Universidade Federal de Santa Maria
Departamento de Eletrônica e Computação
Redes de Comunicação de Dados

AVALIAÇÃO PARCIAL II
PROTOCOLO MODBUS

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

Santa Maria, RS
2022

1 INTRODUÇÃO

Desenvolvido em 1979, o protocolo MODBUS é uma estrutura de mensagem que utiliza a técnica servidor-cliente para estabelecer comunicação entre os dispositivos. Por ser um protocolo padrão, é utilizado em diversas redes industriais atualmente, já que diversos controladores e ferramentas para desenvolvimento de sistemas supervisórios utilizam este protocolo, isto se deve a sua 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.

Por se tratar de uma técnica servidor-cliente, é permitido que apenas um dispositivo, o cliente, inicie transmissões, enquanto os demais (servidores) apenas seguem as regras estabelecidas pelo cliente. Esta comunicação também pode ser chamada de *query*. O cliente pode acessar cada dispositivo através do seu endereço individual ou enviar uma mensagem a todos através de um *broadcast.* Este modelo pode ser visto no exemplo da figura 1 abaixo:

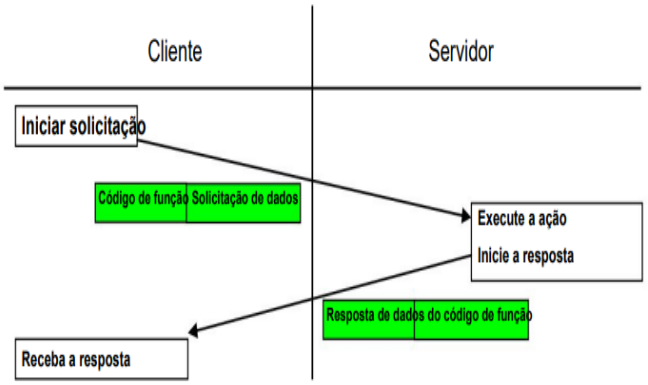

Figura 1 - Modelo de comunicação servidor-cliente

2 TRANSMISSÃO

Na especificação do protocolo Modbus estão definidos dois modos de transmissão para comunicação serial: ASCII e RTU. Esses modos definem como os bytes da mensagem são transportados e como as informações da mensagem são empacotadas em mensagens e descompactadas. Não é possível utilizar os dois

modos de transporte na mesma rede. O modo de transmissão pode ser selecionado através de outros parâmetros da porta serial, mas alguns dispositivos não podem ser selecionados por possuírem um modo de transmissão fixo, por exemplo, alguns CLPs e inversores utilizam o modo RTU por padrão.

Além disso, cada tipo de transmissão possui um *framing*, que nada mais é do que um encapsulamento da mensagem e sua divisão em blocos de identificação, que permitem que os dispositivos servidores, identifiquem, por exemplo, o início e o fim de uma transmissão, através de um caractere específico designado para esta marcação.

2.1 CAMPOS DA MENSAGEM

Além dos campos “início” e “fim”, que determinam onde a transmissão começa e termina, há mais três campos no framing de uma mensagem: endereço, função e dados.

2.1.1. CAMPO ENDEREÇO

No campo endereço, para o modo ASCII, existem dois caracteres e, para o modo RTU, 8 bits. 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. O endereço de broadcast é o único endereço que todos os dispositivos servidor reconhecem além do próprio. Quando um servidor retorna uma resposta ao cliente, ele envia seu endereço na mensagem para que possa ser identificado. Vale destacar, também, que o dispositivo cliente não possui endereço, apenas os demais.

Tabela 1 - Endereçamento do protocolo Modbus

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

2.1.2. CAMPO FUNÇÃO

Assim como no campo de endereço, o campo de funções é preenchido por dois caracteres em modo ASCII ou 8 bits em modo RTU. As funções válidas estão na faixa de 1 a 255, porém nem todas são implementadas e muitas são comuns para diversos tipos de controladores. O cliente sempre verifica o campo de funções porque, caso haja algum erro, será neste campo que o servidor retornará os problemas com a função solicitada com seu bit mais significativo em nível alto e, caso não haja problemas, o campo é devolvido com o mesmo valor da função solicitada.

