A Localized Software-based Approach for Fault-Tolerant Ethernet (LSFTE)

Huy Thao Vu*, Se Mog Kim*, Anh Hoang Pham*, Jong Myung Rhee**

ABSTRACT

Nowadays, there are various networked systems with many computers. In most networked systems, a crucial objective is to keep transmitting and/or receiving data continuously even though failures exist. How can one make a computer continue transmitting and/or receiving data even when there are some errors on a link? Fault-Tolerant Ethernet (FTE) can be a solution to this question. In this paper, we propose a Localized Software-based Fault-Tolerant Ethernet (LSFTE). Our new approach fulfills the general FTE requirements. It takes advantage of redundant cable lines to maintain communication in a faulty environment. A software layer, which uses a simple and effective algorithm, is added above the LAN card driver software to detect and overcome faults. For our approach, there is no need to change the existing hardware or the end-use interfaces. Furthermore, the fault-detection time is reduced significantly compared to the conventional software-based approach.

keywords : fault tolerant Ethernet, Port Monitor, Port Switch, Localized Software-based, LSFTE

Ⅰ. Introduction

Recently, distributed systems have been based on popular network products to reduce the development cycle and cost and to achieve system interoperability.

Most networked systems, e.g., vehicles, military weapon systems and banking systems, need a net work with more reliability as well as high performance and low power. There liability of a system is defined as the ability to performand to maintain system operation evenina faulty environment. It is very difficult to have a system without having any faults. In addition, the initial designs of most networked systems did not take great care about failures [1-3].

Therefore, one of the main requirements for these systems is how to reduce and overcome failures. The less a system fails, the more reliable the system is. It is necessary to have a solution that keeps the systems working well even though failures take place. Furthermore, Ethernet is becoming the central part in networked systems. However, Ethernet was not originally designed to handle network faults. Hence, much effort has been made in research, development and standardization which have been trying to add fault- tolerant capabilities to

* Department of information and communication Engineering, Myongji University Rep. of Korea
(vuhuythao@yahoo.com, kimsemog@empal.com, anhph.cse@gmail.com)
**Corresponding author, Department of information and communication Engineering, Myongji University Rep. of Korea
(jmr77@mju.ac.kr)
Received : 2010년. 8월 31일, Revised : 2010년 9월 10일, Complete : 2010년. 9월 15일

Ethernet-based mission-critical networks. Among these, there have been some noteworthy approaches proposed to create a Fault- Tolerant Ethernet (FTE), such as software- based [4, 5], hardware-based [6], and hybrid [7] approaches. The hardware-based approach uses specialized hardware to detect and recover faults. However, since this approach employs proprietary products, it is not appropriate to use the hardware-based approach in a system which needs FTE modification in order to meet system requirement changes during operation. On the other hand, the software-based approach adds a software layer in order to control separate drivers. Although no special hardware is added to implement FTE, it often takes a long time to detect and recover faults. Taking a combination of the software-based and hardware-based approaches, the hybrid approach is a software implementation of the conventional hardware-based approach. Nonetheless, it depends on incoming packet and timeout intervals to detect a fault, which requires intensive probabilistic analysis to determine an efficient timeout interval.

In this paper, we present a novel approach called Localized Software-based Fault- Tolerant Ethernet (LSFTE). It is based on the rule: a network is healthy only if all of the nodes in the network are healthy. Our approach tries to keep all computers at a healthy status. We can also apply this approach to switches or routers to maintain them at a good status. Based on the rule, we concentrate on detecting and recovering local faults. By doing it locally, our approach removes any dependency on incoming packets. Furthermore, the time to detect a fault and for fail-over switching gets shortened. Since our approach is software-based, it does not need any changes to hardware. It also supports the network and transport protocol standards, e.g., TCP/IP. The key advantages of our proposed approach are:

1. Fast detection and recovery: Because LSFTE is a local algorithm, it takes a short time to detect and recover faults.

2. Simplicity: Since LSFTE is composed of a simple and effective algorithm, it is easy to design, validate and maintain in practice. By separating the algorithm into two functions, the approach is clearer.

