone 2 using register variables	1944025.1 loop/s	1892225 lps	2340428.8 lps
System Call Overhead	60340.9 lps	60366 lps	521107.6 lps
Pipe Throughput	42795.4 lps	42584.1 lps	251399.8 lps
Pipe-based Context Swithing	16668.3 lps	17506.2 lps	89164.7 lps
Process Creation	30 lps	24.6 lps	4479 lps
Excel Throughput	36.7 lps	36.7 lps	1051.9 lps
File Read 1024 bufsize 2000 maxblocks	81011 KBps	80693 KBps	220582 KBps
File Write 1024 bufsize 2000 maxblocks	75644 KBps	75511 KBps	99276 KBps
File Copy 1024 bufsize 2000 maxblocks	39228 KBps	39165 KBps	65518 KBps
File Read 256 bufsize 500 maxblocks	24833 KBps	24721 KBps	81638 KBps
File Write 256 bufsize 500 maxblocks	26400 KBps	26372 KBps	42460 KBps
File Copy 256 bufsize 500 maxblocks	12674 KBps	12664 KBps	25413 KBps
File Read 4096 bufsize 8000 maxblocks	30002 KBps	29997 KBps	392921 KBps
File Write 4096 bufsize 8000 maxblocks	32800 KBps	32799 KBps	158023 KBps
File Copy 4096 bufsize 8000 maxblocks	14064 KBps	14014 KBps	109613 KBps
Shell scripts (1 concurrent)/ (8 concurrent)/ (16 concurrent)	922.5 lps / 212 lps / 112.3 lps	825.6 lps / 207 lps / 111 lps	1452.8 lps / 199 lps / 99 lps
Arithmetic Test (double) / (float)	3656 lps / 7066 lps	3656 lps / 7067.4 lps	250048.8 lps / 259010.4 lps
Arithmetic Test (short)	249705 lps	249774 lps	252042 lps
Arithmetic Test (int) / (long)	261163.6 lps / 261175.5 lps	261163.9 lps / 261159.6 lps	258079.3 lps / 258180.7 lps
Arithoh	4597523.4 lps	4598553 lps	140482502 lps
C Compiler Troughput	43.8 lpm	43.6 lpm	358.7 lmp
Dc: sqrt(2) to 99 decimal places	1721.8 lps	1424.7 lps	38247.5 lps
Recursion Test-Tower of Hanoi	27023.3 lps	27149.8 lps	37328.8 lps

TABLE 1: Unixbench Benchmarks in Minix3 and Linux

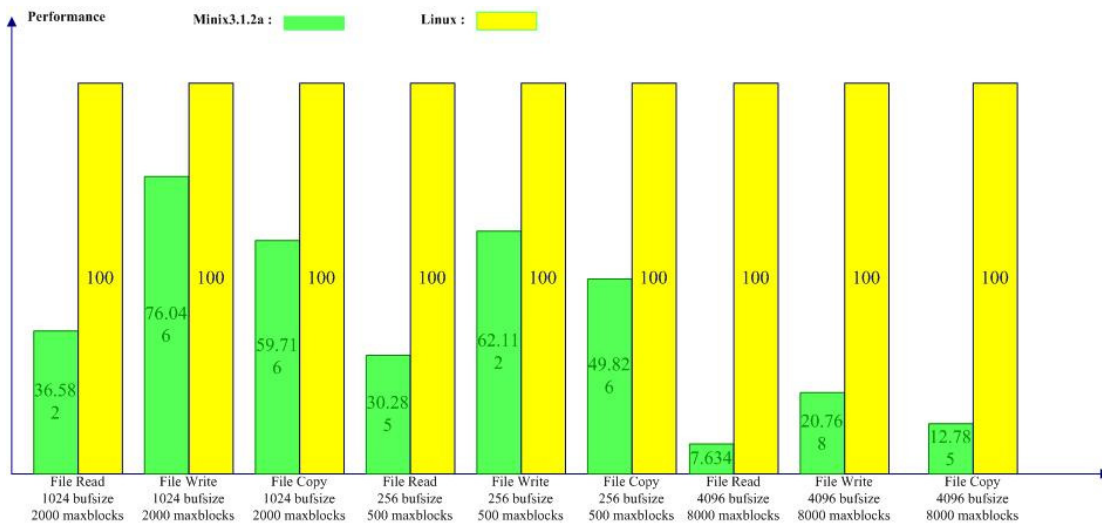


FIGURE 5: File System Benchmarks

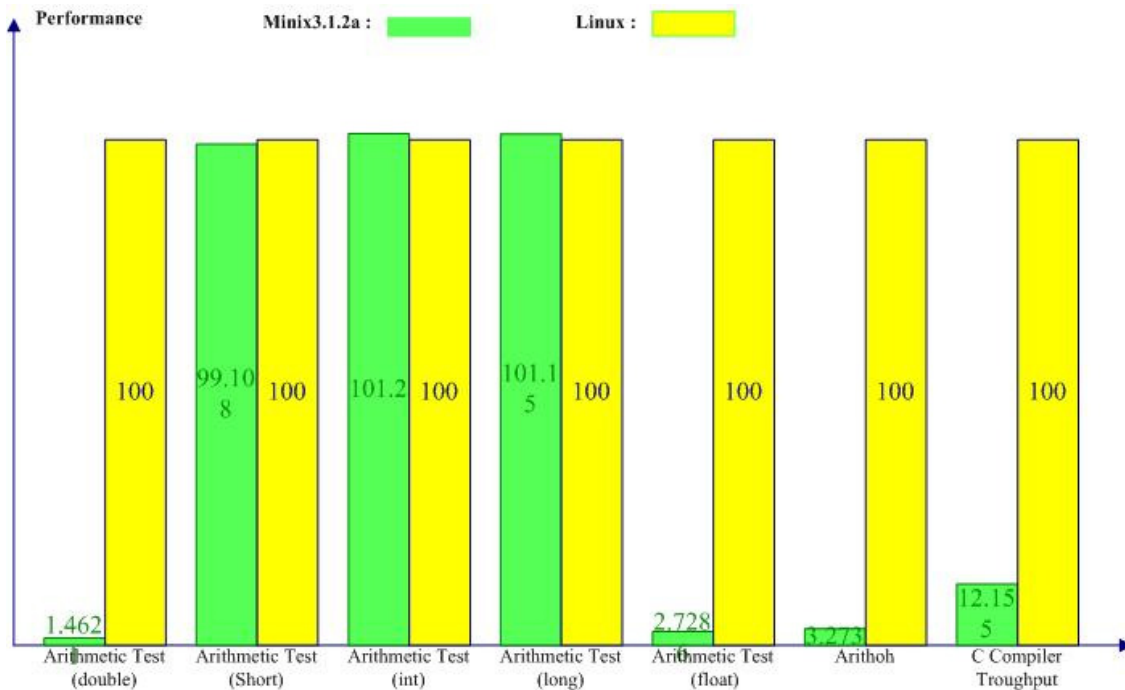


FIGURE 6: Arithmetic Benchmarks

3.2 Microsecond Level Evaluation Benchmarks

As stated in last chapter, Unixbench starts an infinite loop running a test program and uses a global variable to record the number of loops. When the time is up, it uses a signal to interrupt the loop and records how many times the test program ran. The more loops run the better system has performed. Next we want a more precise evaluation result at the microsecond level. So we ran the system call function "gettimeofday()" at the beginning and the end of test program and subtracted the returned values in order to get the microseconds used in running test program. In Unixbench, the main program to evaluate system call overhead is as the following:

Hui Miao

```
while (1) {  
    close(dup(0));  
    getpid();  
    getuid();  
    umask(022);  
    iter++;  
}
```

When the time is up, the signal sent from kernel will interrupt the infinite loop and the value of iter will be recorded. For testing the real execution time on system call overhead, we run "gettimeofday()" system call at the beginning and the end of the program. So the main program codes are the following:

```
gettimeofday(time, tzone); /*get the time befor running */  
for ( i =0 ; i < 1; i++) /* do test program for just once */  
{  
    /* copy from Unixbench */  
    close(dup(0));  
    getpid();  
    getuid();  
    umask(022);  
}  
gettimeofday(time1, tzone1); /*record finish time */
```

Ran the test program on Minix3 and Linux separately. When the test program ran for one iteration, the result could not be measured on Minix3 because the system is too fast. So we increased the running loops of test program. When we run test program for 1000 times, the execution time was get in Minix3. The following table are the evaluation results:

	Minix3					Linux				
Iterations (i)	1	10	100	1000	10000	1	10	100	1000	10000
Time (Microsecs)	None	None	None	15,333	166,666	23	45	220	2,000	22,000

TABLE 2 : Benchmarks on microsecond level

From Table 2, we could see running system call test program for 1000 iterations cost Minix3 15333 microseconds and cost Linux 2000 microseconds. In 10000 iterations, Minix3 spent 166666 microseconds, whereas Linux use 22000 microseconds. In 1000 iterations case, Minix3 is about $15333/2000 = 7.6$ times slower than Linux. In 10000 iterations case, Minix3 is $166666/22000 = 7.58$ times slower than Linux. We back to the evaluation result Unixbench got; Minix3 is about $521107.6 \text{ lps}/60366 \text{ lps} = 8.6$ times slower than Linux. The results of Unixbench and the results of microsecond level evaluation are similar. The purpose of the project is to evaluate the performance of microkernel operating system, which could define whether microkernel operating systems are as practical as current monolithic operating systems. Then the approach was to analyze the benchmarks and separate performance results that are caused by different system implementation from the results that are inherited due to the heavy IPC overhead. Therefore, we could only consider the benchmarks Unixbench gave, if the microsecond level results are similar to the benchmarks Unixbench gave.

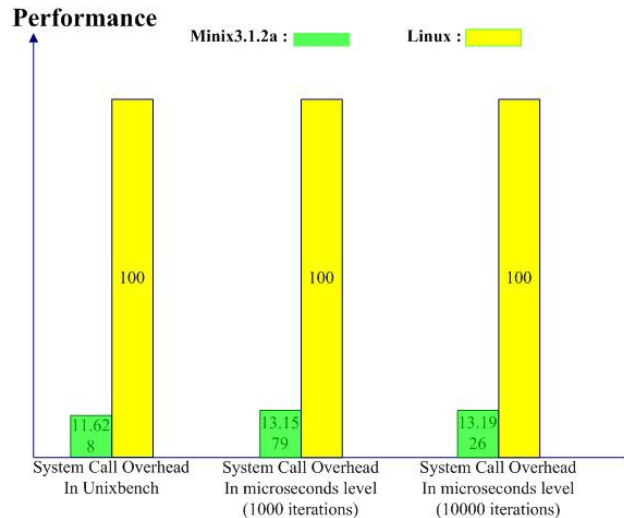


FIGURE 7: Similar Results in Unixbench and Microseconds Level

5. SUMMARY AND CONCLUSION

5.1 Summary of the Conclusion

The project used micro-benchmark tool Unixbench to measure the overall system performance of Minix3.1.2a and the performance of Linux (Fedora Core 6.0). The evaluation test was done in user application layer of the systems. In this way that we could testify the second generation microkernel operating systems whether could be as flexible, portable and secure as monolithic operating systems like Linux. Unixbench gave many benchmarks on MINIX3 and Linux, such as system call overhead, pipe throughput, arithmetic test and so on. The result shows MINIX3 has better performance on Shell Scripts running and Arithmetic test and Linux has better performance on other aspects such as system call overhead, process creation and so on. Linux gave a better performance than Minix3 on the overview of benchmarks. However, after analyzing the benchmarks Unixbench gave, we realized that many benchmarks such as process creation, floating point arithmetic test and Arithoh test in Minix3 could be optimized by system tuning. The following list is the summary of the Unixbench benchmarks discussions:

Process Creation: The benchmark shows that Minix3 is about 182 times slower than Linux. However, COW (Copy-On-Write) technique is a main reason that causes the big performance difference in process creation benchmarks.

