A Light-weight and Dynamically Reconfigurable RMON Agent System

Jun-Hyung Lee*, Zin-Won Park*, and Myung–Kyun Kim*

Abstract: A RMON agent system, which locates on a subnet, collects the network traffic information for management by retrieving and analyzing all of the packets on the subnet. The RMON agent system can miss some packets due to the high packet analyzing overhead when the number of packets on the subnet is huge. In this paper, we have developed a light-weight RMON agent system that can handle a large amount of packets without packet loss. Our RMON agent system has also been designed such that its functionality can be added dynamically when needed. To demonstrate the dynamic reconfiguration capability of our RMON agent system, a simple port scanning attack detection module is added to the RMON agent system. We have also evaluated the performance of our RMON agent system on a large network that has a huge traffic. The test result has shown our RMON agent system can analyze the network packets without packet loss.

Keywords: *Network management, RMON agent system, Dynamic reconfiguration*.

1. Introduction

In traditional SNMP-based centralized network managements, a network manager collects information about network statistics from all of the agents and performs a specific management function, so it becomes a bottleneck due to its great processing load and a large amount of network traffic to the manager. The network management using RMON agent systems can reduce both the processing overhead of the network manager and the amount of traffic to the manager [1]. A RMON agent system locates on a subnet and collects network traffic statistics on the subnet, and transmits the information to the central manager when requested. The management information to be collected by the RMON agent systems are defined in RMON 1, and RMON 2 MIBs [2, 3]. RMON 2 MIB defines the management information on the network traffic statistics based on the upper layer (layer 3 or more) addresses, while RMON 1 MIB defines the network traffic statistics based on the data-link layer addresses. The RMON agent system captures all the packets sent on the subnet and analyzes the packet header to collect the management information defined in the RMON MIBs. So, when the number of packets is large, the processing(analyzing the packet header) load of the RMON agent system becomes great, and the RMON agent system may miss the packets that arrive while processing the packets already received. The traditional SNMP-based RMON agent system cannot easily extend its functionality dynamically [4].

Manuscript received October 5, 2005; accepted August 7, 2006.
The authors would like to thank Ministry of Commerce, Industry and Energy and Ulsan Metropolitan City for their support of this research through the Network-based Automation Research Center (NARC) at University of Ulsan.
Corresponding Author: Myung–Kyun Kim
* School of Computer Engineering and Information Technology, University of Ulsan (mkkim@ulsan.ac.kr)

In this paper, we describe the development of a RMON agent system that can be used on a large network without packet loss. We have implemented the management information only in RMON 2 MIB, and the module structure of the RMON agent system is designed to minimize the packet processing overhead. Our RMON agent system is also designed such that a new functionality can be added dynamically while running, when needed. Our RMON agent system is light-weight in terms of the following two aspects: The first one is that we have designed a data structure including all of the basic RMON 2 MIB information such that redundant information can be eliminated as much as possible and the data structure can be maintained in memory; the second one is that we have designed the module structure to be light, and the network manager can start the RMON agent system just with basic functionality only, and after that, a new functionality can be added when needed. To demonstrate the dynamic reconfiguration capability of our RMON agent system, a simple port scanning attack detection module is added to the RMON agent system. We have also evaluated the performance of our RMON agent system on a large network that has a huge traffic. The test result has shown our RMON agent system can analyze the network packets without packet loss.

Section 2 describes our RMON MIB data structure and the module architecture of our RMON agent system and the description of each of the modules. Section 3 describes the implementation of the RMON agent system and a port scanning attack detection module that is added to show the dynamic reconfiguration capability. Section 4 describes the performance evaluation of our RMON agent system, and in Section 5, we conclude our paper.

2. Design of our RMON Agent System

In this section, we describe our RMON data structure

and the overall architecture of our RMON agent system. The RMON agent system collects the network management information on the subnet and sends it to the network manager when requested. The architecture of our RMON agent system is shown in Fig. 1. The communication between the network manager and the RMON agent system is done using HTTP, which allows an easy access from the network manager to the RMON agent system.

