Compact Modbus TCP/IP protocol for data acquisition systems based on limited hardware resources

Article *in* Journal of Instrumentation · April 2018

DOI: 10.1088/1748-0221/13/04/T04004

5 authors, including:

Qing Bai
Taiyuan University of Technology
55 PUBLICATIONS **700** CITATIONS
SEE PROFILE

Dong Wang
Taiyuan University of Technology
74 PUBLICATIONS **868** CITATIONS
SEE PROFILE

Compact Modbus TCP/IP protocol for data acquisition systems based on limited hardware resources

Qing Bai, Baoquan Jin,* Dong Wang, Yu Wang and Xin Liu

Key Laboratory of Advanced Transducers and Intelligent Control Systems, Ministry of Education, Taiyuan University of Technology,
No.79 Yingzexi Street, Taiyuan, Shanxi, 030024, China
E-mail: jinbaoquan@tyut.edu.cn

ABSTRACT: The Modbus TCP/IP has been a standard industry communication protocol and widely utilized for establishing sensor-cloud platforms on the Internet. However, numerous existing data acquisition systems built on traditional single-chip microcontrollers without sufficient resources cannot support it, because the complete Modbus TCP/IP protocol always works dependent on a full operating system which occupies abundant hardware resources. Hence, a compact Modbus TCP/IP protocol is proposed in this work to make it run efficiently and stably even on a resource-limited hardware platform. Firstly, the Modbus TCP/IP protocol stack is analyzed and the refined protocol suite is rebuilt by streamlining the typical TCP/IP suite. Then, specific implementation of every hierarchical layer is respectively presented in detail according to the protocol structure. Besides, the compact protocol is implemented in a traditional microprocessor to validate the feasibility of the scheme. Finally, the performance of the proposed scenario is assessed. The experimental results demonstrate that message packets match the frame format of Modbus TCP/IP protocol and the average bandwidth reaches to 1.15Mbps. The compact protocol operates stably even based on a traditional microcontroller with only 4-kB RAM and 12-MHz system clock, and no communication congestion or frequent packet loss occurs.

KEYWORDS: Software architectures (event data models, frameworks and databases); Data acquisitionconcepts; Digital electronic circuits.

* Corresponding author.

Contents

1. Introduction

With the rapid development of the distributed sensing network, intelligent communication is increasingly essential for data acquisition systems. Recently, the sensor-cloud platform and the cyber-physical system (CPS) have been also conceptualized and developed rapidly with the extensive use of the Internet during the implementation of Industry 4.0 [1, 2]. This further requires data acquisition systems featuring network expansion and sufficient compatibility with the Internet. Therefore, Ethernet interfaces are progressively equipped in the monitoring systems for easy Internet connectivity [3-6]. Meanwhile, many Ethernet protocols have being inevitably applied to sensors or intelligent instruments for online data acquisition, such as Modbus TCP/IP [7-10]., EtherCAT [11-13], Powerlink [14-16], Profinet [17-20], and so on. Among these protocols, the Modbus TCP/IP has attracted much attention due to its openness, transparency, flexibility, convenience and high efficiency [21].

Originally, the Modbus TCP/IP protocol typically runs on a PC-based human–machine interface to communicate with programmable logic controllers (PLCs) especially in supervisory control and data acquisition (SCADA) systems [22]. For easy handling and low power consumption, the Modbus TCP/IP protocol is subsequently migrated on the ARM hardware platform[23]. Still, it needs to be supported by a full operating system such as WinCE, Linux and μC/OS-II [21, 24-26], which always consume many hardware resources and complex process schedulings. However, numerous existing data acquisition systems based on traditional a single-chip microcontroller possess no enough hardware resources to sustain a full operating system.

Hence, it is of profound significance to deeply simplify the classical Modbus TCP/IP protocol and make it run efficiently and stably even though on the resource-limited hardware platform. The compact Modbus TCP/IP can provide a high-efficiency, low-cost and easy-handling scenario to make the existing traditional data acquisition systems compatible with the Internet directly.

Based on above considerations, a compact Modbus TCP/IP protocol is designed and implemented in this work. Firstly, the Modbus TCP/IP Protocol is simply analyzed and the refined hierarchical structure of the protocol stack is built through deeply simplifying the existing computer-based TCP/IP protocol stack and embedding the Modbus protocol into the application layer. Then, implementation details of every hierarchical layer are respectively presented. Additionally, the refined protocol is verified on a traditional single-chip microcontroller and a three-node monitoring platform is assembled for testing the feasibility of the scenario. Finally, the packet format and communication performance of the refined Modbus TCP/IP protocol are evaluated. More details of the industrial application are also presented.

2. Architecture of the Compact Protocol Stack

2.1 Analysis of the Modbus TCP/IP Protocol

The Modbus TCP/IP protocol provides client/service (C/S) communication rules for devices connected to the Ethernet TCP/IP network or other buses [10]. To be compatible with different types of devices over the communication system, the Modbus TCP/IP protocol defines a simple application data unit (ADU) regardless of the underlying communication devices. The ADUs applied to Modbus TCP/IP protocol are of two types: request frames and response frames. The two types of frames are both composed of the Modbus application protocol (MBAP) header, the function code and the data segment. The specific ADU format and common function codes defined in the Modbus TCP/IP protocol are illustrated in Figure 1. The 7-byte MBAP header comprises four fields: 2-byte transaction identifier, 2-byte protocol identifier, 2-byte length field and 1-byte unit identifier. The function code and the data segment are used to indicate the server which kind of action to perform and the data needed to fulfill the action. Here, detailed descriptions of ADU fields are listed in Table 1 [10].

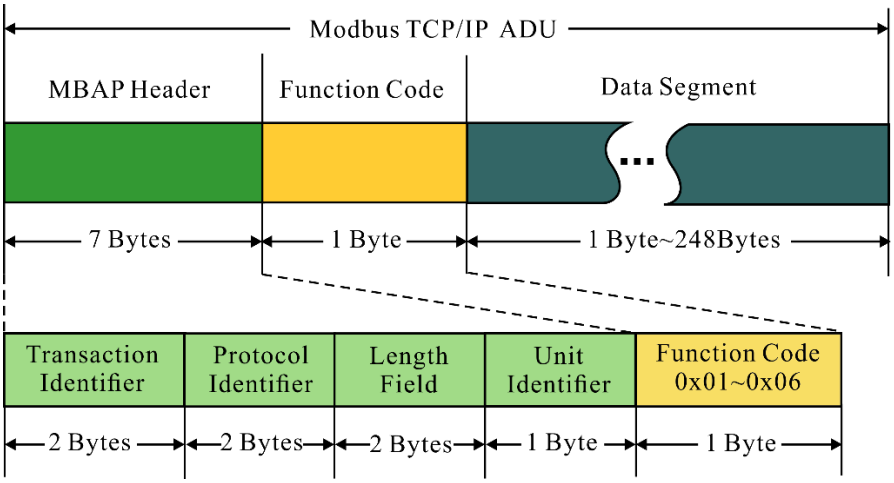

Figure 1. ADU format and common function codes defined in Modbus TCPIP protocol

According to the Modbus TCP/IP, the communication procedure between a client and a server utilizing the Modbus TCP/IP protocol is briefly introduced as below. When starting a

communication, the client generates a request frame, fills every field of the data frame and sends it to a server. Once the server receives the request frame from the client, it verifies the ADU according to the frame format. Here, if the format and contents of the frame are correct, the server will reply a normal response frame, otherwise the server will reply an exception response frame. In a normal response frame, the function code is copied from that of the received request frame, and the data segment carries information that the client requests. On the contrast, in an exceptional response frame, the function code is generated by a bitwise-or operation of the request frame's function code with the value of 0x80, and the data segment is filled with the exception code that indicates which error occurs.

