Journal of the Korea Academia-Industrial cooperation Society
Vol. 19, No. 4 pp. 642-649, 2018

https://doi.org/10.5762/KAIS.2018.19.4.642
ISSN 1975-4701 / eISSN 2288-4688

다중 제어 장치 연결을 위한 Modbus 응용 Protocol 설계

송정호, 김보헌, 김황래*
공주대학교 컴퓨터공학부

A Design of A Modbus Application Protocol for Multiple SCU Connections

Jeong-Ho Song, Bo-Hun Kim, Hwang-Rae Kim*
Division of Computer Engineering, Kongju National University

요 약 Modbus 프로토콜은 하나의 마스터 제어 유닛에 여러 대의 슬레이브 제어 유닛을 연결할 경우 가장 널리 사용되는 프로토콜이다. 그러나 Modbus 프로토콜에서는 SCU의 상태 값을 읽는 서로 다른 서비스 요청 메시지에 대해 결과 값만을 응답하기 때문에 어떤 전송 메시지에 대한 응답인지 식별할 수 없었다. 따라서 MCU 프로그램 작성 시 하나의 서비스 요청 메시지를 전송하고 이를 처리한 후에 다음 메시지를 전송하도록 프로그램을 작성하였다. 본 논문에서는 위 문제를 해결하기 위해 메시지의 전송 순서를 확인할 수 있는 Index 코드 및 응답한 메시지의 서비스 유형을 판단하기 위한 Service 코드를 추가한 Modbus 응용 프로토콜을 설계하고 이를 제안하였다.
실험 결과, MCU에서 전송한 서비스 요청 메시지에 대한 처리가 완료되지 않은 상태에서 다음 서비스 요청 메시지의 전송이 가능하였으며, 통신 에러 발생 시 에러 검색 알고리즘을 이용해 어떤 종류의 에러가 발생했는지 식별할 수 있었다. 또한, 다중의 동시적인 서비스 요청의 경우에 전송 메시지들의 처리시간이 기존의 Modbus 프로토콜보다 약 66.2% 향상되었다.

Abstract The Modbus protocol is the most widely used protocol for connecting multiple slave control units (SCUs) in the master control unit. However, the Modbus protocol does not identify which SCU response message corresponds to the service request message. Therefore, when using the Modbus protocol, one service request message can be transmitted after processing the previous message. In this paper, we propose a Modbus application protocol with an index code and a service code to solve the problem. As a result of experimentation, it is possible to transmit the next service request message without processing the previous service request message. And when a communications error occurs, it is possible to identify what type of error occurred using an error detection algorithm. Also, with simultaneous multiple service requests, the proposed protocol can improve processing time by about 66.2%, compared to the original Modbus protocol.

Keywords : IoT, Master-Slave Communication, Modbus, Protocol, Remote Control

1. 서론

최근 네트워크 통신 기술과 센서 제어 기술의 혁신적인 발전과 변화는 IoT가 빠르게 우리 생활과 산업전반에 파급되는 에너지를 제공하고 있다[1-3]. 산업 장비에서 사용되는 각종 제어 장치들은 일반적으로 서비스를 처리하고 그 결과를 제공하는 슬레이브 제어 유닛(SCU : Slave Control Unit)으로 사용된다. 이런 SCU들은 PC, PLC 또는 Embedded 제어 시스템 등과 같은 마스터 제어 유닛(MCU : Master Control Unit)이 요청하는 온도, 속도 및 압력 제어 등 다양한 서비스를 처리하기 위해 통신 프로토콜을 이용하여 정보 취득 및 제어를 수행한다[4].

Modbus 프로토콜은 다양한 종류의 SCU를 연결하기

*Corresponding Author : Hwang-Rae Kim(Kongju National Univ.)
Tel: +82-41-521-9227 email: plusone@kongju.ac.kr
Received January 24, 2018 Revised February 19, 2018
Accepted April 6, 2018 Published April 30, 2018

위해 일반적으로 사용되는 프로토콜로써 Address와 Function 코드 메시지 포맷을 이용하여 하나의 통신 회선을 통해 여러 SCU들에 대한 서비스를 처리할 수 있다. 또한, SCU에 전송된 메시지에 대한 응답 메시지가 항상 존재하기 때문에 응답 메시지를 통해 서비스 요청 메시지가 정상적으로 전달되었는지를 확인할 수 있다. 그러나 SCU의 응답 메시지는 요청한 서비스에 대한 처리 결과 값만을 반환하기 때문에 어떤 종류의 서비스 요청에 대한 응답인지 식별할 수 없었다. 따라서 MCU 통신 프로그램 작성 시 하나의 서비스 요청 메시지를 전송하고 해당 메시지 처리가 완료된 후에 다음 메시지를 전송할 수 있었다.