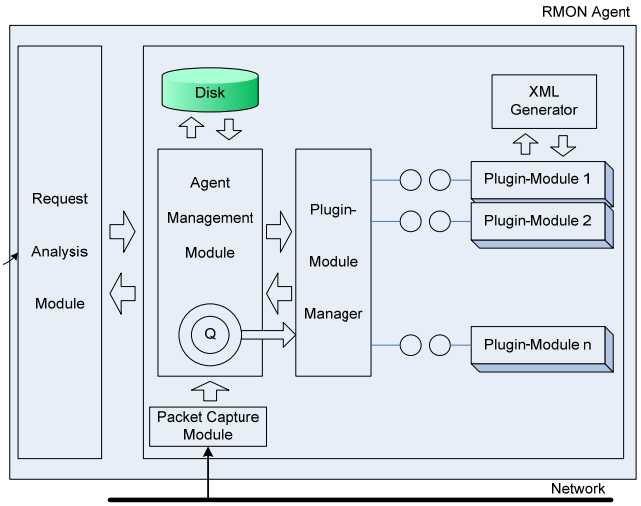


Fig. 1. The architecture of our RMON agent system

2.1 RMON MIB Data Structure

We have implemented RMON 2 MIB only in this paper. RMON 2 MIB defines the management information to be collected by a RMON agent system. RMON 2 MIB consists of 10 groups and, each of which consists of a control table and several data tables. The control table defines a function that describes how to collect the required management information in a data table. In RMON 2 MIB, there was a number of information that was defined redundantly in many groups. For example, IP addresses of source and destination nodes are defined in many groups, so we have to store and maintain such kind of information in multiple places redundantly. In this paper, we have designed a RMON data structure to reduce the redundancy as much as possible. The reduced RMON data structure can be maintained in main memory to minimize the access time to the required table. We have represented each of RMON control tables as an independent table, and all of the data tables have been connected using a hash table whose access key is a source IP address. Fig. 2 shows a RMON data structure represented as a linked list using a hash table.

2.2 Request Analysis Module

The request analysis module interprets the requests from the manager and sends the result to the agent management module. The management operation requested from the manager includes the request of management information, the addition of a plug-in module that collects new management information, or the deletion of a plug-in module, etc. The request analysis module returns the response from the agent management module back to the manager. The RMON agent system returns the requested management information in XML form, which can be handled easily by the web browser.

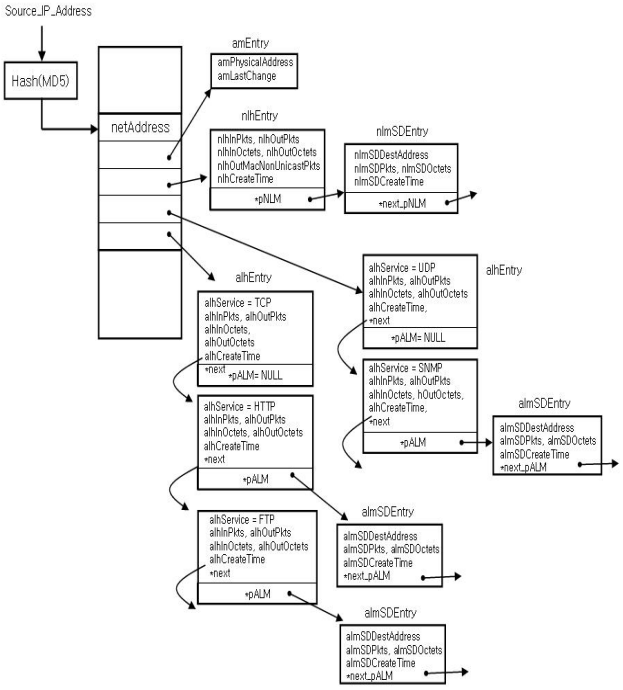


Fig. 2. RMON MIB data structure

2.3 Packet Capture Module

The packet capture module retrieves all of the raw packets from the subnet and stores to the packet buffer queue. We used the libpcap packet capture library to retrieve the packets. To minimize the overhead of capturing and copying packets, the number of bytes to be captured in each packet is set to only include the protocol header of the packet. The packets stored in the packet buffer queue are shared among the packet capture module and the plug-in modules which analyze the packets to collect management information.

2.4 Agent Management Module

The agent management module performs the core functions of the RMON agent system such as the following:

- Control to all of the modules in the RMON agent system that is executed in a separate thread;
- Add, delete, or update a plug-in module through the plug-in manager module;
- Distribute the packets stored in the packet buffer queue among the plug-in modules through the plug-in manager module;

- Store the management information collected by the RMON agent periodically to the disc storage.