Table 1. Detailed descriptions of Modbus TCP/IP ADU fields

	Fields	Description
MBAP Header	Transaction Identifier	Identification of a MODBUS Request / Response transaction, used for transaction pairing and initialized by the client and copied by the server.
	Protocol Identifier	Used for intra-system multiplexing and set to 0 constantly
	Length Field	A byte count of the following fields including the Function Code and Data segment, initialized by the server and the client respectively.
	Unit Identifier	Identification of a remote slave connected on a serial line or on other buses, initialized by the client and copied by the server
Function Code	0x01	Read Coils
	0x02	Read Discrete Inputs
	0x03	Read Holding Registers
	0x04	Read Input Register
	0x05	Write Single Coil
	0x06	Write Single Register

2.2 Hierarchical Structure of the Compact Protocol Stack

The essence of Modbus TCP/IP protocol is to apply Modbus protocol to the application layer of the classical TCP/IP protocol stack. And the Modbus TCP runs theoretically by embedding the ADU of a Modbus frame into a TCP frame. Hence, a compact protocol stack was designed by simplifying the classical computer-based TCP/IP protocol stack to make it suitable for limited hardware resources. As shown in Figure 2, the refined protocol stack was composed of four layers, which were correspondingly (from top to bottom) the application layer, the transport layer, the network layer and the network interface layer. Only some essential protocols were reserved in every layer to perform a quick and efficient TCP/IP communication. The refined protocol suite and frame processing procedure are shown in Figure 2.

The task of the network interface layer was mainly to provide a physical medium and a reliable data link to complete data transmission. The network interface layer encompassed four program modules: MAC layer initialization, PHY layer initialization, frame encapsulation and decapsulation, access program of CP2200 (the Ethernet controller adopted in our hardware scheme). The initialization of the PHY layer and the MAC layer was aimed to initialize relevant registers inside the Ethernet controller. Once completed, the Ethernet controller automatically implemented functions of the MAC layer and PHY layer defined by IEEE 802.3. The access program of CP2200 was utilized to drive the Ethernet controller to receive and transmit Ethernet

frames. The module of frame encapsulation and decapsulation processed received data based on the procedures specified in Ethernet DIX V2 standard.

In the network layer, to simplify protocols, we merely remained the address resolution protocol (ARP), the internet protocol (IP), and the internet control message protocol (ICMP), removing the reverse address resolution protocol (RARP) and unicast routing protocols such as the open shortest path first (OSPF) protocol and the Internet group management protocol (IGMP). Besides, only ping command and port-unreachable messages were answered in the ICMP protocol, regardless of other datagrams.

In the transport layer, the TCP and the UDP were remained. The TCP port of 502 was reserved for Modbus TCP/IP transaction and the port of 7 was reserved for echo server to debug and test network connectivity. Ports for other services were closed and shielded to streamline the processing procedure of data frames. Modbus protocol was utilized in the application layer. We selected the most common function codes from 0x01 to 0x06 to verify the feasibility of the proposed protocol suite.

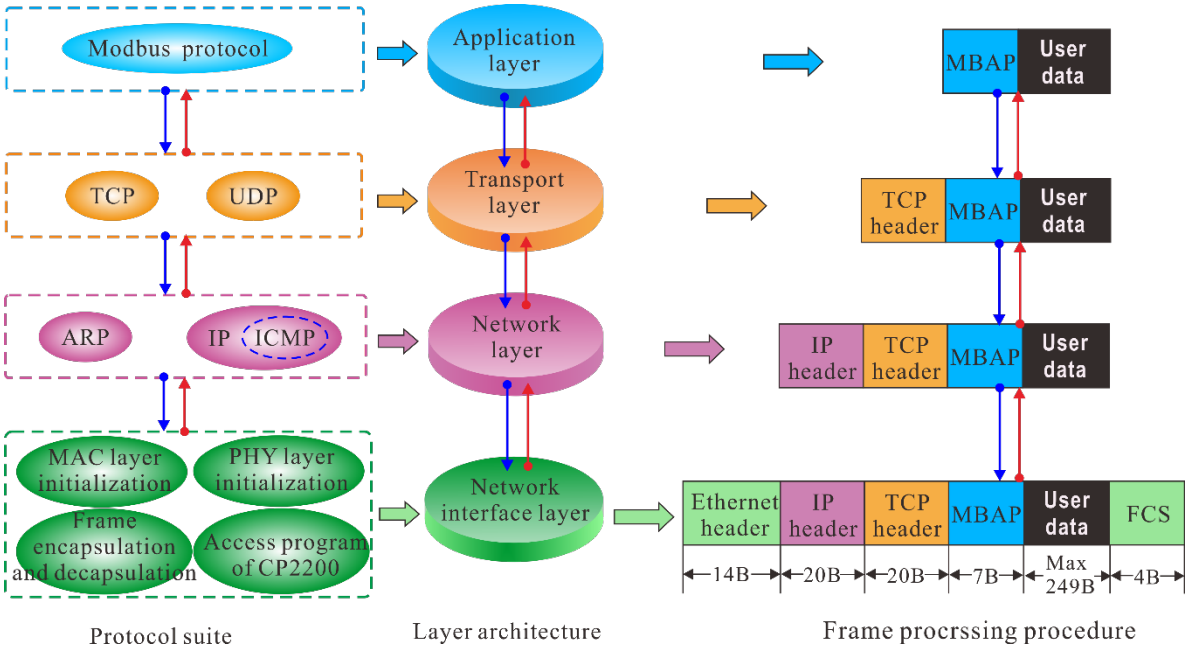

Figure 2. Compact protocol suite and frame processing procedure

Accordingly, the frame processing procedure involved the frame encapsulation and decapsulation, which were both corresponding to the hierarchical structure of our protocol suite. For frame encapsulation, every layer regarded the data from the upper layer as its payload and appended its exclusive protocol header or trailer to the payload. As shown in Figure 2, a MBAP header, a TCP header, an IP header, and an Ethernet header were successively added to user data from the application layer to the network interface layer. Noteworthily, the preamble, start frame delimiter, CRC of an Ethernet frame, were automatically generated by the Ethernet controller (not shown in Figure 2). Until the network interface layer, the user data was eventually encapsulated into an Ethernet frame and was transmitted in the form of bit stream by the Ethernet controller. Frame decapsulation was nearly equivalent to an inverse process of frame encapsulation. The

frames received by the Ethernet controller were strip of the correlative header of every layer from bottom to top. Ultimately the user data was extracted and submitted to the microprocessor.

3. Specific Implementation of the Compact Protocol

3.1 Implementation of the Network Interface Layer

As presented above in Figure 2, four program modules are included in the network interface layer, in which the access program of CP2200 is elementary and defines how to operate registers of the Ethernet controller. For comfortable handling, the Ethernet controller was accessed by typical non-multiplexed connection and the 16-bit base address of Ethernet controller's registers was fixed to 0x9700[27]. Accordingly, the practical address used by the microprocessor to access the Ethernet controller's register was the sum of the base address (0x9700) and an 8-bit on-chip address of the register. We defined global variables for convenient access by utilizing two C-language keywords of "volatile" and "_at_" to specify the access address of the Ethernet controller's register. For instance, if reading or writing a register named MACADDR with an on-chip address of 0x0A, we defined a global variable through a C-language statement (volatile unsigned char xdata MACADDR _at_ 0x970A) and quoted the variable (MACADDR) directly in the program. Modules of MAC layer initialization and PHY layer initialization took charge of loading the 48-bit preprogrammed MAC address, configuring the Ethernet into a full duplex mode and completing the auto-negotiation with the connected Ethernet network. The detailed program code of three modules described above was organized according to the guides specified in CP2200 datasheet[27].

The module of frame encapsulation and decapsulation was designed to add an Ethernet header to the data from the network layer when we were to send a frame, and to extract the payload data and submit it to the network layer when we received a frame. Based on the Ethernet frame format specified in Figure 2, the maximum length of the frame is 314 bytes. Hence, 314-byte RAM space was allocated for one TCP connection used to send and receive the frame. Additionally, the struct named Eth_Header was defined in the module for convenient frame encapsulation. Algorithm 1 describes the main processes in the network interface layer.

Algorithm 1. Pseudo-code for the main processes in the network interface layer

```
Allocate 314-byte RAM space for an array DadaBufer [314]
type Eth_Header
    dim DestHwaddr[6] as unsigned char ARRAY
    dim SourceHwaddr [6] as unsigned char ARRAY
    dim FrameType as unsigned integer
end Eth_Header
Initialize the configuration of the microprocessor
Initialize the MAC layer and the PHY layer
if new data arrives then
    if the new data is a datagram from the network layer then
        Encapsulate an Ethernet frame by adding an Eth_Header to the datagram
        Send the frame through the Ethernet controller
    end if
    if the new data is the Ethernet frame received from the Ethernet controller
        Extract the payload from the frame according to the Eth_Header
        case Eth_Header. FrameType of
        case 1: the Eth_Header. FrameType is "ARP datagram"
```

```
18:              Submit the datagram to the ARP module in the network layer
19:          case 2: the Eth_Header. FrameType is "IP datagram"
20:              Submit the datagram to the IP module in the network layer
21:          end case
22:      end if
23:  end if
```

3.2 Implementation of the Network Layer

The software in the network layer was mainly designed for the ARP and the IP, whose functionalities include updating an ARP cache table, sending and receiving IP datagrams. The flowchart of specific implementation in the network layer is shown in Figure 3.

