Protocolo Modbus

ELC1106 - Redes de Comunicação de Dados

Diana Eva Villegas
Franciuíne Barbosa da Silva de Almeida
Meryane Fernandes Machado

Modbus

- Desenvolvido em 1979;
- Utiliza a técnica servidor-cliente para estabelecer comunicação entre os dispositivos;
- Protocolo padrão utilizado em redes industriais, já que diversos controladores e ferramentas para desenvolvimento de sistemas supervisórios utilizam este protocolo,
- Grande simplicidade e facilidade de implementação, visto que este protocolo define uma estrutura de mensagens compostas por bytes, que os mais diversos tipos de dispositivos são capazes de reconhecer.

Modelo servidor-cliente

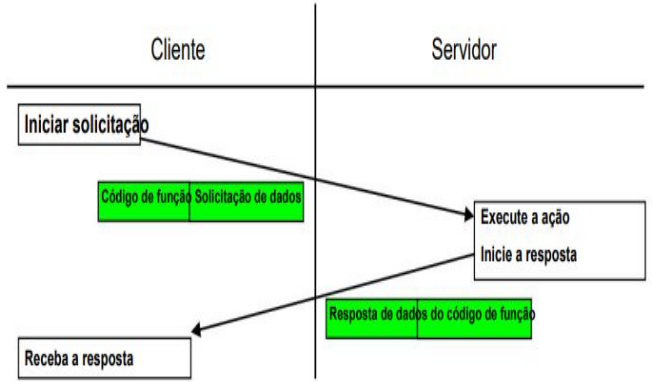


Figura 4: Transação MODBUS (sem erros)

Mensagem

Cada tipo de transmissão possui um encapsulamento da mensagem dividido em blocos de identificação, que permitem que os dispositivos servidores leiam os dados transmitidos.

Os campos da mensagem são: início, fim, endereço, função, dados e *cheksum*.

Campo endereço

- O campo endereço identifica o dispositivo para o qual o cliente deseja se comunicar;
- O endereço 0 é reservado para broadcast e os endereços que correspondem à faixa de 0 a 247 são considerados válidos para recepção e transmissão de dados;
- Quando um servidor retorna uma resposta ao cliente, ele envia seu endereço na mensagem para que possa ser identificado;
- O dispositivo cliente não possui endereço.

Campo endereço

Endereço	Descrição
0 (Zero)	Endereço de broadcast
1 a 247	Endereços dos servidores
248 a 255	Endereços reservados

Campo função

- O campo função determina o comando a ser executado;
- As funções válidas estão na faixa de 1 a 255;
- Caso haja erro na transmissão, o servidor retorna o código do erro neste campo;
- Caso não haja problemas na transmissão, o campo é devolvido com o mesmo valor da função solicitada.

Campo *checksum*

- Responsável pela verificação da integridade e autenticação dos dados enviados;
- É verificado ora pelo dispositivo servidor ora pelo dispositivo cliente;
- No modo ASCII utiliza o método LRC, e no modo RTU o CRC.

Comandos de erro

- O servidor não recebe a mensagem por problemas de comunicação. Nesse caso, nenhuma mensagem é retornada e o cliente verifica o timeout;
- O servidor recebe a mensagem, mas há problemas na comunicação. Para este, também não é retornada uma mensagem e o cliente verifica o timeout.
- O servidor recebe a mensagem sem erros de comunicação, mas não é capaz de atender à solicitação. Nesse caso, o código de uma exception é retornada para o cliente , informado a natureza do erro.

1	Função inválida
2	Registrador inválido
3	Valor de dado inválido
4	Falha no dispositivo
5	Estado de espera
6	Dispositivo ocupado
7	Não reconhecimento
8	Erro de paridade

Campo dados

O campo de dados é, também, formado por dois caracteres em modo ASCII, variando de 00H a FFH, mas, para o modo RTU, possui um byte. Neste campo existem informações relacionadas com o código da função no campo de funções, como por exemplo, o número de variáveis discretas a serem lidas ou ativadas.

Tipos de transmissão

ASCII

- Neste modo de transmissão, cada byte é representado por dois caracteres da tabela ASCII;
- Costuma consumir muitos recursos de rede, já que o monitoramento da rede para o início de uma mensagem é constante;
- A principal vantagem deste modo de transmissão é a possibilidade de haver grandes intervalos entre o envio de dados de uma mesma mensagem.

RTU

- Possui dois caracteres hexadecimais de 4 bits para cada mensagem de 8 bits;
- Sua principal vantagem é a maior densidade de caracteres por mensagem, melhorando o desempenho da transmissão.