3. Separation of mechanisms from algorithm: LSFTE executes two functions, namely network monitoring and network recovering. The two functions are named as the Port Monitor and Port Switch, which are related together, i.e., the Port Switch is based on the Port Monitor. The Port Monitor executes the failure detection mechanism and the Port Switch runs the configurable mechanism.

Again our LSFTE is a software-based approach. Therefore, it also has the advantages that software-based approaches have:

1. Flexibility and interoperability: Today, many distributed systems are built and run with Commercial-Off-The-Shelf (COTS) hardware (hub, switches and NICs) to be deployed easily and reduce the cost. Being based upon software, our approach does not need any changes of hardware. Therefore our approach supports not only failure detection and recovery, but also flexibility and interoperability.

2. Transparency: LSFTE runs like a service in the operating system. It is hidden to the end-users. It is also independent of other network applications. LSFTE and other applications can be executed simultaneously.

The remainder of this paper is structured as follows: Section 2 briefly reviews FTE. The architecture of the framework for FTE implementation is presented in section 3. Section 4 discusses the theory of operation for our approach. Then section 5 shows the implementation and evaluation of our approach.

We will discuss further about data loss in section 6. Finally we conclude our work in section 7.

II. Fault-Tolerant Ethernet

In order to understand the basics of FTE and the approach to FTE implementation, we consider the network model used for FTE as shown in Figure 1. The model is a single network with redundant cables. In this model, each computer has two cables, e.g., each computer is equipped with two LAN cards. One cable is active at any time and the other one is on standby. The active cable is used in normal communication. The standby cable will be changed to be active when the active one fails

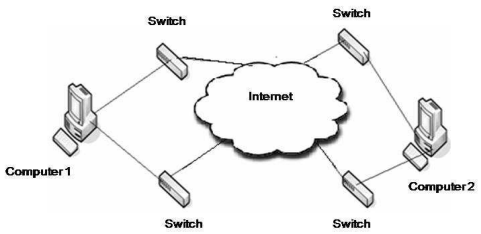

Figure 1 Network model for FTE

As mentioned before there are three typical approaches for FTE: software, hardware and hybrid. Each of them has its own strengths but also contains weaknesses.

1. Hardware-based approach

In this approach, the network adapter usually includes Ethernet transceivers. It may have two or more ports. It can detect local faults, e.g., faults on the cable, the RJ45 and the adjacent switch in the active port and can switch to another port. The approach does not concern itself with determining whether or not the other nodes or the networks connect with the switches. The biggest advantage of the approach is that the process of detecting a fault and switching to another port is very fast. It takes of the order of a few hundred milliseconds at most. However, this approach needs specific propriet ary hardware, e.g., a new network adapter integrated with transceivers in side.

2. Software-based approach

The key advantage of this approach is that it can detect faults not only in the local system but also in the network, i.e., it can discover the fault on a channel to a destination. If the channel has a fault, the software switches to another channel. Furthermore, the approach does not need new hardware or a new network adapter. We can use two COTS single-port Ethernet Network Interface Cards (NICs) instead of designing specific LAN cards. However, it takes more time to detect and recover a fault.

3. Hybrid approach

The hybrid approach is the software implementation of the conventional hardware. The hybrid approach takes advantage of the software and hardware approaches. It is faster than the software approach. In addition, it does not require new hardware. However, it depends on incoming packets and timeout intervals to detect faults and to switch to another port.

4. Our approach

Our approach is also a software-based approach. However, it runs as a permanent service in the operating system. It can detect and recover faults automatically. Furthermore, it utilizes system calls in the operating system.

Our approach operates as an independent service that can detect local faults and switch to another port in a few hundred milliseconds. Furthermore, our approach is based on the rule: the network is healthy only if all of the nodes in a network are healthy. In our approach, each node can detect and recover faults by itself as shown in Figure 2. Since it is a software-based approach, it can leverage advantages of previous software-based approaches. However, it can prevent the problem of needing a long processing time. Moreover, it also does not depend on incoming packets and timeout intervals to detect and recover faults.