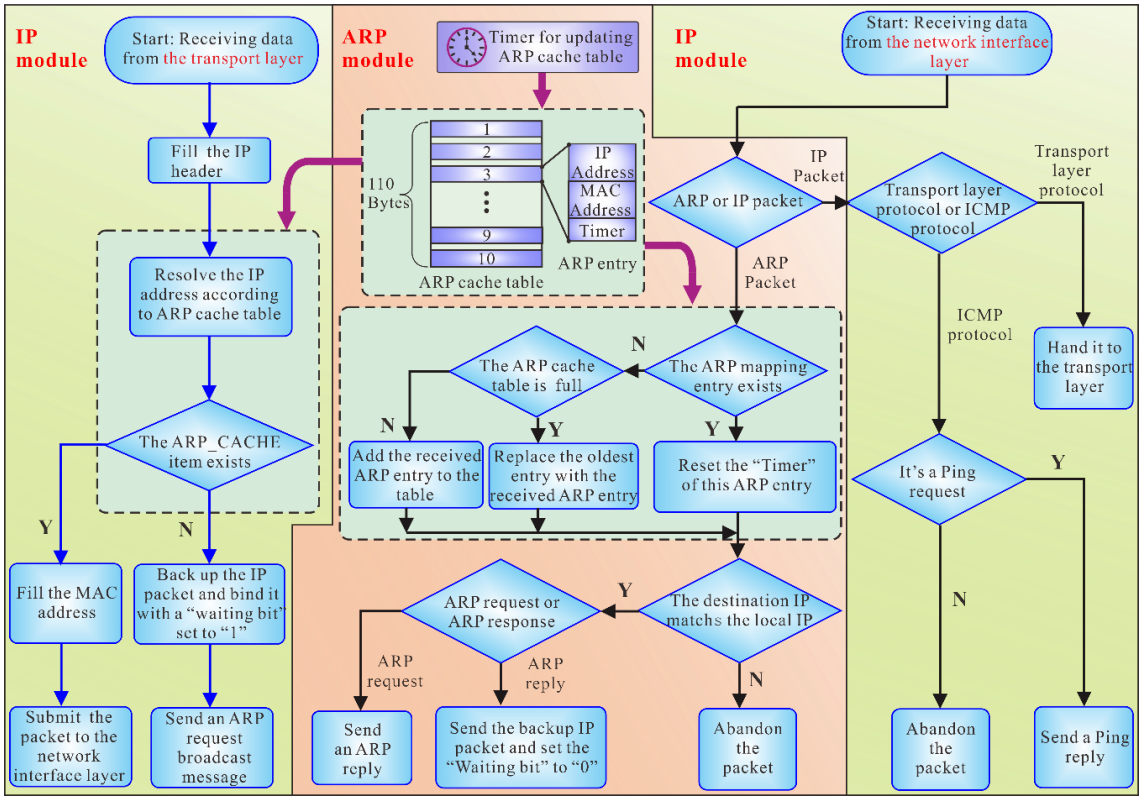

Figure 3. Flowchart of specific implementation in the network layer

The module of IP was responsible for encapsulating or decapsulating an IP datagram. The IP datagram was not to be fragmented, because the length of the Modbus TCP/IP data frame was not beyond that of a maximum transfer unit specified by the Ethernet. As presented in Figure 3, when receiving data from the transport layer, we firstly added the IP header to the data and constructed a to-be-sent IP datagram, then resolved the destination IP address according to the ARP cache table. If the ARP entry existed, the IP datagram obtained an associated MAC address and was submitted to the network interface layer. If the ARP entry did not exist, the IP datagram was stored as a backup and an ARP request broadcast was sent to obtain the MAC address that the destination IP address maps to. When receiving data from the network interface layer, we firstly determined the data was an IP datagram or an ARP packet. If the data was an IP packet, the IP header was derived from the IP packet. Then the remained payload data were submitted to

the transport layer and were processed based on the TCP or ICMP. If the data was an ARP packet, it was handled according to the ARP.

The module of ARP was primarily designed to dynamically update an ARP cache table composed of ten pieces of ARP entry. Each ARP entry consisted of three data fields: a 4-byte IP address, a 6-byte MAC address and a 1-byte ARP age. Thus, only 110-byte RAM space was needed to save the ARP cache table. The ARP age field with an initial value of 40 determined the lifetime of an ARP entry in the table. On one hand, the ARP age was checked by a pre-configured ARP timer. Every a half second (counting by the ARP timer), the ARP age of each entry decreased by one. On the other hand, the ARP age of an entry was also updated when an APR packet was received. As shown in Figure 3, when we received an ARP packet, we firstly checked whether the ARP mapping entry existed or not. If the IP address field of an existing ARP entry accorded with the source IP address carried by the received ARP packet, we refreshed the MAC address of the existing ARP entry based on the source MAC address of the received ARP packet, and simultaneously set the ARP age of the existing entry to 40 afresh; if no such an existing ARP entry was found, we created a new ARP entry by filling an empty entry (when the ARP cache table was not full) or replacing the oldest entry (when the ARP cache table was full), and similarly set the ARP age value of the new entry to 40. Besides, the ARP age of an entry is also set to 40, if the entry was used when we were resolving the destination IP address before sending an IP datagram. Finally, the ARP entry with an ARP age of 0, which means that the entry has not been refreshed or used for 20 minutes, was deleted to free up the table space.

3.3 Implementation of the Transport Layer

For high-efficiency operation based on the limited hardware resource, the simplified TCP and UDP were developed in the transport layer. The transport layer was mainly to ensure reliable transmission of TCP packets and precise state transition of the TCP finite state machine (FSM). Generally, reliable transmission of TCP packets was guaranteed by three simple tools: checksum, acknowledgment, and time-out retransmission. It is effortless for the destination TCP to discard the datagram with an invalid checksum, because a 16-bit checksum field is mandatory in the TCP header of every TCP segment. Hence, acknowledgment mechanism, timeout retransmission and FSM transformation are primarily considered.

To guarantee the acknowledgment mechanism, two variables termed My_Seq and His_Ack were defined in every established TCP connection to respectively indicate the local sequence number and the counterpart's sequence number of a TCP connection. Every time a TCP packet was transmitted, the value of My_Seq increased by the number of bytes that the TCP datagram's payload consumed. When a TCP packet was received from the counterpart, the ACK number of the TCP connection was extracted from the TCP header and assigned to the variable of His_Ack. According to the TCP, if data transmission of an established TCP connection was normal, the two variables of the TCP connection satisfied the relationship of My_Seq = His_Ack. Each established connection was scanned every three seconds. If the two variables of a TCP connection were not in accordance with the relationship of My_Seq = His_Ack, a RST command was sent to reset the TCP connection.

For the timeout retransmission, a timer of the microprocessor was particularly set to record the round-trip time (*RTT*). The round-trip time sequence of RTT_i was stored for calculating retransmission time-out (*RTO*). The retransmission time-out sequence of RTO_i was calculated by the Equation (1) to Equation (6), which are expressed as follows: [28]

$$SRTT_1 = RTT_1 \tag{1}$$

$$RTTD_1 = RTT_1 / 2 \tag{2}$$

$$RTO_1 = \text{Initial value} \tag{3}$$

$$SRTT_i = (1-\alpha)SRTT_{i-1} + \alpha RTT_i, i \geq 2 \tag{4}$$

$$RTTD_i = (1-\beta)RTTD_{i-1} + \beta |SRTT_i - RTT_i|, i \geq 2 \tag{5}$$

$$RTO_i = SRTT_i + 4 \times RTTD_i, i \geq 2 \tag{6}$$

where $SRTT_i$ is a sequence of the smoothed *RTT*, which is a weighted average of RTT_i and $SRTT_{i-1}$; $RTTD_{i-1}$ is a sequence of *RTT* Deviation, which is calculated based on the $RTTD_{i-1}$, $SRTTi_{-1}$ and *RTTi*. The value of α and β was respectively set to 0.125 and 0.25 normally. RTO_1 was set to an initial value according to the site network conditions, which was set to 0.3 (in seconds) in this communication scheme.

