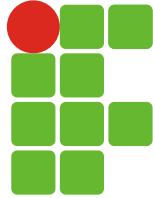


**INSTITUTO FEDERAL
ESPÍRITO SANTO**
Campus Linhares

Apostila de Redes Industriais

Rev. 10 de junho de 2022

Apostila de Redes Industriais



**INSTITUTO FEDERAL
ESPÍRITO SANTO**
Campus Linhares

Apostila de Redes Industriais

Alex Brandão Rossow

Linhares - ES

Junho de 2022

Sumário

I Teoria	17
1 Introdução	18
1.1 Introdução às redes de campo (redes <i>fieldbus</i>)	18
1.1.1 Tecnologia Chave para Automação	18
1.1.2 <i>Fieldbus</i> : um bom exemplo de comunicação em ação (entre os interessados)	19
1.1.3 Vantagens e desvantagens dos barramentos de campo em comparação com a fiação paralela	19
1.2 Comparação entre as redes de campo e o padrão 4-20mA	20
1.2.1 Cabeamento	20
1.2.2 Funcionalidades	23
2 Integração dos diferentes níveis de uma organização	25
2.1 Níveis da pirâmide da automação	25
3 Histórico das redes	29
3.1 Internet/ARPANET	29
3.2 Histórico das redes de campo	30
3.2.1 Modbus	30
3.2.2 PROFIBUS	31

3.2.3	HART	31
3.2.4	DeviceNet	32
3.2.5	Foundation Fieldbus	33
4	Alcance das redes	34
4.1	LAN	34
4.1.1	LAN sem fio (WLAN)	35
4.2	MAN	35
4.3	WAN	36
4.4	VPN	37
5	Topologia de rede	38
5.1	Topologia Ponto a ponto	38
5.2	Topologia Barramento	39
5.2.1	Topologina Barramento com derivação (com <i>spur</i>)	40
5.2.2	Topologia Barramento sem derivação (Daisy Chain)	42
5.3	Estrela	44
5.4	Arvore	45
5.5	Linha	46
5.6	Anel	48
5.7	Combinação de topologias	49
6	Sistemas de comunicação	50
6.1	Comunicação paralela e comunicação serial	51
6.2	Comunicação simplex, half-duplex e full-duplex	51
6.3	Comunicação síncrona e comunicação assíncrona	52

6.3.1	Comunicação síncrona	52
6.3.2	Comunicação assíncrona	52
6.4	Níveis lógicos	53
6.4.1	Níveis TTL	54
6.4.2	Nível CMOS 3.3V	55
6.4.3	Níveis lógicos do Arduino	56
6.5	Universal Asynchronous Receiver-Transmitter (UART)	57
6.6	Rede I2C (Inter-Integrated Circuit)	58
6.6.1	Resumo do funcionamento do protocolo I2C	59
6.7	Rede SPI (Serial Peripheral Interface)	63
6.7.1	Descrição dos sinais e ligações do SPI	63
6.7.2	Transmissão de dados	65
6.8	Padrão RS-232	67
6.8.1	Conexões e níveis de tensão no RS-232	67
6.8.2	Alcance e taxa de transmissão	69
6.8.3	Protocolo de transmissão (padrão UART)	69
6.8.4	Considerações finais sobre o RS-232	70
6.9	Padrão RS-485	71
6.9.1	Sinais elétricos no padrão RS-485	72
6.9.2	Controle por três estados (saída em alta impedância)	73
6.9.3	Velocidade x comprimento do cabo para barramento RS-485	74
6.9.4	Ligações a dois e quatro fios (+ GND)	75
6.9.5	Ligação do GND em um barramento RS-485	77
6.9.6	RS-422	78

6.10	Rede CAN (<i>Controller Area Network</i>)	79
6.10.1	Níveis de tensão do barramento	80
6.10.2	Camada de enlace de dados (<i>Data Link</i>) do CAN	81
7	Dispositivos de Rede	84
7.1	Repetidor	85
7.2	Hub	86
7.3	Ponte (Bridge)	87
7.4	Switch	88
7.5	Roteador	89
7.5.1	Roteamento	90
7.5.2	Classes de endereço IP	91
7.6	Gateway	92
7.6.1	Gateways para redes industriais	93
8	Foundation Fieldbus	96
8.1	Usos do Fieldbus	96
8.2	A rede Fieldbus	97
8.3	Protocolos de comunicação Foundation Fieldbus	97
8.4	Conceitos do padrão H1	98
8.4.1	Links	99
8.4.2	Dispositivos (<i>devices</i>)	100
8.4.3	Blocos e parâmetros	102
8.4.4	Ligações (<i>Linkages</i>)	102
8.4.5	Loops	102
8.5	Conceitos do padrão HSE	104

8.5.1	Dispositivo (<i>device</i>) HSE	104
8.5.2	Dispositivo de campo HSE	105
8.5.3	Linking Device	105
8.5.4	Dispositivo Gateway de E/S (<i>I/O Gateway Device</i>)	105
8.6	Tecnologia Foundation Fieldbus	105
8.6.1	Camada física da rede FF H1	105
8.6.2	Dispositivos virtuais	106
8.6.3	Camada de usuário (<i>User Layer</i>)	106
8.6.4	Blocos	107
II	Meios de Comunicação	111
9	Meios de Comunicação	112
10	Cabeamento metálico (cobre)	115
10.1	Cabeamento para redes seriais	115
11	Cabeamento para redes ethernet	119
11.1	Classes e categorias	119
11.2	Classificação de ambiente (MICE)	121
12	Fibra ótica	124
12.1	Fibra monomodo e multimodo	126
12.2	Condições ambientais	128

13 Wireless	129
13.0.1 IEEE 802.11 (Wi-Fi)	130
13.0.2 Bluetooth e Bluetooth Low Energy (BLE)	130
13.0.3 ZigBee	131
13.0.4 WirelessHART	131
13.0.5 ISA 100.11a (ISA100)	134
13.1 Considerações finais sobre os meios de transmissão	138
14 Distribuição do espectro de radiofrequência e Bandas ISM	139
14.1 Regulamentação do espectro de frequência no Brasil	139
14.2 Bandas de frequência ISM	140
III Considerações práticas sobre portas seriais (portas COM)	142
15 Introdução	143
16 Acesso a portas seriais (COM)	144
16.1 Introdução	144
16.2 Programas genéricos de acesso às portas seriais	144
17 Portas seriais (COM) virtuais	146
17.1 Portas seriais virtuais no Linux: Programa Socat	147
17.1.1 Instalação	147
17.1.2 Execução	147
17.1.3 Teste de funcionamento	148
17.1.4 Exemplo de uso das portas seriais virtuais no Linux	149
17.2 Portas seriais virtuais no Windows: Programa com0com	151

17.2.1	Instalação	151
17.2.2	Execução do programa	152
17.2.3	Interface do com0com, inserção de pares de portas	152
17.2.4	Troca de nome das portas	153
17.2.5	Remoção de portas	154
17.2.6	Teste de comunicação das portas criadas com o com0com	154
18	Universal Asynchronous Receiver-Transmitter - UART	156
18.1	Diferença entre UART e RS-232	156
18.2	Canais de comunicação no padrão UART	157
18.3	Quadro da mensagem UART	157
18.4	Exemplos de configuração	158
18.5	Comunicação com periféricos por UART	159
18.6	Disponibilidade de UARTs no Arduino	160
18.6.1	SoftwareSerial (porta UART por software)	162
18.6.2	Limitações da biblioteca SoftwareSerial	162
IV	Experimentos e considerações práticas sobre Modbus	164
19	Exemplos sensores e atuadores Modbus de baixo custo	165
19.1	Introdução	165
19.2	Sensores e atuadores Modbus de baixo custo	166
20	Exemplos de adaptadores para RS-485	168
20.1	Adaptadores USB/RS-485	168
20.2	Adaptadores TTL/RS-485	170

21 Considerações sobre clientes (mestres) e servidores (escravos) Modbus	173
21.1 Clientes (mestres) em uma rede Modbus	173
21.2 Servidores (escravos) em uma rede Modbus	174
21.3 Programas auxiliares para desenvolvimento Modbus	175
22 QModMaster - Cliente (mestre) Modbus	176
22.1 Instalação	177
22.1.1 Instalação no Windows	177
22.1.2 Instalação no Linux	177
22.2 QModMaster - Configuração de conexão serial para Modbus RTU	178
22.3 Configuração da conexão TCP/IP, para Modbus TCP	178
22.4 Estabelecendo a conexão	179
22.5 Realizando operações de leitura ou escrita	180
22.6 Endereço inicial de registradores e bobinas no qModMaster	182
23 pyModSlave - Servidor (escravo) Modbus	183
23.1 Instalação do pyModSlave	183
23.2 Execução do pyModSlave	184
23.3 Correção de erros do pyModSlave	185
23.3.1 Erro 01	185
23.3.2 Erro 02	186
24 ModbusPal - Servidor (escravo) Modbus	187
24.1 Iniciando o ModbusPal	187
24.2 Criação de um único escravo Modbus/TCP	188
24.3 Criação de múltiplos escravos Modbus/TCP	190

24.4 Criação de registradores em um escravo	191
24.5 Criação de bobinas (coils) em um escravo	193
24.6 Indicação de acesso a um dos escravos (leitura ou escrita)	194
24.7 Consideração sobre o endereço inicial dos registradores e das bobinas	195
25 Base Address no Modbus	196
25.1 Exemplo de mestre e escravo usando bases diferentes	197
25.2 Exemplo de mestre e escravo usando a mesma base	198
25.2.1 Bug de base do QModMaster no Linux	198
26 Servidor (slave) Modbus no Arduino	200
26.1 Instalação da Biblioteca	200
26.2 Circuito simulado	201
26.3 Código executado no Arduino	201
26.4 Testes de comunicação	208
26.4.1 Configuração da comunicação do SimulIDE com QModMaster no Windows	210
26.4.2 Configuração da comunicação com portas virtuais no Linux	212
26.4.3 Leitura de entradas discretas (discret input)	214
26.4.4 Escrita em bobinas (coils)	214
26.4.5 Leitura de bobinas (coils)	214
26.4.6 Leitura de registradores de entrada (input registers)	215
26.4.7 Escrita em registradores (holding registers)	215
26.4.8 Leitura de registradores (holding registers)	216

V Experimentos e considerações práticas sobre OPC-UA **217**

27 Introdução ao OPC-UA	218
27.1 Apresentação do protocolo OPC-UA	218
27.2 Biblioteca Open62541	219
28 Uso do padrão OPC UA associado com outras redes	220
28.1 OPC UA associado com PROFINET	220
29 Exemplo de transmissão de variáveis com OPC UA	221
29.1 Servidor OPC UA	222
29.2 Cliente OPC UA	224
29.3 Resultado do teste	225
29.4 Uso do FreeOpcUa Client	226
29.4.1 Instalação e execução do FreeOpcUa Client	226
29.4.2 Exemplo de uso do FreeOpcUa Client	227
30 Exemplo de aplicação com padrão OPC UA (testes com o Node-RED)	229
30.1 Código do servidor testado	230
30.2 Teste do servidor com o FreeOpcua Client	232
30.3 Sistema montado no Node-RED	234
30.3.1 Configuração dos blocos para envio da amplitude para o servidor OPC UA	235
30.3.2 Configuração dos blocos para leitura da variável “Variavel_Senoide” do servidor OPC UA	237
30.3.3 Configuração do bloco “chart”	239
30.4 Sistema executando	239

VI Experimentos e considerações práticas sobre MQTT	241
31 Introdução ao padrão MQTT	242
32 Exemplo de transmissão de variáveis com MQTT (com broker na nuvem)	244
32.1 Cliente publicador (publisher)	245
32.2 Cliente assinante (subscriber)	245
32.2.1 Resultado do teste	246
32.3 Uso do MQTT Explorer como subscriber	247
33 Eclipse Mosquitto (broker MQTT <i>open source</i>)	251
33.1 Introdução	251
33.2 Instalação e execução	251
33.2.1 Instalação	252
33.2.2 Iniciando e interrompendo a execução do Mosquitto	252
33.2.3 Obtenção de informações sobre o Mosquitto	252
33.3 Teste do Mosquitto com MQTT Explorer	254
34 Configuração do Mosquitto para acesso externo	256
34.1 Configuração do Mosquitto	256
34.2 Teste de conexão ao broker Mosquitto a partir de um telefone celular	258
34.2.1 Descobrindo o IP do broker	258
34.2.2 Uso do MQTIZER	259
35 Exemplo de aplicação do padrão MQTT (testes com o Node-RED)	263
35.1 Código executado	264
35.2 Sistema montado no Node-RED	266
35.2.1 Configuração dos blocos	267
35.3 Sistema executando	270

VII Experimentos e considerações práticas sobre CoAP (*Constrained Application Protocol*)

271

36 Introdução ao padrão CoAP	272
-------------------------------------	------------

37 CoAP vs MQTT	274
------------------------	------------

37.1 Considerações sobre as diferenças entre o CoAP e o MQTT	275
--	-----

38 CoAP User Agent Copper (extensão para Google Chrome)	277
--	------------

38.1 Instalação do Copper	277
-------------------------------------	-----

38.2 Execução do Copper	279
-----------------------------------	-----

39 Exemplo de aplicação com padrão CoAP (implementações)	281
---	------------

39.1 Servidor CoAP	282
------------------------------	-----

39.2 Cliente CoAP executando a função GET	289
---	-----

39.3 Cliente CoAP executando a função PUT	290
---	-----

40 Exemplo de aplicação com padrão CoAP (testes)	292
---	------------

40.1 Execução do servidor CoAP	292
--	-----

40.2 Execução do cliente CoAP com implementação da função GET	293
---	-----

40.3 Execução do cliente CoAP com implementação da função PUT	294
---	-----

41 Exemplo de aplicação com padrão CoAP (testes com o Copper)	296
--	------------

41.1 Acesso ao nó “time”, método GET	298
--	-----

41.2 Acesso ao nó “other/block”, método PUT	299
---	-----

41.3 Acesso ao nó “variável-4”, método POST	300
---	-----

42 Exemplo de aplicação com padrão CoAP (testes com o Node-RED)	303
--	------------

42.1 Acesso ao nó “time”, função GET	303
--	-----

VIII Experiments and practical considerations about Bluetooth	306
43 Introdução	307
44 Programação com Soquete Bluetooth (<i>Bluetooth Socket</i>) usando Python PyBluez	308
44.1 Endereços e dispositivos Bluetooth	308
44.2 Camada de Transporte no Bluetooth	309
44.3 Visualização da pilha de protocolos do Bluetooth	310
45 Bluetooth RFCOMM	313
45.1 Bluetooth protocol stack	313
45.2 O Framework Bluetooth e o RFCOMM	314
46 Configuração do Bluetooth Serial Port Profile (SPP) no Linux/Raspberry Pi	315
46.1 Configuração para habilitação da comunicação serial	315
47 Exemplo de comunicação com Bluetooth RFCOMM	318
47.1 Instalação da biblioteca “PyBluez” no Python	318
47.2 RFCOMM Server	319
47.3 RFCOMM Client	319
47.4 Habilitação do Bluetooth no Computador	321
47.5 Comunicação do “Serial Bluetooth Terminal” com o servidor programado em Python	322
48 Abertura de uma porta serial Bluetooth RFCOMM pelo comando “rfcomm watch”	330
48.1 Pareamento	331
48.2 Abertura da porta serial	332
48.3 Comunicação serial	333
48.3.1 Conexão pelo HTerm	334

IX	Experimentos e considerações práticas sobre I²C	335
49	Exemplo de projeto com Arduino e componentes I²C	336
49.1	Círculo simulado	336
49.2	Código executado	338
X	Exercícios	345
50	Exercícios	346
50.1	Exercícios sobre integração dos níveis de uma organização	346
50.2	Exercícios sobre Histórico das Redes industriais	347
50.3	Exercícios sobre Alcance das Redes	347
50.4	Exercícios sobre Topologia de Redes	347
50.5	Exercícios sobre Sistemas de Comunicação	347
50.6	Exercícios sobre os modelos ISO/OSI e TCP/IP	348
50.7	Exercícios sobre dispositivos de Redes	349
50.8	Exercícios sobre meios de comunicação	349
50.9	Exercícios sobre Redes de Campo Seriais	350
A	Manual do sensor de temperatura e umidade SHT20	352

Parte I

Teoria

Capítulo 1

Introdução

Esse capítulo apresenta algumas informações básicas sobre as redes industriais, seu histórico, principais características e aplicações.

1.1 Introdução às redes de campo (redes *fieldbus*)

Essa seção apresenta a tradução do artigo “*Fieldbus Basics*” publicado na página da empresa Kunbus Kunbus Industrial Communication 2020b

1.1.1 Tecnologia Chave para Automação

O primeiro passo na automação industrial foi a fiação paralela, onde todos os dispositivos eram conectados individualmente. No entanto, o número de componentes aumentou com o aumento do grau de automação, o que levou a um alto gasto com fiação. Agora, a fiação paralela foi amplamente substituída por sistemas *fieldbus* mais baratos e rápidos e redes de comunicação baseadas em Ethernet.

Os sistemas *Fieldbus*, criados na década de 1980, são hoje indispensáveis na indústria. Como um componente fixo de máquinas e instalações complexas, eles são usados principalmente na automação da manufatura. No entanto, o *fieldbus* também é usado na automação de processos e edifícios, bem como na engenharia automotiva.

Sensores e atuadores (chamados de “dispositivos de campo”), bem como motores, interruptores, inversores ou lâmpadas são conectados a controladores lógicos programáveis (SPS) / controladores mestre e de processo com a ajuda de barramentos de campo conectados por

fio e redes de campo seriais. Como tal, o *fieldbus* suporta a troca rápida de dados entre componentes individuais do sistema, mesmo em grandes distâncias. Mesmo cargas externas fortes não podem influenciar o robusto sistema de transmissão de sinal digital. Como a comunicação em uma rede industrial pode ser feita por um único cabo, foi possível diminuir a fiação consideravelmente em comparação com a fiação paralela (coneões 4-20mA, por exemplo).

Um *fieldbus* funciona na chamada operação mestre-escravo. Enquanto o mestre se encarrega de controlar os processos, as estações escravas realizam as tarefas parciais individuais.

Os *fieldbuses* diferem de acordo com sua topologia (estrela, linha, árvore ou anel), seu meio de transmissão e - dependendo do tipo - diferentes protocolos de transmissão (procedimento orientado a mensagem (redes mais tradicionais) ou procedimento de quadro de soma (visto na rede EtherCAT, por exemplo)). Os *fieldbuses* individuais também diferem no que diz respeito ao comprimento do cabo alcançável, o comprimento máx. número de bytes de dados por pacote de dados e o escopo da função. Dessa forma, funções adicionais como tratamento de alarmes, diagnóstico e troca de mensagens entre participantes individuais não são possíveis para *fieldbuses* distintos.

1.1.2 Fieldbus: um bom exemplo de comunicação em ação (entre os interessados)

No caso do *fieldbus*, a comunicação funciona tanto em termos de tecnologia quanto em termos de cooperação com os parceiros envolvidos. No entanto, isso era completamente diferente quando o *fieldbus* foi desenvolvido pela primeira vez. Agora que todas as partes (fabricantes, organizações, consumidores) trocaram ideias sobre os diferentes requisitos e expectativas, hoje existem até normalizações e normalizações nacionais e internacionais, bem como exames periódicos em laboratórios de ensaio.

1.1.3 Vantagens e desvantagens dos barramentos de campo em comparação com a fiação paralela

Vantagens das redes industriais de campo (*Fieldbus*):

- **Agilidade para implementação:** devido ao gasto reduzido com fiação, os sistemas *fieldbus* podem ser planejados e instalados mais rapidamente. Os *fieldbuses* comunicam-se através de apenas um cabo.

- **Confiabilidade:** Caminhos curtos para o sinal aumentam a disponibilidade e também a confiabilidade dos sistemas.
- **Confiabilidade em ambiente ruidoso:** Os *fieldbuses* oferecem maior proteção contra interferência, especialmente no caso de valores analógicos.
- **Uniformidade:** Devido à padronização dos protocolos de barramento à tecnologia de conexão unificada, equipamentos de diferentes fabricantes podem ser usados ??e substituídos. Dessa forma, não é necessário que todos os componentes individuais sejam do mesmo fabricante.
- **Flexibilidade:** Até mesmo expansões e mudanças podem ser executadas fácil e rapidamente com os barramentos de campo. Dessa forma, os sistemas podem ser adaptados de diferentes formas a novos requisitos.

Desvantagens do *fieldbus*:

- **Complexidade:** Como um *fieldbus* representa um sistema completo, é necessário pessoal qualificado para seu uso.
- **Custos:** Os componentes individuais do *fieldbus* são consideravelmente mais caros (embora o sistema como um todo possa ficar mais barato, conforme sua complexidade).
- **Perigos em caso de falha do barramento:** O barramento pode desconectado acidentalmente dos sensores e atuadores. Para evitar isso, sistemas de barramento redundantes devem ser usados, se necessário.

1.2 Comparação entre as redes de campo e o padrão 4-20mA

Nessa seção serão destacadas algumas diferenças entre a comunicação por barramento de campo (*fieldbuss*) e a comunicação analógica, tomando como exemplo o padrão 4-20mA.

1.2.1 Cabeamento

Para a transferência do sinal dos instrumentos de campo para a sala de controle, onde estarão os dispositivos de controles, como CLPs.

Na transmissão por sinal analógico (4-20mA) os cabos provenientes dos instrumentos passam por uma caixa de junção onde é feita a transição para um cabo composto por diversos pares. Isso tem apenas a função de organizar os condutores para percorrer uma longa distância, mas cada instrumento continua com um par de condutores exclusivo. A Figura 1.1 apresenta essa estrutura.

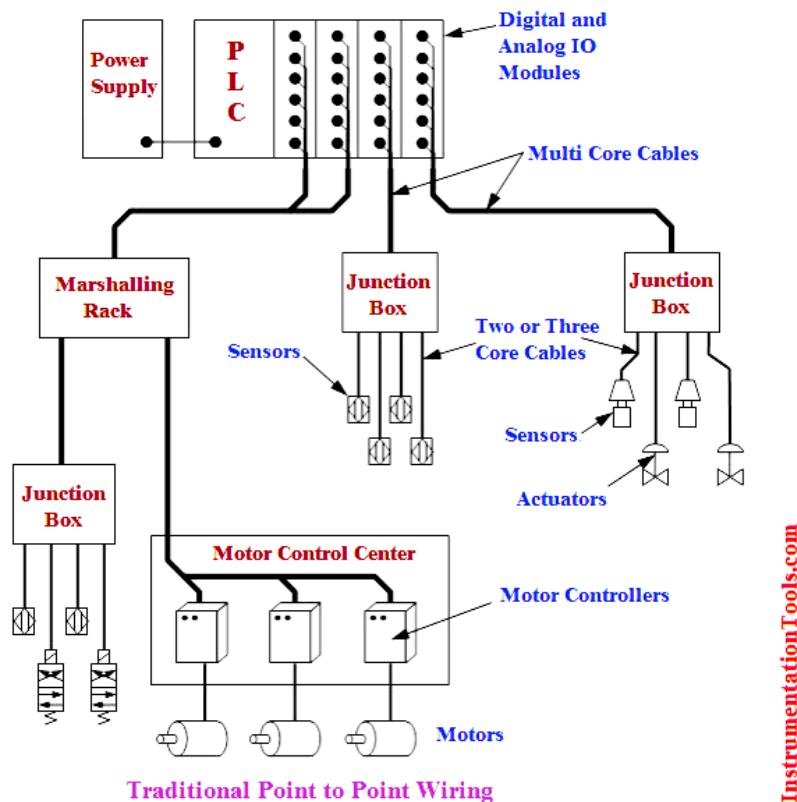


Figura 1.1: Cabeamento em um sistema tradicional 4-20mA Inst Tools 2020b.

A Figura 1.2 apresenta o esquema de uma caixa de junção para ligação por sinais analógicos.

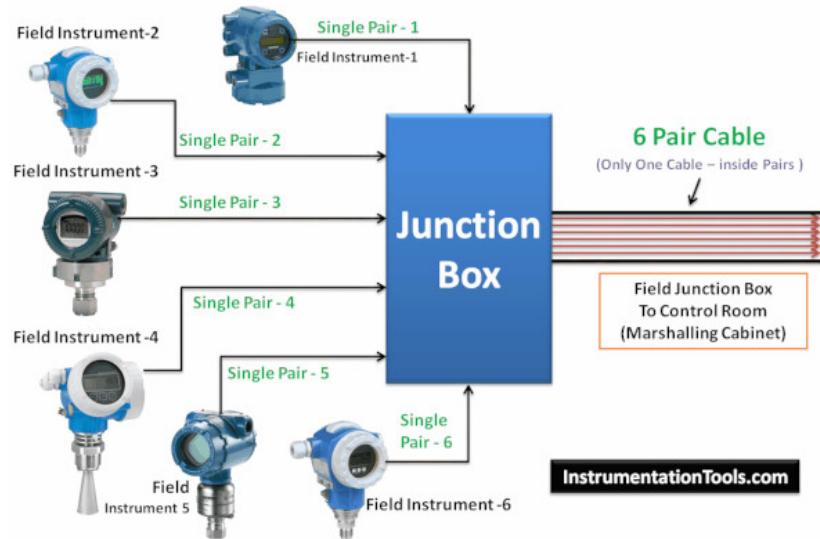


Figura 1.2: Caixa de junção para instrumentos analógicos (4-20mA) Inst Tools 2020a.

Na ligação por meio de redes de campo, um mesmo par de condutores pode ligar vários componentes de campo, conforme apresentado na Figura 1.3.

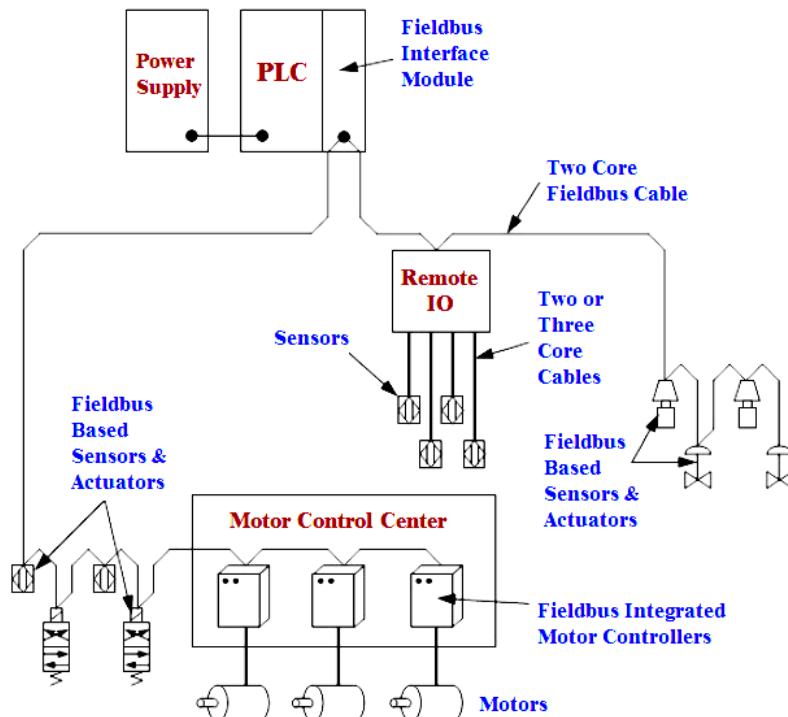


Figura 1.3: Cabeamento em uma rede de campo Inst Tools 2020b.

A Figura 1.4 apresenta uma caixa de junção para ligação do ponto onde se encontram

os instrumentos e a sala de controle, observa-se que a transmissão é por um único par de condutores.

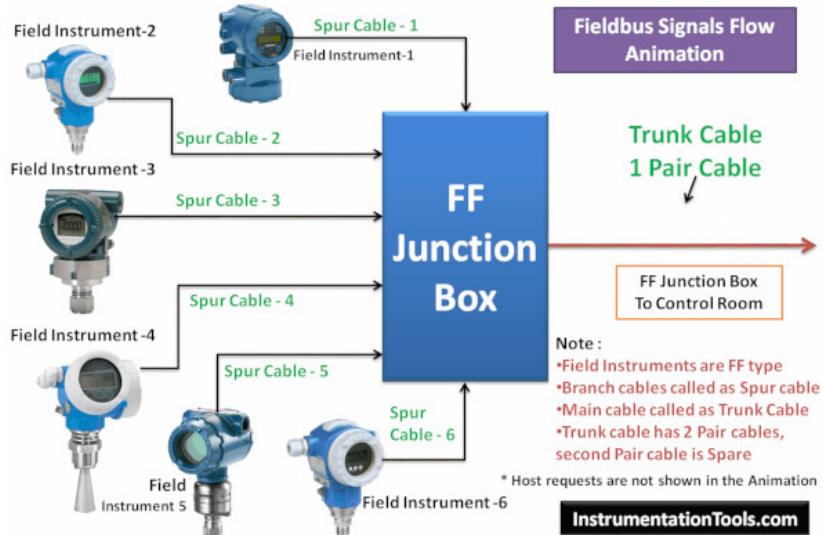


Figura 1.4: Caixa de junção para instrumentos em uma rede de campo Inst Tools 2020a.

1.2.2 Funcionalidades

Podemos listas as seguintes diferenças entre transmissores convencionais e transmissores com tecnologia *field-bus* AutomationForum.Co 2018:

Transmissores convencionais (4-20mA)

- Sinal analógico unidirecional
- Apenas uma variável
- Um par de cabos para cada instrumento
- Conversão de sinal mA para subsistema de I/O digitais.

Transmissores field-bus

- Sinal digital bidirecional
- Uma grande quantidade de informações é transferida
- Vários instrumentos em um cabo de barramento (multiponto)

- Nenhum sistema de conversão de I/O necessário
- Padrão aberto - instrumentos de qualquer fabricante podem estar no mesmo barramento de campo.

Diagnóstico pela rede: Os dispositivos ligados a uma rede de campo podem fornecer informações de diagnóstico de volta à sala de controle para ajudar a diagnosticar e localizar falhas de dispositivos. Essa função não pode ser feita pelos atuadores e sensores tradicionais, pela limitação de transmitir apenas uma variável Inst Tools 2020b.

Configuração e teste remotamente: Em uma rede de campo os sensores e atuadores, costumam oferecer recursos adicionais, como configuração do dispositivo remotamente e/ou teste através do barramento Inst Tools 2020b.

Capítulo 2

Integração dos diferentes níveis de uma organização

Neste capítulo é apresentado o conceito de integração entre os níveis de uma organização, indo desde o chão de fábrica com dispositivos de campo até o nível gerencial com programas de gestão e bancos de dados.

2.1 Níveis da pirâmide da automação

O conteúdo desta seção foi extraído, principalmente, do artigo “Redes Industriais” da Smar Smar Technology Company 2020a.

A tecnologia da informação tem sido determinante no desenvolvimento da tecnologia da automação alterando hierarquias e estruturas nos mais diversos ambientes industriais assim como setores, desde as indústrias de processo e manufatura até prédios e sistemas logísticos. A capacidade de comunicação entre dispositivos e o uso de mecanismos padronizados, abertos e transparentes são componentes indispensáveis do conceito de automação de hoje. A comunicação vem se expandindo rapidamente no sentido horizontal nos níveis inferiores (field level), assim como no sentido vertical integrando todos os níveis hierárquicos. De acordo com as características da aplicação e do custo máximo a ser atingido, uma combinação gradual de diferentes sistemas de comunicação oferece as condições ideais de redes abertas em processos industriais.

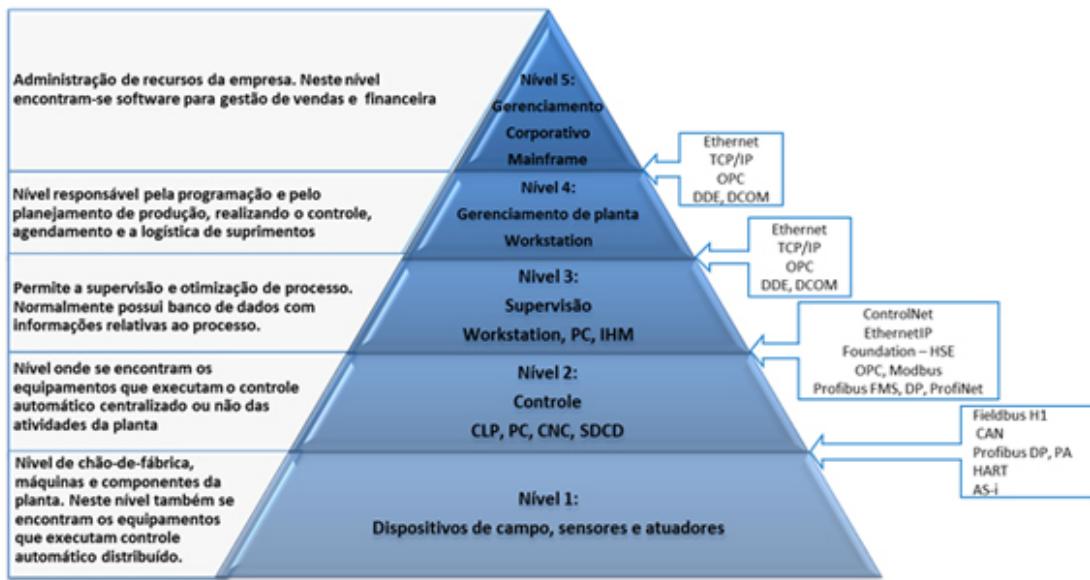


Figura 2.1: Níveis da pirâmide da automação (Smar Technology Company 2020a).

A Figura 2.1, apresenta os diferentes níveis de um sistema de automação. A mesma informação pode ser vista na Figura 2.2, onde é possível visualizar diferentes equipamentos e sistemas , como sensores, CLPs e o ERP (*Enterprise resource planning*), nessa figura o termo “*shop floor*” pode ser traduzido como “chão de fábrica”.

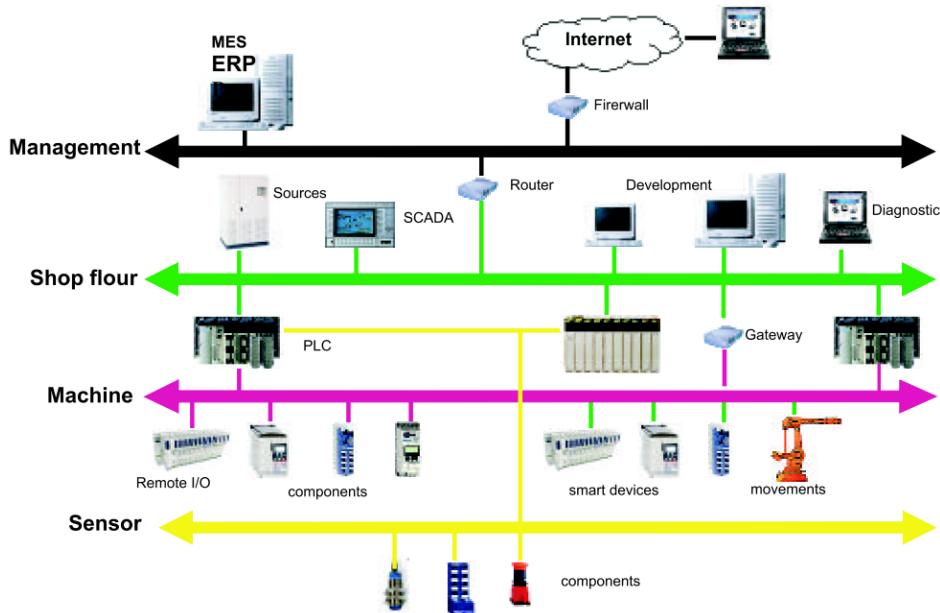


Figura 2.2: Visualização alternativa da pirâmide da automação (Schneider Electric 2006).

→ No **nível de atuadores/sensores** existem algumas redes industriais, onde podemos citar a AS-Interface (AS-i) onde os sinais binários de dados são transmitidos via um bar-

ramento extremamente simples e de baixo custo, juntamente com a alimentação (24 Vdc) necessária para alimentar estes mesmos sensores e atuadores. Outra característica importante é que os dados são transmitidos ciclicamente, de uma maneira extremamente eficiente e rápida. Veremos mais detalhes posteriormente.

→ No **nível de campo**, a periferia distribuída, tais como módulos de Entrada/Saída (E/S), transdutores, acionamentos (drives), válvulas e painéis de operação, comunicam-se com sistemas de automação via um eficiente sistema de comunicação em tempo real (PROFIBUS-DP ou PA, Foundation Fieldbus, HART, etc.). A transmissão de dados do processo e diagnósticos é efetuada ciclicamente, enquanto alarmes, parâmetros e também diagnósticos são transmitidos aciclicamente, somente quando necessário.

→ No **nível de célula**, os controladores programáveis, tais como CLPs e PCs comunicam-se uns com os outros, o que requer grandes pacotes de dados e um grande número de funções poderosas de comunicação. Além disto, uma integração eficiente aos sistemas de comunicação corporativos existentes, tais como: Intranet, Internet e Ethernet é um requisito absolutamente mandatório, o que várias redes podem suprir. A rede PROFINet, HSE (High Speed Ethernet), Ethernet IP, suportam dispositivos de campo simples e aplicações de tempo crítico, bem como a integração de sistemas de automação distribuídos baseados em componentes.

A Tabela 2.1 apresenta as estimativas de requisitos de velocidade e volumes de dados em cada camada.

	Volume de Dados	Tempo de Transmissão	Frequência de Transmissão
Enterprise Level	MBytes	Hora / Minuto	Dia / Turno
Cell Level	KBytes	Segundos	Horas/Minutos
Field Level	Bytes	100µs...100ms	10...100ms
Sensor Level	Bits	1 ...10ms	milisegundos

Tabela 2.1: Requisitos de comunicação de sistemas de automação industria Smar Technology Company 2020a.

Essa informação de velocidade e volume de dados também é apresentada de forma bastante intuitiva pela Schneider. A Figura 2.3 apresenta a descrição dos níveis do sistema com seus requisitos.

Level	Requirement	Volume to data to transmit	Response time	Distance	Network topology	Number of addresses	Medium
Management	Data exchange. Computer security . Standards between software packages.	Files Mbits	1 min	World	Bus, star	Unlimited	Electrical, optic, radio
Shop floor	Synchronisation of PLC's in the same data exchange automation cell in client/server mode with the control tools (HMI, supervision). Real-time performances.	Data Kbits	50-500 ms	2-40 Km	Bus, star.	10-100	Electrical, optic, radio
Machine	Distributed architecture. Embedded functions and exchange. Transparency. Topology and connection costs.	Data Kbits	5-100 ms (PLC cycle)	10 m to 1K m	Bus, star	10-100	Electrical, optic, radio
Sensor	Simplification of distribution wiring for power supply to sensors and actuators. Optimised wiring costs.	Data Bits		1- 100m	No constraint	10-50	Electrical, Radio

Figura 2.3: Requisitos de comunicação de sistemas de automação industrial Schneider Electric 2006.

A mesma informação pode ser visualizada, com algumas tecnologias possíveis, na Figura 2.4. Nesse gráfico o eixo “y” nomeado “Response time” significa velocidade, quanto maior o valor mais rápida é a rede, isso pode ser visto na quarta coluna da tabela apresentada na Figura 2.3.

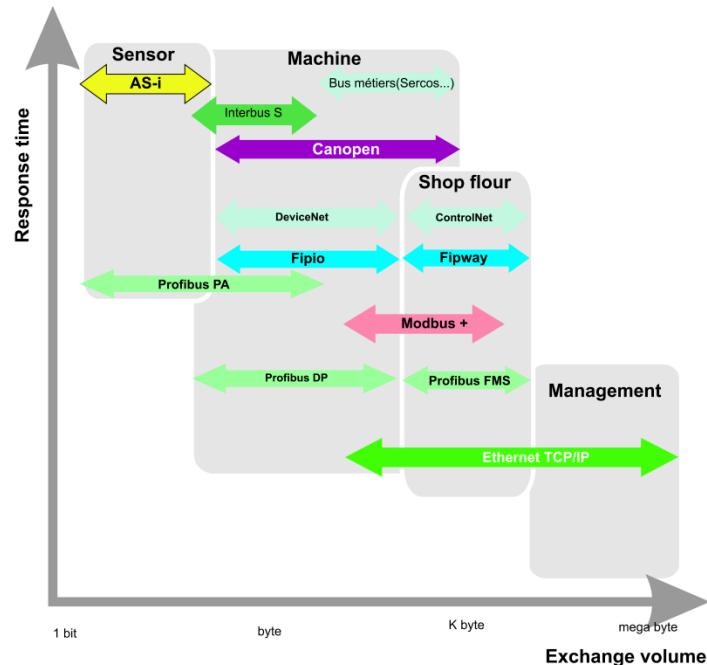


Figura 2.4: Requisitos de comunicação de sistemas de automação industrial Schneider Electric 2006.

Capítulo 3

Histórico das redes

Esse capítulo apresenta um breve resumo sobre o surgimento da internet e de algumas redes industriais.

3.1 Internet/ARPANET

Em Computer Hope 2019 é apresentada uma linha do tempo de 1961 até 2018, apresentando o momento do surgimento de tecnologias importantes para o desenvolvimento da internet. Como o foco desta apostila são as redes industriais, será apresentado apenas o surgimento da ARPANET, de modo a apresentar o contexto onde as redes industriais surgiram.

Abreviação de Advanced Research Projects Agency Agency Network, ARPANET ou ARPAnet começou o desenvolvimento em 1966 pela ARPA dos Estados Unidos. ARPANET era uma rede de longa distância que conectava muitas universidades e centros de pesquisa, foi a primeira a usar comutação de pacotes e foi o início do que consideramos a Internet hoje. A ARPANET foi criada para facilitar o acesso das pessoas aos computadores, melhorar os equipamentos de informática e ter um método de comunicação mais eficaz para os militares Computer Hope 2017.

A ARPANET começou quando os dois primeiros nós foram estabelecidos entre a UCLA e o SRI (Stanford Research Institute) em 1969, seguida logo depois pela UCSB e pela Universidade de Utah. A Figura 3.1 apresenta a estrutura da ARPANET em março de 1977.

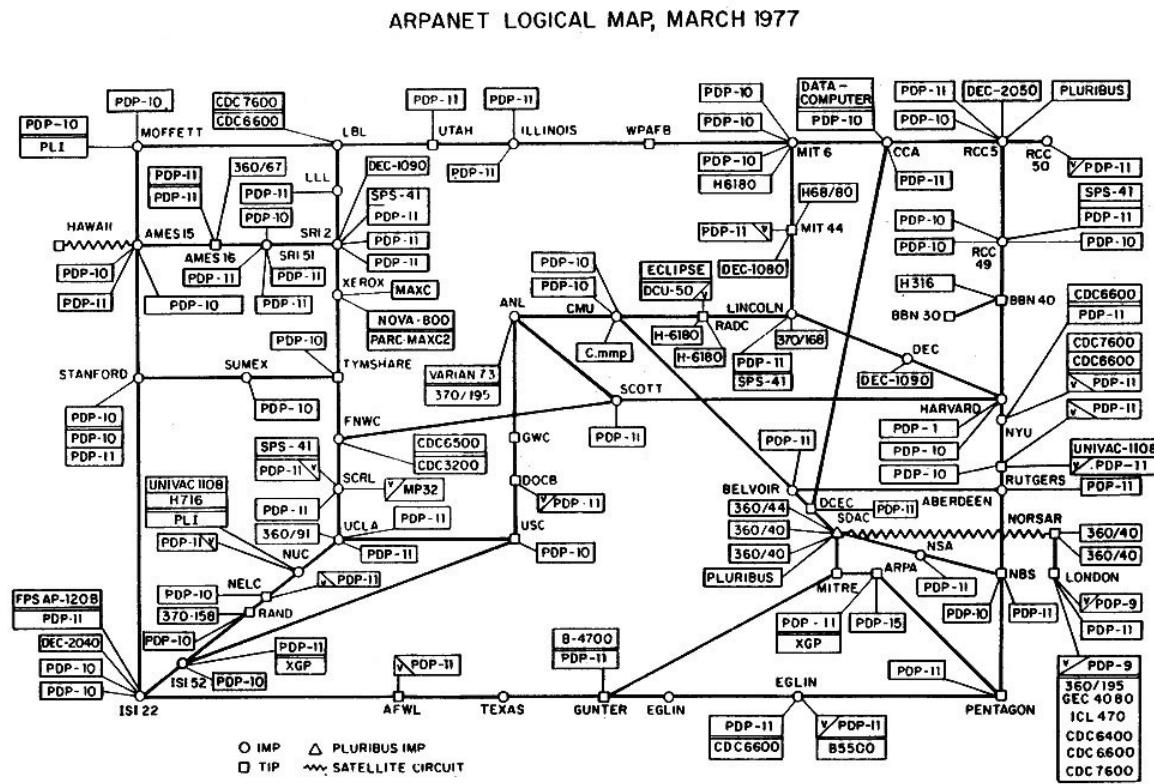


Figura 3.1: Estrutura da ARPANET em 1977 Computer Hope 2017.

A ARPANET concluiu sua transição para TCP/IP em 2 de janeiro de 1983, foi posteriormente substituída pela NSFNET em 1990 e, em seguida, desativada em 28 de fevereiro de 1990.

3.2 Histórico das redes de campo

Esta seção apresentará um breve resumo da criação de algumas redes de campo.

3.2.1 Modbus

O Modbus é um protocolo industrial **desenvolvido em 1979** para possibilitar a comunicação entre dispositivos de automação. Originalmente implementado como um protocolo de nível de aplicação destinado a transferir dados por uma camada serial, o Modbus foi ampliado para incluir implementações em comunicações seriais, TCP/IP e UDP (user datagram protocol)NI 2019.

Popularidade do Modbus

Simplicidade é a razão pela qual o Modbus é tão difundido. Também o fato de o Modbus ter sido criado por um dos maiores fabricantes de CLPs da época (Modicon, atualmente Schneider Electric) e torná-lo aberto e amplamente disponível. O Modbus também requer muito pouco em termos de espaço de código do processador ou RAM. Embora isso não seja tão importante hoje, dados os poderosos processadores e tecnologia disponíveis para nós, era muito importante nos primeiros anos da automação industrial, quando os processadores usavam tecnologia de 8 bits e recursos como RAM e ROM eram extremamente caros e escassos RTA 2020.

A verificação de mensagens é outra razão pela qual o Modbus se tornou tão popular. A verificação CRC e LRC significa que os erros de transmissão são verificados com precisão de 99

3.2.2 PROFIBUS

A história do PROFIBUS começa com um empreendimento de associação apoiado pelo poder público, que **começou em 1987** na Alemanha. Dentro deste empreendimento, 21 empresas e institutos uniram forças e criaram um projeto estratégico de fieldbus. O objetivo foi a realização e o estabelecimento de um fieldbus serial, o requisito básico era a padronização da interface dos dispositivos de campo.

Uma primeira etapa abordou a especificação dos protocolos para comunicações complexas PROFIBUS FMS (*Fieldbus Message Specification*), que era adaptado para demandas de comunicação tarefas de instalação. Um passo seguinte em 1993 realizou a conclusão da especificação para o protocolo mais simples e rápida PROFIBUS DP (*Decentralized Periphery*). Esse protocolo agora está disponível em três versões funcionalmente escaláveis DP-V0, DP-V1 e DP-V2 PROFIBUS Nutzerorganisation 2002.

3.2.3 HART

O protocolo Highway Addressable Remote Transducer (HART) é uma das opções iniciais de fieldbus, um protocolo digital de automação industrial. A maior vantagem é o fato de se basear no padrão convencional de 4 a 20 mA (cabeamento) que atende a transmissão de sinais analógicos. A fiação existente do sistema antigo ainda pode ser usada, e a operação paralela de ambos é possível.

Por sua base no padrão de 4 a 20 mA, o protocolo HART é um dos mais amplamente distribuídos protocolos da indústria. Os dados são transmitidos de acordo com o padrão Bell 202 usando a tecnologia de chaveamento de frequência. O protocolo Highway Addressable Remote Transducer oferece uma ótima facilidade de uso para usuários que usam o protocolo convencional de 4 a 20 mA que desejam usar um sistema “mais inteligente” com um escopo mais amplo de serviços.

O protocolo foi desenvolvido pela Rosemount Inc. em **meados da década de 1980** como um protocolo de comunicação proprietário que deveria ser usado nos próprios dispositivos fieldbus inteligentes da empresa. Isso logo evoluiu para o protocolo Highway Addressable Remote Transducer. Em 1986, foi alterado para um protocolo aberto. Desde então, as habilidades do protocolo foram expandidas por mudanças sucessivas nas especificações Kunbus Industrial Communication 2002.

3.2.4 DeviceNet

DeviceNet existe desde o **final da década de 1990** e ainda é amplamente utilizado em muitas instalações de fabricação em todo o mundo. Seu sucessor, EtherNet/IP, há muito assumiu o trono, mas DeviceNet foi mantido vivo devido à adoção lenta típica da indústria de manufatura de novas tecnologias e ao uso de dispositivos de gateway DeviceNet, que permitem que dispositivos que suportam outras redes controlem dispositivos DeviceNet.

O DeviceNet foi inventado pela Allen-Bradley (atualmente pertencente à Rockwell Automation), mas o IP foi entregue a uma entidade independente, a Open Device Vendors Association, uma associação global de automação industrial, agora conhecida como ODVA para torná-lo um protocolo "aberto" que seria adotado por outros OEMs de produtos industriais. ODVA define DeviceNet como o seguinte: DeviceNet é uma rede digital multiponto que conecta e atua como uma rede de comunicação entre controladores industriais e dispositivos de E / S, fornecendo aos usuários uma rede econômica para distribuir e gerenciar dispositivos simples em toda a arquitetura. DeviceNet utiliza CAN (Controller Area Network) para sua camada de enlace de dados, a mesma tecnologia de rede usada em veículos automotivos para comunicação entre dispositivos inteligentes Pyramid Solutions 2018.

Atualmente o DeviceNet é gerenciado pela independente North American Open DeviceNet Vendors Association (ODVA). A ODVA gerencia não apenas as especificações, mas também o posterior desenvolvimento e aderência à norma por meio de testes de conformidade. Mais tarde, a IDVA decidiu combinar DeviceNet como Common Industrial Protocol (CIP) com as seguintes tecnologias Kunbus Industrial Communication 2020a:

- EtherNet/IP (Industrial Protocol)
- ControlNet
- DeviceNet
- CompoNet

3.2.5 Foundation Fieldbus

O Foundation Fieldbus é um padrão fieldbus aberto para IEC 61158 e atualmente está sendo integrado ao Padrão Europeu EN 50170. Ele foi desenvolvido por um consórcio de fornecedores líderes de controle e instrumentação em todo o mundo. O padrão especifica dois níveis de comunicação:

- Um nível de comunicação de alta velocidade (HSE) no qual o tráfego entre os controladores, computadores, conversores de frequência e outros instrumentos.
- Um nível de comunicação de baixa velocidade (H1) no qual o tráfego entre o processo sensores e atuadores são manipulados.

Os primeiros dispositivos de ligação HSE/H1 são **lançados em 2001**. Por este motivo, vários empresas produziram soluções pragmáticas, por exemplo usando placas FF I/O ou ligações por meio de rede ControlNet, que se baseia em Ethernet Endress+Hauser 2020.

Capítulo 4

Alcance das redes

Neste capítulo é apresentada uma divisão das redes de dados quanto à extensão da cobertura.

4.1 LAN

As redes locais são construídas para pequenas áreas geográficas dentro do intervalo de 1 a 5 km, como escritórios, escolas, faculdades, pequenas indústrias ou um conjunto de edifícios.

Computadores pessoais, laptops e estações de trabalho em um escritório são geralmente interconectados usando redes LAN através das quais podemos compartilhar arquivos de dados, software, e-mail e acessar hardware como impressoras, FAX etc.

Ethernet, token ring, intercâmbio de dados distribuídos de fibra (FDDI), TCP / IP e modo de transferência assíncrona (ATM) são os protocolos mais comuns usados nesta rede.

As redes LAN são de vários tipos, dependendo do tipo de mídia, topologia e protocolos que estão usando para comunicação.

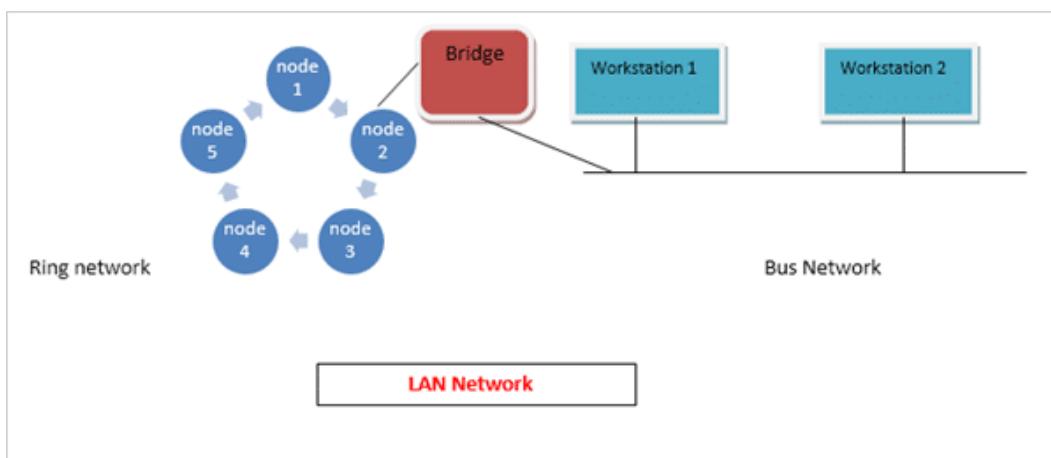


Figura 4.1: Estrutura de uma LAN com ligação de segmentos distintos, com uso de um Bridge Software Testing Help 2020.

4.1.1 LAN sem fio (WLAN)

Este tipo de rede de computadores é uma contrapartida sem fio da rede local. Ele normalmente usa a tecnologia WiFi definida de acordo com os padrões IEEE 802.11.

As redes WiFi podem alcançar até algumas centenas de metros. No entanto, assim como a LAN com fio, seu alcance pode ser aumentado usando repetidores e pontes sem fio.

4.2 MAN

Uma MAN, apresentada na Figura 4.2, cobre uma área geográfica maior do que a rede LAN, por exemplo, cidades e distritos. Também pode ser considerada uma versão superior da rede LAN. Como a LAN cobre apenas uma pequena área da rede, a MAN foi projetada para conectar uma cidade ou duas aldeias através dela.

A área coberta pela MAN é geralmente de 50-60 km. Cabo de fibra óptica e cabos de par trançado são usados para conectividade para comunicação por meio de redes MAN.

MAN também pode ser considerado como um grupo de uma ou mais redes LAN conectadas por meio de um único cabo. RS-232, X-25, Frame Relay e ATM são a prática de protocolo comum para comunicação em MAN Software Testing Help 2020.

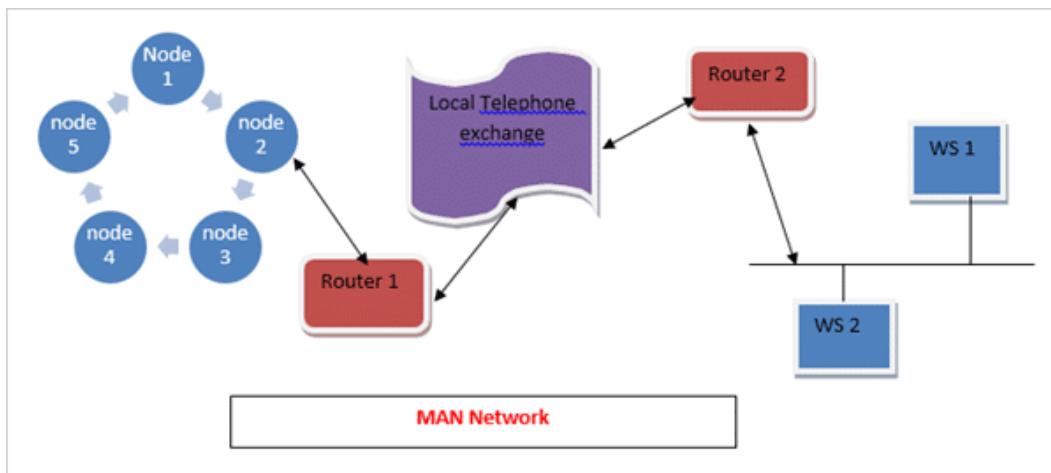


Figura 4.2: Estrutura de uma MAN Software Testing Help 2020.

4.3 WAN

Uma WAN, apresentada na Figura 4.3 abrange grandes áreas, ou seja, de um estado a um país.

Geralmente, o cabo de fibra ótica é usado como meio de transmissão neste sistema. WAN funciona em camadas físicas, de link de dados e de rede do modelo de referência OSI.

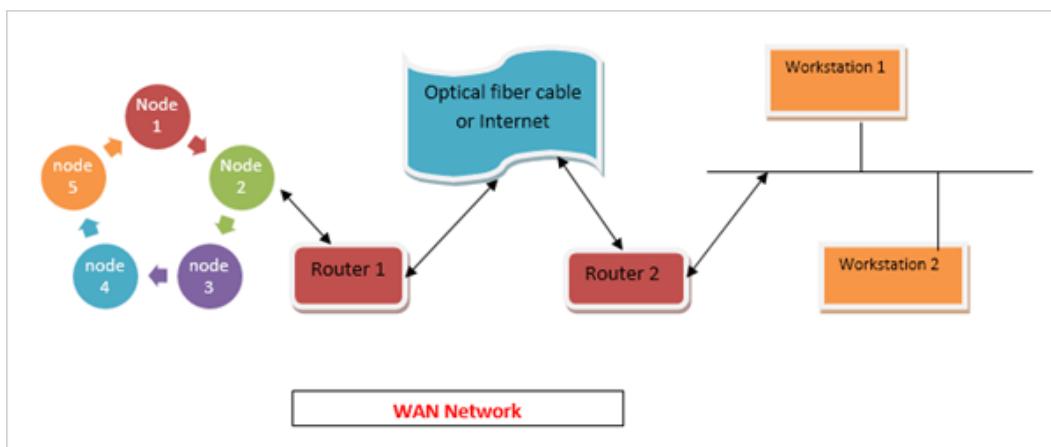


Figura 4.3: Arquitetura de uma WAN Software Testing Help 2020.

A Internet também é um tipo de WAN que se estende por todo o planeta. Várias tecnologias como ADSL, 4G LTE, fibra óptica, por exemplo, são usadas para se conectar à internet. No entanto, essas tecnologias se limitam principalmente a um país no máximo. Cabos são utilizados para ligar vários países e continentes e estabelecer a conectividade global.

4.4 VPN

Rede privada virtual (*Virtual Private Network*), ou VPN, é uma rede criada pela Internet e não tem existência física. Os dispositivos conectados a uma VPN podem ter comunicação contínua usando a estrutura de comunicação da internet por meio de um canal virtual criptografado Fossbytes 2020.

Normalmente, a tecnologia VPN é usada para configurar uma rede privada na Internet para compartilhar os recursos de uma intranet corporativa com usuários remotos e outros escritórios da empresa. As pessoas também podem usar VPN para acessar sua rede doméstica remotamente.

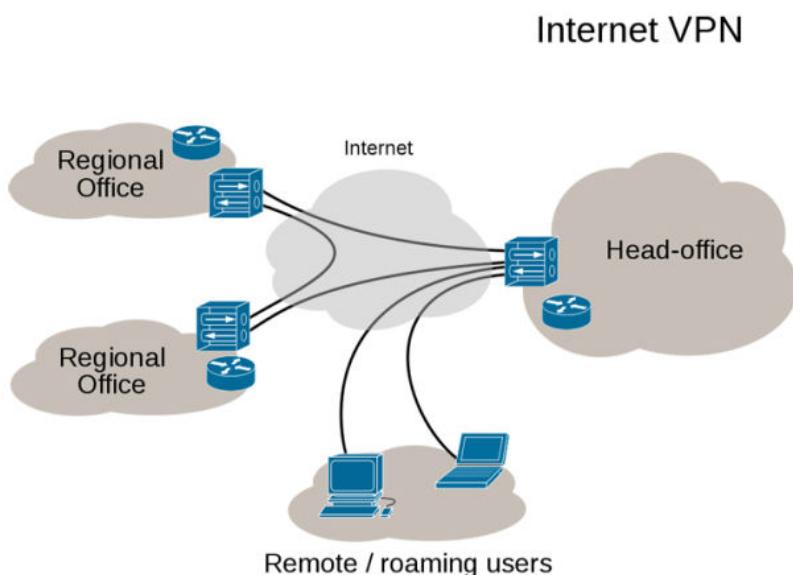


Figura 4.4: Ligação entre escritórios distintos de uma empresa por meio de uma VPN Fossbytes 2020.

Uma conexão VPN é usada para fornecer acesso direto a uma rede corporativa para um usuário que não está na cobertura geográfica da rede. Logicamente, o usuário remoto está conectado como um usuário normal que está dentro do campus de uma organização corporativa.

A VPN também é usada para fornecer um ambiente de rede homogêneo para uma empresa corporativa com escritórios em diferentes partes do mundo. Assim, criando um compartilhamento ininterrupto de recursos, contornando os obstáculos geográficos.

Capítulo 5

Topologia de rede

Uma rede industrial é composta de PLCs, interfaces homem-máquina (IHMs), computadores e dispositivos de I/O ligados entre si por links de comunicação, como cabos elétricos, fibras ópticas, links de rádio e elementos de interface, como placas de rede e gateways. O layout físico de uma rede é o topologia de hardware ou arquitetura de rede.

As topologias geralmente são divididas da seguinte forma:

- Ponto a ponto
- Barramento
- Daisy Chain
- Estrela
- Árvore
- Linha
- Anel

Se sequência essas topologias são apresentadas resumidamente.

5.1 Topologia Ponto a ponto

Na conexão ponto a ponto, apenas dois dispositivos de rede são usados. Esta é a mais simples das redes. Cada dispositivo se conecta diretamente ao outro; portanto, nenhum hub é necessário. Esse sistema é apresentado na Figura 5.1.

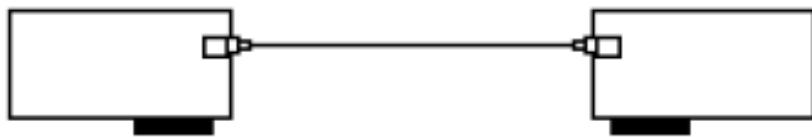


Figura 5.1: Ligação ponto a ponto Contemporary Controls 2020a.

5.2 Topologia Barramento

Este é um dos layouts mais simples; todos os elementos estão ligados juntos ao longo da mesma linha de transmissão. A palavra barramento se refere à linha física. Esta topologia é facilmente implementada e a falha de um nó ou elemento não impede que os outros dispositivos funcionem.

Redes no nível de máquinas e sensores, também conhecidas como barramentos de campo (field buses), usam este sistema.

A topologia do barramento é implementada ligando dispositivos em uma cadeia, conforme apresentado na base da Figura 5.2, ou ligando os dispositivos ao cabo principal através de uma caixa de conexão conforme apresentado no topo da Figura 5.2.

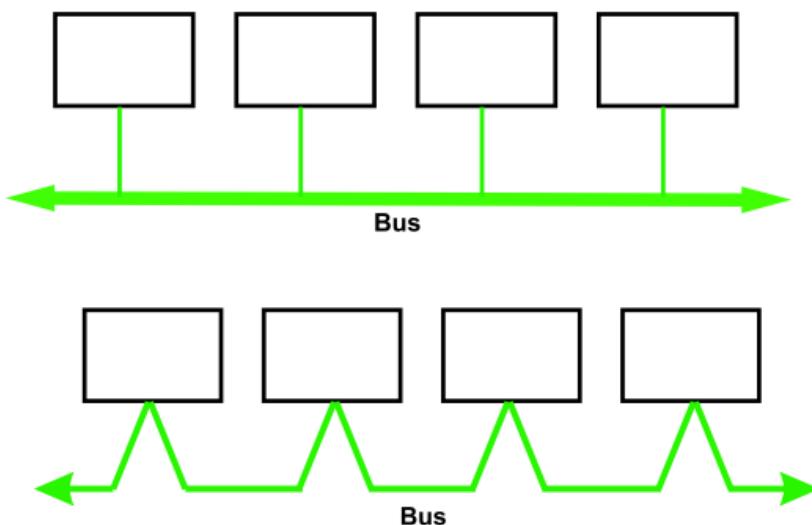


Figura 5.2: Barramento Schneider Electric 2006.