Basically, there are two tasks to be considered to implement an FTE software or hardware:

1. Detecting the failure.
2. Performing the fail-over switching.

In the following sections, Ⅲ and Ⅳ, we will explain the two tasks in detail.

Ⅲ. The Proposed Framework

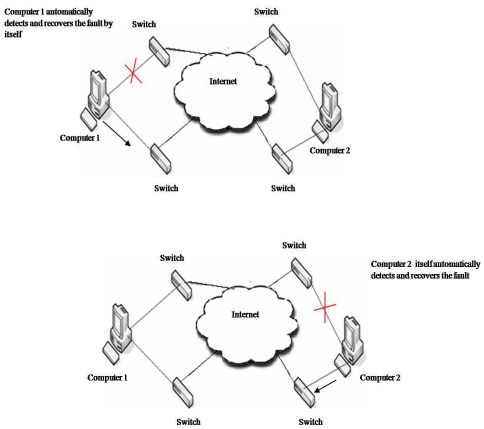

Figure 2. Self-healing network model

Figure 3 presents our proposed Framework Architecture. We use two LAN cards in a station. As a result, we have two MAC addresses, i.e., MAC 1 and MAC 2 and two drivers, i.e., Driver 1 and Driver 2 in Figure 3, respectively. In this architecture, we refer to the OSI network model with seven layers. We only concentrate on the data link layer. We add some middleware called the LSFTE layer on the top of the data link layer.

It is above the two drivers and directs the two LAN cards by using these drivers. In other words, the software is an interface between the data link layer and the network layer. It resides in the kernel. The software monitors the two LAN cards and overcomes faults by automatically changing the active LAN card to the standby LAN card. This unique approach does not require any change to the COTS hardware (switch, hub, Ethernet physical link and Network Interface Card (NIC)) or software (Ethernet driver and protocol), yet. It is transparent to application software. Our approach concentrates on detecting and recovering the fault in the local host.

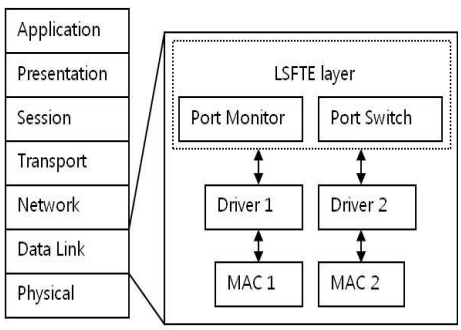

Figure 3. Proposed Framework Architecture

Ⅳ. Theory of Operation

1. Algorithm

LSFTE is software that runs independently of other network applications. It includes two parts: a Port Monitor and a Port Switch. The Port

Monitor checks if the link status of an active LAN card is good. If there is a fault, the Port Switch exchanges the function of the two LAN cards – changing the status of the standby LAN card to active and the status of the active LAN card to standby; i.e., the standby LAN card is used to transmit and receive data in place of the active LAN card. The algorithm used in our approach is shown in Figure 4. Here it is assumed that the computer uses two LAN cards, i.e., Network Adapter 1 (NA1) and Network Adapter 2 (NA2). We also assume that the Network Adapter 1 is active and the Network Adapter 2 is on standby at the beginning.

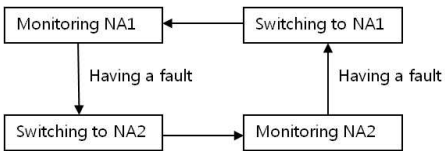

Figure 4. Algorithm

1.1 Port Monitor

The Port Monitor is one part of LSFTE. It monitors the status of the active LAN card. There are several techniques to check if the link status of a LAN card is good. One of them uses the ECHO message like the "Ping" command. With the ECHO message, the sender can get the reply message from the receiver within a very small amount of time. Based on the reply message, the sender can know exactly the link status of the active LAN card. For another way, we can use system calls in the operating system to investigate the link status. Because we only need to discover the link status locally, we could do this with an ECHO message to the Default-Gateway of the local host. In addition, these methods require a small amount of time for checking the link status.