본 논문에서는 이런 문제점을 개선하기 위해 에러 발생 위치 및 응답 메시지에 대한 식별이 용이하고 서비스 처리 속도를 향상시킬 수 있도록 기존 Modbus 프로토콜에 Index 코드와 Service 코드를 추가한 응용 프로토콜을 설계하고 이를 구현하였다.

2. 관련 통신 기술

2.1 Modbus 프로토콜 종류 및 기능

다양한 SCU 장치들을 연결할 수 있도록 표준화된 Modbus 프로토콜은 전 세계적으로 널리 보급되어 사용되고 있는 자동화 프로토콜의 하나로서, RS-232/422/485 및 Ethernet 디바이스를 지원한다. 또한 PC, PLC, DCS, HMI, 계측기, 등의 수많은 공업 기기 및 국가 주요 핵심 기반 시설을 관리하는 SCADA 시스템에 사용되고 있으며[5-7], 전송 방식으로는 ASCII 모드와 RUT 모드가 있다. Fig. 1은 Modbus 프로토콜의 메시지 포맷을 나타낸다[8].

1) Ascii Mode

Start	Address	Function	Data	LRC	End
1 char	2 char	2 char	0 up to 2 x 252 char(s)	2 char	2 char CR LF

2) RTU Mode

Start	Address	Function	Data	LRC	End
≥ 3.5char	8 bits	8 bits	N x 8 bits	16 bits	≥ 3.5char

Fig. 1. Send message Format of Modbus Protocol

Table 1은 Modbus 프로토콜에서 정의된 Function 코드를 나타내는 것으로, 각 Function 코드를 이용하여 SCU들의 상태 취득 및 제어 명령을 수행할 수 있다[9].

Table 1. Function Code Definition

Description	Hex Code	Sub Code
Read Discrete Inputs	02	
Read Coils	01	
Write Single Coil	05	
Write Multiple Coils	0F	
Read Input Register	04	
Read Holding Registers	03	
Write Single Register	06	
Write Multiple Registers	10	
Read/Write Multiple Registers	17	
Mask Write Register	16	
Read FIFO queue	18	
Read File recode	14	
Write File recode	15	
Read Exception status	07	
Diagnostic	08	00-18,20
Get Com event counter	0B	
Get Com Event Log	0C	
Report Server ID	11	
Read device Identification	2B	
Encapsulate Interface Transport	2B	13,14
CAN Open General Reference	2B	

2.2 Modbus 응용 프로토콜

Modbus 프로토콜은 SCU 제조업체에서 제공하는 메모리 맵의 레지스터 값을 쓰거나 읽음으로써, SCU의 정보를 취득하거나 제어를 수행할 수 있기 때문에, 숫자 코드만을 이용하는 Modbus 프로토콜의 단점을 보완하기 위해 일부 SCU 제조업체에서는 별도로 작성한 Modbus 응용 프로토콜을 추가로 제공하기도 한다.

1) Protocol Format

Start Char	Address	Command	Data	Check Sum	End Char
STX 0x02	1 ~ 99	Reference of Command		Sum Addr to Data	CR LF 0x0D 0x0A

2) Command List

Command	Descrip tion
RSD	D-Register Continuous Read
RRD	D-Register Random Read
WSD	D-Register Continuous Write
WRD	D-Register Random Write
•••	•••
AMI	Information of Controller

Fig. 2. Modbus application protocol example

Fig. 2는 SCU 제조업체에서 제공하는 Modbus 응용 프로토콜의 예를 나타내는 것으로[10-11], 대부분의 Modbus 응용 프로토콜은 시작문자와 끝 문자를 재 정의하고 Function 코드를 대신하여 문자열로 인식할 수 있도록 하는 Command 코드를 별도로 제공하고 있다. 그러나 여전히 SCU의 응답 메시지만으로는 어떤 서비스 요청에 대한 응답인지 식별할 수 없고 연속적인 서비스 요청에 대한 처리가 어렵다.