Floating Point Arithmetic Test: The benchmark shows that Minix3 is about 68 times and 36 times slower than Linux in double and float data type respectively while Minix3 perform as well as Linux in integer arithmetic test. The reason that causes the poor performance on floating point arithmetic test is Minix3 does not support Floating Point Unit (FPU) that is integrated into CPU. Minix3 used software to emulation the floating point operations, which is much slower than hardware floating point operation supported by Linux.

Shell Script Running: In Unixbench's benchmarks, Minix3 performed better than Linux in shell script running in 8 and 16 concurrent users cases. Because Minix3 does not have virtual file system, so the buffer cache implementation in the systems would be different from Minix3 to Linux. Linux with virtual memory system may require more CPU cycle on page allocation than Minix3.

5.2 Future Works

At the middle of this project, a new experimental version of Minix3 was released. Minix3.1.3, an experimental version of Minix3 which was released at 13th April 2007. A few changes were made

in Minix3.1.3, such as enlarged file system, adding virtual file system, new boot procedure and so on. Notice that virtual file system was imported to Minix3; therefore there must be a big change in file system in Minix3. So we should measure the performance of new Minix3 virtual system again using same benchmark tool and benchmarks in Linux. Measurement on process creation may give us a different benchmark compare with the current benchmark. Another benchmark we should focus on is the shell script running. In current benchmarks, Minix3 gave better performance on shell script running with 8 and 16 concurrent users. We guess in this report that the reason that causes better performance in Minix3 in shell script running is Minix3 does not implement virtual file system. In Minix3.1.3, virtual file system was ported on Minix3. Therefore, the benchmarks of shell script running in Minix3.1.3 could give crucial information that whether the virtual memory system is the reason causes Linux slower than Minix3 in shell script running benchmark.

From the benchmarks in chapter 3, we could see that there are many benchmarks such as system call overheads, pipe throughput, context switch overheads, excel overheads and C compiler throughput which we did not discuss yet. Research on those benchmarks need to be done in the future. By doing that, we could make a very clear figure that on Minix3 microkernel performance and could find out that which part of system could be improved by tuning the microkernel system. Then try to optimize the system performance by tuning the microkernel system if we know exactly why Minix3 behaved such a low performance in the benchmark.

From the Minix3 official web site, we also found that some future works should be done by researchers [9]:

1. Testing MINIX 3 on different platforms
2. Porting programs and applications to MINIX 3
3. Porting drivers to MINIX 3
4. Building a driver framework to use FreeBSD or Linux drivers
5. Porting MINIX 3 to different architectures

6. REFERENCES

- [1] Jochen Liedtke, "*toward real microkernels*". Communications of ACM September 1996. Vo139, No. 9.
- [2] Chen, J.B. and Bershad, B.N. "*The impact of operating system structure on memory system performance*". In Proceedings of the 14th ACM Symposium on Operating System Principles (SOSP) (Asheville, N.C., Dec. 1993). ACM Press, 1993, pp. 120—133.
- [3] Liedtke, J. "*On microkernel construction*". In Proceedings of the 15th ACM Symposium on Operating System Principles (SOSP) (Copper Mountain Resort, Cob., Dec. 1995). ACM Press, New York, 1995, pp. 237-250.
- [4] Liedtke, J. "*Improving the IPC by design Kernel*". 14th ACM Symposium on Operating System Principles (SOSP) Asheville. 1993, pp. 10-11.
- [5] Andrew S Tanenbaum & Albert S Woodhull. "*Operating System Design and Implementation*" (3rd Edition). Prentice Hall Software Series. 2006.
- [6] L4 Kickstart < <http://www.l4ka.org/projects/pistachio/kickstart.php> >. Edited by University of Karlsruhe. 2000-2006. Viewed on 24th Mar. 2010.
- [7] D. R. Engler, M. F. Kaashoek, J. O'Toole. "*Exokernel: an operation system architecture for application-level resource management*". ACM SIGOPS Operating Systems Review , Proceedings of the fifteenth ACM symposium on Operating systems principles SOSP '95, Volume 29 Issue 5.

Hui Miao

- [8] Lmbench home website <<http://www.bitmover.com/lmbench/>> Lmbench - Tools for Performance Analysis. Viewed on 24th Mar. 2010.
- [9] Minix3 home website < <http://www.minix3.org/doc/environ.html>> MiniFAQ about MINIX 3 Programming. Viewed on 14th May 2010.
- [10] Minix tip website < <http://www.minixtips.com/>> Tips For Running the Minix OS Version 3. Viewed on 14th May 2010.
- [11] FTP of Unixbench <<http://www.tux.org/pub/tux/benchmarks/System/unixbench>> Viewed on 13th Mar 2010.
- [12] Daniel P. Bovet & Marco Cesati. "*Understanding the Linux Kernel*". O'REILLY Press, Nov 2005.
- [13] Floating Point Unit, <http://en.wikipedia.org/wiki/Floating_point_unit>, From Wikipedia, the free encyclopaedia. Viewed on 24th Mar 2010
- [14] Comparing Linux and Minix, <<http://lwn.net/Articles/220255/>>, LWN.net article, Viewed on 16th May 2010.
- [15] Hbench-OS Operating system Benchmarks <<http://www.eecs.harvard.edu/vino/perf/hbench/index.html>>, Viewed on 16th May 2010.
- [16] H. Hartig, M. Hohmuth, J. Liedtke, S. Schänberg, J. Wolter, "*The Performance of μ -Kernel-based Systems*", 16th SOSP TU Dresden, Fakultät Informatik, Heft Jan 1997.
- [17] Ben Leslie, Carl van Schaik and Gernot Heiser, "*Wombat: a portable user-mode Linux for embedded systems*", Proceedings of the 6th Linux Conference Australia, Canberra, April, 2005.
- [18] ERTOS Website <<http://www.ertos.nicta.com.au/research/l4/performance.pml>>, National ICT Australia United, Viewed on 16th May 2010.
- [19] Release Notes of MINIX 3.1.3 - Developer's Interim Release, <<http://www.minix3.org/download/releasenotes-3.1.3.html>>, Minix3 Home Website, Viewed on 17th May 2010.

Mutual Authentication Between Base and Subscriber Station Can Improve the Security of IEEE 802.16Wimax Network

Mohammad Hossain

*Telecommunication Systems
Blekinge Institute of Technology
Karlskrona, 37179, Sweden*

reganmh8@gmail.com

Mohammad Zavid Parvez

*Signal Processing
Blekinge Institute of Technology
Karlskrona, 37179, Sweden*

zavidparvez@hotmail.com

Mohammad Hamidul Islam

*Information and Communication Systems Security
Royal Institute of Technology
Stockholm, 10044, Sweden*

rubeldiu@yahoo.com

Abstract

High throughput broadband connection over long distance is greatly demanded in the present web application. IEEE 802.16/WiMax technology is one of the latest additions on internet broadband. When wireless devices are connected to the broadband wireless access, security comes on the front line to ensure the communication safe and protected from any kind of attacks or threats. Strong and effective security must be confirmed to make the wireless environment reliable and risk less. Base station authentication is an important part of WiMax security which must be confirmed to make the environment more secure. This paper derived the technique to secure the environment by confirming the authentication of base station.

Keywords: WiMax, Authentication, Base Station, Broadband Connection, Security.

1. INTRODUCTION

Since the last decade of twentieth century, data networks have successfully acknowledged a progressive expansion and getting update with time. The whole world is the prime target to come under the coverage of fixed internet to facilitate the communication easier and faster. For this extra large coverage, wireless access is chosen because of its last end focusing power and reaching ability rather than that of wired network. The expansion of high speed wireless data access i.e., in MB/s, is going to make wired network a history which is just a matter of time in the present twenty first century.

The world telecommunication became instantly prosperous in consideration of its high speed data transmission and coverage to the end user when WiMax added to it. The resource scarcity has been eliminated instantly which was concerning present service providers even some few years before. All of the major telecommunication services like as voice (mobile and static), video and data sharing got the new shape in true market based competition.

WiMax provides fixed, portable or mobile non line-of-sight (NLOS) service from a base station (BS) to subscriber station (SS) and so also known as customer premise equipment (CPE). Fig. 1 shows the transmission of WiMax between point-to-point and point-to-multipoint scenario. With a throughput of 72 Mbps it covers 30 miles of area around it in point-to-point communication. In the case of point-to-multipoint scenario it covers 6 miles NLOS range with a throughput of 40 Mbps.

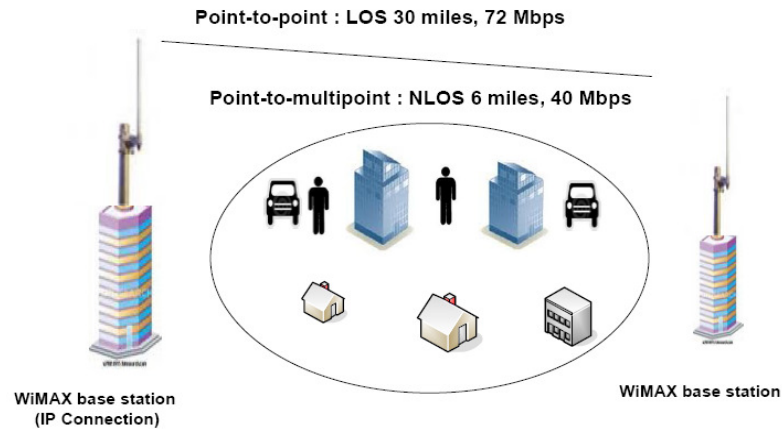


FIGURE 1: Fixed WiMAX showing point to point and point to multipoint communication [1].

During the time of designing of IEEE 802.11 wireless standard, scientist and manufacturer considered the security issues after all other infrastructure and implementation [8]. So the security of this standard familiarized as part of one of most vulnerable wireless protocols. In the case of IEEE 802.16 standard, security was thought to be implemented within the protocol but the depressing thing is that it was also left aside and not considered as a front line topic. The consequence is that, there are still questions about the transmission security of this standard. Many papers published on different security issues and researchers are still working to make it robust, secured and trustworthy.