2.5 Plug-in Manager Module

The plug-in manager module controls the plug-in modules which analyzes the packets and collects necessary management information. It also delivers the packets received from the agent management module to the plug-in module. To minimize the packet copying overhead, only the pointer to the packet in the packet buffer queue is delivered. The type of messages delivered from the plug-in manager module to the plug-in modules is shown in Table 1.

Table 1. Message type delivered from the plug-in manager module to the plug-in modules

Message	Function
InitModule	Initialize the plug-in module
StartModule	Start the plug-in module
StopModule	Suspend the plug-in module
CloseModule	Terminate the plug-in module
AddPacket	Add a packet to the plug-in module
GetData	Request management information
SendMessage	User-defined control message

2.6 XML Generation Module

The XML generation module receives management information from a plug-in module and converts it to a XML document to be sent to the network manager [5]. The agent management module receives the XML documents from the plug-in modules and combines them into a complete XML document to be delivered to the manager via the request analyzer module or to be stored to the disc storage. Fig. 3 shows an example of XML documents generated by the XML generation module.

```
<?xml version="1.0"?>
<RMON2>
<Thread id=0>
 <addressMap><Address IP="1.2.3.4" ... /></addressMap>
 <nlHost><Address IP="1.2.3.4" ... /></nlHost>
 <nlMatrix><Source IP="1.2.3.4" ...><Dest IP="1.2.3.5" /></Source> ... </nlMatrix>
<alHost>
<Source IP="1.2.3.4" ... >
<Service Port="80" ... />
...
</Source>
...
</alHost>
 <alMatrix>
<Source IP="1.2.3.4" ... >
<Service Port="80" ... >
 <Dest IP="1.2.3.5" ... />
...
</Service>
</Source>
...
</alMatrix>
</Thread>
</RMON2>
```

Fig. 3. XML document generated by the XML generation module

2.7 Plug-in Module

The plug-in module which is executed in a separate thread analyzes the packets retrieved by the packet capture module and collects the required management information. We can set accordingly the number of plug-in modules that collect RMON management information in order to minimize the number of packet losses during analyzing.

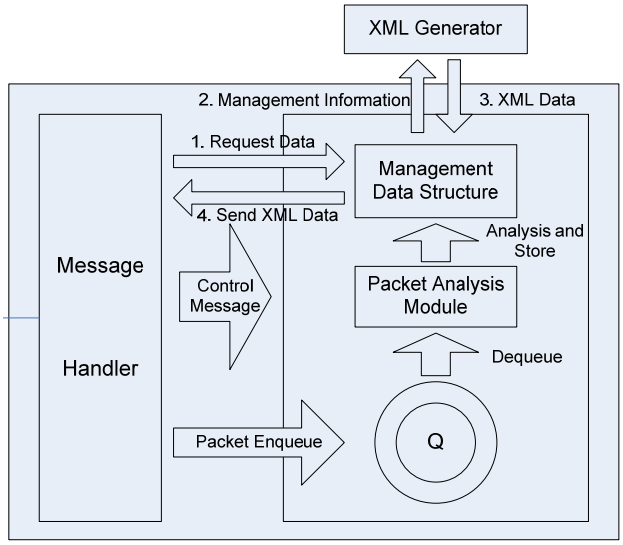


Fig. 4. Structure of the plug-in module

Table 2. Types of Dynamic RMON Operations.

RMON operations	Description
GetConfiguration	Request configuration information of a plug-in module
GetData	Request management information
InsertPlugin	Insert a plug-in module
DeletePlugin	Delete a plug-in module
UpdatePlugin	Update a plug-in module
SendMessage	Send user-defined data

We can add the functionality of the RMON agent system by preparing the plug-in module that executes the function, and simply adding it through the plug-in manager module. For the dynamic control (addition, suspension, or deletion) of the added plug-in module, the plug-in module must have the structure as shown in Fig. 4. The plug-in manager module controls the plug-in module according to the messages shown in Table 1. The message handler of the plug-in module parses the message from the plug-in manager module and performs the required operation. The packet analyzing module analyzes the packets in its packet queue. To minimize the packet copying overhead, there are only pointers to the actual packets in the packet queue of the plug-in module. If the message from the plug-in manager module is 'GetData', the message handler sends the collected management information to the XML generation module and requests to convert it into the XML document. The message handler sends the XML document

to the agent management module though the plug-in manager module.