1.2 Port Switch

The Port Switch is the other part of LSFTE. It takes action based on the result from the Port Monitor. If the Port Monitor detects any errors with an active LAN card, the Port Switch swaps two LAN cards; i.e., the active LAN card's status is changed to standby and the standby LAN card is assigned to the active status. The combination of statuses and given values for the two LAN cards is illustrated in Table 1. In the table NA 1 is an abbreviation for Network Adapter 1 and NA 2 denotes Network Adapter 2. Actually, LSFTE makes the computer use only one LAN card – the enabled LAN card – at a time. The other one is a back-up for using when the enabled LAN card is broken.

Table 1. Status table

NA 1	NA 2	Active	Stanby
OK	OK	NA 1	NA 2
OK	Not OK	NA 1	NA 2
Not OK	OK	NA 2	NA 1

2. Flowchart

Here we introduce flowcharts for our approach. Firstly, we show in Figure 5 the flowchart for the case in which each computer has only two LAN cards. After that, we demonstrate in Figure 6 the data flow for the case in which each computer is equipped with more than two LAN cards.

2.1 A computer with two LAN cards

In this case, each computer has two LAN cards, i.e., Network Adapter 1 and Network Adapter 2. At the beginning, LSFTE assigns an active status to Network Adapter 1 and gives Network Adapter 2 standby status. After that, LSFTE checks the link status of the active

LAN card. If there is any fault, LSFTE continues checking the standby LAN card. If the standby LAN card has a good link status, LSFTE does the switching step – changing the status of the active LAN card to standby and the status of the standby LAN card to active, i.e., the standby LAN card is used to transmit and receive data in place of the previously active LAN card.

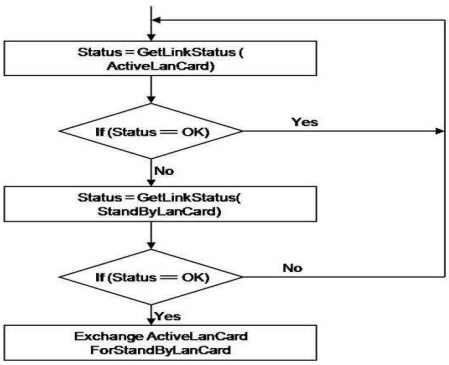

Figure 5. Flow chart for each computer with two LAN cards

2.2 A computer with more than two LAN cards

In this situation, each computer has more than two LAN cards. One card is the active LAN card and the others are the standby LAN cards. Periodically, LSFTE verifies the active LAN card. If it detects any fault in the active LAN card, it attempts to find a good standby LAN card among all the standby LAN cards. The good standby LAN card refers to a standby LAN card with a good link status, i.e., we can use the standby LAN card to send and receive data. After that, LSFTE swaps the active LAN card and the good standby LAN card – altering the status of the active LAN card to standby and the status of the standby LAN card to active, i.e., the standby LAN card is used to transmit and receive data instead of the active LAN card. In both cases, LSFTE tries to find a good standby LAN card – a good backup LAN card to replace the active LAN card with a fault. LSFTE makes an effort to reduce the time for fail-over switching. It also decreases the data loss. However, in the case of each computer with more than two LAN cards, we meet two key issues: how many LAN cards are good for one computer and how can we utilize the standby LAN cards? In this paper, we do not go further into these issues. We only focus on the case with two LAN cards.