Embora as duas variações da Figura 5.2 tenham o mesmo esquema de ligação elétrica, podem existir nomenclaturas distintas. A Schneider, por exemplo, é uma das empresas que adota o termo “**daisy chain**” para uma configuração que corresponde à topologia barramento

sem derivações (spurs) além de recomendar essa configuração para barramentos RS485, conforme apresentado na Figura 5.3 (a legenda original foi mantida para verificação da recomendação). Pela Figura 5.9 verifica-se que a ligação adotada como daisy chain corresponde à topologia barramento (ligação elétrica) sem o *spur*.

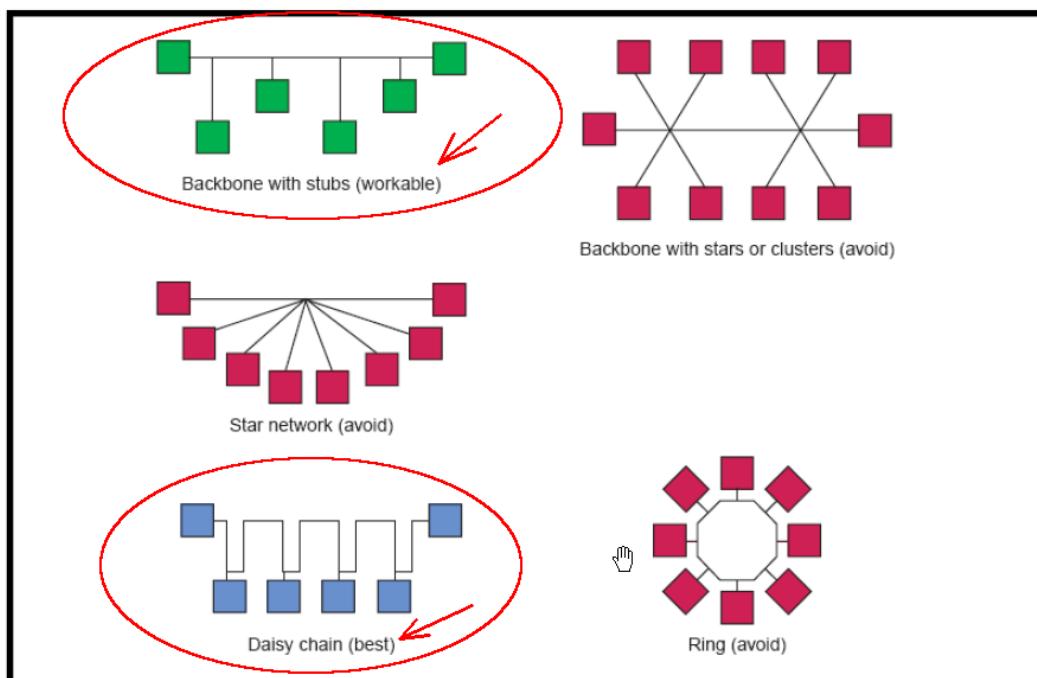


Figure 5—Many common network topologies exist, but the daisy chain is the most reliable for RS-485 networks

Figura 5.3: Recomendação da configuração daisy chain para barramentos RS-485 Schneider Electric 2007.

5.2.1 Topologina Barramento com derivação (com spur)

Nesta seção será considerado barramento com derivação (spur) o esquema apresentado no topo Figura 5.2. Para esse tipo de ligação adotam-se caixas que possibilitam uma, ou mais derivações a partir do barramento.

A Figura 5.4 apresenta o esquemático característico para redes Profibus-PA ou Foundation Fieldbus H1.

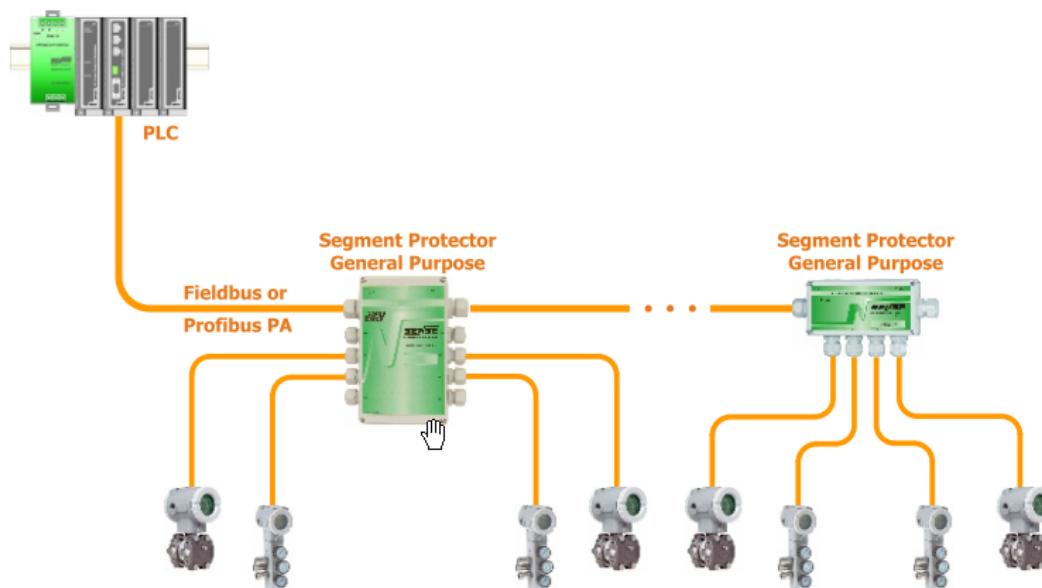


Figura 5.4: Topologia barramento com derivações (spur) Sense - Sensores e Instrumentos 2014.

Os elementos de conexão dessa rede podem apresentar variações de formato, número de saídas para instrumentos existencia de proteção do circuito e nível de proteção do encapsulamento. A Figura 5.5 apresenta um conector “T” e a Página 42 apresenta conectores em caixa com uma e quatro derivações (existem configurações com uma quantidade maior de saídas).

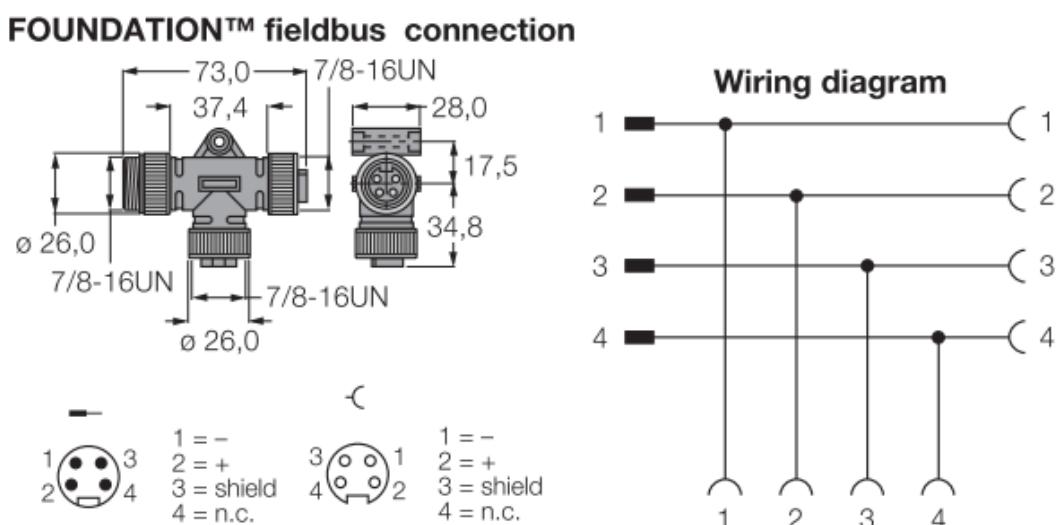


Figura 5.5: Conector “T” com uma derivação para rede em barramento Turck Industrial Automation 2020.

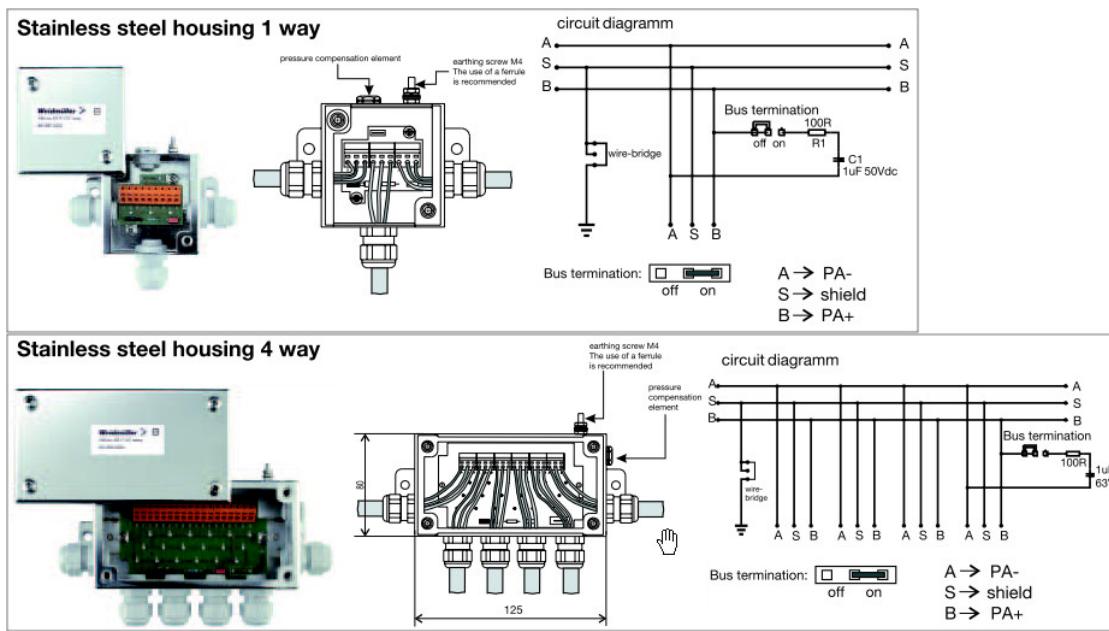


Figura 5.6: Conexões com uma e quatro derivações para redes em barramento Weidmuller 2003.

5.2.2 Topologia Barramento sem derivação (Daisy Chain)

Segundo alguns manuais, o termo ‘‘Daisy Chain’’ em um sentido mais rigoroso corresponde a equipamentos ligados em série, conforme apresenta a Figura 5.7. Nesse sistema cada nó repassa as mensagens até que seja atingido o nó destino.

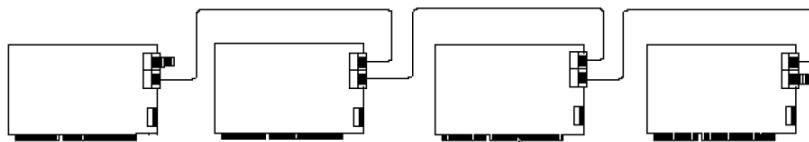


Figura 5.7: Daisy Chain Contemporary Controls 2020a.

Porém aparenta ser usual o uso da nomenclatura ‘‘daisy chain’’ para ligações do tipo barramento onde não existam derivações, como apresentado na Figura 5.8

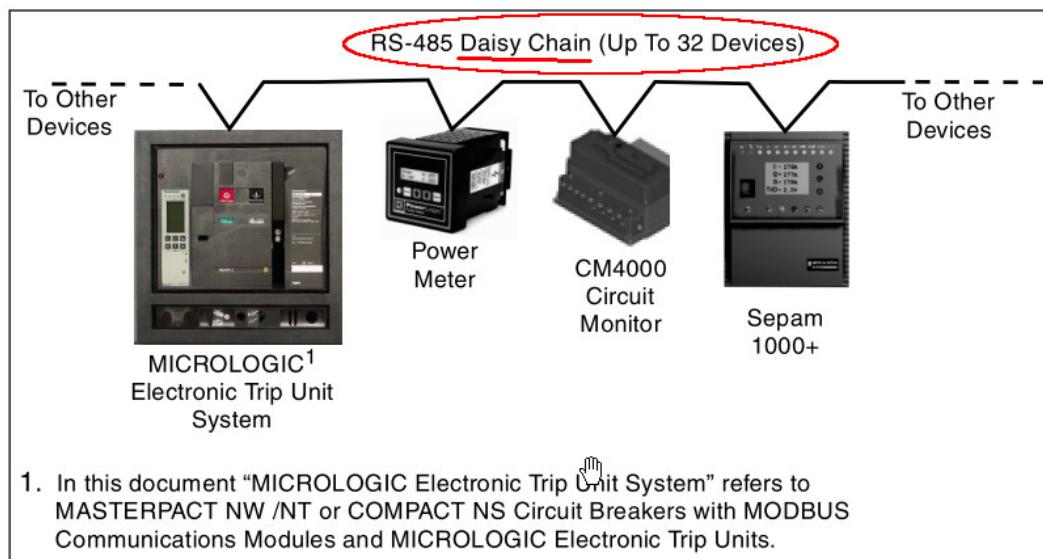


Figura 5.8: Exemplo de arquitetura do tipo barramento sem derivações, definida como daisy chain em alguns manuais Schneider Electric 2000.

A Figura 5.9 apresenta o esquema de ligação elétrica de uma rede daisy chain (apresentando a ligação de três dispositivos) de um manual da Schneider, observa-se que eletricamente é uma rede do tipo barramento.

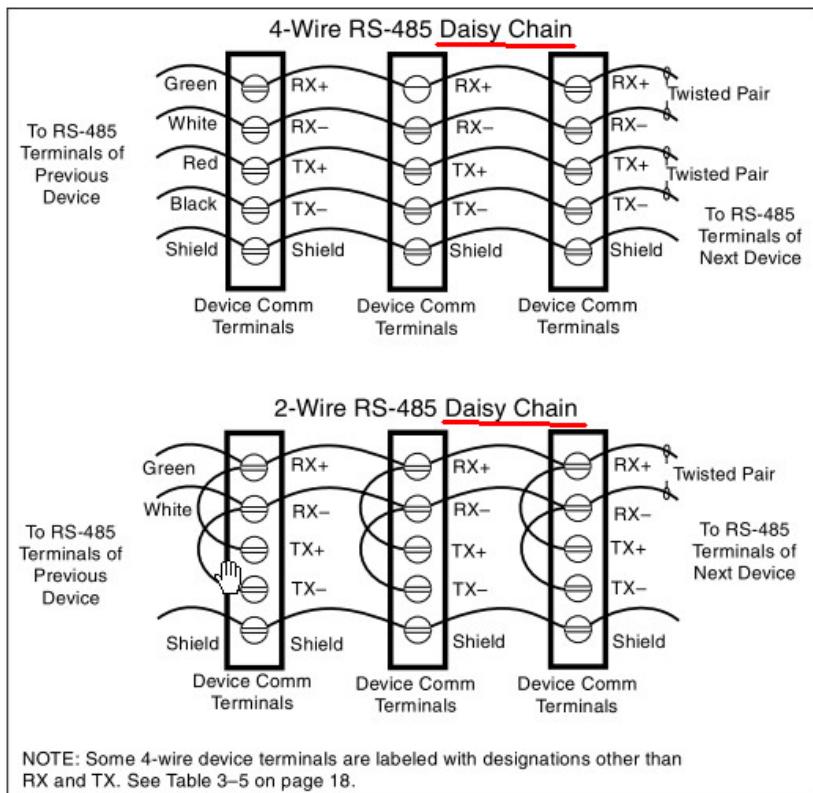


Figura 5.9: Esquema de ligação de uma rede daisy chain (com três dispositivos) obtido de um manual Schneider Electric 2000.

5.3 Estrela

Esta é a topologia usada em redes Ethernet, a mais comum no nível de gerenciamento e de chão de fábrica. A Figura 5.10 apresenta esse modo de ligação. Tem a vantagem de ser muito flexível para funcionar e reparar. As estações finais são conectadas por meio de um dispositivo intermediário (repetidor, switch). A falha de um nó não impede a rede como um todo continuar funcionando, embora o dispositivo intermediário, responsável por ligar os nós, é um ponto fraco.

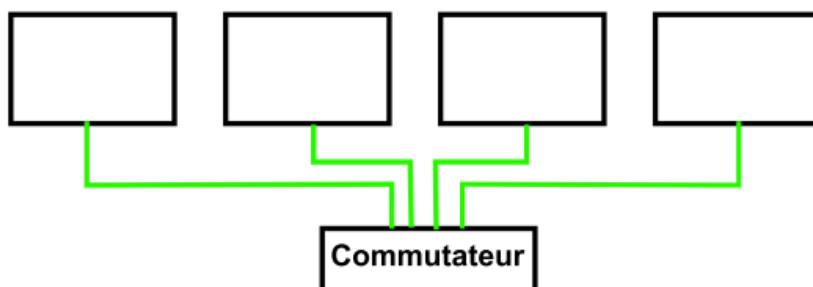


Figura 5.10: Topologia estrela Schneider Electric 2006.

A Figura 5.11 apresenta uma rede com dois switchs formando estruturas de estrela (switches X-400 e X106-1).

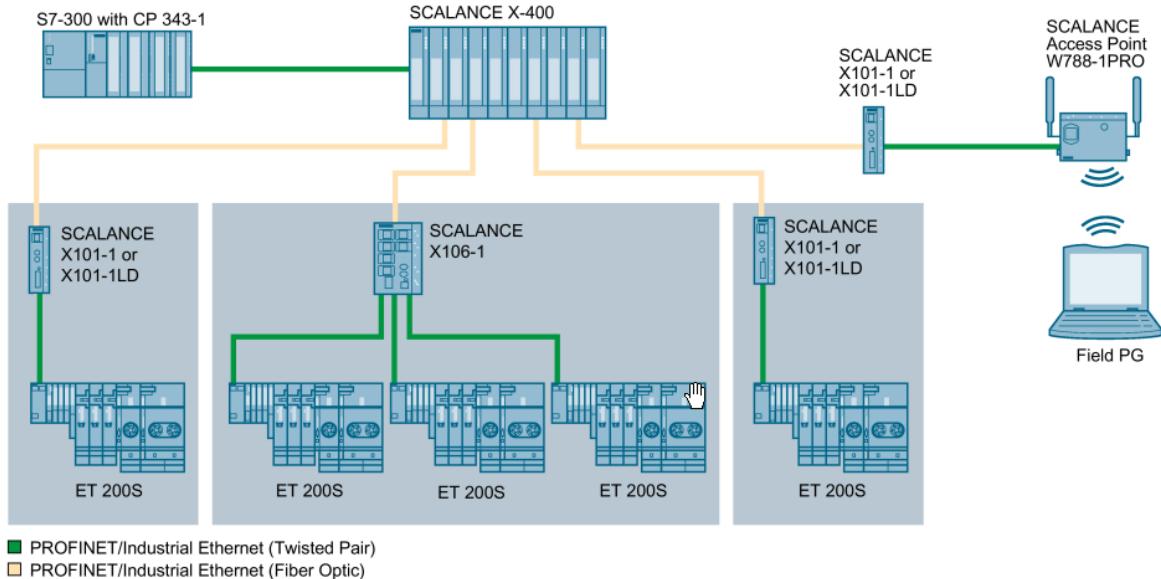


Figura 5.11: Exemplo de rede com topologia estrela com par trançado e fibra ótica Siemens 2016.

5.4 Arvore

Uma topologia em árvore, apresentado na Figura 5.12, é criada combinando várias redes em forma de estrela em uma rede. As partes da planta que formam uma unidade funcional são combinadas em pontos de estrela. Eles são interconectados por meio de switches vizinhos.

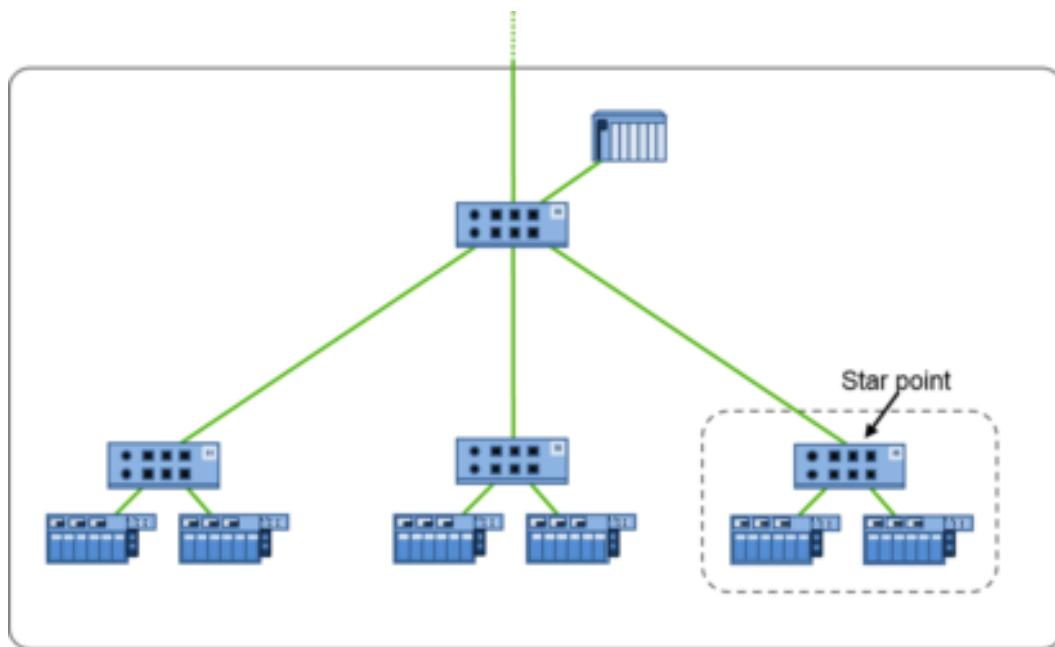


Figura 5.12: Topologia Arvore Helmholtz 2016.

Um switch opera como distribuidor de sinal no ponto estrela. Uma vez que a central encaminha as mensagens com base em um endereço, apenas as mensagens chegarão a um distribuidor vizinho que realmente são necessárias neste distribuidor.

A topologia em árvore é um exemplo típico de uma planta de automação sendo agrupada em diferentes ilhas de manufatura.

5.5 Linha

O arranjo em linha, apresentado na Figura 5.13, é uma topologia bem conhecida para automação. Dispositivos PROFINET equipados com switch integrado facilitam a implementação de topologias de linha.

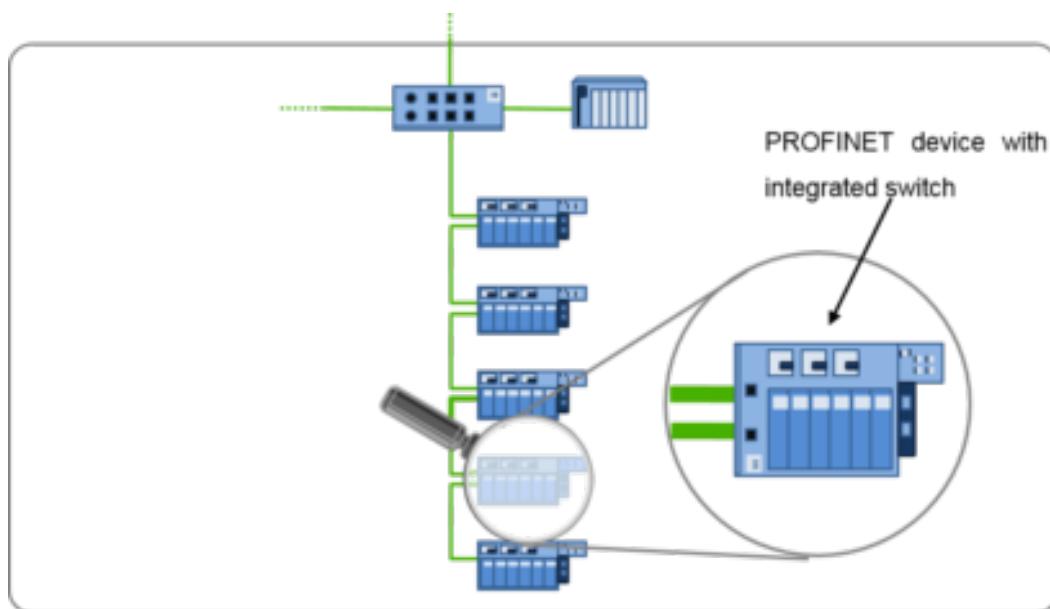


Figura 5.13: Rede em linha Helmholtz 2016.

Com as topologias de linha, em caso de interrupção da linha (por exemplo, falha de um equipamento), os dispositivos localizados após o dispositivo com falha não podem mais ser contatados. Isso pode ser evitado estendendo a linha para uma estrutura em anel usando um protocolo de redundância.

Nota: Visualmente, a topologia em linha tem a mesma configuração da estrutura de barramento (ou da arquitetura daisy chain). Quando se trata apenas de uma derivação de cabeamento, parece comum o termo barramento. Já quando dispositivos como switchs são ligados em série (cada dispositivo com um porta de entrada e uma de saída), entendi que as duas nomenclaturas podem ser encontradas. Em Helmholtz 2016 é usado o termo “topologia de linha” (*line topology*) (Figura 5.14), já em Siemens 2016 é usado o termo “topologia de barramento” (*bus topology*) (Figura 5.15) para arquiteturas que aparentam ser semelhantes, com switchs ligados em série.

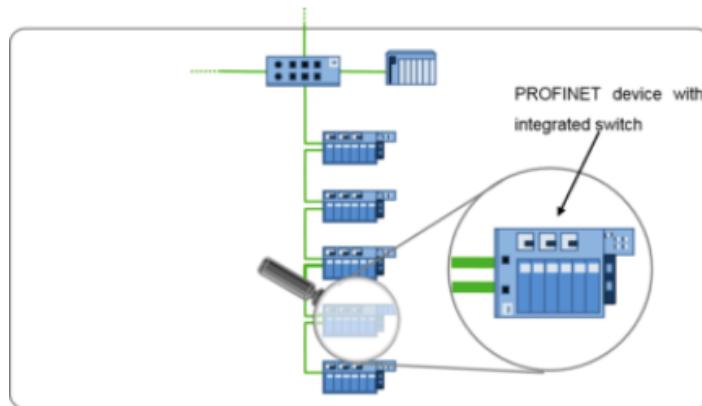


Figure 3-3: Line topology

Figura 5.14: Topologia Linha/Barramento com nomenclatura “Linha” Helmholz 2016.

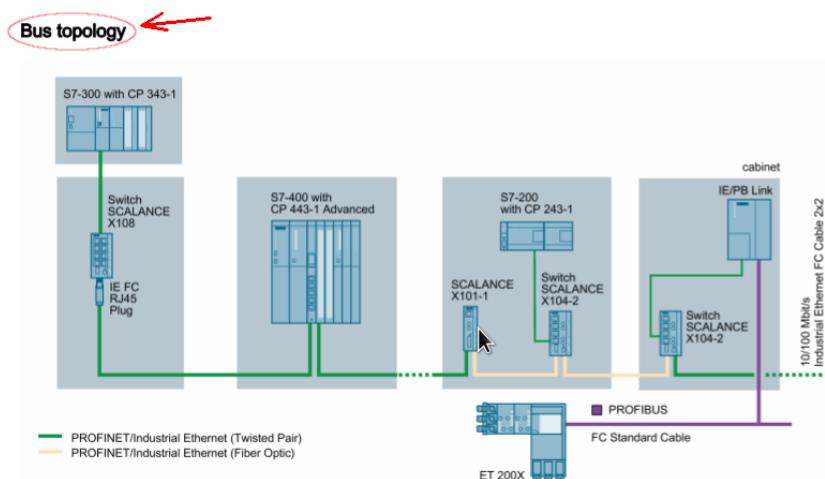


Figura 5.15: Topologia Linha/Barramento com nomenclatura “Barramento” Siemens 2016.

5.6 Anel

Topologias em anel estabelecem redundância de caminho. Se um fio se romper em uma seção do anel, há uma conexão secundária com a rede.

Geralmente, a Ethernet não deve ser conectada em um anel sem executar algum gerenciamento de rede. Sem gerenciar o anel, os pacotes Ethernet podem ficar em círculos para sempre, usando largura de banda. Alguns protocolos Ethernet industrial, como PROFINET, fornecem recursos para gerenciar topologias em anel PI North America 2019.

O diagrama apresentado na Figura 5.16 mostra um exemplo de topologia em anel, observa-se a possibilidade de meios de transmissão distintos entre as seções que compõem o anel.

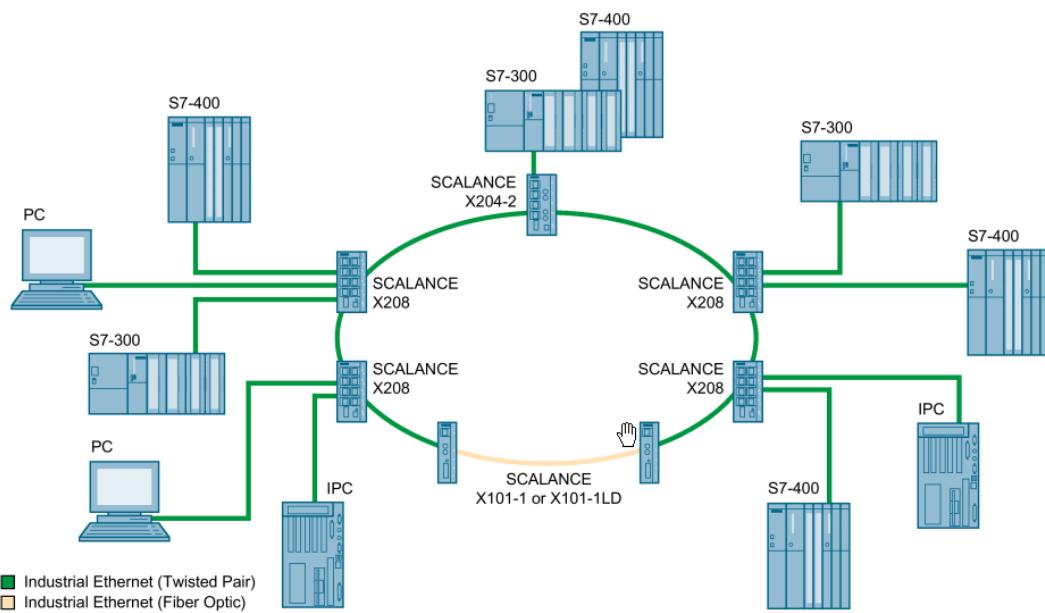


Figura 5.16: Topologia Anel Siemens 2016.

5.7 Combinação de topologias

Uma rede complexa pode conter diferentes topologias combinadas, conforme ilustrado na Figura 5.17

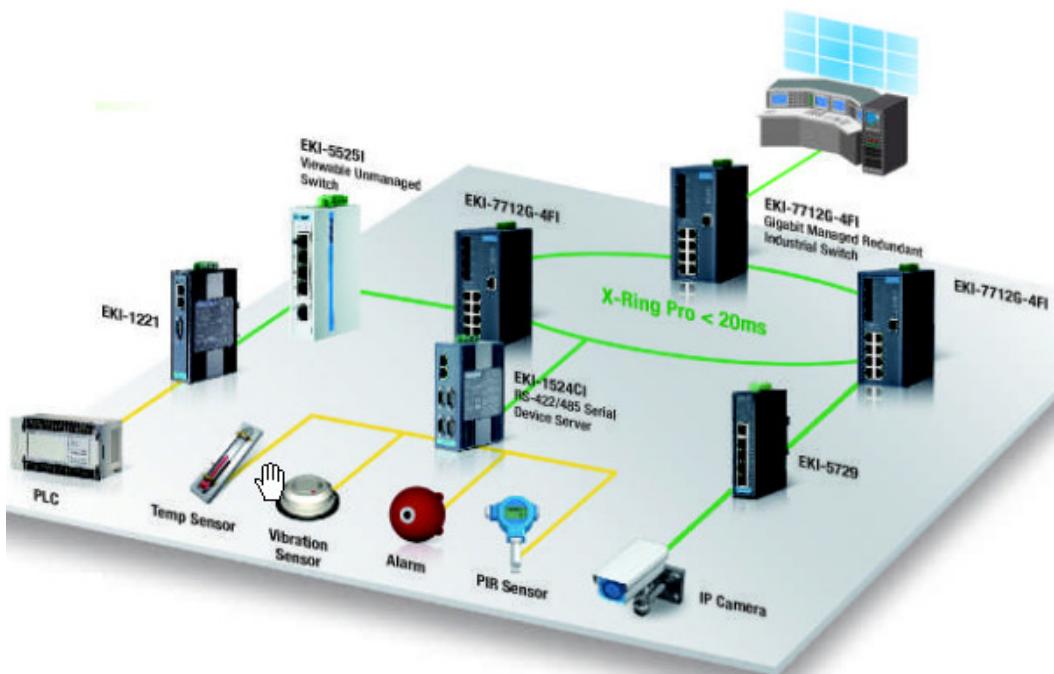


Figura 5.17: Combinação de topologias Advantech 2019.

Capítulo 6

Sistemas de comunicação

Este capítulo apresenta os principais conceitos de comunicação diferenciando comunicação paralela e serial e comunicação síncrona e assíncrona. Além da diferença conceitual serão apresentados os seguintes protocolos:

Comunicação síncrona

- I2C (Inter-Integrated Circuit)
- SPI (Serial Peripheral Interface)

Comunicação assíncrona

- RS-232
- RS-485
- CAN (Controller Area Network)

Comunicação síncrona com clock incorporado aos dados

- Codificação Manchester.
 - Ethernet,
 - IEC 1158-2 (MBP - Manchester bus powered)
 - * FOUNDATION fieldbus H1
 - * PROFIBUS PA

6.1 Comunicação paralela e comunicação serial

Os protocolos de comunicação com fio podem ser paralelos ou seriais. A **comunicação paralela** usa vários fios para transmitir vários bits de dados de uma vez. Os **protocolos seriais**, por outro lado, usam entre um e quatro fios para enviar um único bit por vez em um fluxo contínuo.

Como o padrão serial é o utilizado nas redes industriais, somente esse padrão será visto nesta apostila.

6.2 Comunicação simplex, half-duplex e full-duplex

Os sistemas de transmissão diferem quanto à direção em que os dados fluem e quando as mensagens podem ser transmitidas. Basicamente, existem três formas diferentes de comunicação:

- **simplex**: troca de dados em apenas uma direção,
- **half-duplex**: as estações se revezam para transmitir os dados,
- **full-duplex**: os dados podem ser trocados em ambas as direções simultaneamente.

A Figura 6.1 apresenta esses três esquemas de comunicação.

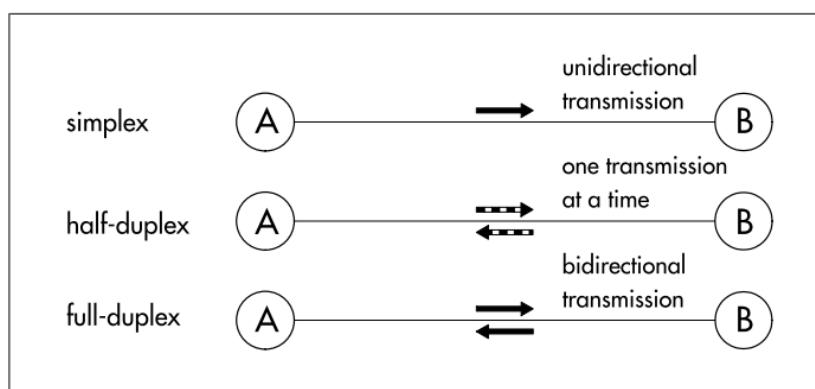


Figura 6.1: Diferentes formas de comunicação, “A” e “B” representam os nós que se comunicam SAMSON 1999.

6.3 Comunicação síncrona e comunicação assíncrona

Essa seção apresenta as características da comunicação síncrona e da comunicação assíncrona.

6.3.1 Comunicação síncrona

Na transmissão síncrona, os sinais nas linhas de dados são válidos sempre que um sinal de relógio, que é usado por ambas as estações, assume um certo estado predefinido (por exemplo, disparo de borda como mostrado na Figura 6.2). O sinal de relógio deve ser transmitido separadamente em uma linha adicional ou pode ser derivado do sinal de dados (codificação Manchester).

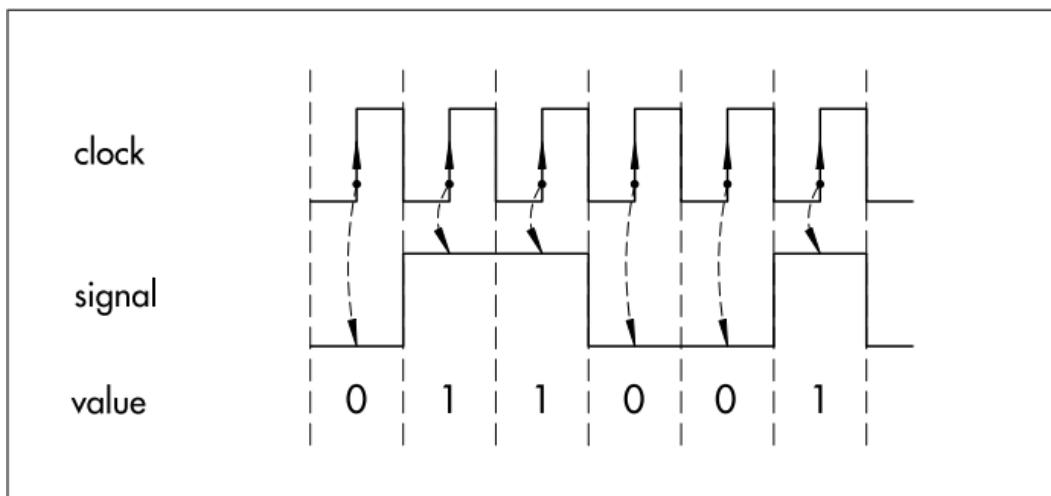


Figura 6.2: Comunicação síncrona com amostragem na borda positiva do relógio SAMSON 1999.

6.3.2 Comunicação assíncrona

Na transmissão assíncrona, nenhum sinal de clock é transmitido. A transmissão se baseia no tempo de duração de cada bit, configurado como taxa de transmissão. Os dispositivos que se comunicam devem ser configurados para usar a mesma taxa de transmissão.

Em uma longa sequência de bits, a comunicação poderia ser prejudicada por pequenas diferenças de relógio entre os dispositivos. Para que essa perda de sincronismo não ocorra, existe uma indicação de início de transmissão para cada conjunto de dados.

Segundo a codificação do padrão UART (*Universal Asynchronous Receiver Transmitter*), sincronização ocorre no início de cada caractere que é marcado com um bit adicional de início e fim, conforme apresentado na Figura 6.3.

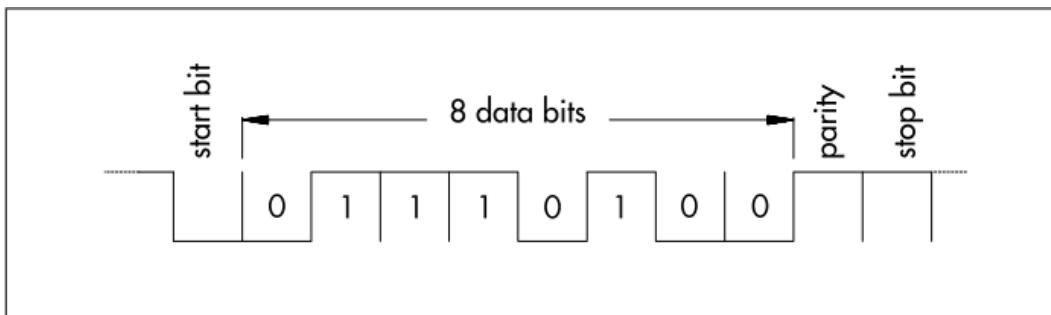


Figura 6.3: Transmissão assíncrona usando padrão de caractere UART (Universal Asynchronous Receiver Transmitter) SAMSON 1999.

Começando com a primeira borda do sinal do bit de início, o receptor sincroniza seu relógio interno com os dados de recepção.

Os bits a seguir são amostrados no meio do tempo de bit.

Após os sete ou oito bits de dados, um bit de paridade é acrescentado para detecção de erro e um ou dois bits de parada para marcar o fim.

A mensagem só é aceita quando o bit de paridade e a polaridade do bit de parada estão de acordo com os padrões de formato.

Uma vez que o receptor sincroniza constantemente, a consistência do tempo da frequência do relógio entre o transmissor e o receptor não precisa ser alta.

6.4 Níveis lógicos

Para a comunicação é necessária a definição dos níveis de tensão correspondentes a cada um dos bits (0 e 1).

Esta seção apresenta alguns níveis para que possam ser comparados com padrões como RS-232, RS-485 e CAN.

Como os sinais elétricos podem sofrer atenuação e ruído ao longo da linha de transmissão, os padrões costumam definir o nível de tensão do “0” e do “1” para o dispositivo que envia (onde ainda não tem atenuação nem ruído) e os níveis para o dispositivo que recebe

(onde o sinal pode chegar atenuado e ruidoso). Para essa forma de especificação temos a definição dos seguintes níveis de tensão:

- V_{OH} - Nível mínimo de tensão de SAÍDA (*output*) que um dispositivo fornecerá para um sinal ALTO (*high*) (1).
- V_{IH} - Nível mínimo da tensão de ENTRADA (*input*) para ser considerado ALTO (1).
- V_{OL} - Nível máximo de tensão de SAÍDA que um dispositivo fornecerá para um sinal BAIXO (*low*) (0).
- V_{IL} - Nível máximo da tensão de ENTRADA para ainda ser considerado BAIXO (0).

6.4.1 Níveis TTL

A maioria dos sistemas que usamos depende de níveis de 3,3 V ou 5 V TTL. TTL é um acrônimo para Transistor-Transistor Logic. Ele se baseia em circuitos construídos a partir de transistores bipolares para realizar a comutação e manter os estados lógicos Sparkfun 2020. Os limites para o padrão TTL são apresentados na Figura 6.4.

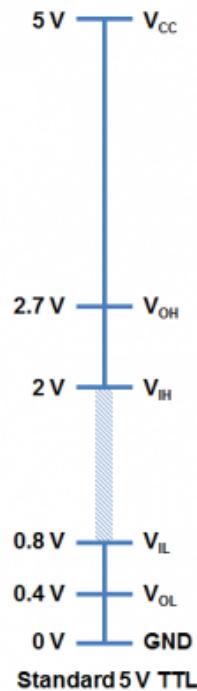


Figura 6.4: Níveis de tensão para definição dos bits no padrão TTL Sparkfun 2020.

A Figura 6.5 mostra a mesma informação isolando os níveis para a escrita e para a leitura do canal de comunicação TTL. Observa-se na figura que as faixas de tensão do bit 0 e do bit

1 são mais largas para a leitura (lado esquerdo da figura) que para a escrita (lado direito da figura), isso se deve ao fato do sinal recebido poder ter sido atenuado e apresentar ruídos.

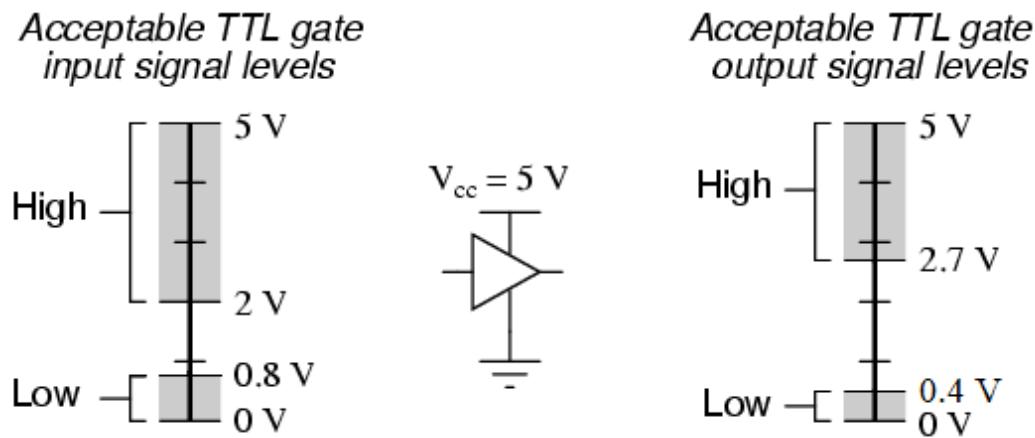


Figura 6.5: Níveis de tensão dos LOW (0) e HIGH (1) para leitura (esquerda) e escrita (direita) em um canal TTL Sparkfun 2020.

6.4.2 Nível CMOS 3.3V

Conforme a tecnologia avançou, surgiram dispositivos que exigem menor consumo de energia e funcionam com uma tensão de base mais baixa ($V_{cc} = 3,3 \text{ V}$ em vez de 5 V). A técnica de fabricação também é um pouco diferente para dispositivos de $3,3 \text{ V}$, o que permite uma pegada menor e menores custos gerais do sistema Sparkfun 2020. A Figura 6.6 apresenta os níveis de tensão para o padrão CMOS

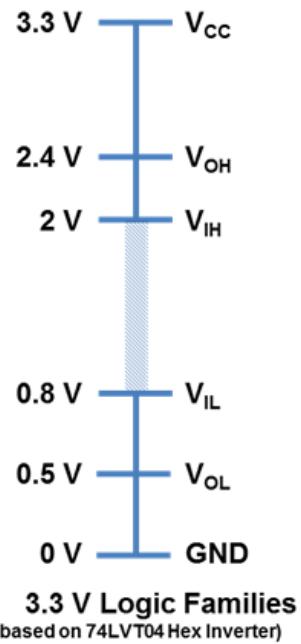


Figura 6.6: Níveis de tensão para o padrão CMOS 3V Sparkfun 2020.

6.4.3 Níveis lógicos do Arduino

Pelo datasheet do ATMega328 (o microcontrolador principal por trás do Arduino Uno e do Sparkfun RedBoard), é possível notar que os níveis de tensão são ligeiramente diferentes do padrão TTL, conforme apresentado na Figura 6.7.

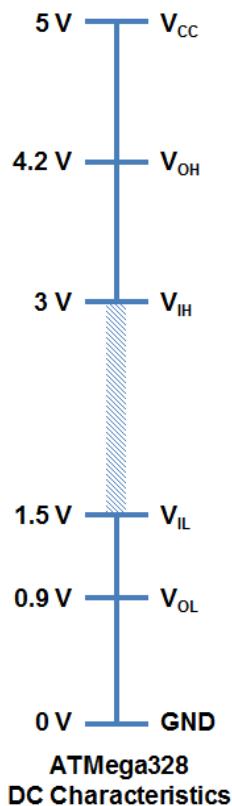


Figura 6.7: Níveis de tensão para comunicação do Arduino (ATMega328)Sparkfun 2020.

O Arduino é construído em uma plataforma um pouco mais robusta. A diferença mais notável é que a região inválida de tensões está apenas entre 1,5 V e 3,0 V. A margem de ruído é maior no Arduino e tem um limite mais alto para um sinal BAIXO (no TTL o nível baixo vai somente até 0,8V, conforme mostra a Figura 6.4). Isso torna a construção de interfaces e o trabalho com outro hardware muito mais simplesSparkfun 2020.

6.5 Universal Asynchronous Receiver-Transmitter (UART)

Devido à grande quantidade de conteúdo sobre esse assunto, a comunicação assíncrona no padrão UART é apresentada no Capítulo 18.

6.6 Rede I2C (Inter-Integrated Circuit)

A Philips Semiconductors (agora NXP Semiconductors) desenvolveu um barramento bidirecional simples de 2 fios para um controle inter-IC (entre circuitos integrados) eficiente. Esse barramento é chamado de barramento Inter-IC ou I²C. A especificação original foi lançada em 1982 e a versão 1.0 da especificação foi lançada em 1992NXP Semiconductors 2014.

Algumas características do barramento I2C são NXP Semiconductors 2014:

- São necessárias apenas duas linhas de barramento; uma linha de dados (**SDA**) e uma linha de relógio (**SCL**).
- Cada dispositivo conectado ao barramento é endereçável por software por um endereço único. O sistema opera no modo mestre/escravo; os mestres podem operar como transmissores ou como receptores.
- É um verdadeiro barramento mult mestre, incluindo detecção de colisão e arbitragem para evitar a corrupção de dados se dois ou mais mestres iniciarem simultaneamente a transferência de dados.
- Transferências de dados serial a 8 bits. As transmissões bidirecionais podem ser feitas em até 100 kbit/s no modo padrão (Standard-mode), até 400 kbit/s no modo rápido (Fast-mode), até 1 Mbit/s no modo Fast-mode Plus (Fm+), ou até 3,4 Mbit/s no modo High-speed mode.
- Transferência de dados unidirecional, serial a 8 bits, de até 5 Mbit/s no modo Ultra Fast-mode.
- A filtragem on-chip rejeita picos na linha de dados do barramento para preservar a integridade dos dados.
- O número de ICs que podem ser conectados ao mesmo barramento é limitado apenas pela capacidade máxima do barramento.

A Figura 6.8 apresenta um exemplo de aplicação do barramento I2C.

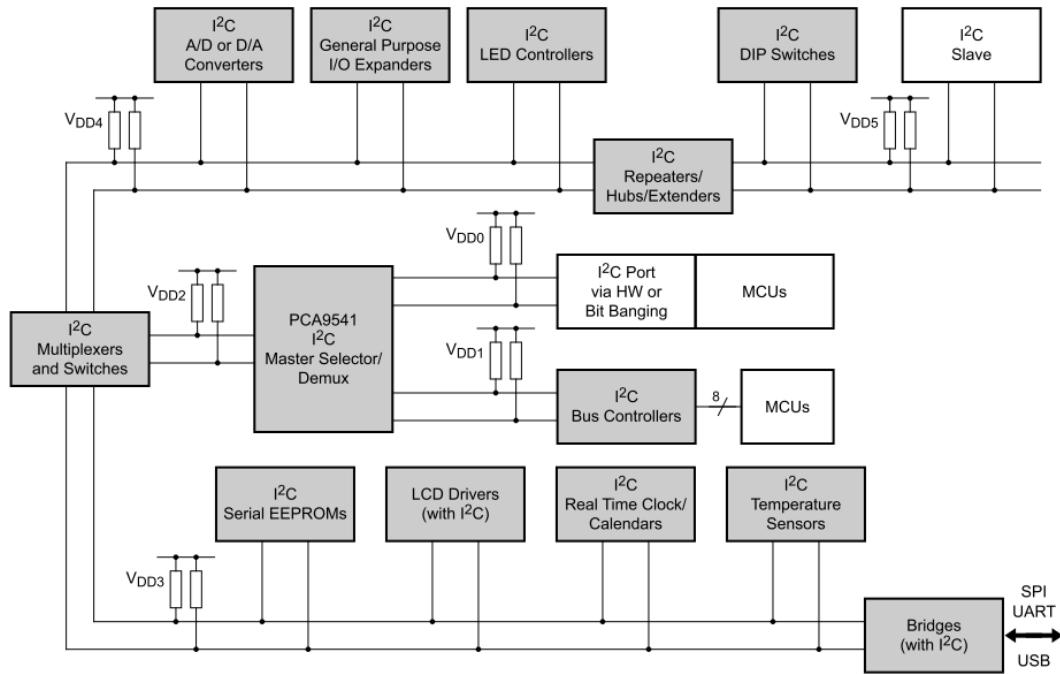


Figura 6.8: Exemplo de aplicação do barramento I²C NXP Semiconductors 2014.

6.6.1 Resumo do funcionamento do protocolo I²C

Essa seção apresenta um resumo da forma como a transmissão I²C é realizada, a forma de comunicação com mais de uma mestre e outros detalhes podem ser encontrados no manual oficial da NXP NXP Semiconductors 2014.

6.6.1.1 Sinais SDA e SCL

Ambos SDA e SCL são caminhos bidirecionais, conectadas a uma tensão de alimentação positiva por meio de uma fonte de corrente ou resistor pull-up, conforme Figura 6.9. Quando o barramento está livre, ambos caminhos estão em nível alto.

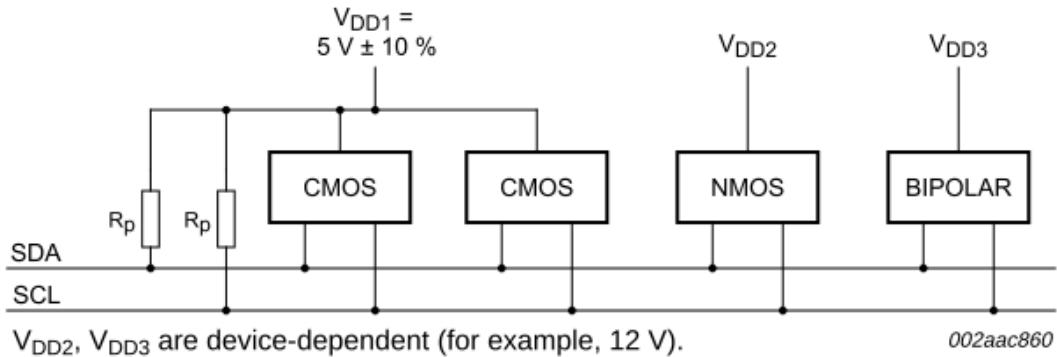


Figura 6.9: Dispositivos com várias fontes de alimentação compartilhando o mesmo barramento NXP Semiconductors 2014.

6.6.1.2 Níveis lógico de SDA e SCL

Devido à variedade de dispositivos de tecnologia diferentes (CMOS, NMOS, bipolar) que podem ser conectados ao barramento I₂C, os níveis lógicos ‘0’ (BAIXO) e ‘1’ (ALTO) não são fixos e dependem do nível associado de VDD. Os níveis de referência de entrada são definidos como 30% e 70% de VDD, conforme:

- VIL (nível baixo) é 0,3VDD,
- VIH (nível alto) é 0,7VDD.

6.6.1.3 Validade dos dados

Durante a transmissão, os dados na linha SDA devem ser estáveis durante o nível ALTO do relógio. O estado ALTO ou BAIXO da linha de dados só pode mudar quando o sinal de clock na linha SCL é BAIXO (ver Figura 6.10). Um pulso de clock é gerado para cada bit de dados transferido.

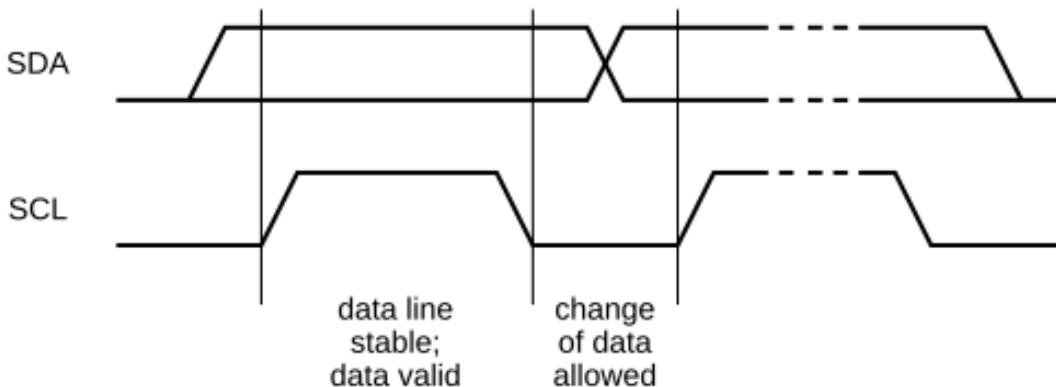


Figura 6.10: Transferência de bits em barramento I₂C NXP Semiconductors 2014.

6.6.1.4 Condições de START e STOP

Durante a transmissão o nível de SDA pode mudar de valor somente com SCL em nível baixo. A transição de SDA com SCL em nível alto serve para marcar o início (START) e o fim (STOP) da transmissão.

Todas as transações começam com START (S) e são encerradas por STOP (P) (ver a Figura 6.11).

Uma transição de ALTO para BAIXO na linha SDA enquanto SCL é ALTO define uma condição de START.

Uma transição de BAIXO para ALTO na linha SDA enquanto SCL é ALTO define uma condição de STOP.

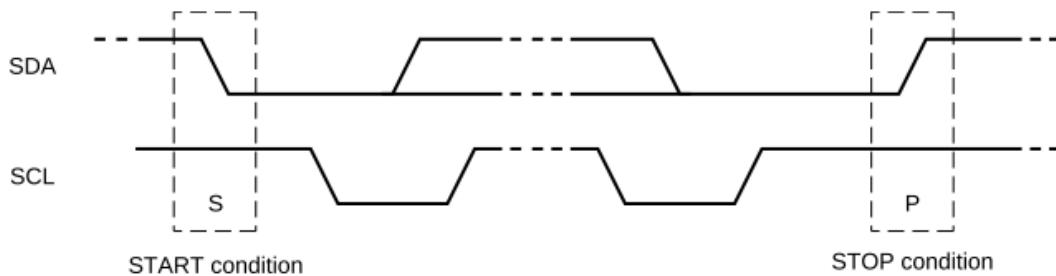


Figura 6.11: Condições de START e STOP NXP Semiconductors 2014.

As condições de START e STOP são sempre geradas pelo mestre. O barramento é considerado ocupado após a condição de PARTIDA. O barramento é considerado livre novamente um certo tempo após a condição STOP.

O barramento permanece ocupado se um START repetido (Sr) for gerado em vez de uma condição STOP. As condições de START (S) e START (Sr) repetido são funcionalmente idênticas.

Para o restante deste documento, portanto, o símbolo S é usado como um termo genérico para representar as condições START e START repetidas, a menos que Sr seja particularmente relevante.

A detecção das condições de START e STOP por dispositivos conectados ao barramento é fácil se eles incorporarem o hardware de interface necessário. No entanto, os microcontroladores sem essa interface têm que amostrar a linha SDA pelo menos duas vezes por período de clock para detectar a transição.

6.6.1.5 Formato do Byte, reconhecimento e não reconhecimento

O que foi apresentado até aqui é um resumo do processo de transmissão. Além dessas operações abordadas, existe a possibilidade do escravo comandar um momento de espera entre a transmissão de bytes (mantendo a linha SCL em nível baixo). Também é necessário o envio de um sinal de reconhecimento pelo escravo, o nono bit da transmissão escrito pelo escravo na linha SDA (bit ACK, visto na Figura 6.12). Como o objetivo desta seção é apresentar apenas uma introdução ao barramento I2C essas funções não serão apresentadas aqui. Esse conteúdo é de fácil entendimento e pode ser encontrado no manual da NXP NXP Semiconductors 2014.

6.6.1.6 Transmissão completa (com endereço do slave e R/W bit)

As transferências de dados seguem o formato mostrado na Figura 6.12.

Após a condição START (S), um endereço de escravo é enviado. Este endereço tem **sete bits** de comprimento seguido por um oitavo bit que é um bit de direção de dados (R/W) - um ‘zero’ indica uma transmissão (WRITE), um ‘um’ indica uma solicitação de dados (READ).

Uma transferência de dados é sempre encerrada por uma condição STOP (P) gerada pelo mestre. No entanto, se um mestre ainda deseja se comunicar no barramento, ele pode gerar uma condição de START repetida (Sr) e endereçar outro escravo sem primeiro gerar uma condição de STOP. Várias combinações de formatos de leitura/escrita são possíveis nessa transferência.

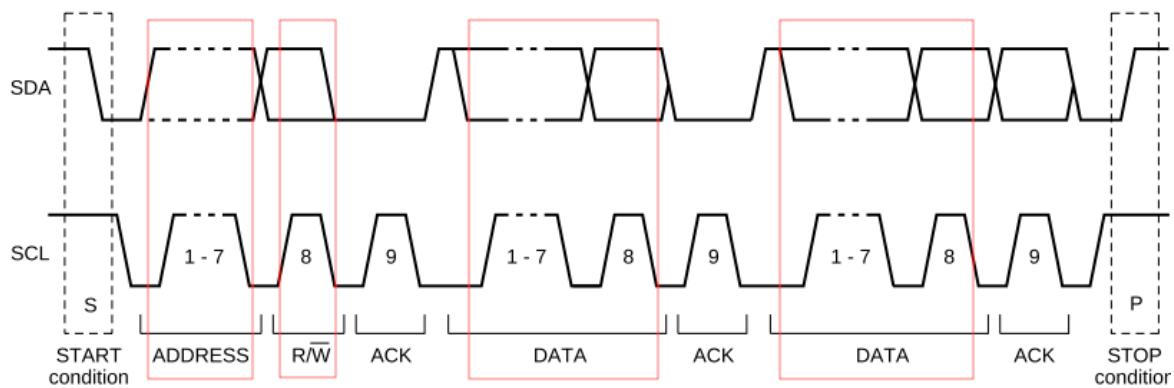


Figura 6.12: Transferência completa (endereço do escravo e dados) NXP Semiconductors 2014.

6.7 Rede SPI (Serial Peripheral Interface)

O SPI é uma porta de entrada/saída serial síncrona de alta velocidade que permite que um fluxo serial de bits com comprimento programado (2 a 16 bits) em uma taxa de transferência programada.

O SPI é normalmente usado para comunicação entre o dispositivo e periféricos externos. As aplicações típicas incluem interface para E/S externa ou expansão periférica por meio de dispositivos como registradores de deslocamento, drivers de vídeo, SPI EPROMS e conversores analógico-digital Texas Instruments 2012.

A Motorola criou a porta SPI em meados dos anos 1980 para usar em suas famílias de microcontroladores. O SPI é usado principalmente para permitir aos microcontroladores se comunicarem com dispositivos periféricos, como E²PROMs.

Os dispositivos SPI se comunicam usando uma relação mestre-escravo. Devido à falta de endereçamento de dispositivo embutido, o SPI requer mais esforço e mais recursos de hardware do que I²C quando mais de um escravo está envolvido. Mas o SPI tende a ser mais simples e eficiente do que I²C em aplicativos ponto a ponto (mestre único, escravo único) pela mesma razão; a falta de endereçamento de dispositivo significa menos a sobrecarga Renesas Electronics Corporation 2010.

6.7.1 Descrição dos sinais e ligações do SPI

O barramento de interface periférica serial possui quatro linhas externas, descritas na Tabela 6.1.

Sinal	Descrição
SCLK	Sinal de relógio serial gerado pelo mestre e transmitido para o escravo.
MOSI/SIMO	(MasterOutput-SlaveInput / SlaveInput-MasterOutput) Sinal de saída do mestre / entrada do escravo. Realiza a transmissão de dados do mestre para o escravo em largura de 8 bits, 12 bits, 16 bits, 20 bits, 24 bits ou 32 bits.
MISO/SOMI	(MasterInput-SlaveOutput / SlaveOutput-MasterInput) Sinal de entrada do mestre / saída do escravo. Realiza a transmissão de dados do escravo para o mestre em largura de 8 bits, 12 bits, 16 bits, 20 bits, 24 bits ou 32 bits.
SS (CS em alguns manuais)	Sinal de seleção de escravo, iniciado pelo mestre para selecionar o dispositivo escravo para comunicação (substitui o endereçamento realizado em padrões como o I ² C).

Tabela 6.1: Descrição dos sinais SPI.

A Figura 6.13 mostra como o dispositivo escravo é conectado ao mestre na implementação de SPI de mestre único, escravo único. A Figura 6.14 mostra implementações SPI de mestre único e vários escravos.



Figura 6.13: Mestre único com escravo único com barramento SPI NXP Semiconductors 2009.