To ensure the simple and efficient state transition of FSM, we only retained seven essential states of the classical TCP protocol as a server. They were respectively CLOSED, LISTEN, SYN_RCVD, ESTABLISHED, LAST_ACK, FIN_WAIT1 and FIN_WAIT2. The FSM of a simplified TCP used in our work is shown in Figure 4.

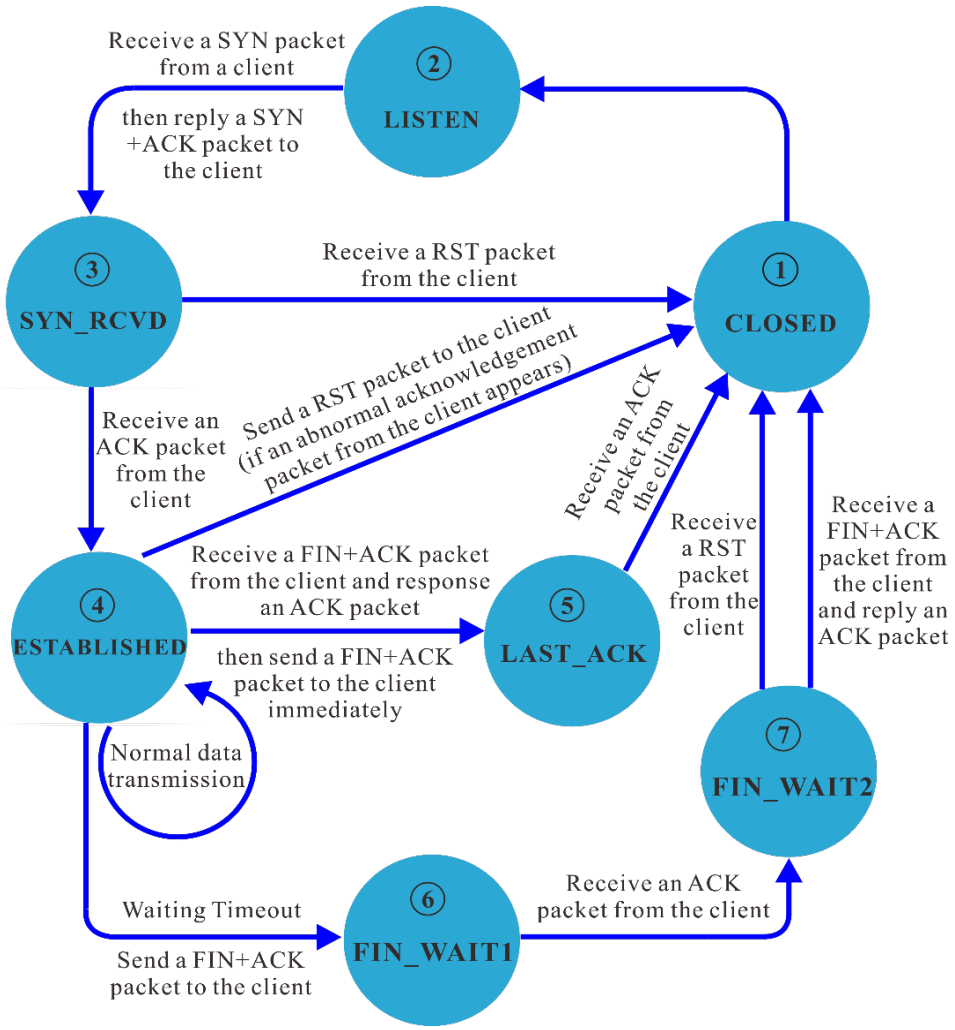

Figure 4. Finite-state machine of a simplified TCP used in our work

When a normal TCP data transmission was in progress, the state of a TCP connection transformed as the path describes: ①→②→③→④→⑤→①. If the server received a RST packet from the client before the TCP connection is established, the state was directly changed to CLOSED shown by ③→①. In the ESTABLISHED state, when an abnormal acknowledgement packet from the client appeared, the server closed the TCP connection by sending a RST packet shown by ④→①. If a waiting-timeout fault occurred, which means no response comes from the client in a certain period, the server actively sent a FIN+ACK packet and eventually transformed to the CLOSE state shown by ④→⑥→⑦→①. Algorithm 2 describes the main processes in the transport layer.

Algorithm 2. Pseudo-code for the main processes in the transport layer.

```
1:  Define some variables (My_Seq, His_Ack,…) and two struct types (TCP_Header,UDP_Header)
2:      Initialize a timer of the microcontroller for recording RTT_i
3:      if new data arrives then
4:        if the new data is a message from the application layer  then
5:          Generate an 16-bit checksum, adds an TCP_Header and encapsulates an TCP packet
6:          Hand the TCP packet to the network layer
7:        end if
8:        if the new data is the datagram received from the network layer then
9:          if the new data is a UDP packet then
10:           Verify the packet UDP according to the checksum and discard the invalid
                    packet
11:           if the packet is for UDP port 7 then
12:             Echo a received datagram back to the sender
13:           else reply a destination-unreachable message
14:           end if
15:         end if
16:         if the new data is a TCP packet then
17:           Verify the TCP packet according to the checksum and discard the invalid packet
18:           if the TCP packet is not for port 502 (Modbus service) then
19:             Abandon the TCP packet
20:           end if
21:           Update the state of the TCP connection according to Figure 6
22:           Update and check the variables of My_Seq and His_Ack
23:           if His_Ack > My_Seq then
24:             Close the TCP connection
25:           end if
26:           if His_Ack < My_Seq then
27:             Start a retransmission
```

28: **end if**

29: **end if**

30: **end if**

31: **end if**

3.4 Implementation of the Application Layer

The application layer mainly concentrated on how to allocate the limited storage space and how to build the request/response mechanism in accordance with the Modbus protocol. According to the Modbus data model, the storage space of the Modbus device was mandatorily divided into four areas correspondingly for discrete input, coils, input registers and holding registers [10]. In this paper, a traditional single-chip microcontroller only with the 4-kbyte on-chip RAM was selected to implement the refined protocol. Allowing for the limited RAM, storage space allocation is illustrated in Figure 5. 16 bytes with the address from 0x0000 to 0x000F were allocated to the read-only area for 128 discrete inputs, that is, every byte stored 8 discrete states. Similarly, the read-write area for 128 coils states also obtained 16 bytes with the address from 0x0010 to 0x001F. Because a register specified by Modbus protocol consists of two bytes, we respectively assigned 1024 bytes to the read-only area for 512 input registers and to the read-write area for 512 holding registers. Thus, the input and holding registers totally consumed 2048-byte RAM with the address from 0x0020 to 0x081F. Finally, residual 2016 bytes were reserved for other variables and data buffer used in the program.

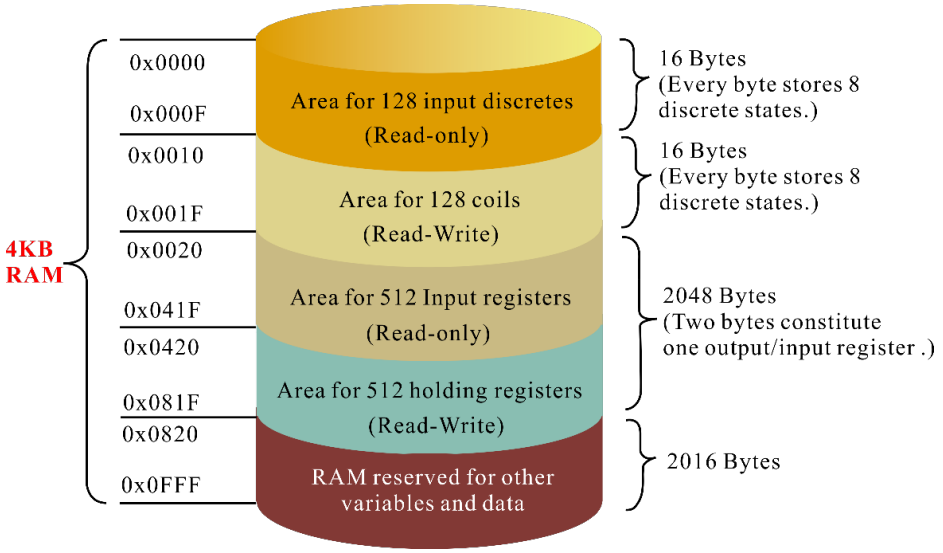

Figure 5. Storage space allocation

Additionally, the Modbus request/response mechanism was built based on the Algorithm 3. When a server received a Modbus TCP/IP request frame from a client, it firstly checked the MBAP header. If the protocol identifier of the request frame was not "0", the frame was discarded and a response frame with an exception code of 0x06 was sent to indicate that the server device was busy. After MBAP check was completed, the server analyzed the function code field and performed the action that the client demanded. Only function codes of 0x01~0x06 were supported

in the scenario, hence the requests with other function codes were ignored and a response with an exception code of 0x01 was sent to suggest the query was not an allowable action for the server.