2. DIFFICULTY CLASSIFICATION

It is not always unreachable for the attackers to intercept the wireless network as because it is based on radio waves which keep the medium open more or less during transmission. A secured radio transmission was always a concerning issue to the protocol designers. The different issues which primarily considered during the designing time were handling with intrinsic untrustworthiness of wireless medium, good mobility, featuring of protocol to be confirmed while delivering frames in mobility and also power saving [2]. In the former specification IEEE 802.11 wireless standard, security proved to be an inadequately considered issue which was not mentioned in the frontage line. The implementation of the wireless network was also not secured enough so that it can easily handle service disruption and theft. Different kind of attacks took place by the intruders which make the network poorly secured due to the various attacks like as interception, fabrication, modification, interruption and so on. Although in IEEE 802.16 standard designing in MAC layer was especially focused on the security mechanism, yet it might provide inadequate security in multihop scenarios and demand the needs of rising applications in WiMax networks [3]. Paper [4] mentioned that WiMax is suitable to physical layer attacks like as jamming which is a source of noise and so strong that it significantly reduce the capacity of transmission channel. It also mentioned of scrambling which is kind of jamming occurs to specific frames for short intervals of time. In IEEE 802.16 standard securities are implemented as a sub-layer at the bottom of MAC layer in order to protect the data exchange between the MAC and PHY layer but does not protect the PHY layer itself against the attacks which intends to malfunction the internally built weakness of the wireless links [5]. So, security is such an important issue in data transmission of wireless network that a little mistreat or falsification of data may generate huge chaos and disable the complete system for long run and cause immeasurable sufferings.

3. BASE STATION SPOOFING

When a rogue BS gets credentials from a legitimate SS to process further transmission and so cheat the SS called BS spoofing. Initially the SS tries to get authorization and traffic keying material from the BS. For this, it uses the PKM (Key Management Protocol) protocol. The

protocol has the necessary features to support the periodic reauthorization and key refreshment. This protocol also uses the RSA public key encryption algorithm and X.509 digital certificates. The key exchanging between the SS and the BS takes place in presence of some strong encryption algorithm like as DES (Data Encryption Standard), AES (Advanced Encryption Standard) etc. A client-server model is performed during transmission by the PKM protocol. The SS acts as a PKM client and the BS acts a PKM server. Being a client, the SS requests keying materials from the BS and the BS provides the requested keying materials to the SS. When the SS is authorized, it gets keying materials from the BS. MAC management messages of the protocol are used between the BS and the SS which are PKM-REQ and PKM-RSP as already mentioned. To establish a shared and secret Authentication Key (AK) between the SS and the BS, the protocol uses public key cryptography. Consecutive PKM exchanges happened by maintaining a secure traffic encryption keys (TEKs) using the authentication key. A BS authenticates an SS during the initial authorization.

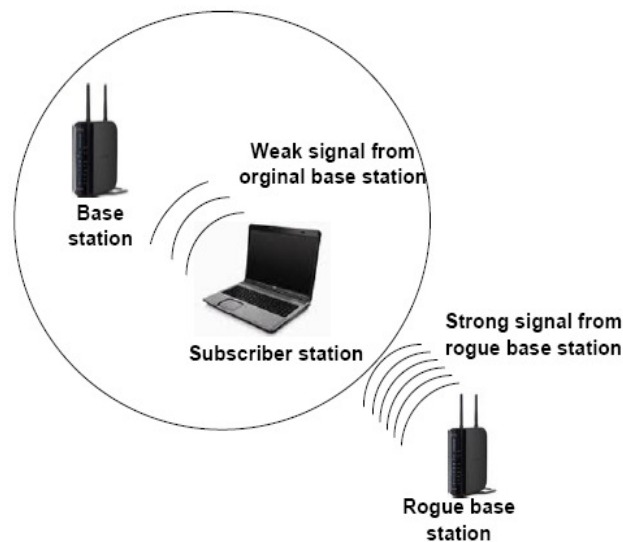


FIGURE 2: Base Station spoofing by a rogue node.

The manufacturer issues X.509 digital certificate for each SS which are given to them. The certificate has the SS's public key and MAC address. The SS presents its digital certificate to the BS when requesting an AK. After verifying the digital certificate the BS uses the verified public key to encrypt an AK and transmits to the SS. This way exchanging the AK, the BS established an authenticated identity of a client SS. Now, the SS is authorized to proceed for further services. Because of this authorization process, it is very difficult for an attacker to get inside into the network being a false SS. But, there is no way of BS authentication. That is the place where the attacker starts its working. Fig. 2 shows how an attacker disguised himself as a rogue BS and creates BS spoofing.

4. MAC ADDRESS SPOOFING

Each SS uses an SS certificate during the time of authentication. This certificate is issued and signed by the manufacturer. The subject field of a certificate contains the MAC address and identifying credentials of an SS. 48-bit MAC address is used in IEEE 802.16 standard. Using this subject field documents the BS and the SS identifies each other during the time of initial ranging and authentication process. SS includes the MAC address while sending Ranging Request (RNG-REQ) to the BS. The BS sends back the MAC address again in its Ranging Response (RNG-RSP). An eavesdropper may try to get the MAC address of the authorized SS by intercepting either the uplink or the downlink of the connection. But, it becomes difficult for the attacker because of maintaining a private key public key method between the BS and the SS. The attacker compromises it with the BS as the BS is not authorized like the SS. There is no way

of authentication or BS certificate that the BS sends to the SS to introduce it as a legitimate BS and not the false one.

5. SECURITY IMPROVEMENT

In IEEE 802.16 network, an SS must be authenticated and verified by the BS to obtain the network credentials and to be a part of the network as a legitimate node. But, there is no such verification or authentication procedure by which an SS can verify and authenticate a BS before it starts to give all its credentials to the unknown BS. The easy thing is that a BS can also pass through some verification procedure by which the SS will understand that it is communicating with a legitimate BS and not with the rogue one. Therefore, mutual authentication is must to establish a both side secure and trustworthy transmission. It is rather an intelligent thinking to work on the process besides running behind the intruders. Different attacks are discussed so far which are related more or less only because the base station authentication is not present in the existing authentication protocol which is shown below in Fig. 3. It shows the present scenario of the SS verification and authentication but not for the BS.

<p>Message 1: SS → BS: Cert (SS) (Auth Req message) Message 2: SS → BS: Cert (SS) Capabilities BCID Message 3: BS → SS: KU_{SS} (AK) SeqNo Lifetime SAIDList</p>
--

FIGURE 3: Authentication Protocol of IEEE 802.16 Standard.

In Fig. 3, Cert (SS) is the digital certificate which the SS obtained from the manufacturer. The basic fields which are included on it are certificate version, serial number, signature, issuer, validity, subject, subject public key info, issuer unique ID, subject unique ID, and extensions. Capabilities contain the SS supported authentication and data encryption algorithms. Basic Connection ID of SS is termed as BCID. KU_{SS} (AK) is the Authorization key generated by the BS for the SS and is encrypted with the public key of SS. SeqNo is a 4-bit sequence number. Lifetime is the number of seconds. SAIDList contains the identities and the properties of the SAs (Security Associations) due to which an SS is authorized to obtain keying information.

When the rogue BS obtained all the credentials from the SS being a legitimate BS, it can put the SS out of the network by repeating the received messages from SS to the legitimate BS again and again. When the preliminary message repeatedly comes, the BS stops receiving any more messages from the source considering it as a fraud or disturbing element. So, the existing protocol must be renewed to ensure secure and trustworthy communication especially in banking, government activities or other important sectors only to maintain high security from the intruders and hackers.

5.1 Mutual Authentication can be Established as the Proposed Algorithm

During the time of transmission, an SS initiates the session. It sends its identifications, capabilities and other requirements to the BS. After checking the documents the BS sends back Authorization Reply to the SS. This reply must be checked whether it is from the legitimate BS or rogue BS. As the SS has no ability to check it, it can get help of a trusted third party. This third trusted party is an Authentication Server (AS) which must be in the knowledge of SS. The AS and the BS know each other as they are manufactured by the manufacturer this way. After getting the Auth Reply from the BS, the SS will send it to the Authentication Server (AS). The BS will also forward information containing its own ID, SSID and SS credentials to the AS. The AS will judge both side information's received from the BS and the SS and return the confirmation to the SS. In this message, if SS finds that the BS is a legitimate one, it will continue its transmission. Otherwise, it will end further communication with the BS. The Fig. 4 shows the new authentication protocol to avoid rogue BS. Here the BS sends back the Auth Reply message to the legitimate SS where it also includes its ID which the SS will present to Authentication Server (AS). If any attacker tries to involve the network, it will be captured by Authentication Server. However,

legitimate BS will not allow any other party but the legitimate SS as it checks its ID and other credentials. DES (Data Encryption Standard) encryption can be used in all private-public key cases.

6. SIMULATION RESULT

The proposed algorithm is demonstrated in a simulation process only to represent its accuracy, perfections and way of working.

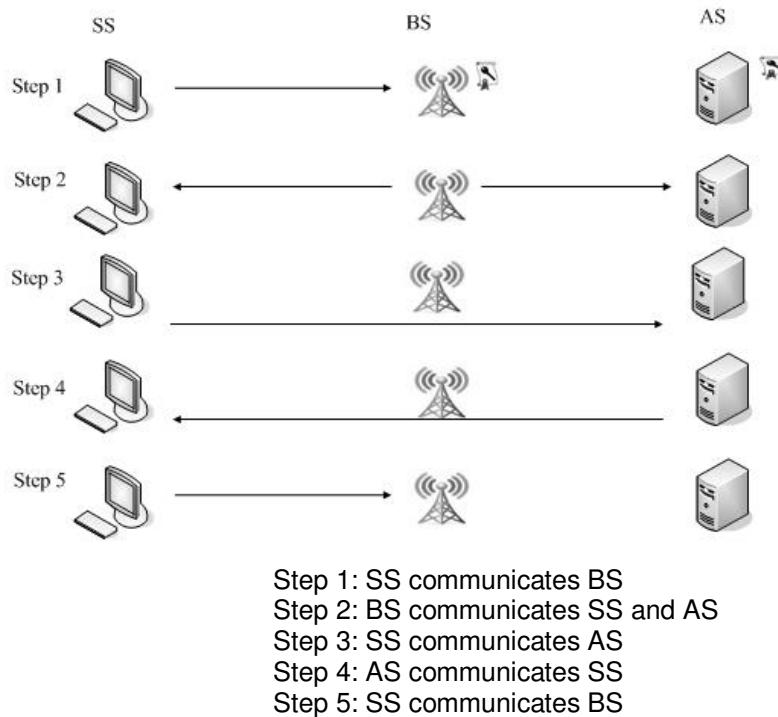


FIGURE 4: The Mutual Authentication process to avoid Rogue BS attack.

The obtained result of the simulation process confirmed that the algorithm works according to its theme and process. The testing environment of the simulation and its way of working is given below.

6.1 Testing Environment

Testing environment expressed the manipulation of the result. A simple implementation of a TCP client server relationship has been considered where the SS works as client, the BS the AS work as server respectively. The algorithm has some prerequisite conditions like as the BS and the AS are previously trusted to each other and the SS and the BS would use public key cryptography for message encryption or decryption.