3. Modbus 응용 프로토콜 설계

3.1 응용 프로토콜 설계

본 논문에서 제안하는 프로토콜은 기존 Modbus 프로토콜에 두 개의 필드를 추가하였다. Index 코드를 추가하여, 통신 장애 등으로 인한 에러 검출 및 서비스 요청 메시지를 연속적으로 전송할 수 있도록 하였으며, Service 코드를 추가함으로써 SCU의 응답 메시지가 어떤 서비스 요청에 대한 응답 메시지인지 식별할 수 있도록 하였다. Fig. 3은 본 논문에서 설계한 프로토콜포맷을 나타낸다.

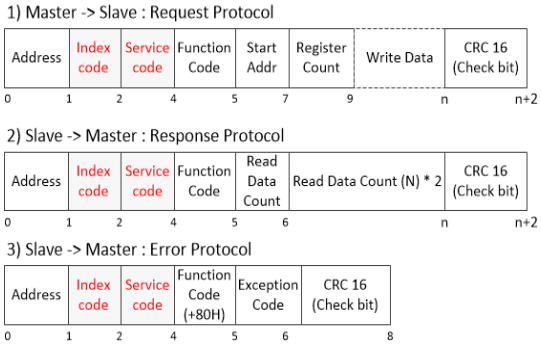


Fig. 3. Proposed Protocol Format

3.2 통신 에러 검색 알고리즘 설계

Table 2는 서비스 요청 메시지에 대한 에러 검색을 위해 Index 코드번호와 메시지 정보 및 전송 시간 등을 정의한 테이블이다.

Table 2. Index Code Definition Table

Index No	Send Message	Kind of Receive	Send Time Tick Count
0	Message #0	1	500
1	Message #1	1	700
2	Message #2	1	800
...			
255	Message #255	1	5000
0	Message #256	0	5100
...			
n	Message #idx	1	6000

테이블에서 Index No 항목은 전송 메시지의 Index 코드를 나타내며, 0~255까지 Loop 형태로 설정한다. 예를 들어, 첫 번째 전송된 메시지는 0을 설정하고 그 다음 전송되는 메시지는 이전 전송 메시지의 Index 코드 값에 1을 증가시킨 값을 순차적으로 부여한다. 단, 이전 메시지의 Index 코드가 255인 경우 그 다음 전송 메시지의 Index 코드는 0으로 설정하여 메시지 전송 Index 코드가 중복되지 않도록 한다.

Kind of Receive 항목은 전송 메시지에 대한 상태정보를 나타내는 것으로, -1 (미 전송), 0 (전송 시 / 미 응답 상태), 1 (응답 완료), 2 (에러 응답) 등의 값으로 정의하였으며, Send Time Tick Count는 전송 시점에 현재 시스템의 Tick Count를 설정하여 전송된 메시지가 얼마동안 미 응답 상태인가를 측정하는데 사용하였다.

```
*Define List
 - CUR_RECEIVE_STEP : Last Receive Index (Initialize value = 0)
 - _ERR_TIME_OUT_ : Send Message Timeout define (default : 1000msec)
 - GET_Kind_of_Receive() : Kind of Receive Value (in Table)
 - GET_TimeElapsed() : Measurement of elapsed time (unit : msec)
*Return Value : Error is greater than or equal to 0, No error is -1

Sub Get_ErrorSearch() as INTEGER
 INTEGER iIndexCode = 0
 FOR I = (CUR_RECEIVE_STEP + 256) TO (CUR_RECEIVE_STEP + 1) STEP -1
  iIndexCode = I MOD 256
  IF GET_Kind_of_Receive(iIndexCode) = 0 THEN
     IF GET_TimeElapsed(iIndexCode) > _ERR_TIME_OUT_THEN
        RETURN iIndexCode
     END IF
  ELSE IF GET_Kind_of_Receive(iIndexCode) = 2 THEN
     RETURN iIndexCode + 256
  END IF
 NEXT I
 RETRUN -1
End Sub
```

Fig. 4. Error Search Algorithm

Fig. 4는 Table 2의 Index Code 정의 테이블을 이용한 통신 에러 검색 알고리즘에 대한 함수 처리 과정을 나타내는 의사코드이다. 정상은 -1, SCU의 미 응답 에러는 0~255까지의 Index 코드 값, SCU에서 에러를 응답

한 경우 해당 Index 코드에 256을 더한 256부터 511까지의 값을 반환한다.