Algorithm 3. Pseudo-code for the main processes in the application layer.

```
Define a struct type: MBAP_Header
Allocate storage space for every specific area
if a Modbus request arrives then
    if MBAP_Header. Protocol Identifier is not "0" then
        Discard the Modbus request
        Send a Modbus response with an exception code of 0x06
    end if
    case Function Code of
    case 0x01: Verify the rationality of the Modbus request
            Obtain coil status that the client demands and send a response to the client
    case 0x02: Verify the rationality of the Modbus request
            Obtain discrete inputs that the client demands and send a response to the client
    case 0x03: Verify the rationality of the Modbus request
            Read holding registers that the client demands and send a response to the client
    case 0x04: Verify the rationality of the Modbus request
            Read input registers that the client demands and send a response to the client
    case 0x05: Verify the rationality of the Modbus request
            Write the coil that the client demands and send a response to the client
    case 0x06: Verify the rationality of the Modbus request
            Write the holding register that the client demands and send a response to the client
    default: Send a Modbus response with an exception code of 0x01
    end case
end if
```

4. Verification and Application

4.1 Experimental Verification

To verify the feasibility of the refined Modbus TCP/IP protocol, a small test bench was built on the basis of a C/S model, as shown in Figure 6. A software platform in an industry personal computer (IPC) was designed by a configurable software named KingView 6.55 to work as a Modbus TCP/IP client. The IP address of the client was set to 192.168.1.110. Three Ethernet nodes were prepared to work as Modbus TCP/IP servers. The IP addresses of three servers were correspondingly set to 192.168.1.33, 192.168.1.34 and 192.168.1.35. Three Ethernet nodes were connected with the IPC by a router.

To check the physical network connectivity, a ping command was transmitted from the client to the server. Wireshark 1.12 was utilized to evaluate the performance of our communication scheme. Figure 7 shows the captured ARP and Ping packets. When the Ping command was sent

to the Ethernet node 1# (a server) with an IP address of 192.168.1.33 at the first time, the client firstly obtained MAC address of the server by sending an ARP request (Broadcast). Then four Ping requests were transmitted and their replies were received in order. Testing results demonstrate that the network interface layer and the network layer of the scenario work normally. Obviously, the UDP and physical network connectivity were ensured.

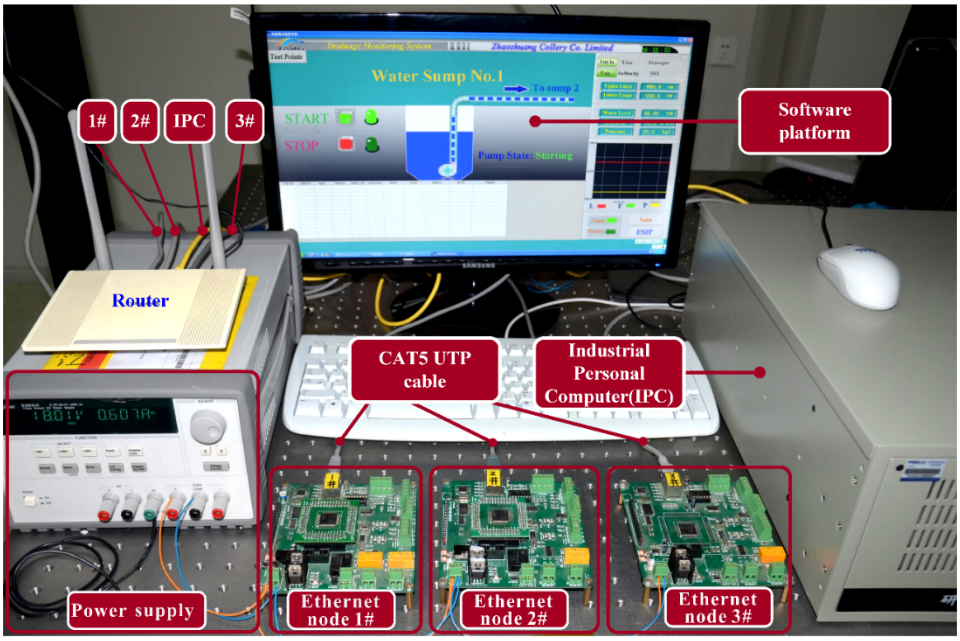

Figure 6. Test bench for the compact Modbus TCP/IP prptocol

No.	Source	Destination	Protocol	Length	Info
1	AioLcdPc_6e:e6:83	Broadcast	ARP	42	Who has 192.168.1.33? Tell 192.168.1.110
2	CygnalIn_5e:f8:7b	AioLcdPc_6e:e6:83	ARP	64	192.168.1.33 is at 00:0b:3c:5e:f8:7b
3	192.168.1.110	192.168.1.33	ICMP	74	Echo (ping) request id=0x0001, seq=53/13568, ttl=64 (reply in 4)
4	192.168.1.33	192.168.1.110	ICMP	74	Echo (ping) reply id=0x0001, seq=53/13568, ttl=32 (request in 3)
5	192.168.1.110	192.168.1.33	ICMP	74	Echo (ping) request id=0x0001, seq=54/13824, ttl=64 (reply in 6)
6	192.168.1.33	192.168.1.110	ICMP	74	Echo (ping) reply id=0x0001, seq=54/13824, ttl=32 (request in 5)
7	192.168.1.110	192.168.1.33	ICMP	74	Echo (ping) request id=0x0001, seq=55/14080, ttl=64 (reply in 8)
8	192.168.1.33	192.168.1.110	ICMP	74	Echo (ping) reply id=0x0001, seq=55/14080, ttl=32 (request in 7)
9	192.168.1.110	192.168.1.33	ICMP	74	Echo (ping) request id=0x0001, seq=56/14336, ttl=64 (reply in 10)
10	192.168.1.33	192.168.1.110	ICMP	74	Echo (ping) reply id=0x0001, seq=56/14336, ttl=32 (request in 9)

Figure 7. Captured ARP and Ping packets

To analyze stability and reliability of the TCP connection in the transport layer, we traced a TCP connection, and the TCP packets during about one hour were captured. Figure 8 shows three stages of the TCP flow graph generated by Wireshark 1.12 according to the captured TCP packets. It can be seen that the TCP port of all packets were 502, which is an exclusive port for the Modbus transaction service. Three-way handshaking for opening a TCP connection is specified in Figure 8(a). After the TCP connection was established, normal data transmission process performs as the flow presented in Figure 8(b). Eventually, four-way handshaking was necessary when the TCP connection was closed, which is shown in Figure 8(c). According to the comment column of the flow graphs, the acknowledgement mechanism of TCP was verified. It is obvious that the “Ack” value sent by a server at one time point is the sum of the “Seq” value sent by a client and the sequence number that the client’s payload data consumed at the last time point. For instance, as

shown in Figure 8 (a), the server sent a SYN+ACK packet to the client with the "Ack" value of 1 at the time point of 0.001424 s, because the client sent a SYN packet to the server with the "Seq" value of 0 at last time point of 0.000000 s, and the sequence number consumed by the payload data of the SYN packet was 1.