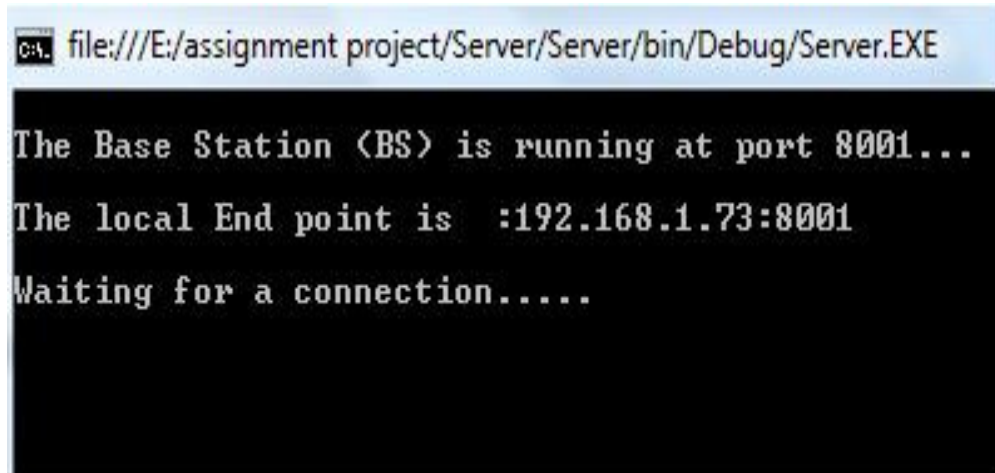
The simulation process used the following things in its testing environment.

- Microsoft Windows Vista platform
- Processing power of the machine 2.0GHZ
- The Socket class in the .NET framework
- TCP/IP protocol.
- Some encryption and decryption capabilities.
- Programming language C#.NET.

6.2 Overall Communication Process

The overall communication processes are as follows:

Step 1: The BS is waiting to get someone's request like as the SS. Here the BS is running at port 8001 and the local IP is 192.168.1.73 shown in Fig. 5.



```
file:///E:/assignment project/Server/Server/bin/Debug/Server.EXE
The Base Station (BS) is running at port 8001...
The local End point is :192.168.1.73:8001
Waiting for a connection.....
```

FIGURE 5: The Base Station (BS) is waiting for the connection.

Step 2: The SS has connected to the BS with port 8001 and Local end point is 192.168.1.73. The SS is sending its subscriber ID and all necessary credentials to the BS shown in Fig. 6.

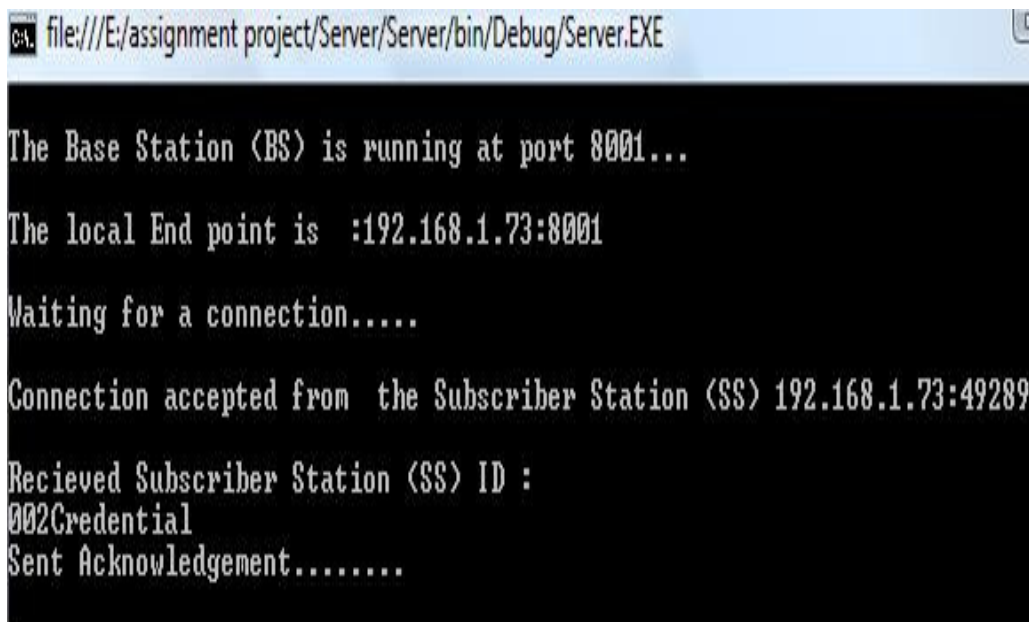


```
file:///E:/assignment project/Client/Client/bin/Debug/Client.EXE
Connecting.....
Connected to the Base Station (BS)
Enter the Subscriber Station (SS) ID to be transmitted to the Base Station (BS)
: 002Credential
Transmitting.....
```

FIGURE 6: The SS is sending information to the BS.

Step 3: The BS received successfully the SS credentials. The BS verified the SS credentials. After the verification the BS starts further communication with the SS. The BS encrypted its ID by using DES encryption algorithm and sends it to the SS shown in Fig. 7. The BS does the

verification procedure by its predefined knowledge about the SS. The ID or the information of the SS is attached inside the simulation code so that the BS can verify the SS if the correct ID or address is used otherwise would discard the SS which is shown in Fig. 7.



```
file:///E:/assignment project/Server/Server/bin/Debug/Server.EXE

The Base Station (BS) is running at port 8001...

The local End point is :192.168.1.73:8001

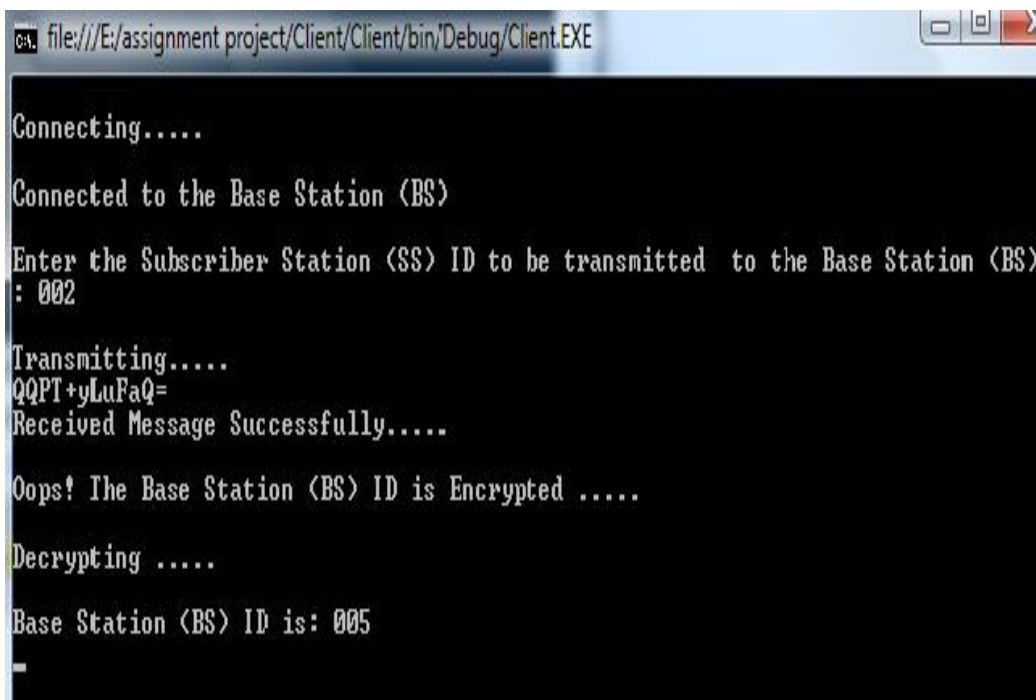
Waiting for a connection.....

Connection accepted from the Subscriber Station (SS) 192.168.1.73:49289

Recieved Subscriber Station (SS) ID :
002Credential
Sent Acknowledgement.....
```

FIGURE 7: The BS is sending information to the SS.

Step 4: The SS received encrypted information from the BS. The SS is decrypting the BS information by using DES decryption and found the BS ID.



```
file:///E:/assignment project/Client/Client/bin/Debug/Client.EXE

Connecting.....

Connected to the Base Station (BS)

Enter the Subscriber Station (SS) ID to be transmitted to the Base Station (BS)
: 002

Transmitting.....
QQPT+yLuFaQ=
Received Message Successfully.....

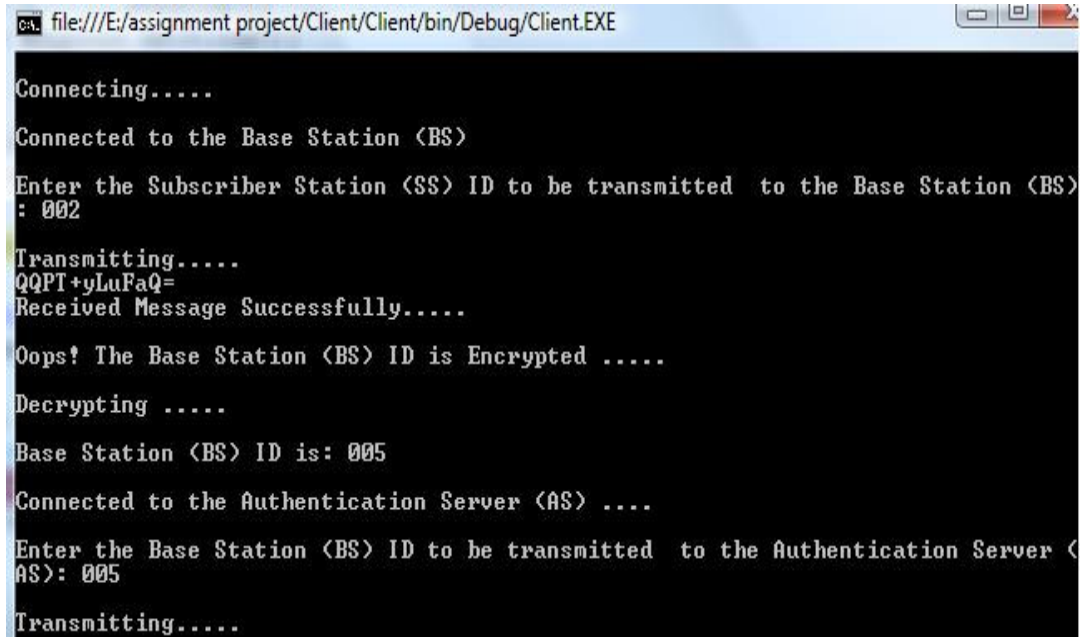
Oops! The Base Station (BS) ID is Encrypted .....

Decrypting .....

Base Station (BS) ID is: 005
```

FIGURE 8: The SS received message from the BS and decrypting the message.