Tabela 2 - Principais funções do protocolo Modbus

Código	Descrição
1	Leitura de bloco de bits do tipo saída
2	Leitura de bloco de bits do tipo entrada
3	Leitura do bloco de registradores holding
4	Leitura do bloco de registradores input
5	Escrita em um único bit de saída
6	Escrita em um único registrador holding
7	Leitura do conteúdo de 8 estados de exceção
8	Teste de verificação de erros de comunicação
11	Retorna contador de eventos
12	Retorna relatório de eventos
15	Escrita em um bloco de bits do tipo saída
16	Escrita em um bloco de registradores holding
17	Ler informações do dispositivo

20	Ler informações de um arquivo
21	Escrever informações em arquivo
22	Modificar conteúdo de registradores de espera
23	Ler e escrever em registradores em uma só transação
24	Ler conteúdo da fila de registradores
43	Identifica modelo do dispositivo

2.1.3. CAMPO DE 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.

2.1.4. CAMPO CHECKSUM

Checksum ou soma de verificação é a técnica responsável pela verificação da integridade e autenticação dos dados enviados. Este campo é verificado ora pelo dispositivo servidor ora pelo dispositivo cliente. No modo ASCII utiliza o método LRC, e no modo RTU o CRC. Estes métodos serão tratados de forma individual adiante.

2.1.5. RETORNO DE ERROS

Há três possibilidades de erro em uma transmissão:

- 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.

Tabela 3 - Principais possíveis exceptions

Código	Descrição
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

2.2 ASCII

Neste modo de transmissão, cada byte é representado por dois caracteres da tabela ASCII. Embora as mensagens sejam legíveis pela própria codificação ASCII, este modo costuma consumir muitos recursos de rede, já que o monitoramento da rede para o início de uma mensagem é constante. Por outro lado, a principal vantagem deste modo de transmissão é a possibilidade de haver grandes intervalos entre o envio de dados de uma mesma mensagem, sendo estes de até 1 segundo.

Quando o cliente inicia a transmissão, todos os dispositivos são responsáveis por decodificar o campo de endereço para determinar qual servidor receberá a mensagem. Neste caso, o início de uma mensagem é identificado pelo caractere “:” (dois pontos), representado no padrão hexadecimal pelo caractere 3AH.

A quantidade de bits de uma palavra em modo ASCII sempre será igual a 10, sendo possível que haja 3 diferentes configurações:

- 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.

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. No padrão decimal, de 0 a 15.

O término de cada mensagem é composto pelos conjuntos *Carriage Return* (CR) e *Line Feed* (LF), respectivamente representados pelos valores ASCII 0DH e 0AH.

Além disso, antes do fim de cada mensagem, existe uma lacuna para o *checksum* que utiliza o método *Longitudinal Redundancy Check* (LRC), abordado logo a seguir no presente documento.

Tabela 4 - Representação do framing do modo ASCII

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

2.2.1 LRC

O LRC é um método de detecção de erro para analisar dados transmitidos e armazenados utilizando paridade de bits. O cálculo deste método é aplicado a todos os campos da mensagem exceto o campo inicial (3AH), sendo este feito pelo servidor imediatamente após o envio da mensagem e, logo, comparado com o valor de LRC recebido.

O LRC é gerado através de 4 passos:

- 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.

Um exemplo deste cálculo pode ser visto abaixo, implementado por ARAÚJO, João ; BERRETTA DE LUCENA, Pedro:

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

No padrão decimal, este framing corresponde a:

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

Já em hexadecimal:

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

Realizando o somatório destes valores, temos: 12 + 3 + 10 + 1 = 26, ou, em hexadecimal, 1AH. O próximo passo é a aplicação do complemento de 1, resultando em FFH - 1AH = E5H. E então, submetendo ao complemento de 2, E5H + 01H = E6H. Criptografando este valor de acordo com a tabela ASCII, o “E” equivale a 45H e o valor 6 a 36H.

2.3 RTU

O modo RTU, ou *Remote Terminal Unit*, possui dois caracteres hexadecimais de 4 bits para cada mensagem de 8 bits e, para cada palavra de dados da mensagem, é enviado apenas um caractere hexadecimal. Em relação ao ASCII, pode-se dizer que sua principal vantagem é a maior densidade de caracteres por mensagem, melhorando o desempenho da transmissão. Nesta configuração, o tamanho da palavra de dados sempre será de 11 bits, tendo, também, três possíveis configurações:

- 1 Start bit | 8 data bits, sem paridade | 2 stop bits;
- 1 Start bit | 8 data bits, paridade PAR | 1 stop bits;
- 1 Start bit | 8 data bits, paridade IMPAR | 1 stop bits.