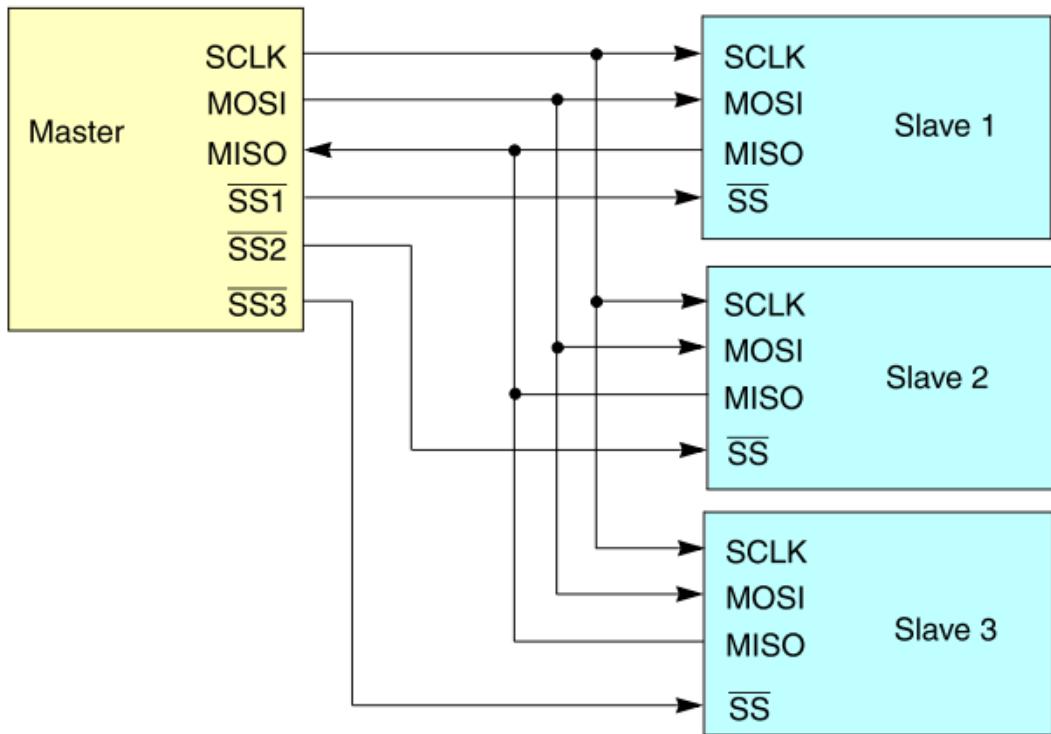


Figura 6.14: Mestre único com vários escravos com barramento SPI NXP Semiconductors 2009.

6.7.2 Transmissão de dados

A Figura 6.15 apresenta outra representação do esquema da Figura 6.13, onde o nome \overline{SS} foi substituído por \overline{CS} (trata-se do mesmo sinal), esse é o nome adotado na figura que apresenta os sinais durante a comunicação.

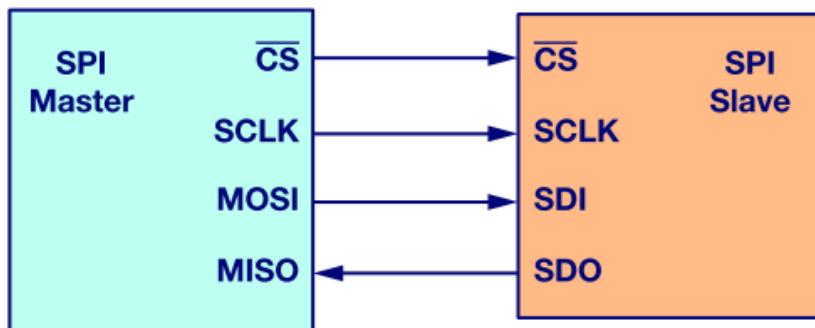


Figura 6.15: SPI, configuração com um mestre e um escravo Dhaker 2018.

Para iniciar a comunicação SPI, o mestre deve enviar o sinal do relógio e selecionar o escravo habilitando o sinal CS. Normalmente, a seleção de chip é um sinal baixo ativo;

portanto, o mestre deve enviar um 0 lógico neste sinal para selecionar o escravo. SPI é uma interface full-duplex; o mestre e o escravo podem enviar dados ao mesmo tempo por meio das linhas MOSI e MISO, respectivamente. Durante a comunicação SPI, os dados são transmitidos simultaneamente (escritos serialmente no barramento MOSI/SDO) e recebidos (os dados no barramento (MISO/SDI) são amostrados ou lidos). A transição do relógio serial sincroniza a escrita e amostragem dos dados.

A interface SPI fornece ao usuário flexibilidade para selecionar a borda ascendente ou descendente do relógio para amostrar e/ou escrever os dados. Nesta apostila será apresentada apenas uma das configurações possíveis, com a amostragem na borda de subida, e a escrita (deslocamento) na borda de descida, ver a Figura 6.16. A leitura do barramento é indicado pelas linhas pontilhadas laranja (borda de subida de CLK) e a escrita é indicada pelas linhas pontilhadas azuis (borda de descida), observe que a transição nas linhas de dados MOSI e MISO ocorre sempre com CLK em nível baixo. As formas de onda das outras configurações podem ser encontradas no seguinte manual da Analog Devices Dhaker 2018.

O número de bits de dados transmitidos usando a interface SPI varia conforme o dispositivo, sendo essa informação encontrada no *datasheet*.

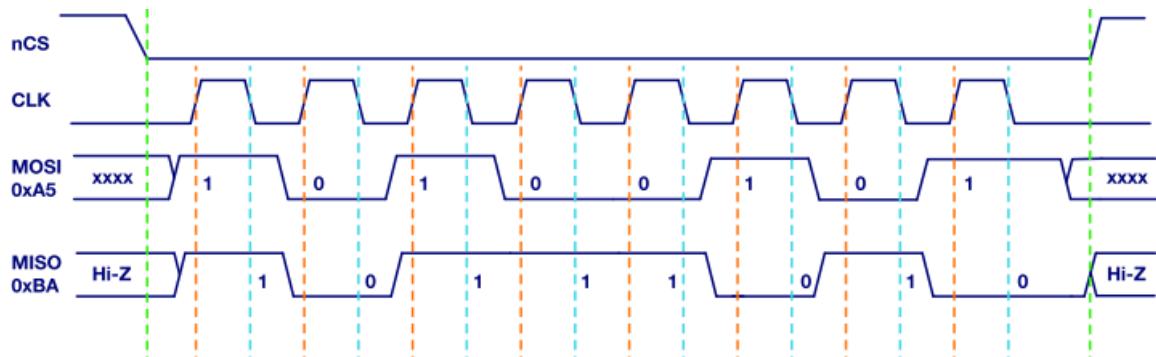


Figura 6.16: Formas de onda na comunicação SPI Dhaker 2018.

6.8 Padrão RS-232

O EIA introduziu o padrão RS-232 em 1962 em um esforço para padronizar a interface entre um terminal de dados (*Data Terminal Equipment* - DTE), e o Equipamento de Comunicação de Dados (*Data Communication Equipment* - DCE). O DTE compreende a fonte de dados, o link de dados ou ambos. O DCE fornece as funções para estabelecer, manter e encerrar uma conexão e para codificar/decodificar os sinais entre o DTE e o canal de dados (por exemplo um modem) Texas Instruments 2002. A Figura 6.17 ilustra o uso do RS-232.

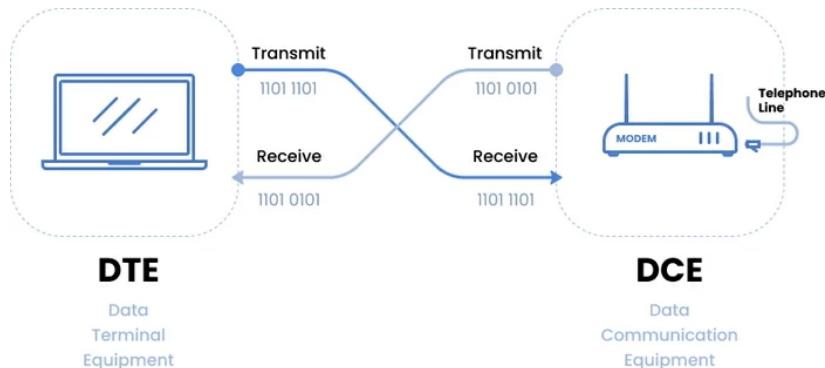


Figura 6.17: Comunicação por barramento RS-232 Weis 2020.

Embora a ênfase tenha sido colocada na interface entre uma unidade de modem e DTE, outras aplicações foram rápidas em adotar o padrão 232. O uso crescente do computador pessoal (PC) rapidamente garantiu que o 232 se tornasse o padrão da indústria para todas as interfaces seriais de baixo custo entre o DTE e o periférico. Mouse, plotter, impressora, scanner, digitalizador, e tracker ball, além dos modems externos e equipamentos de teste, são todos exemplos de periféricos que se conectam a uma porta 232. Usar um padrão comum permite ampla compatibilidade, além de um método confiável para interconectar um PC às funções periféricas Texas Instruments 2002.

Observa-se que o padrão RS-232 permite apenas a comunicação entre dois dispositivos. Cada periférico RS-232 ligado computador, por exemplo, deve ter uma porta RS-232 dedicada.

6.8.1 Conexões e níveis de tensão no RS-232

→ Um 0 lógico é representado por uma tensão acionada entre 5 V e 15 V e uma lógica 1 entre -5V e -15V (algumas fontes consultadas indicam limites em -25V e +25V).

→ Na **extremidade receptora**, uma tensão entre 3 V e 15 V representa um 0 e uma tensão entre -3 V e -15 V representam um 1 (pois o sinal pode enfraquecer até chegar ao receptor).

As tensões entre ± 3 V são indefinidas e estão na região de transição.

A Figura 6.18 apresenta os esquemas de ligação (variando conforme os tipos de conectores) de todos os pinos necessários para comunicação.

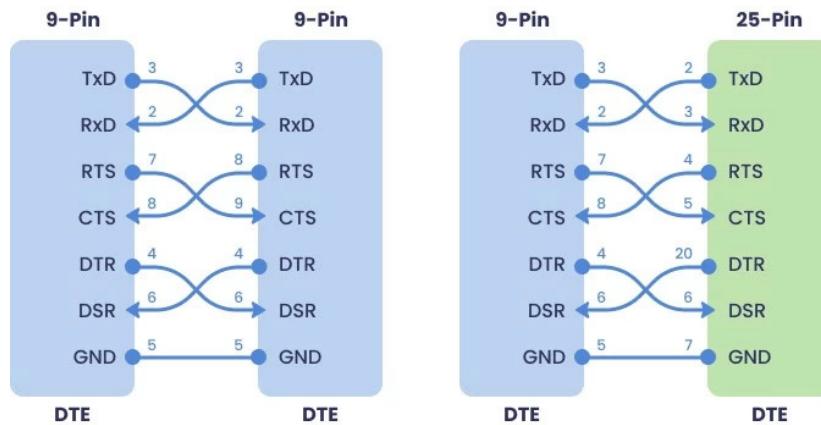


Figura 6.18: Conexões no padrão RS-232 Weis 2020.

A Figura 6.19 apresenta uma simplificação da conexão apresentando apenas um canal de comunicação (TxD - RxD) destacando que trata-se de um **canal de comunicação não balanceado**, ou seja, a tensão no caminho de dados tem como referência o nível GND (no padrão RS-485, por exemplo, o canal é balanceado).

O uso de um canal não balanceado deixa o padrão RS-232 vulnerável a ruídos, o que limita o tamanho do cabo a 15 metros e a taxa de transmissão a 20kbps B+B SmartWorx 2010.

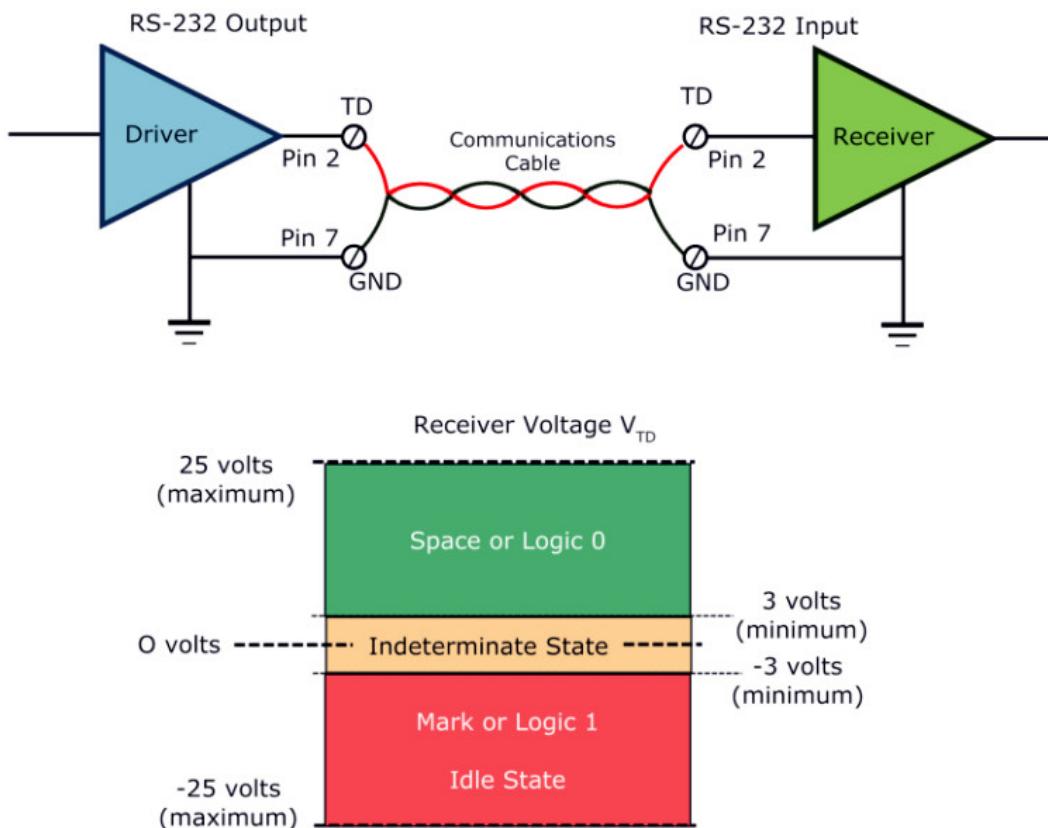


Figura 6.19: Detalha de um caminho de dados RS-232 e níveis de tensão dos níveis lógicos B+B SmartWorx 2010.

6.8.2 Alcance e taxa de transmissão

O comprimento máximo do cabo foi definido originalmente no RS-232-C como 15 metros; no entanto, isso foi revisado em EIA-232-D e TIA/EIA-232-E e agora é especificado mais corretamente como uma carga capacitiva máxima de 2500 pF. Isso equivale a cerca de 15 a 20 metros de comprimento de linha, dependendo da capacidade do cabo Texas Instruments 2002.

A taxa de transmissão limite do padrão RS-232 é 20kbps. Existem estudos sobre o uso de outras taxas.

6.8.3 Protocolo de transmissão (padrão UART)

Embora não seja o interesse desta apostila abordar os detalhes do protocolo RS-232, o protocolo de transmissão é importante, pois esse protocolo será usado com o barramento

RS-485 e parâmetros como o uso do bit de paridade e o número de stop bits devem ser configurados em situações práticas, como na configuração de CLPs, microcontroladores e sistemas supervisórios.

A Figura 6.20 apresenta a estrutura de comunicação para envio de cada caractere.

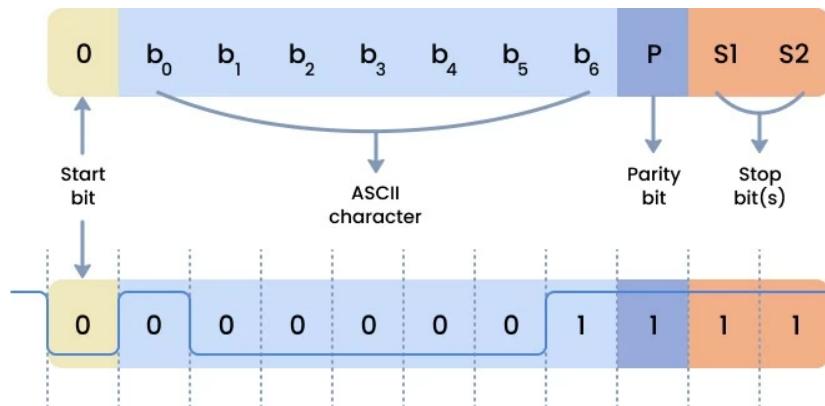


Figura 6.20: Formato de comunicação RS-232 (também usado para o padrão RS-485) Weis 2020.

O Capítulo 18 apresenta o padrão UART de forma mais detalhada, com suas possíveis configurações.

6.8.4 Considerações finais sobre o RS-232

O objetivo desta seção é apresentar as características básicas do RS-232 para comparação com os barramentos RS-485 e CAN, que são os barramentos de interesse para as redes industriais.

Outros detalhes sobre o barramento RS-232, como o sistema de handshaking no RS-232, podem ser encontrados no manual da Eltima Sofrware Weis 2020

6.9 Padrão RS-485

Transmissão de dados entre os componentes de sistemas computacionais e periféricos em longas distâncias e sob condições de alto ruído geralmente se revelam difíceis, senão impossíveis.

Os padrões recomendados que empregam tensão digital balanceada para interface se mostram como uma solução universal para atender aos requisitos de sistema de longa distância.

O 485 é uma interface de linha de transmissão digital balanceada (diferencial) desenvolvida para melhorar o desempenho em comparação com as limitações do padrão TIA/EIA-232 (RS-232) Texas Instruments 2008. As vantagens são:

- Alta taxa de transmissão - até 50M bits/s
- Comprimento de linha mais longo - até 1200 metros
- Transmissão diferencial - menos emissões de ruído
- Múltiplos transmissores e receptores em um barramento

O padrão de linha de transmissão balanceada RS-485 foi desenvolvido em 1983 para fazer a interface de dados entre computador com seus periféricos. O padrão especifica apenas a camada elétrica. Protocolos, tempo, dados seriais ou paralelos e escolha do conector são deixados para serem definidos pelo designer ou por um padrão de nível superior.

Os circuitos de transmissão de dados que empregam RS-485, drivers, receptores ou transceptores são usados em praticamente qualquer aplicação que requer uma interconexão econômica e robusta entre dois ou mais dispositivos de computação.

Uma aplicação típica pode usar sinalização RS-485 entre terminais de ponto de venda e um computador central para débito automático de estoque.

Como resultado de sua versatilidade, um número crescente de comitês de padronização está adotando o padrão RS-485 como a especificação da camada física de seu padrão de comunicação. Os exemplos incluem o ANSI (American National Standards Institute), o Small Computer Systems Interface (SCSI), o padrão Profibus e o barramento de medição DIN (DIN measurement bus)Texas Instruments 2008.

6.9.1 Sinais eléticos no padrão RS-485

Em um sistema de transmissão de dados balanceado, a tensão produzida pelo driver aparece em um par de linhas de sinal. Essas linhas produzem sinais de saída complementares (opostos). Quando um está baixo, o outro está alto e vice-versa. É importante entender que a transmissão de dados balanceada RS-422/RS-485 também **requer uma conexão GND**, embora a conexão GND não seja usada pelo receptor para determinar o estado lógico dos dados. A Figura 6.21 apresenta o esquema de ligação balanceada e os níveis de tensão para definição dos bits 0 e 1.

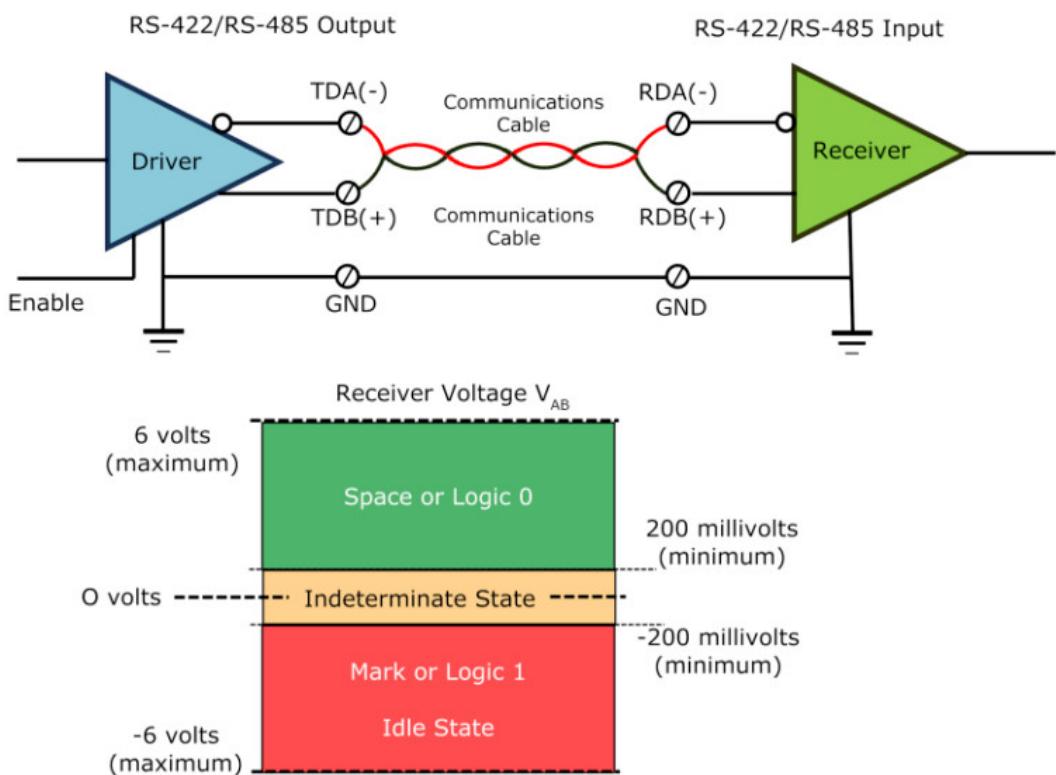


Figura 6.21: Canal de transmissão balanceado RS-422e RS-485 B+B SmartWorx 2010.

Diferente do padrão RS-232, o RS-485 permite até 32 dispositivos em um único barramento. A Figura 6.22 apresenta essa indicação da quantidade de unidades conectadas ao barramento, também apresenta os resistores de terminação nas extremidades do barramento (a linha de GND foi omitida na figura, mas é necessária).

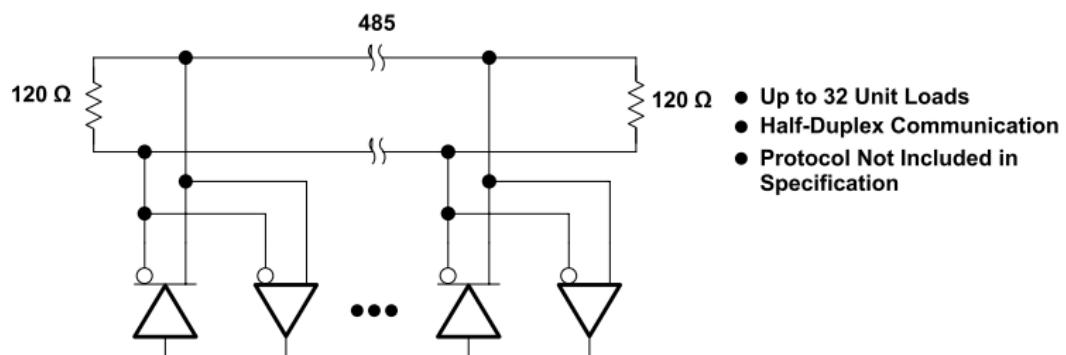


Figura 6.22: Barramento RS-485 com terminação nas extremidades (Até 32 dispositivos)Texas Instruments 2008

6.9.2 Controle por três estados (saída em alta impedância)

Para que vários dispositivos possam compartilhar o mesmo barramento, todos dispositivos que não estejam transmitindo devem estar desconectados da linha, pois em caso contrário forçariam a tensão na linha para o nível alto ou baixo, o que iria conflitar com os bits enviados pelo transmissor ativo. Quando desconectados dispositivos apresentam uma alta impedância entre suas saídas e o barramento, conforme mostra a Figura 6.23.

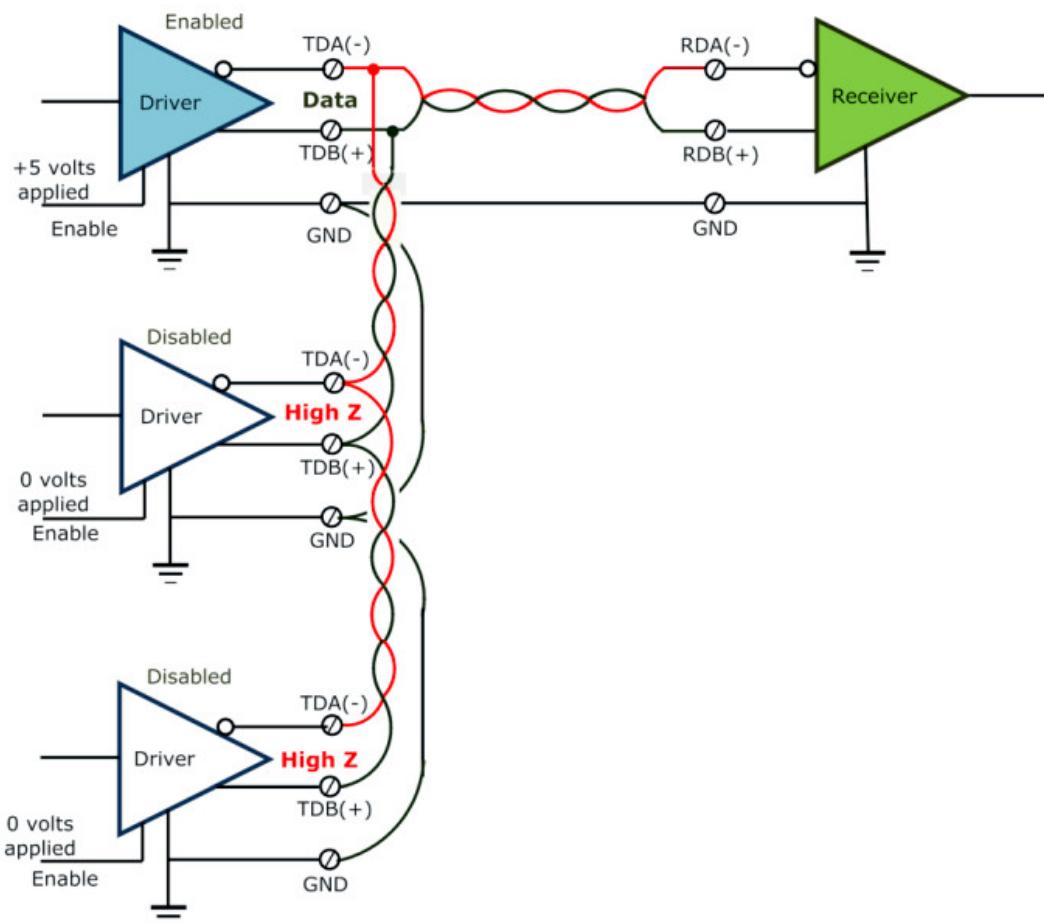


Figura 6.23: Controle por três estados B+B SmartWorx 2010.

6.9.3 Velocidade x comprimento do cabo para barramento RS-485

A distância alcançável é uma função do cabo. Quanto mais comprido for o cabo, maior será a atenuação. Como a atenuação aumenta com a frequência, os cabos também exibem um comportamento de filtro passa-baixa, de forma que a distância alcançável diminui com a taxa de dados. As distâncias recomendadas pelo padrão RS485 são mostradas no gráfico da Figura 6.24. A distância do gráfico está em pés, a conversão aproximada dos valores apresentados é:

$$10 \text{ ft} \approx 3\text{m}, 100 \text{ ft} \approx 30\text{m}, 1\text{k ft} \approx 304\text{m} \text{ e } 10\text{k ft} \approx 3038\text{m}.$$

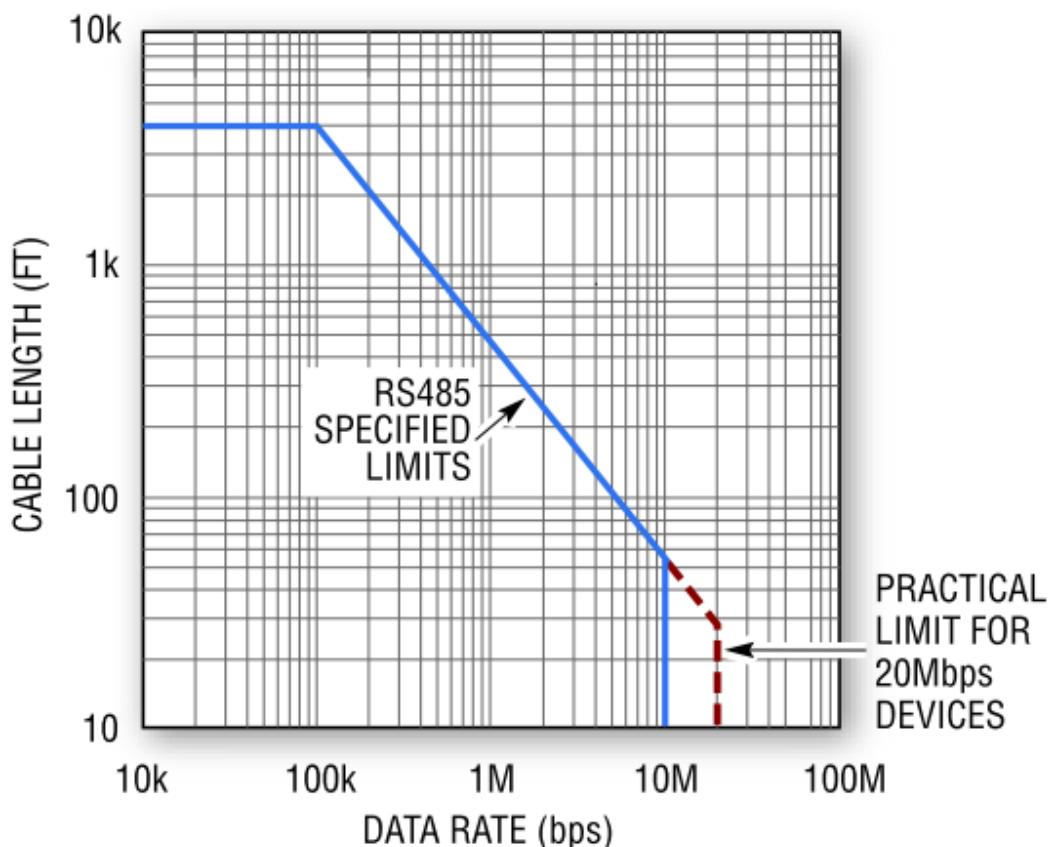


Figura 6.24: Taxa de transmissão para barramento RS-485 Analog Devices 2020.

Muitos cabos são capazes de velocidades e distâncias maiores. Consulte a curva de desempenho típica do fabricante do cabo de 0 a 50% do tempo de subida vs comprimento do cabo Analog Devices 2020.

6.9.4 Ligações a dois e quatro fios (+ GND)

Na **ligação a dois fios**, um único par de fios transmite a informação no dois sentidos de forma alternada, o que caracteriza uma comunicação half-duplex. Esse esquema de ligação é apresentado na Figura 6.25.

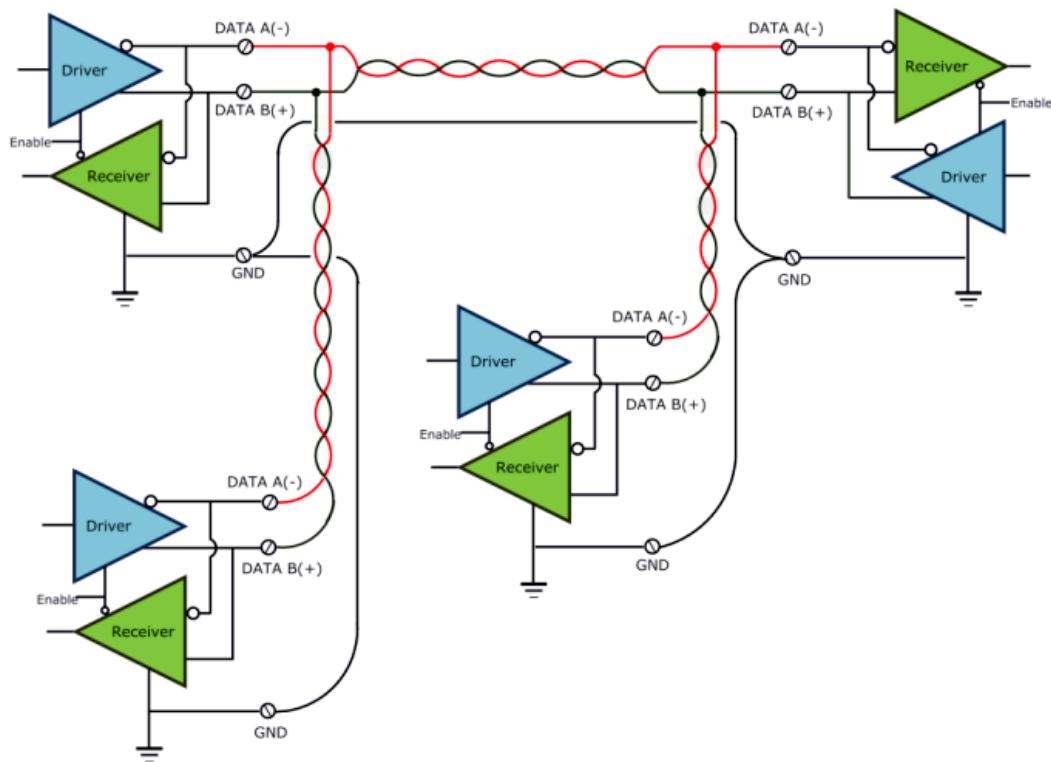


Figura 6.25: RS-485, ligação a dois fios B+B SmartWorx 2010.

Na **ligação a quatro fios**, um par de condutores liga o transmissor do mestre a todos receptores dos escravos e o segundo par de condutores liga o receptor do mestre a todos transmissores dos escravos. Essa ligação possibilita a comunicação full-duplex. Esse esquema de ligação é apresentado na Figura 6.26

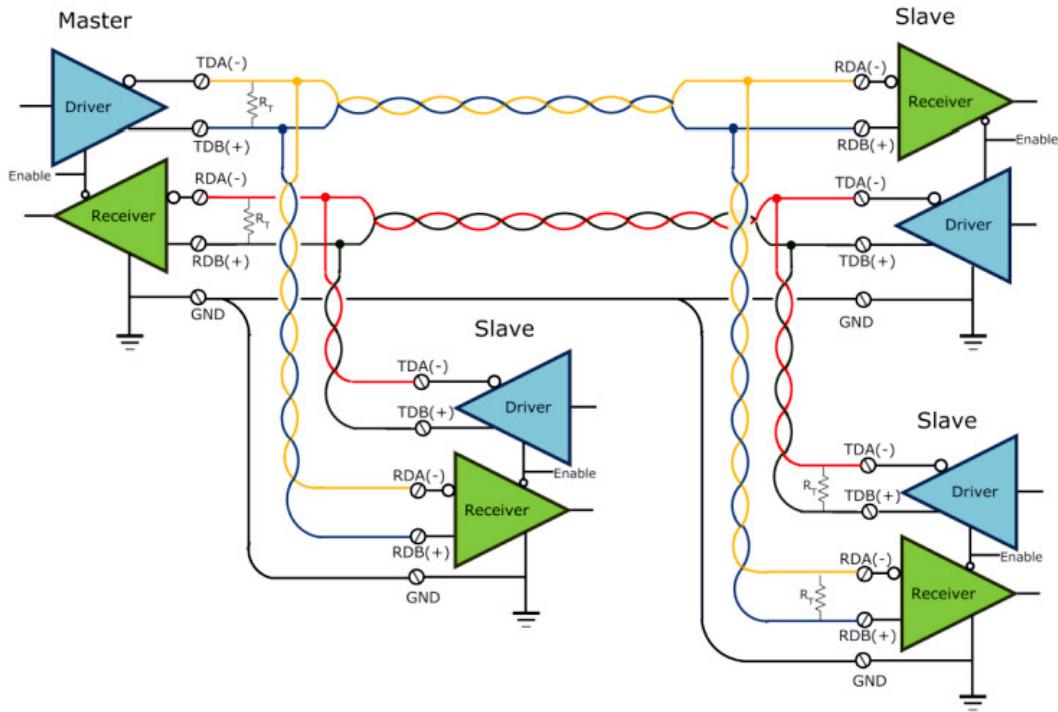


Figura 6.26: RS-485, ligação a quatro fios B+B SmartWorx 2010.

Observa-se que nos dois casos é necessário um condutor a mais para transmitir o GND.

6.9.5 Ligação do GND em um barramento RS-485

Se houver uma diferença de potencial de alta tensão entre os aterramentos remotos, especialmente durante os transientes, uma corrente fluirá pelo aterramento remoto porque existe um loop de aterramento. Se o loop de aterramento não apresentar uma resistência, a corrente de aterramento é grande e cria alguns problemas. Por este motivo, o padrão RS-485 recomenda adicionar alguma resistência entre a lógica e o aterramento do chassi para evitar correntes de loop de aterramento excessivas Texas Instruments 2010. A figura Figura 6.27 apresenta uma ligação com a presença desses resistores de proteção (em destaque).

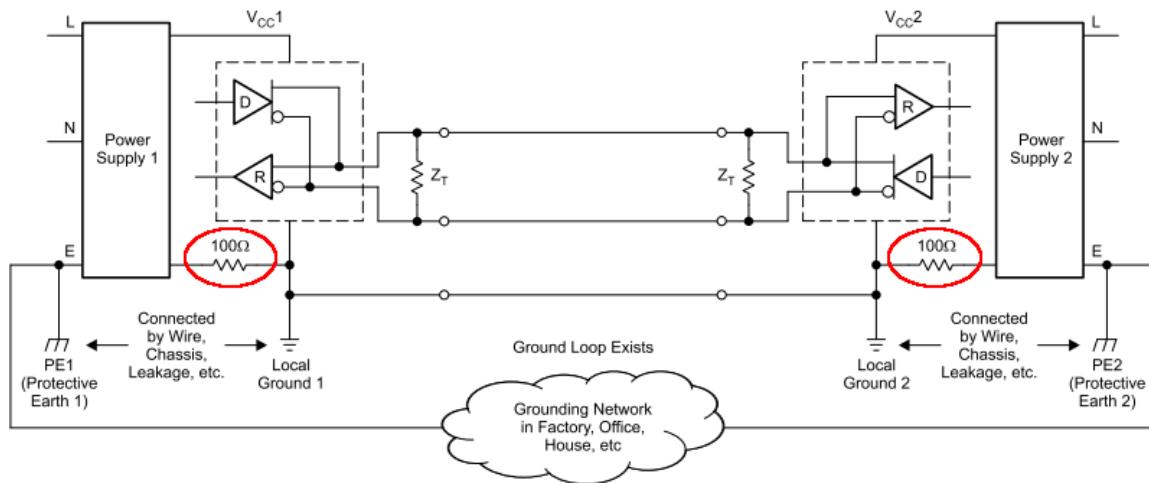


Figura 6.27: Ligação de barramento RS-485 com resistor de proteção do GND Texas Instruments 2010.

6.9.6 RS-422

É comum existir referência ao protocolo RS-422 em manuais que abordam o RS-485. Aparentemente os dois protocolos são bastante semelhantes. Pelo baixo uso do padrão RS-422, esse padrão não será abordado isoladamente. No *Application Report* da Texa Instruments, AN-759, existe uma comparação entre os padrões RS-485 e RS-422. Esse documento lista problemas do protocolo RS-422 que justificam a preferência pelo RS-485 Texas Instruments 2004.

6.10 Rede CAN (*Controller Area Network*)

O barramento CAN foi desenvolvido pela BOSCH como um sistema de transmissão de mensagens multimestre que especifica uma taxa máxima de sinalização de 1 Mbps (megabit por segundo).

O barreimento CAN foi lançado em 1986 e é um protocolo automotivo baseado em mensagem que permite aos microcontroladores se comunicarem sem a necessidade de um computador host. Esses microcontroladores também são chamados de Unidade Eletrônica de Controle (*Electronic Control Units ECU*) ou nós.

O barramento CAN conecta todos os nós por meio de um único par de fios trançados (CAN-H e CAN-L) (ver a Figura 6.28). Todos os sinais enviados de um nó atingem todos os outros nós, onde os identificadores dentro de cada mensagem especificam o(s) receptor(es) Innodisk 2017.

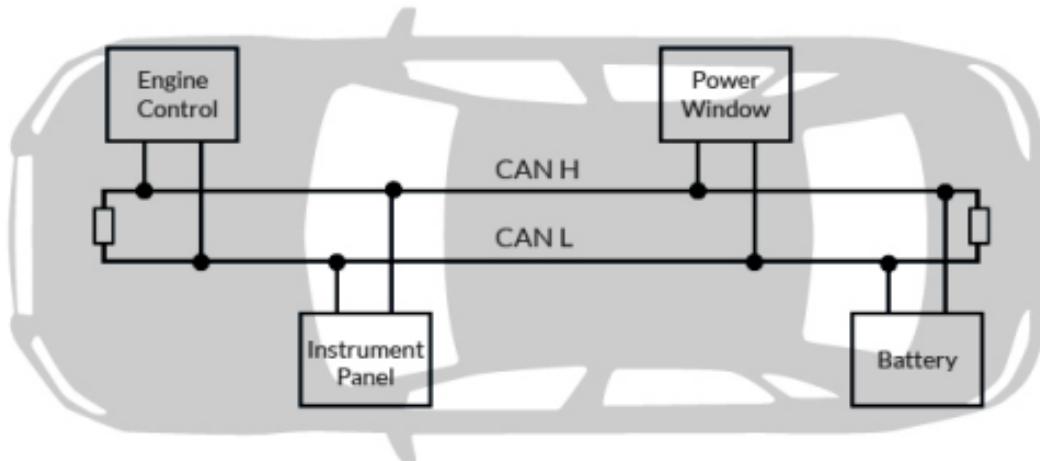


Figura 6.28: Estrutura de um sistema com barramento CAN Innodisk 2017.

Ao contrário de uma rede tradicional como USB ou Ethernet, o CAN não envia grandes blocos de dados ponto a ponto do nó A ao nó B sob a supervisão de um barramento mestre central. Em uma rede CAN, muitas mensagens curtas como temperatura ou rotação do motor (RPM) são transmitidas para toda a rede, o que fornece consistência de dados em todos os nós do sistema Texas Instruments 2016.

O ISO 11898 é o padrão internacional para o padrão de comunicação de alta velocidade para veículos CAN.

6.10.1 Níveis de tensão do barramento

O padrão CAN especifica dois estados lógicos: recessivo e dominante. A norma ISO-11898 define uma tensão diferencial para representar os estados recessivos e dominantes (ou bits), como mostrado na Figura 6.29.

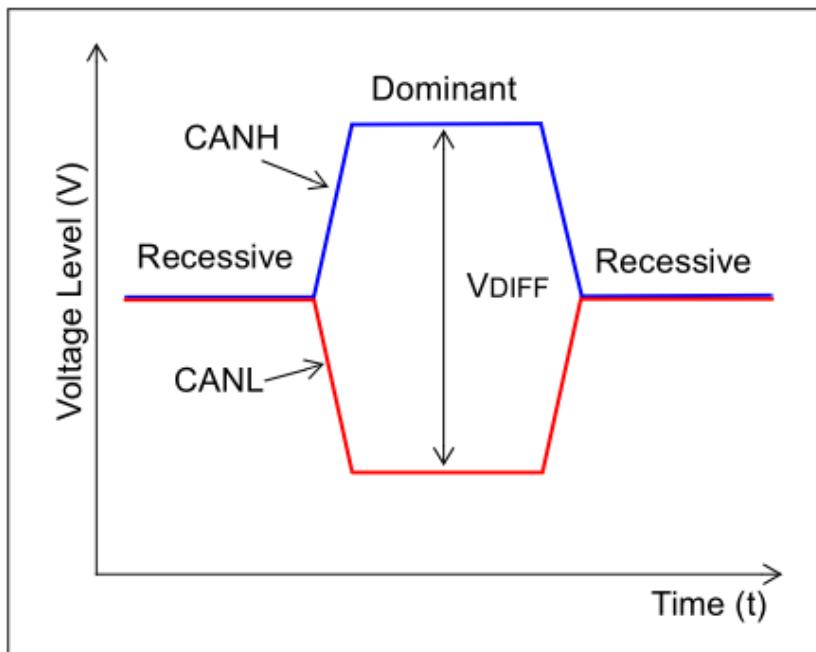


Figura 6.29: Tensão diferencial em um barramento CANMicrochip 2002.

No **estado recessivo (ou seja, lógica '1')**, a tensão diferencial entre CANH e CANL é menor que o limite mínimo (receptor $<0,5$ V ou $<1,5$ V saída do transmissor) Figura 6.30.

No **estado dominante (ou seja, '0')**, a tensão diferencial entre CANH e CANL é maior que o limite mínimo. Um bit dominante se sobrepõe um bit recessivo no barramento de modo a alcançar uma arbitragem não destrutiva bit a bit Microchip 2002.

Na arbitragem não destrutiva, um transmissor de menor prioridade interrompe a transmissão ao identificar que um transmissor de maior prioridade também está usando o barramento (iniciaram a transmissão ao mesmo tempo). Como a colisão é não destrutiva (o bit recessivo não destrói o bit dominante na colisão) o transmissor de maior prioridade segue sua transmissão.

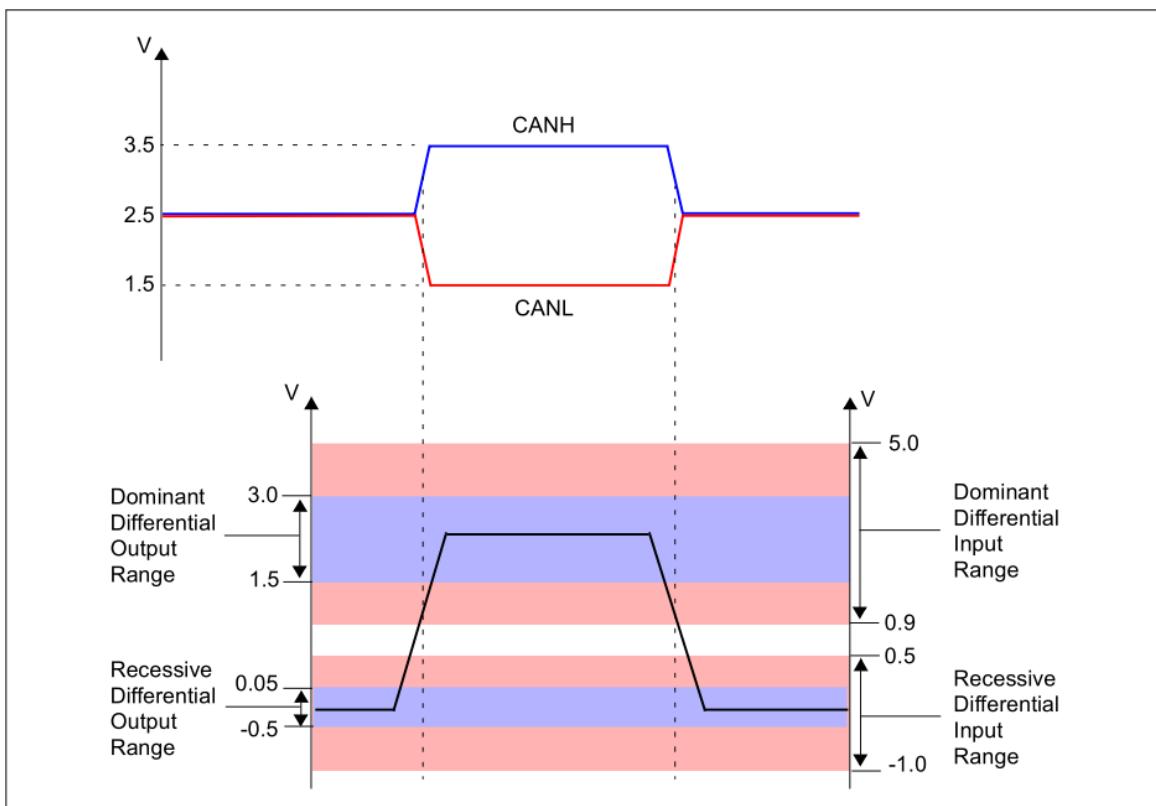


Figura 6.30: Níveis de tensão nas linhas CANL e CANH e as faixas de tensão diferencial para definição de cada um dos estados, nível lógico 1 (recessivo) e nível lógico 0 (dominante)Microchip 2002.

6.10.2 Camada de enlace de dados (*Data Link*) do CAN

O padrão RS-485 é limitado à camada física, ou seja, define aspectos do sinal correspondentes a cada bit (por exemplo, nível de tensão e duração) mas não trata do endereçamento dos dispositivos, isso deve ser feito por um protocolo que atue em uma camada superior (Modbus, por exemplo).

O padrão CAN já atua na camada física (definição dos sinais que compõem os bits) e na camada de enlace de dados (ver Figura 6.31), definindo formas das mensagens serem entregues aos destinatários de interesse e aspectos de prioridade.

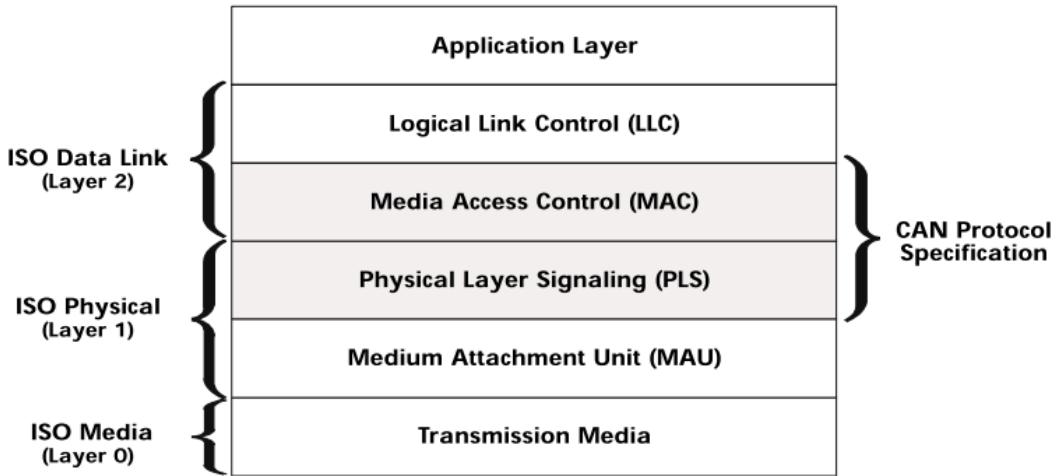


Figura 6.31: Posicionamento do padrão CAN no modelo de camadas ISO/OSI Contemporary Controls 2020b.

As transmissões CAN operam usando o modelo produtor/consumidor. Quando os dados são transmitidos por um dispositivo CAN, **nenhum outro dispositivo é endereçado**. Em vez disso, o conteúdo da mensagem é designado por um campo identificador. Este campo identificador, que deve ser único dentro da rede, não só fornece conteúdo, mas a prioridade da mensagem também. Todos os outros dispositivos CAN ouvem o remetente e aceitam apenas as mensagens de interesse. Essa filtragem dos dados é realizada usando um filtro de aceitação que é um componente integrante do chip controlador CAN. Dados que falham nos critérios de aceitação são rejeitados. Portanto, dispositivos receptores consomem apenas os dados de interesse, conforme definido pelo produtor Contemporary Controls 2020b.

Um quadro CAN consiste principalmente em um campo identificador, um campo de controle e um campo de dados (Figura 6.32).

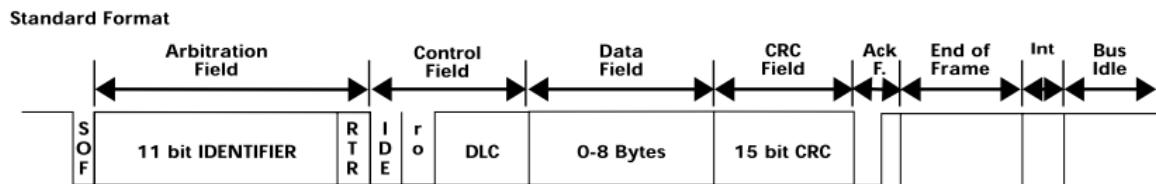
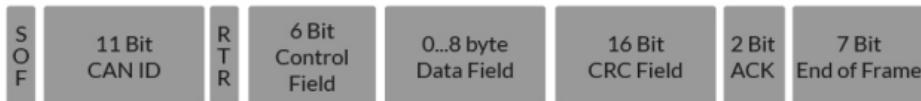


Figura 6.32: Formato do frame na especificação do CAN 2.0A Contemporary Controls 2020b.

O campo de controle tem seis bits de comprimento, o campo de dados tem zero a oito bytes de comprimento e o campo identificador tem 11 bits de comprimento para quadros padrão (especificação CAN 2.0A) ou 29 bits longo para quadros estendidos (especificação CAN 2.0B) (ver a Figura 6.33).

O protocolo de enlace de dados (*data link*) do CAN não utiliza o conceito de endereço de nó fonte e de nó destino, Contemporary Controls 2020b.

CAN 2.0A



CAN 2.0B



Figura 6.33: Frame CAN 2.0A e CAN 2.0B Innodisk 2017.

Capítulo 7

Dispositivos de Rede

Roteadores, Hubs, Switches e Bridges são todos dispositivos de conexão de rede. Um dispositivo de conexão de rede é um dispositivo que conecta dois ou mais dispositivos que estão presentes na mesma rede ou em redes diferentes.

Um dispositivo de conexão de rede pode ser um repetidor, hub, ponte, switch, roteador ou gateway.

Todos esses dispositivos de conexão operam em algumas camadas específicas do modelo OSI (Open System Interconnection). A Figura 7.1 apresenta as diferentes camadas de atuação de cada dispositivo.

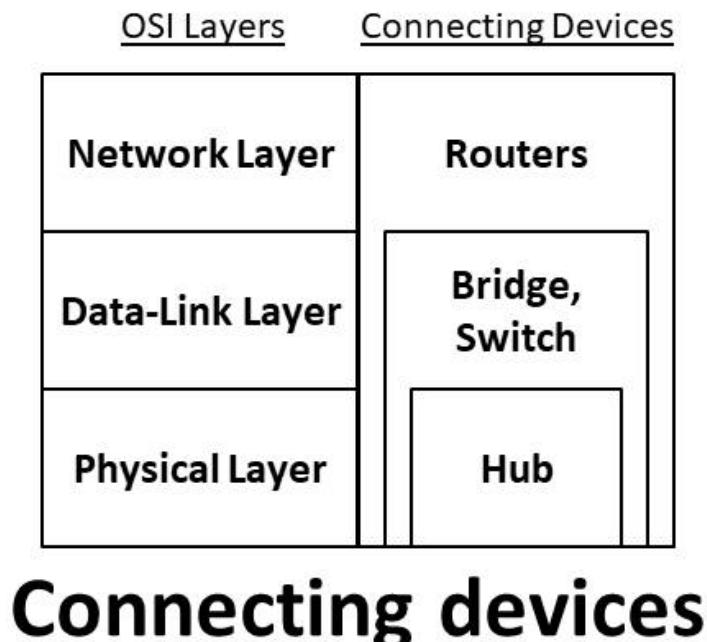


Figura 7.1: Dispositivos de rede nas camadas do modelo OSI AfterAcademy 2020.

7.1 Repetidor

Um repetidor é um dispositivo de rede usado para regenerar ou replicar sinais que estão enfraquecidos ou distorcidos pela transmissão em longas distâncias e através de áreas com altos níveis de interferência eletromagnética (EMI)

Os repetidores não amplificam o sinal. Quando o sinal fica fraco, eles copiam o sinal bit a bit e o regeneram com a intensidade original. É um dispositivo de 2 portas.

Funcionalmente, um repetidor pode ser considerado como dois transceptores unidos e conectados a dois segmentos diferentes de cabo coaxial. O repetidor passa o sinal digital bit a bit em ambas as direções entre os dois segmentos. Conforme o sinal passa por um repetidor, ele é amplificado e regenerado na outra extremidade.

O repetidor não isola um segmento do outro, se houver uma colisão em um segmento, ele é regenerado no outro segmento. Portanto, os dois segmentos formam uma única LAN e são transparentes para o resto do sistema.

A Figura 7.2 apresenta um repetidor wireless.

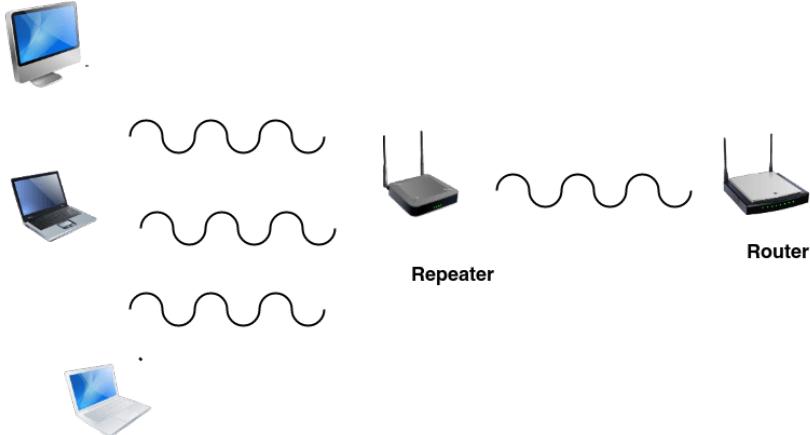


Figura 7.2: Repetidor Wireless KenCorner 2018.

A Figura 7.3 apresenta o esquema de ligação de um repetidor ligado por cabeamento.

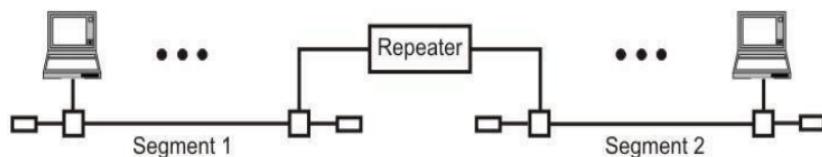


Figura 7.3: Esquema de ligação de um repetidor em uma rede cabeada IIMC Hyderabad 2020.

O repetidor (o mesmo vale para o hub) é um dispositivo de camada 1, opera apenas na rede física do modelo OSI. Por funcionar na camada física, trata principalmente dos dados na forma de bits ou sinais elétricos. A Figura 7.4 apresenta um esquema de atuação dos repetidores (e hubs) com relação ao modelo OSI.

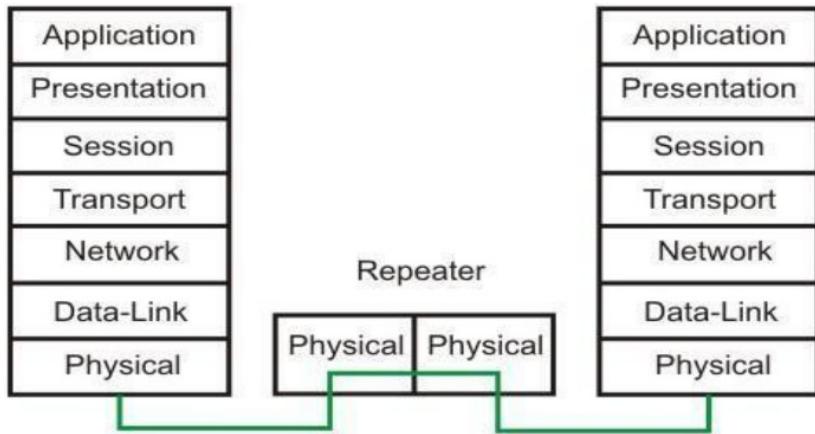


Figura 7.4: Atuação dos repetidores (também dos hubs) em relação ao modelo OSI IIMc Hyderabad 2020.

7.2 Hub

Um hub é basicamente um repetidor multiporta. Um hub conecta vários fios vindos de diferentes ramos, por exemplo, o conector em topologia em estrela que conecta diferentes estações.

Os hubs não podem filtrar dados, portanto, os pacotes de dados são enviados a todos os dispositivos conectados. Hub é um termo genérico, mas geralmente se refere a um repetidor multiporta. Ele pode ser usado para criar vários níveis de hierarquia de estações, conforme apresentado na Figura 7.5.

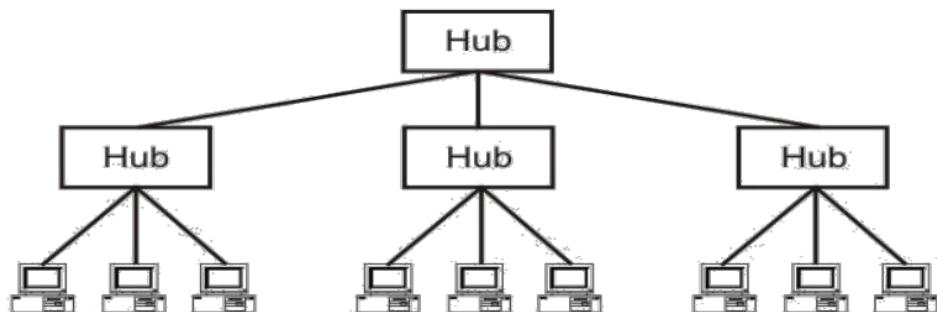


Figura 7.5: Exemplo de LAN com uso de HUBs IIMc Hyderabad 2020.

Um Hub (assim como o repetidor) não é um dispositivo inteligente, ele encaminha as mensagens recebidas para outros dispositivos sem verificar erros ou processá-los. Assim como o repetidor, o hub é um dispositivo que atua na camada física do modelo OSI, conforme o esquema da Figura 7.4.

Ele não mantém nenhuma tabela de endereços para dispositivos conectados. Ele só sabe que um dispositivo está conectado a uma de suas portas. Na figura Figura 7.6 a mensagem é enviada para todos os dispositivos conectados ao hub.

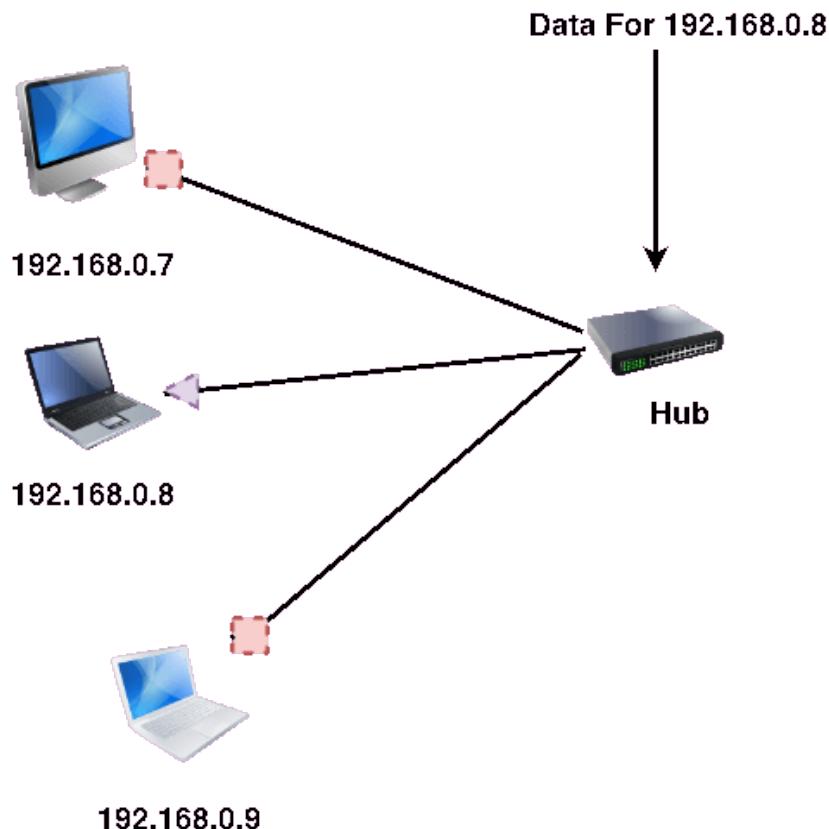


Figura 7.6: Envio de mensagens em um hub KenCorner 2018.

7.3 Ponte (Bridge)

Bridges são usadas para conectar duas LANs diferentes ou dois segmentos de rede semelhantes, para fazê-los operar como se fossem uma rede.