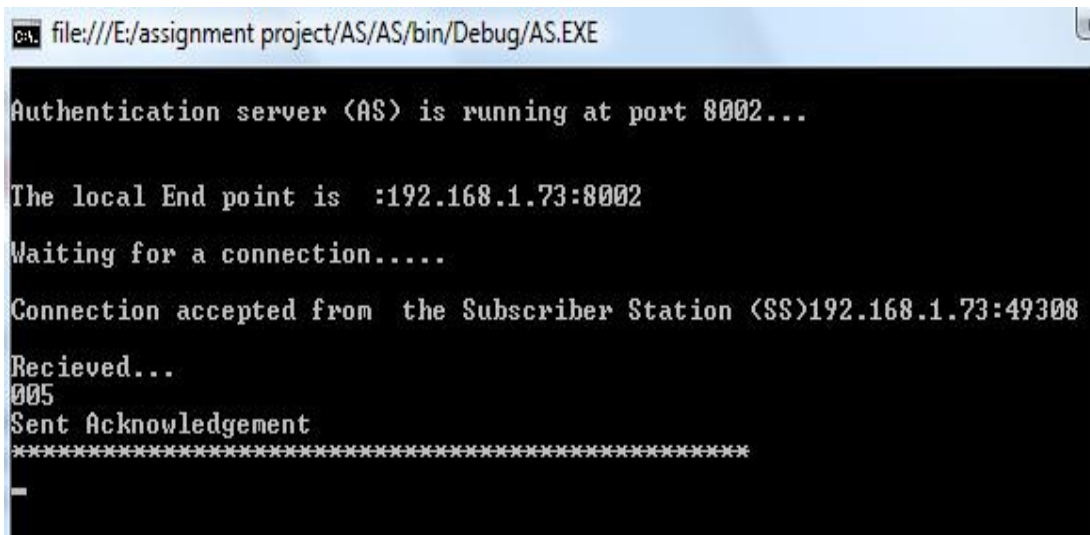
Step 5: To verify the BS, the SS is sending the BS ID to the Authentication Server (AS). Local end point of the AS is 192.162.1.73 and TCP port 8002.



```
ca. file:///E:/assignment project/Client/Client/bin/Debug/Client.EXE
Connecting.....
Connected to the Base Station <BS>
Enter the Subscriber Station <SS> ID to be transmitted to the Base Station <BS>
: 002
Transmitting.....
QQPT+yLuFaQ=
Received Message Successfully.....
Oops! The Base Station <BS> ID is Encrypted .....
Decrypting .....
Base Station <BS> ID is: 005
Connected to the Authentication Server <AS> ....
Enter the Base Station <BS> ID to be transmitted to the Authentication Server <
AS>: 005
Transmitting.....
```

FIGURE 9: The SS transmitting message to the AS.

Step 6: The AS received message from the SS, verified the BS as rouge or trusted and sent acknowledgement to the SS.



```
ca. file:///E:/assignment project/AS/AS/bin/Debug/AS.EXE
Authentication server <AS> is running at port 8002...
The local End point is :192.168.1.73:8002
Waiting for a connection.....
Connection accepted from the Subscriber Station <SS>192.168.1.73:49308
Recieved...
005
Sent Acknowledgement
*****
```

FIGURE 10: The AS verified the BS and sent message to the AS.

Step 7: After receiving the acknowledgement of the AS, the SS got the confirmation that the BS is trusted or not. If the BS is trusted then protected communication will start between the SS and the BS otherwise all communication will remain stop in this phase. In this simulation process, the SS is ID is 002 and the BS ID is 005. When the SS sends the BS ID 005 to the AS, it is accepted as a trusted BS which is shown in Fig. 11. Here any other ID except 005 for BS would have

considered as a false or unknown one as in the simulation code only 005 is inserted as a trusted BS which is in the knowledge of AS.

```

file:///E:/assignment project/Client/Client/bin/Debug/Client.EXE
Connecting.....
Connected to the Base Station <BS>
Enter the Subscriber Station <SS> ID to be transmitted to the Base Station <BS>
: 002
Transmitting.....
QQPT+yLuPaQ=
Received Message Successfully.....
Oops! The Base Station <BS> ID is Encrypted .....
Decrypting .....
Base Station <BS> ID is: 005
Connected to the Authentication Server <AS> ....
Enter the Base Station <BS> ID to be transmitted to the Authentication Server <
AS>: 005
Transmitting.....
ThisBase Station <BS> is Trusted !!!!!!!
*****
    
```

FIGURE 11: The SS verified the BS.

7. COMPARATIVE STUDY

Paper [5] said the way to save the data exchange between MAC layer and the PHY layer. It also mentioned that the PHY layer cannot save itself against the attacks which intercept inside the wireless channel. Paper [6] shown the security aspects of the IEEE 802.16 Standard and point out the security vulnerabilities, threats and risks associated with this standard shortly. [7] Examined the MAC layer of the 802.16 standard to determine the presence of the denial of service attacks and also the attacks that may be unique to the 802.16 standard. But it did not point out the way to prevent these problems. It did not solve the BS authentication to save the legitimate SS except just discussing the problems a bit detail which left curiosity to researchers to find a way to figure out a solution and make this communication environment more secured and trustworthy.

8. CONCLUSION

Although being a new technology, this standard works with strong encryption algorithm, data encryption standard (DES) and with a strong key management scheme. Attacks on privacy, integrity and authentication can be overcome by taking some few necessary steps. Besides, the standard itself provides adequate solutions to defend against others major attacks which were somewhat concerning issues in previous standards. Base station authentication will make the whole communication secure and reliable which was not defined in the architecture of the IEEE 802.16 network though different separate works have been done so far and is still a concerning issue in data transmission. This paper solved this problem by ensuring mutual authentication technique for both base and subscriber station in a way that no intruders or outsiders can penetrate the network disguising themselves as part of it and doing unnecessary activities. The simulation result proved that when mutual authentication is established, a secure and reliable transmission can be achieved in point to point or point to multipoint communication in IEEE 802.16/WiMax network.

9. REFERENCES

- [1] Frank Ohrtman, WiMAX Handbook, building 802.16 WiMAX networks, McGraw-Hill 2005.
- [2] Mahmoud Nasreldin, Heba Aslan, Magdy El-Hennawy, Adel El- Hennawy. WiMax Security, 22nd International Conference on Advanced Information Networking and Applications Workshops 2008, p. 1335- 1340.
- [3] Kejie Lu and Yi Qian, University of Puerto rico, Hsiao-Hwa Chen, National Sun Yat-Sen University. A Secure and Service-Oriented Network Control Framework for WiMax Networks, IEEE Communications Magazine, May 2007.
- [4] Michel Barbeau, School of Computer Science, Carleton University, Canada. WiMax Threat Analysis, Q2SWinet'05, October 13, 2005, Montreal, Quebec, Canada.
- [5] Hyung-Joon Kim, IEEE 802.16/WiMax Security, Dept. of Electrical and Computer Engineering, Stevens Institute of Technology, Hoboken, New Jersey, Unpublished.
- [6] Jamshed Hasan, School of Computer and Information Science, Edith Cowan University, Australia Security Issues of IEEE 802.16 (WiMax), [http://scissec.scis.ecu.edu.au/conference_proceedings/2006/aism/Hasan%20-%20Security%20Issues%20of%20IEEE%20802.16%20\(WiMAX\).pdf](http://scissec.scis.ecu.edu.au/conference_proceedings/2006/aism/Hasan%20-%20Security%20Issues%20of%20IEEE%20802.16%20(WiMAX).pdf).
- [7] Derrick D. Boom "Denial of Service Vulnerabilities in IEEE 802.16 Wireless Networks" IEEE C802.16e-04/406.
- [8] Loutfi Nuaymi, WiMAX Technology for Broadband Wireless Access, John Wiley & Son Ltd.

Design Model-free Fuzzy Sliding Mode Control: Applied to Internal Combustion Engine

Farzin Piltan

Department of Electrical and Electronic Engineering, Faculty of Engineering, Universiti Putra Malaysia 43400 Serdang, Selangor, Malaysia

SSP.ROBOTIC@yahoo.com

N. Sulaiman

Department of Electrical and Electronic Engineering, Faculty of Engineering, Universiti Putra Malaysia 43400 Serdang, Selangor, Malaysia

nasri@eng.upm.edu.my

Payman Ferdosali

Industrial Electrical and Electronic Engineering SanatkadeheSabze Pasargad. CO (S.S.P. Co),NO:16 , PO.Code 71347-66773, Fourth floor Dena Apr , Seven Tir Ave , Shiraz , Iran

SSP.ROBOTIC@yahoo.com

Iraj Assadi Talooki

Industrial Electrical and Electronic Engineering SanatkadeheSabze Pasargad. CO (S.S.P. Co),NO:16 , PO.Code 71347-66773, Fourth floor Dena Apr , Seven Tir Ave , Shiraz , Iran

SSP.ROBOTIC@yahoo.com

Abstract

Modeling and control of engine systems are vital due to wide range of their applications. As it is obvious stability is the minimum requirement in any control system, however the proof of stability is not trivial especially in the case of nonlinear systems. One of the most active research areas in field of internal combustion engine (IC engine) is control of the fuel ratio. The strategies for control of engines are classified into two main groups: classical and non-classical methods, where the classical methods used the conventional control theory and non-classical methods used the artificial intelligence theory such as fuzzy logic, neural networks and/or neurofuzzy. One of the best nonlinear robust controllers which can be used in uncertainty nonlinear systems is sliding mode controller (SMC). Chattering phenomenon is the main challenge in this controller. Fuzzy logic and neuro control have been applied successfully in many applications. Therefore stable control of an internal combustion engine is challenging because it has uncertain dynamic parameters. This research presents design a fuzzy sliding mode control with improved in sliding mode algorithm which offers a model-free sliding mode methodology. The fuzzy sliding mode controller is designed as a 49 rules Mamdani's error-based fuzzy sliding-like equivalent part instead of nonlinear dynamic equation of equivalent part. Various performance indices like the minimum error, trajectory, disturbance rejection, and chattering control are used for comparison.

Keywords: Internal Combustion Engine, Sliding Mode Controller, Chattering Phenomenon, Fuzzy Sliding Mode Controller, Minimum Error, Trajectory, Disturbance Rejection, and Chattering Control.

1. INTRODUCTION

The internal combustion (IC) engine is designed to produce power from the energy that is contained in its fuel. More specifically, its fuel contains chemical energy and together with air, this mixture is burned to output mechanical power. There are various types of fuels that can be used in IC engines which include petroleum, diesel, bio-fuels, and hydrogen [1]. The output power produced by an IC engine results from the fuel, that it uses, and also its mechanical parts [2].

Modeling of an entire IC engine is a very important and complicated process because engines are nonlinear, multi inputs-multi outputs and time variant. One purpose of accurate modeling is to save development costs of real engines and minimizing the risks of damaging an engine when validating controller designs. Nevertheless, developing a small model, for specific controller design purposes, can be done and then validated on a larger, more complicated model. [3], [4], [5].

Controller design is the main part in IC engines as well as the major objectives is stability and robustness. One of the significant challenges in control algorithms is a linear behavior controller design for nonlinear systems. When system works with various parameters and hard nonlinearities this technique is very useful in order to be implemented easily but it has some limitations such as working near the system operating point[2]. Some of IC engines which work in industrial processes are controlled by linear PID controllers, but the design of linear controller for IC engines is extremely difficult because they are nonlinear, uncertainty, and MIMO[1, 6]. To reduce above challenges the nonlinear robust controllers is used to systems control. One of the best nonlinear robust controller that can used in uncertainty nonlinear systems (e.g., IC engines), is sliding mode controller. But, SMC also has attachment to dynamic equation using equivalent control, so used fuzzy logic system instead equivalent control (e.g., proposed fuzzy sliding mode controller). To have the best solution, this paper focuses on self tuning robust controller (e.g., self tuning fuzzy sliding mode controller) [6], [7].