O início e o fim de uma mensagem em modo RTU é dado através da técnica *silent*, que define que não deve existir nenhuma palavra de dados para estes

campos por um mínimo de 3.5 vezes o tamanho da palavra. Dessa forma, é possível identificar estes blocos através da lacuna entre os blocos. Por exemplo, Para uma taxa de 4800 bps, o tempo total de envio de uma palavra é de 2291,6 us (11 x (1/4800)), ou seja, para identificar o início e fim de um *framing*, não deve ocorrer nenhuma transmissão por um período de 6,785ms (3,5 x 2291,6 us). Sendo assim, os dispositivos ficam observando os intervalos de tempo *silent* que, após detectado, dá se início ao recebimento da mensagem. No final da mensagem o servidor deve gerar outro intervalo de tempo, equivalente ao início, para caracterizar o final da mensagem.

Para que não haja conflito na detecção de início e fim da palavra, os dados devem ser transmitidos constantemente, caso contrário, ele descarta todos os dados que já recebeu e assume que o novo caractere que recebeu é o campo de endereço de uma nova mensagem, da mesma forma, se uma mensagem for recebida em um tempo menor que o *silent*, ocorrerá um erro.

Para um dispositivo configurado neste modo, o campo checksum do framing é gerado pelo método *Cyclical Redundancy Check* (CRC).

Tabela 5 - Representação do framing do modo RTU

Início	Endereço	Função	Dados	Checksum (CRC)	Fim
silent	1 *char*	1 *char*	N *chars*	2 *chars*	*silent*

2.3.1 CRC

Com este método é gerado um valor de 16 bits, onde os 8 bits menos significativos são enviados primeiros e os 8 bits mais significativos após. Este método exige uma maior complexidade para o seu desenvolvimento, porém é mais confiável.

2.3.2 EXERCÍCIO EXEMPLO

Para o framing a continuação:

Tabela 6 - Encapsulamento RTU para resolução de exercícios.

ADU			
Endereço	**Função**	**Dados**	**Checksum (CRC)**
1 *byte*	1 *byte*	N *bytes*	2 *bytes*
	PDU		

Usar a Tabela 7 para realizar a leitura do registrador 30009 que contém o valor 000A no servidor número 17(dec).

Tabela 7 - Tabela de Código de Função RTU para resolução de exercícios.

Função	**Ação**	**Alvo**		
		Tipo	**Numeração**	**Endereço**
01h	Leitura	Saídas Digitais (output coil)	1…9999	0000 … 270Eh
02h	Escrita única			
0Fh	Escrita múltipla			
02h	Leitura	Entradas Digitais	10001 … 19999	0000 … 270Eh
04h	Leitura	Entrada Analógica	30001 … 39999	0000 … 270Eh
03h	Leitura	Saídas Analógicas (output holding registers)	40001 … 49999	0000 … 270Eh
06h	Escrita única			
16h	Escrita múltipla			

Sem erro, é obtido o framing request e response:

Tabela 8 - Representação do framing do modo RTU

Endereço	**Função**	**Dados**		**Checksum (CRC)**
		RegAdd	*NRegs*	
11(hex)	*04*	0008	0001	2 bytes

Tabela 9 - Representação do framing do modo RTU

Endereço	**Função**	**Dados**		**Checksum (CRC)**
		NBytes	*Data*	
11	*04*	02	000A	2 bytes

Usando a tabela 7, realizar a escrita no registrador 40002 no servidor número 17(dec).

Tabela 10 - Representação do framing do modo RTU

Endereço	**Função**	**Dados**		**Checksum (CRC)**
		RegAdd	*Data*	
11	*06*	0001	0003	2 *chars*

Tabela 11 - Representação do framing do modo RTU

Endereço	**Função**	**Dados**		**Checksum (CRC)**
		regAdd	*Data*	
11	*06*	0001	000A	2 *chars*

3 MODBUS/TCP

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.

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.

Tabela 12 - Representação do framing do modo TCP

MBAP	Função	Dados
7 Bytes	1 Byte	N Bytes

A comunicação de soquete pode ser usada para transmitir mensagens Modbus/TCP. Socket é a camada de abstração de middleware para comunicação entre a camada de aplicação e o conjunto de protocolos TCP/IP. É um conjunto de interfaces que oculta o complexo conjunto de protocolos TCP/IP por trás da interface Socket. Para os usuários, um simples conjunto de interfaces é tudo, deixe o Socket organizar os dados para cumprir o protocolo especificado. Várias funções de Socket podem ser usadas para programação de Socket no desenvolvimento de programas. A implementação, conforme mostrado na figura 2.