A ponte cria uma tabela de ponte de endereços de dispositivos físicos que é usada para determinar a ponte correta ou o destino MAC (*Media Access Control*) para uma mensagem. O MAC é um endereço de camada 2 (enlace de dados). A Figura 7.7 apresenta um exemplo de bridge com a tabela de endereço dos dispositivos conectados.

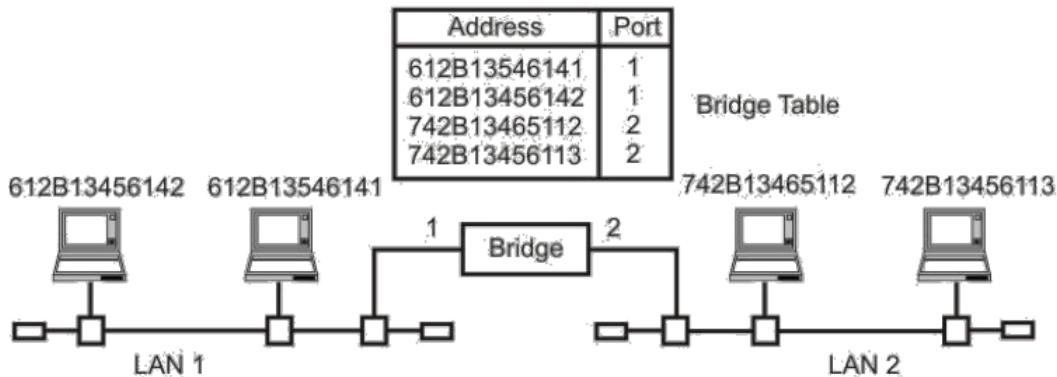


Figura 7.7: Exemplo de tabela de endereços (MAC address) de um bridge IIMc Hyderabad 2020.

Como uma ponte envia mensagens apenas para a parte da rede na qual existe o nó de destino, o efeito geral de uma ponte em uma rede é a redução do tráfego de rede e menos gargalos de mensagens.

Como o bridge (o mesmo vale para o switch) atua sobre o endereço MAC, trata-se de um dispositivo que atua na camada 2 do modelo OSI, isso é apresentado na Figura 7.8

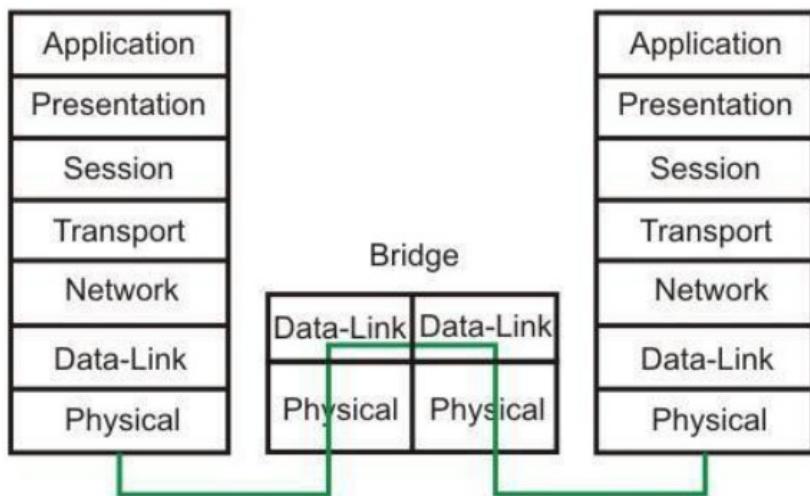


Figura 7.8: Atuação dos bridges (também dos switchs) em relação ao modelo OSI IIMc Hyderabad 2020.

7.4 Switch

Um switch é uma **ponte multiporta** com um buffer e um design que pode aumentar sua eficiência desempenho (grande número de portas implica menos tráfego).

O switch é um dispositivo de camada de enlace de dados (segunda camada do modelo OSI), seguindo o mesmo esquema da Figura 7.8. O switch pode executar verificação de erros antes de encaminhar dados, o que o torna muito eficiente, pois não encaminha pacotes com erros e encaminha pacotes bons seletivamente apenas para a porta correta.

Um switch é essencialmente uma ponte rápida com sofisticação adicional que permite um processamento mais rápido de quadros. Algumas das funcionalidades importantes são:

- As portas possuem buffer,
- Possui uma tabela com a associação “endereço - porta” para cada dispositivo conectado,
- Cada frame é encaminhado após examinar o endereço do destinatário e encaminhado para a porta adequada.

A principal diferença entre um **hub** e um **switch** é que um switch não transmite uma mensagem de entrada para todas as portas, mas, em vez disso, envia a mensagem apenas para a porta na qual existe a estação de trabalho do destinatário, com base em uma tabela MAC criada pela escuta aos nós da rede.

7.5 Roteador

Um roteador é um dispositivo como um switch que roteia pacotes de dados com base em seus **endereços IP**. O roteador é principalmente um dispositivo de camada de rede (terceira camada do modelo OSI), conforme apresentado na Figura 7.9.

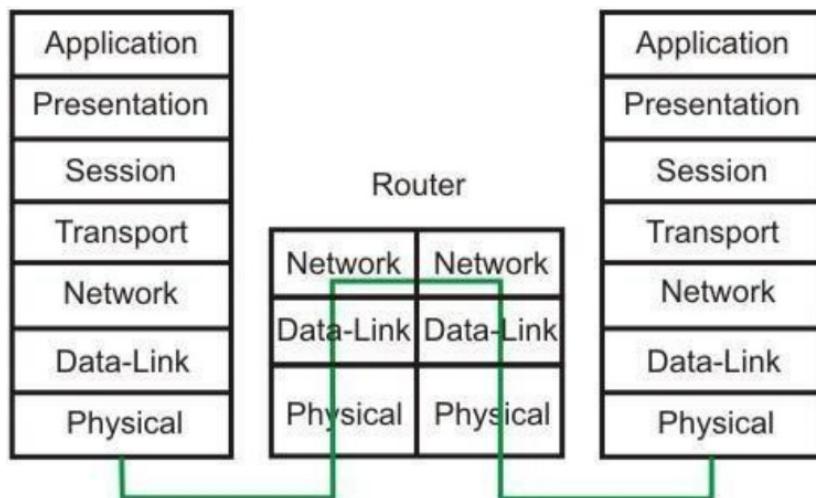


Figura 7.9: Atuação dos roteadores em relação ao modelo OSI IIMc Hyderabad 2020.

Os roteadores normalmente conectam LANs, ver Figura 7.10, e WANs e têm uma tabela de roteamento de atualização dinâmica com base na qual eles tomam decisões sobre o roteamento dos pacotes de dados. O roteador divide os domínios de broadcast dos hosts conectados por meio dele.

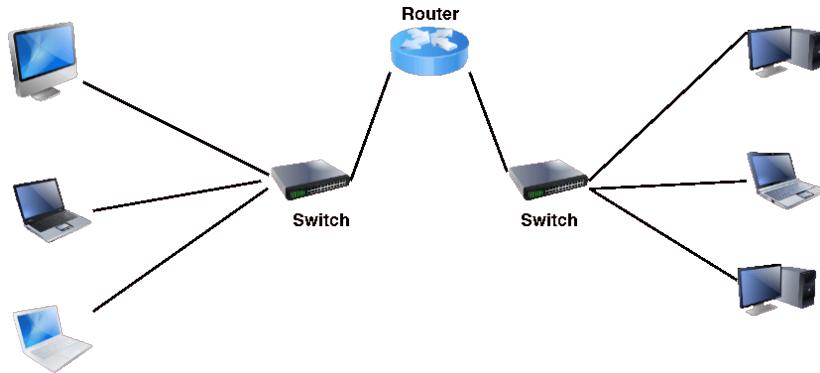


Figura 7.10: Roteador conectando duas LANs KenCorner 2018.

Os roteadores são dispositivos da **Camada 3 do modelo OSI** e encaminham dados dependendo do endereço de rede, não do endereço de hardware (MAC). Para redes TCP / IP, isso significa o endereço IP da interface de rede.

Os roteadores isolam cada LAN em uma sub-rede separada, de forma que cada endereço IP do adaptador de rede terá um terceiro "octeto" diferente (exemplo: 192.168.1.1 e 192.168.2.1 estão em sub-redes diferentes). Eles são necessários em redes grandes porque o esquema de endereçamento TCP / IP permite apenas 254 endereços por segmento de rede (Classe C).

Roteadores, como pontes, fornecem controle de largura de banda, mantendo os dados fora das sub-redes às quais não pertencem. No entanto, os roteadores precisam ser configurados antes de começarem a funcionar, embora, uma vez configurados, eles possam se comunicar com outros roteadores e aprender como chegar a partes de uma rede que são adicionadas após a configuração inicial de um roteador.

Os roteadores também são o único desses quatro dispositivos que permitirão que você compartilhe um único endereço IP entre vários clientes da rede.

7.5.1 Roteamento

As tabelas de roteamento contêm os dados para a tomada de decisões de encaminhamento. A Figura 7.11 apresenta um exemplo de tabela de roteamento.

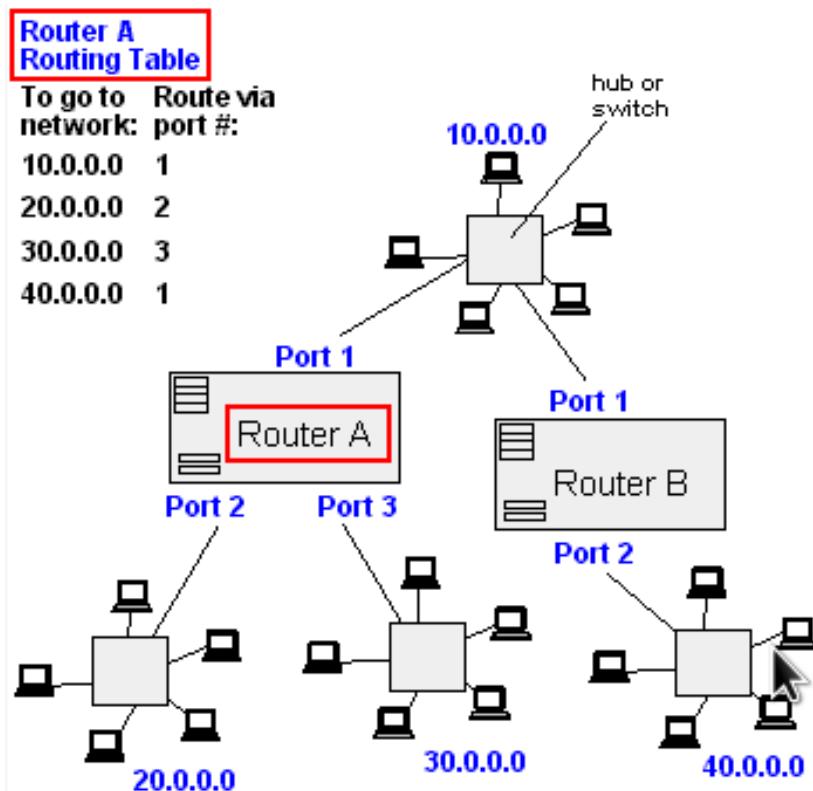


Figura 7.11: Exemplo de tabela de roteamento (Roteador A) PCMag 2020.

Embora este seja um exemplo simples, as tabelas de roteamento se tornam muito complexas. O roteamento estático usa tabelas fixas, mas o roteamento dinâmico usa protocolos de roteamento que permitem que os roteadores troquem dados entre si.

7.5.2 Classes de endereço IP

Os endereços da internet são divididos em classe, indicadas pelo valor do primeiro octeto do endereço IP. As classes são divididas conforme descrito a seguir (“N” equivale Network e “H” equivale a Host).

A **classe A** possui um conjunto de endereços que vão desde o **1.0.0.0 até 127.0.0.0**, onde o primeiro octeto (primeiros 8 bits **N.H.H.H**) de um endereço IP identifica a rede e os restantes 3 octetos (24 bits) irão identificar um determinado host nessa rede.

→ Exemplo de um endereço Classe A - 120.2.1.0

A **classe B** possui um conjunto de endereços que vão desde o **128.0.0.0 até 191.255.0.0**, onde os dois primeiros octetos (16 bits **N.N.H.H**) de um endereço IP identificam a rede e os restantes 2 octetos (16 bits) irão identificar um determinado host nessa rede.

→ Exemplo de um endereço Classe B - 152.13.4.0

A **classe C** possui um conjunto de endereços que vão desde o **192.0.0.0 até 223.255.255.0**, onde os três primeiros octetos (24 bits **N.N.N.H**) de um endereço IP identificam a rede e o restante octeto (8 bits) irão identificar um determinado host nessa rede.

→ Exemplo de um endereço Classe C - 192.168.10.0

A **classe D** possui o primeiro octeto variando de **224 a 239** e está reservado para Multi-cast.

A **classe E** possui o primeiro octeto variando de **240 a 255** e é uma faixa experimental.

7.6 Gateway

Um gateway (traduções possíveis: porta de entrada, portal, portão, acesso), como o nome sugere, é uma passagem para conectar duas redes que podem funcionar em diferentes modelos de rede. Eles funcionam basicamente como agentes de mensageiros que pegam dados de um sistema, os interpretam e os transferem para outro sistema.

Os gateways também são chamados de **conversores de protocolo e podem operar em qualquer camada de rede**. Em geral, os gateways são mais complexos que o switch ou roteador.

Um gateway pode traduzir informações entre diferentes formatos de dados de rede ou arquiteturas de rede. Ele pode traduzir TCP / IP em AppleTalk para que os computadores com suporte a TCP / IP possam se comunicar com os computadores da marca Apple.

A maioria dos gateways opera na camada de aplicativo, mas pode operar na camada de rede ou na camada de sessão do modelo OSI. A atuação do gateway é apresentada na Figura 7.12.

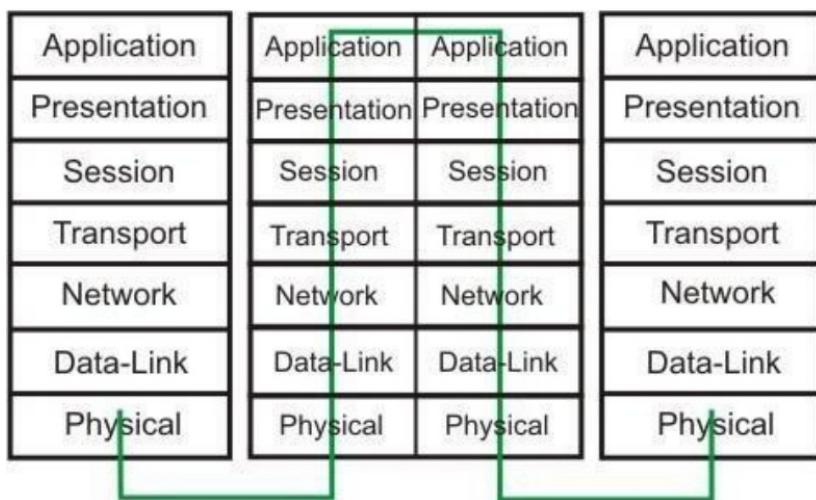


Figura 7.12: Atuação do gateway em relação ao modelo OSI IIMc Hyderabad 2020.

Os gateways começarão no nível mais baixo e removerão as informações até que cheguem ao nível necessário e reempacotarão as informações e trabalharão de volta para a camada de hardware do modelo OSI.

Para confundir as questões, ao falar sobre um roteador usado para fazer interface com outra rede, a palavra gateway é frequentemente usada. Isso não significa que a máquina de roteamento seja um gateway conforme definido aqui, embora pudesse ser IIMc Hyderabad 2020.

7.6.1 Gateways para redes industriais

Como visto, por definição o gateway atua alterando o padrão usado em uma dada camada para compatibilizar a comunicação de sistemas com arquiteturas distintas

Dada a grande variedade de protocolos usados a área industrial, o uso de gateways é importante para a compatibilização de instrumentos e redes que adotem padrões distintos. A Figura 7.13 apresenta a página da empresa Anybus para escolha dos dois protocolos a serem compatibilizados para indicação dos gateways disponíveis em catálogo.

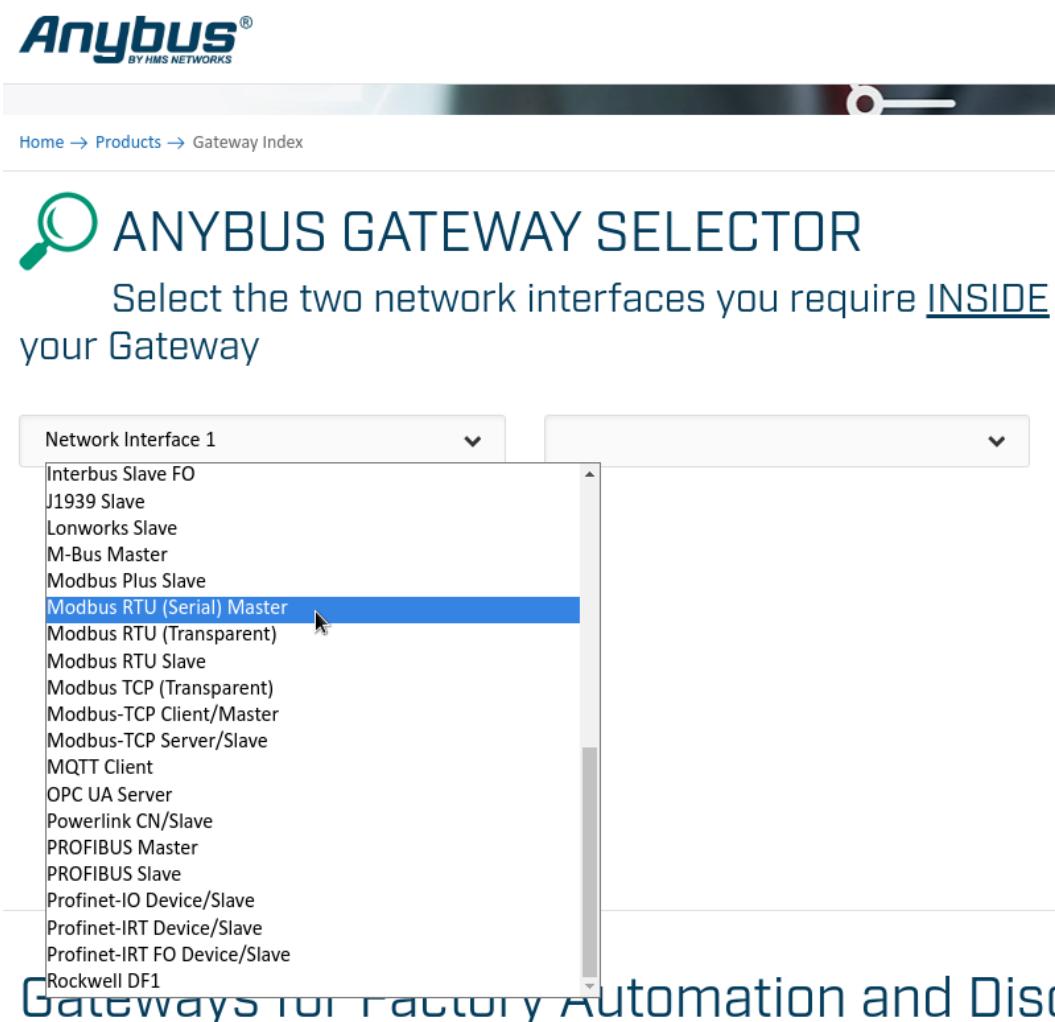


Figura 7.13: Ferramenta, da empresa Anybus, para a escolha de gateways a partir dos protocolos de interesse Anybus 2020a.

A Figura 7.14 apresenta um exemplo de resultado com o gateway disponível para possibilitar a comunicação de um dispositivo DeviceNetScanner/Master com um dispositivo PROFIBUS Slave.

ANYBUS GATEWAY SELECTOR

Select the two network interfaces you require INSIDE your Gateway

DeviceNet Scanner/Master

PROFIBUS Slave

GATEWAY SEARCH RESULTS

Anybus X-gateway – DeviceNet Scanner – PROFIBUS Slave

The chosen interfaces within this Anybus gateway are:

- DeviceNet Scanner/Master
- PROFIBUS Slave

ORDER CODE: AB7663

The Anybus X-gateway allows you to seamlessly inter-connect PLC control systems and their connected devices between DeviceNet and PROFIBUS networks. FAST COPYING OF I/O DATA The...

→ READ MORE

DeviceNet PROFIBUS

A red arrow points from the 'READ MORE' button to the gateway device image.

Figura 7.14: Exemplo de resultado da busca na página da empresa Anybus Anybus 2020a.

A Figura 7.15 apresenta a função desempenhada pelo gateway escolhido, as características de cada gateway são disponibilizadas por um link junto ao resultado da busca.



Figura 7.15: Esquema da função desempenhada por um gateway Anybus 2020b.

Capítulo 8

Foundation Fieldbus

A Fieldbus Foundation é a organização que define as especificações Foundation Fieldbus e certifica os produtos em conformidade com o padrão. O padrão Foundation Fieldbus define a maneira como novos dispositivos são adicionados à rede, como instalá-los e configurá-los. Qualquer empresa com os recursos adequados pode fazer um dispositivo Foundation Fieldbus (se passar no Teste de Conformidade) que funcionará com todos os outros dispositivos e programas certificados Foundation Fieldbus.

O objetivo do Fieldbus Foundation é ajudar a criar produtos que utilizem uma rede industrial robusta com base em padrões existentes e outras tecnologias testadas. Foundation Fieldbus é um padrão aberto, que permite o uso de produtos Foundation Fieldbus de diferentes fornecedores de forma intercambiável.

8.1 Usos do Fieldbus

O Foundation Fieldbus é usado no controle e monitoramento de processos. Controle de processo se refere a processos contínuos como controle de fluxo, controle de temperatura e controle de nível do tanque. Esses tipos de processos são normalmente encontrados em locais como refinarias de petróleo, fábricas de produtos químicos e fábricas de papel. O Foundation Fieldbus também pode ser usado para monitoramento em longas distâncias.

O Foundation Fieldbus **implementa controle distribuído**, o que significa que o controle é feito pelos dispositivos em vez de um computador de monitoramento. Entradas, saídas e dispositivos de controle de processo configurados em uma rede Fieldbus pode ser executado independentemente de um sistema computadorizado National Instruments 2014.

8.2 A rede Fieldbus

Foundation Fieldbus é um sistema de comunicação multiponto, totalmente digital, que traz os algoritmos de controle para os instrumentos. O Foundation Fieldbus é uma rede local (LAN) para dispositivos Foundation Fieldbus, incluindo sensores de controle de processo, atuadores e dispositivos de controle. O Foundation Fieldbus oferece suporte à codificação digital de dados e muitos tipos de mensagens National Instruments 2014.

Uma configuração simples de rede Foundation Fieldbus é mostrada na Figura 8.1.

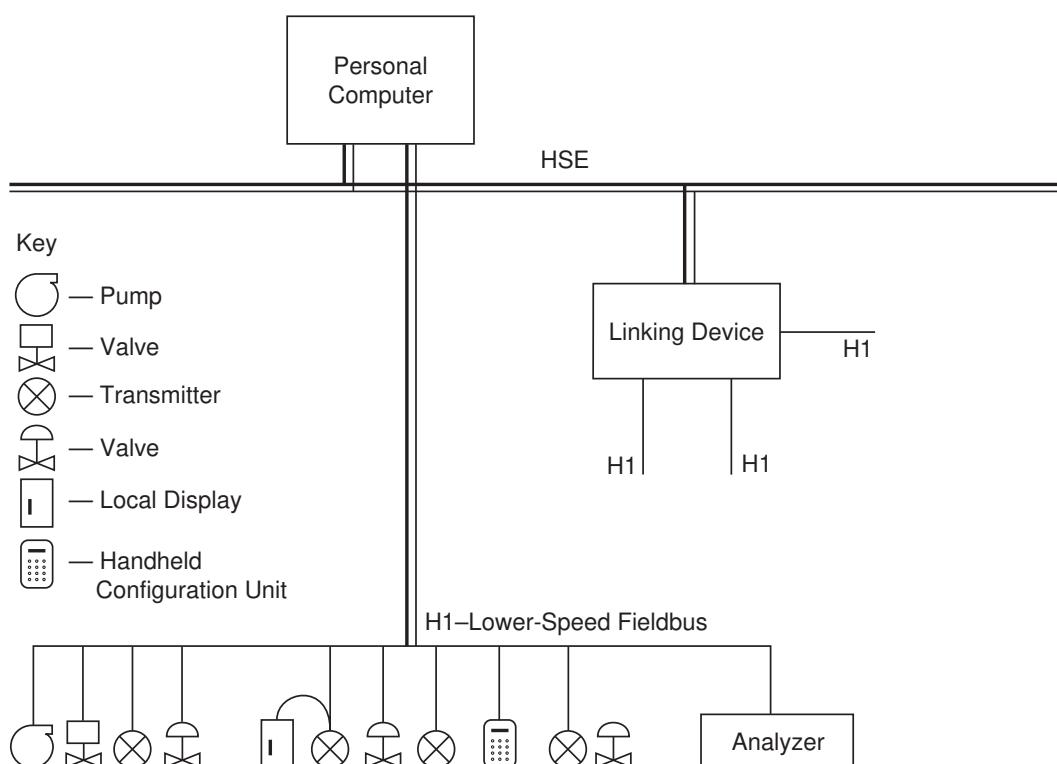


Figura 8.1: Exemplo de um sistema de controle Foundation Fieldbus National Instruments 2014.

8.3 Protocolos de comunicação Foundation Fieldbus

O Foundation Fieldbus possui dois protocolos de comunicação: **H1** e **HSE**. O primeiro, H1, transmite em 31,25 Kb/s e é usado para conectar os dispositivos de campo. O segundo protocolo, Ethernet de alta velocidade - *High Speed Ethernet* (HSE) - usa Ethernet de 10 ou 100 Mbps como camada física e fornece um backbone de alta velocidade para a rede. National Instruments 2014.

O **H1** é um sistema de comunicação totalmente digital, serial e bidirecional rodando a 31,25 kbit/s que interconecta equipamentos de campo, como sensores, atuadores e controladores. O H1 é uma rede local (LAN) para instrumentos usados em processos e automação de fabricação com capacidade embarcada para distribuir as tarefas de controle pela rede. National Instruments 2014.

O **HSE** é baseado em protocolos padrão Ethernet/IP/TCP/UDP de 10/100 Mbps e suporta as mesmas funções que o H1, mas com uma largura de banda muito maior (10/100 Mbps). Sua grande capacidade de transmissão de dados, junto com a funcionalidade inerente do Foundation Fieldbus, e acesso publish/subscribe (publish/subscribe é uma forma de comunicação alternativa ao sistema mestre/escravo), permite a integração de toda a planta em processos indústrias National Instruments 2014.

As redes Foundation Fieldbus podem ser compostas por um ou mais destes segmentos interconectados. As sub-redes HSE podem usar uma variedade de dispositivos de interconexão disponíveis comercialmente como hubs, switches, bridges, roteadores e firewalls. Os links H1 são interconectados fisicamente apenas por pontes Foundation Fieldbus H1 Data Link. As interconexões HSE para H1 são realizadas pelos Foundation Fieldbus Linking Devices. National Instruments 2014.

Uma topologia de rede típica tem conexões HSE entre computadores e executa links H1 mais lentos (31,25 Kbps) entre os próprios dispositivos. Dispositivos projetados para HSE podem ser conectados à rede HSE diretamente. A maioria dos dispositivos é projetada para usar um protocolo ou outro National Instruments 2014.

H1 e HSE foram especificamente projetados como redes complementares. O padrão H1 é otimizado para aplicações de controle de processo tradicionais, enquanto HSE, que emprega equipamento Ethernet comerciais de baixo custo, foi projetado para aplicações de controle de alto desempenho e integração de informações da planta. A solução combinada H1/HSE Fieldbus permite a integração total de processos de controle básicos e avançados National Instruments 2014.

8.4 Conceitos do padrão H1

Existem seis partes conceituais para uma rede Fieldbus:

- links,
- dispositivos (*devices*),

- blocos e parâmetros,
- ligações (*linkages*),
- loops,
- programações (*schedules*).

8.4.1 Links

Uma rede Foundation Fieldbus é composta de dispositivos conectados por um barramento serial. **Este barramento serial é chamado de *link*** (também conhecido como segmento). Uma rede Fieldbus consiste em um ou mais links. Cada link é configurado com um identificador de link exclusivo National Instruments 2014.

Cada link na rede Fieldbus conecta dispositivos físicos. Os dispositivos podem ser dispositivos de campo (transmissores de temperatura, válvulas e assim por diante) ou dispositivos *host* (PCs, sistemas de controle distribuído). Cada dispositivo físico é configurado com uma etiqueta (tag) de dispositivo físico, um endereço e um ID de dispositivo. O tag do dispositivo físico deve ser único dentro de um sistema Fieldbus, e o endereço deve ser único dentro de cada link. O fabricante do dispositivo atribui um ID de dispositivo que é exclusivo para o dispositivo. National Instruments 2014.

A Figura 8.2 apresenta o esquema de um Link da rede FF.

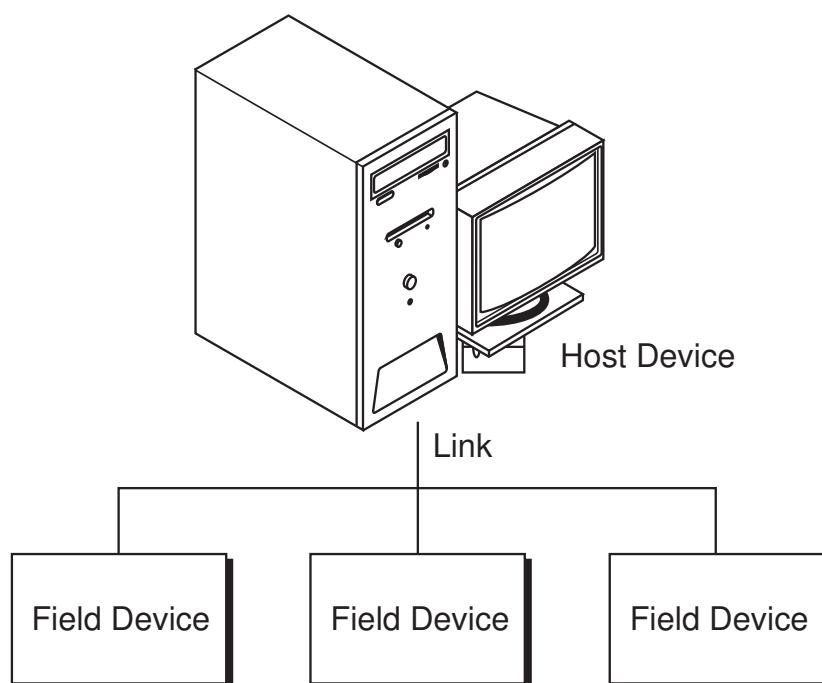


Figura 8.2: Link na rede FF National Instruments 2014.

8.4.2 Dispositivos (*devices*)

Os dispositivos são identificados de forma única na rede Fieldbus por um nome (string de caracteres) ou tag. O **tag** do dispositivo é um atributo configurável do dispositivo que geralmente descreve o tipo do dispositivo. Os tags de dispositivo são únicos para cada dispositivo em uma rede Fieldbus.

Outro identificador exclusivo de um dispositivo é o **ID** do dispositivo, que inclui um número de série único para o dispositivo. O ID do dispositivo é atribuído pelo fabricante do dispositivo, não sendo possível o usuário configurá-lo.

Existem três tipos de dispositivos em uma rede Fieldbus H1: link masters, dispositivos básicos e pontes H1 (*bridges*).

1. **Link master** - Um dispositivo de link master é capaz de controlar o tráfego de comunicações em um link sincronizando a comunicação na rede. Cada rede Fieldbus precisa de pelo menos um dispositivo com capacidade de link master. Um link master pode ser uma placa de interface em um PC, um sistema de controle distribuído ou qualquer outro dispositivo, como uma válvula ou um transdutor de pressão. Os link masters não precisam ser dispositivos separados; eles podem ter funcionalidade de E/S (para por exemplo, você pode comprar transmissores de temperatura com ou sem capacidade de ser um link master).

O Foundation Fieldbus pode operar independentemente de um sistema de computador por causa dos link masters no barramento. Os link masters têm capacidade de processamento e são capazes de controlar o barramento. Depois de realizar a configuração do(s) dispositivo(s), a malha de controle pode continuar a operar, mesmo se o computador de monitoramento estiver desconectado.

Todos os links masters recebem as mesmas informações no momento do download (configuração dos dispositivos), mas **apenas um link master controlará ativamente o barramento em um determinado momento**. O link master que está atualmente o controle do barramento é denominado **Link Active Scheduler (LAS)**. Se o **Link Active Scheduler** falha, o próximo **link master** assumirá de forma transparente e começará a controlar as comunicações do barramento onde o **Link Active Scheduler** anterior parou. Portanto, nenhuma configuração especial é necessária para implementar a redundância.

O dispositivo **Link Active Scheduler** segue a programação realizada para ele e para os outros link masters durante o processo de configuração. Nos momentos apropriados, ele envia comandos para outros dispositivos, dizendo-lhes quando transmitir dados. O

Link Active Scheduler também publica informações de tempo e concede permissão aos dispositivos para permitir que eles transmitam mensagens não programadas (acíclicas), como alarmes e eventos, manutenção e informações de diagnóstico, invocação de programa, permissivos e intertravamentos, informações de exibição e tendência e configuração.

2. **Dispositivo básico (*Basic device*)** - Um dispositivo básico é um dispositivo que não é capaz de controlar a comunicação do barramento. Dispositivos básicos não podem se tornar o Link Active Scheduler (LAS).
3. **Ponte H1 (*H1 bridge*)** - Os dispositivos ponte (bridge) conectam links juntos em uma árvore estendida. Eles são sempre dispositivos mestre do link (link master devices) e eles devem ser o Link Active Scheduler (LAS) (Nota: Aqui entendi que caso o bridge deixe de operar outro dispositivo Link Master assume a função de LAS dentro do seu barramento, mantendo cada segmento da rede FF funcionando). Uma ponte H1 é um dispositivo conectado a vários links H1, cuja camada de enlace realiza encaminhamento e republicação entre os links.

Nota Existem diferenças entre uma ponte e um gateway. Enquanto uma ponte conecta redes de diferentes velocidades e/ou camadas físicas, um gateway conecta redes que usam diferentes protocolos de comunicação.

A Figura 8.3 apresenta os dispositivos de rede FF.

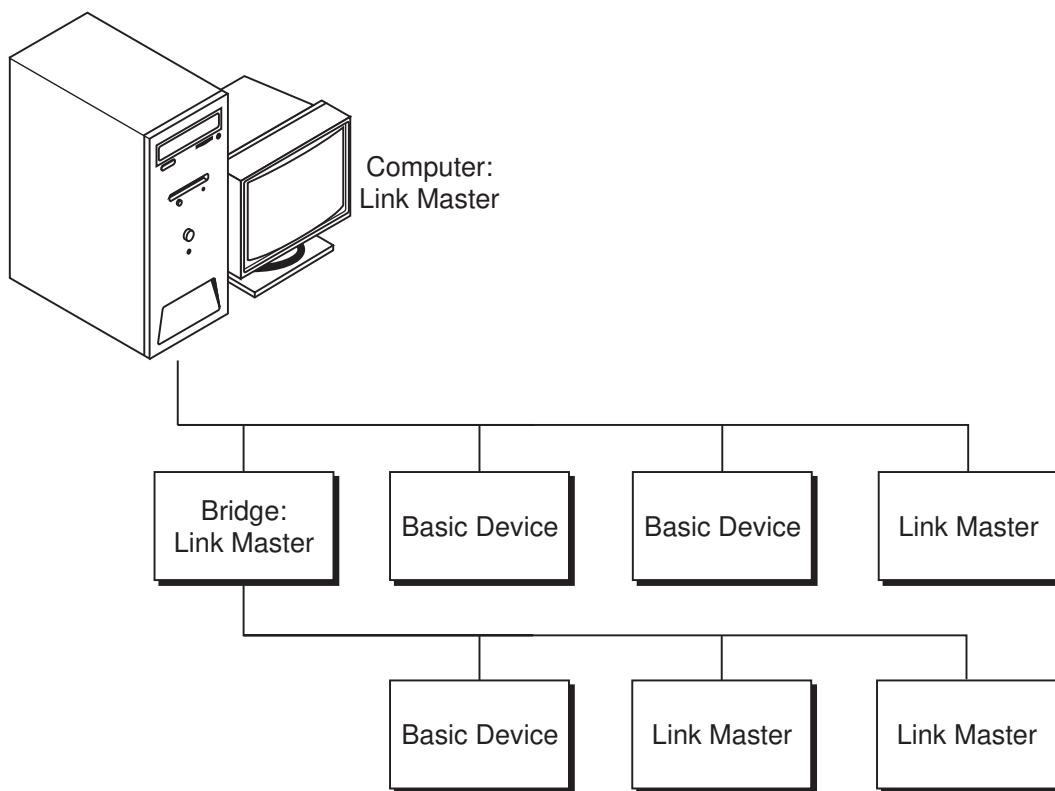


Figura 8.3: Dispositivos de redes FF National Instruments 2014.

8.4.3 Blocos e parâmetros

Os blocos são apresentados na Seção 8.6.

8.4.4 Ligações (*Linkages*)

Os blocos de funções configurados para controlar um processo são vinculados ou conectados por objetos de configuração dentro dos dispositivos. Essas ligações permitem o envio de dados de um bloco para outro. Uma ligação (*linkage*) é diferente de um link. Um link é um par de fios físicos que conecta dispositivos em um Rede Fieldbus e uma ligação (*linkage*) é uma conexão lógica que conecta dois blocos de funções National Instruments 2014.

8.4.5 Loops

Um *loop* (ou loop de controle) é um grupo de blocos de função conectados por ligações (*linkages*) executando em um taxa configurada. Cada bloco executa na taxa configurada e os

dados se movem pelas ligações entre os blocos na taxa configurada. A Figura 8.4 mostra um exemplo de malha de controle National Instruments 2014.

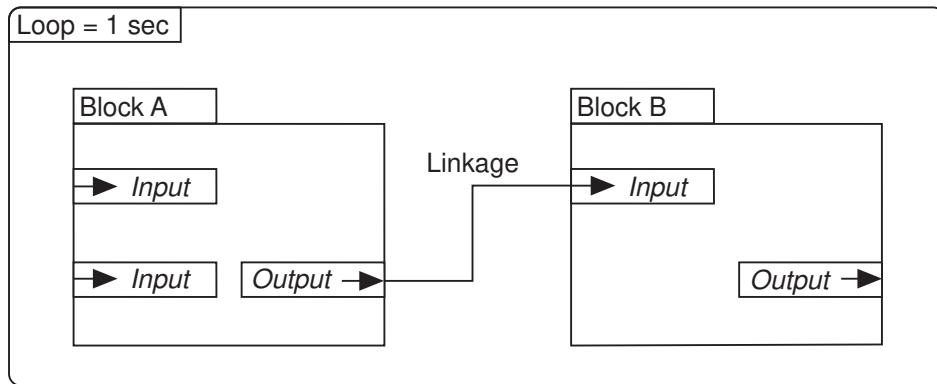


Figura 8.4: Malha de controle National Instruments 2014.

Múltiplos loops

É possível ter vários loops em execução em taxas diferentes em um link. A Figura 8.5 mostra um exemplo de múltiplos loops.

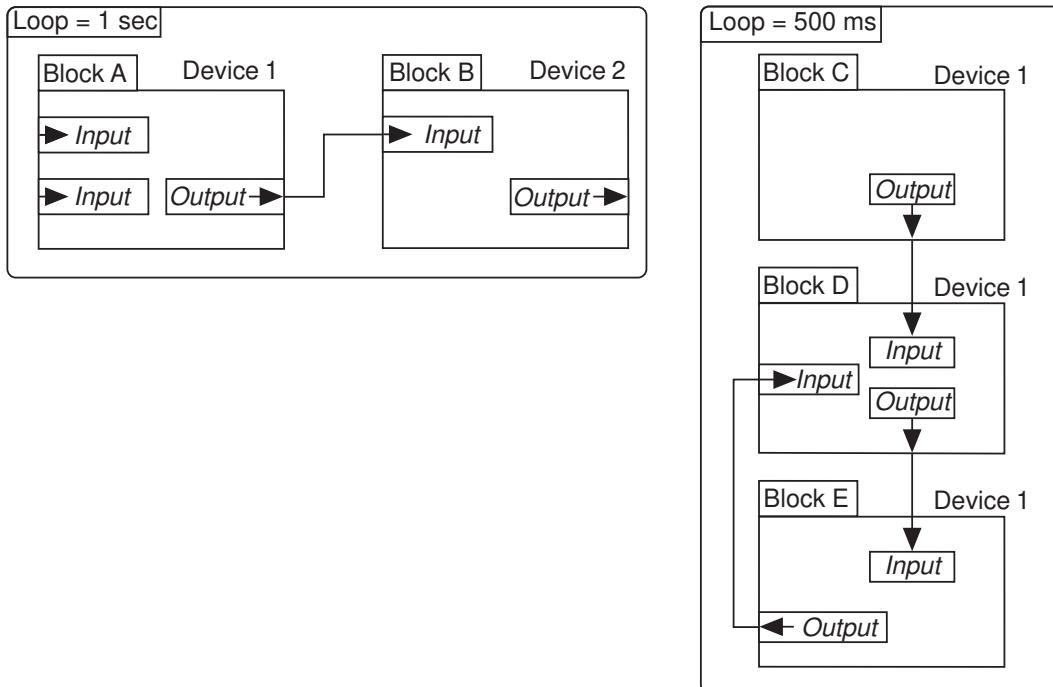


Figura 8.5: Múltiplos loops executando com diferentes taxas de transmissão National Instruments 2014.

Mesmo se os loops estiverem sendo executados em taxas diferentes, eles podem enviar

dados uns aos outros por meio de ligações. A Figura 8.6 mostra um exemplo de ligação entre dois loops. Todos os loops em um link são executados em um macrociclo. Um macrociclo (*macrocycle*) é o mínimo múltiplo comum de todos os tempos de loop em um determinado link. Por exemplo, o macrociclo na Figura 8.6 é de 1 segundo.

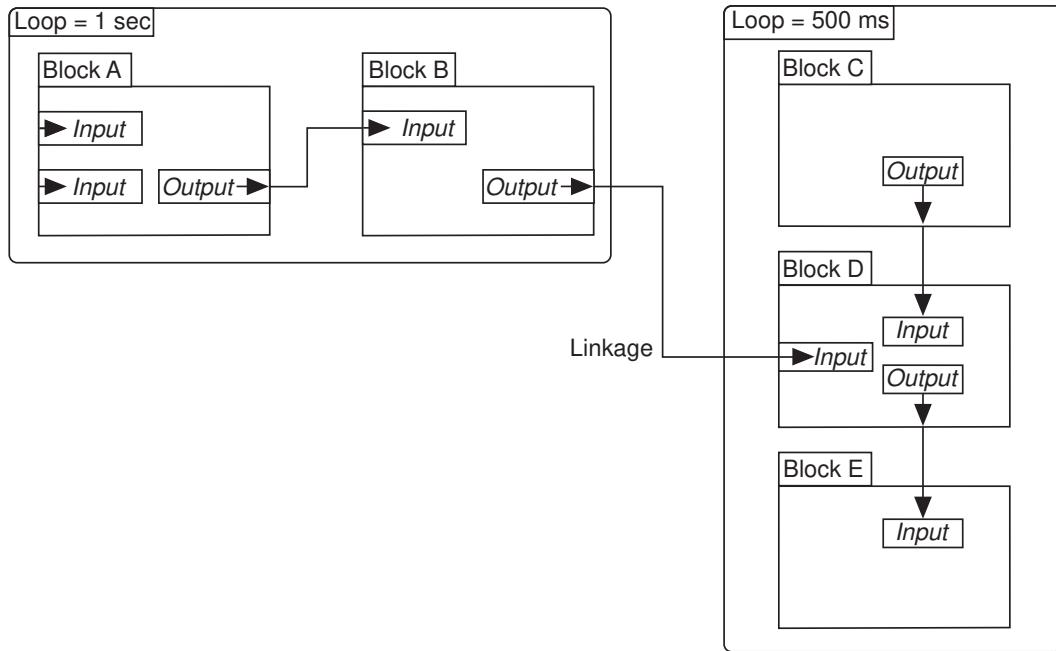


Figura 8.6: Ligação entre dois loops National Instruments 2014.

8.5 Conceitos do padrão HSE

Esta seção discute os conceitos da arquitetura de HSE. Existem quatro partes conceituais em uma rede HSE: Dispositivo HSE, Dispositivo de campo HSE, Dispositivo de link e Dispositivo de gateway de E/S.

8.5.1 Dispositivo (*device*) HSE

Um dispositivo HSE é qualquer tipo de dispositivo Foundation Fieldbus conectado diretamente ao barramento HSE. Todos os dispositivos HSE contêm um FDA Agent, um HSE SMK (System Management Kernel) e um HSE NMA (Network Management Agent) virtual field device (VFD). Os exemplos incluem dispositivos de link, dispositivos de gateway de E/S e dispositivos de campo HSE National Instruments 2014.

8.5.2 Dispositivo de campo HSE

Um dispositivo de campo HSE é um dispositivo HSE que também contém pelo menos uma aplicação de processo por bloco de função - *Function Block Application Process* (FBAP) National Instruments 2014.

8.5.3 Linking Device

Dispositivos de link são dispositivos HSE usados para conectar links **H1** à rede **HSE**. Eles fornecem acesso entre dispositivos HSE e dispositivos H1 e acesso entre dispositivos **H1** interconectados por uma rede HSE. Um dispositivo de link também pode conter uma ponte H1 que fornece comunicações **H1** para **H1** entre segmentos H1 ligados por ponte National Instruments 2014.

8.5.4 Dispositivo Gateway de E/S (*I/O Gateway Device*)

Um dispositivo de gateway de E/S é um dispositivo HSE usado para fornecer acesso para redes HSE a dispositivos que não são Foundation Fieldbus por meio de blocos de função National Instruments 2014.

8.6 Tecnologia Foundation Fieldbus

Essa seção apresenta alguns aspectos característicos do Foundation Fieldbus.

8.6.1 Camada física da rede FF H1

A rede Foundation Fieldbus H1 é usada para comunicação entre instrumentação de campo e automatização de processos e sistemas de controle. A camada física (fiação, sinais elétricos e mecanismo de transferência de dados) é baseado no padrão internacional IEC 61158-2 Endress+Hauser 2020. Isso significa que:

- os equipamentos podem obter energia do barramento, que é opcionalmente intrinsecamente seguro,
- o comprimento máximo do barramento (sem repetidores) é 1900m,

- até 32 dispositivos podem ser conectados a um segmento operando em uma área segura.
- o número de dispositivos permitidos em uma área classificada depende do conceito de segurança adotado (Ex i ou Ex d) e de suas propriedades individuais.

A camada física da rede FF H1 é a mesma utilizada para o PROFIBUS-PA, de forma que muitos dos princípios básicos para dimensionamento, fiação e aterrramento do barramento são idênticos. Os dois diferem no momento em que o Foundation Fieldbus ainda precisa adotar o conceito FISCO para instrumentação intrinsecamente segura (um grupo de trabalho agora foi criado). Isso significa que uma prova individual deve ser fornecida de que os barramentos operando em áreas com risco de explosão as áreas são seguras Endress+Hauser 2020.

8.6.2 Dispositivos virtuais

O dispositivo de campo virtual - *virtual field device* (VFD) - é um modelo para visualização remota de dados descritos no dicionário de objetos. Os serviços fornecidos pela **Fieldbus Messaging Specification** permitem que se leia e escreva informações sobre o dicionário de objetos, leia e grave as variáveis de dados descritas no dicionário de objetos e execute outras atividades, como upload/download de dados e chamada de programas dentro de um dispositivo.

Cada dispositivo físico no Fieldbus pode ter um ou mais dispositivos virtuais de campo. Um aplicativo de configuração de rede pode atribuir a cada dispositivo virtual de campo um tag que é exclusivo dentro do dispositivo. A maioria dos dispositivos possui apenas um dispositivo de campo virtual. Cada dispositivo virtual de campo possui um bloco de recursos e um ou mais blocos de função e blocos transdutores. **Cada bloco deve receber um tag único no sistema Fieldbus.**

8.6.3 Camada de usuário (*User Layer*)

A camada do usuário fornece a interface para interação do usuário com o sistema. A camada do usuário usa a **descrição do dispositivo** para informar o sistema host sobre os recursos do dispositivo. A camada do usuário define **blocos** e **objetos** que representam as funções e dados disponíveis em um dispositivo.

Em vez de fazer a interface com um dispositivo por meio de um conjunto de comandos, como a maioria dos protocolos de comunicação, o **Foundation Fieldbus** permite que se interaja com os dispositivos por meio de um conjunto de blocos e objetos que definem os recursos do dispositivo de maneira padronizada.

A camada do usuário para um dispositivo consiste no bloco de recursos e um ou mais blocos transdutores e blocos de função, conforme ilustrado na Figura 8.7.

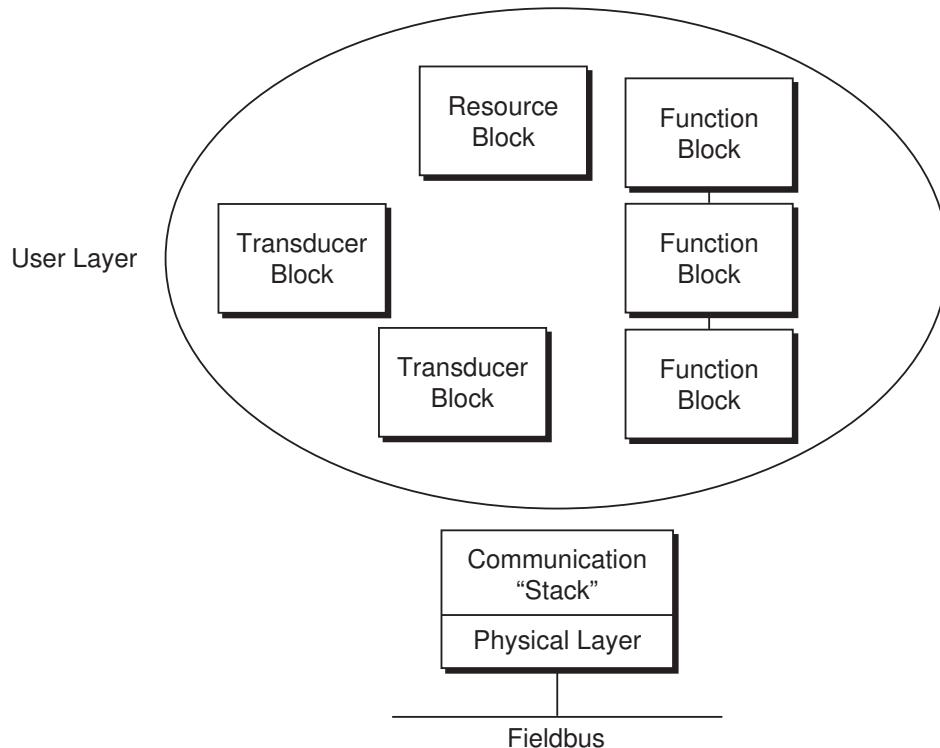


Figura 8.7: Camada de usuário National Instruments 2014.

8.6.4 Blocos

Os blocos podem ser considerados unidades de processamento. Eles podem ter entradas, configurações para ajustar o comportamento e um algoritmo que executam para produzir saídas. Eles também sabem como se comunicar com outros blocos. Os três tipos de blocos são o bloco de recursos (*resource block*), bloco transdutor (*transducer block*) e bloco de função (*function block*).

8.6.4.1 Resource Block

Um bloco de recursos especifica as características gerais do recurso (ou bloco). Isso inclui o tipo e revisão do dispositivo, ID do fabricante, número de série e estado do recurso.

Cada dispositivo possui apenas um bloco de recursos. O bloco de recursos também contém o estado de todos os outros blocos do dispositivo.

O bloco de recursos deve estar no modo automático para que quaisquer outros blocos no dispositivo sejam executados. O bloco de recursos é um bom lugar para começar a solucionar problemas se o dispositivo não estiver se comportando conforme desejado. O bloco de recursos possui parâmetros de diagnóstico que ajudam a determinar a causa de problemas.

8.6.4.2 Transducer Blocks

Os blocos transdutores realizam a leitura de sensores físicos nos blocos de função. Os blocos transdutores separam os blocos de função dos detalhes de hardware de um determinado dispositivo, permitindo a indicação genérica de entrada e saída do bloco de função.

O bloco transdutor conhece os detalhes dos dispositivos de E/S e como realmente ler o sensor ou comandar o atuador. O bloco transdutor realiza a digitalização, filtragem e conversões de escala necessárias para fornecer o valor do sensor aos blocos de função e/ou fazer a alteração na saída conforme ditado pelo bloco de função. **Geralmente, haverá um bloco transdutor por canal do dispositivo.** Em alguns dispositivos, os multiplexadores permitem que vários canais sejam associados a um bloco transdutor.

Os fabricantes podem definir seus próprios blocos transdutores. Para alguns dispositivos, a funcionalidade do bloco transdutor está incluída no bloco de funções. Não existem blocos transdutores separados para tais dispositivos.

Nota: Existem muitos parâmetros que podem ser alterados para modificar a funcionalidade de E/S.

8.6.4.3 Function Blocks

Os blocos de funções fornecem o controle e o comportamento de E/S.

Normalmente, um dispositivo possui um conjunto de funções que pode executar. Essas funções são representadas como blocos de funções dentro do dispositivo. Um bloco de função pode ser visto como uma unidade de processamento. Os blocos de funções são usados como blocos de construção na definição da aplicação de monitoramento e controle.

A especificação **Function Block Application Process** do Foundation Fieldbus define um conjunto padrão de blocos de função, incluindo 10 para controle básico e 19 para controle

avançado. A Tabela 8.1 mostra os 10 blocos de função para o controle mais básico e as funções de E/S.

Function Block Name	Symbol
Analog Input	AI
Analog Output	AO
Bias/Gain	BG
Control Selector	CS
Discrete Input	DI
Discrete Output	DO
Manual Loader	ML
Proportional/Derivative	PD
Proportional/Integral/Derivative	PID
Ratio	RA

Tabela 8.1: Dez Blocos de Funções (Function Blocks) definidos pelo Foundation Fieldbus.

Nem todos os dispositivos contêm todos os 10 blocos de função padrão. Além disso, os fabricantes também podem definir seus próprios blocos de funções. Um arquivo de descrição de dispositivo fornecido com cada dispositivo informa ao software de configuração sobre os blocos de função adicionados. Portanto, os blocos de função definidos pelo fabricante são tão fáceis de usar quanto os blocos de função padrão.

Blocos de funções diferentes fazem coisas diferentes. Pode haver muitos blocos de função presentes em um dispositivo ao mesmo tempo. Os blocos de funções são armazenados na memória do dispositivo. Alguns dispositivos vêm com blocos de funções específicos pré-carregados na memória. Eles não podem ser excluídos, nem podem ser adicionados novos blocos de funções. Outros dispositivos permitem que blocos de funções sejam instanciados (criados) e excluídos conforme necessário.

8.6.4.4 Parâmetros dos Blocos de função (*Function Block*)

Você pode alterar o comportamento de um bloco alterando as configurações de seus parâmetros. Os parâmetros do bloco de funções são classificados da seguinte forma:

- Os parâmetros de entrada recebem dados de outro bloco.
- Os parâmetros de saída enviam dados para outro bloco.

- Os parâmetros contidos não recebem ou enviam dados; eles estão contidos no bloco.

Alguns parâmetros contêm várias configurações chamadas campos.

Parte II

Meios de Comunicação

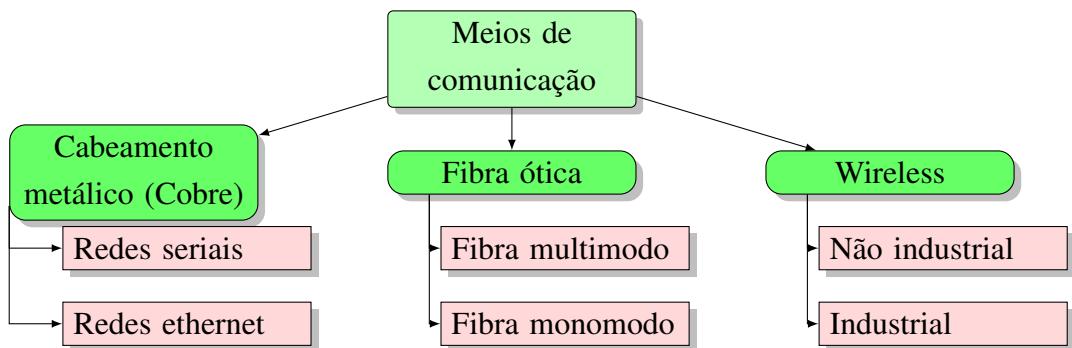
Capítulo 9

Meios de Comunicação

Neste capítulo são apresentados, de forma resumida, os meios de comunicação utilizados em redes industriais.

Devido ao grande número de opções e variantes, serão apresentados os principais conceitos, de forma que se tenha uma visão geral sobre o tema.

A estruturação do tema pode seguir diferentes esquemas. A divisão conforme o princípio físico de transmissão, eletricidade, luz ou radiofrequência parece ser uma das divisões mais fáceis de ser assimilada.



Outra divisão bastante importante é conforme o nível ocupado na estrutura de pirâmide da automação, conforme apresentado na Figura 9.1.

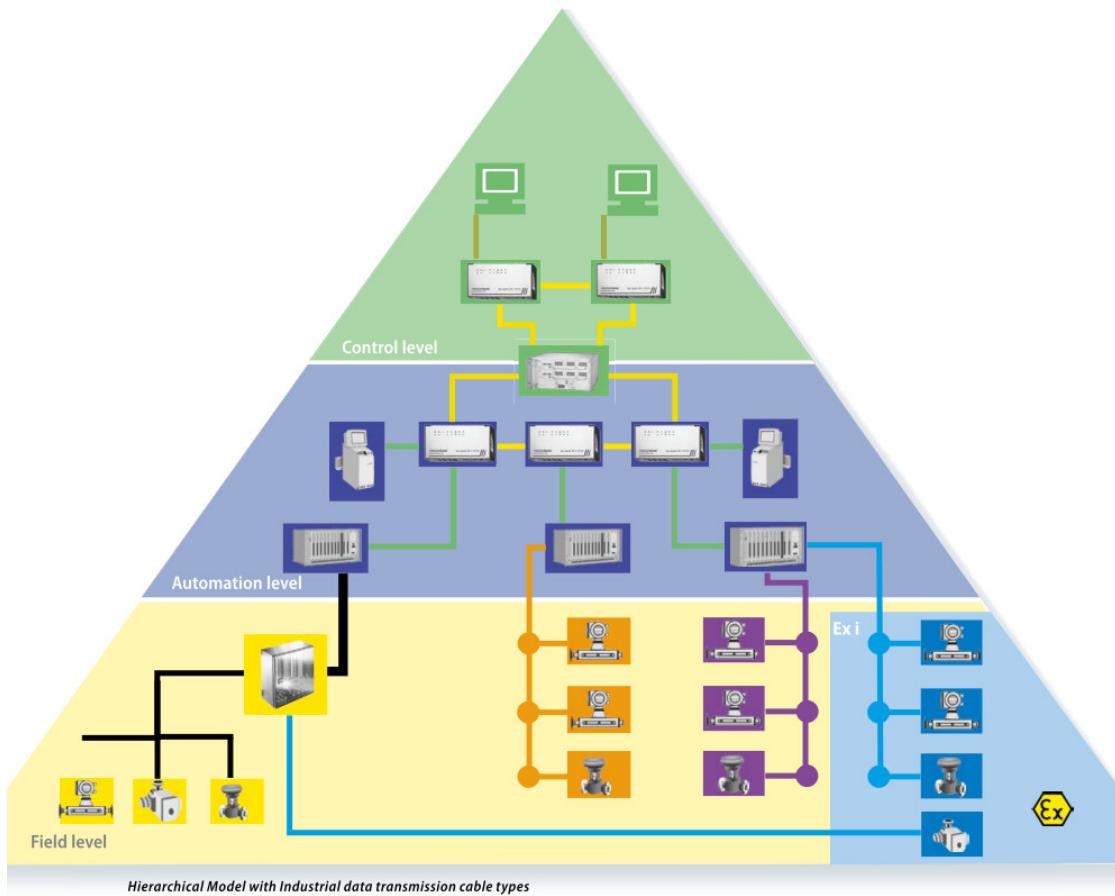


Figura 9.1: Nível hierárquico para tipos de cabos (Leoni Group 2020).

Essa divisão é relevante porque os diferentes níveis possuem requisitos característicos, como velocidade ou questões de segurança para áreas potencialmente perigosas, o que é apresentado na Figura 9.2.

Control level

This level controls and monitors higher functions with bus cycle times of <1000 ms.

- Cables:
- Ethernet TCP/IP copper cable
 - Fiber optic cable

Automation level

This level controls the actual processes and control loops with bus cycle times of <100 ms.

- Cables:
- HSE-Industrial Ethernet cable
 - Profibus FMS/DP cable

Field level

This level transfers data of the actuators and sensors; this requires a bus cycle time of <10 ms.

- Cables:
- 4 ... 20 mA/Hart Instrumentation cable
 - FOUNDATION™ Fieldbus cable
 - Profibus PA cable
 - Ex i

Figura 9.2: Tipos de cabos para cada nível hierárquico (Leoni Group 2020).

Capítulo 10

Cabeamento metálico (cobre)

Esta seção apresenta alguns aspectos relacionados às redes cabeadas. Adotou-se aqui a divisão entre os cabos usados em redes seriais e as redes ethernet. Os padrões seriais são os mais tradicionais apresentando uma ampla variedade de protocolos. As redes ethernet industriais apresentam uma maior taxa de transmissão, apresentando o conceito de categoria para sua classificação.

10.1 Cabeamento para redes seriais

Com relação às redes cabeadas pode-se listar uma série de características, como impedância, taxa máxima de transmissão, resistência mecânica.

Como cada padrão de rede possui requisitos próprios, os catálogos de fornecedores são um recurso importante para a escolha do modelo de cabo. Os catálogos podem trazer modelos de cabos já destinados para determinada rede, ou tabelas com a indicação de códigos para modelos de cabos que atendam determinada rede.

O catálogo da empresa LEONI apresenta, de forma bastante didática, aspectos relevantes sobre cabeamentos, como o atendimento ao padrão IEC 61158-2, adotado nas redes PROFIBUS PA, FOUNDATIONTM Fieldbus H1. Esse catálogo também possui muitos detalhes construtivos com boas imagens para a visualização da estrutura do cabo, como o exemplo apresentado na Figura 10.1. O link para o catálogo está na referência da figura.




Basic 70 °C	Basic 70 °C, lead sheath with steel wire armour	FOUNDATION Fieldbus H1 100 Ω
Spur and trunk cable for fixed installation indoor and outdoor, on racks, trays, in conduits, in dry and wet locations.	Spur and trunk cable for fixed installation indoor and outdoor, on racks, trays, in conduits, in dry and wet locations. Recommended for direct burial, especially in presence of oil and aggressive chemical substances.	Application
76770605	82380002	Design
FB-02YS(St+Ce)Y-fl	FB-02YS(St+Ce)YMSWAY-fl	Part. No.
1 x 2 x AWG 18/7	1 x 2 x AWG 18/7	Type description
orange	orange	Product size
		Colour

Figura 10.1: Detalhamento da estrutura interna de um cabo de comunicação FOUNDATION™ Fieldbus H1 (Leoni Group 2020).

A Figura 10.2 apresenta um *print* da página da General Cable com parte de uma tabela para escolha de cabos para diferentes padrões redes . A tabela apresenta os tipos de rede na primeira coluna e links para os cabos disponíveis na última coluna (a tabela completa pode se acessada no link da referência da figura).

A Figura 10.3 apresenta um exemplo de cabo disponível em catálogo (apenas parte da página), acessível a partir dos links da última coluna da tabela apresentada na Figura 10.2.