One of the powerful nonlinear robust controllers is sliding mode controller (SMC), although this controller has analyzed by many researchers recently but the first proposed was in the 1950 [7]. This controller is used in wide range areas such as in robotics, in control process, in aerospace applications, and in IC engines because it has an acceptable control performance and solve some main challenging topics in control such as resistivity to the external disturbance [18-24]. Even though, this controller is used in wide range areas but, pure sliding mode controller has the following disadvantages: Firstly, chattering problem; which caused the high frequency oscillation in the controllers output. Secondly, equivalent dynamic formulation; calculate the equivalent control formulation is difficult because it depends on the dynamic equation [8, 9]. The classical sliding mode controller is classified into two main parts: discontinuous (hitting) controller which is based on discontinuous switching function and equivalent controller which is based on dynamic equations of IC engine.

On the other hand, after the invention of fuzzy logic theory in 1965, this theory was used in wide range applications that fuzzy logic controller (FLC) is one of the most important applications in fuzzy logic theory because the controller has been used for nonlinear and uncertain (e.g., robot manipulator) systems controlling. Conversely pure FLC works in many areas, it cannot guarantee the basic requirement of stability and acceptable performance[10, 11].

Although both SMC and FLC have been applied successfully in many applications but they also have some limitations. The boundary layer method is used to reduce or eliminate the chattering and proposed method focuses on substitution error-base fuzzy logic system instead of dynamic equivalent equation to implement easily and avoid mathematical model base controller [20-24]. To reduce the effect of uncertainty in proposed method, self tuning method is applied in error-base fuzzy sliding mode controller [18-24] in IC engine. This paper is organized as follows: In section 2, main subject of engine operating cycle are presented. Detail of modelling of fuel ratio in IC engine is presented in section 3. Detail of classical sliding mode controller is presented in

section 4. In section 5, the main subject of proposed fuzzy sliding mode controller is presented. In section 6, the simulation result is presented and finally in section 7, the conclusion is presented.

2. ENGINE OPERATING CYCLE

In an internal combustion engine, a piston moves up and down in a cylinder and power is transferred through a connecting rod to a crank shaft. The continual motion of the piston and rotation of the crank shaft as air and fuel enter and exit the cylinder through the intake and exhaust valves is known as an engine cycle. The first and most significant engine among all internal combustion engines is the Otto engine, which was developed by Nicolaus A. Otto in 1876 (Figure 1). In his engine, Otto created a unique engine cycle that consisted of four piston strokes. These strokes are:

1. Intake stroke
2. Compression stroke
3. Expansion stroke
4. Exhaust stroke

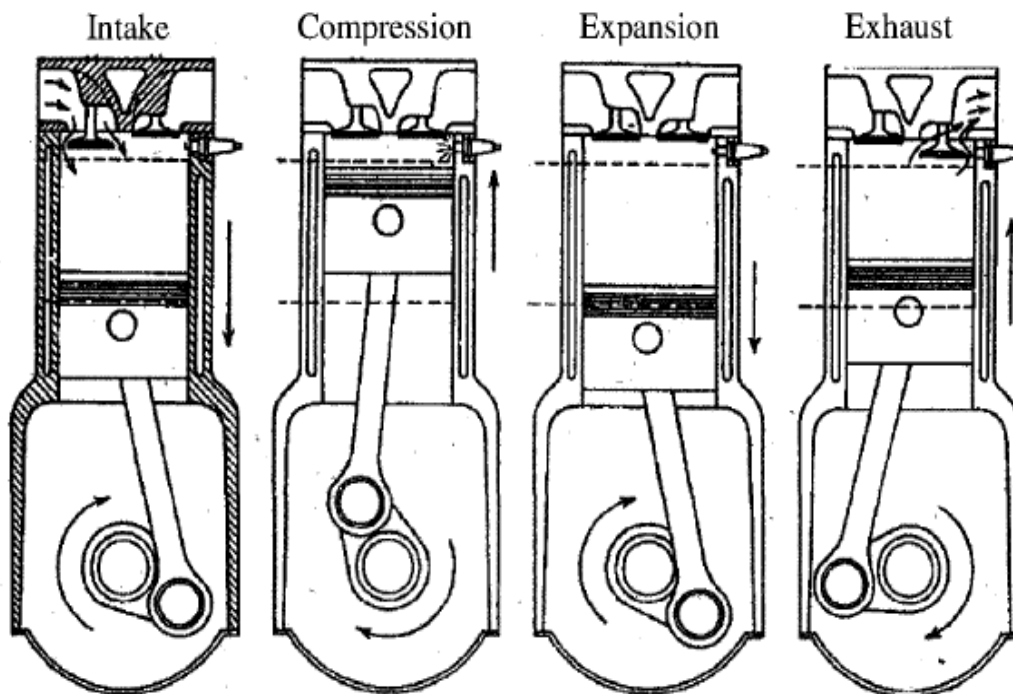


FIGURE 1: The Four Stroke Engine Cycle [1]

During the intake stroke, the piston begins at top-dead-center (TDC) and ends at bottom-dead-center (BDC). An air and gasoline mixture enters the cylinder through the intake valve and in some cases this valve opens slightly before the intake stroke begins to allow more air-fuel mixture into the cylinder. During the compression stroke, the intake and exhaust valves are closed and the mixture is compressed to a very small fraction of its initial volume. The compressed mixture is then ignited by a spark causing the pressure to rise very rapidly [12].

During the expansion stroke, the piston begins at (TDC). Due to the high pressure and temperature gases in the cylinder, the piston is now pushed down, causing the crank to rotate. As the piston approaches (BDC) the exhaust valve opens. During the exhaust stroke, the burned gases exit the cylinder due to the high cylinder pressure and low exhaust pressure and also due to the piston moving up towards TDC. The cycle starts again after the exhaust valve closes.

A complete engine cycle is divided into 720 crank angle degrees, where the crank angle is between the piston connecting rod at TDC and the connecting rod away from TDC. This means that the piston will move up and down in the cylinder two times during one complete engine cycle. Since there are two revolutions in one engine cycle, time duration (in seconds) of one engine cycle can be found given the rotations-per-minute (RPM). For example, at 1500 RPM, an engine cycle lasts 80 milliseconds (ms) and at 3000 RPM an engine cycle lasts 40 ms [13].

3. MODELLING OF ENGINE

In developing a valid engine model, the concept of the combustion process, abnormal combustion, and cylinder pressure must be understood. The combustion process is relatively simple and it begins with fuel and air being mixed together in the intake manifold and cylinder. This air-fuel mixture is trapped inside cylinder after the intake valve(s) is closed and then gets compressed [13].

When the air-fuel mixture is compressed it causes the pressure and temperature to increase inside the cylinder. Unlike normal combustion, the cylinder pressure and temperature can rise so rapidly that it can spontaneously ignite the air-fuel mixture causing high frequency cylinder pressure oscillations. These oscillations cause the metal cylinders to produce sharp noises called knock, which it caused to abnormal combustion.

The pressure in the cylinder is a very important physical parameter that can be analyzed from the combustion process. After the flame is developed, the cylinder pressure steadily rises, reaches a maximum point after TDC, and finally decreases during the expansion stroke when the cylinder volume increases. Since cylinder pressure is very important to the combustion event and the engine cycle in spark ignition engines, the development of a model that produces the cylinder pressure for each crank angle degree is necessary. A cylinder pressure model that calculates the total cylinder pressure over 720 crank angle degrees was created based upon the following formulation [12-13], [17]:

$$P_{cyl}(\theta) = P_m(\theta) + P_{net}(\theta) \quad (1)$$

where $P_{cyl}(\theta)$ is pressure in cylinder, $P_m(\theta)$ is Wiebe function, and $P_{net}(\theta)$ is motoring pressure of a cylinder. Air fuel ratio is the mass ratio of air and fuel trapped inside the cylinder before combustion starts. Mathematically it is the mass of the air divided by the mass of the fuel as shown in the equation below:

$$Air\ to\ Fuel = \frac{\dot{m}_{air}}{\dot{m}_{fuel}} \quad (2)$$

If the ratio is too high or too low, it can be adjusted by adding or reducing the amount of fuel per engine cycle that is injected into the cylinder. The fuel ratio can be used to determine which fuel system should have a larger impact on how much fuel is injected into the cylinder. Since a direct fuel injector has immediate injection of its fuel with significant charge cooling effect, it can have a quicker response to the desired amount of fuel that is needed by an engine [17].

4. CLASSICAL SLIDING MODE CONTROL

Sliding mode controller (SMC) is a powerful nonlinear controller which has been analyzed by many researchers especially in recent years. This theory was first proposed in the early 1950 by Emelyanov and several co-workers and has been extensively developed since then with the invention of high speed control devices[15-16].

A time-varying sliding surface $s(x, t)$ is given by the following equation [18-24]:

$$s(x, t) = \left(\frac{d}{dt} + \lambda\right)^{n-1} \ddot{x} = 0 \quad (3)$$

where λ is the constant and it is positive. A simple solution to get the sliding condition when the dynamic parameters have uncertainty is the switching control law:

$$U_{dis} = \hat{U} - K(\vec{x}, t) \cdot sgn(s) \tag{4}$$

Where the function of $sgn(S)$ defined as;

$$sgn(s) = \begin{cases} 1 & s > 0 \\ -1 & s < 0 \\ 0 & s = 0 \end{cases} \tag{5}$$

and the $K(\vec{x}, t)$ is the positive constant. To reduce or eliminate the chattering it is used the boundary layer method; in boundary layer method the basic idea is replace the discontinuous method by saturation (linear) method with small neighborhood of the switching surface. This replace is caused to increase the error performance [20-24].

$$B(t) = \{x, |S(t)| \leq \emptyset\}; \emptyset > 0 \tag{6}$$

Where \emptyset is the boundary layer thickness. Therefore, to have a smote control law, the saturation function $Sat(S/\emptyset)$ added to the control law:

$$U = K(\vec{x}, t) \cdot Sat(S/\emptyset) \tag{7}$$

Where $Sat(S/\emptyset)$ can be defined as

$$sat(S/\emptyset) = \begin{cases} 1 & (S/\emptyset > 1) \\ -1 & (S/\emptyset < -1) \\ S/\emptyset & (-1 < S/\emptyset < 1) \end{cases} \tag{8}$$

Based on above discussion, the control law for a multi degrees of freedom robot manipulator is written as [18-24]:

$$U = U_{eq} + U_r \tag{9}$$

Where, the model-based component U_{eq} is compensated the nominal dynamics of systems. Therefore U_{eq} can calculate as follows:

$$U_{eq} = [M^{-1}(P_m(\theta) + P_{net}(\theta)) + \dot{s}]M \tag{10}$$

Where

$$M^{-1} = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}^{-1} \quad M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}$$

5. DESIGN PROPOSED FUZZY SLIDING MODE CONTROLLER

The most important objective in fuzzy sliding mode controller (FSMC) is design sliding mode control combined to fuzzy logic systems to resolve most important problems in pure sliding mode controller. This research focuses on resolve the equivalent nonlinear dynamic sliding mode controller by use a new method. To compensate the nonlinearity of dynamic equivalent control some researchers is used model base fuzzy controller instead of classical equivalent controller. This technique was employed to obtain the desired control behavior with a number of information about dynamic model of system and a fuzzy switching control was applied to reinforce system performance. In contrast proposed methodology is used error based fuzzy instead of classical equivalent dynamic to have an acceptable performance and easy to implementation. According to the new method model free controller design is the basis of research so it's found that there is a big difference between this new method with the old one that was based on equivalent in order to undefined dynamic models compensation.