ASCII

- Todos os dispositivos são responsáveis por decodificar o campo de endereço;
- O início da mensagem é identificado por dois pontos (:). 3AH em hexadecimal;
- 10 bits por palavra:
    - 1 Start bit | 7 data bits, sem paridade | 2 stop bits;
    - 1 Start bit | 7 data bits, paridade PAR | 1 stop bits;
    - 1 Start bit | 7 data bits, paridade IMPAR | 1 stop bits.

ASCII

- O framing de dados é composto dos números 0 ao 9 e do A ao F no padrão hexadecimal, ou seja, dos valores 30H a 39H e 41H a 46H, respectivamente;
- O término de cada mensagem é composto pelos conjuntos Carriage Return (CR) e Line Feed (LF), respectivamente representados pelos valores ASCII 0DH e 0AH.

ASCII

Início	Endereço	Função	Dados	LRC	Fim
3AH :	2 *chars*	2 *chars*	N *chars*	2 *chars*	CRLF

LRC

- Soma todos os bytes da mensagem, excluindo o início;
- Descarta os carries;
- Subtrai o campo final do valor FF em hexadecimal, aplicando complemento de um;
- Adiciona um, submetendo o valor à lógica de complemento de dois.

Exemplo

3AH | 30H 43H | 30H 33H | 30H 30H 30H 41H | 30H 30H 30H 31H | 45H 35H | 0DH 0AH

- Convertendo para o padrão decimal:

: | 12 | 3 | 10 | 1 | 230 | CLRF

- Ou no padrão hexadecimal:

: | 0CH | 03H | 000AH | 0001H | E6H | CLRF

- Realizando a soma destes valores:

12 + 3 + 10 + 1 = 26 (ou 1AH em hexadecimal)

Exemplo

- Aplicando complemento de 1:

FFH - 1AH = E5H

- Aplicando complemento de 2:

E5H + 01H = E6H

- Criptografando este valor de acordo com a tabela ASCII:

E = 45H

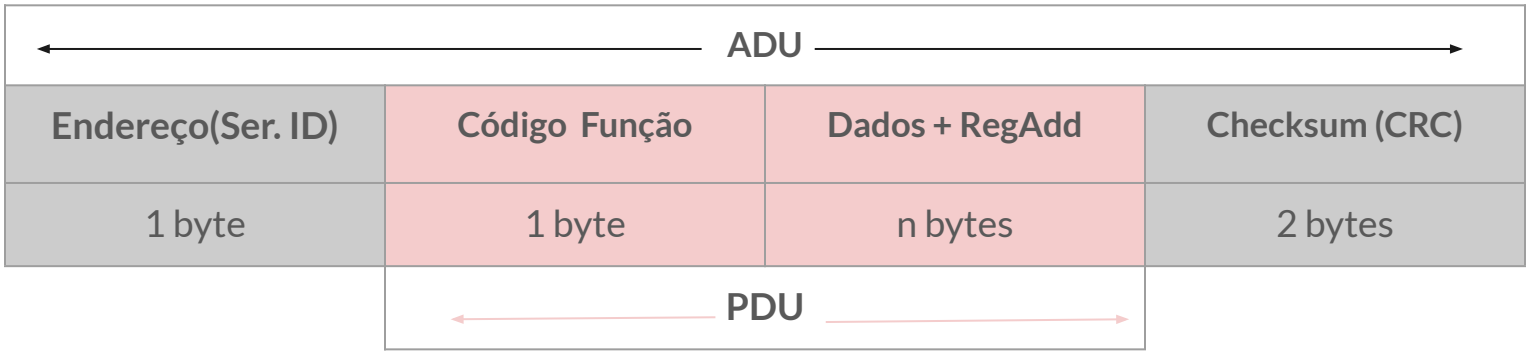
6 = 36H

RTU

- Sua principal vantagem é a maior densidade de caracteres por mensagem, melhorando o desempenho da transmissão;
- Diferente do modo ASCII, o modo RTU não possui bytes que indiquem o início e fim do framing;
- Para identificar cada framing, não deve haver nenhuma transmissão de dados por um período mínimo equivalente a 3,5 vezes o tamanho da palavra de dados.

Encapsulamento de dados

De forma genérica, o Modbus RTU define o encapsulamento em 4 partes distintas, cada uma com uma função específica para formar a interpretação final da mensagem.



Encapsulamento de dados

- Servidor ID:

  Para cada Servidor na rede é atribuído um único endereço de 1 a 247 e quando o Cliente requisita os dados, o primeiro byte da mensagem contém o endereço do Servidor. Dessa forma, cada Servidor sabe se deve ou não ignorar a mensagem.

Encapsulamento de dados

- Código de Função:

  O segundo byte da mensagem enviada pelo Cliente é o código de função e este número diz ao Servidor qual tabela deve acessar e se deve somente ler ou ler e escrever.