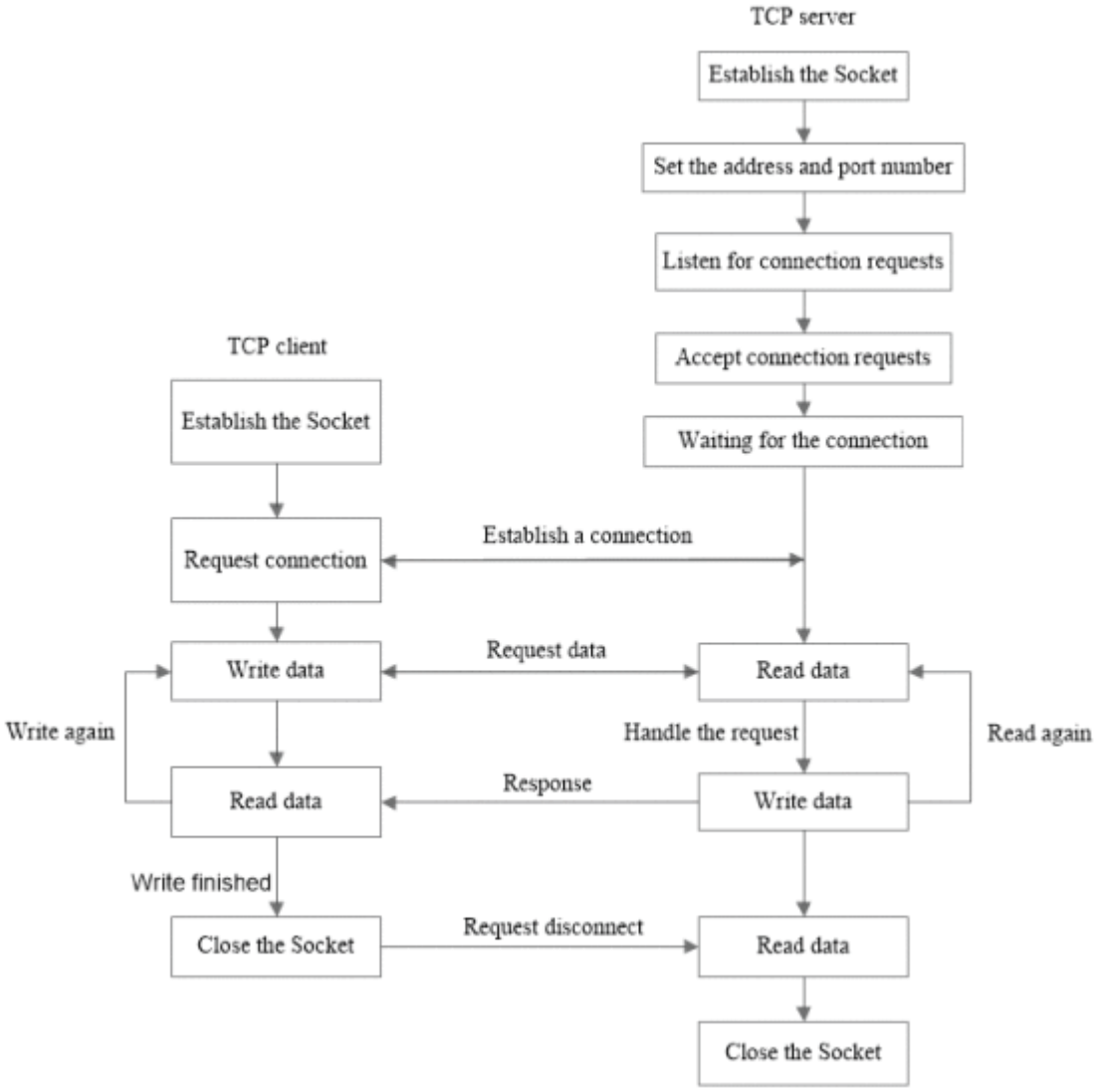

Figura 2 - Fluxo de comunicação de socket

3.1 PROJETO DE REDE DE COMUNICAÇÃO UTILIZANDO MODBUS/TCP

A comunicação de soquete é dividida em lado do servidor e lado do cliente. O processo de envio e recebimento de mensagens do servidor para o cliente é mostrado na Figura 3. Para enviar os dados primeiro, uma conexão deve ser estabelecida. Ao estabelecer uma conexão, primeiro é configurada a janela para receber as mensagens, posteriormente é definido o endereço IP e o número da porta do dispositivo de destino, na sequência usa-se a função de conexão para estabelecer um Socket e, em seguida, é executada a função de envio para enviar os dados. Ao receber uma mensagem, primeiro é armazenado os dados recebidos

enviados para a janela em uma matriz, converta os dados da matriz para o número hexadecimal necessário e exibindo o mesmo.