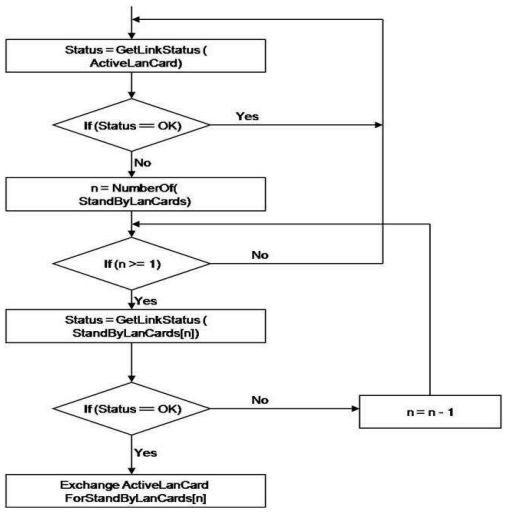

Figure 6. Flow chart for each computer with more than two LAN cards

V. EXPERIMENTS

1. Implementation

We implemented LSFTE as a service that runs automatically in the kernel of the operating system (OS). The service always checks two LAN cards and does switching if it detects any fault in the active LAN card. The implementation structure is shown in Figure 7. As mentioned, we used two LAN cards for each computer.

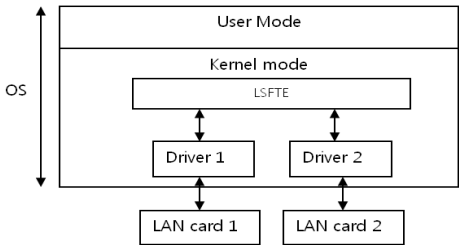

Figure 7. Implementation Infrastructure

2. Evaluation

2.1 Time Estimation

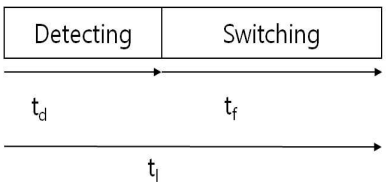

Figure 8. Time Estimation

As we mentioned above, the time for fail-over switching including:

- Time to detect the failure (tf)
- Time to do switching (td)

Totally, the time we to enable switch from the active LAN card to the standby LAN card in a computer is tl=tf+td.

2.2 Configuration

As shown in table 2, we use a computer that is equipped with the design. We also implemented a checking software to evaluate.

We also implemented some checking software to evaluate the LSFTE service on Windows XP. We installed two network adapters in our computer: Adapter 1 and Adapter 2. Then we demonstrated our approach by using the checking software with the following initial and testing situations.

Table 2. Computer Setup

Parameters	Value
CPU Speed	1.8 GHz
RAM	2GHz
Dual Ethernet LAN card	1Gbps
OS	Windows XP Service Pack 2
Framework	.Net Framework 2.0

2.3 Initial Situation

At the beginning, Adapter 1 is connected and Adapter 2 is disconnected, i.e., the status of Adapter 1 is active and the status of Adapter 2 is standby, as shown in Figure 8. Additionally, as shown in the figure, each adapter has one control button named "Disable". These control buttons are used to disable each adapter during our testing operations.

Figure 9. Initial Condition

2.4 Testing Situation

First, the computer is in the initial situation. Then we disable Adapter 1 by using its control button. The LSFTE service automatically detects the error on Adapter 1 and swaps the status of Adapter 1 and the status of Adapter 2 as shown in Figure 9. After swapping, the status of Adapter 2 is active and the status of Adapter 1 is standby, i.e., Adapter 2 is used to transmit and receive data instead of Adapter 1. The time for fail-over switching in this case is found to

be about 0.5 seconds.

Figure 10. Fail-over Switching

In our experiment, we need a few hundred milliseconds to check the link status of the active LAN card by using the library inside the OS. We also need a few hundred milliseconds to disable or enable the LAN cards in a faulty situation, i.e., the active LAN card is disabled and the standby LAN card is enabled. After we do experiment 20times, we get the average fail-over switching time of 0.415 second and the variance of 0.003 second. This is the key advantage of our approach. The advantage originates from the algorithm in which LSFTE does fail-over switching locally by directly using the library inside the OS.

Ⅵ. DISCUSSION

As presented in the previous sections, we know how to control two LAN cards. We completed the theory and conducted an experiment, as mentioned above, to decrease the switching time. However, one of the key questions is how to reduce the packet loss? We would like to present a theoretical method of how to make the loss become smaller. We also use the result from the LSFTE experiment in the method. Nonetheless, here we will just deal with theory of the method; an experiment for this method is beyond the scope of this paper.