Fig. 4의 Get_ErrorSearch() 함수에서 에러 검색은 현재 수신된 메시지의 Index 코드인 CUR_RECEIVE_STEP을 기준으로 최근에 전송된 메시지 순으로 전체 테이블의 메시지를 검색한다. GET_Kind_of_Receive() 함수는 Table 2의 Kind of Receive 값을 검색하는 함수로 반환된 값이 0인 경우 GET_TimeElapsed() 함수를 이용하여 전송메시지의 전송 경과 시간을 측정하고, 측정된 시간이 설정된 통신 타임아웃 시간보다 크면 해당 Index 코드를 반환한다. 또한, GET_Kind_of_Receive() 함수에서 반환된 값이 2인 경우 SCU에서 보낸 에러를 응답한 메시지로 해당 Index 코드에 256을 더한 값을 반환하고, 에러가 없으면 -1을 반환한다.

3.3 통신 클래스 설계

Fig. 5는 본 논문에서 설계한 통신 클래스의 구성을 나타내는 것으로, MCU 응용 프로그램 영역, 각 SCU 제조업체에서 제공하는 다양한 서비스를 처리하기 위한 SCU 제조업체별 처리 클래스 영역 및 SCU와 통신을 수행하는 통신 클래스 영역으로 구성하였다.

동작 과정으로는 MCU 응용 프로그램에서 SCU 처리 클래스를 이용하여 SCU의 정보를 취득하거나 제어를 수행하는 함수를 호출하면 SCU 처리 클래스 에서는 프로토콜 포맷에 맞춰 변환한 메시지를 내부에 상속된 통신 클래스로 전달한다. 통신 클래스에서는 해당 메시지를 큐에 등록하고, 이후 등록된 메시지는 통신 클래스 내부의 Send Thread에 의해 SCU로 메시지를 전송하게 된다.

또한, Send Thread에서는 전송 시점에 Index Code Definition Table (ICDT)을 작성하고, 일정 주기 간격으로 ICDT의 정보를 이용하여 통신 에러를 검색하도록 한다. 통신 에러 발생 시 에러 정보와 함께 EVENT를 통해 MCU 응용 프로그램 내의 MainFrame.cpp에 전달한다.

MainFrame.cpp 내의 Event 처리 함수 내에서 전달받은 메시지는 각 SCU 처리 클래스에 전달하며, SCU 처리 클래스에서는 에러 형태에 따라 메시지 박스 등을 통해 알람을 발생하거나 메시지를 재전송할 수 있다.

SCU에서 응답한 메시지는 수신 스레드인 Receive Thread에서 메시지 구문 분석을 통해 수신된 메시지를 취득한 후, EVENT를 통해 MCU 응용 프로그램 내의 MainFrame.cpp에 전달한다. MainFrame.cpp 내의 Event 처리 함수에서는 수신된 메시지를 각 SCU 처리 클래스에 전달한다. SCU 처리 클래스에서는 수신된 메시지의 Index 코드 및 Service 코드를 이용하여 서비스 유형을 판단 한 후, 해당 서비스 요청에 따라 수신된 값을 설정하거나 정보로 이용할 수 있다.

4. 실험 및 분석

본 실험은 기존 Modbus 프로토콜과 제안한 프로토콜의 처리 성능을 비교·분석하기 위해 혈액 검사 시 사용되는 진공 채혈관 조립 설비의 운영 환경을 토대로 실험을 실시하였으며, ① MCU의 연속적인 서비스 요청 시 SCU의 처리 과정에 대한 비교 실험, ② SCU 장치의 서비스 처리 에러 또는 통신 중 발생하는 에러에 대한 처