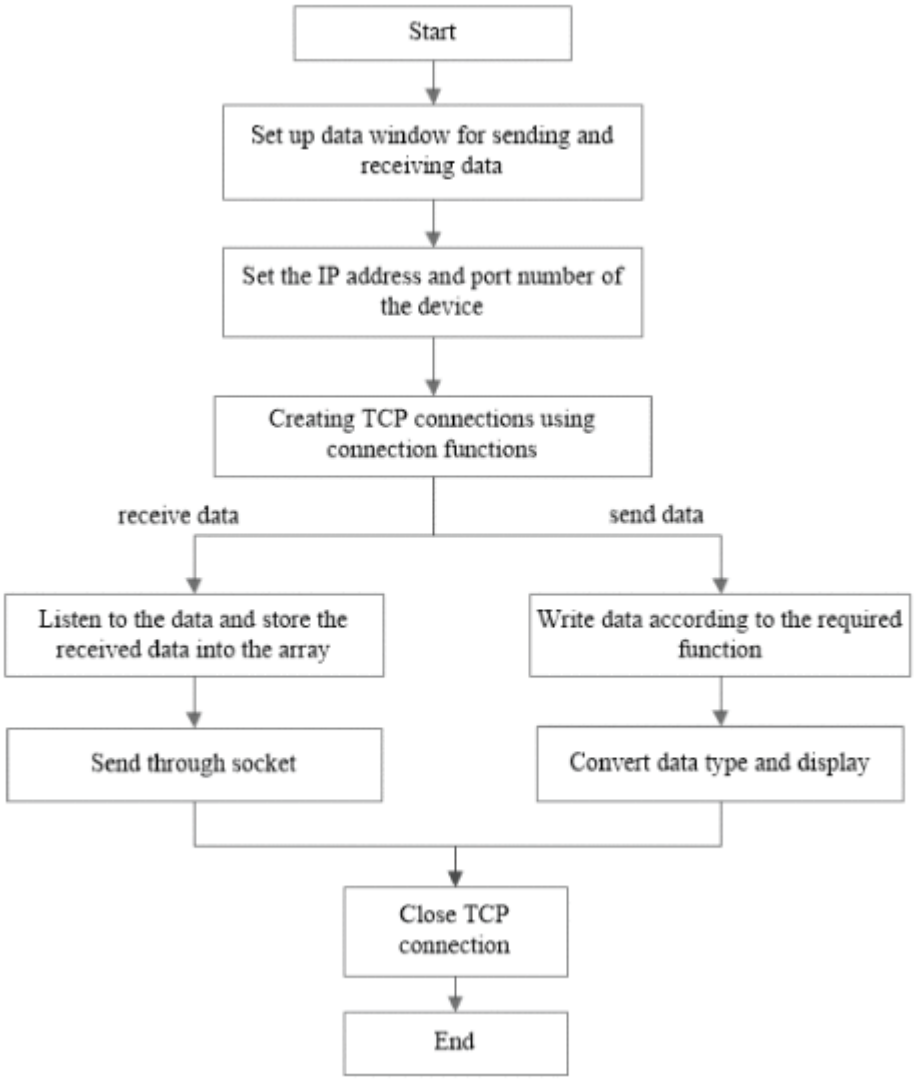

Figura 3 - Fluxo de controle Modbus/TCP

4 SIMULAÇÃO

Antes de iniciar a simulação foi necessário entender o que é a simulação por eventos discretos. Nesse sentido, esta trata sobre os Modelos Discretos que funcionam no estado do sistema, mudando 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.

Para este trabalho 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.

Algumas alterações foram realizados no código, tais como:

- 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.

A relação de eventos de transmissão(inicio e fim), como de recepção(início e fim) é apresentada na figura 4.

```
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
```

Figura 4 - Fluxo de controle Modbus/TCP

5 RESULTADOS

O código ajustado para simulação por eventos discretos do protocolo Modbus RTU consta no repositório: dianaedv/ELC1106: Redes de Comunicação de Dados. Trabalho Final: Modbus RTU. (github.com).