2.8 Control Message Between The Network Manager and The RMON Agent System

The communication between the manager and the dynamic RMON agent system is done by HTTP. We have added a ISAPI [6] module 'RMON' to the web server for the dynamic RMON operation. The URL to access the dynamic RMON agent system has the following form:

http://rmon_host/RMON ?command=<commandString >&arg1=<argument1>&arg2=<argument2>& ...

'command' specifies the RMON operation to be performed by the dynamic RMON agent system. The type of operation to be performed is shown in Table 2.

2.9 Packet queue management

The packets captured by the packet capture module are stored in the packet buffer queue in the agent management module. The packets are needed by the plug-in modules to analyze and collect necessary management information. The packets are distributed by the plug-in manager module to the packet queues in the plug-in modules. To minimize the overhead due to packet copying, the plug-in manager module copies to the packet queue in the plug-in modules only the pointers to the actual packet in the packet buffer queue. Thus, the packets in the packet buffer queue are shared among the plug-in modules by the pointers in the packet queues as shown in Fig. 5. Each packet in the packet buffer queue has a reference counter which denotes the number of plug-in modules sharing the packet. After the plug-in module processes a packet pointed by the pointer in its packet queue, the reference counter to the packet is decremented. When the reference counter becomes 0, the packet is deleted from the packet buffer queue.

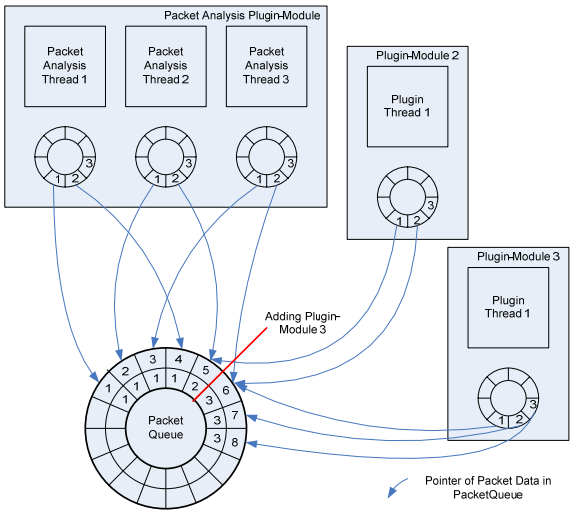
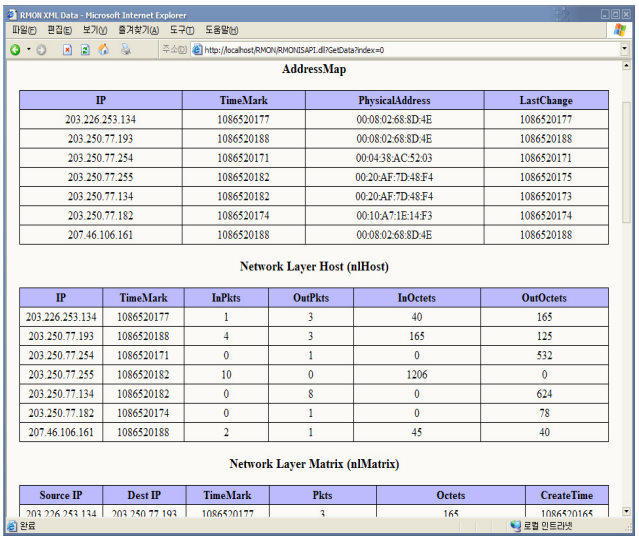


Fig. 5. Sharing of packets among plug-in modules

3. Implementation of Our RMON Agent System

We have implemented our RMON agent system using Visual C++ on Windows XP, and have used 'winpcap' packet capture library [6] for packet capturing. The management information is returned from the RMON agent in XML document. Fig. 6 shows the result returned from the RMON agent for the request of management information. We have used a low-level standard C file I/O library to improve generating XML files instead of Microsoft's XML parser APIs.