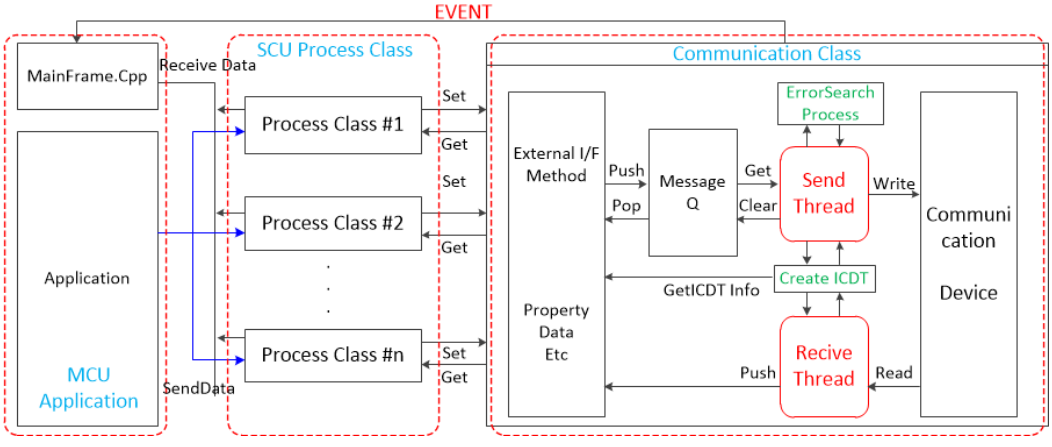


Fig. 5. Communication Class Diagram

리 과정과 전체적인 서비스 처리 시간을 비교 실험하였다.

4.1 실험 환경 및 구성

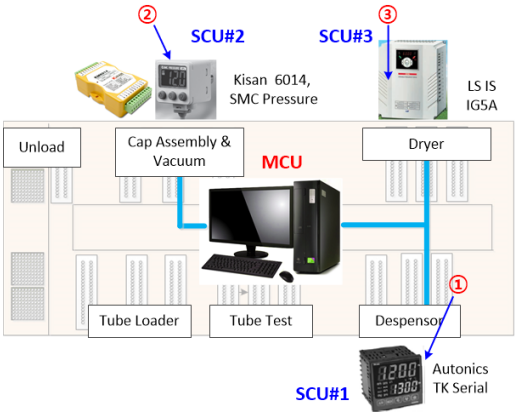


Fig. 6. Experimental environment & configuration

Fig. 6은 실험에서 사용된 진공 채혈관 조립 설비의 시스템 구성으로, SCU 장치는 약품 주입 공정에서 온도를 제어하기 위해 사용하는 온도 제어기, Dryer 공정에서 사용되는 인버터, 일정량의 혈액량을 주입하기 위해 사용되는 진공압력 제어기 등으로 구성된다. 각 SCU는 Table 3과 같이 주소를 정하여 실험을 진행하였다.

Table 3. SCU Information in Vacuum Blood Equipment

SCU	Address	Usage explanation
Temperature	01	Temp Control & Monitoring
Pressure	02	Vacuum pressure control.
Inverter	03	Air volume & speed Control

제안한 프로토콜에서 Index 코드는 MCU 응용 프로그램에서 서비스 요청 메시지 전송 시 통신 클래스 내부에서 자동으로 설정하였으며, 본 실험에서 사용된 Service 코드는 Table 4와 같이 정의하여 실험을 진행하였다.

Table 4. Service code list

Svc code	Description	Svc code	Description
00	Read PV	03	Read vacuum
01	Read SV	04	Set the Vacuum
02	Set the SV	05	Read air speed
		06	Set the air speed

4.2 응답 메시지 비교 실험

본 실험에서는 서론에서 제시한 SCU의 응답 메시지가 어떤 서비스 요청 메시지에 대한 응답인지 식별할 수 없는 문제점을 확인하기 위해 Modbus 프로토콜 및 제안 프로토콜에서 아래와 같은 시나리오를 작성하여 실험을 진행하였다.

Step 1: 제어 온도 설정 값(SV) 50.0도 설정
Step 2: 설정 온도 값(SV) Read 요청
Step 3: 현재 온도 값(PV) Read 요청

Fig. 7은 프리웨어 버전인 COM ANALYZER 프로그램을 이용하여 위 시나리오와 같은 순서로 실험을 수행한 화면이다.

Fig. 7에서 A~C는 Modbus 프로토콜을 이용한 전송 메시지를 나타내고, D~F는 제안한 프로토콜의 전송 메시지를 나타내는 것으로, A와 D는 SV 설정 요청 메시지, B와 E는 SV Read 요청 메시지, C와 F는 PV 값 Read 요청 메시지를 의미한다. 또한, ①~③은 A~C에서 전송한 메시지에 대한 응답 메시지를 나타내며, ④~⑥은 D~F에서 전송한 메시지에 대한 응답 메시지를 나타내는 것으로, 프로토콜 메시지 포맷 형태로 분류하면 Table 5와 같다.