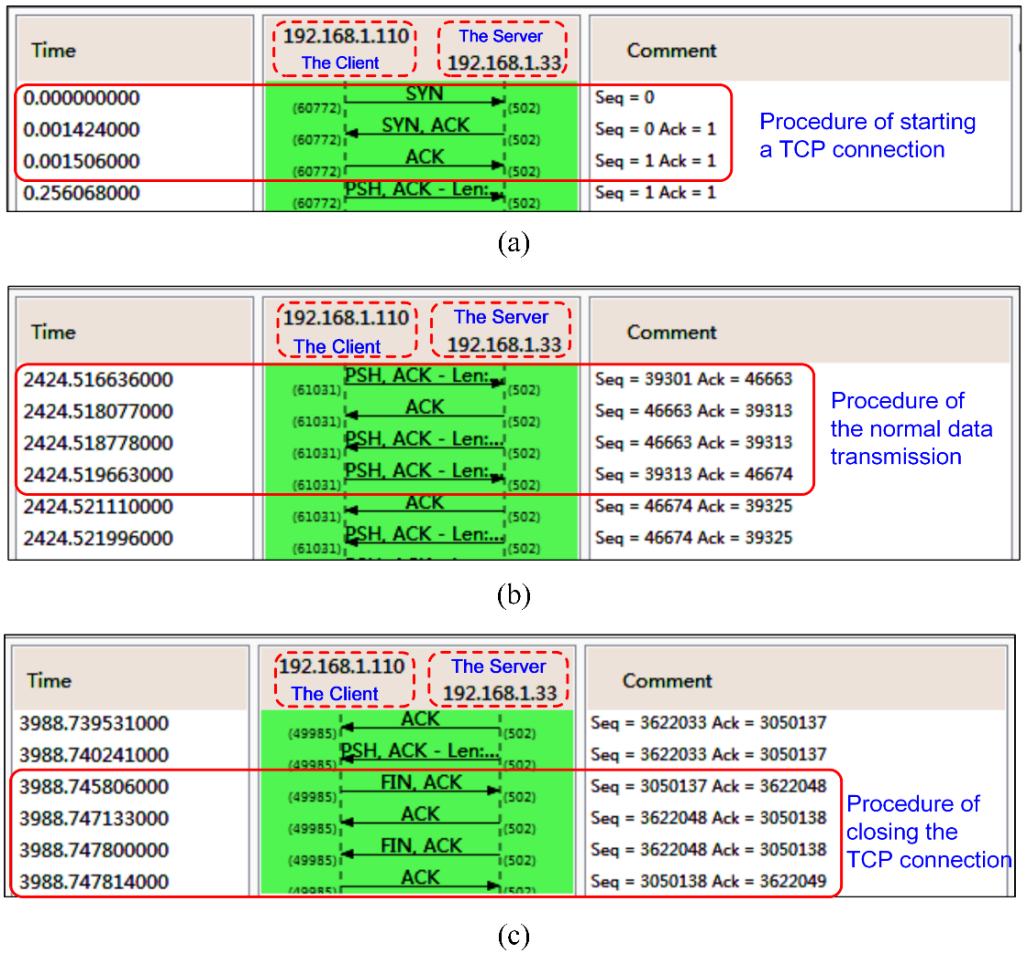

Figure 8. Three stages of the TCP flow graph: (a) Three-way handshaking for opening a TCP connection; (b) Normal data transmission; (c)Four-way handshaking for closing a TCP connection

Table 2 shows some typical Modbus TCP/IP messages, which were selected from the packets captured by Wireshark. The table respectively lists request and response packets with 7 different function codes. As outlined in Section 2.1, only 6 function codes were supported in our device; hence the request messages with function codes of 0x01~0x06 obtained normal response, shown in the first five rows of the table. The function code of the normal response frame is the same as that of a request frame. Noticeably, the request message with unsupported function codes (such as 0x0F) induced an abnormal response, shown in the last row of the table. The function code (0x8F) of the exception response differs from that (0x0F) of the request message, and the data segment (0x01) indicates that the function of writing multiple coils is not supported in our protocol. By contrast, the messages presented in Table 2 conform to the message format defined in the Modbus application protocol specification [10]. The results demonstrate that the Modbus protocol in the application layer performed well.

Table 2 Typical Modbus TCP/IP messages captured by Wireshark

Function	Request/ Response	Transaction Identifier	Protocol Identifier	Length Field	Unit Identifier	Function Code	Data Segment
Read Coils	Request	00 00	00 00	00 06	01	01	00 00 00 02
	Response	00 00	00 00	00 04	01	01	01 FF
Read Discrete Inputs	Request	00 00	00 00	00 06	01	02	00 00 00 0A
	Response	00 00	00 00	00 05	01	02	02 FF 9A
Read Holding Registers	Request	00 00	00 00	00 06	01	03	00 00 00 02
	Response	00 00	00 00	00 07	01	03	04 00 50 00 1E
Read Input Registers	Request	00 00	00 00	00 06	01	04	00 20 00 02
	Response	00 00	00 00	00 07	01	04	04 42 84 CC CD
Write Single Coil	Request	00 00	00 00	00 06	01	05	00 00 FF 00
	Response	00 00	00 00	00 06	01	05	00 00 FF 00
Write Single Register	Request	00 00	00 00	00 06	01	06	00 00 00 32
	Response	00 00	00 00	00 06	01	06	00 00 00 32
Write Multiple Coils	Request	00 00	00 00	00 08	01	0F	00 00 00 08 01 AA
	Response	00 00	00 00	00 03	01	8F	01

Furthermore, we plotted the sequence number and data rate versus time in Figure 9. The lines were derived based on the captured packets from the Ethernet node 1#. Through the test results in an hour, it is noticeable that the sequence number increased continuously with time and no breakpoints existed. This demonstrated that no communication interruption occurred. Besides, the maximum, minimum and average data rate were correspondingly 117.38kbps, 97.78kbps and 102.8kbps.

It is worthy to mention that the above tests were performed on the condition that a commercial configuration software named KingView 6.55 was selected as the PC software platform. Inevitably, the test results of data rate were limited by the message-updating mechanism of the configuration software. Therefore, a program specially for evaluating the bandwidth performance, Jperf 2.0, was utilized to develop the test. A total of 9 tests were conducted and the bit rate of every Ethernet node was measured three times. The value of the bit rate was recorded per second. The measured results are presented in Figure 10. The test results of the Ethernet node 1#, Ethernet node 2# and Ethernet node 3# are correspondingly demonstrated in Figure 10.(a), Figure 10.(b) and Figure 10.(c). From Figure 10.(d), it can be seen that the maximum rate and the minimum rate of three Ethernet nodes respectively fluctuate around 1.18Mbps and 1.115Mbps. After all, the average rate was nearly 1.15Mbps.