AddressMap

IP	TimeMark	PhysicalAddress	LastChange
203.226.253.134	1086520177	00:08:02:68:8D:4E	1086520177
203.250.77.193	1086520188	00:08:02:68:8D:4E	1086520188
203.250.77.254	1086520171	00:04:38:AC:52:03	1086520171
203.250.77.255	1086520182	00:20:AF:7D:48:F4	1086520175
203.250.77.134	1086520182	00:20:AF:7D:48:F4	1086520173
203.250.77.182	1086520174	00:10:A7:1E:14:F3	1086520174
207.46.106.161	1086520188	00:08:02:68:8D:4E	1086520188

Network Layer Host (nlHost)

IP	TimeMark	InPkts	OutPkts	InOctets	OutOctets
203.226.253.134	1086520177	1	3	40	165
203.250.77.193	1086520188	4	3	165	125
203.250.77.254	1086520171	0	1	0	532
203.250.77.255	1086520182	10	0	1206	0
203.250.77.134	1086520182	0	8	0	624
203.250.77.182	1086520174	0	1	0	78
207.46.106.161	1086520188	2	1	45	40

Network Layer Matrix (nlMatrix)

Source IP	Dest IP	TimeMark	Pkts	Octets	CreateTime

Fig. 6. Request result of management information.

The functionality of our RMON agent system can be extended dynamically by simply adding a new plug-in module. To show the dynamic reconfiguration capability of our RMON agent system, we have implemented a simple port scan detection plug-in module and added it to our RMON agent system. Port scanning attack is done by a malicious user to find a system's weak point for future attacks. There are many ways of detecting port scanning attacks [7, 8]. In this paper, we have implemented a simple port scanning attack detection to simplify the implementation of the plug-in module. Fig. 7 shows the flow of the port scan detection module. The agent maintains a host list table that sends packets to the host. Each entry of the host list table contains the source IP address, the packet inter-arrival time among successive packets from the source, and the number of ports to which the source sends packets. We have classified a packet as a port scanning attack packet from a malicious host if the packet inter-arrival time from the host is less than T_THRESHOLD (time threshold) and the number of ports to which the host sends packets is greater than PC_THRESHOLD (port count threshold).

The port scan detection plug-in module is dynamically added to the RMON agent system. Fig. 8 shows the result when the RMON agent system detects the port scan attacks.

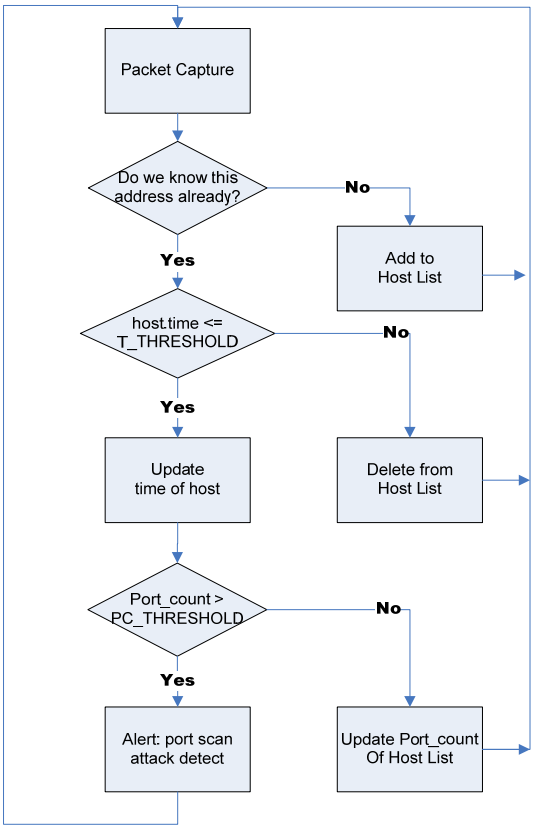


Fig. 7. The flow of port scan detection module

Fig. 8. An example showing port scan attack detection

4. Performance Evaluation

We have performed two experiments of our RMON agent system using a computer that has a Pentium 4-1.7GHz single CPU and 512MBytes main-memory.

We have performed the experiments for 2 minutes on 10Mbps and 100Mbps networks, respectively. In the experiments, a large number of packets were generated using FTP applications among many clients and servers. We have evaluated the resource utilization of the RMON agent system to see the number of packet losses and the processing overhead of the RMON agent system. Table 3 shows the resource utilization of the RMON agent system. As shown in the table, there were no packet losses on both numbers, and CPU utilization was less than 10 % on the 10 Mbps network, and less than 20% on the 100 Mbps network.