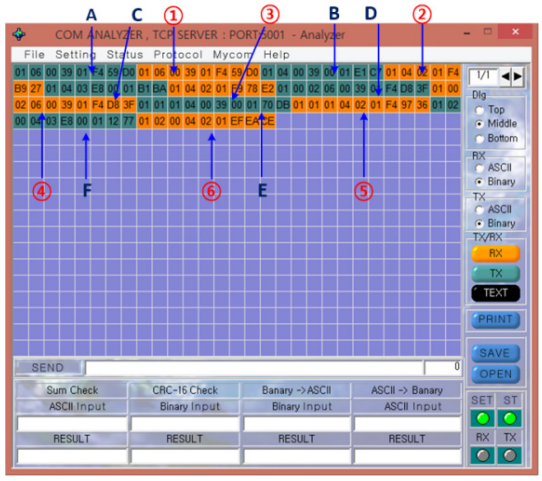


Fig. 7. Experiment Screen by COM ANALYZER

Table 5. Classification of response message

No	Addr	Idx	Svc	Fun	Data Cnt	Data Register	Set/Read Value	Crc Check
①	01			06		0039	01f4	59d0
②	01			04	02		01f4	b927
③	01			04	02		01f9	78e2
④	01	00	02	06		0039	01f4	d83f
⑤	01	01	01	04	02		01f4	9736
⑥	01	02	00	04	02		01ef	eace

Table 5에서 ②는 온도 설정 값인 SV Read 요청 메시지에 대한 응답 메시지로서, Read Value가 01f4이므로 현재 온도 설정 값이 50.0도를 나타내며, ③은 온도의 현재 값인 PV Read 요청 메시지에 대한 응답 메시지로서 Read Value가 01f9로서 현재 온도 값이 50.5도를 나타낸다. ②와 ③ 메시지를 보면 서비스 요청 메시지에 대한 값만을 나타내고 있어 어떤 서비스 요청 메시지에 대한 값인지 식별할 수 없다.

그러나 제안한 프로토콜에서는 동일 요청에 대한 응답 메시지인 ⑤와 ⑥에서는 Service 코드가 추가됨에 따라, Table 4를 참조하여 해당 메시지가 어떤 서비스 요청에 대한 응답인지 식별할 수 있었다.

4.3 SCU 에러 발생 시 비교 실험

본 실험은 특정 SCU에서 에러가 발생할 경우 처리 과정에 관한 실험으로, Table 6과 같은 순서 및 조건으로 실험을 진행하였다.

Table 6. Experiment sequence and conditions

Contents	Explanation
Sequence of experiment	1) Read of current vacuum value 2) Read of current temp value 3) Read of current vacuum value 4) Read of current temp value
Conditions of experiment	1) Send message interval: 100 ms 2) Time out value: 1000 ms 3) Error SCU address: #2

Fig. 8은 SCU에서 에러가 발생한 경우의 MCU 화면이다.

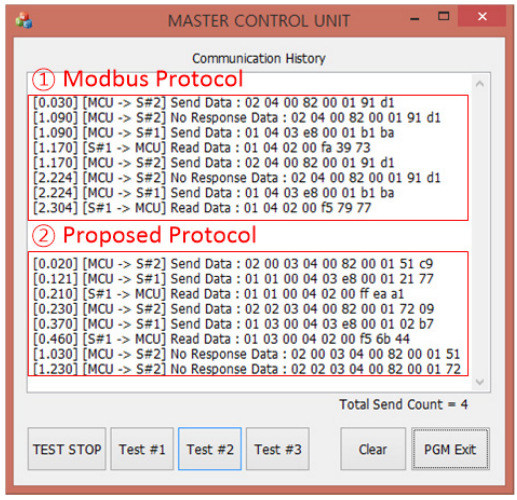


Fig. 8. Experiment Screen in case of SCU error

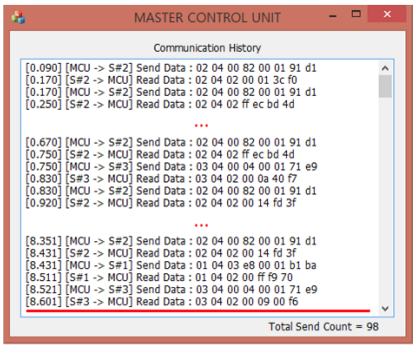
Fig. 8에서 ① 기존 Modbus 프로토콜에서는 Vacuum 읽기 요청 후 미 응답 에러가 약 1초 후에 발생하고 난 이후 다음 메시지가 전송되는 것을 확인할 수 있다. 그러나 ② 제안한 프로토콜에서는 Vacuum 읽기 요청에 대한 응답을 기다리지 않고 다음 서비스인 PV값 읽기 요청 메시지가 전송되고, 이후 약 1초 후에 Vacuum 읽기 요청에 대한 미 응답 에러가 발생하는 것을 확인할 수 있다. 이는 제안한 프로토콜에서는 이미 전송된 여러 서비스 요청 메시지에 대해 Fig. 4의 에러 검색 알고리즘을 이용하여 에러 검색이 가능하므로 먼저 전송한 메시지의 응답이 없는 상태에서도 다음 서비스 요청 메시지를 전송할 수 있었기 때문이다.