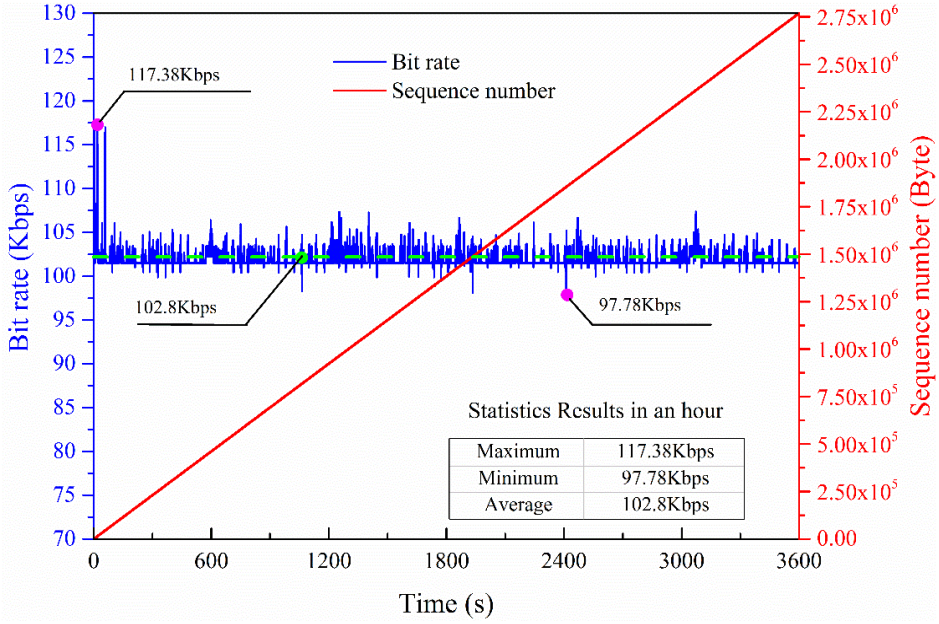

Figure 9. Data rate and sequence number versus time

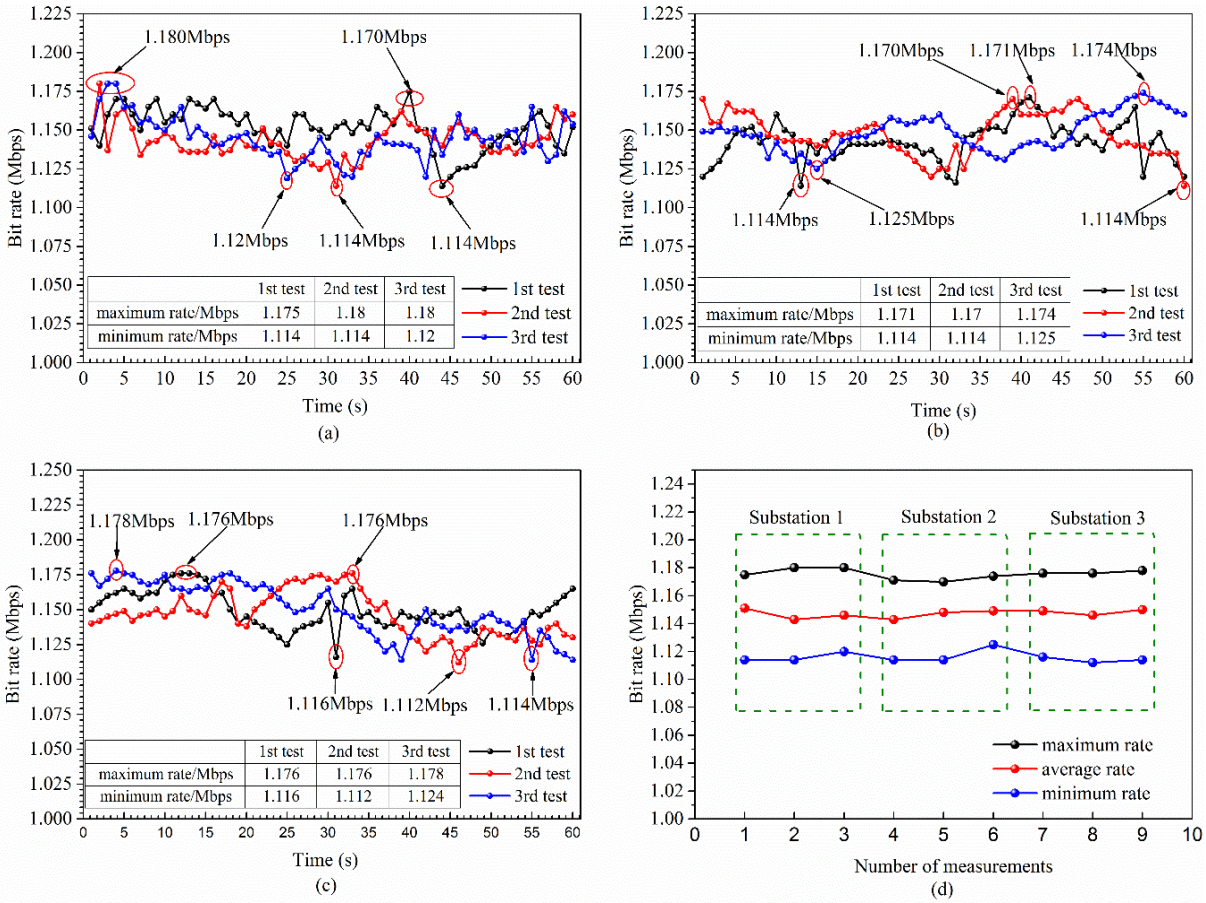

Figure 10. Bit rate test results: (a) Three test results of the Ethernet node 1; (b) Three test results of the Ethernet node 2; (c) Three test results of the Ethernet node 3; (d) Maximum, average and minimum rate of each Ethernet node during every test.

4.2 Industrial Application

The proposed implementation has been applied to a smart supervisory control network of the drainage system in an underground coalmine for 3 years. The system monitored the water level of sumps and controlled the dewatering pump automatically. The supervisory and control signals were both transmitted through the hardware platform shown in Figure 11. The Ethernet interface mainly contained a microprocessor (C8051F040), an Ethernet controller (CP2200) and a RJ45 connector (HR911105A). Moreover, typical measurement interfaces of standard signals including the current (4 to 20 mA), the voltage (0 to 5 V), and the frequency (200 to 1000 Hz) were assembled to be compatible with output signals of most sensors. Also, two RS485-bus interfaces and one CAN-bus interface were appended to support existing widely-used communication protocols. Besides, the power management provided dual levels of over-voltage and over-current protection circuit to match safety requests of underground coalmine devices.

The network with ten Modbus TCP/IP nodes and one monitoring host was utilized in the industrial application, shown in Figure 12.(a) . Every underground node was embedded with a hardware platform shown in Figure 11, and communicated with the monitoring host on the ground utilizing the proposed compact Modbus TCP/IP. The spare machine was used for data backup. Because taking photos is forbidden in the underground coalmine, Figure 12.(b) only gives the photo of applied products when we debugged the system on the ground.

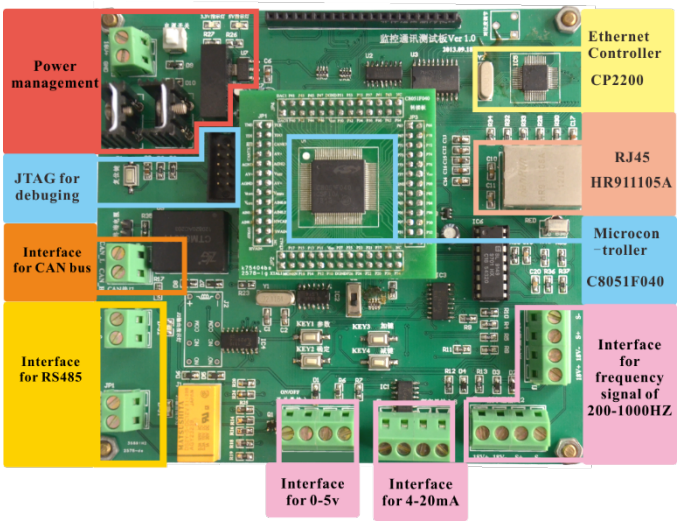

Figure 11. Details of the designed hardware platform

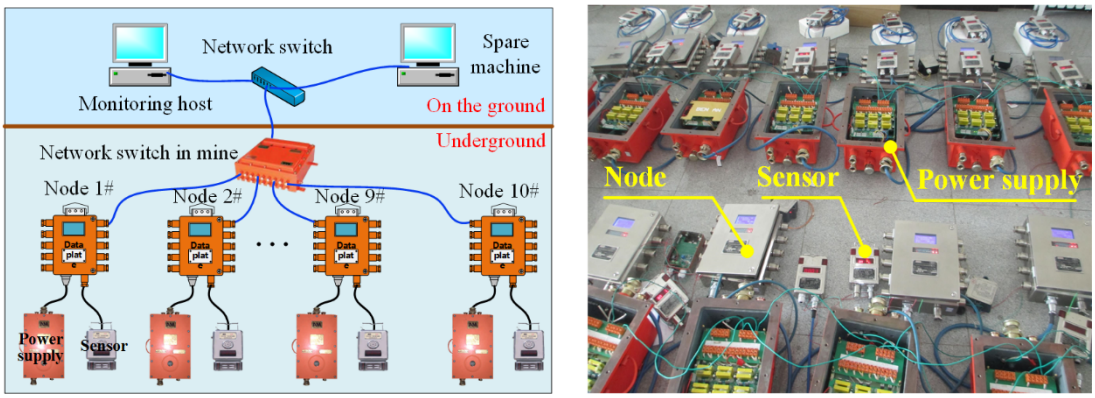

Figure 12. Network structure and applied products: (a) Network structure of monitoring system; (b) Photo of applied products

5. Discussion

As stated previously, the purpose of this study was to build a compact Modbus TCP/IP protocol especially for data acquisition systems based on limited hardware resources. Given the trade-off between resource availability and functionalities of Modbus TCP/IP, the protocol stack of the implementation was simplified. It is clear from the experimental results that the proposed implementation operated stably even based on a traditional microcontroller with only 4-kB RAM and 12-MHz system clock.

However, the simplification of network protocol will arise some application limitations and functional deficiencies inevitably. In the network layer, routing protocols were removed such as OSPF, RIP and BGP, which means routing management was not supported. Besides, IGMP was also forbidden, that is, group management of the network was uninvolved. In the transport layer, the stream control transmission protocol (SCTP) was not in use, resulting in the lack of some modern multistream-service applications such as media gateway control. In TCP, only port 502 was open for Modbus data exchange, and other well-known servives such as FTP, HTTP, SMTP etc. were all rejected. Besides, because no enough buffer cache exists in a traditional microcontroller, advanced TCP functionalities such as SACK were not responsed. Data security scheme of the implementation was not added similarly subject to the limited resources and low computation speed of the traditional microcontroller. Allowing for the fact that the Modbus TCP/IP typically operates in a self-built LAN without frequent congestion or network attacks, the proposed implementation can perform well. For compensation, the database encryption and firewall could be appended to the monitoring host when the data were exposed to the Internet. Actually, we did so in the industrial application.