General Cable's Complete Protocols Offering Supports the Following:

Network Type / Protocol	Description	Applications	Cable Offering
Data Highway, DH-485	Local Area Network (LAN) designed for factory floor applications; based on EIA RS-485 standard	Light industrial environment, relatively clean, minimum temperature changes	GCR1314
Data Highway Plus, DH+	Baseband LAN used to connect a small number of nodes on a common link or with other industrial networks as part of a plant-wide Computer Integrated Manufacturing facility	Light industrial environment with variations including plenum, high flex, LSZH, Aluminum or Galvanized Steel Interlock Armor, CCW®, gel-filled	GCR1314 , GCR1300
RS-485	Serial communication methods for computers and devices; widely used communication interface in data acquisition and control applications where multiple nodes communicate	Light industrial environment, relatively clean, minimum temperature changes	C4841A , C4851A , C4842A , C4852A , C4843A , C4844A , C7112A , C7114A , C7116A , C7118A
HART® Protocol	HART® provides digital communication to microprocessor/microcomputer-based smart analog process control instruments	Light industrial environment, relatively clean, minimum temperature changes	See RS-485; additional cables C2534A , C8123 , C8101
ControlNet™	Real-time control network that provides high-speed transport of time-critical inputs, outputs and interlocking data; designed to accommodate high-level information and control needs of many sub-networks and controllers	Light industrial environment with variations including plenum, high flex, high braid coverage, Aluminum or Galvanized Steel Interlock Armor, CCW®	GCR1309
DeviceNet™	Digital, multi-drop network that connects and serves as a communication network between industrial controllers and input and output devices (I/O)	Light industrial environment with variations including high flex, CPE jacket for harsh environments, Class 1 600 V rated for cable tray use, Class 2 300 V for trunk or drop applications	GCR1305 , GCR1306 , GCR1312 , GCR1311 , GCR1313 , GCR1317 , GCR1307 , GCR1308
Foundation Fieldbus™	Digital, serial bi-directional communications protocol used in process control networks that standardizes the interconnection of sensors, actuators, and control devices to one another	Light industrial environment, relatively clean, minimum temperature changes, trunk and/or spur, CCW®	GCR1302 , GCR1303 , 9989.FB01801120 , 9989.FB01802120 , 9989.FB01804120 , 9989.FB01601118 , 9989.FB01602118 , 9989.FB01604118
Profibus	Achieved through a fieldbus; based on universal international standards and oriented to the Open System Interconnection reference model per ISO/IEC 7498	Achieved through a fieldbus; based on universal international standards and oriented to the Open System Interconnection reference model per ISO/IEC 7498	GCR1304 , GCR1302 , 9899.PB02201000
Interbus	Sensor/actuator bus system for process data to increase productivity of machines and manufacturing plants	Light industrial environment, relatively clean, minimum temperature changes	GCR1319 , GCR1318

Figura 10.2: Tabela para escolha de modelo de cabo a partir dos padrões de rede (General Cable - A Brand of Prysmian Group 2021a).

**Product Construction:****Conductor:**

- Fully annealed stranded tinned copper per ASTM B33

Insulation:

- Foam-Polyethylene
- Polypropylene
- FEP
- Foam - HDPE
- Polyolefin (GCR1301)

Jacket:

- PVC
- Flexguard
- FEP (C8101)

Shield:

- Flexfoil with tinned copper braid
- Flexfoil

Applications:

- Modbus applications
- HART® applications
- RS-485 applications
- Interlocked armored (steel or aluminum)
- Industrial type applications

Compliances:

- UL, c(UL)
- See below for individual product complia

Figura 10.3: Exemplo de cabo com a indicação de padrões de redes atendidos (applications) (General Cable - A Brand of Prysmian Group 2021b).

Capítulo 11

Cabeamento para redes ethernet

Para as ethernets industriais também é recomendável a escolha do cabeamento a partir da indicação dos fornecedores para cada protocolo.

Os conceitos de classe, categoria e classificação ambiental são importantes, pois são informações disponibilizadas em catálogos e podem influenciar na escolha do cabeamento mais adequado. Na sequência esses conceitos são apresentados de forma resumida.

11.1 Classes e categorias

Embora possam parecer conceitos semelhantes observa-se que são distintos, conforme as definições a seguir Phoenix Contact 2021 Weidmuller 2012 :

- **Classe** - Descreve todo o caminho de transmissão (depende dos diversos componentes que compõem o caminho).
- **Categoria** - É a classificação de cada componente.

A tabela da Figura 11.1 apresenta as características de classe/categoria. Por essa tabela observa-se que se um canal Classe E_A é dimensionado com componentes CAT6_A pode ser criada uma rede 10 Gigabit Ethernet, de acordo com as normas ISO/IEC 11801, AM1 and AM2 (Phoenix Contact 2021).

Na combinação de componentes de diferentes categorias a classe do caminho resultante é a do componente de pior classe, conforme exemplificado pela combinação das categorias de cabos e conectores na tabela da Figura 11.2 (Weidmuller 2012).

Channel class	Component category	Max. frequency	Speed	Application	Standard
Class D	CAT5	100 MHz	100 Mbps (4-pos.)	Fast Ethernet, PROFINET	IEC 11801:2002 EN 50173-1:2011
			1 Gbps (8-pos.)	Gigabit Ethernet	
Class E	CAT6	250 MHz	100 Mbps	Fast Ethernet	IEC 11801:2002
			1 Gbps	Gigabit Ethernet	EN 50173-1:2011
Class E _A	CAT6 _A	500 MHz	100 Mbps	Fast Ethernet	IEC 11801 AM1:2008
			1 Gbps	Gigabit Ethernet	IEC 11801 AM2:2010
			10 Gbps	10 Gigabit Ethernet	
Class F	CAT7	600 MHz	100 Mbps	Fast Ethernet	IEC 11801:2002
			1 Gbps	Gigabit Ethernet	EN 50173-1:2011
			10 Gbps	10 Gigabit Ethernet	
Class F _A	CAT7 _A	1000 MHz	100 Mbps	Fast Ethernet	IEC 11801 AM1:2008
			1 Gbps	Gigabit Ethernet	IEC 11801 AM2:2010
			10 Gbps	10 Gigabit Ethernet	
			40 Gbps	40 Gigabit Ethernet	EN 50173-1:2011

Figura 11.1: Características da classes/padrões para redes industriais (Phoenix Contact 2021).

Cable category	Connector category					
	Cat. 5	Cat. 5e	Cat. 6	Cat. 6 _A	Cat. 7	Cat. 7 _A
Cat. 5	Class D 1995	Class D 1995	Class D 1995	Class D 1995	Class D 1995	Class D 1995
Cat. 5e	Class D 1995	Class D 2002	Class D 2002	Class D 2002	Class D 2002	Class D 2002
Cat. 6	Class D 1995	Class D 2002	Class E	Class E	Class E	Class E
Cat. 6 _A	Class D 1995	Class D 2002	Class E	Class E _A	Class E _A	Class E _A
Cat. 7	Class D 1995	Class D 2002	Class E	Class E _A	Class F	Class F
Cat. 7 _A	Class D 1995	Class D 2002	Class E	Class E _A	Class F	Class F _A

Figura 11.2: Classe resultante conforme a categoria dos componentes. Observa-se que prevalece a menor classificação (Weidmuller 2012).

Não é o objetivo desta apostila entrar nos detalhes construtivos que caracterizam cada categoria de cabo. Uma descrição das características construtivas da Cat-1 até a Cat-8 pode ser encontrada no artigo “Ethernet Cable: Types, Performance and Pinout - Cat 5, 5e, 6, 6a, 7, 8” da página Electronicsnotes (Eletronicnotes 2020).

11.2 Classificação de ambiente (MICE)

MICE é um método de categorizar o ambiente que suporta três níveis chamados classificações: $M_1I_1C_1E_1$, $M_2I_2C_2E_2$ e $M_3I_3C_3E_3$.

As três classificações podem ser mapeadas para os seguintes níveis de severidade:

- 1 = Escritório,
- 2 = Industrial leve e
- 3 = Industrial.

Cada nível de gravidade crescente é mais severo. As áreas industriais podem ser generalizadas em quatro áreas típicas: chão de fábrica, área de trabalho, área de máquina e controle, equipamentos, sala de comunicações Rockwell Automation 2009. A Figura 11.3 apresenta um esquema com a relação entre o tipo de ambiente e os requisitos MICE.

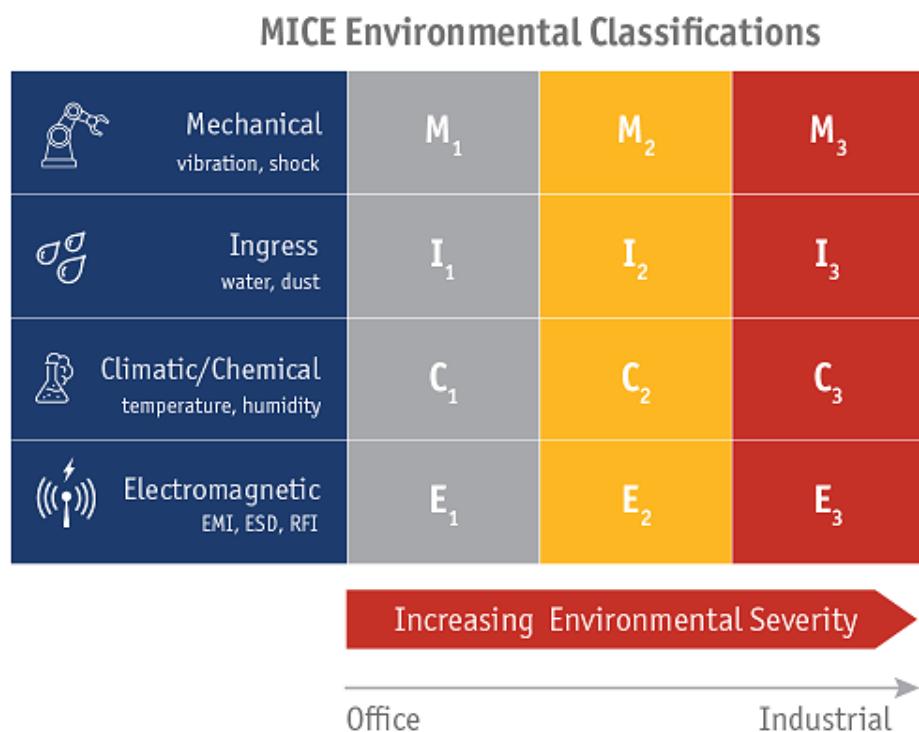


Figura 11.3: Requisitos MICE conforme o tipo de ambiente (Fluke Networks 2020).

Geralmente uma área não possui o mesmo nível para todas as categorias (Mecânica, Ingresso, Climática / Química e Eletromagnética). Por exemplo, uma máquina pode estar em uma área onde a vibração é muito alta, portanto, M₃, a área pode estar livre de poeira e líquidos, portanto, I₁, as temperaturas podem ser altas, portanto C₃, e os níveis eletromagnéticos são baixo, portanto, E₁ Rockwell Automation 2009. A Figura 11.4 apresenta algumas subcategorias de cada categoria M, I, C, E.

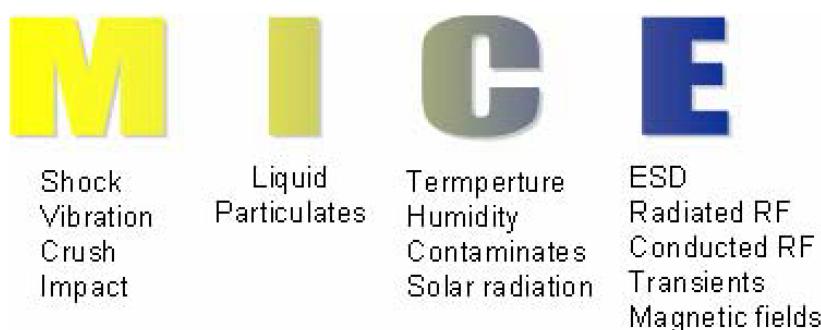


Figura 11.4: Subcategorias para M, I, C, E (Rockwell Automation 2009).

A tabela da Figura 11.5 mostra as classificações (níveis) típicas do MICE para cada área da fábrica.

Typical MICE Classification and Industrial Areas			
Area	Factory Floor	Work area	Machine Area
Typical Classification (MICE)	1&2	2	2&3

Figura 11.5: Classificação MICE típica para áreas industriais (Rockwell Automation 2009).

O detalhamento da classificação MICE foge do escopo desta apostila, mais detalhes e recomendações de leitura podem ser encontrados nos seguintes documentos:

- Rockwell Automation: Deploying a Fiber Optic Physical Infrastructure within a Converged Plantwide Ethernet Architecture Rockwell Automation 2009.
- Nexans: Industrial Cabling - Technical Paper Nexans 2005.
- Fromm: Considerations When Applying TIA-1005-A to Network Infrastructure Fromm 2020

Capítulo 12

Fibra ótica

A fibra óptica pode ser encontrada em muitas aplicações, de backbones de rede que alimentam a Internet para instalações de manufatura, para redes de comunicação submarinas em sondas de perfuração. A capacidade de transporte de informações de uma fibra óptica é muito maior do que para fio de cobre, coaxial cabos e links de microondas. As fibras ópticas são muito pequenas, leves, não corroem e são imune a ruídos elétricos de tempestades com raios e interferência eletromagnética (EMI / RFI). Além disso, os cabos de fibra óptica não transportam energia elétrica e são aprovados para locais perigosos. O custo do cabo de fibra óptica e seus associados conectores e hardware têm diminuído constantemente ao longo dos anos. Hoje, os benefícios da fibra óptica pode superar em muito os custos de produção, fazendo da fibra óptica a escolha preferida para industrial automação de fábricas e redes de controle de processos Weed Instrument 2003.

Várias abordagens podem ser usadas para configurar o canal. Por exemplo, um *patch cord* duplex (Figura 12.1) pode ser usado para conectar dois equipamentos que estão próximos um do outro.



Figura 12.1: *Patch cord* para conexão duplex por fibra ótica (Rockwell Automation 2018).

Atenção deve ser dada para alcançar a polarização correta das conexões, isto é, que a porta de transmissão (**TX**) de um dispositivo conecte para a porta de recepção (**RX**) do outro equipamento e vice-versa, conforme mostra a Figura 12.2. Esta polarização é realizada por construção de *patch cord* e codificação padronizada dos conectores.

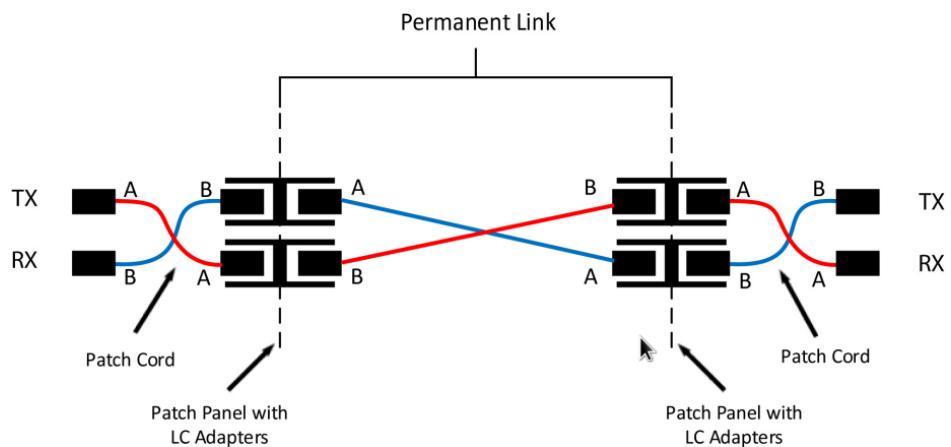


Figura 12.2: Diagrama de ligação de um canal duplex constituído por duas fibras óticas (Rockwell Automation 2018).

A Figura 12.3 apresenta um exemplo de arquitetura onde são utilizados links de fibra ótica (o catálogo com a descrição dos equipamentos utilizados pode ser acessado pelo link da referência).

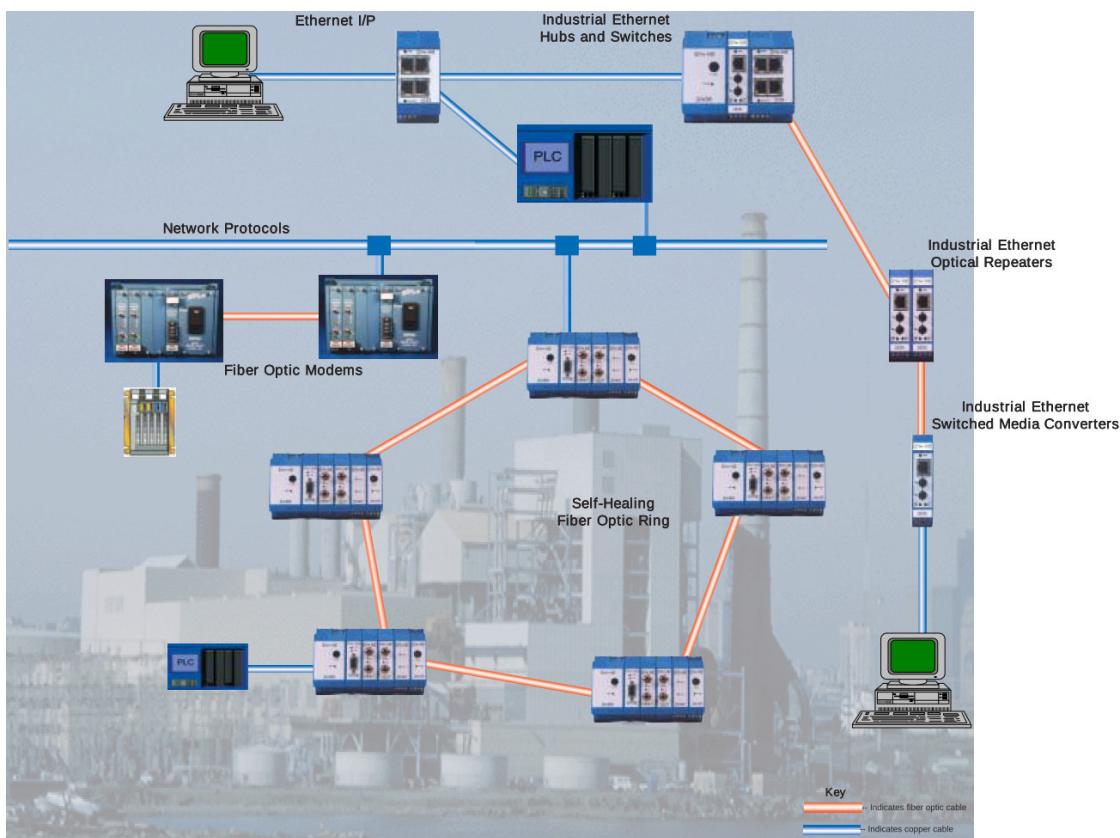


Figura 12.3: Exemplo de arquitetura com empredo de fibra ótica (Weed Instrument 2003).

12.1 Fibra monomodo e multimodo

Existem dois tipos diferentes de fibra usados na conexão de redes, **monomodo** e **multimodo**, que variam pelo tamanho do núcleo de vidro e pelo design e operação dos transceptores usados. A fibra **multimodo** é comumente usado para distâncias de transmissão mais curtas devido à eficiência de custo que oferece. Fibra **monomodo** usa mais precisão transceptores para alcançar distâncias de transmissão maiores e é mais cara para implementar do que a fibra multimodo. Os projetistas e instaladores precisam se certificar de que o tipo de transceptor e o tipo de fibra são compatíveis para otimizar desempenho.

A fibra óptica **multimodo** incorpora um diâmetro de núcleo maior do que o tipo de fibra **monomodo**. Além disso, o núcleo o diâmetro usado na fibra multimodo varia dependendo do tipo de desempenho usado. Por exemplo, OM1 (*optical multimode 1*) foi o primeiro tipo de fibra multimodo a ser implantado e usa um diâmetro de núcleo de $62,5 \mu\text{m}$ com um diâmetro de revestimento total de $125 \mu\text{m}$. Este tipo de fibra quase nunca é implantado em novas instalações, mas pode ser necessário para instalações legadas com uma base de equipamento instalada. Tipos de fibra multimodo OM2, OM3 e OM4 são baseados no uso de um diâmetro

de núcleo de $50 \mu\text{m}$ (novamente com um diâmetro de revestimento de $125 \mu\text{m}$) e oferecem melhor desempenho em termos de comprimento máximo do canal. OM4 é recomendado para novas instalações e representa o melhor compromisso disponível entre o custo total do link (fibra óptica mais transceptores) e comprimento do canal.

Os tipos de fibra **monomodo** usados em aplicações de redes de plantas têm um diâmetro de núcleo de $9 \mu\text{m}$ e um revestimento de $125 \mu\text{m}$ de diâmetro. Os transceptores usados com a fibra monomodo incorporam **fontes de laser mais caras** e portanto, o custo geral do link é maior do que a fibra multimodo; no entanto, comprimentos de canal maiores podem ser obtidos. Existem duas designações comuns para fibra ótica monomodo (OS, *optical singlemode*), identificadas como OS1 e OS2. OS1 é um tipo de fibra legado e não é recomendado para nenhuma implantação, embora possa ser visto em implantações legadas. OS2 é a designação de fibra monomodo padrão, embora a literatura comumente a chame OS1/OS2 Rockwell Automation 2018.

As designações para cada os tipos de fibra são apresentadas na tabela da Figura 12.4.

Designation	Core/Cladding Diameter	Fiber Type	100 Mbps Maximum Distance	1 Gbps Maximum Distance
OM1	62.5/125 μm	Multimode	2000m	220m
OM2	50/125 μm	Multimode	2000m	275m
OM3	50/125 μm	Multimode	>2000m	500m
OM4	50/125 μm	Multimode	>2000m	550m
OS2	9/125 μm	Singlemode	10km	10km

Figura 12.4: Características dos diferentes tipos de fibra ótica (Rockwell Automation 2018).

A Figura 12.5 mostra um diagrama esquemático das diferentes construções de fibra óptica.

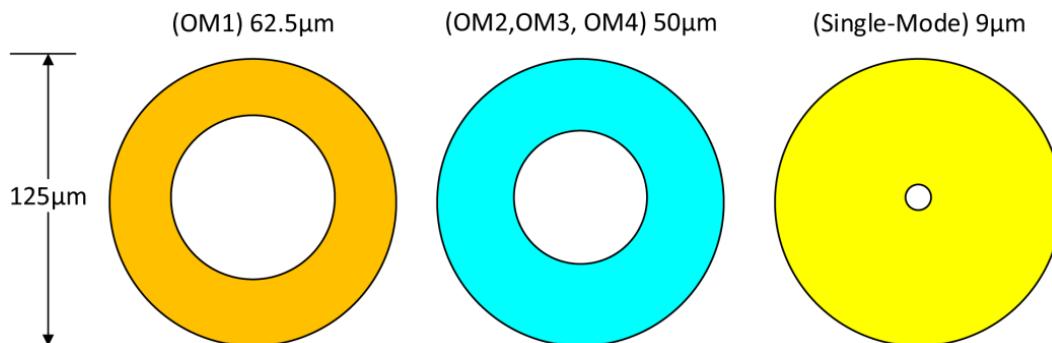


Figura 12.5: Comparação dos núcleos para as fibras ópticas multimodo (*optical multimode*) OM1-OM4 e monomodo (*optical singlemode*) OS2 (Rockwell Automation 2018).

12.2 Condições ambientais

O cabeamento de fibra ótica em ambientes industriais pode estar sujeito a agentes corrosivos, umidade, vibração e ruídos elétricos. Portanto, durante o design da camada física, os projetistas devem avaliar esses fatores ambientais em cada área onde a rede será distribuída. Semelhante ao cabeamento metálico, pode-se utilizar o padrão MICE para realizar a classificação das diferentes áreas, conforme apresentado na Seção 11.2 (Rockwell Automation 2018).

Capítulo 13

Wireless

Dentre os diversos padrões de transmissão sem fio existem aqueles mais genéricos, que podem vir a ser úteis em aplicações de automação, e os protocolos desenvolvidos esse enfoque. Nesta seção serão apresentados alguns desses protocolos.

Hoje no mercado vemos várias redes proprietárias e também algumas padronizadas. Existem muitos protocolos relacionados com as camadas superiores da tecnologia (ZigBee, WirelessHARTTM, ISA SP100) e o protocolo IEEE 802.15.4 (2006) para as camadas inferiores. O protocolo IEEE 802.15.4 define as características da camada física e do controle de acesso ao meio para as LR-WPAN (Low-Rate Wireless Personal Area Network) Smar Technology Company 2020b. A Figura 13.1 apresenta uma projeção para 2012 (sugestão de atividade: pesquise informações atualizadas) do mercado das redes wireless baseadas no padrão IEEE 802.15.4 Smar Technology Company 2020b.

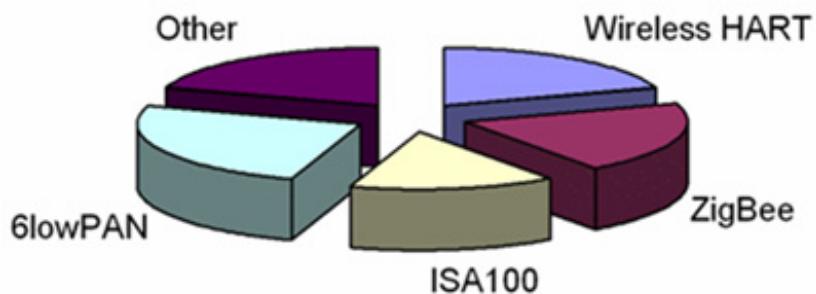


Figura 13.1: Projeção para 2012 do Market Share para as redes baseadas no padrão IEEE 802.15.4 Smar Technology Company 2020b.

13.0.1 IEEE 802.11 (Wi-Fi)

Provavelmente o padrão wireless mais popular, devido ao seu uso em redes de computadores. Embora não seja um padrão voltado para área industrial, pode vir a ser útil quando usado com protocolos que funcionam na camada de aplicação do modelo OSI (acima da camada de transporte TCP).

São exemplos de protocolos que atuam sobre o padrão TCP o **Modbus TCP** e o **OPC UA**. A combinação destes padrões com microcontroladores que tenham o padrão IEEE 802.11 embarcado, como o ESP32, pode ser explorada para transmissão de variáveis entre microcontroladores ou para implementação de sistemas supervisórios executando em computadores, por exemplo.

Notas sobre o microcontrolador ESP32: Já consegui fazer a comunicação entre dois ESP32 e entre um ESP32 e um supervisório Elipse E3 por meio do protocolo OPC UA com uso da biblioteca open62541 compilada para arquitetura RTOS.

Com relação ao Modbus TCP, a ferramenta de desenvolvimento do ESP32, ESP-IDF, traz na sua pasta de exemplos uma implementação do Modbus. A versão antiga, que analisei, ainda não tinha a implementação do Modbus TCP (trazia apenas o Modbus RTU). Aparentemente a nova versão traz um implementação do Modbus TCP, que poderia ser usado com Wi-Fi, mas ainda não testei.

13.0.2 Bluetooth e Bluetooth Low Energy (BLE)

O Bluetooth é um padrão wireless de curto alcance, bastante popular para transmissão de áudio e arquivos entre telefones celulares e computadores.

O **Bluetooth Low Energy** chegou ao mercado em 2011 como Bluetooth 4.0. Ao falar sobre Bluetooth Low Energy vs. Bluetooth, a principal diferença está no baixo consumo de energia do Bluetooth 4.0. Embora isso possa soar como algo negativo, na verdade é extremamente positivo quando falamos sobre comunicação M2M (*machine to machine*). Com o consumo de energia do Bluetooth LE, os aplicativos podem funcionar com uma pequena bateria por quatro a cinco anos. Embora não seja ideal para falar ao telefone, é vital para aplicativos que precisam apenas trocar pequenas quantidades de dados periodicamente Link Labs 2015.

São considerados protocolos adequados para aplicações de Internet das Coisas (IoT - *Internet of Things*). Também são padrões disponibilizados pelo microcontrolador ESP32, o

que possibilita a criação de aplicações de baixo custo com essa tecnologia de comunicação sem fio.

13.0.3 ZigBee

Semelhante ao Bluetooth, é um protocolo wireless de curto alcance. Apesar de não ser um padrão tão popular quanto o Bluetooth, o ZigBee é bastante conhecido na área de IoT. O ZigBee se baseia no padrão IEEE802.15.4. Existem módulos de transmissão ZigBee, de baixo custo, disponíveis para uso com microcontroladores, ou placas baseadas em microcontroladores, como o Arduino e as placas com ESP32.

13.0.4 WirelessHART

O WirelessHART busca contornar características indesejadas dos dois padrões wireless mais populares, o Wi-Fi e o Bluetooth. O padrão Wi-Fi apresenta grande capacidade de transmissão de dados, porém não é otimizado para economia de energia, o que torna esse padrão inadequado para sensores alimentados por bateria. Já o padrão Bluetooth apresenta menor taxa de transmissão e maior otimização para redução do consumo.

Embora o padrão Bluetooth possua taxas de transmissão adequadas para uso em sensores, a sua limitação de potência em 10mW, ou 100mW em alguns países, torna esse protocolo inadequado para a área industrial. Apesar dessa potência permitir um alcance de transmissão entre 100 e 200m, o que atenderia a maior parte das aplicações, as condições ambientais das indústrias podem reduzir muito essa distância Pepperl+Fuchs 2011.

Para atender às características desejadas de transmissão em ambiente industrial, com otimização para transmissão de pequenas quantidades de informação e economia de energia o WirelessHART adotou como base o padrão IEEE802.15.4, que é o mesmo padrão adotado pelo ZigBee.

O WirelessHART adota o conceito de “Flat Mesh Network”, onde todos os sensores de campo formam uma rede, por onde as mensagens são roteadas. Nessa rede o nó de origem envia a mensagem para o vizinho mais próximo, que encaminha a mensagem para o próximo vizinho, esse processo se repete até a mensagem chegar ao destinatário. Esse estrutura de rede mesh permite que a rede se estenda por uma longa distância, o limite de 100 a 200m passa a valer somente para nós que se comunicuem diretamente. Outra vantagem da arquitetura mesh é a possibilidade de alojar caminhos alternativos em caso de falha do

caminho em uso (remoção ou defeito de algum equipamento). A Figura 13.2 apresenta um esquema da arquitetura mesh usada pelo WirelessHART Pepperl+Fuchs 2011.

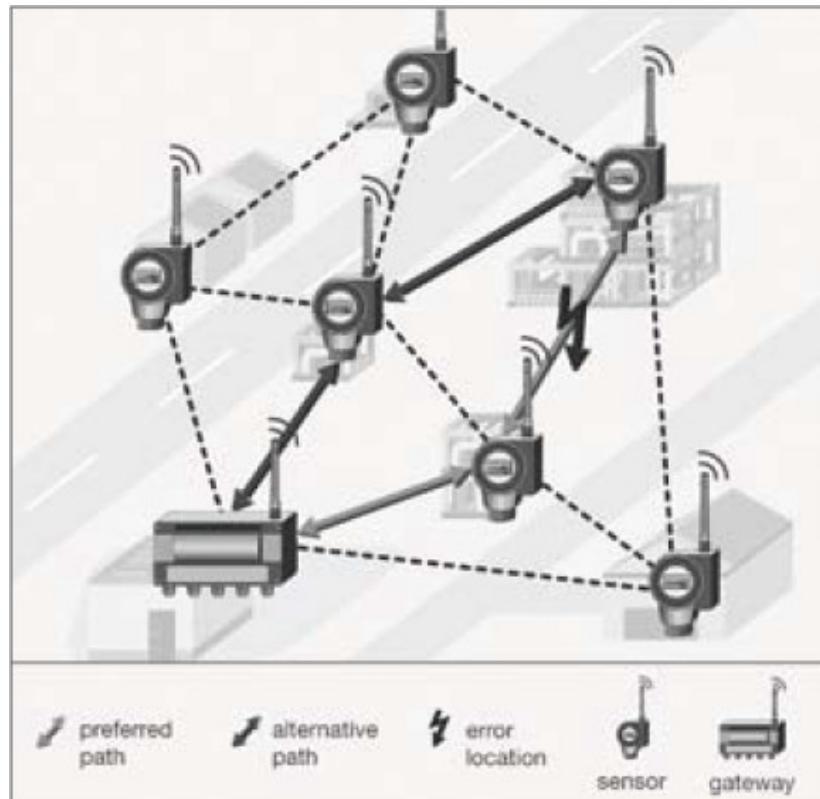


Figura 13.2: Rede Mesh Pepperl+Fuchs 2011.

A Figura 13.3 mostra os três dispositivos envolvidos na implementação de uma rede WirelessHART. A Figura 13.4 apresenta a topologia da rede WirelessHART com algumas de suas características.

There are 3 basic WirelessHART device types defined in the HART 7 standard:



Figura 13.3: Dispositivos WirelessHART Phoenix Contact 2011.

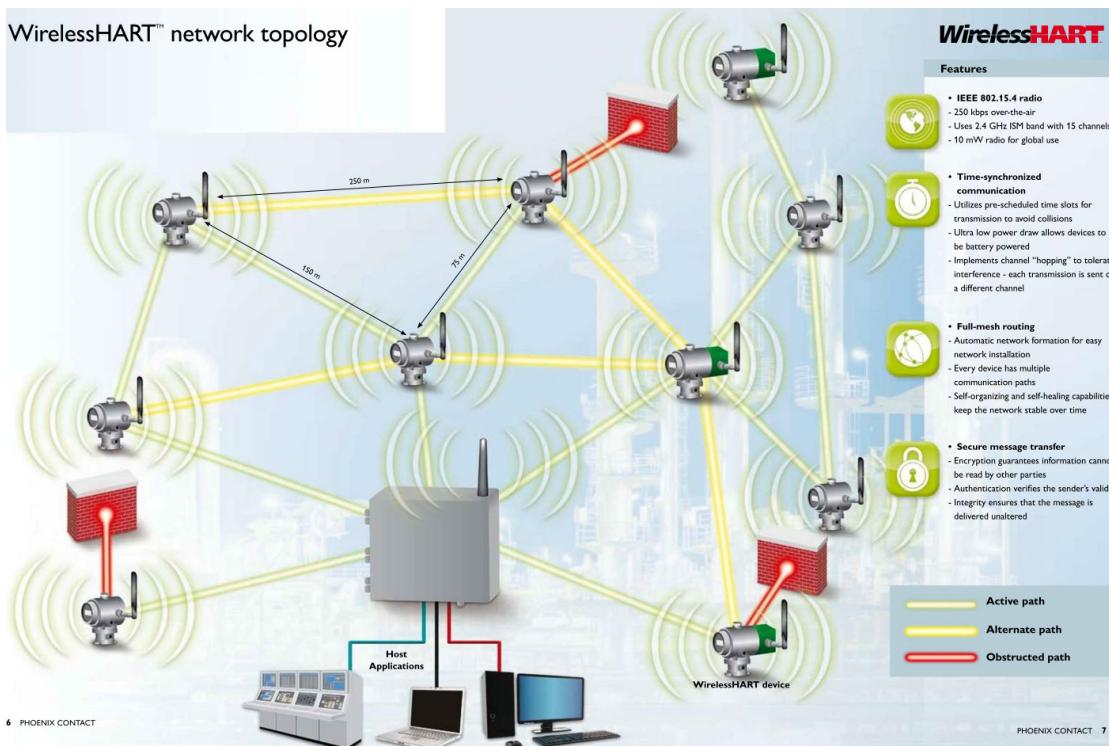


Figura 13.4: Topologia de rede WirelessHART Phoenix Contact 2011.

13.0.5 ISA 100.11a (ISA100)

Os padrões WirelessHART e ISA 100.11a apresentam muitas semelhanças. Eles são projetados para atender ao mesmo mercado. No nível de aplicação, eles desempenham a mesma função e apresentam os mesmos benefícios. Ambos ISA 100.11a e WirelessHART implementam o padrão de rádio IEEE 802.15.4 como camada física. Ambos os protocolos usam DDL e Arquivos de descrição do dispositivo. Ambos podem eliminar muito do hardware de I/O dos PLCs, cabeamento e acessórios associados AutomationWorld 2013.

A principal diferença entre os dois protocolos está na especificação da camada de aplicação. O WirelessHART especifica HART como a camada de aplicação enquanto ISA100.11a deixa essa camada indefinida. Isso significa que os dados na camada de aplicação do padrão ISA100.11a podem ser transferidos usando Foundation Fieldbus, Profibus, Modbus, HART ou outros protocolos. Embora isso torne ISA100.11a altamente flexível, o cliente deve decidir qual protocolo a ser usado. A decisão do WirelessHART de especificar apenas HART na camada de aplicação foi feita para entregar simplicidade por meio do uso de uma única especificação de comunicação de dados através da rede, o que significa que a comunicação de dados na rede é bem definida e compreendida AutomationWorld 2013.

ISA100.11a define a pilha de protocolo, gerenciamento de sistema e funções de segurança para uso sobre redes sem fio de baixa potência e baixa taxa de transmissão (atualmente IEEE 802.15.4). ISA100.11a não especifica um protocolo de automação na camada de aplicação ou uma interface para um protocolo existente. Ele somente especifica ferramentas para construção de uma interface Nixon 2012. A arquitetura da rede ISA100.11a é mostrada na Figura 13.5 e os elementos de uma rede ISA100 são descritos na Figura 13.6.

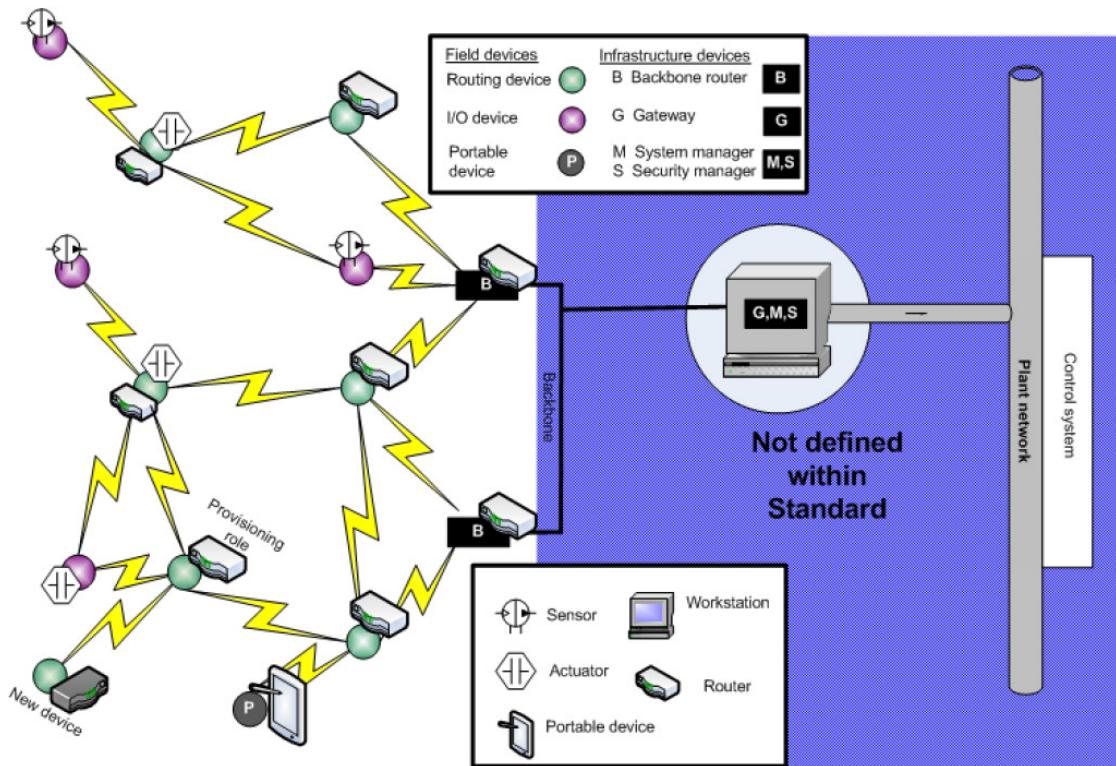


Figura 13.5: Arquitetura da rede ISA100.11a Nixon 2012.

Note: Devices operating in the field typically incorporate multiple logical roles.

Role	Role Definition and Responsibilities
Input/Output	Sources or consumes data. Does not route.
Router	Routes messages for other devices operating in the wireless subnet.
Backbone Router	Routes data via the backbone. Mitigates between devices operating in the wireless subnet and devices operating on the backbone.
System Manager	The "brains" of the network. Manages all network devices through policy-controlled configurations based on collection of performance parameters reported.
Security Manager	Enables, controls and supervises the secure operation of all devices present in the network.
Gateway	Provides an application interface between the wireless network and the plant network.
Provisioning	Provisions devices with configurations required for operation within the network.
System Time Source	Responsible for maintaining the master time source of the network.

Figura 13.6: Dispositivos de uma rede ISA100 ISA100 Wireless Compliance Institute 2020.

As camadas de rede e transporte são baseadas nos padrões 6LoWPAN, IPv6 e UDP. A camada de enlace de dados ISA100.11a é exclusiva para o ISA100.11a e usa uma forma não compatível de padrão MAC IEEE802.15.4. A camada de enlace de dados implementa roteamento por arquitetura mesh grid, salto de frequência e recursos de acesso múltiplo por *slots* de tempo de tempo. O encaminhamento de mensagens dentro da rede sem fio é realizada na camada de enlace de dados, em uma arquitetura mesh grid Nixon 2012.

A Figura 13.7 apresenta a pilha de protocolos do padrão ISA100.11a. 6LoWPAN (acrônimo para IPv6 over Low power Wireless Personal Area Networks) é o grupo de desenvolvimento da IETF, que cria e mantém as especificações que nos permitem usar IPv6 nas redes IEEE 802.15.4.

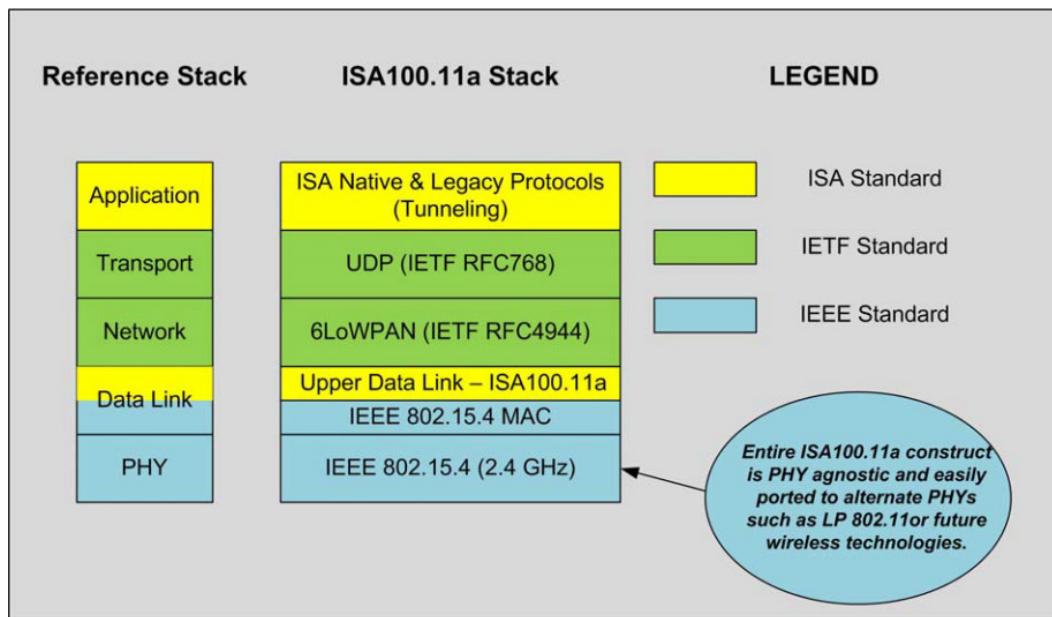
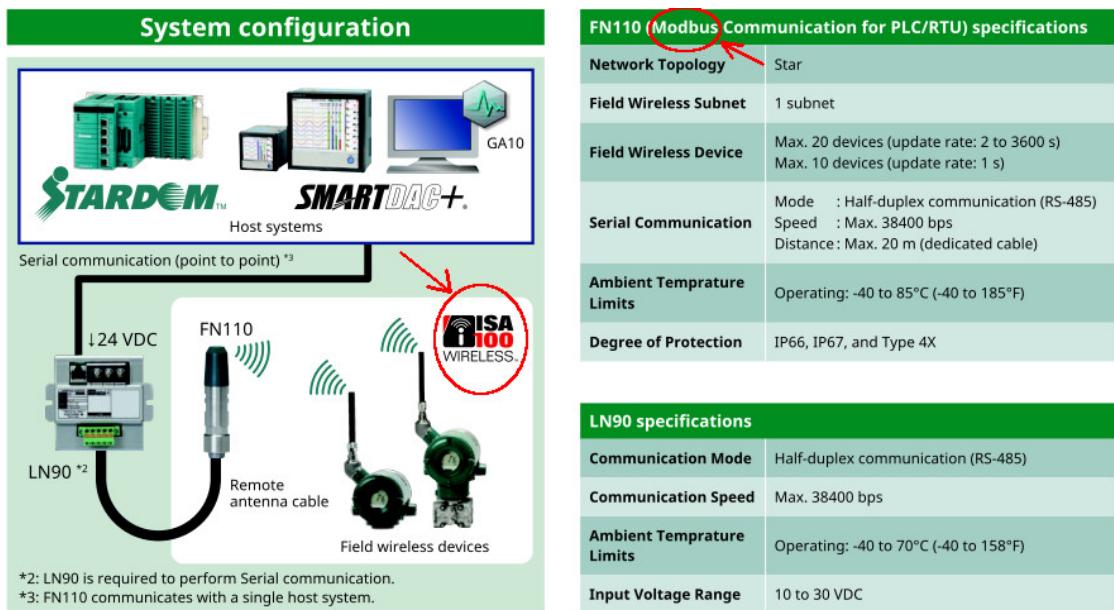


Figura 13.7: Pilha de protocolos ISA100.11a ISA100 Wireless Compliance Institute 2020.

A Figura 13.8 apresenta um exemplo de dispositivo comercializado pela Yokogawa para a implementação de rede ISA100 com dispositivos Modbus.



Application example

Adding an isolated repeater between a host system and LN90 extends serial communication distance up to 1.2 km (4000 ft)^{*4}.

*4: Result obtained with GX10/20 as a host system and MOXA TCC-120/120i as an isolated repeater.

Your result may vary depending on hardware and environmental factors

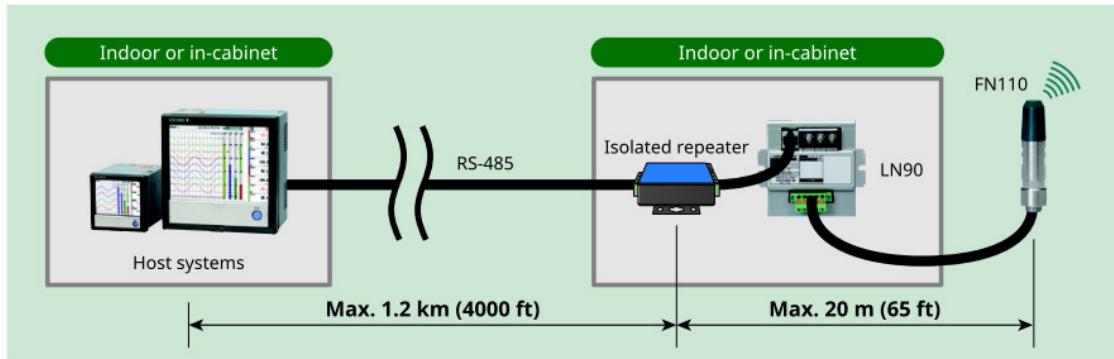


Figura 13.8: Exemplo de dispositivos que implementam a comunicação pelo padrão ISA100 Yokogawa 2018.

A Figura 13.9 apresenta outra configuração para comunicação por ISA100 de equipamentos HART ou Modbus. Essa configuração utiliza o gateway FN110, que implementa a comunicação por ISA100, como visto na Figura 13.8.

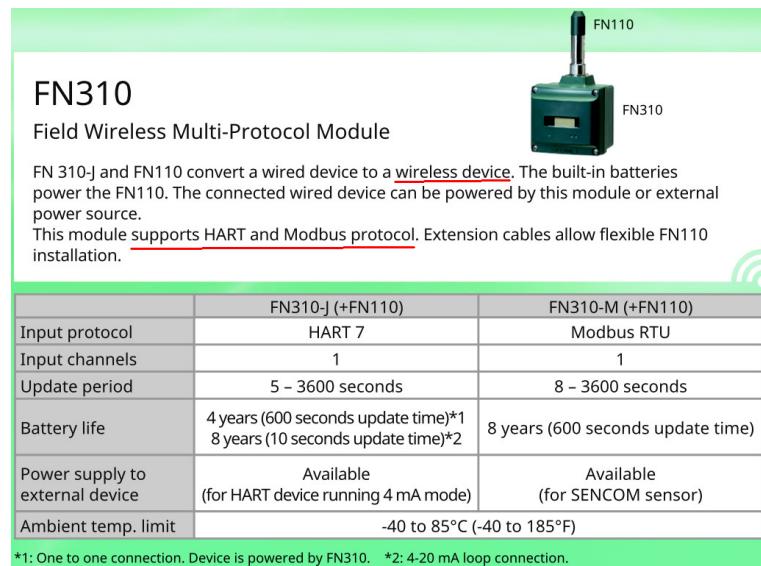


Figura 13.9: Exemplo de equipamento para comunicação por ISA100 de dispositivos HART ou Modbus Yokogawa 2017.

13.1 Considerações finais sobre os meios de transmissão

O objetivo deste capítulo foi apresentar os principais aspectos e tecnologias envolvidos na escolha do meio de transmissão. Como exemplo final, a Figura 13.10 apresenta uma estrutura onde os três meios de transmissão são combinados (cabo metálico, fibra ótica e wireless).

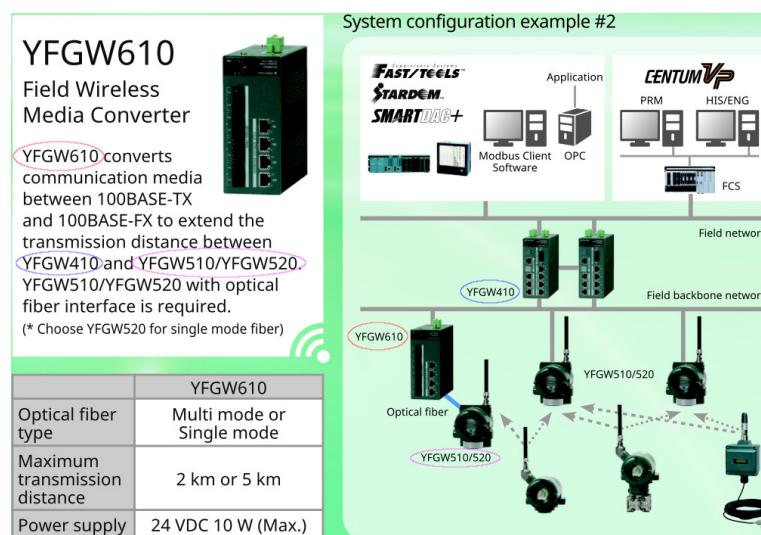


Figura 13.10: Exemplo de aplicação dos três meios de comunicação (cabo metálico, fibra ótica e wireless) Yokogawa 2017.

Capítulo 14

Distribuição do espectro de radiofrequência e Bandas ISM

Este capítulo apresenta a distribuição do espectro de frequência no Brasil e o conceito de Bandas ISM.

14.1 Regulamentação do espectro de frequência no Brasil

O espectro de frequência é gerenciado pelo Governo.

O link a seguir aponta para a RESOLUÇÃO Nº 716, DE 31 DE OUTUBRO DE 2019, que aprova o Plano de Atribuição, Destinação e Distribuição de Faixas de Frequências no Brasil (PDFF).

https://sei.anatel.gov.br/sei/modulos/pesquisa/md_pesq_documento_consulta_externa.php?eEP-wqk1skrd8hS1k5Z3rN4EVg9uLJqrLYJw_9INc07xT8h5b1HTFJoMNTJSQiAQ SJFIG919KSQ08WmdOHlwUrYp2e_GJegapZEt1C19ygA2Yn6freTFTggpi9bcPHxi

O endereço a seguir apresenta um mapa com a distribuição do espectro de radiofrequência no Brasil, revisão 2019:

<https://sistemas.anatel.gov.br/anexar-api/publico/anexos/download/df7afdda635a8941488379eb1b4638fe>

O link a seguir apresenta o Plano de Atribuição, Destinação e Distribuição de Frequência no Brasil, edição 2020:

<https://sistemas.anatel.gov.br/anexar-api/publico/anexos/download/db36871563204c812e300856bd9b2794>

O link a seguir apresenta o Plano de Uso do Espectro de Radiofrequência no Brasil - Plano de Uso do Espectro de Radiofrequências para o período de 2021 a 2028, Atualização fevereiro 2021:

<https://sistemas.anatel.gov.br/anexar-api/publico/anexos/download/d402b717998a27a99764e1efb875e74c>

14.2 Bandas de frequência ISM

As faixas de frequência ISM (Industrial Sientific and Medical) são bandas reservadas internacionalmente para o desenvolvimento Industrial, científico e médico. Em 1985 o FCC (*Federal Communications Commission*), que é um órgão regulador da área de telecomunicações e radiodifusão fundado em 1934 nos Estados Unidos, desvencilhou parte do espectro de frequência para desenvolvimentos livres, sem a necessidade de licenciamento de utilização de frequência, e introduziu normas para limitação de potência de transmissão e técnicas de modulação dentro destas faixas (Teleco 2022). A Figura 14.1 apresenta as bandas ISM disponíveis no Brasil.

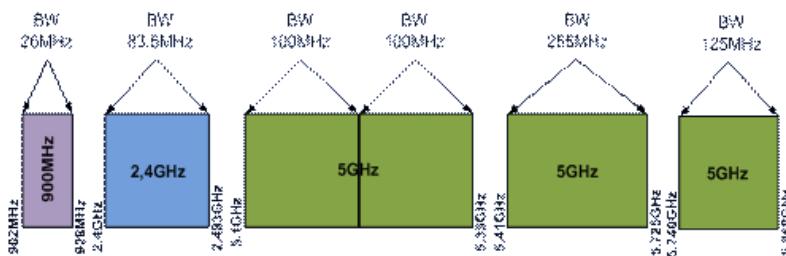


Figura 14.1: Bandas ISM disponíveis no Brasil (Teleco 2022).

Este padrão foi internacionalmente difundido e adotado em diversos países, e adotado no Brasil com algumas ressalvas. No Brasil a legislação para este tipo de sistema foi inicialmente definida pela ANATEL, através da Norma 02/93, posteriormente pela Norma 012/96

(resolução 209 de Jan/2000) e atualmente pela resolução 305 de Jul/2002 – Regulamento sobre Equipamentos de Radiocomunicação de Radiação Restrita (Teleco 2022).

Os serviços de Radiocomunicação operando nestas faixas de frequência devem aceitar a interferência prejudicial que possa resultar de dispositivos operando nesta mesma faixa de frequência. A confiabilidade das comunicações de dados não pode ser assegurada com técnicas simples, porque não existem restrições ao número de transmissores, nem existem protocolos definidos. Na banda ISM os dispositivos tem que compartilhar o espaço com outros serviços, e a existência de outras fontes potenciais de interferência podem ser inevitáveis (Teleco 2022).

Parte III

Considerações práticas sobre portas seriais (portas COM)

Capítulo 15

Introdução

Ao lado das conexões por TCP/IP, em conexões Ethernet ou Wi-Fi, as portas seriais são uma das formas de conexão mais comuns dos computadores.

Nesse capítulo pode ser adotado o termo "Porta COM" como sinônimo de comunicação serial. Onde não for feita referência ao sistema operacional Windows, pode-se considerar que existe um equivalente no Linux, como a porta /dev/ttyUSB0 (dispositivo serial conectado à porta USB) ou /dev/ttyS0, por exemplo.

Podemos ter, por exemplo a comunicação de um computador com os seguintes equipamentos por meio de uma porta serial:

- CLP,
- Placa baseada em microcontrolador (Arduino ou ESP32, por exemplo),
- Sensor ou atuador em barramento RS-485 acessado com adaptador USB/RS-485.

A comunicação serial pode ser útil, por exemplo, para:

- Programação do dispositivo,
- Acesso por sistema supervisório (Elipse E3 ou ScadaBR, por exemplo),
- Acesso a sensores para automação com computador (PC ou Raspberry Pi, por exemplo).

Capítulo 16

Acesso a portas seriais (COM)

Esse capítulo aborda questões sobre o acesso às portas seriais (portas COM). sec

16.1 Introdução

Ao acessar um dispositivo físico, normalmente por meio de uma conexão USB, é criada uma porta COM para esse dispositivo. No Windows essa porta pode ser vista no Gerenciador de Dispositivos. Também pode-se descobrir o nome da porta COM de um dispositivo conectado utilizando uma ferramenta de acesso a portas seriais, como o HTerm, executando o comando de *refresh* após conectar o dispositivo (funciona para Windows e Linux).

O acesso às portas seriais pode ser feito, por exemplo, pelos seguintes programas:

- Um programador de microcontrolador ou placa baseada em microcontrolador (Arduino e ESP32, por exemplo),
- Um programador de CLP.
- Uma ferramenta genérica de acesso à porta serial,
- Um programa próprio de alguma rede industrial, como o mestre Modbus QModMaster.

16.2 Programas genéricos de acesso às portas seriais

Existe uma grande quantidade de terminais para escrita e leitura em portas seriais. Cada um desses programas apresenta pontos fortes e fracos. A seguir são apresentados alguns desses programas, destacando-se algumas características importantes.

Embora alguns programas sejam mais interessantes em determinado momento, é bom avaliar as alternativas, pois podem existir funções disponíveis apenas em um dos programas.

- **HTerm:** (*Windows e Linux*)

Página: <https://www.der-hammer.info/pages/terminal.html>

Até agora aparenta ser o melhor programa. Boa interface, listou as portas virtuais criadas no Linux (o que o CuteCom e o Moserial não fizeram). Também apresenta como vantagem a disponibilidade de uma versão executável para download (para o Windows e para o Linux).

- **CuteCom:** (*Linux*)

Página: No Debian pode ser instalado por repositório (não instalei por nenhum site de download).

Apresenta uma boa interface, porém não listou as portas virtuais criadas no Linux e não permite que o endereço da porta seja digitado.

- **Moserial:** (*Linux*)

Página: <https://gitlab.gnome.org/GNOME/moserial/>

É uma opção do Linux que aparenta ter uma interface razoável. Como desvantagem precisa ser compilado para uso (não existe no repositório do Debian). Assim como o CuteCom, não listou as portas seriais virtuais.

- **RealTerm:** (*Windows*)

Página: <https://sourceforge.net/projects/realterm/>

Apresenta muitos recursos e funciona bem, porém a interface é bem confusa, com uma quantidade exagerada de abas.

- **Terminal by Br@y++:** (*Windows*)

Página: <https://sites.google.com/site/terminalbpp/>.

Apresenta uma boa interface (não tão boa quanto a do HTerm). Um item inconveniente é a forma adotada para entrar números hexadecimais, que é \$xx (por exemplo \$A9 para o hexadecimal 0xA9), isso é bem ruim para compor uma mensagem Modbus RTU, por exemplo. Existem programas, como HTerm, que ativam a numeração hexadecimal por um botão.

Capítulo 17

Portas seriais (COM) virtuais

Quando um programa deve acessar um dispositivo físico, deve-se configurar o programa com o endereço da porta serial criada quando o dispositivo é conectado ao computador (normalmente por uma entrada USB). Um exemplo onde vemos essa situação é o acesso a um Arduino (placa real conectada ao computador por uma entrada USB) por um sistema supervisório criado no Elipse E3 (usando o padrão Modbus).

Em uma situação onde deseja-se acessar um Arduino virtual, simulado no SimulIDE, por meio de um supervisório criado no Elipse E3 (usando protocolo Modbus), tanto o Arduino simulado quanto o Elipse E3 conseguem acessar portas seriais (portas COM). Porém, cada porta COM pode ser acessada por um único programa. Ou seja, não é permitido que o Arduino simulado e o Elipse E3 acessem a mesma porta COM.

A solução para esse problema é a criação de portas seriais virtuais. Com essa solução um programa (**com0com** no Windows e **socat** no Linux) cria um par de portas COM que se comunicam, por exemplo COM3 e COM4. Então pode-se configurar cada um dos programas para acessar uma das portas virtuais criadas, por exemplo o Arduino Virtual com a COM3 e o supervisório do Elipse E3 com a COM4. Esse esquema de ligação de duas aplicações (programas) por portas seriais virtuais é apresentada na Figura 17.1. Dessa forma, o supervisório criado no Elipse E3 pode acessar o Arduino simulado pelo SimulIDE.

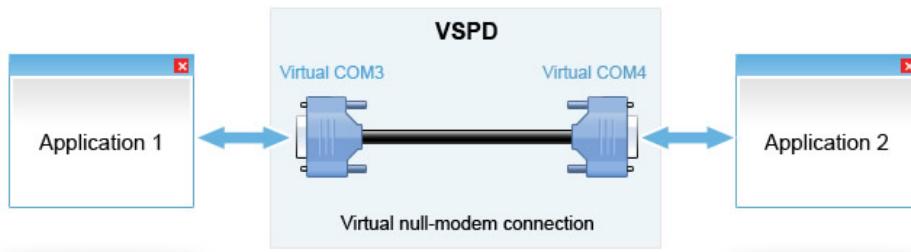


Figura 17.1: Estrutura de comunicação entre duas aplicações por meio de um par de portas seriais virtuais (criadas com o **com0com** no Windows ou com o **socat** no Linux) SourceForge (v frolov) 2018.

17.1 Portas seriais virtuais no Linux: Programa Socat

Esta seção aborda a instalação e uso do programa "socat".

17.1.1 Instalação

O socat está disponível no repositório do Debian, portanto é essa a forma de instalação apresentada aqui. Para quem utiliza outra distribuição é recomendável verificar a disponibilidade no repositório. Para o Debian (e Ubuntu) o comando de instalação é:

```
sudo apt-get install socat
```

17.1.2 Execução

A execução do socat pode ser com o comando:

```
socat -d -d pty,raw,echo=0 pty,raw,echo=0
```

Esse comando de execução foi obtido em um fórum da internet. Apenas usei o comando, sem analisar os detalhes.

A Figura 17.2 apresenta a janela com a execução deste comando. A figura destaca os endereços atribuídos às portas virtuais criadas.

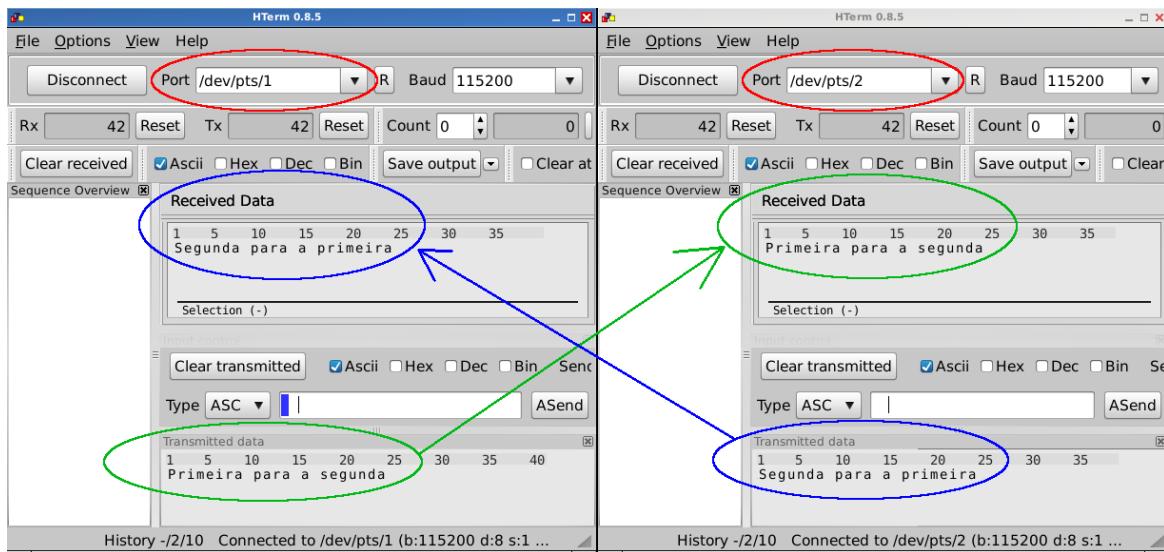


Figura 17.3: Teste de comunicação com as portas seriais virtuais criadas pelo comando “socat”.