4.4 전송 메시지 처리시간 비교 실험

본 실험은 통신 처리시간에 관한 실험으로 Table 7과 같이 현재 온도 값 읽기는 1000 msec 간격, 진공 값 읽기는 60 msec 간격, 인버터의 Air speed 읽기는 500 msec 간격으로 5초 동안 서비스 요청 메시지를 전송하도록 프로그램을 작성하여 실험하였다.

Table 7. Experiment conditions

Service message	Send interval
Read the PV	1000 msec
Read the Vacuum Value	60 msec
Read the Air Speed	500 msec



(A) Test Screen in Modbus Protocol

(B) Test Screen in Proposed Protocol

Unit : sec

Test Count	Modbus Protocol	Proposed Protocol
1	8.601	5.160
2	8.581	5.150
3	8.601	5.160
4	8.600	5.160
5	8.542	5.181
6	8.610	5.211
7	8.534	5.170
8	8.601	5.150
9	8.638	5.171
10	8.583	5.151
AVG	**8.589**	**5.166**
		66.2%

(C) Test Result

Fig. 9. Processing Time Result

Fig. 9는 다중의 동시적인 서비스 요청 메시지 전송 시 처리 시간 결과를 나타내는 것으로, (A)는 Modbus 프로토콜 사용 시의 실험 화면을 나타내고, (B)는 제안한 프로토콜을 실험 화면을 나타내며, 총 98개의 메시지를 전송하였다.

실험은 총 10회에 걸쳐 반복 실험을 하여 평균 처리 시간을 구했다. Fig. 9의 (C)는 실험 결과를 나타내는 것으로, Modbus 프로토콜 사용 시 평균 처리 시간은 8.589 sec가 소요되었으며, 제안한 프로토콜 사용 시 평균 처리 시간은 5.166 sec가 소요되어 제안한 프로토콜 사용 시 약 66.2% 처리속도가 향상되었음을 확인하였다. 이는 Modbus 프로토콜에서는 하나의 메시지 완료 후 다음 메시지를 보내는 반면, 제안한 프로토콜 메시지는 이전에 전송한 메시지에 대한 응답이 없는 경우에도 다음 메시지를 전송할 수 있기 때문이다.

5. 결론

기존 Modbus 프로토콜은 응답한 메시지가 어떤 서비스 요청에 대한 응답인지를 식별할 수 없어 하나의 서비스 요청이 처리된 후 다음 서비스 요청 메시지를 전송할 수 있었다.

본 논문에서는 이런 문제를 해결하기 위해 Index 코드 및 Service 코드를 추가한 Modbus 응용 프로토콜을 제안하였다. 제안한 프로토콜에서 Index 코드는 메시지의 전송 순서를 파악하기 위한 코드로 해당 코드를 추가함으로써 이전 전송한 서비스 요청 메시지가 처리되지 않은 상태에서도 또 다른 서비스 요청 메시지를 전송할 수 있도록 하였다. 또한, Service 코드를 추가함으로써 응답한 메시지가 어떤 서비스에 대한 응답 메시지인지 식별할 수 있도록 하였다.

실험 결과, 제안한 프로토콜에서는 하나의 서비스 요청 메시지 전송 후 응답 메시지 처리가 완료되지 않은 상태에서 다음 서비스 요청 메시지의 전송이 가능하였으며, 에러 발생 시 Index 코드 및 Service 코드를 이용하여 에러가 발생한 SCU 위치 및 어떤 서비스 요청 메시지인지를 확인할 수 있었다. 또한, 다중의 동시적인 서비스 요청 메시지 전송 처리시간에 관한 실험에서는 기존 Modbus 프로토콜보다 약 66.2%의 처리 시간이 향상된 것을 확인하였다.