Compared with other alternative implementations, such as the implementations based on the ARM platform [23], full operating systems [24-26] or advanced microcontrollers [29], the biggest advantage of the proposed compact Modbus TCP/IP was that it consumed less hardware resources. Besides, it integrated commonly-used measurement interfaces of analog signals and communication interfaces including RS485 and CAN. Hence, it can operate not only as an independent sensing system but also as a multi-protocol converter. Additionally, from the perspective of industrial application, it met intrinsically safe requirements and can be applied in the underground coal mine directly.

6. Conclusion

A compact Modbus TCP/IP protocol is designed and evaluated in this work to make it run efficiently and stably even in data acquisition systems built on limited hardware resources. Firstly, the frame format and request/response mechanism of the Modbus TCP/IP protocol were analyzed. Then, the protocol stack was refined and rebuilt by simplifying the traditional TCP/IP protocol suite and merely retaining essential modules for high efficiency and easy handling. Furthermore, according to the protocol stack, we developed programs respectively for different layers allowing for the limited hardware resources. Finally, the compact protocol was implemented in a traditional single-chip microcontroller only with 4-kB RAM and 12-MHz system clock. The performance evaluation was conducted by a network test bench consisting of one IPC as the Modbus client and three Ethernet nodes as Modbus servers.

The experimental results demonstrate that the compact Modbus TCP/IP guaranteed the stable Ethernet data communication on the resource-limited hardware platform. The maximum bit rate reached up to 1.18Mbps and the average bit rate fluctuated around 1.15Mbps. The proposed

scenario was designed based on the limited hardware resources and the Modbus TCP/IP protocol which has been an industry's actual standard. Hence, this configuration easily enables heterogeneous data of existing traditional-microprocessor-based data acquisition systems access the Internet directly. Once the data are transmitted through the Internet, it will be quite convenient to enlarge transmission distance efficiently and facilitate network expansion robustly, by utilizing a router, a hub or a switch. Definitely, more issues such as data security and advanced functionalities should be paid more attention and solved in the further research work. It has the potential to be applied extensively in the intelligent communication layer of a sensor network, a sensor-cloud platform or a cyber-physical system.,

Acknowledgments

This work is supported by Natural Science Foundation of Shanxi Province, China (Grant No. 201601D102026), Coal-Bed Methane Joint Research Fund of Shanxi Province (Grant No. 2015012005), Postgraduate Education Innovation Project of Shanxi Province (Grant No. 2016BY066) and Science and Technology Major Special Research Project of Shanxi Province (Grant No. MJ2016-01). Additionally, the authors sincerely thank the anonymous reviewers for their valuable and professional comments.

References

[1] P. You and Z. Huang, *Towards an extensible and secure cloud architecture model for sensor information system, INT J DISTRIB SENS N*. (2013) .

[2] X. Yue, H. Cai, H. Yan, C. Zou and K. Zhou, *Cloud-assisted industrial cyber-physical systems: An insight, MICROPROCESS MICROSY*. **39** (2015) 1262.

[3] M. Querol et al., *A programmable, multichannel power supply for SIPMS with temperature compensation loop and Ethernet interface,* 2016 *JINST* **11** C12035.

[4] N. J. Gomes, P. Chanclou, P. Turnbull, A. Magee and V. Jungnickel, *Fronthaul evolution: From CPRI to Ethernet, OPT FIBER TECHNOL*. **26** (2015) 50.

[5] A. Flammini, P. Ferrari, E. Sisinni, D. Marioli and A. Taroni, *Sensor interfaces: From field-bus to Ethernet and Internet, Sensors & Actuators A Physical*. **101** (2002) 194.

[6] B. Mindur and L. Jachymczyk, *The Ethernet based protocol interface for compact data acquisition systems,* 2012 *JINST* **7** T10004.

[7] K. Staniec and M. Habrych, *Telecommunication platforms for transmitting sensor data over communication Networks-State of the art and challenges, SENSORS-BASEL*. **16** (2016).

[8] A. Kozyra et al., *Remote controlled water craft for water measurement, MEASUREMENT*. **111** (2017) 105.

[9] M. R. Caldieri, J. A. Bigheti and E. P. Godoy, *Implementation and evaluation of wireless networked control systems using modbus, IEEE LAT AM T*. **15** (2017) 206.

[10] *Modbus messaging on TCP/IP implementation guide v1.0b*. Avaliable online: http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf (accessed on 15 01 2018).

[11] S. Park, H. Kim, H. Kim, C. N. Cho and J. Choi, *Synchronization improvement of distributed clocks in EtherCAT networks, IEEE COMMUN LETT*. **21** (2017) 1277.

[12] L. Lo Bello, E. Bini and G. Patti, *Priority-Driven Swapping-Based scheduling of aperiodic Real-*

Time messages over EtherCAT networks, IEEE T IND INFORM. **11** (2015) 741.

[13] L. Wang, M. Li, J. Qi and Q. Zhang, *Design approach based on EtherCAT protocol for a networked motion control system, INT J DISTRIB SENS N*. (2014).

[14] S. Vitturi, L. Peretti, L. Seno, M. Zigliotto and C. Zunino, *Real-time Ethernet networks for motion control, COMPUT STAND INTER*. **33** (2011) 465.

[15] K. Erwinski, M. Paprocki, L. M. Grzesiak, K. Karwowski and A. Wawrzak, *Application of ethernet powerlink for communication in a linux RTAI open CNC system, IEEE T IND ELECTRON*. **60** (2013).

[16] M. Knezic, B. Dokic and Z. Ivanovic, *Theoretical and experimental evaluation of ethernet powerlink PollResponse chaining mechanism, IEEE T IND INFORM*. **13** (2017) 923.

[17] P. Ferrari, A. Flammini and S. Vitturi, *Performance analysis of PROFINET networks, COMPUT STAND INTER*. **28** (2006) 369.

[18] R. Schlesinger, A. Springer and T. Sauter, *Automatic packing mechanism for simplification of the scheduling in profinet IRT, IEEE T IND INFORM*. **12** (2016) 1822.

[19] Z. Hanzalek, P. Burget and P. Sucha, *Profinet IO IRT message scheduling with temporal constraints, IEEE T IND INFORM*. **6** (2010) 369.

[20] R. Garoby et al., *The european spallation source design, PHYS SCRIPTA*. **93** (2018).

[21] J. Sun, F. Zou and S. Fan, *A token-ring-like real-time response algorithm of Modbus/TCP message based on μC/OS-II, AEU-INT J ELECTRON C*. **70** (2016) 179.

[22] N. Goldenberg and A. Wool, *Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems, INT J CRIT INFR PROT*. **6** (2013) 63.

[23] D. Peng, H. Zhang, K. Zhang, H. Li and F. Xia, *Research and development of the remote I/O data a cquisition system based on embedded ARM platform*, in proceedings of *International Conference on Electronic Computer Technology*, February, 20 - 22, 2009, Macau, China. INSPEC Accession Num ber: 10479680.

[24] S. Chen, C. Li, S. Han and F. Pan, *The design and implementation of Modbus/TCP communication on WinCE platform*, in proceedings of *30th Chinese Control Conference*, July, 22 - 24, 2011, Yantai, China. INSPEC Accession Number: 12256700.

[25] A. Mejías et al., *Easy handling of sensors and actuators over TCP/IP networks by open source Hardware/Software, SENSORS-BASEL*. **17** (2017).

[26] M. Weiss, A. Schiller, P. O'Leary, E. Fauster and P. Schalk, *Development of a distributed vision system for industrial conditions*, in proceedings of *Sixth International Conference on Quality Control by Artificial Vision*, May, 1, 2003, Gatlinburg, United States. Proc. SPIE 5132.

[27] *Single-chip Ethernet controller datasheet*. Avaliable online: https://www.silabs.com/Support%20D ocuments/TechnicalDocs/CP2200.pdf (accessed on 15 01 2018).

[28] B. A. Forouzan, *TCP/IP protocol suite*, McGraw-Hill, New York 2010.

[29] *FreeMODBUS - A Modbus ASCII/RTU and TCP implementation*. Avaliable online: https://www.fre emodbus.org/index.php?idx=92 (accessed on 19 03 2018).