Encapsulamento de dados

Código de Função	Ação	Nome da Tabela
01 (01 hex)	Read	Saídas Discretas (Bobinas)
05 (05 hex)	Write single	Saídas Discretas (Bobinas)
15 (0F hex)	Write multiple	Saídas Discretas (Bobinas)
02 (02 hex)	Read	Entradas Discretas (Contatos)
04 (04 hex)	Read	Registro de Entrada Analógica
03 (03 hex)	Read	Registro de Saída Analógica
06 (06 hex)	Write single	Registro de Saída Analógica
16 (10 hex)	Write multiple	Registro de Saída Analógica

Encapsulamento de dados

- CRC:

  O CRC é uma checagem de redundância cíclica e trata-se de dois bytes adicionados ao final de cada mensagem Modbus para detecção de erro. Cada byte na mensagem é utilizado para calcular o CRC e o dispositivo receptor também calcula o CRC e realiza a comparação com o recebido pelo Cliente.

  Se qualquer bit enviado na mensagem estiver incorreto, o CRC calculado será diferente do recebido e um erro será gerado.

Exemplo

Imagine dois dispositivos interligados por uma rede serial que se comunicam em Modbus. Em determinado momento, o Cliente precisa acessar o Servidor com endereço 17(dec) e ler os valores de registro de saída analógica armazenados entre os endereços 40108 a 40110.

Exemplo

A requisição do Cliente para o exemplo fica da seguinte forma:

11 03 006B 0003 7687

- 11: É o endereço do Servidor (11hex = 17)
- 03: Código de Função 03 => ler registo de saída analógica
- 006B: O endereço de dados do primeiro registrador requisitado (006B hex = 107, + 40001 de offset = 40108)
- 0003: O número total de registros requisitados (ler 3 registos de 40108 a 40110);
- 7687: o CRC (cyclic redundancy check) para checagem de erro.

Exemplo

ADU			
Endereço(Ser. ID)	Código Função	RegAdd + Dados	Checksum (CRC)
11(hex)	03(hex)	006B 0003	7687
	PDU		

Exemplo

Assim que o dispositivo com o endereço 17 receber a mensagem do cliente, ele responderá com a seguinte mensagem:

11 03 06 AE41 5642 4340 49AD

Exemplo

- 11: É o endereço do cliente (11hex = 17)
- 03: Código de Função 03 = ler registo de saída analógica
- 06: O número de bytes de dados contidos na mensagem (3 registros x 2 bytes cada = 6 bytes)
- AE41 (44609 (dec)): O dado armazenado no registro 40108
- 5642 (22082 (dec)): O dado armazenado no registro 40109
- 4340 (17216 (dec)): O dado armazenado no registro 40110
- 49AD: o CRC (cyclic redundancy check) para checagem de erro.

Modbus/TCP

O Modbus TCP é uma implementação do protocolo Modbus baseado em TCP/IP. O Modbus/TCP usa Ethernet para transmissão e seu limite superior de taxa de transmissão é alto, a eficiência de transmissão é alta e o custo também é maior que o Modbus/RTU.

Encapsulamento de dados

Para o encapsulamento de mensagem TCP, é utilizado o cabeçalho MBAP, composto por 7 bytes:

2 bytes para identificação da resposta da transação;

2 bytes para identificação do protocolo Modbus;

2 bytes de length para contagem dos próximos bytes;

1 byte para identificação do cliente remoto na rede.

Os demais campos da comunicação TCP são reservados para o código da função e dados.

Encapsulamento de dados

<table>
<tr><td colspan="3">ADU</td></tr>
<tr><td>MBAP</td><td>FUNÇÃO</td><td>DADOS</td></tr>
<tr><td>7 Bytes</td><td>1 Byte</td><td>N Bytes</td></tr>
<tr><td></td><td colspan="2">PDU</td></tr>
</table>

Simulação por eventos discretos

Os Modelos Discretos funcionam de forma que o estado do sistema muda somente no instante que ocorre um evento, ou seja, para todos os demais instantes de tempo, nada muda no sistema. Para isto, utiliza-se os conceitos de filas, como por exemplo, sistemas que podem ser modelados como filas de espera.

A simulação utilizada foi a conhecida como Simulação a Eventos Discretos que pode ser definida como, diretamente ou indiretamente, situações de fila, em que clientes chegam, aguardam em fila se necessário e então recebem atendimento antes de deixar o sistema. Com isso, existem apenas dois eventos que controlam a simulação: chegadas e atendimentos.