향후 연구에서는 스마트 제어를 위한 다중 IoT 장치 연결 시 취약한 보안 문제를 해결하기 위한 경량화 인증 방식 및 메시지 암호화 방법에 관한 연구를 수행하고자 한다.

References

[1] Soo Jeong, Jong Jin Lee, Won Ki Jung, "A Indoor Management System using Raspberry Pi", *Journal of the KAIS*, vol. 17, no. 9, pp. 745-752, 2016.
DOI: https://doi.org/10.5762/KAIS.2016.17.9.745

[2] Martin Wollschlaeger, Thilo Sauter, Juergen Jasperneite, "The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0", *IEEE Industrial Electronics Magazine,* vol. 11, no. 1, pp. 17-27, 2017.
DOI: https://doi.org/10.1109/MIE.2017.2649104

[3] DaeHeon Park, "A Study on Integrated Operation Systems of Plant Factory based on Internet of Things", Ph.D Thesis, SunChon University, 2015.

[4] Bo-Hun Kim, "A Design and Implementation of PC-based Control System for Efficiency Improvement of Industrial Equipments", Ph.D Thesis, KongJu University, 2017.

[5] Jeyasingam Nivethan, Mauricio Papa, "A SCADA Intrusion Detection Framework that Incorporates Process Semantics", *ACM CISRC '16 Proceedings of the 11th Annual Cyber and Information Security Research Conference*, 2015.

[6] Sungmo Jung, Jae-gu Song, Taihoon Kim, Yohwan So, Seoksoo Kim, "Design of Idle-time Measurement System for Data Spoofing Detection", *Journal of the Korea Academia-Industrial cooperation Society*, vol. 11, no. 1, pp. 151-158, 2010.
DOI: http://doi.org/10.5762/KAIS.2010.11.1.151

[7] Saptarshi Naskar, Krishnendu Basuli, Samar Sen Sarma, "ASerial port data communication using Modbus protocol", *Journal of ACM Ubiquity*, vol. 2008, no. 1, 2008.

[8] Bo-Heon Kim, Jeong-Ho Song, Hwang-Rae Kim, "A Study on Enhancement of the MOD-BUS RTU Protocol for Multi-Device Connection", *Proceedings of the KAIS Fall Conference*, 2016.

[9] Modbus Application Protocol Manual V1_1b3: Available From: http://www.Modbus.org (accessed Aug, 15, 2017)

[10] SamWon Tech Nova Communication Manual: Available From: http://www.samwontech.com/bbs/bbs/ board.php?bo_table=data_04&wr_id=10 (accessed July, 21, 2017)

[11] Kisan System KM60xx Modbus Manual: Available From:http://www.kisansystem.kr/datasheet/KM60xx_Modbus_Manual.pdf (accessed May, 10, 2017)

송 정 호(Jeong-Ho Song) **[정회원]**

- 1999년 2월 : 호서대학교 컴퓨터공학과 공학사
- 2001년 8월 : 아주대학교 공학대학원 컴퓨터공학과 공학석사
- 2016년 8월 : 공주대학교 대학원 컴퓨터공학과 박사수료
- 1999년 3월 ~ 현재 : 동일공업고등학교 컴퓨터미디어보안과 교사

<관심분야>
IoT, 스마트 제어, 네트워크 보안, NCS, KQF, 정보보호, 공업 교육

김 보 헌(Bo-Hun Kim) **[정회원]**

- 2004년 2월 : 호서대학교 컴퓨터공학과 공학사
- 2013년 8월 : 공주대학교 대학원 IT공학과 공학석사
- 2017년 8월 : 공주대학교 대학원 컴퓨터공학과 공학박사
- 2015년 3월 ~ 현재 : ㈜이지에스 기술이사

<관심분야>
IoT, 스마트 제어, 스마트 팩토리, 장비 제어 프로그램, 컴퓨터 네트워크, 네트워크 보안, 개발 표준화

김 황 래(Hwang-Rae Kim) **[정회원]**

- 1982년 8월 : 중앙대학교 전자계산학과 이학사
- 1991년 2월 : 중앙대학교 대학원 컴퓨터공학과 공학석사
- 2007년 8월 : 대전대학교 대학원 컴퓨터공학과 공학박사
- 1983년 3월 ~ 1994년 2월 : 한국전자통신연구원 책임연구원
- 1994년 3월 ~ 현재 : 공주대학교 컴퓨터공학부 교수

<관심분야>
컴퓨터 네트워크, 네트워크 보안, 네트워크 생존성관리, 스마트 제어