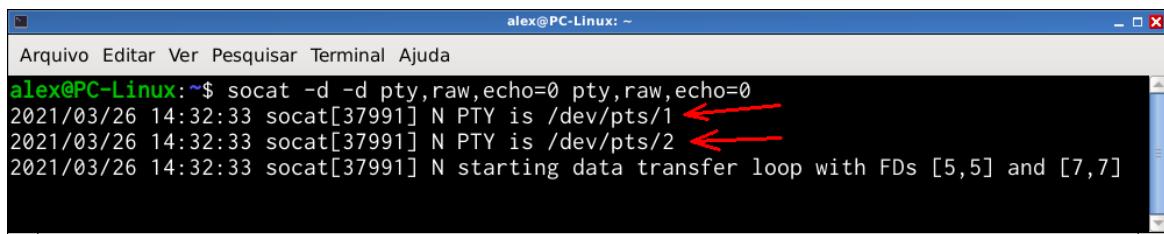


Figura 17.2: Execução e saída do comando “socat”.

Para desativar as portas criadas pelo socat, utiliza-se a combinação de teclas (padrão do Linux para desativar a maioria dos programas executados em linha de comando):

<ctrl> <c>

17.1.3 Teste de funcionamento

Antes de partir para o uso das portas virtuais é recomendável realizar o teste funcionamento. Para esse teste é possível usar duas janelas de um programa de acesso à portas seriais, como o HTerm, configuradas com os endereços das portas virtuais, verificando se a mensagem enviada por uma das janelas é recebida na outra. A Figura 17.3 apresenta esse teste (com as portas indicadas na Figura 17.2).

17.1.4 Exemplo de uso das portas seriais virtuais no Linux

Esta seção apresenta um exemplo de uso de portas seriais virtuais para comunicação entre um Arduino simulado com o SimulIDE, atuando como server Modbus (escravo), com o QModMaster.

O código executado apresentado abaixo foi obtido da página da biblioteca ‘ArduinoModbus’, e pode ser encontrado no seguinte endereço:

<https://github.com/arduino-libraries/ArduinoModbus/tree/master/examples/RTU/ModbusRTUServerLED>

```
1 /*
2 Modbus RTU Server LED
3 This sketch creates a Modbus RTU Server with a simulated coil.
4 The value of the simulated coil is set on the LED
5 Circuit:
6 - MKR board
7 - MKR 485 shield
8 - ISO GND connected to GND of the Modbus RTU server
9 - Y connected to A/Y of the Modbus RTU client
10 - Z connected to B/Z of the Modbus RTU client
11 - Jumper positions
12 - FULL set to OFF
13 - Z \/\ Y set to OFF
14 created 16 July 2018
15 by Sandeep Mistry
16 */
17
18 #include <ArduinoRS485.h> // ArduinoModbus depends on the ArduinoRS485 library
19 #include <ArduinoModbus.h>
20
21 const int ledPin = LED_BUILTIN;
22
23 void setup() {
24     Serial.begin(9600);
25
26     Serial.println("Modbus RTU Server LED");
27
28     // start the Modbus RTU server, with (slave) id 1
29     if (!ModbusRTUServer.begin(1, 9600)) {
30         Serial.println("Failed to start Modbus RTU Server!");
31         while (1);
32     }
33
34     // configure the LED
```

```

35     pinMode(ledPin, OUTPUT);
36     digitalWrite(ledPin, LOW);
37
38     // configure a single coil at address 0x00
39     ModbusRTUServer.configureCoils(0x00, 1);
40 }
41
42 void loop() {
43     // poll for Modbus RTU requests
44     ModbusRTUServer.poll();
45
46     // read the current value of the coil
47     int coilValue = ModbusRTUServer.coilRead(0x00);
48
49     if (coilValue) {
50         // coil value set, turn LED on
51         digitalWrite(ledPin, HIGH);
52     } else {
53         // coil value clear, turn LED off
54         digitalWrite(ledPin, LOW);
55     }
56 }
```

A Figura 17.4 apresenta a configuração da porta serial do Arduino, também apresenta o led acesso pelo comando enviado através do QModMaster.

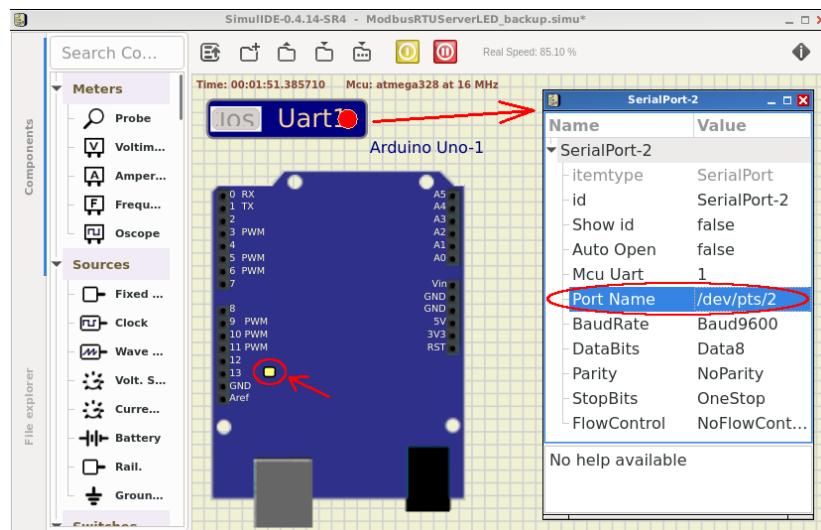


Figura 17.4: Configuração da porta serial de um Arduino simulado pelo SimulIDE.

A Figura 17.5 apresenta a configuração da porta serial no QModMaster para acesso ao Arduino virtual. A figura apresenta o QModMaster já conectado e acessando o Arduino.

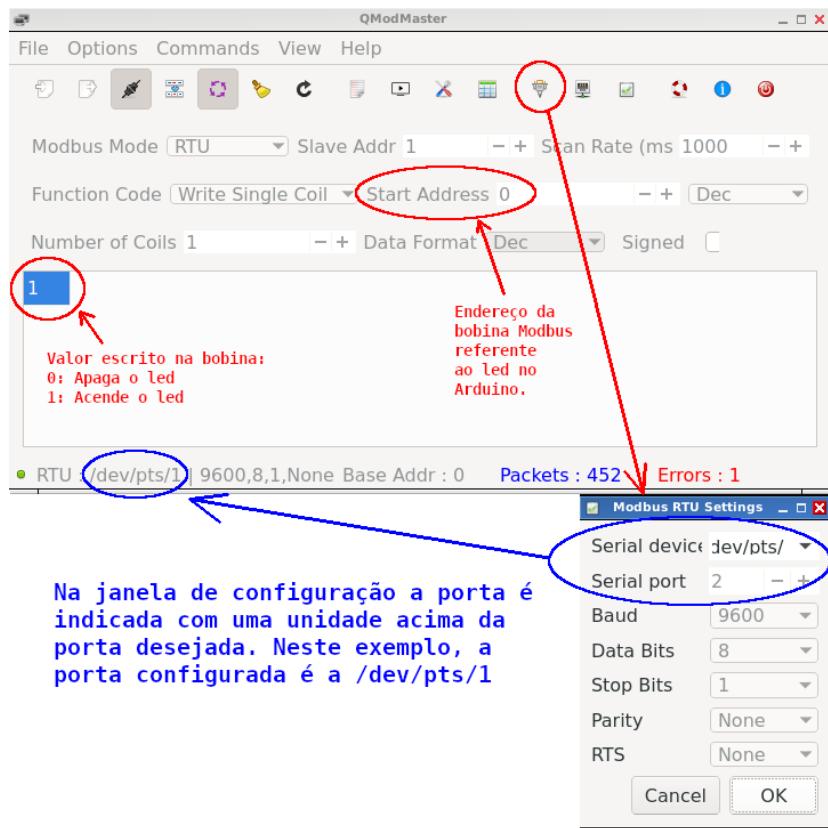


Figura 17.5: Configuração da porta serial virtual para acesso ao Arduino virtual simulado com o SimulIDE.

17.2 Portas seriais virtuais no Windows: Programa com0com

Esta seção apresenta o programa “com0com”, que tem a função de gerar portas seriais (portas COM) virtuais.

17.2.1 Instalação

O programa “com0com” pode ser obtido pela página do Sourceforge (na página de download o programa consta como “Null-modem emulator”)

<https://sourceforge.net/projects/com0com/>

A instalação segue o padrão do Windows, com a execução do instalador baixado do Sourceforge.

17.2.2 Execução do programa

A Figura 17.6 apresenta o ícone de início do com0com.

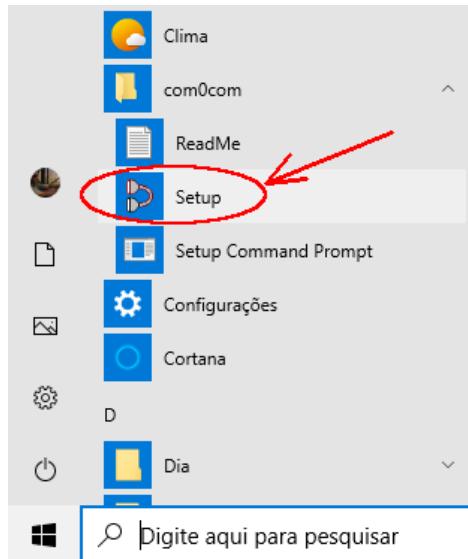


Figura 17.6: Execução do com0com.

17.2.3 Interface do com0com, inserção de pares de portas

A Figura 17.7 apresenta a interface do com0com, apresentando as portas criadas no Gerenciador de Dispositivos do Windows.

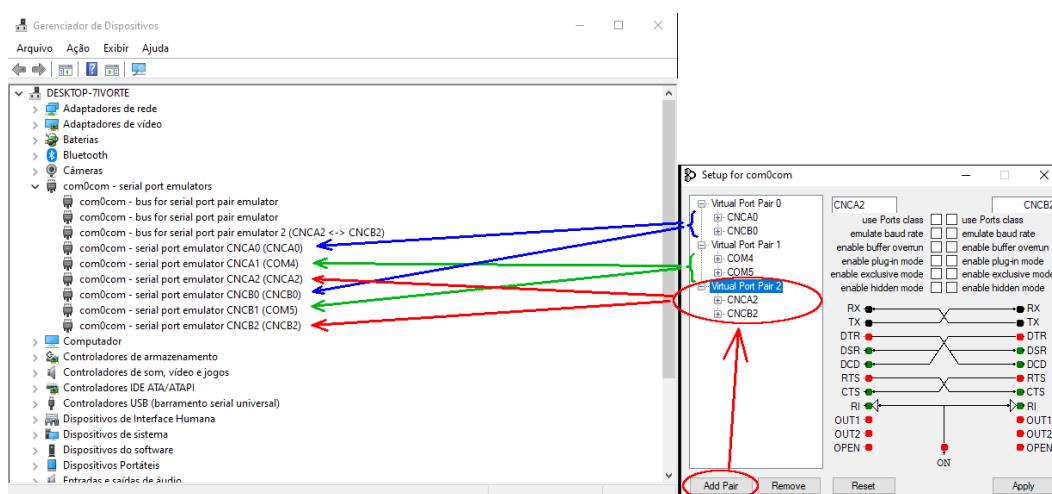


Figura 17.7: Tela do com0com e exemplo de criação de pares de portas.

17.2.4 Troca de nome das portas

O nome das portas criadas pode ser alterado nos campos apresentados na Figura 17.8. Ao digitar o nome e clicar sobre uma das portas é apresentada a janela de confirmação. Caso o nome da porta já esteja em uso, é apresentada uma mensagem informando a impossibilidade de troca do nome.

Neste exemplo no nome está sendo trocado de “CNCB2” para “COM8”, que aparece na Figura 17.7, já teve o nome trocado para “COM6” anteriormente.

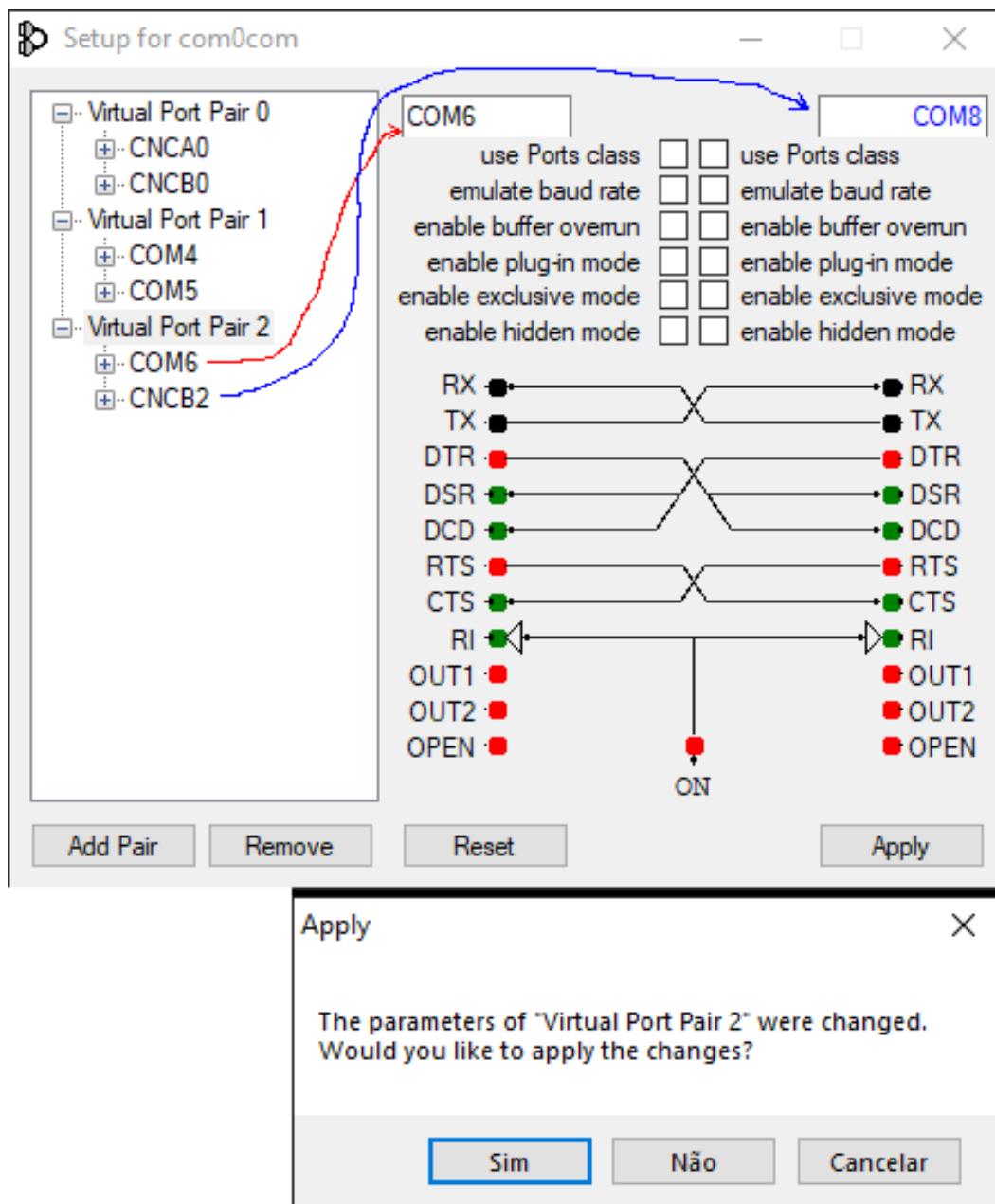


Figura 17.8: Traca do nome de portas virtuais.

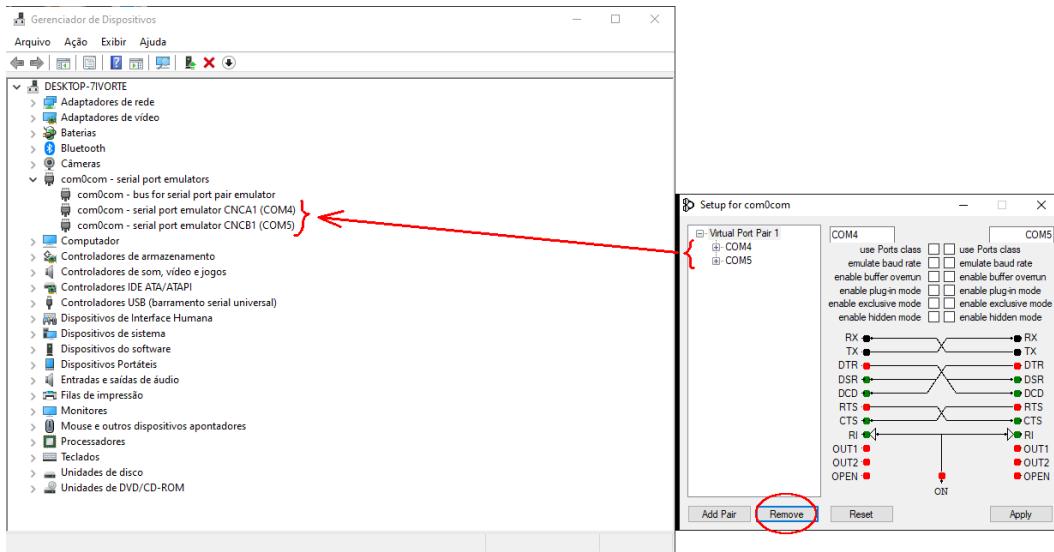


Figura 17.9: Remoção de pares de portas no com0com.

17.2.5 Remoção de portas

É possível remover pares de portas, o que é recomendável para evitar uma quantidade exagerada de portas criadas no Windows.

Para a remoção, seleciona-se o par, clicando sobre o nome (“Virtual Port Pair 2”, por exemplo) e posteriormente clica-se no botão “Remove”.

A Figura 17.9 apresenta o efeito da remoção de alguns pares de portas.

17.2.6 Teste de comunicação das portas criadas com o com0com

A Figura 17.10 apresenta um teste de comunicação com as portas virtuais criadas pelo com0com.

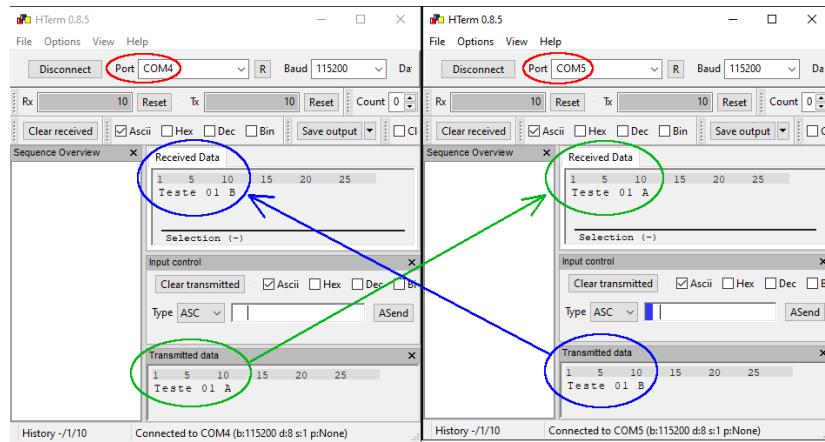


Figura 17.10: Teste de comunicação das portas criadas com o com0com.

Capítulo 18

Universal Asynchronous Receiver-Transmitter - UART

O padrão de comunicação assíncrono UART é um dos mais populares, sendo amplamente utilizado para comunicação entre microcontroladores e periféricos. Mesmo com o surgimento de novos padrões mais modernos, como I²C e SPI, que permitem vários dispositivos em um único barramento, o padrão UART ainda é bastante usado. A seguir são listados alguns periféricos para microcontroladores (Arduino, ESP32, etc) encontrados no mercado que utilizam o padrão UART (podem existir versões com outros padrões, como I²C e SPI):

- Módulo GPS.
- Módulo de comunicação por rede celular: GPRS (2G) e GSM (3G).
- Conversor TTL/RS-485 (útil para uso em uma rede Modbus).

18.1 Diferença entre UART e RS-232

Enquanto o RS-232 é um padrão que determina aspectos físicos do canal de comunicação, o padrão UART define a lógica de envio dos bits. Esses bits podem ser enviado por um canal RS-232, RS-422, RS-485 ou algum outro padrão proprietário Newbedev 2021.

Na comunicação entre microcontroladores e componentes eletrônicos é comum o uso do nível de tensão no padrão TTL na comunicação com o padrão UART.

18.2 Canais de comunicação no padrão UART

Para a comunicação UART cada dispositivo possui uma porta “TX” para transmissão dos dados e uma porta “RX” para recepção. A Figura 18.1 apresenta a ligação entre dois dispositivos que se comuniquem pelo padrão UART.

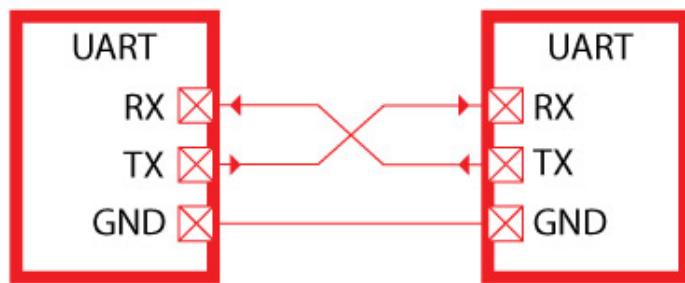


Figura 18.1: Ligação entre dispositivos com padrão UART Mikroe 2016.

18.3 Quadro da mensagem UART

A Figura 18.2 apresenta o quadro de uma mensagem no padrão UART.



Figura 18.2: Quadro (*frame*) da mensagem UART Mikroe 2016.

Os campos observados no quadro são Mikroe 2016:

- **START**: Bit que marca o início da transmissão. Normalmente a linha fica em nível alto e o “Start bit” consiste em levar a linha para o nível baixo.
- **D0-D7**: Os bits de dados com a informação a ser transmitida.
- **P (parity bit)**: é usado para identificar erro na transmissão, para esse bit existem três opções de configuração:

1. **none**: não usar,
 2. **even (par)**: quando a soma dos bits de dados é par esse bit é configurado como 1 na transmissão (se essa relação não ocorre na recepção houve erro na transmissão),
 3. **odd (ímpar)**: quando a soma dos bits de dados é ímpar esse bit é configurado como 1 na transmissão (se essa relação não ocorre na recepção houve erro na transmissão).
- **stop bits (S1 e S2)**: Pode ser configurado em 1 (somente S1) ou 2 (S1 e S2). Corresponde ao tempo mínimo que a linha é mantida em nível alto ao final de uma transmissão.

Na configuração da comunicação serial pelo padrão UART, deve-se configurar os seguintes parâmetros:

- **Taxa de transmissão** número de símbolos por segundo (baud rate). O valor mais comum é 9600. Outros valores padronizados são: 1200, 2400, 4800, 19200, 38400, 57600 e 115200.
- Data bits: Número de bits de dados, de 5 a 8.
- Paridade: “N” (None, sem paridade), E (even, paridade par) ou O (odd, paridade ímpar).
- Stop bits: 1 bit ou 2 bits.

18.4 Exemplos de configuração

Pode adotar uma codificação para a configuração da comunicação serial no padrão UART, por exemplo:

- **9600 8N1**: 9600 baud rate, 8 bits of data, No parity, 1 stop bit
- **115200 8E2** 115200 baud rate, 8 bits of data, Even parity, 2 stop bits

Configuração do Arduino:

Na programação do Arduino é comum a presença da linha de início da porta serial na parte de configuração, conforme apresentado a seguir.

```
void setup() {
    Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
}
```

A configuração padrão do Arduino é SERIAL_8N1 (8 bits de dados, sem paridade com 1 stop bit). A lista completa de códigos para alterar essa configuração pode ser encontrada no seguinte endereço:

<https://www.arduino.cc/reference/en/language/functions/communication/serial/begin/>.

Configuração de ferramentas de comunicação serial:

Ferramentas genéricas de comunicação serial apresentam campos para as configurações dos parâmetros necessários. A Figura 18.3 destaca os campos de configuração do HTERM.

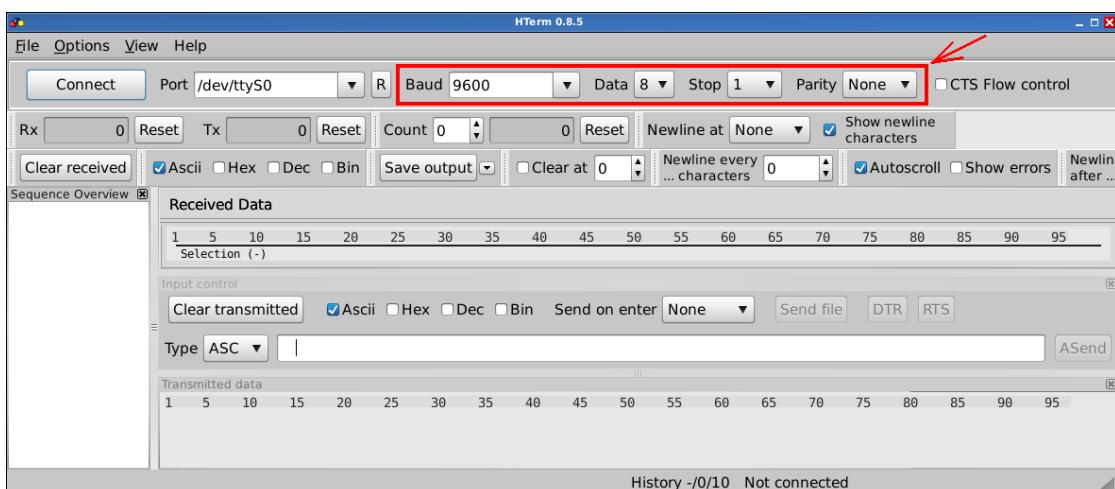


Figura 18.3: Configuração da comunicação serial no HTERM.

18.5 Comunicação com periféricos por UART

Para comunicação com componentes ligados a microcontroladores (Arduino, ESP32, etc) deve-se observar o padrão de comunicação do componente e configurar o microcontrolador para usar o mesmo padrão. Também pode ser possível que o componente aceite ser configurado por algum comando. Para esse tipo de configuração é importante ler as informações disponíveis em datasheets, nos anúncios em fóruns e tutoriais da internet.

A Figura 18.4 apresenta como exemplo um módulo GPS para uso com microcontroladores, observa-se a presença dos pinos “RX” e “TX” e a informação da taxa de transmissão

padrão (a possibilidade de mudança deve ser consultada no datasheet do chip adotado por esse módulo).

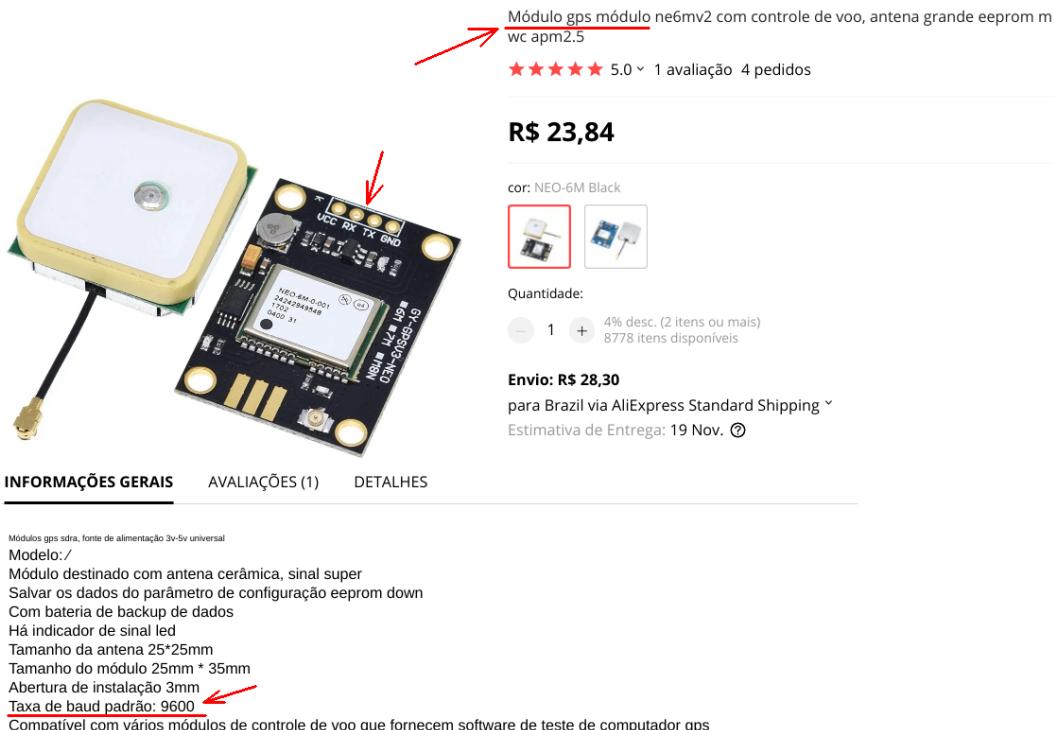


Figura 18.4: Exemplo de componente que usa o padrão de comunicação UART.

Módulos como esse de GPS costumam ter bibliotecas para acesso às suas funções.

18.6 Disponibilidade de UARTs no Arduino

Cada versão do Arduino apresenta um determinado número de UARTs, conforme o microcontrolador usado na placa. Deve-se observar que ao menos uma das UARTs é compartilhada com a conexão USB do Arduino, portanto, se a o Arduino for ser usado conectado ao computador pela USB os pinos da primeira UART (Serial 0, referenciada somente como “Serial”) podem não estar disponíveis para uso com outros componentes. A Figura 18.5 apresenta os pinos usados pelas UARTs nos diferentes modelos de Arduino, destacando o compartilhamento dos pinos 0 e 1 com a porta USB. Foram destacados os modelo Uno e Mega, para comparação do número de UARTs disponíveis.

Used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART), and some have several.

BOARD	USB CDC NAME	SERIAL PINS	SERIAL1 PINS	SERIAL2 PINS	SERIAL3 PINS
Uno, Nano, Mini		0(RX), 1(TX)			
Mega		0(RX), 1(TX)	19(RX), 18(TX)	17(RX), 16(TX)	15(RX), 14(TX)
Leonardo, Micro, Yún	Serial		0(RX), 1(TX)		
Uno WiFi Rev.2		Connected to USB	0(RX), 1(TX)	Connected to NINA	
MKR boards	Serial		13(RX), 14(TX)		
Zero	SerialUSB (Native USB Port only)	Connected to Programming Port	0(RX), 1(TX)		
Due	SerialUSB (Native USB Port only)	0(RX), 1(TX)	19(RX), 18(TX)	17(RX), 16(TX)	15(RX), 14(TX)
101	Serial		0(RX), 1(TX)		

On Uno, Nano, Mini, and Mega, pins 0 and 1 are used for communication with the computer. Connecting anything to these pins can interfere with that communication, including causing failed uploads to the board.

Figura 18.5: Pinos das UARTs nos diferentes modelos de Arduino Arduino® 2021b.

A Figura 18.6 apresenta uma imagem de um Arduino Uno, destacando os pinos da sua UART. A Figura 18.7 apresenta uma imagem de um Arduino Mega, destacando os pinos da suas UARTs.

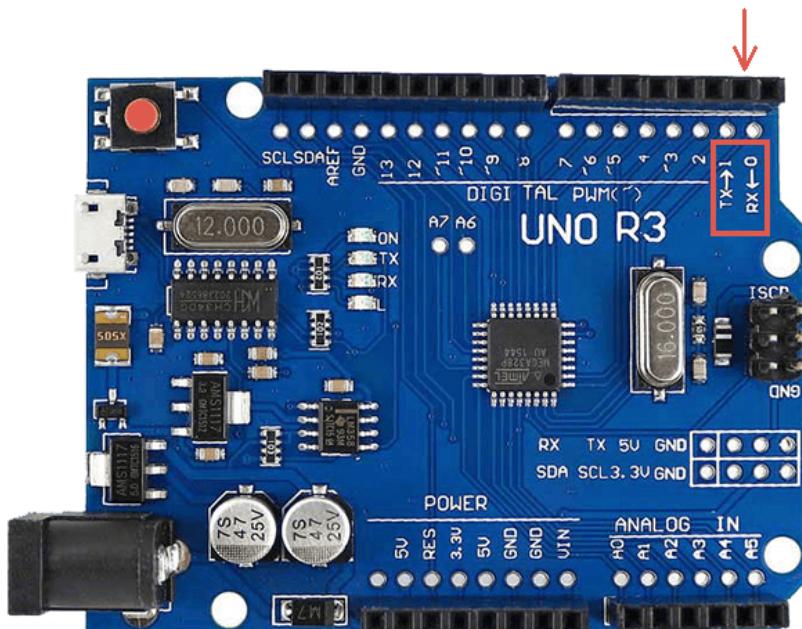


Figura 18.6: Arduino Uno, destaque dos pinos da UART.

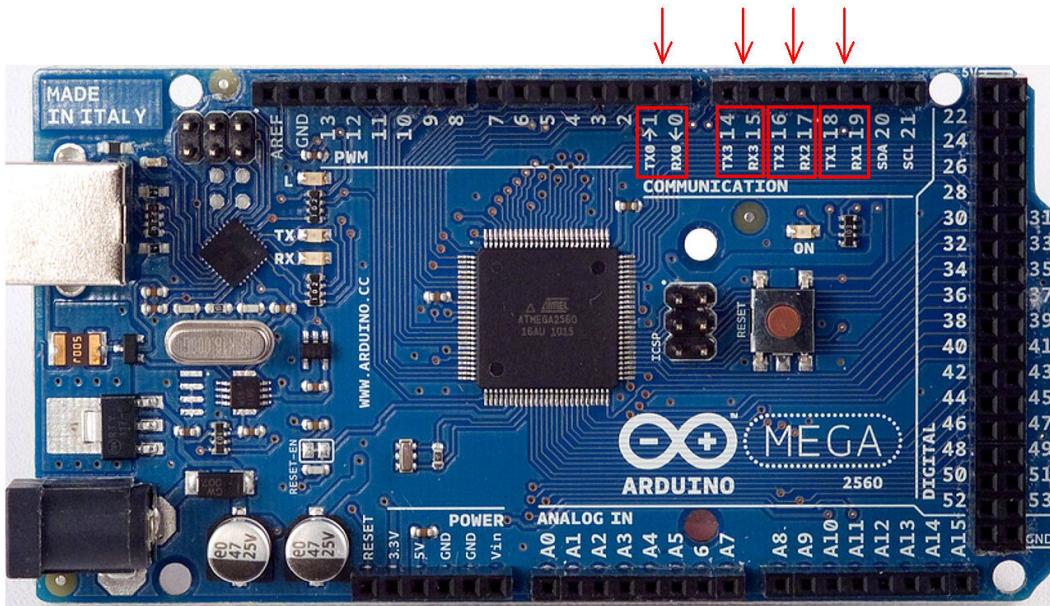


Figura 18.7: Arduino Mega, destacando os pinos das UARTs.

Essas portas seriais do Arduino são implementadas por hardware no microcontrolador usado para construção da placa. É possível a criação de portas seriais por software, usando outros pinos do Arduino. Essa possibilidade é apresentada na Subseção 18.6.1

18.6.1 SoftwareSerial (porta UART por software)

Caso não se tenha uma porta serial disponível (para o Arduino Uno basta que a conexão USB esteja em uso) pode ser usada a biblioteca “SoftwareSerial”. Com essa biblioteca outros pinos podem ser usados para comunicação serial no padrão UART. Instruções para uso dessa biblioteca e exemplos podem ser encontrados na página oficial da biblioteca “SoftwareSerial Library” Arduino® 2021c.

18.6.2 Limitações da biblioteca SoftwareSerial

A biblioteca SoftSerial pode apresentar limitações, algumas são listadas na página oficial da biblioteca Arduino® 2021c.

Um caso particular que encontrei é a impossibilidade de uso de uma porta serial (UART) criada por software com a biblioteca “ArduinoRS485” Arduino® 2021a. Essa biblioteca é usada para comunicação por RS-485 com um módulo de conversão TTL/RS-485. Até a versão dessa biblioteca que testei, somente portas seriais definidas em hardware eram aceitas

para a comunicação entre o Arduino e o conversor TTL/RS-485. Essa limitação pode gerar problemas, principalmente com o Arduino Uno, onde a única UART definida por hardware (pinos 0 e 1) é compartilhada com a porta USB.

Parte IV

Experimentos e considerações práticas sobre Modbus

Capítulo 19

Exemplos sensores e atuadores Modbus de baixo custo

19.1 Introdução

Devido à simplicidade do Modbus, em comparação a padrões como PROFIBUS e FOUNDATION Fieldbus, esse protocolo possui uma grande variedade de bibliotecas para sua implementação em programas que executem em computadores (PCs ou Raspberry pi) e em microcontroladores e placas baseadas em microcontroladores (Arduino e ESP32).

Outro fator que torna o Modbus uma boa escolha para automação com baixo custo, é o fato de usar como camada física mais comum o padrão RS-485. Existem sensores, como o sensor de temperatura e umidade SHT20, e atuadores, como relés, que usam o padrão Modbus sobre RS-485.

Barramentos RS-485 são facilmente acessíveis por computadores e microcontroladores com o uso de adaptadores tipo USB/RS-485 ou TTL/RS-485.

A ferramenta de programação do ESP32, ESP-IDF, disponibiliza uma biblioteca Modbus/TCP, além do Modbus/RTU. Isso possibilita a comunicação com padrão Modbus/TCP por meio de um canal Wi-fi, já presente no ESP-32. Nota: Ainda não testei o Modbus/TCP no ESP32.

19.2 Sensores e atuadores Modbus de baixo custo

Essa seção apresenta um exemplo de sensor e um de atuador que adotam o padrão Modbus sobre RS-485. Esses dispositivos atuam como escravos Modbus, os mestres podem ser computadores, microcontroladores (Arduino) e CLPs, por exemplo.

A Figura 19.1 apresenta um sensor de umidade e temperatura Modbus/RS-485. O código SHT20 é referente ao chip central desse dispositivo, existem encapsulamentos distintos que usam o mesmo chip como base.



Figura 19.1: Sensor de temperatura SHT20.

A Figura 19.2 apresenta um grupo de relés acessível por Modbus.

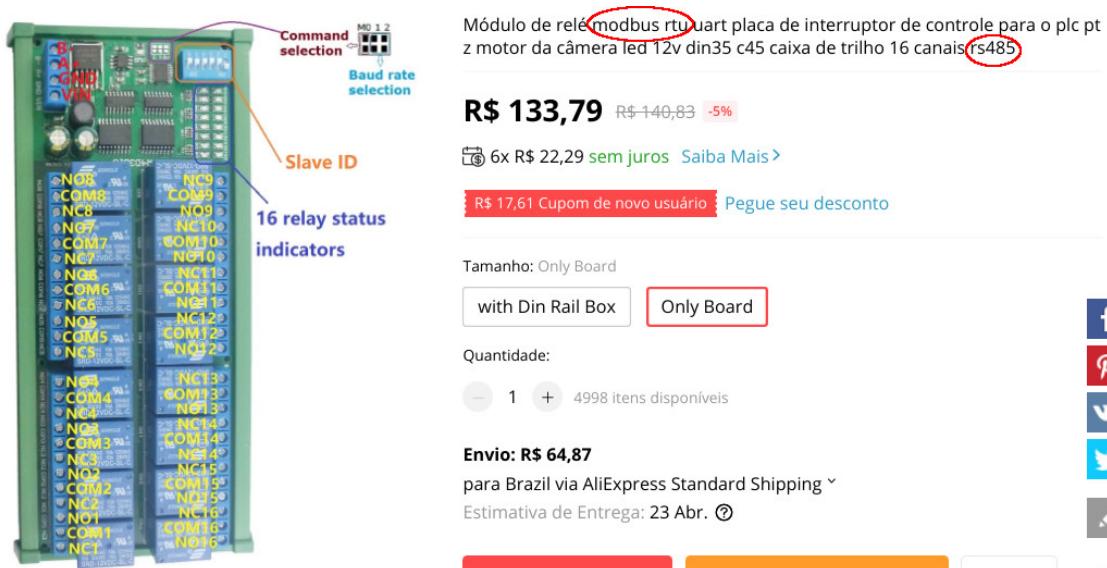


Figura 19.2: Conjunto de relés acessível por Modbus/RS-485.

Importante: Na compra desses dispositivos de baixo custo, os vendedores não costumam enviar manuais de instrução. Portanto, é importante que exista uma forma de acesso

ao manual, pois o uso do Modbus depende de informações como o endereço do escravo, endereço dos registradores e taxa de transmissão.

Para o sensor de temperatura da Figura 19.1, por exemplo, consegui o manual através do código do chip, SHT20, após uma longa busca. Após encontrar o manual o acesso não foi difícil. O manual do sensor SHT20 está disponível no Apêndice A.

No caso dos relés, que não comprei, o acesso parece mais simples, pois existem chaves para atribuir o endereço do escravo e a taxa de transmissão (visto pela Figura 19.2).

Capítulo 20

Exemplos de adaptadores para RS-485

O RS-485 é um dos padrões de camada física mais populares. Como é amplamente usado como camada física do protocolo Modbus, os adaptadores de RS-485 são importantes para comunicação com os dispositivos Modbus de baixo custo vistos na seção anterior a partir de computadores e microcontroladores.

20.1 Adaptadores USB/RS-485

Esse adaptadores possibilitam a ligação de computadores a barramentos RS-485. Esses adaptadores costumam usar chips amplamente usados no mercado, portanto o reconhecimento pelo Windows ou Linux não deve ser difícil, se necessário deve-se buscar o driver do chip usado no adaptador (o nome do chip costuma estar no anúncio).

No Windows, esses adaptadores criam uma porta COM, facilmente reconhecível pelo gerenciador de dispositivos. A nova porta COM deve surgir quando o adaptador for conectado à uma porta USB. No Linux o canal serial deve ser acessível por algum endereço como /dev/ttyUSB0 (exemplo do meu computador).

A leitura e escrita nos dispositivos do barramento RS-485 pode ser feita com programas genéricos de acesso a porta serial, como por exemplo:

- RealTerm (Windows)
- CuteCom (Linux)

Esse canal de comunicação também pode ser usado para sistemas supervisórios (criados como o Elipse E3, por exemplo) acessarem dispositivos físicos (como um Arduino) pelo protocolo Modbus.

Também podem ser usados mestres Modbus, como o **QModMaster**, para acesso aos escravos do barramento acessado pelo computador.

A Figura 20.1 apresenta um adaptador USB/RS-485 duas conexões para o cabeamento (“A” e “B” do RS-485).



Figura 20.1: Adaptador USB/RS-485 com apenas duas conexões para o cabeamento (linha “A” e linha “B” do RS-485).

A Figura 20.2 e Figura 20.3 apresentam um adaptador com conexão para o GND, além dos canais “A” e “B”.



Figura 20.2: Adaptador USB/RS-485 com conexões “A”, “B” e GND para o cabeamento RS-485.



Figura 20.3: Adaptador USB/RS-485 com conexões “A”, “B” e GND para o cabeamento RS-485 (dois dos quatro parafusos são para GND).

Consideração prática: Comprei esse adaptador sem a conexão de GND, apresentado na Figura 20.1. Utilizando o adaptador para comunicação do computador com um Arduino (usando um supervisório criado com Elipse E3), observei que o barramento RS-485 se mantinha estável com o cabo USB ligando o computador ao Arduino (o cabo USB estava sendo usado para programação do Arduino, o supervisório estava acessando pela porta COM do adaptador), porém a conexão ficava instável sem o cabo USB (mantendo a alimentação do Arduino por uma fonte CC de 5V). Não solucionei o problema, acredito a ausência de ligação entre o GND do computador e o GND do Arduino (perdida com a desconexão do cabo USB) tenha gerado a instabilidade.

Portanto, acredito que seja mais adequado o uso dos adaptadores que disponibilizam o conector de GND para o barramento (atenção para os resistores de $100\ \Omega$ na ligação do condutor GND, abordados na Subseção 6.9.5). Ainda não testei essa hipótese.

20.2 Adaptadores TTL/RS-485

Existe um grande número de bibliotecas Modbus para uso com o Arduino. Essas bibliotecas possibilitam, por exemplo, o acesso de sistemas supervisórios ao Arduino.

Com o Arduino ligado ao computador diretamente pelo cabo USB, não existe a necessidade de adaptadores para o padrão RS-485, pois o Windows cria uma porta COM para o Arduino, e essa porta COM já pode ser usada para comunicação Modbus, permitindo a comunicação entre o Arduino e um sistema supervisório, por exemplo.

Porém, caso seja necessário ligar o Arduino a uma barramento RS-485 existente ou a um instrumento que use o RS-485, deve-se utilizar um adaptador do nível TTL para o padrão RS-485.

Esse tipo de adaptador faz a compatibilização do sinal elétrico, sendo ligado a uma das portas seriais do Arduino (pinos RX e TX da UART do Arduino). Do ponto de vista lógico não existe nenhuma alteração de código, escrevendo na porta serial em que o adaptador esteja, a mensagem e inserida no barramento RS-485. Os esquemas de ligação desses adaptadores são facilmente encontrados na internet.

Nota: O Arduino Uno possui apenas uma UART (compartilhada com a conexão USB da placa), o Arduino Mega possui três UARTs (sendo uma compartilhada com a conexão USB da placa). Em todos os casos existem pinos (RX e TX) dedicados para as UARTs, mesmo para as compartilhadas com a conexão USB. Existem bibliotecas para implementação de comunicação serial por software, para aumentar a quantidade de UARTs disponíveis (soft UART). Porém, existem bibliotecas (para Modbus e RS-485, por exemplo) que não aceitam usar uma soft UART. Portanto, deve-se verificar essas características para a biblioteca Modbus usada.

A Figura 20.4 apresenta um adaptador com apenas dois conectores para o barramento (linha “A” e linha “B”).



Figura 20.4: Adaptador TTL / RS-485 com apenas duas conexões para o barramentos RS-485 (linha “A” e linha “B”).

A Figura 20.5 apresenta um adaptador TTL/RS-485 com três conexões para o cabeamento (linha “A”, linha “B” e GND).



Figura 20.5: Adaptador TTL / RS-485 com três conexões para o cabeamento (linha “A”, linha “B” e “GND”).

Aqui vale a mesma consideração feita na seção que abordou o adaptador USB/RS-485, acredito que o adaptador com conexão de GND seja mais adequado.

Capítulo 21

Considerações sobre clientes (mestres) e servidores (escravos) Modbus

Esse capítulo apresenta algumas considerações sobre a estrutura Mestre/Escravo, com enfoque principalmente em Modbus, por ser um padrão de comunicação facilmente implementado em computadores e microcontroladores com uso de bibliotecas.

A estrutura **Mestre/Escravo** também é conhecida como **Cliente/Servidor**. O **mestre** corresponde ao **cliente** e o **escravo** corresponde ao **servidor**.

Embora a estrutura Mestre/Escravo seja uma das mais populares, existem outras filosofias de comunicação na redes industriais. A seguir são citados alguns padrões alternativos às redes Mestre/Escravo:

- Passagem de token: sistema usado em redes com vários mestres, como pode acontecer com o padrão PROFIBUS.
- Sistema da rede CAN: a rede CAN utiliza um sistema de codificação das mensagens com uma lógica de prioridade e controle não destrutivo de conflitos no barramento.
- Publish/Subscribe: sistema padrão do MQTT, e uma das alternativas de comunicação nos protocolos FOUNDATION Fieldbus e OPC-UA.

21.1 Clientes (mestres) em uma rede Modbus

A seguir são listados alguns dispositivos ou programas que podem atuar como mestre Modbus:

- Um CLP,
- Um microcontrolador (Arduino, ESP32, etc), com ou sem uma biblioteca de mestre Modbus.
- Um computador executando um programa genérico de acesso à porta serial, por exemplo o **RealTerm** (Windows) e o **CuteCom** (Linux).
- Um computador executando um programa mestre Modbus, como o **QModMaster**.
- Um computador executando um sistema supervisório (desenvolvido no Elipse E3 ou no ScadaBR, por exemplo).

Do ponto de vista de programação, o mestre é mais simples que o escravo. Para o Modbus, por exemplo, pode-se acessar um escravo escrevendo a mensagem de solicitação a partir de um programa genérico de acesso a porta serial, como o RealTerm ou o CuteCom. Para um Arduino, atuando como mestre, acessar um escravo Modbus também basta realizar escrita da mensagem de solicitação na porta serial e a leitura da resposta enviada pelo escravo.

21.2 Servidores (escravos) em uma rede Modbus

A seguir são listados alguns dispositivos ou programas que podem atuar como escravos Modbus:

- Um microcontrolador (Arduino, ESP32, etc), com uso de alguma biblioteca para escravos Modbus. Um microcontrolador atuando como escravo pode ser acessado por um sistema supervisório ou por outro microcontrolador, que esteja atuando como mestre.
- Um sensor, como o SHT20 (sensor de umidade e temperatura Modbus).
- Um atuador, como um inversor de frequência ou um conjunto de relés comandados por Modbus.
- Um computador com algum programa de teste que funcione como escravo Modbus, como o **pyModSlave** e o **ModbusPal**.

Diferente da função de Mestre, que pode ser executada com qualquer programa capaz acessar as portas seriais para leitura e escrita, a função de escravo demanda alguma biblioteca ou programa próprio. Isso se deve à maior complexidade da função executada pelo escravo, que deve ouvir, interpretar e responder às solicitações.

Existem bibliotecas para implementação de escravos Modbus em diferentes linguagens de programação além das bibliotecas para microcontroladores, como o Arduino e o ESP32, atuarem com escravos Modbus.

21.3 Programas auxiliares para desenvolvimento Modbus

Para o desenvolvimento de aplicações com adoção de comunicação por Modbus, não é recomendável o desenvolvimento simultâneo do mestre e do escravo, pois o processo de depuração e localização de falhas pode ser difícil.

Considerando, por exemplo, a situação em que se deseje realizar a comunicação de dois Arduinos por meio do padrão Modbus, um atuando como mestre e ou outro como escravo. Havendo falha de comunicação, pode ser difícil identificar onde está o erro.

O procedimento mais adequado é a utilização de um programa que atue como mestre ou escravo Modbus para então realizar a programação de apenas um dos nós desejados por vez.

Esta apostila apresenta os seguintes programas para realização de testes de comunicação Modbus:

- Cliente (mestre) Modbus
 1. **QModMaster** (Capítulo 22)
- Servidor (escravo) Modbus
 1. **pyModSlave** (Capítulo 23)
 2. **ModbusPal** (Capítulo 24)

Capítulo 22

QModMaster - Cliente (mestre) Modbus

Este capítulo apresenta o QModMaster. O programa pode ser obtido no Sourceforge (para Windows e Linux), no seguinte endereço:

<https://sourceforge.net/projects/qmodmaster/>

O QModMaster é um programa mestre Modbus, funcionando nas seguintes versões do protocolo:

- Modbus RTU
- Modbus TCP

Embora seja possível realizar a leitura de um escravo Modbus RTU por meio de um programa genérico de acesso a portas seriais, como o RealTerm (Windows) ou CuteCom (Linux) esses programas genéricos apresentam os seguintes inconvenientes:

- Deve-se compor toda a mensagem de solicitação, byte por byte, formando o código de Modbus.
- Deve-se usar algum programa ou site para gerar o código CRC da solicitação.
- Deve-se isolar os bytes de interesse da resposta obtida e realizar a recuperação da variável transmitida.

Com QModMaster a leitura dos escravos Modbus é muito mais simples, pois o QModMaster se encarrega de compor as mensagens de solicitação, incluindo o cálculo do CRC. O QModMaster também recebe a resposta do escravo e se encarrega de exibir o valor da variável lida.

22.1 Instalação

22.1.1 Instalação no Windows

No Windows a instalação é feita de forma simples, executando o instalador obtido na página do Sourceforge (ou outra).

22.1.2 Instalação no Linux

Para o Linux, o Sourceforge disponibiliza um arquivo .zip com o código. Para compilar o programa o usuário deve executar os seguintes passos:

1. Instalar as os programas para compilação. No Debian, Ubuntu e derivados:

```
sudo apt-get install build-essential  
sudo apt-get install qt5-qmake
```

(outros que forem necessários)

2. Descompactar o arquivo com o código e acessar o diretório descompactado por um terminal de comando.
3. Executar o comando:

```
qmake
```

4. Executar o comando:

```
make
```

5. Com isso o executável já está disponível para uso (não é necessário executar “*sudo make install*”). Na pasta com o arquivo executável, executar seguinte comando para iniciar o qModMaster:

```
./qModMaster
```

22.2 QModMaster - Configuração de conexão serial para Modbus RTU

A Figura 22.1 apresenta a janela de configuração da porta serial do QModMaster junto da janela do Gerenciador de Dispositivos do Windows. Neste exemplo o dispositivo listado é um adaptador USB/RS-485 igual ao apresentado na Figura 20.1. Esse mesmo adaptador é acessado no Linux (situação do meu computador) com o primeiro campo “Serial device = /dev/ttyUSB” e o segundo campo “Serial port = 1”.

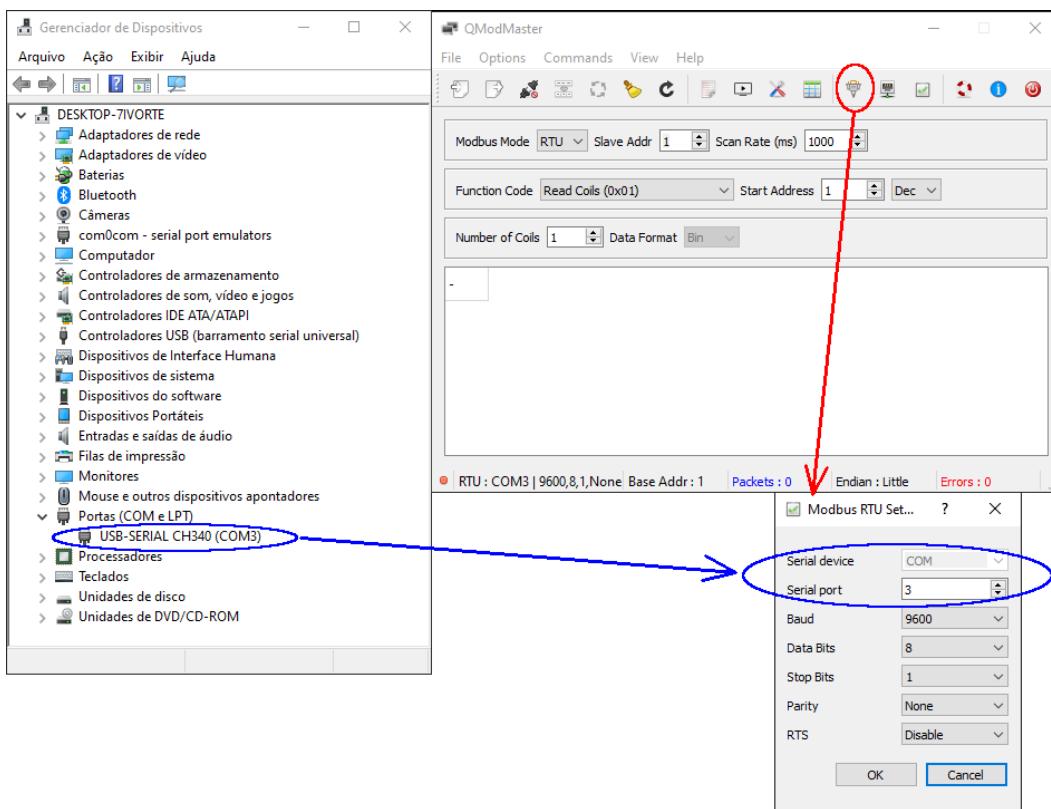


Figura 22.1: Configuração de conexão serial no QModMaster.

22.3 Configuração da conexão TCP/IP, para Modbus TCP

A Figura 22.2 apresenta a janela de configuração para conexão TCP/IP (para acesso a uma escravo Modbus/TCP). O endereço IP informado é o do servidor (escravo). Para um servidor executando no próprio computador do QModMaster, é usado o endereço apresentado na figura, 127.0.0.1. A porta deve ser aquela configurada no servidor (escravo), a porta padrão do Modbus/TCP é a 502.

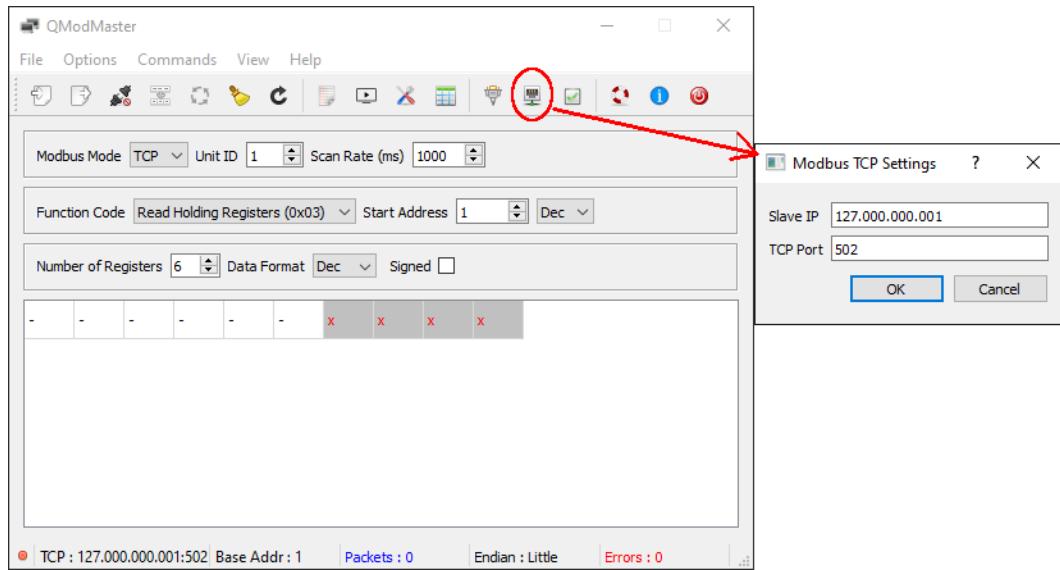


Figura 22.2: Configuração de conexão TCP no QModMaster.

22.4 Estabelecendo a conexão

A Figura 22.3 apresenta o campo de escolha do modo de operação do Modbus.

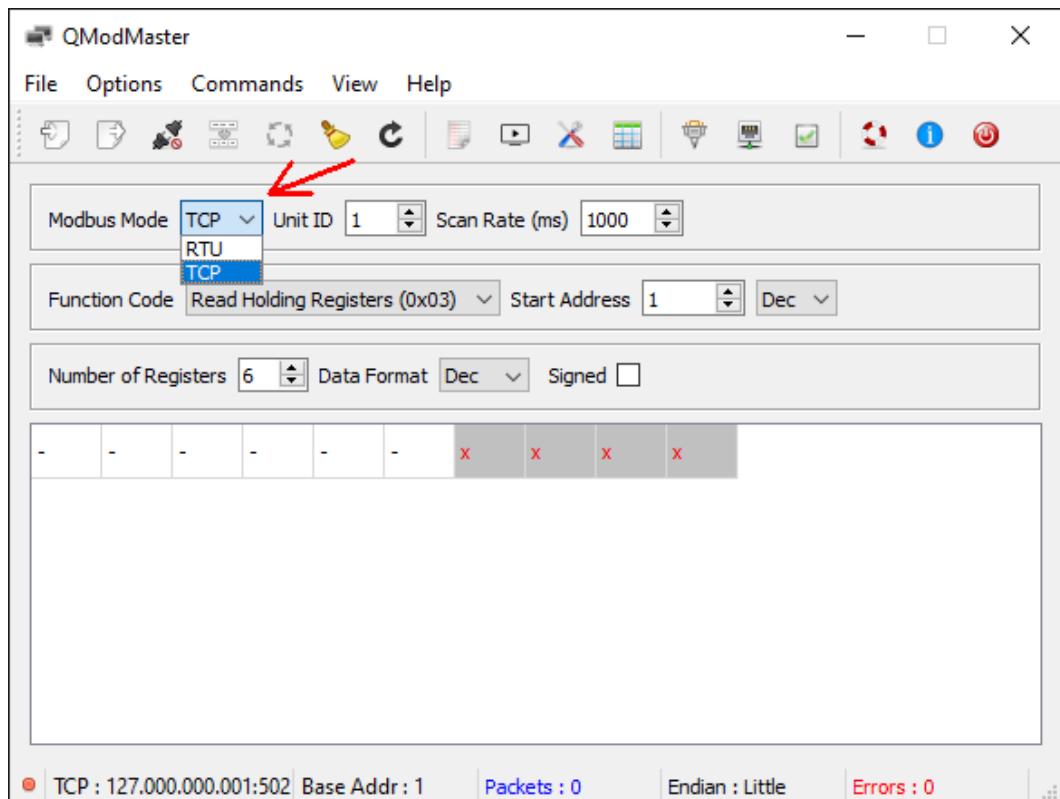


Figura 22.3: Escolha do tipo padrão Modbus.

A Figura 22.4 apresenta o botão de conexão (campo 1) usado para estabelecer a conexão com o servidor (escravo).

O campo 2 apresenta a área de escolha da função (operação) Modbus a ser executada. Essa função pode ser alterada a qualquer momento após a conexão estabelecida.

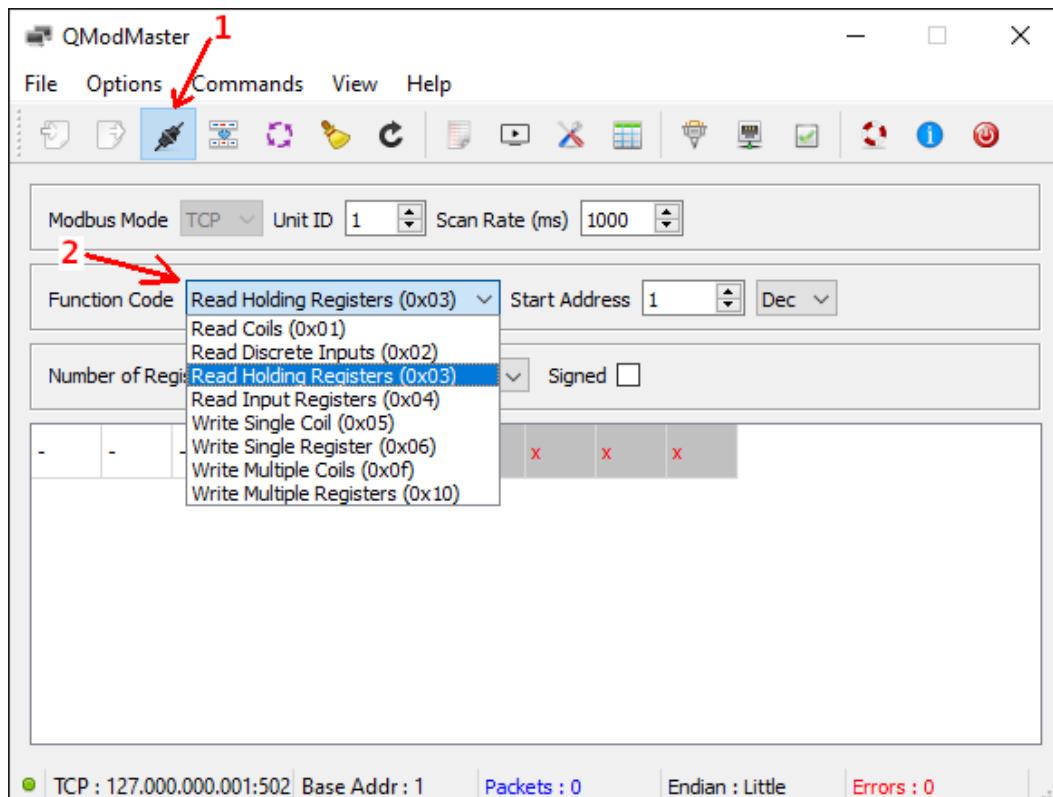


Figura 22.4: Lista de operações e botão para conexão.