Simulação por eventos discretos

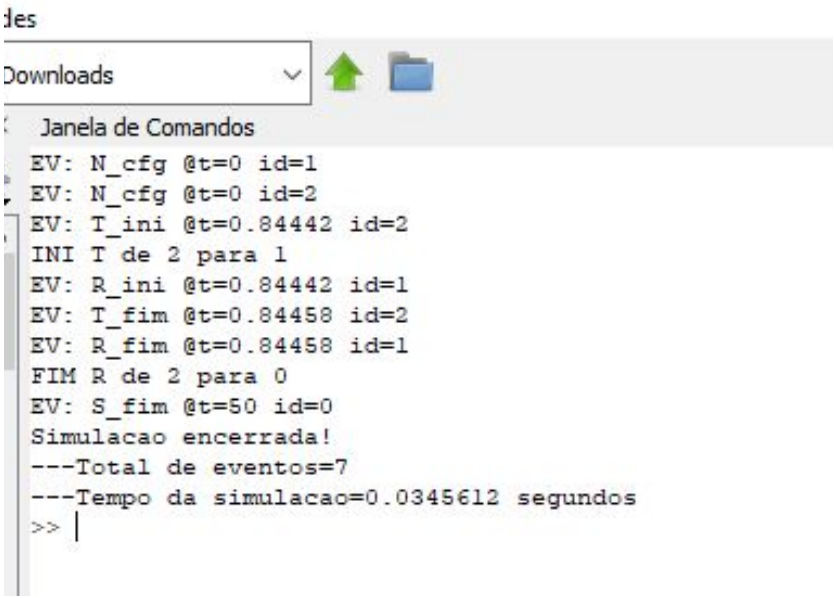
Alterações no código:

- Nos Parâmetros principais, o tempo simulação foi alterado de 100 para 50;
- O número de nós da rede foi de 2, a fim de mostrar como uma rede pequena se comporta.
- A mensagem foi alterada de “hello” para "006B", seguindo o exemplo anterior de leitura.
- O pacote passou a conter : origem (src), destino (dst), o código da função, os dados e o crc.

Simulação por eventos discretos: Eventos

```
switch tipo_evento
    case 'N_cfg' % configura nos, inicia variaveis de estado, etc.
        nos(id).Tx = 'desocupado';
        nos(id).Rx = 'desocupado';
        nos(id).ocupado_ate = 0;
        nos(id).stat = struct("tx", 0, "rx", 0, "rxok", 0, "col", 0);

        if(id == 2) % servidores id =1
          tabela(id).tab = struct('reg0', 0, 'reg1', 1);
        end


        % pacote contem origem (src), destino (dst), tamanho (tam) e os dados
        if (id ==2) % cliente id=2
        pct =  struct('src', id, 'dst', 0, 'tam', 2, 'codigo', 03, 'dado', '006B');

        e = evento_monta(tempo_atual+rand(1) , 'T_ini', id, pct);
        NovosEventos =[NovosEventos;e];
        end

  case 'T_ini' %inicio de transmissao
       if strcmp(nos(id).Tx, 'ocupado') % transmissor ocupado?
         tempo_entre_quadros = 0.2*8*pct.tam/taxa_dados; %20\% do tempo de transmissao
         e = evento_monta(nos(id).ocupado_ate+tempo_entre_quadros, 'T_ini', id, pct);
         NovosEventos =[NovosEventos;e];
       else
         if pct.dst == 0 %pacote de broadcast
            for nid = find(rede(id,:)>0) % envia uma copia do pacote para cada vizinho
              disp(['INI T de ' num2str(id) ' para ' num2str(nid)]);
              e = evento_monta((tempo_atual+tempo_prop), 'R_ini', nid, pct);
              NovosEventos =[NovosEventos;e];
            end
```

Simulação por eventos discretos: Resultado



Considerações Finais

Contudo, o que foi apresentado, podemos concluir que não existe um protocolo melhor ou pior que o outro, mas sim aplicações mais adequadas para uma ou outra arquitetura/protocolo.

Vantagens do Modbus

O Modbus é um protocolo simplificado que garante a transmissão extremamente rápida de dados. Uma estrutura de dados independente do fabricante também permite a comunicação entre dispositivos de diferentes fabricantes.

Vantagens do Modbus

O Modbus é um protocolo aberto. Isso significa que os fabricantes de graça podem incorporá-lo em seu equipamento. Ele tornou-se um protocolo padrão de comunicação na indústria, e atualmente é o meio mais comum de conexão industriais com dispositivos eletrônicos. Ele é amplamente utilizado por muitos fabricantes em muitos setores.

Desvantagens do Modbus

Com o advento de conceitos como Indústria 4.0, IoT e IIoT, 3G/4G e dispositivos wireless geograficamente distribuídos, surgem novas necessidades que dificultam bastante a utilização do Modbus TCP e seu modelo Cliente-Servidor.