The most important objects in fuzzy sliding mode controller (FSMC) are applied fuzzy logic controller in sliding mode controller to solve equivalent problems in classical sliding mode controller. In proposed fuzzy sliding mode controller error based Mamdani's fuzzy inference system has considered with two inputs, one output and totally 49 rules instead of the dynamic equivalent part.

For both SMC and FSMC applications the system performance is sensitive to the sliding surface slope (λ). For instance, if large value of λ is chosen the response is very fast but the system is very unstable and conversely, if small value of λ is considered the response of system is very slow but the system is very stable. Therefore, calculation the optimum value of λ is the other important challenge works. A block diagram for proposed fuzzy sliding mode controller is shown in Figure 2. In this method a model free Mamdani's fuzzy inference system has considered based on fuzzy logic controller instead of equivalent control. In FSMC the equation can be written as;

$$U = U_{eq\ fuzzy} + U_r \tag{11}$$

As mentioned as Figure 2, as a summary the design of fuzzy like equivalent part based on Mamdani's fuzzy inference method has four steps , namely, fuzzification, fuzzy rule base and rule evaluation, aggregation of the rule output (fuzzy inference system), and defuzzification.

Fuzzification: the first step in fuzzification is determine inputs and outputs which, it has two inputs (e, \dot{e}) and one output (U_{fuzzy}). The inputs are error (e) which measures the difference between desired and actual inputs, and the change of error (\dot{e}) which measures the difference between desired and actual velocity and output is fuzzy equivalent estimator. The second step is chosen an appropriate membership function for inputs and output which, for simplicity in implementation and also to have an acceptable performance the researcher is selected the triangular membership function. The third step is chosen the correct labels for each fuzzy set which, in this research namely as linguistic variable. The linguistic variables for error (e) are; Negative Big (NB), Negative Medium (NM), Negative Small (NS), Zero (Z), Positive Small (PS), Positive Medium (PM), Positive Big (PB), and it is quantized in to thirteen levels represented by: -1, -0.83, -0.66, -0.5, -0.33, -0.16, 0, 0.16, 0.33, 0.5, 0.66, 0.83, 1 the linguistic variables for change of error (\dot{e}) are; Fast Left (FL), Medium Left (ML), Slow Left (SL), Zero (Z), Slow Right (SR), Medium Right (MR), Fast Right (FR), and it is quantized in to thirteen levels represented by: -6, -5, -0.4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, and the linguistic variables to find the output are; Large Left (LL), Medium Left (ML), Small Left (SL), Zero (Z), Small Right (SR), Medium Right (MR), Large Right (LR) and it is quantized in to thirteen levels represented by: -6, -5, -0.4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6.

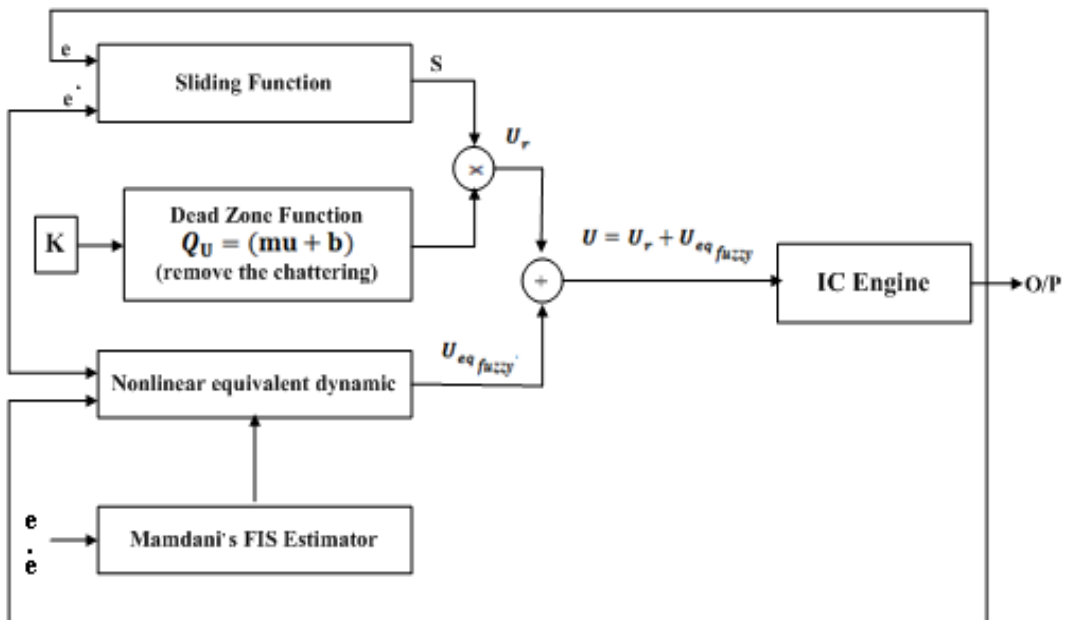


FIGURE 2: Block diagram of proposed fuzzy sliding mode controller

Fuzzy Rule Base and Rule Evaluation: the first step in rule base and evaluation is provide a least structured method to derive the fuzzy rule base which, expert experience and control engineering knowledge is used because this method is the least structure of the other one and the researcher derivation the fuzzy rule base from the knowledge of system operate and/or the classical controller. Design the rule base of fuzzy inference system can play important role to design the best performance of fuzzy sliding mode controller, that to calculate the fuzzy rule base the researcher is used to heuristic method which, it is based on the behavior of the control of IC engine suppose that two fuzzy rules in this controller are;

$$\begin{aligned}
 \mathbf{F.R}^1: & \text{IF } e \text{ is NB and } \dot{e} \text{ is FL, THEN } U \text{ is LL.} \\
 \mathbf{F.R}^2: & \text{IF } e \text{ is PS and } \dot{e} \text{ is FL THEN } U \text{ is ML}
 \end{aligned}
 \tag{12}$$

The complete rule base for this controller is shown in Table 1. Rule evaluation focuses on operation in the antecedent of the fuzzy rules in fuzzy sliding mode controller. This part is used **AND/OR** fuzzy operation in antecedent part which **AND** operation is used.

$\dot{e} \backslash e$	NB	NM	NS	ZE	PS	PM	PB
NB	PB	NB	NB	NM	NS	NS	ZE
NM	NB	NM	NM	NM	NS	ZE	PS
NS	NB	NM	NS	NS	ZE	PS	PM
ZE	NB	NM	NS	ZE	PS	PM	PB
PS	NM	NS	ZE	PS	PS	PM	PB
PM	NS	ZE	PS	PM	PM	PM	PB
PB	PS	PS	PM	PB	PB	NB	ZE

TABLE 1: Modified fuzzy rule base table

Aggregation of the Rule Output (Fuzzy inference): Max-Min aggregation is used to this work which the calculation is defined as follows.

$$\mu_U(x_k, y_k, U) = \mu_{U_{FR^i}}(x_k, y_k, U) = \max \left\{ \min_{i=1}^n \left[\mu_{R_{pq}}(x_k, y_k), \mu_{P_m}(U) \right] \right\}
 \tag{13}$$

Defuzzification: The last step to design fuzzy inference in our fuzzy sliding mode controller is defuzzification. This part is used to transform fuzzy set to crisp set, therefore the input for defuzzification is the aggregate output and the output of it is a crisp number. In this design the Center of gravity method (**COG**) is used and calculated by the equation 14.

$$\mathbf{COG}(x_k, y_k) = \frac{\sum_i U_i \sum_{j=1}^n \mu_{ij}(x_k, y_k, U_i)}{\sum_i \sum_{j=1}^n \mu_{ij}(x_k, y_k, U_i)}
 \tag{14}$$

Table 2 is shown the lookup table in fuzzy sliding mode controller which is computed by COG defuzzification method. These output values were obtained from trial and error after some manual adjustment to reach the best performance in fuzzy sliding mode controller. Table 2 has 169 cells to shows the fuzzy like equivalent part behavior. For instance if $e = 0$ and $\dot{e} = 0$ then the output or **fuzzy like equivalent torque = 6**. By comparing between the COG defuzzification and the equivalent part it found that this controller works well because it can be reducing the chattering and error with respect to eliminate the dynamic equation in equivalent part.

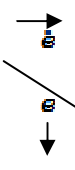
	Membership Function												
	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
-1	-5	-5	-5	-5	-3	-2	-1	0	1	2	3	3	3
-0.83	-5	-5	-4	-3	-2	-1	-1	0	1	2	3	3	4
-0.66	-5	-5	-4	-3	-2	-1	0	1	1	2	3	4	5
-0.5	-5	-4	-3	-2	-1	-1	0	1	2	3	3	4	5
-0.33	-6	-5	-3	-2	-1	-1	0	1	2	3	3	4	5
-0.16	-6	-5	-3	-2	-1	-1	0	1	2	3	4	5	6
0	-5	-5	-4	-3	-2	-1	0	1	2	3	5	5	6
0.16	-5	-4	-3	-3	-2	-1	0	1	2	4	5	5	6
0.33	-4	-4	-3	-3	-1	0	0	2	3	4	5	5	6
0.5	-3	-3	-2	-2	0	0	0	2	3	4	5	6	6
0.66	-2	-1	-1	0	0	1	2	3	4	5	5	6	6
0.83	-1	0	0	1	1	2	2	3	4	5	6	6	6
1	0	1	2	2	2	3	4	4	5	5	5	6	6

TABLE 2: COG lookup table in fuzzy sliding mode controller: applied to IC engine

6. RESULTS

PD Matlab-based sliding mode controller (PD-SMC) and PD Matlab-based fuzzy sliding mode controller (PD-FSMC) were tested to Step response trajectory. The simulation was implemented in Matlab/Simulink environment. Fuel ratio trajectory, disturbance rejection and error are compared in these controllers. It is noted that, these systems are tested by band limited white noise with a predefined 40% of relative to the input signal amplitude which the sample time is equal to 0.1. This type of noise is used to external disturbance in continuous and hybrid systems.

6.1 Fuel Ratio Trajectory

Figure 3 shows the fuel ratio in PD-SMC and PD-FSMC without disturbance for Step trajectory. The best possible coefficients in Step PD-FSMC are; $K_p = K_v = 30$, $\phi = 0.1$, and $\lambda = 6$ as well as similarly in Step PD-SMC are; $K_p = K_v = 10$, $\phi = 0.1$, and $\lambda = 8$.

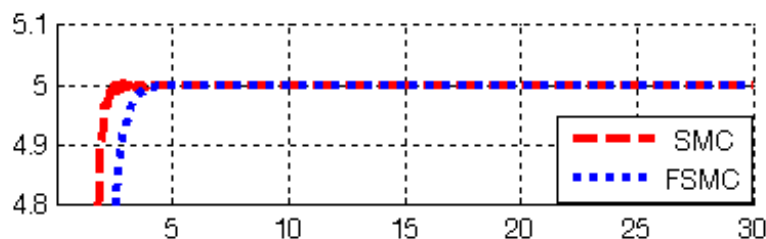


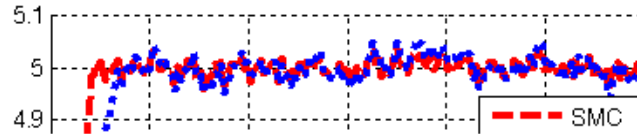
FIGURE 3: SMC Vs. FSMC: fuel ratio

By comparing step response, Figure 3, in PD-SMC and PD-FSMC, conversely the FSMC's overshoot (0%) is lower than SMC's (1%), the SMC's rise time (0.483 Sec) is dramatically lower than FSMC's (0.9 Sec); in addition the Settling time in FSMC (Settling time=0.65 Sec) is fairly lower than SMC (Settling time=1.4 Sec).