22.5 Realizando operações de leitura ou escrita

Além do campo 2 da Figura 22.4 existem outros campos e funções que podem ser manipulados após a conexão estabelecida.

A Figura 22.5 apresenta alguns campos e botões utilizados após o estabelecimento da conexão.

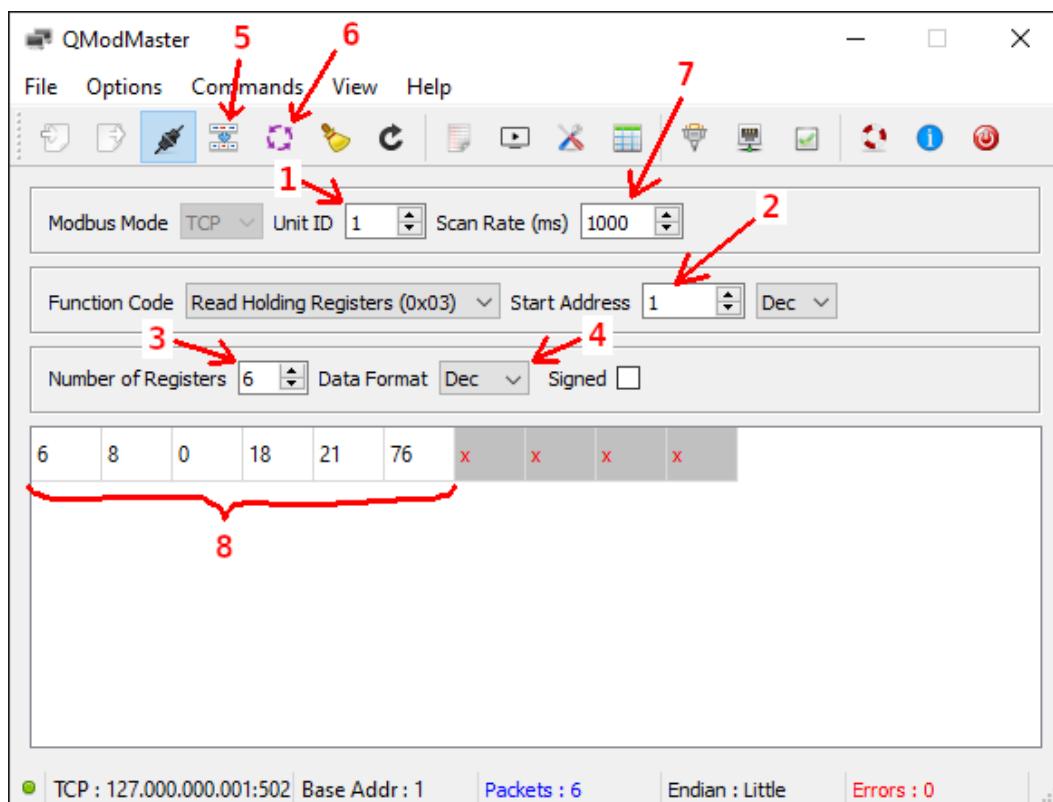


Figura 22.5: Configurações para realização de leitura.

A seguir são listados os campos e botões destacados na Figura 22.5 (segue a numeração marcada na figura).

1. Endereço do escravo a ser acessado.
2. Endereço inicial dos registradores ou bobinas a serem acessados para leitura ou escrita.
3. Número de registradores ou bobinas a serem lidas ou escritas, iniciando do valor informado no campo (2) (a tentativa de acesso a registradores ou bobinas inexistentes no escravo gera uma mensagem de erro).
4. Tipo de dado do registrador lido.
5. Botão para realização de uma operação (selecionada no campo “Function Code”) com os registradores ou bobinas escolhidas.
6. Execução cíclica da função do botão (5).
7. Taxa de execução da operação de leitura ou escrita quando o botão (6) estiver ativo.

8. Área de apresentação dos registradores ou bobinas. Nas operações de leitura esses campos apresentam os valores lidos dos escravos. Nas operações de escrita esses são os campos para o operador digitar os valores a serem escritos no escravo.

22.6 Endereço inicial de registradores e bobinas no qMod-Master

Ver as considerações sobre endereço base no Capítulo 25.

Capítulo 23

pyModSlave - Servidor (escravo) Modbus

O pyModSlave é um servidor (escravo) Modbus com interface semelhante à interface do QModMaster.

23.1 Instalação do pyModSlave

O pyModSlave pode ser executado a partir do projeto obtido no SourceForge.

O primeiro passo para execução é a obtenção do projeto a partir do seguinte endereço:

<https://sourceforge.net/projects/pymodslave/>.

O projeto é salvo como uma pasta compactada “.zip”, na versão obtida no momento de criar esse tutorial o arquivo é o “pyModSlave-code-0.4.3-2.zip”.

Atenção

Na versão 0.4.3-2 foram encontrados alguns erros (podem existir outros). As correções são fáceis de serem feitas e estão na Seção 23.3. Para facilitar vou deixar a versão corrigida no seguinte endereço:

https://drive.google.com/file/d/1PsihHUm_Xd15j2oIQvPaXIzt4w6fX-RR/view?usp=sharing.

Se a versão do Google Drive não estiver disponível basta baixar a versão do SourceForge e realizar as mudanças indicadas na Seção 23.3.

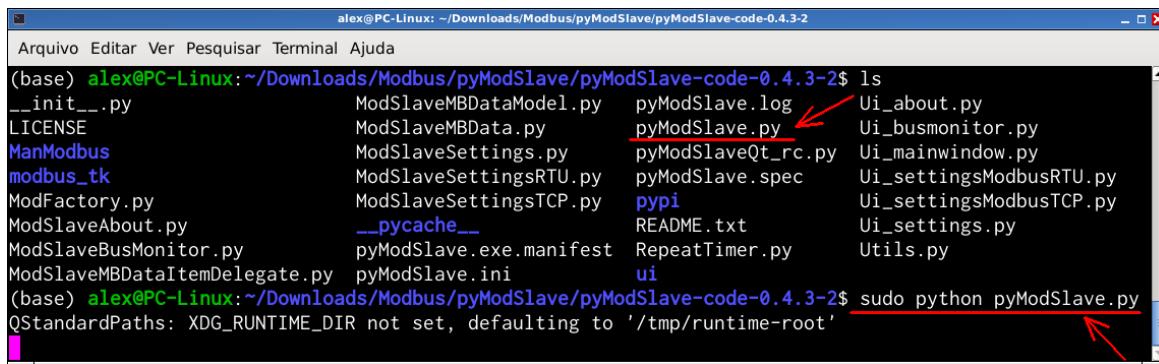
Caso a versão do SourceForge seja atualizada, é recomendável testar a nova versão, pois esses e outros erros podem ser corrigidos.

23.2 Execução do pyModSlave

Após descompactar o arquivo obtido do SourceForge, o pyModSlave deve ser executado a partir do arquivo “”, conforme apresentado na Figura 23.1.

No Linux, é necessário executar com privilégios de superusuário com `sudo` somente se for ser utilizada a comunicação TCP (Modbus TCP). Para a comunicação por uma porta serial virtual (Modbus RTU), criada com o “socat” (ver o Capítulo 17) não é necessário o uso do `sudo` no comando.

No Windows deve-se observar a necessidade ou não de execução como administrador (não testei ainda).



A screenshot of a terminal window titled "alex@PC-Linux: ~/Downloads/Modbus/pyModSlave/pyModSlave-code-0.4.3-2". The window shows a list of files in the current directory. Two specific files are highlighted with red arrows: "pyModSlave.py" and "sudo python pyModSlave.py". The terminal command "ls" is shown at the top, followed by a list of files including __init__.py, LICENSE, ManModbus, modbus_tk, ModFactory.py, ModSlaveAbout.py, ModSlaveBusMonitor.py, ModSlaveMBDataItemDelegate.py, ModSlaveMBDataModel.py, ModSlaveMBData.py, ModSlaveSettings.py, ModSlaveSettingsRTU.py, ModSlaveSettingsTCP.py, __pycache__, pyModSlave.exe.manifest, pyModSlave.ini, pyModSlave.log, pyModSlaveQt_rc.py, pymodSlave.spec, README.txt, RepeatTimer.py, ui, and Utils.py. At the bottom, the command "sudo python pyModSlave.py" is entered, with a red arrow pointing to it. The terminal also displays a warning about QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'.

Figura 23.1: Execução do pyModSlave.

A Figura 23.2 apresenta a janela do pyModSlave, observa-se que a interface é semelhante à interface do QModMaster.

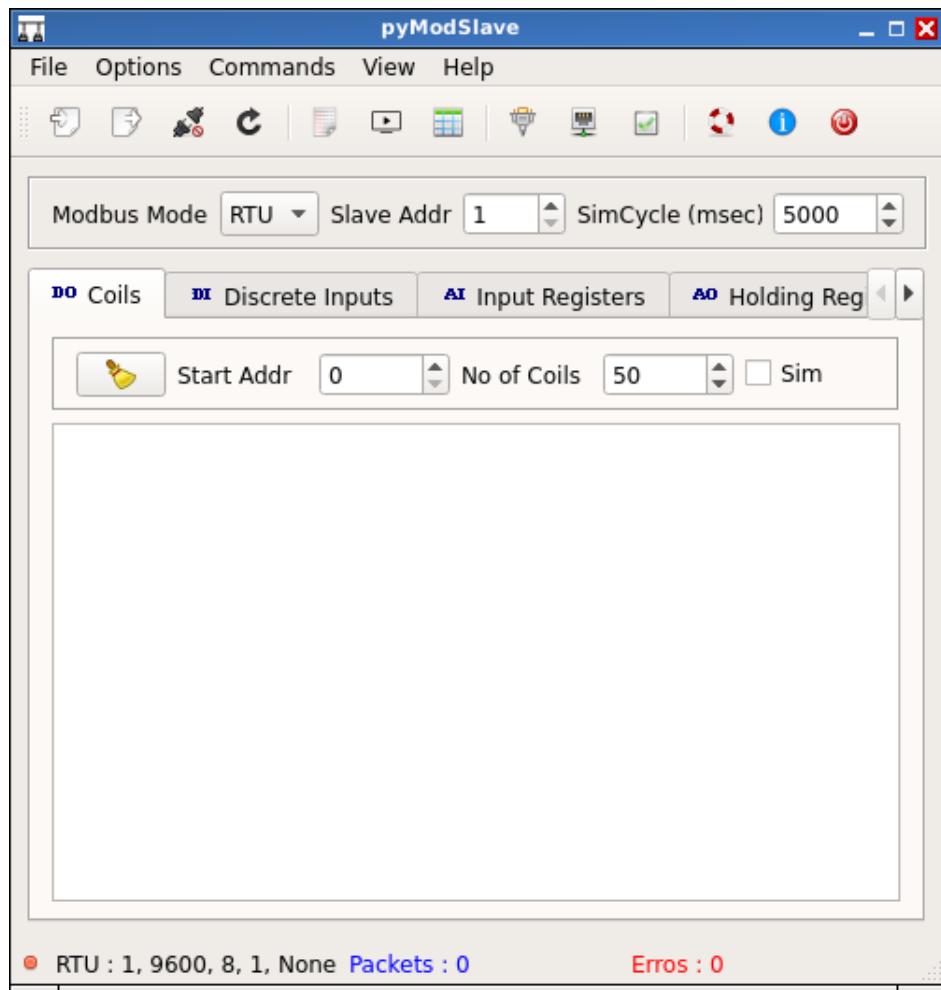


Figura 23.2: Janela do pyModSlave.

23.3 Correção de erros do pyModSlave

Na execução do programa surgiram alguns erros, aparentemente ocasionados por mudanças na implementação do Python. Como os erros são indicados durante a execução, foi possível encontrar as soluções em fóruns da internet.

23.3.1 Erro 01

Esse erro surgia no momento de desativar a conexão. Sendo apresentada a seguinte mensagem:

```
File "/home/alex/Downloads/Modbus/pyModSlave/pyModSlave-code-0.4.3-2/modbus_tk/modbus.py", line 918, in stop
```

```
    if self._thread.isAlive():
AttributeError: 'Thread' object has no attribute 'isAlive'
```

Correção do erro 01:

No arquivo “**modbus_tk/modbus.py**” substitui a linha 918 (linha indicada na mensagem de erro) de

```
1 if self._thread.isAlive():
```

por

```
1 if self._thread.is_alive():
```

23.3.2 Erro 02

Erro apresentado no momento de exibir alguma mensagem na tela. Foi observado na tentativa de estabelecer uma conexão com uma porta inexistente.

Caso esse erro surja em outra situação onde deva ser apresentada uma mensagem, a solução deve ser semelhante.

A seguinte mensagem foi exibida devido a esse erro:

```
File "/home/alex/Downloads/Modbus/pyModSlave/pyModSlave-code-0.4.3-2/Utils.py", line
 19, in errorMessageBox
    QtGui.QMessageBox.critical(None, "Error", msg, QtGui.QMessageBox.Ok, QtGui.QMessageBox
                               .NoButton)
AttributeError: module 'PyQt5.QtGui' has no attribute 'QMessageBox'
```

Correção do erro 02:

Incluir a seguinte linha na parte de import do arquivo “**Utils.py**” (arquivo indicado na mensagem de erro):

```
1 from PyQt5 import QtWidgets
```

Também no arquivo “**Utils.py**” substituir a linha 19 (linha indicada na mensagem de erro) de

```
1 QtGui.QMessageBox.critical(None, "Error", msg, QtGui.QMessageBox.Ok, QtGui.QMessageBox.
                               NoButton)
```

para

```
1 QtWidgets.QMessageBox.critical(None, "Error", msg, QtWidgets.QMessageBox.Ok, QtWidgets.
                               QMessageBox.NoButton)
```

Capítulo 24

ModbusPal - Servidor (escravo) Modbus

Este capítulo apresenta o ModbusPal. O programa pode ser obtido no Sourceforge (programa em java, o mesmo arquivo .jar funciona no Windows e no Linux), no seguinte endereço:

<https://sourceforge.net/projects/modbuspal/>

24.1 Iniciando o ModbusPal

O ModbusPal é um programa feito em Java, no download é disponibilizado um arquivo chamado “ModbusPal.jar”.

Execução do programa no Windows: No Windows basta um duplo clique sobre o arquivo “ModbusPal.jar” para iniciar o ModbusPal. No meu computador o programa executou e permitiu a criação do servidor Modbus/TCP sem necessidade de outros passos.

Caso seu computador não aceite criar o servidor Modbus/TCP, verifique se é necessário liberar essa execução no Firewall do Windows, ou executar o programa como Administrador.

Execução do programa no Linux: O ModbusPal inicia com a execução sem privilégios de superusuário. Porém, não permite a criação do servidor Modbus/TCP (o erro só é percebido ao tentar conectar um cliente (mestre) Modbus/TCP, como o qModMarter). Para a criação do servidor é necessário executar o programa como superusuário (root), com o comando abaixo:

```
sudo java -jar ModbusPal.jar
```

Nota: As imagens deste capítulo foram captadas no Windows, os testes de comunicação foram feitos usando o programa qModMaster como mestre Modbus. Os mesmos testes foram feitos no Linux, também apresentando um funcionamento adequado.

A Figura 24.1 apresenta a interface do server (escravo) Modbus, ModbusPal.

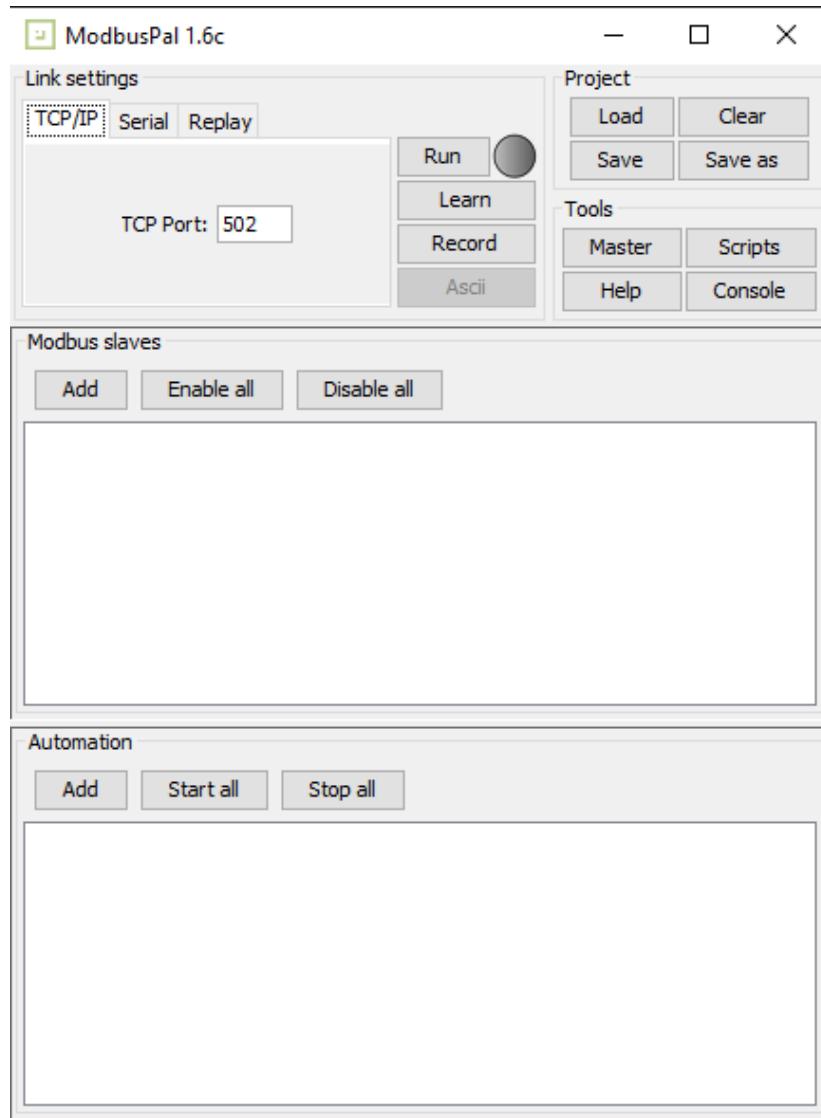


Figura 24.1: Interface do server (escravo) ModbusPal.

24.2 Criação de um único escravo Modbus/TCP

A Figura 24.2 apresenta a forma de criação de um único escravo Modbus/TCP.

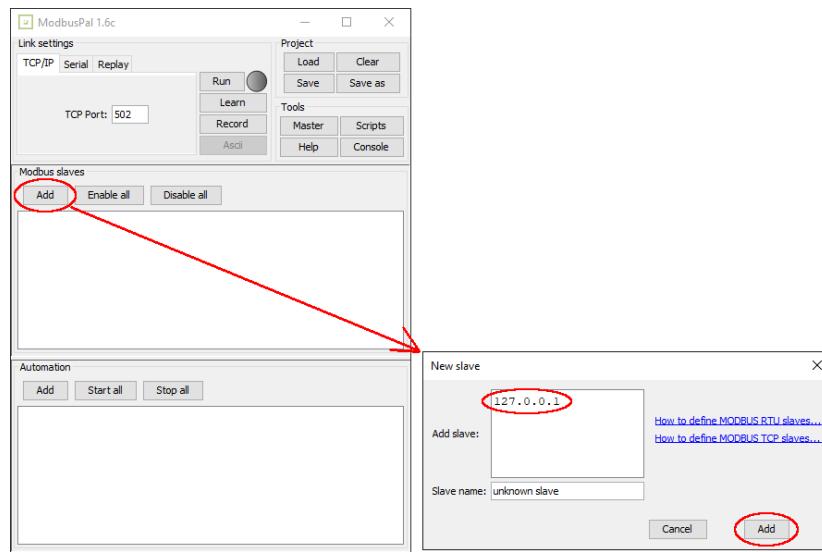


Figura 24.2: Adição de um escravo Modbus/TCP.

A Figura 24.3 apresenta o programa pronto para execução com um único escravo Modbus/TCP.

A Figura 24.3 também apresenta a janela para inserção, visualização e escrita dos registradores e bobinas (coils) do escravo criado.

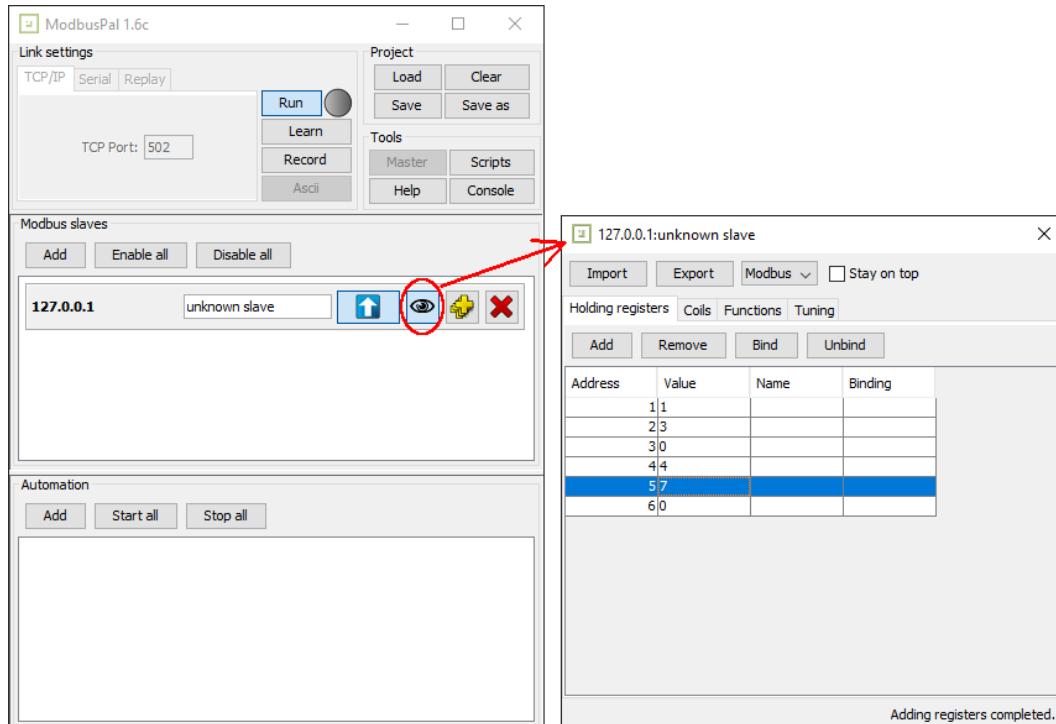


Figura 24.3: Execução com um único escravo Modbus/TCP.

Escolha do endereço do server(escravo):

Mesmo para a criação de um único escravo é possível fazer a escolha do endereço deste escravo, fazendo a indicação do valor na frente do IP, como apresentado na Seção 24.3. A declaração de um escravo com endereço 14, por exemplo, seria “127.0.0.1(14)”.

Escolha do endereço IP:

Caso esse servidor (escravo) for ser acessado por outro computador, através de uma rede, deve-se configurar o(s) escravo(s) com o endereço IP do computador que está executando o ModbusPal. Esse endereço pode ser obtido com o comando “**ipconfig**” em um prompt de comando do Windows e com o comando “**sudo ifconfig**” no Linux. Podem existir outras formas de obter o IP pela interface gráfica, isso pode ser facilmente encontrado na internet.

Para as situações onde o acesso será feito por um programa executando no próprio computador, como o qModMaster, utiliza-se o endereço padrão “127.0.0.1”, tanto para o Windows quanto para o Linux.

24.3 Criação de múltiplos escravos Modbus/TCP

A Figura 24.4 apresenta o exemplo da criação de três escravos Modbus/TCP. A número entre parenteses indica o endereço de cada escravo.

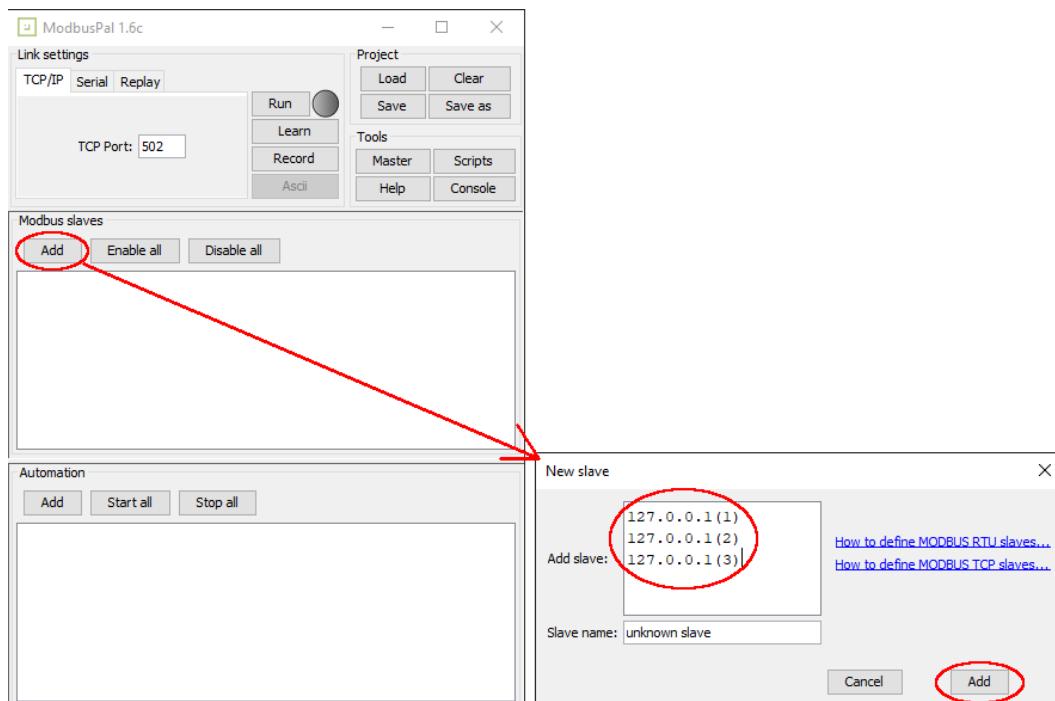


Figura 24.4: Adição de três escravos Modbus/TCP.

A Figura 24.5 apresenta o programa pronto para execução com um único escravo Modbus/TCP.

A Figura 24.5 também apresenta a janela para inserção, visualização e estrita dos registradores e bobinas (coils) dos escravos criados.

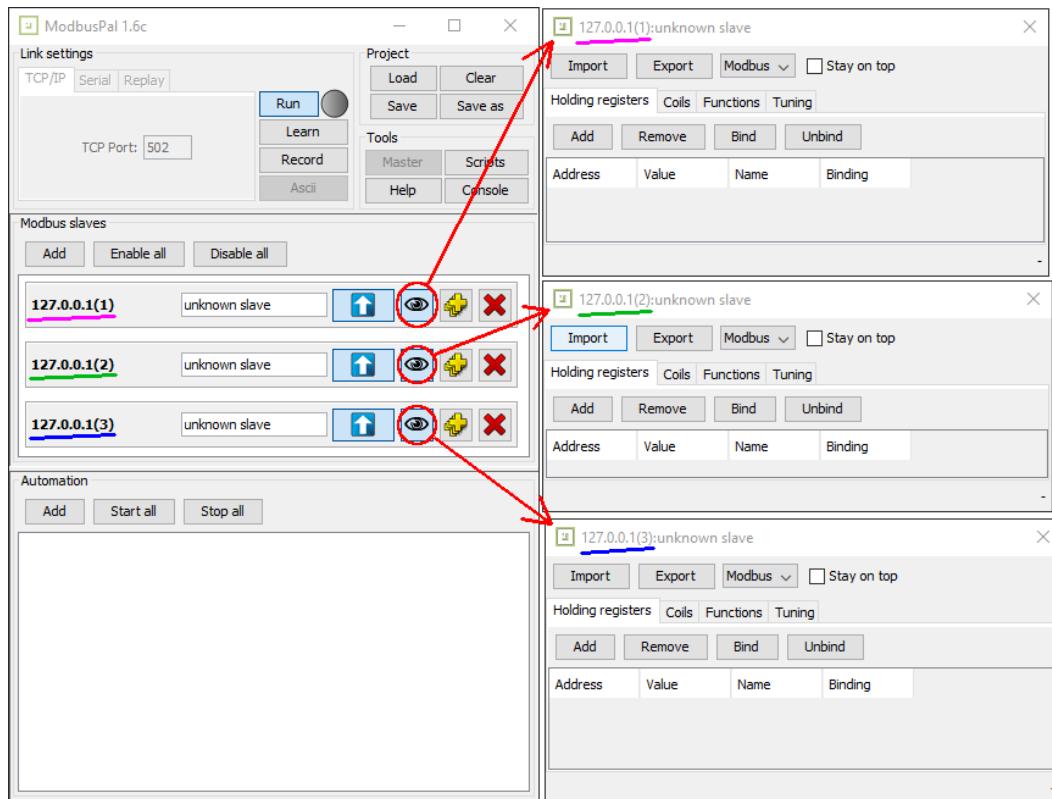


Figura 24.5: Execução com um três escravo Modbus/TCP.

24.4 Criação de registradores em um escravo

A Figura 24.6 apresenta a janela para inserção de registradores (valores de 16 bits) no escravo selecionado. Neste exemplo estão sendo criados 10 registradores, iniciando do endereço 1.

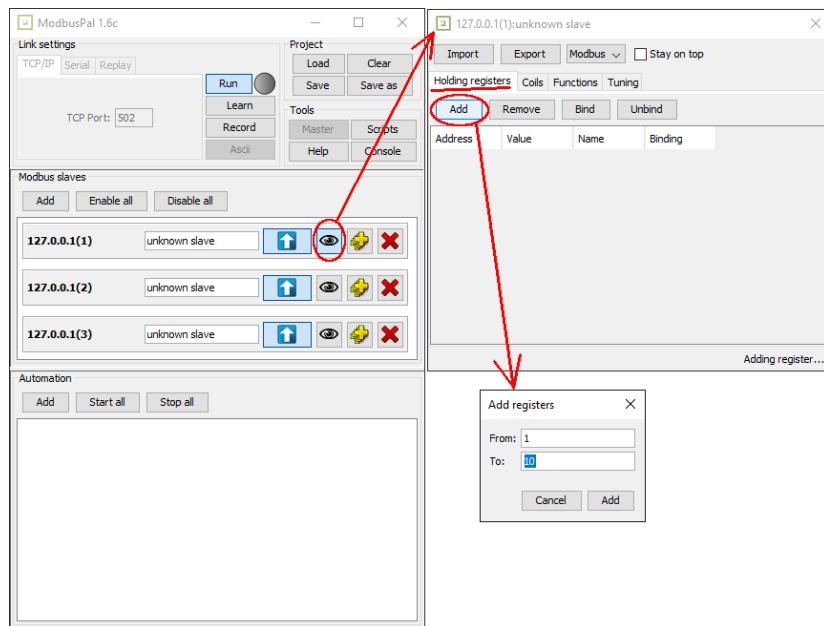


Figura 24.6: Criação de registradores em um escravo.

A Figura 24.7 apresenta os registradores já criados, com alguns valores escritos para exemplificação.

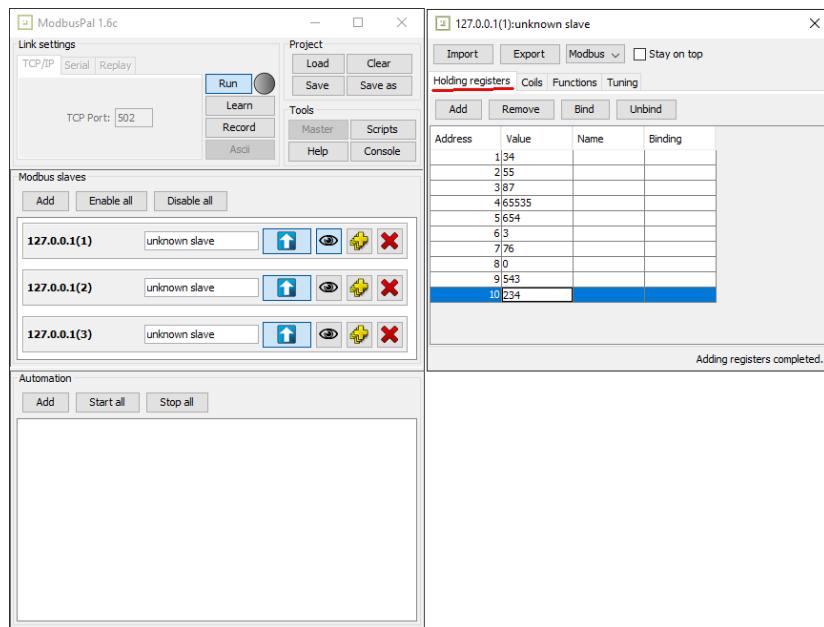


Figura 24.7: Registradores criados em um escravo.

24.5 Criação de bobinas (coils) em um escravo

A Figura 24.8 apresenta a janela para inserção de bobinas (valores de discretos, 1 bit) no escravo selecionado. Neste exemplo estão sendo criadas 20 bobinas, iniciando do endereço 1.

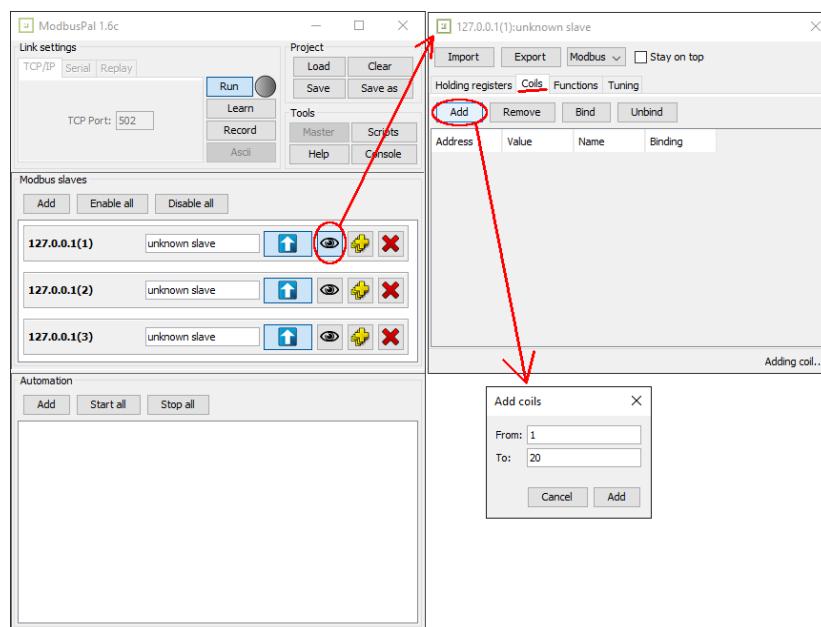


Figura 24.8: Criação de bobinas (coils) em um escravo.

Na criação de registradores não é obrigatório iniciar do endereço 1 (From 1). Pode-se escolher qualquer valor inicial (que não ultrapasse o número máximo de registradores disponíveis). O mesmo é válido para as bobinas.

A Figura 24.9 apresenta as bobinas já criadas, com alguns valores escritos (sómete valores “0” ou “1”) para exemplificação.

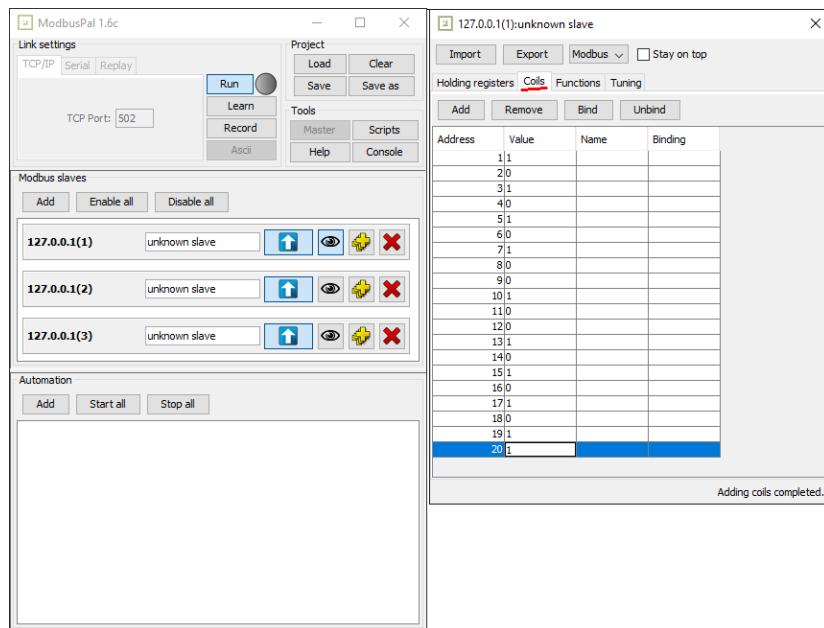


Figura 24.9: Bobinas (coils) criadas em um escravo.

24.6 Indicação de acesso a um dos escravos (leitura ou escrita)

Quando existe um cliente (mestre) realizando a leitura ou escrita em um dos escravos, o ModbusPal indica piscando a luz destacada na Figura 24.10. Observa-se que para aceitar conexões o botão "Run" ao lado da luz destacada deve estar acionado. Neste exemplo o acesso está sendo feito pelo qModMaster (executando no Windows).

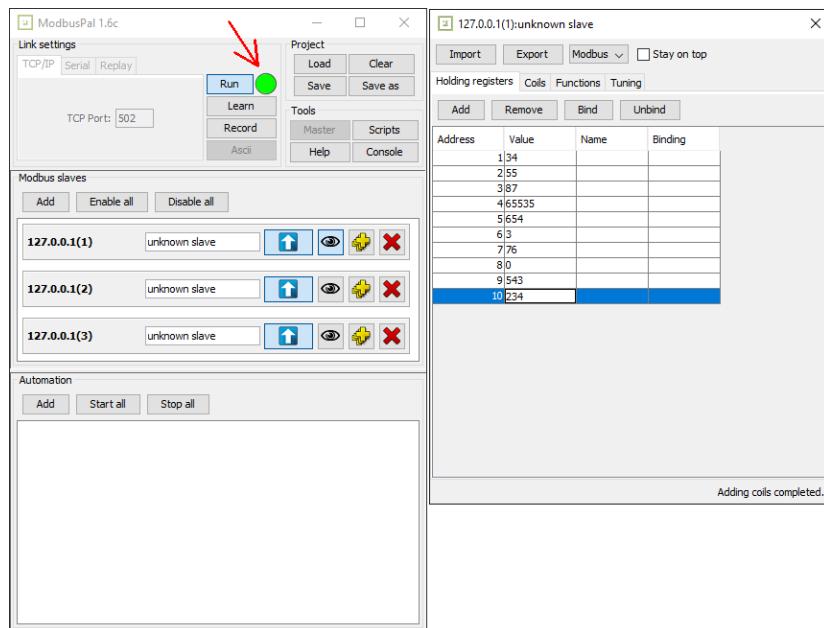


Figura 24.10: Indicação do acesso (leitura ou escrita) a um dos escravos criados no Modbus-Pal.

24.7 Consideração sobre o endereço inicial dos registradores e das bobinas

Ver as considerações sobre endereço base no Capítulo 25.

Capítulo 25

Base Address no Modbus

Por um problema de documentação existem duas implementações de endereço inicial no padrão Modbus:

- **Endereço base = 0:** Considera o primeiro endereço de cada grupo de registrador (offset) como 0. Os dispositivos transmitem o endereço de cada registrador pela mensagem Modbus sem mudança do valor (Ex.: O endereçamento do registrador 3 é transmitido como 3 na mensagem Modbus).
- **Endereço base = 1:** Considera o primeiro endereço de cada grupo de registrador (offset) como 1. Os dispositivos transmitem o endereço de cada registrador pela mensagem Modbus com a subtração de uma unidade (Ex.: O endereçamento do registrador 3 é transmitido como 2 na mensagem Modbus).

Observa-se que para dois dispositivos conversarem usando o mesmo endereçamento, devem adotar o mesmo endereço base (0 ou 1). Para permitir que isso sempre ocorra os dispositivos mestre (clientes) costumam ter um campo para configuração da base (por exemplo o QModMaster ou o driver de comunicação Modbus do Elipse E3). A Figura 25.1 apresenta o caminho para acesso à janela de configuração do endereço de base no QModMaster.

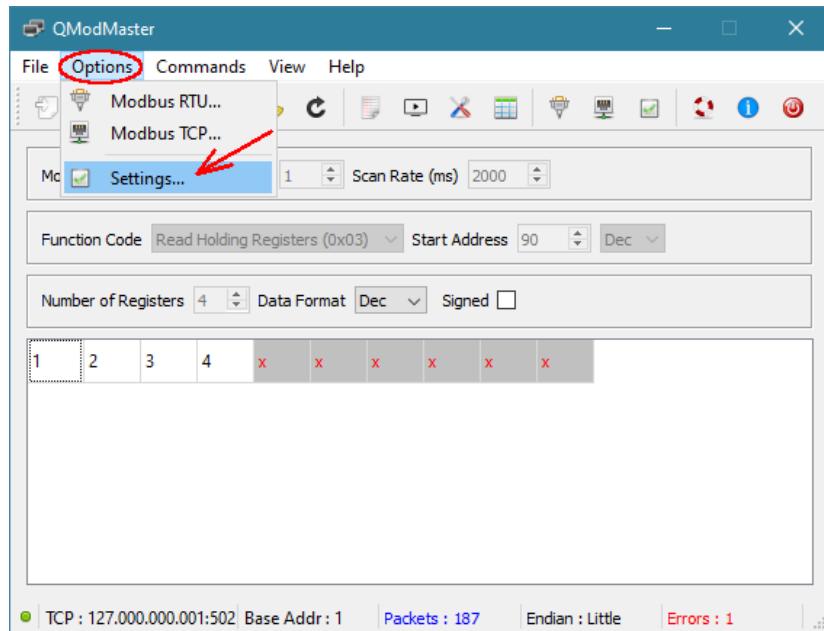


Figura 25.1: Acesso à janela de configuração para escolha do endereço de base no QModMaster.

25.1 Exemplo de mestre e escravo usando bases diferentes

O escravo (servidor) Modbuspal (no Windows) adota o endereço base igual a 1.

A Figura 25.2 apresenta o efeito de utilização do mestre (cliente) configurado com base 0 (observa-se o deslocamento dos endereços).

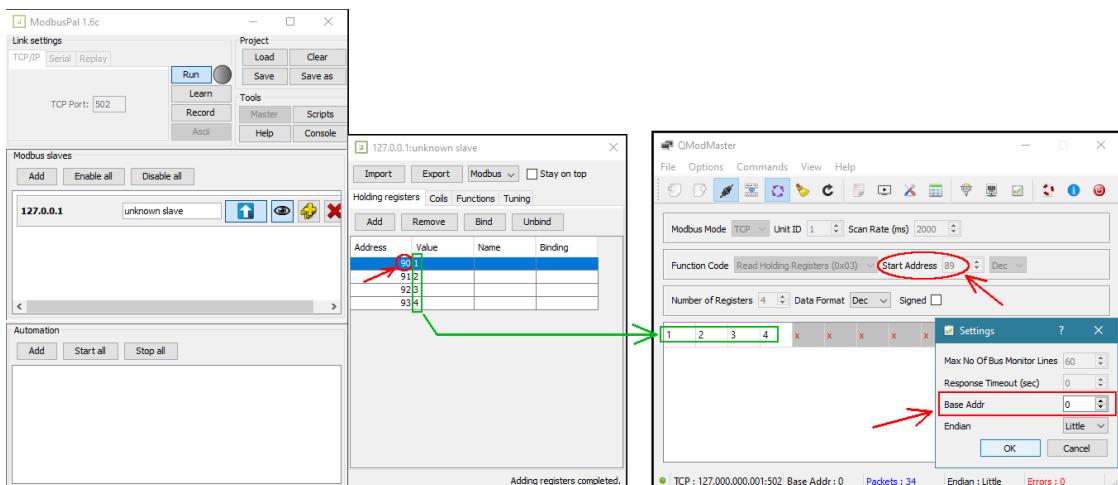


Figura 25.2: Escravo (ModbusPal) usando o endereço de base 1 e mestre usando o endereço de base 0 (exemplo executando no Windows).

Nesse exemplo o endereço de registrador 89 configurado no QModMaster é transmitido pela mensagem Modbus como 89 (pois está configurado com endereço de base 0). No ModbusPal o valor 89 é convertido para acessar o registrador 90 (pois está configurado com endereço de base 1). Essa não é a configuração ideal, pois o mestre e o escravo ficam com um deslocamento para endereçamento das variáveis transmitidas.

25.2 Exemplo de mestre e escravo usando a mesma base

A Figura 25.3 apresenta o mestre e o escravo usando a mesma base (endereço de base igual a 1).

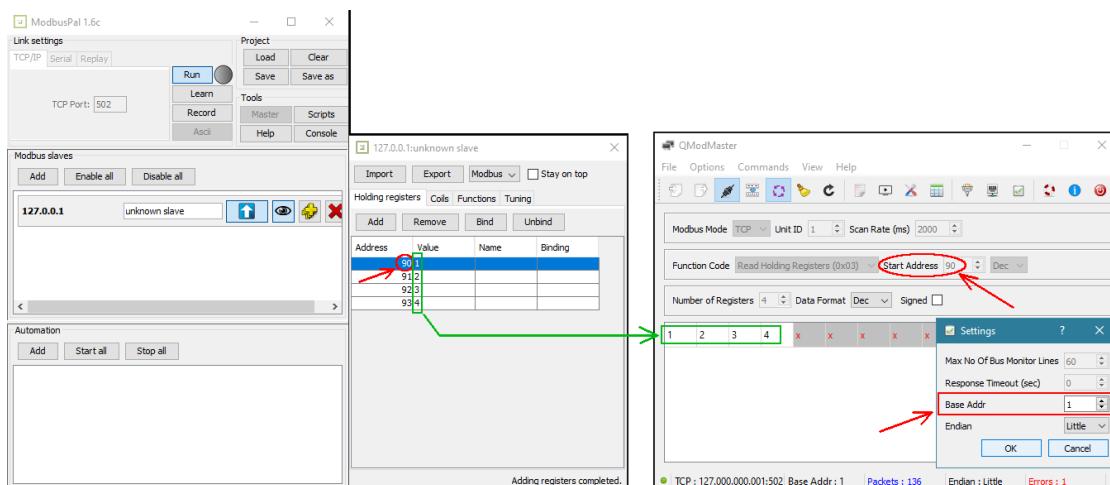


Figura 25.3: Escravo (ModbusPal) e mestre usando o endereço de base 1 (exemplo executando no Windows).

Nesse exemplo o endereço de registrador 90 configurado no QModMaster é transmitido pela mensagem Modbus como 89 (pois está configurado com endereço de base 1). No ModbusPal o valor 89 é convertido para acessar o registrador 90 (pois também está configurado com endereço de base 1). Essa é a configuração adequada, pois o mestre e o escravo usam o mesmo endereço para cada variável.

25.2.1 Bug de base do QModMaster no Linux

No Linux, o QModMaster (versão 0.5.2-3) possui um bug onde a mudança de base de 0 para 1 agrava o deslocamento de endereço dos registradores.

Portanto, no Linux é mais adequado utilizar a base 0 no QModMaster e aceitar que haverá o deslocamento de uma unidade se o escravo trabalhar com base 1. Caso o escravo adote o endereço de base 0 não haverá esse deslocamento.

A Figura 25.4 apresenta o QModMaster (Linux) configurado com endereço de base igual a 0 (lembrando que o ModbusPal trabalha com endereço de base igual a 1).

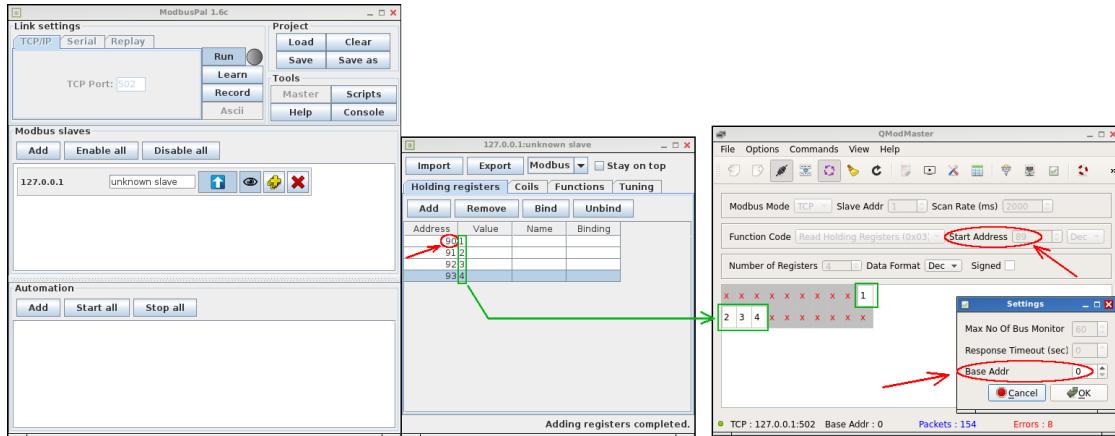


Figura 25.4: QModMaster (Linux) configurado com endereço de base igual a 0.

A Figura 25.5 apresenta o QModMaster (Linux) configurado com endereço de base igual a 1 (o ModbusPal também trabalha com endereço de base igual a 1). Devido ao bug do QModMaster do Linux o deslocamento ficou maior, ao invés de ser corrigido como no Windows.

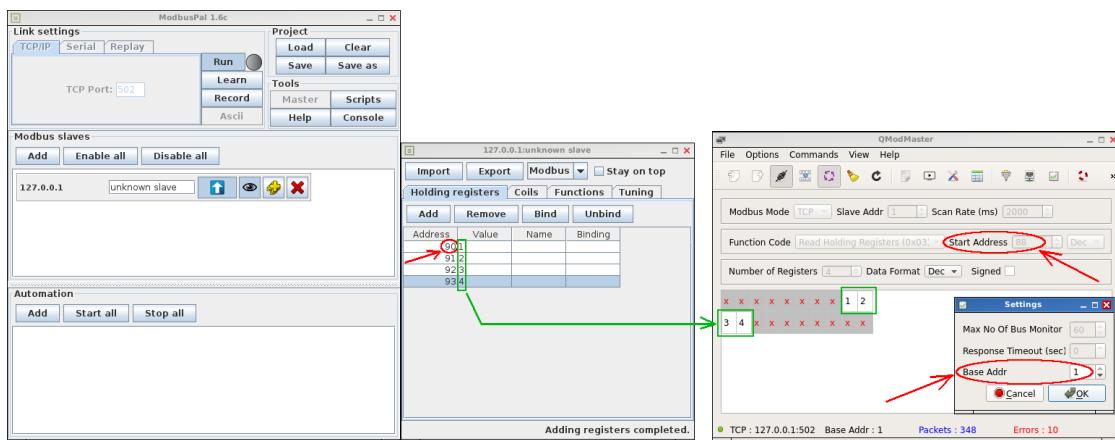


Figura 25.5: QModMaster (Linux) configurado com endereço de base igual a 1. Devido ao bug do QModMaster do Linux o deslocamento foi agravado.

Capítulo 26

Servidor (slave) Modbus no Arduino

Esse capítulo apresenta a implementação de um servidor (slave) Modbus no Arduino com a biblioteca “ArduinoModbus”.

O exemplo deste capítulo foi desenvolvido no Linux com um Arduino virtual simulado pelo SimulIDE e acessado pelo QModMaster, por meio de portas com virtuais criadas com o **socat** (o mesmo pode ser feito no Windows substituindo o **socat** pelo programa **com0com**).

26.1 Instalação da Biblioteca

A Figura 26.1 apresenta a instalação da biblioteca “ArduinoModbus”.

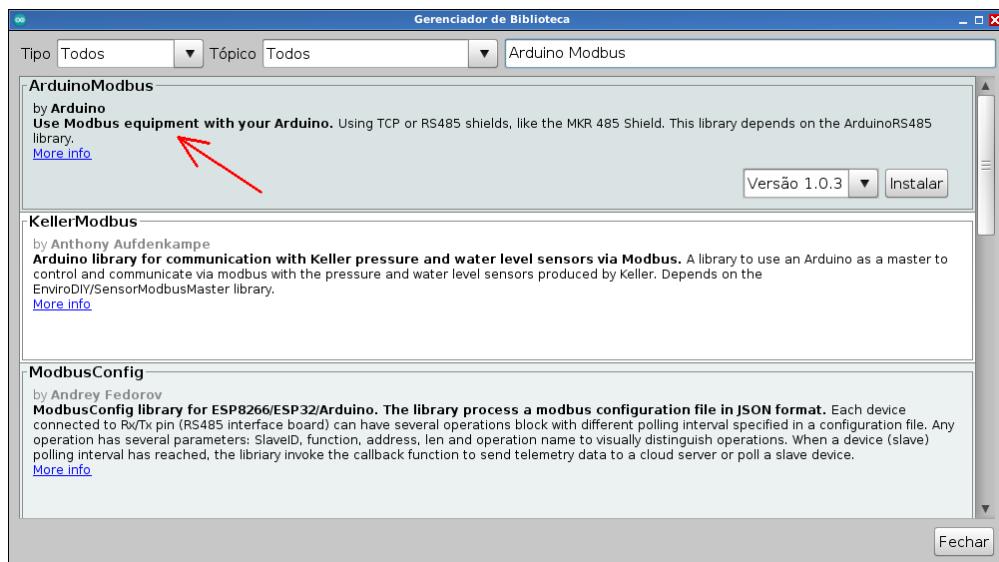


Figura 26.1: Instalação da biblioteca “ArduinoModbus”.

26.2 Circuito simulado

A Figura 26.2 apresenta o circuito simulado no SimulIDE com o Arduino executando o código do servidor (slave) Modbus.

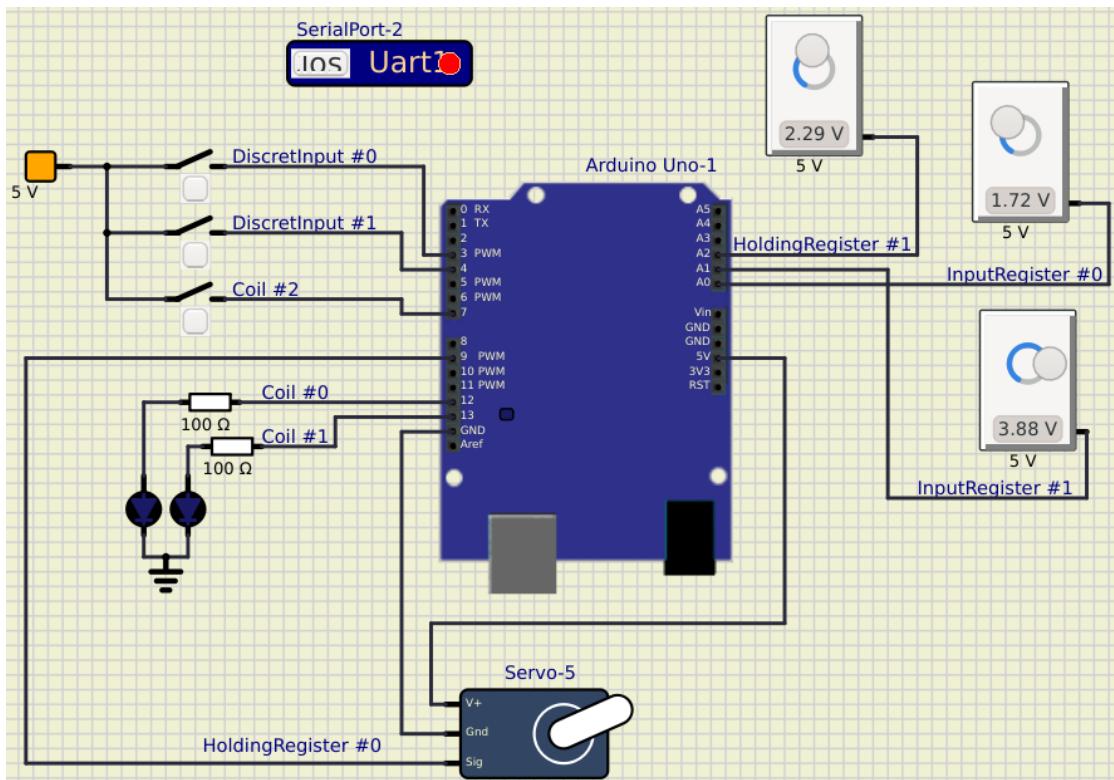


Figura 26.2: Circuito simulado.

26.3 Código executado no Arduino

Conteúdo programa executado no Arduino, arquivo “ModbusRTUServerKitchenSink.ino”.

```

1 // Alex Brandão Rossow
2 // Baseado no exemplo da biblioteca ArduinoModbus
3 // Disponível em:
4 // https://github.com/arduino-libraries/ArduinoModbus/blob/master/examples/RTU/
      ModbusRTUServerKitchenSink/ModbusRTUServerKitchenSink.ino
5
6 /*
7  Modbus RTU Server (Escravo) Kitchen Sink
8  This sketch creates a Modbus RTU Server and demonstrates
9  how to use various Modbus Server APIs.
10 Circuit:

```

```
11   - MKR board
12   - MKR 485 shield
13   - ISO GND connected to GND of the Modbus RTU server
14   - Y connected to A/Y of the Modbus RTU client
15   - Z connected to B/Z of the Modbus RTU client
16   - Jumper positions
17     - FULL set to OFF
18     - Z \/\ Y set to OFF
19 created 18 July 2018
20 by Sandeep Mistry
21 */
22
23 // ATENÇÃO:
24 // Existem pinos reservados pela biblioteca "ArduinoRS485"
25 // (que é requisito para a biblioteca "ArduinoModbus").
26 // Acredito que seja para comandar o conversor TTL/RS-485
27 // ligado ao Arduino para conexão ao barramento RS-485,
28 // que possui pinos para controle da transmissão/recepção.
29 // O pino 2 é um exemplo de pino reservado. Portanto, não deve
30 // se usado. Os pinos reservados podem ser encontrados no arquivo
31 // "RS485.h" na pasta das libs do Arduino IDE ou na página da
32 // biblioteca no Github:
33 // https://github.com/arduino-libraries/ArduinoRS485/blob/master/src/RS485.h
34
35 #include <ArduinoRS485.h> // ArduinoModbus depends on the ArduinoRS485 library
36 #include <ArduinoModbus.h>
37
38 #include <Servo.h> // Para comando do servomotor
39
40 // #####
41 // ##### Variáveis do Modbus #####
42 // #####
43
44 int slaveID; // Endereço do servidor (slave)
45 const int numCoils = 3;
46 const int numDiscreteInputs = 2;
47 const int numHoldingRegisters = 2;
48 const int numInputRegisters = 2;
49 // #####
50 // #####
51
52
53 // #####
54 // ##### Variáveis do Servomotor #####
55 // #####
56 int posServo = 0; // variable to store the servo position
```

```
57 long posServoModbus = 0;
58 Servo myservo; // create servo object to control a servo
59 // #####
60 // #####
61
62 // #####
63 // ##### Variáveis para leitura analógica #####
64 // #####
65 // #####
66 int pot_pin0 = A0; // Pino para leitura analógica (potenciômetro) (InputRegister #0)
67 int pot_pin1 = A1; // Pino para leitura analógica (potenciômetro) (InputRegister #1)
68 int pot_pin2 = A2; // Pino para leitura analógica (potenciômetro) (HoldingRegister #1)
69
70 int coilValue = 0; //Variável auxiliar, recebe os valores dos coils
71 // #####
72 // #####
73
74
75
76 // #####
77 // ##### Variáveis digitais (E/S) #####
78 // #####
79 // Não usar o pino 2, ver nota de atenção no início deste arquivo.
80 const int buttonPin1 = 3; // Pino do botão 1 (DiscreteInput #0)
81 const int buttonPin2 = 4; // Pino do botão 2 (DiscreteInput #1)
82 const int buttonPin3 = 7; // Pino do botão 2 (Coil #3)
83
84
85 const int ledPin0 = 12; //pino para o LED0 (Coil #0)
86 const int ledPin1 = 13; //pino para o LED1 (LED onboard) (Coil #1)
87
88
89 int buttonPin1_State = 0;
90 int buttonPin2_State = 0;
91 // #####
92 // #####
93
94
95
96 void setup() {
97
98 // Configuração do Servomotor #####
99 myservo.attach(9); // define o pino 9 para o sinal
100 // de controle do servomotor
101 // #####
```

```
103 // Configuração dos elementos discretos #####
104 // inicializa os pinos dos LEDs como "OUTPUT":
105 pinMode(ledPin0, OUTPUT);
106 pinMode(ledPin1, OUTPUT);
107
108 // Inicializa os pinos dos botões como "INPUT":
109 pinMode(buttonPin1, INPUT);
110 pinMode(buttonPin2, INPUT);
111 pinMode(buttonPin3, INPUT);
112 // #####
113
114
115 Serial.begin(9600);
116 while (!Serial);
117
118 Serial.println("Modbus RTU Server Kitchen Sink");
119
120 // start the Modbus RTU server, with (slave) id 42
121 slaveID = 42;
122 if (!ModbusRTUServer.begin(slaveID, 9600)) {
123   Serial.println("Failed to start Modbus RTU Server!");
124   while (1)
125   {
126     Serial.println("Falhour em iniciar o Modbus RTU Server!');");
127   }
128 }
129
130 // ### Configura as bobinas #####
131 // configure coils at address 0x00
132 ModbusRTUServer.configureCoils(0x00, numCoils);
133 // #####
134
135
136 // ##### Configura as entradas discretas #####
137 // configure discrete inputs at address 0x00
138 ModbusRTUServer.configureDiscreteInputs(0x00, numDiscreteInputs);
139 // #####
140
141
142 // ##### Configura os holding registers #####
143 // configure holding registers at address 0x00
144 ModbusRTUServer.configureHoldingRegisters(0x00, numHoldingRegisters);
145 // #####
146
147
148 // ##### Configura os input registers #####
```

```
149 // configure input registers at address 0x00
150 ModbusRTUServer.configureInputRegisters(0x00, numInputRegisters);
151 // #####
152 }
153
154
155
156 void loop() {
157
158 // poll for Modbus RTU requests
159 // Entendimento:
160 // O comando "ModbusRTUServer.poll()" é responsável pelo
161 // acesso do mestre a todos registradores (de todos os
162 // tipos) do escravo (leitura e escrita).
163
164 // Os demais comandos Modbus presentes no loop são para o
165 // escravo acessar os próprios registradores (com um comando
166 // próprio para cada tipo de registrador).
167 ModbusRTUServer.poll();
168
169
170 // #####
171 // Processa os "coils" Modbus #####
172 // Fluxo da informação: (ESCRAVO) <==> (MESTRE) #####
173 // #####
174 // map the coil values to the discrete input values
175 for (int i = 0; i < numCoils; i++) {
176
177     switch (i) {
178         case 0: // acesso ao "coil #0" (recebe um valor do mestre)
179             coilValue = ModbusRTUServer.coilRead(i);
180             if (coilValue == -1) // Erro de leitura
181                 break;
182             if (coilValue == 1)
183                 digitalWrite(ledPin0, HIGH);
184             else
185                 digitalWrite(ledPin0, LOW);
186             break;
187
188
189         case 1: // acesso ao "coil #1" (recebe um valor do mestre)
190             coilValue = ModbusRTUServer.coilRead(i);
191             if (coilValue == -1) // Erro de leitura
192                 break;
193             if (coilValue == 1)
194                 digitalWrite(ledPin1, HIGH);
```

```
195     else
196         digitalWrite(ledPin1, LOW);
197     break;
198
199
200     case 2: // acesso ao "coil #2" (envia valor para o mestre)
201     int valorEntrada = digitalRead(buttonPin3);
202     if (valorEntrada == 1)
203         ModbusRTUServer.coilWrite(i, HIGH);
204     else
205         ModbusRTUServer.coilWrite(i, LOW);
206     break;
207
208     default:
209     break;
210 }
211 }
212 // #####
213 // #####
214
215
216
217 // #####
218 // Processa os "discretInputs" Modbus #####
219 // Fluxo da informação: (ESCRAVO) ==> (MESTRE) #####
220 // #####
221 buttonPin1_State = digitalRead(buttonPin1);
222 buttonPin2_State = digitalRead(buttonPin2);
223 for (int i = 0; i < numDiscreteInputs; i++) {
224
225     switch (i) {
226     case 0: // valor escrito no "discreteInput #0")
227         if (buttonPin1_State == HIGH){
228             ModbusRTUServer.discreteInputWrite(i, 1); //Envia o valor "1" para o mestre
229         }
230         else{
231             ModbusRTUServer.discreteInputWrite(i, 0); //Envia o valor "0" para o mestre
232         }
233         break;
234
235     case 1: // valor escrito no "discreteInput #1")
236         if (buttonPin2_State == HIGH){
237             ModbusRTUServer.discreteInputWrite(i, 1); //Envia o valor "1" para o mestre
238         }
239         else{
240             ModbusRTUServer.discreteInputWrite(i, 0); //Envia o valor "0" para o mestre
```

```
241     }
242     break;
243 
244     default:
245         break;
246     }
247 }
248 // #####
249 // #####
250 
251 
252 // #####
253 // Processa os "HoldingRegisters" Modbus #####
254 // Fluxo da informação: (ESCRIVO) <==> (MESTRE)
255 // Nota: A função "ModbusRTUServer.holdingRegisterRead" parece
256 // incompatível com o "switch" (não achei outra explicação
257 // para os cases depois do "case 0" não executarem).
258 // Troquei o "switch" por "if" por essa incompatibilidade.
259 // #####
260 
261 // map the holding register values to the input register values
262 for (int i = 0; i < numHoldingRegisters; i++) {
263 
264     if (i==0){ // acessa o "HoldingRegister #0" (recebe um valor do mestre)
265         long holdingRegisterValue = ModbusRTUServer.holdingRegisterRead(i);
266         if (holdingRegisterValue != -1){ // Verifica se não houve erro de leitura
267             posServoModbus = holdingRegisterValue;
268         }
269     }
270 }
271 
272 else if (i==1){ // acessa o "HoldingRegister #1" (envia um valor para o mestre)
273     ModbusRTUServer.holdingRegisterWrite(i, analogRead(pot_pin2));
274 }
275 }
276 // #####
277 // #####
278 
279 
280 // #####
281 // Processa os "inputRegisters" Modbus #####
282 // Fluxo da informação: (ESCRIVO) ==> (MESTRE)
283 // #####
284 for (int i = 0; i < numInputRegisters; i++) {
285     switch (i) {
```

```
287     case 0: // valor escrito no "InputRegister #0")
288         ModbusRTUserver.inputRegisterWrite(i, analogRead(pot_pin0)); //Envia o valor de
289             A0 para o mestre
290         break;
291
291     case 1: // // valor escrito no "InputRegister #1")
292         ModbusRTUserver.inputRegisterWrite(i, analogRead(pot_pin1)); //Envia o valor de
293             A1 para o mestre
294         break;
295
295     default:
296         break;
297     }
298 }
299 // #####
300 // #####
301
302
303
304 posServo = (int) posServoModbus; // para variar de 0 a 180 graus
305 myservo.write(posServo); // tell servo to go to position in variable 'pos'
306 //for (int i = 0; i < numHoldingRegisters; i++) {
307 // Serial.print("Executando escravo");
308 //}
309 }
```

26.4 Testes de comunicação

Esta seção apresenta os testes de comunicação com uso de um Arduino simulado através do SimulIDE.