Table 3. Resource Utilization

Bandwidth	Drop	CPU (%)	Memory	Network Utilization
10Mbps	0	5~10	8,900Kb	90%
100Mbps	0	10~20	9,800Kb	70%

Fig. 9 shows the maximum number of packets remained in the shared packet queue according to the number of packet analysis threads. As shown in the Fig., for a given number of packet analysis threads, the number of packets remained in the shared packet queue has increased for some time, but the maximum number of packets in the queue was saturated to a certain value (about 110 packets in the case of 1 thread, and about 45 packets in the case of 5 threads, etc.). In this result, we can see the required packet queue size according to the number of packet analysis threads, and select the best tradeoff between the number of packet analysis threads and the amount of packet queue when the amount of traffic on the network has been expected.

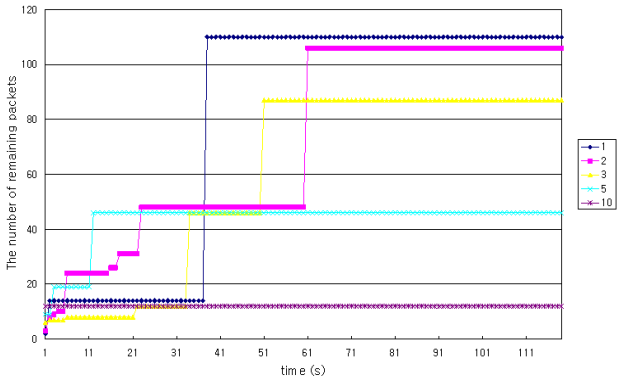


Fig. 9. The number of packets in the queue

5. Conclusion

We have described the development of a dynamic RMON agent system whose functionality can be added dynamically when needed. The RMON agent system was designed to have a low processing overhead in order to be used effectively on a large network that has a huge amount of traffic. The packet capture module and the packet analysis modules (the plug-in modules) are separated to be executed in independent threads, and the number of packet analysis modules can be conFig.d according to the amount of traffic on the network in order to minimize the packet loss due to the packet processing overhead. The plug-in modules can also be added dynamically when the network manager needs. To show the dynamic reconfiguration capability of our RMON agent system, we have developed and added a simple port scanning attack detection plug-in module to the agent system. The performance evaluation result of our RMON agent system demonstrates that is has

no packet loss and shows a low processing overhead (CPU load less than 20 % on a 100 Mbps network with 70 % network traffic load). The number of packets in the shared packet queue was shown to decrease as the number of packet analysis modules increases, and the number of packets in the packet queue remains saturated to a certain value, which shows the required packet queue size according to the number of packet analysis modules.

We plan to extend our RMON agent system in order to have the network intrusion detection capability by adding a plug-in module that detects network intrusion and notifies the event to the manager.

References

[1] W. Stallings, "SNMP, SNMPv2, SNMP v3, and RMON 1 and 2," Addison-Wesley, 1998.
[2] S. Waldbusser, Remote Monitoring Management Information Base (RFC 1757), Feb. 1995.
[3] S. Waldbusser, Remote Monitoring Management Information Base Version 2 (RFC 2021), Jan. 1997.
[4] The Simple Times, Agent Extensibility, Vol. 4, No 2, Apr. 1996.
[5] Microsoft, http://msdn.microsoft.com/library/xml
[6] Microsoft, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/iissdk/iis/isapiextensions.asp
[7] NetGroup, Politecnico di Torino, http://winpcap.polito.it/default.htm
[8] Sys-Security Group, http://www.sys-security.com/archive/papers/Network_Scanning_Techniques.pdf

Jun-Hyung Lee
Jun-Hyung Lee received the M.S. degree in School of Computer Engineering and Information Technology of University of Ulsan, Korea in 2005. He is now working at Initech in Korea. His research interest is in real-time communications in industrial communication networks and security in wireless networks.

Zin-Won Park
Zin-Won Park received the M.S. degree in School of Computer Engineering and Information Technology of University of Ulsan, Korea in 2004. He is now working at NetSecure Technology in Korea. His research interest is in real-time communications in industrial communication networks and security in wireless networks.

Myung-Kyun Kim
Myung-Kyun Kim received the M.S. and Ph.D. degrees in Department of Computer Science in KAIST of Korea in 1986 and 1996, respectively. From 1989 to 1998, he worked at Woosuk University in Jeon-Ju, Korea. He is now at School of Computer Engineering and Information Technology of University of Ulsan. He is interested in real-time communications in industrial communication networks, real-time communications on wireless networks, and security in wireless networks.