6.2 Disturbance Rejection

Figure 4 is indicated the power disturbance removal in SMC and FSMC. As mentioned before, SMC is one of the most important robust nonlinear controllers. Besides a band limited white noise

FIGURE 4: SMC Vs. FSMC: fuel ratio with external disturbance



with predefined of 40% the power of input signal is applied to the step SMC and FSMC; it found slight oscillations in trajectory responses.

Among above graph, relating to step trajectory following with external disturbance, SMC and FSMC have slightly fluctuations. By comparing overshoot, rise time, and settling time; FSMC's overshoot (**0.9%**) is lower than SMC's (**1.1%**), SMC's rise time (**0.48 sec**) is considerably lower than FSMC's (**0.9 sec**) and finally the Settling time in FSMC (**Settling time=0.65 Sec**) is quite lower than SMC (**Settling time=1.5 Sec**).

6.3 Errors in the Model

Although SMC and FSMC have the same error rate (refer to Table.3), they have oscillation tracking which causes chattering phenomenon at the presence of disturbances. As it is obvious in Table.1 FSMC is a SMC which estimate the equivalent part so FSMC have acceptable performance with regard to SMC in presence of certain and uncertainty. Figure 5 is shown steady state and RMS error in SMC and FSMC in presence of external disturbance. However both of SMC and FSMC have slight oscillation but FSMC in presence of uncertainty has better response.

<i>RMS Error Rate</i>	SMC	FSMC
Without Noise	1e-3	0.6e-3
With Noise	0.012	0.0012

TABLE 3: RMS Error Rate of Presented controllers

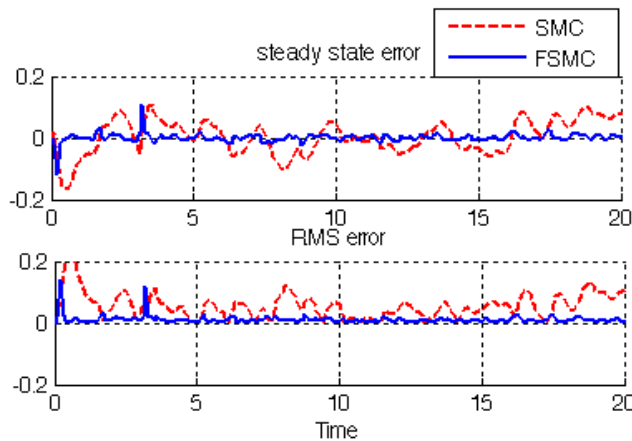


FIGURE 5: SMC Vs. FSMC: Steady state and RMS error in presence of external disturbance

In these methods if integration absolute error (IAE) is defined by (15), table 4 is shown comparison between these two methods.

$$IAE = \int_0^{\infty} |e(t)| dt \tag{15}$$

Method	FSMC	Traditional SMC
IAE	442.1	484.8

TABLE 4: Calculate IAE

7. CONCLUSION

Refer to the research, a fuzzy sliding mode control design and application to IC engine has proposed in order to design high performance nonlinear controller in the presence of uncertainties and external disturbances. Regarding to the positive points in sliding mode controller and fuzzy logic controller the output responses have improved. Fuzzy logic method by adding to the sliding mode controller has covered negative points. Obviously IC engine is nonlinear so this paper focuses on comparison between sliding mode controller and fuzzy sliding mode controller, to opt for better control method for the IC engine. Higher implementation quality of response and model free controller versus an acceptable performance in chattering, trajectory and error is reached by designing fuzzy sliding mode controller. This implementation considerably reduces the chattering phenomenon and error in the presence of uncertainties. As a result, this controller will be able to control a wide range of IC engine with a high sampling rates because its easy to implement versus high speed markets.

REFERENCES:

- [1] Heywood, J., "Internal Combustion Engine Fundamentals", McGraw-Hill, New York, 1988.
- [2] Ferguson, C., "Internal Combustion Engines: Applied Thermosciences", John Wiley & Sons, Inc., New York, 2001.
- [3] Guzzella, L., "Introduction to Modeling and Control of Internal Combustion Engine Systems" Springer, New York, 2004.
- [4] Ramos, J., "Internal Combustion Engine Modeling", Hemisphere Publishing corporation, New York, 1989.
- [5] Blair, G., "Design and Simulation of Four Stroke Engines", Society of Automotive Engineers, Warrendale, Pa, 1999.
- [6] G. Zhu, et al, "Closed-Loop Ignition Timing Control for SI Engines Using Ionization Current Feedback," IEEE Trans on Control Systems, pp. 416-427, May 2007.
- [7] I. Haskara, et al, "On Combustion Invariants For MBT Timing Estimation and Control," in ASME Internal Combustion Engine Division, 2004.
- [8] Frank L.Lewis. Nonlinear dynamics and control, Handbook, pages 51-70. CRC press, 1999.
- [9] Thomas R.Kurfess., "Dynamic plant and Automation Handbook", CRC press, 2005.

- [10] Lotfi A. Zadeh” Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic” *Fuzzy Sets and Systems* 90 (1997) 111-127
- [11] Lotfi.A.Zadeh”Fuzzy logic,Nural network, and Soft computing” *communications of the ACM*, March 1994, Vol.37.No.3
- [12] Dawson, J., “An experimental and Computational Study of Internal Combustion Engine Modeling for Controls Oriented Research” Ph.D. Dissertation, The Ohio State University, 2005.
- [13] Lee, B., “Methodology for the Static and Dynamic Model Based Engine Calibration and Optimization” Ph.D. Dissertation, The Ohio State University, 2005.
- [14] Okyak Kaynak, “Guest Editorial Special Section on Computationally Intelligent Methodologies and Sliding-Mode Control”, *IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS*, VOL. 48, NO. 1, 2001
- [15] Norsinnira Zainul Azlan and Johari Halim Shah Osman,” Modeling and Proportional Integral Sliding Mode Control of Hydraulic Manipulators”, *SCORed* 2006, 2006.
- [16] Soteris A. Kalogirou,” Artificial intelligence for the modeling and control of combustion processes: a review”, *Progress in Energy and Combustion Science*, science direct, 2003.
- [17] J. G. Rivard, "Closed-loop Electronic Fuel Injection Control of the IC Engine," in *Society of Automotive Engineers*, 1973.
- [18] F. Piltan, *et al.*, "Artificial Control of Nonlinear Second Order Systems Based on AFGSMC," *Australian Journal of Basic and Applied Sciences*, 5(6), pp. 509-522, 2011.
- [19] Piltan, F., *et al.*, 2011. Design sliding mode controller for robot manipulator with artificial tunable gain. *Canadian Journal of pure and applied science*, 5 (2): 1573-1579.
- [20] Piltan, F., *et al.*, 2011. Design Artificial Nonlinear Robust Controller Based on CTLC and FSMC with Tunable Gain, *International Journal of Robotic and Automation*, 2 (3): 205-220.
- [21] Piltan, F., *et al.*, 2011. Design Mathematical Tunable Gain PID-Like Sliding Mode Fuzzy Controller with Minimum Rule Base, *International Journal of Robotic and Automation*, 2 (3): 146-156.
- [22] Piltan, F., *et al.*, 2011. Design of FPGA based sliding mode controller for robot manipulator, *International Journal of Robotic and Automation*, 2 (3): 183-204.
- [23] Piltan, F., *et al.*, 2011. A Model Free Robust Sliding Surface Slope Adjustment in Sliding Mode Control for Robot Manipulator, *World Applied Science Journal*, 12 (12): 2330-2336.
- [24] Piltan, F., *et al.*, 2011. Design Adaptive Fuzzy Robust Controllers for Robot Manipulator, *World Applied Science Journal*, 12 (12): 2317-2329.

INSTRUCTIONS TO CONTRIBUTORS

The *International Journal of Engineering (IJE)* is devoted in assimilating publications that document development and research results within the broad spectrum of subfields in the engineering sciences. The journal intends to disseminate knowledge in the various disciplines of the engineering field from theoretical, practical and analytical research to physical implications and theoretical or quantitative discussion intended for both academic and industrial progress.

Our intended audiences comprises of scientists, researchers, mathematicians, practicing engineers, among others working in Engineering and welcome them to exchange and share their expertise in their particular disciplines. We also encourage articles, interdisciplinary in nature. The realm of International Journal of Engineering (IJE) extends, but not limited, to the following:

To build its International reputation, we are disseminating the publication information through Google Books, Google Scholar, Directory of Open Access Journals (DOAJ), Open J Gate, ScientificCommons, Docstoc and many more. Our International Editors are working on establishing ISI listing and a good impact factor for IJE.

The initial efforts helped to shape the editorial policy and to sharpen the focus of the journal. Starting with volume 5, 2011, IJE appears in more focused issues. Besides normal publications, IJE intend to organized special issues on more focused topics. Each special issue will have a designated editor (editors) – either member of the editorial board or another recognized specialist in the respective field.

We are open to contributions, proposals for any topic as well as for editors and reviewers. We understand that it is through the effort of volunteers that CSC Journals continues to grow and flourish.

IJE LIST OF TOPICS

The realm of International Journal of Engineering (IJE) extends, but not limited, to the following:

- Aerospace Engineering
- Biomedical Engineering
- Civil & Structural Engineering
- Control Systems Engineering
- Electrical Engineering
- Engineering Mathematics
- Environmental Engineering
- Geotechnical Engineering
- Manufacturing Engineering
- Mechanical Engineering
- Nuclear Engineering
- Petroleum Engineering
- Telecommunications Engineering
- Agricultural Engineering
- Chemical Engineering
- Computer Engineering
- Education Engineering
- Electronic Engineering
- Engineering Science
- Fluid Engineering
- Industrial Engineering
- Materials & Technology Engineering
- Mineral & Mining Engineering
- Optical Engineering
- Robotics & Automation Engineering

CALL FOR PAPERS

Volume: 6 - **Issue:** 1 - February 2012

i. Paper Submission: November 30, 2011

ii. Author Notification: January 01, 2012

iii. Issue Publication: January / February 2012

CONTACT INFORMATION

Computer Science Journals Sdn Bhd
B-5-8 Plaza Mont Kiara, Mont Kiara
50480, Kuala Lumpur, MALAYSIA

Phone: 006 03 6207 1607
006 03 2782 6991

Fax: 006 03 6207 1697

Email: cscpress@cscjournals.org

CSC PUBLISHERS © 2011
COMPUTER SCIENCE JOURNALS SDN BHD
M-3-19, PLAZA DAMAS
SRI HARTAMAS
50480, KUALA LUMPUR
MALAYSIA

PHONE: 006 03 6207 1607
006 03 2782 6991

FAX: 006 03 6207 1697
EMAIL: cscpress@cscjournals.org