In the method, we combine the results from the fail-over switching part and the new part with a useful technique. The useful technique is to queue the sending packets, i.e. the sending packets will be kept in a FIFO queue and data is transmitted based on the status of the connection on the active LAN card. Figure 10 shows the data flow of the new model that we designed by using the results from the LSFTE experiment in previous sections. In almost OS, the system is divided into two modes: User mode and Kernel mode. When we talk about User mode, we mean the applications made by application programmers. Kernel mode contains system calls and kernel functions that application programmers do not change. Kernel mode is like a bridge between the software in User mode and the hardware. As shown in Figure 10, the application in User mode wants to send some data to another computer. The application calls a system call to interact with kernel functions. The data will be sent from functions in User mode to functions in Kernel mode. In Kernel mode, the data will traverse over some layers, e.g., the TCP layer, IP layer and Data Link layer. In the Data Link layer, we place an LSFTE module that we call the Controlling LSFTE module to do fail-over switching as mentioned in previous sections. When the data is in here, the data will be kept in a queue by another LSFTE module that we call the LSFTE sending module. This module asks the Controlling LSFTE module about the connection in the active LAN card before sending the data to the lower layer. If the connection on the active LAN card is good, i.e. if it is connected well, the LSFTE sending module gets data from the queue and sends the data to the active LAN card. Conversely if there is no connection on the active LAN card, i.e. the connection is disconnected, so that we cannot send the data over the active LAN card then the Controlling LSFTE module does fail-over switching. In the

fail-over switching time, the application still continues to transmit data to the destination because it does not know anything that is happening in the Kernel mode and the data will be queued by the LSFTE sending module. When the Controlling LSFTE module finishes doing the fail-over switching, it notifies the connection status as good to the LSFTE sending module and the LSFTE sending module can now send the data over the connection. Here, data will be stored in a special queue that is a FIFO (First In First Out) queue. That means the data coming first will be transmitted first.

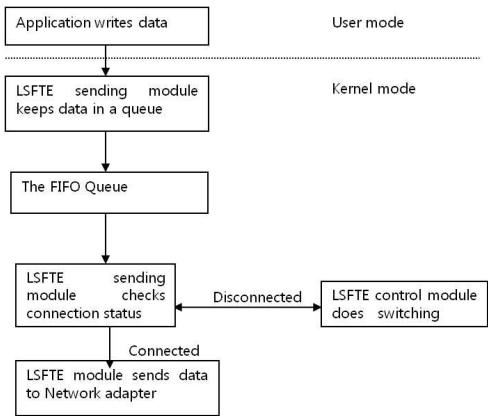

Figure 11. Data flow for the less packet loss algorithm

Although the proposed method will reduce the data loss in the fail-over switching time, we have a trade-off here. Each sending packet needs a delay time in the queue. However, we believe that the delay time is not significant because we will implement these LSFTE modules in the kernel. As is known, we can customize the Linux kernel, so we can create the LSFTE modules inside the kernel. Another reason is that computers are very powerful, so the time for local processing is very short. As a result, the delay time will be rapidly reduced on a fast computer. On the other hand, in most mission-critical systems, it is essential to minimize the packet loss. The new model can satisfy these requirements well.

Ⅶ. CONCLUSION

In this paper, we presented a novel approach called LSFTE that meets the typical requirements for a Fault Tolerant Ethernet. With our approach, a networked system can keep communication functioning, since it takes advantage of the redundant cable lines with LSFTE. For the LSFTE implementation a software layer, which uses a simple and effective algorithm, has been added above existing drivers' line ports to detect and recover faults. LSFTE does not require any change to COTS products (i.e., COTS hubs, switches, NICs and drivers) or to the end-user interfaces. Furthermore the fault-detecting time is significantly reduced compared to the conventional software approach. In addition, the fault-detection and recovery do not depend on the packet arrivals. It is expected that we can expand our approach for the multiple LAN cards case. Moreover, we discuss a new way to implement the new model by using LSFTE to reduce the loss of packets. Possible future study areas can be: (1) how many LAN cards for each computer is best? (2) How do we effectively find a good card among the standby LAN cards? and (3) how to reduce the packet loss?