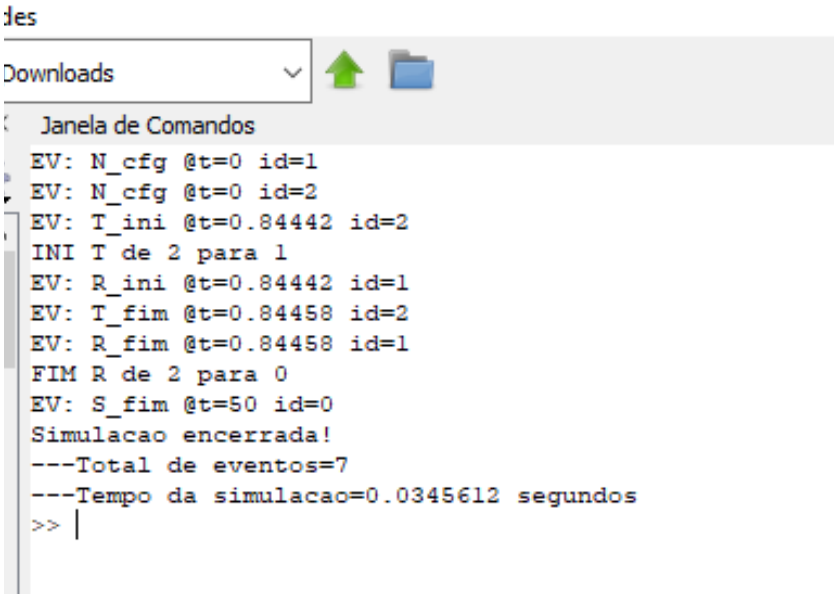

Figura 5 - Fluxo de controle Modbus/TCP

6 VANTAGENS DA APLICAÇÃO DO PROTOCOLO 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.

Além disso, 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.

Se comparado ao protocolo MQTT:

- Não necessita de um intermediário (broker), pois a conexão é feita ponto a ponto.
- Garante o recebimento da informação no destinatário final, ou seja, o cliente sabe que a sua pergunta chegou ao servidor e o servidor sabe que a resposta chegou ao cliente.

- Servidores que permitem múltiplas conexões conseguem atender a múltiplos clientes simultaneamente, o que torna possível, por exemplo, que um software SCADA e um supervisório mobile se comuniquem com o mesmo dispositivo para ler seus dados.
- Clientes podem obter informações do servidor a qualquer momento, sem ter que aguardar o envio espontâneo por parte dos clientes.
- Praticamente todos os softwares SCADA(sistemas supervisórios de controle e aquisição de dados) e supervisórios em geral dão suporte ao protocolo, pois há décadas é largamente utilizado no meio industrial.
- É possível que um dispositivo servidor também funcione como gateway de uma rede Modbus RTU. Desta forma, os clientes podem se comunicar com todos os dispositivos na mesma conexão simplesmente endereçando as suas perguntas ao dispositivo correto.

7 DESVANTAGENS DA APLICAÇÃO DO PROTOCOLO MODBUS

O modelo Cliente-Servidor do protocolo Modbus TCP tem sido utilizado há anos em aplicações de automação, resolvendo uma grande gama de aplicações. Porém, 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.

8 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.

7 REFERÊNCIAS BIBLIOGRÁFICAS

ALFA INSTRUMENTOS. **Protocolo de Comunicação Modbus RTU/ASCII, Alfa Instrumentos,** 2000.

CARLOS MÁRCIO FREITAS. **Protocolo Modbus: Fundamentos e Aplicações.** Disponível em: <https://embarcados.com.br/protocolo-modbus/>. Acesso em: 14 jul. 2022.

ARAÚJO, João ; BERRETTA DE LUCENA, Pedro. UFRN - Redes para Automação Industrial (Julho de 2003). **PROTOCOLO MODBUS**. Disponível em: <https://www.dca.ufrn.br/~affonso/FTP/DCA447/trabalho3/trabalho3_13.pdf>. Acesso em: 14 jul. 2022.

BARBOSA, Rui. FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO **Detecção de erros de comunicação de dados CRC**, 2011. Disponível em: <https://paginas.fe.up.pt/~ee06166/documentos/Deteccao_de_erros.pdf>.Acesso em: 14 jul. 2022.

NOGUEIRA, Fernando. Simulação a Eventos Discretos. Disponível em: https://www.ufjf.br/epd042/files/2009/02/Simulacao1.pdf. Acesso em: 28 de julho. 2022.