Ⅷ. ACKNOWLEDGMENTS

This work was supported by the Agency for Defense Development (ADD) and the Defense Acquisition Program Administration (DAPA) under Grant No. ADD-07-06-02.

REFERENCES

[1] A. Romanovsky. "A Looming Fault Tolerance Software Crisis", ACM SIGSOFT Software Engineering, pp. 1-4, 2007

[2] M. Bruntink, A. van Deursen, and T. Tourwe. "Discovering Faults in Idiom-Based Exception Handling", International Conference on Software Engineering, ICSE 2006, pp.242- 251, 2006.

[3] F. Cristian. "Exception handling", Dependability of Resilient Computers. Blackwell Scientific Publications, pp.68-97, 1989.

[4] J. Huang, S. Song, L. Li, P. Kappler, R. Freimark, J. Gustin, and T. Kozlik. "An open solution to fault-tolerant Ethernet: design, prototyping, and evaluation", IEEE International Performance, Computing and Communications Conference, IPCCC'99, pp. 461 - 468, 1999.

[5] S. Song, J. Huang, P. Kappler, R. Freimark, J. Gustin, and T. Kozlik. "Fault-tolerant Ethernet for IP-based process control: Ademonstration", Dependable Systems and Networks, DSN, 2000.

[6] PMC675/RM675 family of next generation dual intelligent Ethernet Controllers, http://gefanuc.com.

[7] A. P. Hoang, J. M. Rhee, S. M. Kim, and D. H. Lee. "A Novel Approach for Fault-Tolerant Ethernet Implementation", Fourth International Conference on Networked Computing and Advanced Information Management, NCM'08. pp. 58-61, 2008.

저자약력

Huy Thao Vu **Non-member**

Feb. 2007 : Graduated Technology, Computer Science and Engineering of HoChiMinh University

March. 2009~Now : M.A in Dept. Communication Engineering of Myongji University

<Interested> Network Communication, Fault Tolerant System.

Semog Kim **Non-member**

Feb. 1998 : Graduated Dept. of Electronic Engineering of PuKyong National University

March 2007~Now : (PH.) D. course in Dept. of Communication Engineering of Myongji University

Aug. 1999~ Aug. 2001 : General Manager in NeoDigital

Jan. 2002~March. 2008 : Director in SungWon Telecom

Jan. 2010~Now : Manager in RingNet

<Interested> Fault Tolerant System, HFC(Hybrid Fiber Coaxal) System

Anh Pham Hoang **Non-member**

Feb. 2005 : Graduated Technology, Computer Science and Engineering of HoChiMinh University

Feb. 2010 : M.A in Dept. of Communication Engineering of Myongji University

March. 2010~Now : (PH.) D. course, Dept. of Communication Engineering of Myongji University

<Interested> Network Communication, Fault Tolerant System.

JongMyung Rhee **Member**

Feb. 1976 : Graduated Dept. of Electronic Engineering of Seoul National University
Feb. 1978 : M.A. in Dept. of Electronic Engineering of Seoul National University
Dec. 1987 : Ph. D. in Dept. Electron Engineering of North Carolina State University
March 1978~Dec. 1997 : Principal Researcher in ADD(Agency for Defense Development)
Sep. 1992~Aug. 1994 : Associate Professor(Adjunct) in Dept. Information Communications Engineering of ChungNam National University
Dec. 1997~Oct. 1999 : Director in Laboratory of Dacom
Oct. 1999~Oct. 2005 : Vice-president(CTO) in Hanaro Telecom(Now, SK Broadband)
Sep. 2006~Now : Professor in Dept. of Communication Engineering of Myongji University and Leader of Industry-Academic Cooperation Foundation of Myongji University

<Interested> Network Communication, Fault Tolerant System.