Os teste desta seção são feitos com uso do QModMaster, que é um cliente(mestre) Modbus. Em uma aplicação prática, o QModMaster poderia ser substituído por outro dispositivo com função de cliente Modbus (microcontrolador ou CLP) ou por um sistema supervisório, como o Elipse E3 ou ScadaBR, por exemplo.

Uso com um Arduino real:

O mesmo programa (sem nenhuma adaptação) pode ser usado para comunicação com um Arduino real. Nesse caso, deve-se utilizar o endereço da porta serial do Arduino conectado ao computador para a configuração no QModMaster.

No **Linux** a porta para acesso deve ser algo tipo `/dev/ttyUSB0`. É possível utilizar o HTerm para listar as portas disponíveis, conectando o Arduino ao computador e pressionando o botão de refresh no HTerm, conforme apresentado na Figura 26.3.

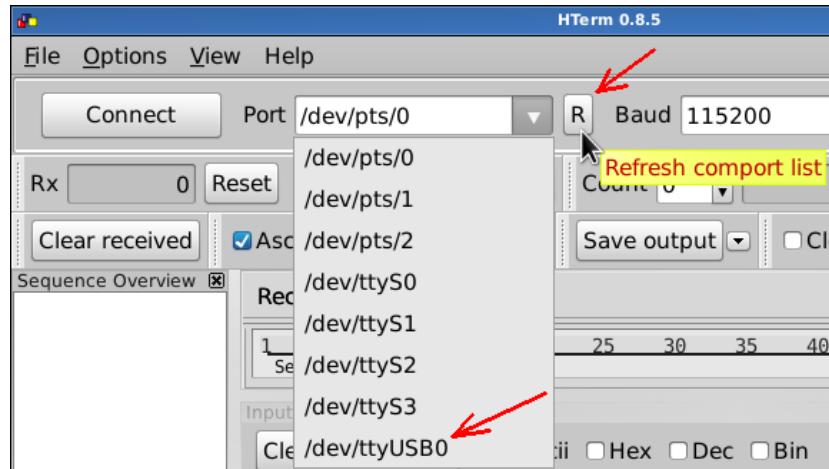


Figura 26.3: Descobrindo a porta serial para acesso a um Arduino real com o HTerm, no Linux.

A Figura 26.4 apresenta a configuração para a comunicação com um Arduino real no Linux.

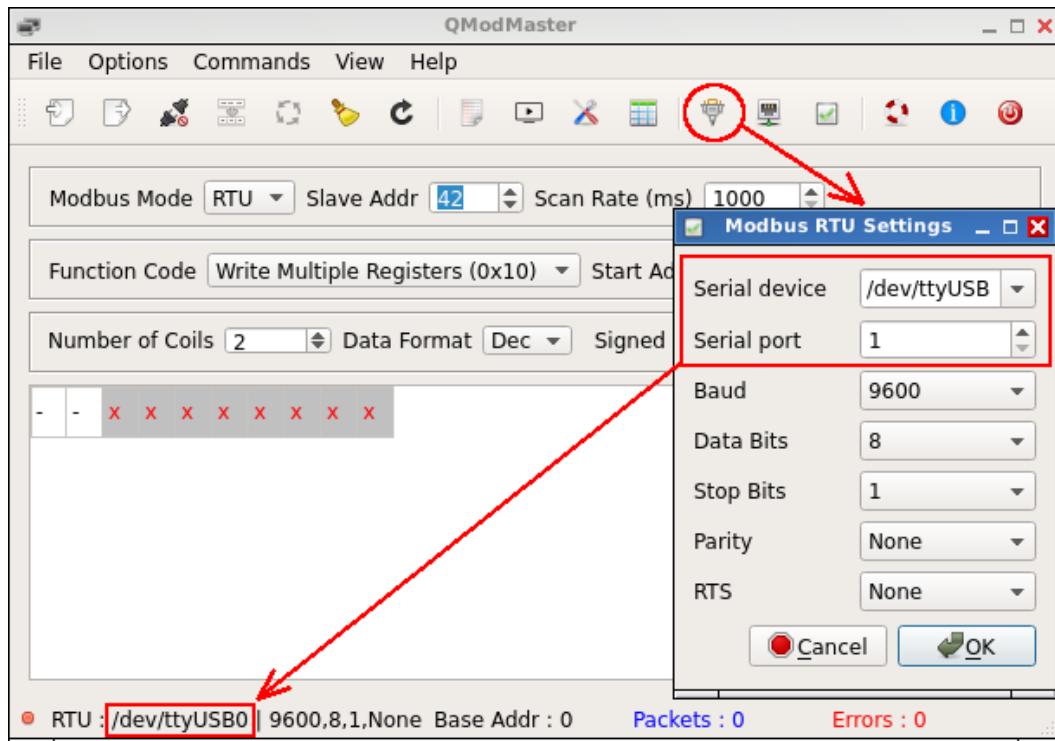


Figura 26.4: Configuração do QModMaster, no Linux, para acesso a um Arduino real.

No Windows pode-se descobrir a porta COM do Arduino através do **Gerenciador de Dispositivos** ou com o HTerm (de modo semelhante ao feito no Linux).

26.4.1 Configuração da comunicação do SimulIDE com QModMaster no Windows

Essa seção apresenta a configuração da comunicação entre o SimulIDE e o QModMaster.

A Figura 26.5 apresenta a configuração das duas portas no com0com. Os detalhes de instalação e configuração do com0com estão na Seção 17.2.

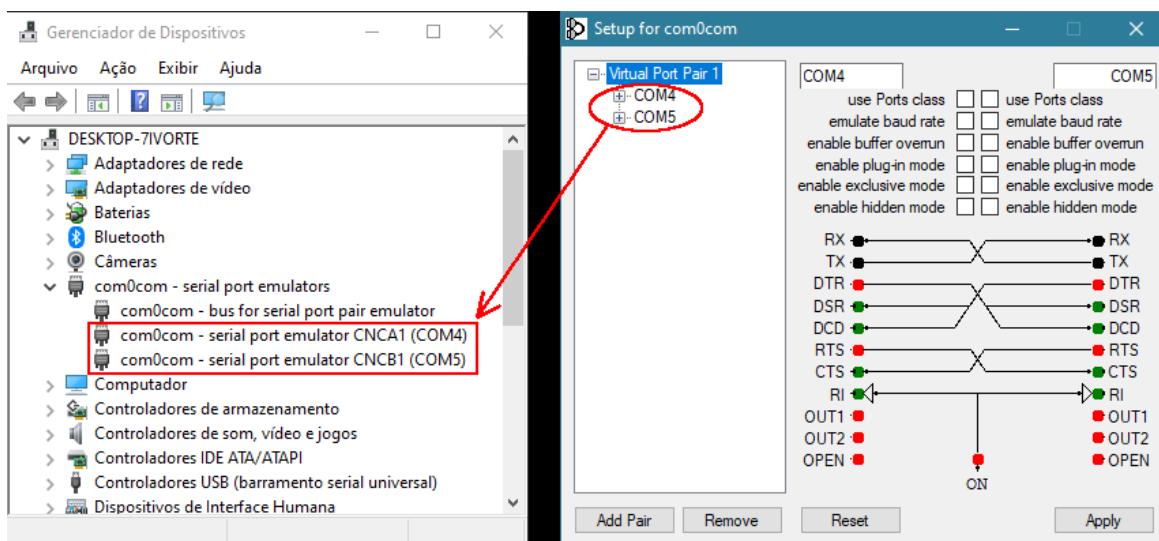


Figura 26.5: Configuração das portas seriais no com0com.

A Figura 26.6 apresenta a configuração da porta serial no SimulIDE, acessível pelos seguintes passos:

1. Abrir a porta serial (janela “Uart”) clicando com o botão direito do mouse sobre o Arduino e selecionar o item “**Open Serial Port**”.
2. Clicar com o botão direito do mouse sobre a janela “Uart” e selecionar o item “**Properties**”.

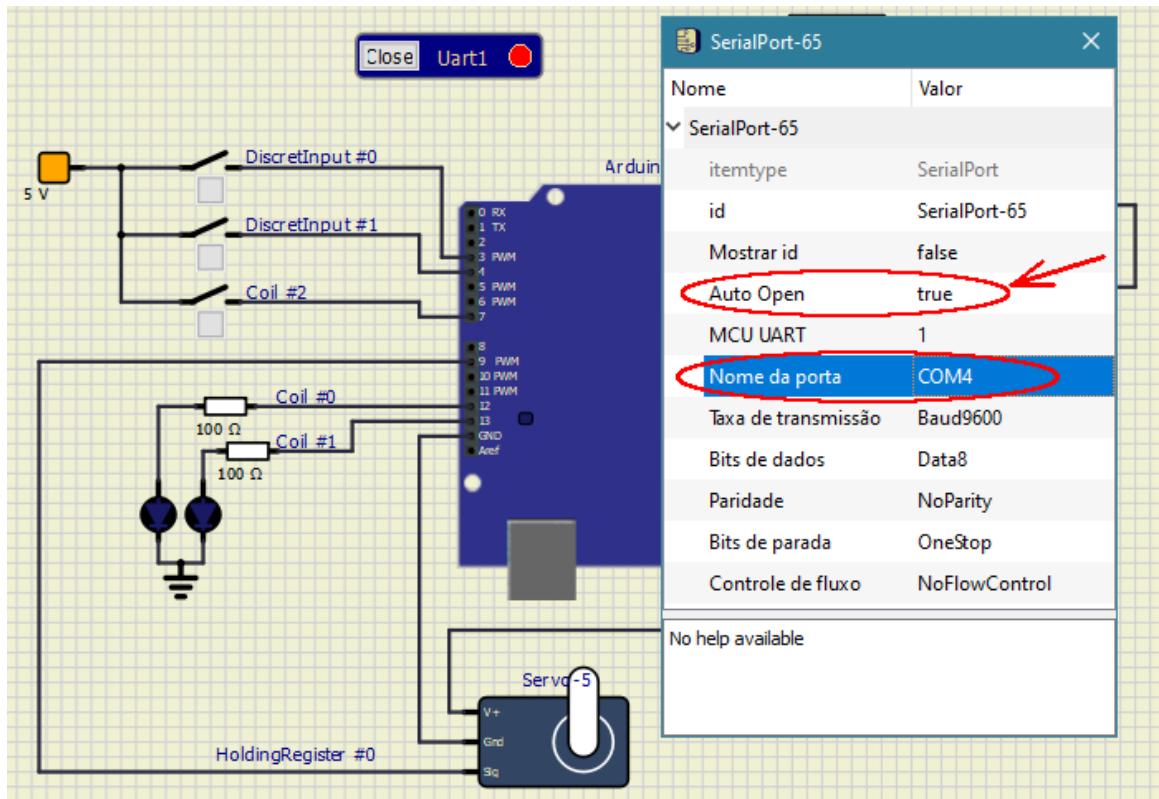


Figura 26.6: Configuração da porta serial no SimuliIDE no Windows.

A Figura 26.7 apresenta a configuração da porta serial no QModMaster.

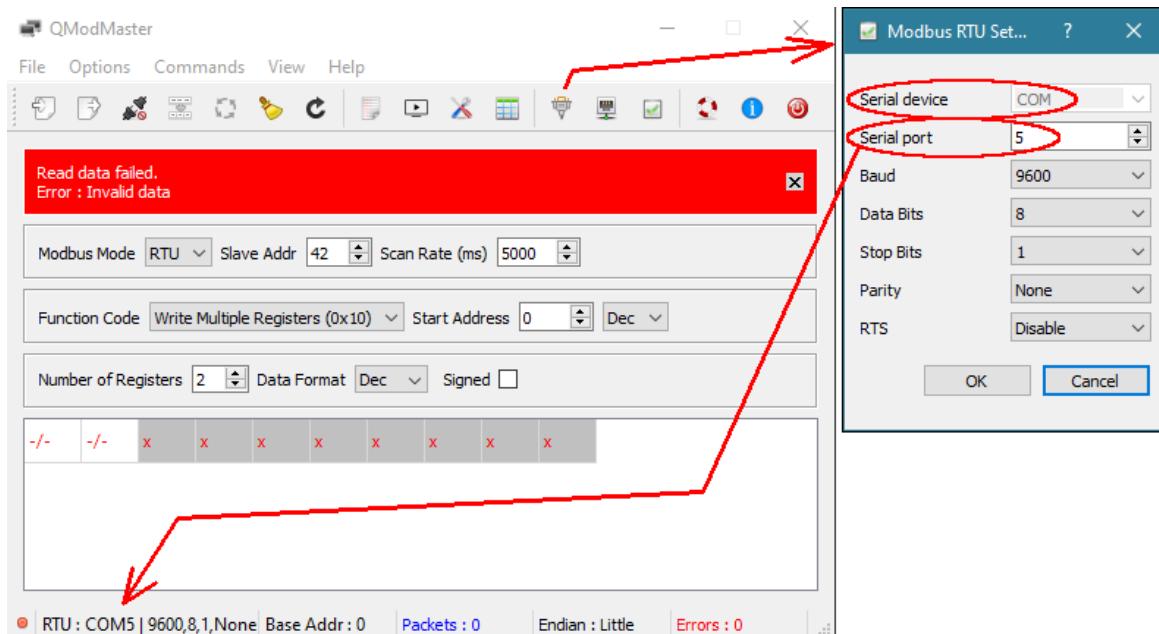


Figura 26.7: Configuração da porta serial no QModMaster no Windows.

A Figura 26.8 apresenta o teste de comando do servomotor pelo Holding Register número 0.

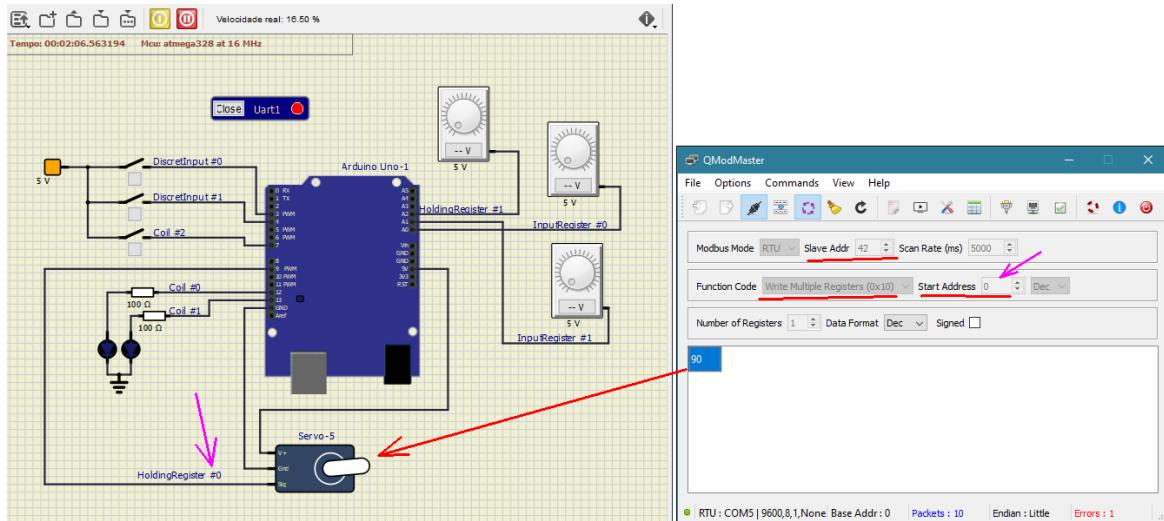


Figura 26.8: Teste de comunicação, comando do servomotor no Windows.

26.4.2 Configuração da comunicação com portas virtuais no Linux

A Figura 26.9 apresenta o início das portas seriais virtuais conectadas com uso do programa **socat**. Na figura são destacados os endereços das duas portas seriais virtuais criadas, que devem ser configuradas no SimulIDE e no QModMaster.

```
alex@PC-Linux: ~/bin
Arquivo Editar Ver Pesquisar Terminal Ajuda
(base) alex@PC-Linux:~/bin$ socat -d -d pty,raw,echo=0 pty,raw,echo=0
2021/07/04 22:57:05 socat[92833] N PTY is [/dev/pts/1]
2021/07/04 22:57:05 socat[92833] N PTY is [/dev/pts/2]
2021/07/04 22:57:05 socat[92833] N starting data transfer loop with FDs [5,5] and [7,7]
```

A red arrow points to the second line of output, highlighting the path '/dev/pts/2'.

Figura 26.9: Início das portas seriais virtuais conectadas no Linux.

A Figura 26.10 apresenta a configuração da comunicação serial no SimulIDE, usando uma das portas seriais virtuais.

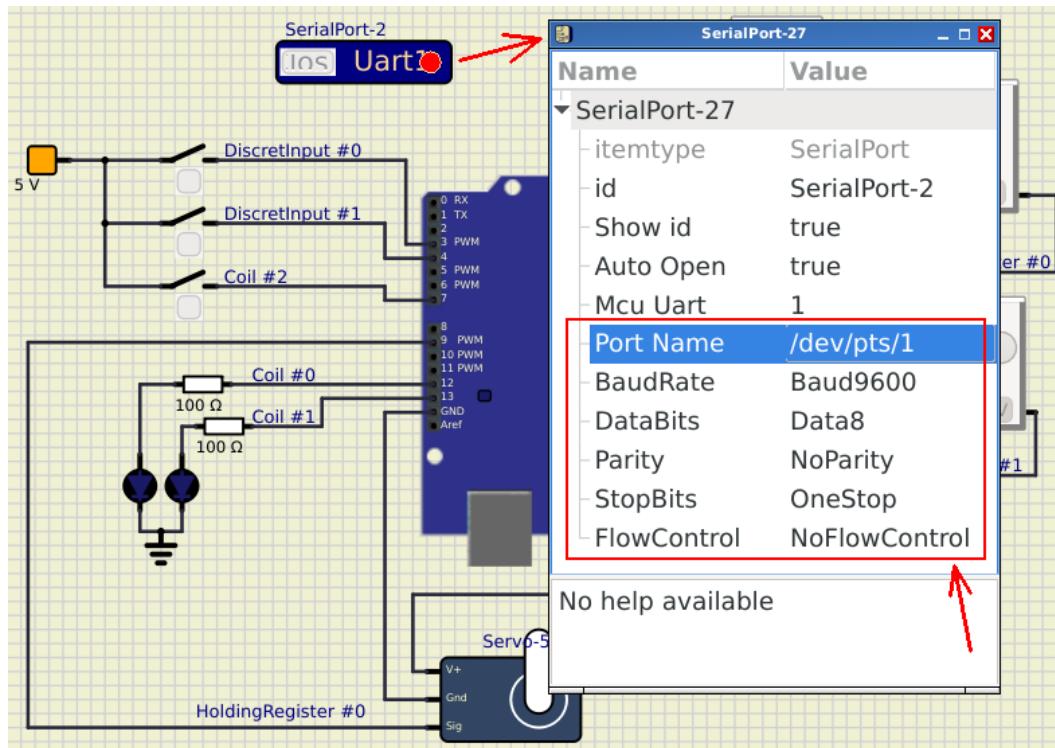


Figura 26.10: Configuração da comunicação serial no SimulIDE.

A Figura 26.11 apresenta a configuração da comunicação serial no QModMaster, usando a outra porta serial virtual.

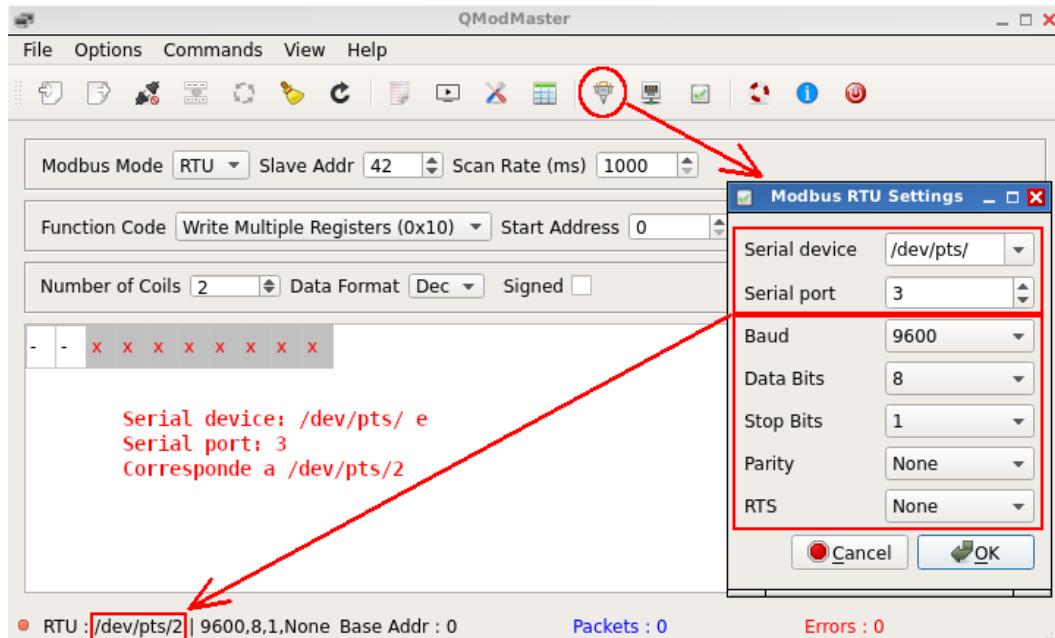


Figura 26.11: Configuração da comunicação serial no QModMaster (deve-se inserir a barra "/" depois de pts no campo "Serial device").

26.4.3 Leitura de entradas discretas (discret input)

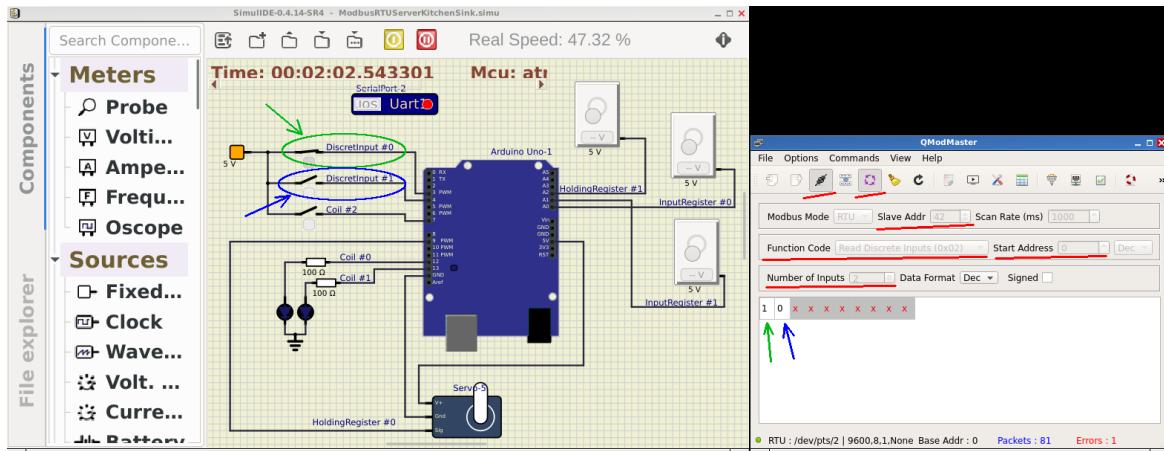


Figura 26.12: Leitura de entradas discretas (discret input).

26.4.4 Escrita em bobinas (coils)

Neste teste, os valores são escritos no QModMaster, comandando os leds ligados ao Arduino.

O campo “Data Format” no QModMaster está com a configuração “Dec”, mas poderia estar configurado para “Bin”, visto que as bobinas armazenam valores binários.

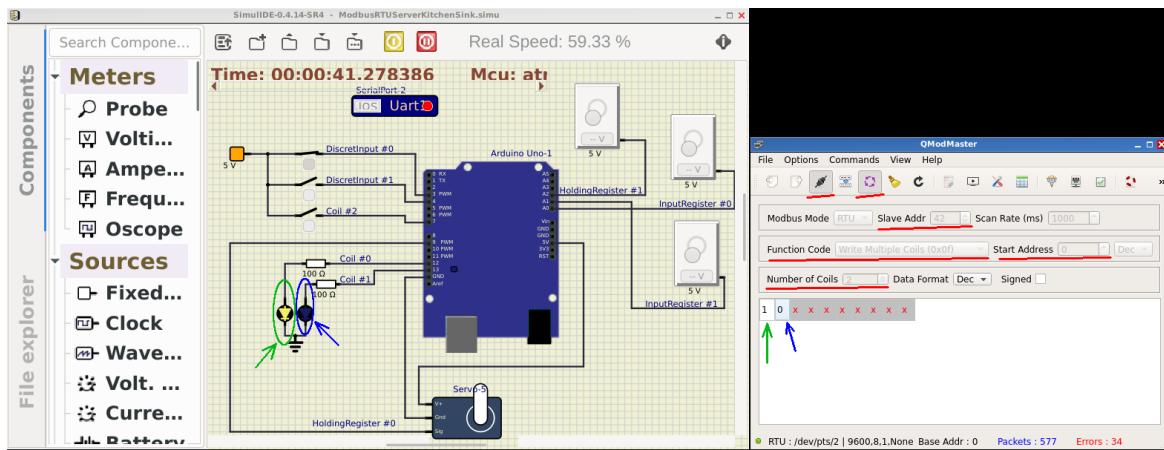


Figura 26.13: Escrita em bobinas (coils).

26.4.5 Leitura de bobinas (coils)

Neste teste os valores das bobinas são lidos. Observa-se que as bobinas que comandam os leds, escritas anteriormente, estão sendo lidas aqui.

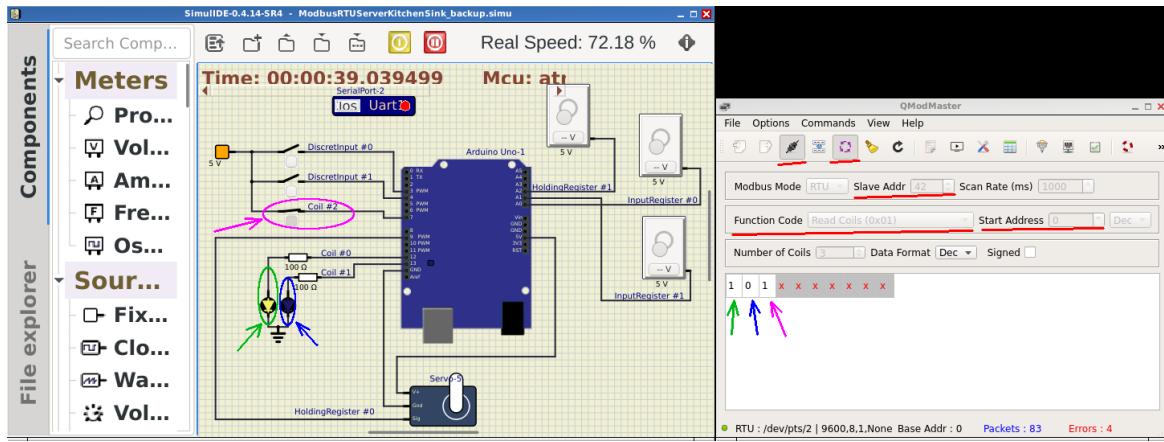


Figura 26.14: Leitura de bobinas (coils).

26.4.6 Leitura de registradores de entrada (input registers)

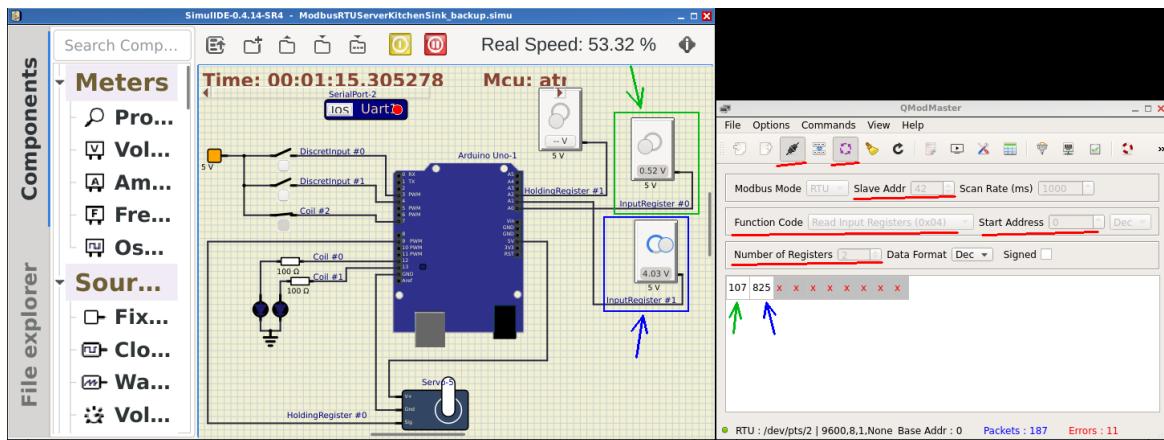


Figura 26.15: Leitura de registradores de entrada (input registers).

26.4.7 Escrita em registradores (holding registers)

Neste teste o valor é escrito no QModMaster e enviado para o Arduino, comandando o servomotor.

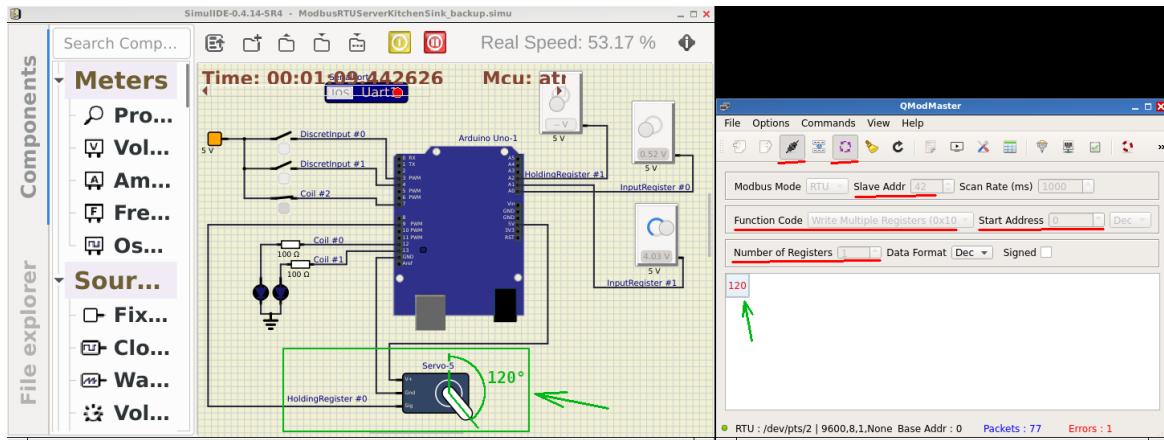


Figura 26.16: Escrita em registradores (holding registers).

26.4.8 Leitura de registradores (holding registers)

Neste teste os valores dos registradores são lidos do Arduino. Observa-se que o valor do registrador de comando do servomotor, escrito anteriormente, está sendo lido.

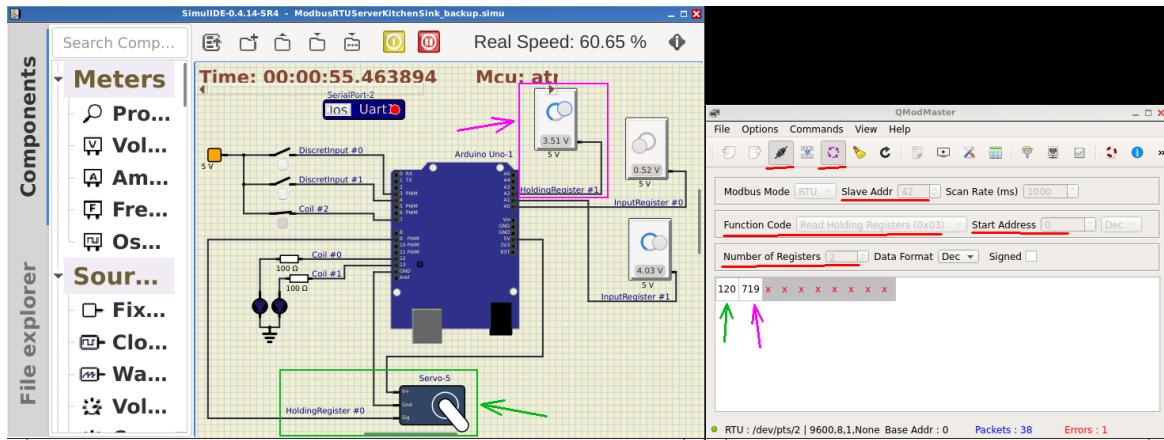


Figura 26.17: Leitura de registradores (holding registers).

Parte V

Experimentos e considerações práticas sobre OPC-UA

Capítulo 27

Introdução ao OPC-UA

Este capítulo apresenta um breve histórico do padrão OPC-UA e suas principais características.

27.1 Apresentação do protocolo OPC-UA

O OPC é um padrão de interoperabilidade para a troca segura e confiável de dados no espaço industrial. Ele é independente de plataforma e garante o fluxo de informações entre dispositivos de vários fornecedores. O padrão OPC é uma série de especificações desenvolvidas por fornecedores do setor, usuários finais e desenvolvedores de software. Essas especificações definem formas de interface entre clientes e servidores, bem como entre vários servidores, incluindo acesso a dados em tempo real, monitoramento de alarmes e eventos, acesso a dados históricos e outros aplicativos OPC FOUNDATION 2021.

Inicialmente, o padrão OPC era restrito ao sistema operacional Windows. Com o surgimento de novas necessidades, foram desenvolvidas as especificações do OPC-UA, que fornece uma arquitetura de plataforma aberta, não se limitando a um único sistema operacional.

Apesar de o OPC-UA tradicional não ser usado para aplicações que demandem grande velocidade, existem novas funcionalidades sendo implementadas no protocolo que visam preencher essa lacuna. Uma das novas capacidades em estudo é a troca de mensagens utilizando o protocolo UDP como camada de transporte, em uma estrutura chamada PubSub (referência a Publish-Subscribe) Unified Automation 2018.

27.2 Biblioteca Open62541

O open62541 é uma implementação em código livre do OPC-UA. É uma biblioteca compatível com os principais compiladores e que fornece as ferramentas necessárias para a implementação de clientes e servidores OPC-UA dedicados, ou para incluir a capacidade de comunicação baseada em OPC UA em aplicativos existentes. A biblioteca open62541 é independente da plataforma OPEN62541 2021.

O open62541 pode ser compilado para o sistema RTOS (*Real-Time Operating System*), o que permite seu uso com o microcontrolador ESP32. O processo de compilação gera um arquivo de código “.c” e um de header “.h” que são importados no código do ESP32 para implementar as funções do OPC-UA. Testei com o ESP-IDF e funciona. Não testei com o Arduino-IDE. Nos testes, a versão do github do open62541 não funcionou adequadamente. Portanto, é recomendável a utilização da versão disponibilizada no site oficial da biblioteca (<https://open62541.org/>), pois essa parece ser a versão estável. Os passos para compilação da biblioteca para RTOS são encontrados no manual oficial do open62541.

Pode-se utilizar o open62541 para a criação de blocos personalizados do Xcos, no Scilab, permitindo a comunicação com outros programas por meio do protocolo OPC-UA, como o OpenModelica Rossow 2018.

Capítulo 28

Uso do padrão OPC UA associado com outras redes

Como padrão de camada de aplicação, o OPC UA deve executar sobre algum conjunto de padrões que forneçam as funcionalidades das camadas inferiores, como camada física, camada de enlace de dados e camada de transporte.

O padrão OPC UA pode ser associado com outros padrões de redes de duas formas:

- Com emprego de gateways, passando as informações de uma rede de campo para o padrão OPC-UA.
- Com o OPC UA atuando na camada de transporte de uma rede de campo.

Este capítulo apresenta possibilidades de uso do OPC UA em associação com outros padrões de redes industriais.

28.1 OPC UA associado com PROFINET

Historicamente, os dois protocolos tiveram duas funções distintas. PROFINET é normalmente usado para comunicação de dados em tempo real entre dispositivos de campo e controladores locais. Por outro lado, OPC UA é geralmente usado para se comunicar entre esses controladores e historiadores de nível superior, MES e sistemas SCADA PROFINET University 2021.

Capítulo 29

Exemplo de transmissão de variáveis com OPC UA

Esse exemplo é uma adaptação dos códigos “server-minimal.py” e “client-minimal.py”, presentes no conjunto dos exemplos do projeto “python-opcua”, disponíveis no seguinte link:

<https://github.com/FreeOpcUa/python-opcua/tree/master/examples>

O exemplo está em python, e para sua execução deve-se instalar a biblioteca “opcua”, com um dos comandos a seguir (conforme a versão do python em uso):

Para o python de repositório linux (caso do Debian): <https://freeopcua.github.io/>

```
pip install opcua
```

ou

Para o Anaconda: <https://anaconda.org/conda-forge/opcua> (escolher uma das opções somente para o Anaconda)

```
conda install -c conda-forge opcua
conda install -c conda-forge/label/cf202003 opcua
```

29.1 Servidor OPC UA

Conteúdo do arquivo “01_server-minimal.py”.

```
1 # Alex Brandão Rossow
2 # Adaptado do exemplo disponível em:
3 # https://github.com/FreeOpcUa/python-opcua/blob/master/examples/server-minimal.py
4
5 import sys
6 sys.path.insert(0, "..")
7 import time
8
9
10 from opcua import ua, Server
11
12
13 if __name__ == "__main__":
14
15     # setup our server
16     # Aqui pode ser definido um IP específico, como o "127.0.0.1"
17     # ou o IP da interface de rede (Ethernet ou Wi-Fi).
18     # O IP "0.0.0.0" é uma configuração flexível, onde o servidor
19     # atende em qualquer um dos IPs da máquina (o "127.0.0.1" ou
20     # o IP da interface de rede).
21     server = Server()
22     server.set_endpoint("opc.tcp://0.0.0.0:4840/freeopcua/server/") # servidor
23     #server.set_endpoint("opc.tcp://127.0.0.1:4840/freeopcua/server/") # servidor
24     # atende em qq IP da máquina
25     #atende em um IP específico
26
27     # setup our own namespace, not really necessary but should as spec
28     uri = "http://examples.freeopcua.github.io"
29     idx = server.register_namespace(uri)
30
31     # get Objects node, this is where we should put our nodes
32     objects = server.get_objects_node()
33
34     # populating our address space
35     myobj = objects.add_object(idx, "MyObject")
36     myvar = myobj.add_variable(idx, "MyVariable", 6.7) # "6.7" é o parâmetro "val"
37     myvar.set_writable() # Set MyVariable to be writable by clients
38
39     # Variáveis de Teste
40     # O parâmetro "val" define o valor inicial da variável.
41     # Também está sendo responsável por definir o tipo. Se
```

```
41 # for um número inteiro, a variável aparece como "Int64
42 # no FreeOpcUa Client. Se for um real aparece como
43 # "Double", funcionando como real.
44     myobjTeste1 = objects.add_object(idx, "MyObjectTeste01")
45     myvarTeste1 = myobjTeste1.add_variable(idx, "MyVariableTeste01", val=2)
46     myvarTeste2 = myobjTeste1.add_variable(idx, "MyVariableTeste02", val=3.0)
47     #myvarTeste1.set_writable() # Set MyVariable to be writable by clients
48
49
50     # starting!
51     # Inicia o servidor OPC-UA
52     server.start()
53
54     try:
55         count = 0
56         while True:
57             time.sleep(1)
58             count += 0.1
59             myvar.set_value(count)
60
61             print("Digite o valor da variável myvarTeste1:")
62             try:
63                 myvarTeste1_Valor = int(input())
64                 myvarTeste2_Valor = 2*myvarTeste1_Valor
65                 myvarTeste1.set_value(myvarTeste1_Valor)
66                 myvarTeste2.set_value(myvarTeste2_Valor)
67             except ValueError:
68                 print("Não é um número válido!")
69
70     finally:
71         #close connection, remove subcriptions, etc
72         server.stop()
```

29.2 Cliente OPC UA

Conteúdo do arquivo “02_client-minimal.py”.

```
39         print("myvar is: ", myvar)
40         print("myobj is: ", obj)
41
42
43     # Now getting a variable node using its browse path
44     # O mesmo feito anteriormente, agora com as variáveis de Teste
45     # Para acessar um objeto é passado todo caminho até o objeto
46     # Para acessar uma variável é passado todo caminho até a variável
47     objTeste01 = root.get_child(["0:Objects", "2:MyObjectTeste01"])
48     myvarTeste01 = root.get_child(["0:Objects", "2:MyObjectTeste01", "2:
49                               MyVariableTeste01"])
50     myvarTeste02 = root.get_child(["0:Objects", "2:MyObjectTeste01", "2:
51                               MyVariableTeste02"])
52
53
54
55
56     # Lê e exibe os valores das duas variáveis de teste
57     # de forma contínua.
58     while True:
59         varTeste01_valor = myvarTeste01.get_value();
60         varTeste02_valor = myvarTeste02.get_value();
61         print("_____")
62         print(f"O valor de varTeste01_valor é: {varTeste01_valor}")
63         print(f"O valor de varTeste02_valor é: {varTeste02_valor}")
64         time.sleep(5)
65
66
67     # Stacked myvar access
68     # print("myvar is: ", root.get_children()[0].get_children()[1].get_variables()
69     #       [0].get_value())
70
71     finally:
72         client.disconnect()
```

29.3 Resultado do teste

A Figura 29.1 apresenta o resultado do teste. Neste teste o usuário fornece um valor e o servidor envia duas variáveis para o cliente, a primeira variável com o valor digitado e a segunda com o valor digitado multiplicado por dois.

The screenshot shows two terminal windows side-by-side. The left window (server) runs `python 01_server-minimal.py`. It prints the server's endpoint information, listens on port 4840, and prompts for values for variables `myvarTeste1`. The user inputs 4 (labeled "Primeiro valor digitado") and 5 (labeled "Segundo valor digitado"). The right window (client) runs `python 02_client-minimal.py`. It lists the root node's children, which include `myvar`, `myobj`, and two objects (`objTeste01` and `myvarTeste01`). It then reads values from the server for `varTeste01` and `varTeste02`. The values 2, 4, and 5 are printed, with their respective descriptions ("Valor inicial declarado no servidor OPC UA", "Primeiro valor digitado no servidor OPC UA", and "Segundo valor digitado no servidor OPC UA") highlighted in red.

```

alex@PC-Linux:~/programacao_Alex/OPC_Alex/opcua.github/server_client_alex$ python 01_server-minimal.py
Endpoints other than open requested but private key and certificate are not set.
Listening on 0.0.0.4840
Digite o valor da variável myvarTeste1:
4                                         Primeiro valor digitado
Digite o valor da variável myvarTeste1:
5                                         Segundo valor digitado
Digite o valor da variável myvarTeste1:

alex@PC-Linux:~/programacao_Alex/OPC_Alex/opcua.github/server_client_alex$ python 02_client-minimal.py
Objects node is: i=84
Children of root are: [Node(NumericNodeId(i=85)), Node(NumericNodeId(i=86)), Node(NumericNodeId(i=87))]
myvar is: ns=2;i=2
myobj is: ns=2;i=1
objTeste01 é: ns=2;i=3
myvarTeste01 é: ns=2;i=4
myvarTeste02 é: ns=2;i=5
-----
0 valor de varTeste01_valor é: 2      Valor inicial declarado no servidor OPC UA
0 valor de varTeste02_valor é: 3.0

0 valor de varTeste01_valor é: 4      Primeiro valor digitado no servidor OPC UA
0 valor de varTeste02_valor é: 8

0 valor de varTeste01_valor é: 5      Segundo valor digitado no servidor OPC UA
0 valor de varTeste02_valor é: 10

```

Figura 29.1: Resultado do teste. Os valores digitados na servidor (janela superior) são acessados pelo cliente e exibidos (janela inferior).

É possível que apareçam algumas mensagens de erro na janela do servidor pelo fato de alguns parâmetros não estarem implementados. Nesses casos de erro, pode-se pressionar a tecla Enter para que o programa siga e apresente novamente a mensagem solicitando que o usuário digite o valor desejado para a variável de teste.

29.4 Uso do FreeOpcUa Client

O FreeOpcUa Client é uma ferramenta importante para o desenvolvimento de servidores ou realização de teste com equipamentos que desempenham o papel de servidor OPC UA.

O projeto está disponível no seguinte endereço:

<https://github.com/FreeOpcUa/opcua-client-gui>.

29.4.1 Instalação e execução do FreeOpcUa Client

A instalação é feita com o comando:

```
pip install opcua-client
```

A execução é feita com o comando:

```
opcua-client
```

29.4.2 Exemplo de uso do FreeOpcUa Client

Nesta seção o FreeOpcUa Client é usado para acessar o servidor da Seção 29.1.

A Figura 29.2 apresenta a janela do FreeOpcUa Client configurada para acessar o servidor da Seção 29.1. Para a conexão o servidor já deve estar rodando.

O campo de endereço do servidor, configurado aqui como: “**opc.tcp://0.0.0.0:4840**”, deve seguir a configuração do servidor, se for um servidor local, ou apontar para o endereço IP do dispositivo que está executando o servidor, como outro computador, um instrumento ou microcontrolador ESP32, por exemplo.

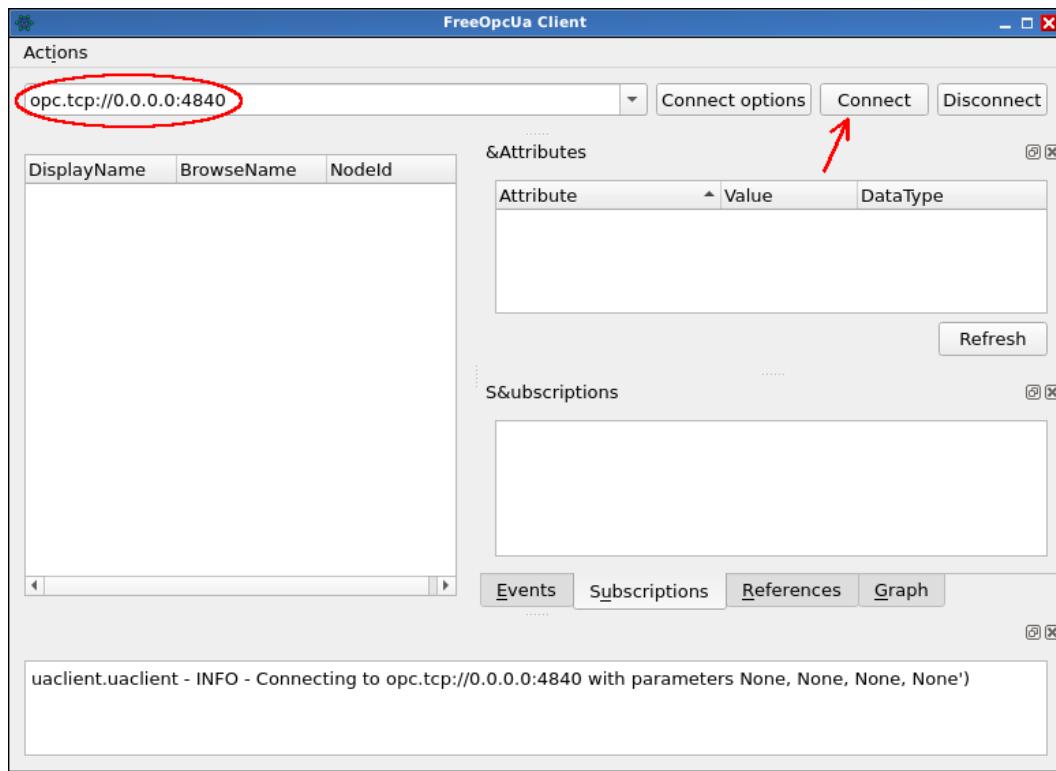


Figura 29.2: Janela do FreeOpcUa Client, destacando-se o campo de configuração do endereço do servidor e o botão de conexão.

A Figura 29.3 apresenta o FreeOpcUa Cliente conectado ao servidor da Seção 29.1. Observa-se nessa figura os elementos do servidor, como objetos e variáveis, na coluna da esquerda. Ao clicar sobre um item do servidor, são exibidas características deste item na coluna da direita. Na figura destacou-se o tipo da variável e seu valor. A figura também destaca o botão de atualização, para que essas informações sejam atualizadas por meio da leitura do servidor.

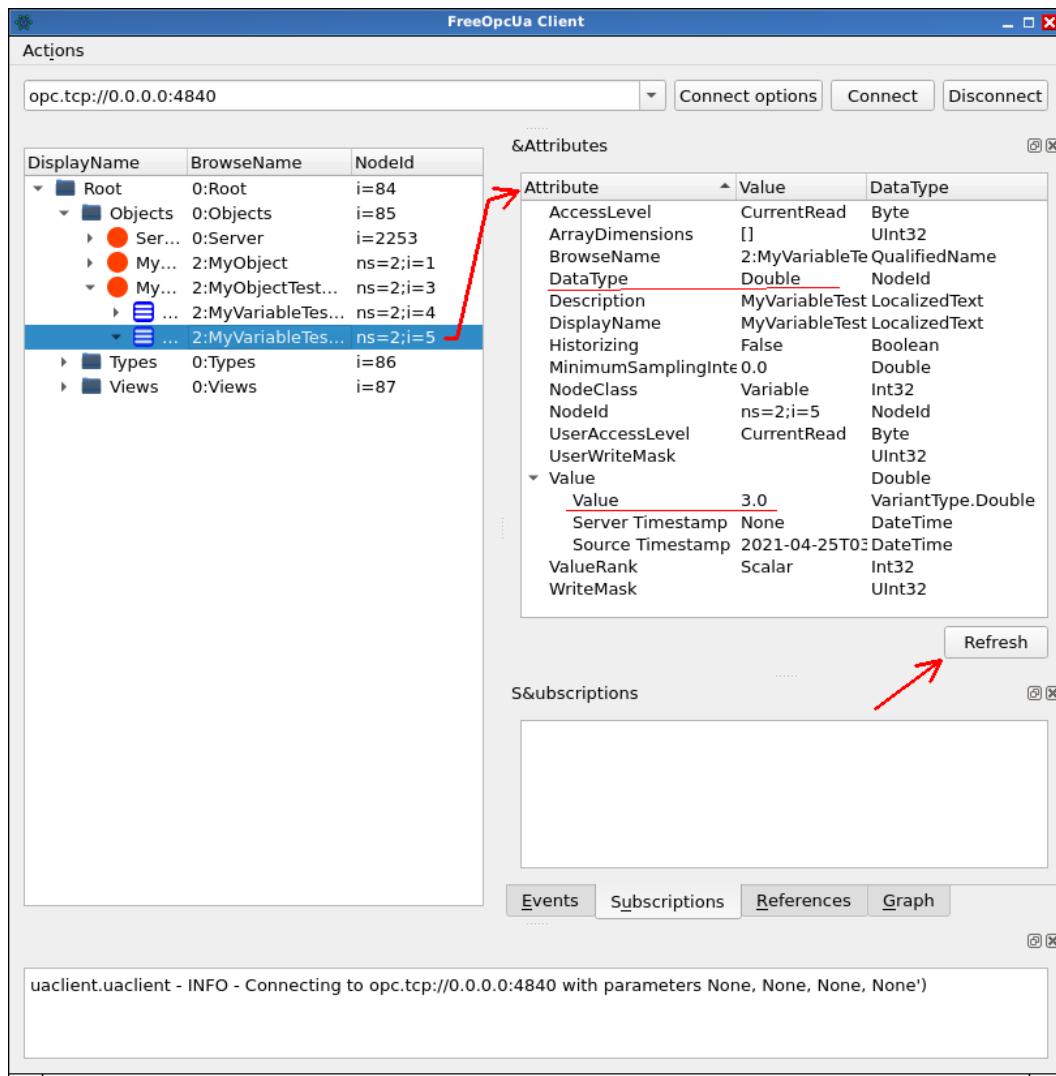


Figura 29.3: FreeOpcUa Client conectado ao servidor OPC UA.

Capítulo 30

Exemplo de aplicação com padrão OPC UA (testes com o Node-RED)

Este capítulo apresenta o acesso a um servidor OPC UA a partir do Node-RED.

O programa possui duas variáveis acessíveis por OPC UA, descritas a seguir:

- amplitude: Valor passado pelo usuário para definição da amplitude de uma onda senoidal criada pelo programa;
- Variavel_Senoide: O valor da variável senoidal criada pelo programa. A amplitude é configurada por uma variável OPC UA. Os demais parâmetros (período de amostragem e frequência) são configurados no código.

Esse programa foi implementado com threads independentes para o tratamento de cada variável OPC UA. Esse não é um requisito do OPC UA, foi apenas uma escolha com objetivo de isolar o tratamento de cada variável em um loop próprio. Isso permite o uso de frequências diferentes para o recebimento do valor de amplitude e para o envio de amostras da senoide criada.

30.1 Código do servidor testado

```
1 # Alex Brandão Rossow
2 # Cria uma onda senoidal em uma variável OPC UA
3 # com amplitude configurada por outra variável
4 # OPC UA
5
6 import sys
7 sys.path.insert(0, "..")
8 import time
9 import math
10
11 from threading import Thread # Import para operar processos paralelos
12
13 from opcua import ua, Server
14
15
16 # Variáveis globais
17 amplitude = 1.0;
18
19
20
21 # Task responsável por receber o valor da amplitude
22 def recebeAmplitude_OPCTask(idx, myobjTestel):
23
24     global amplitude
25
26     print(">>> Entrou em receiveAmplitude_OPCTask")
27     myvarTestel = myobjTestel.add_variable(idx, "amplitude", val=1.0)
28     myvarTestel.set_writable() # Set MyVariable to be writable by clients
29
30     try:
31         while True:
32             time.sleep(1);
33             print(">>>")
34             print(f'>>>amplitude = {amplitude}')
35             try:
36                 amplitude = myvarTestel.get_value()
37             except ValueError:
38                 print("Não é um número válido!")
39         finally:
40             return;
41
42
43
44
```

```
45 # Gera uma senoide e
46 # envia do servidor OPC UA para o cliente
47 def variavel_Senoide_OPCTask(idx, myobjTeste1):
48
49     # O período de amostragem mínimo exibido pelo
50     # opcua-client é 1 segundo.
51     print(">>> Entrou em variavel_Senoide_OPCTask")
52     t = 0; # tempo
53     ts = 1; # período de amostragem em segundos
54     f = 0.05; # frequência em Hz
55     pi = 3.14;
56     periodo = 1/f; # período da senoide em segundos
57     senoide = 0;
58
59     myvarTeste2 = myobjTeste1.add_variable(idx, "Variavel_Senoide", val=0.0)
60
61     try:
62         while True:
63             time.sleep(ts)
64             t += ts;
65             senoide = amplitude*(math.sin(2*pi*f*t));
66             if (t>=periodo):
67                 t = 0;
68             try:
69                 print(f'>>> senoide = {senoide}')
70                 myvarTeste2.set_value(senoide)
71             except ValueError:
72                 print("Erro ao escrever a variável!")
73     finally:
74         return;
75
76
77
78
79 def iniciaOPC():
80     server = Server();
81     server.set_endpoint("opc.tcp://0.0.0.0:4840/freeopcua/server/")
82
83     # setup our own namespace, not really necessary but should as spec
84     uri = "http://examples.freeopcua.github.io"
85     idx = server.register_namespace(uri)
86
87     # get Objects node, this is where we should put our nodes
88     objects = server.get_objects_node()
89     myobjTeste1 = objects.add_object(idx, "MyObjectTeste01")
```

```
91 # Variáveis de Teste
92 # O parâmetro "val" define o valor inicial da variável.
93 # Também está sendo responsável por definir o tipo. Se
94 # for um número inteiro, a variável aparece como "Int64"
95 # no FreeOpcUA Client. Se for um real aparece como
96 # "Double", funcionando como real.
97
98
99 # Inicia o servidor OPC-UA
100 server.start()
101
102 # Declara as tasks
103 # Threads são elementos do Python, não do OPC.
104 t1 = Thread(target=recebeAmplitude_OPCTask, args=(idx, myobjTeste1))
105 t2 = Thread(target=variavel_Senoide_OPCTask, args=(idx, myobjTeste1))
106
107 # Inicia as tasks
108 t1.start()
109 t2.start()
110
111 # Espera todas as tasks finalizarem
112 t1.join()
113 t2.join()
114
115 # Finaliza o servidor OPC UA quando todas as tasks forem finalizadas
116 server.stop()
117
118
119
120 if __name__ == "__main__":
121     iniciaOPC();
```

30.2 Teste do servidor com o FreeOpcua Client

Para desenvolvimento do servidor é conveniente utilizar o programa **FreeOpcUA Client**, comando `opcua-client`, apresentado na Seção 29.4.

A Figura 30.1 apresenta o teste de ajuste da amplitude e visualização da senoide gerada pelo servidor.

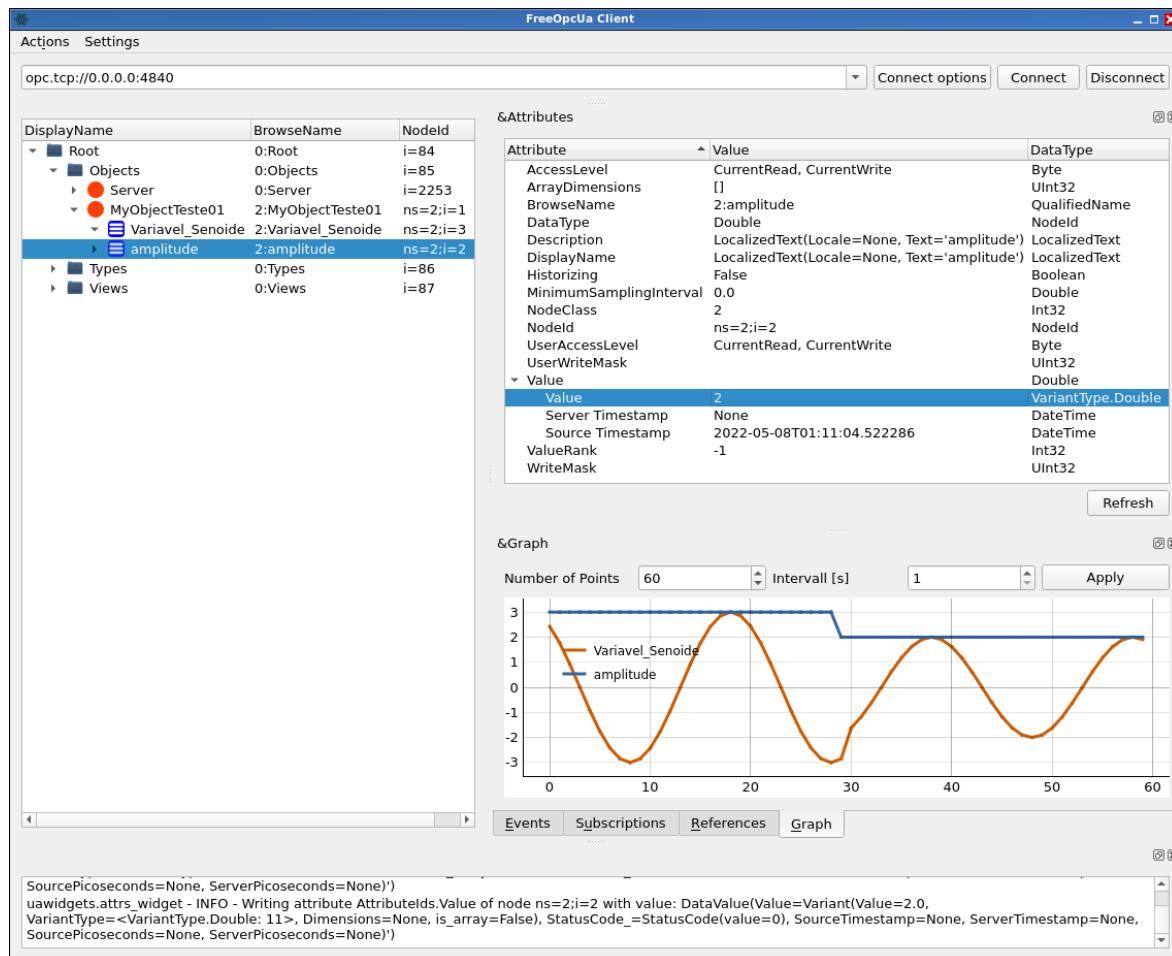


Figura 30.1: Teste do servidor com FreeOpcUa Client.

Para exibir um gráfico deve-se clicar com o botão direito do mouse sobre a variável desejada e selecionar a opção “Add to Graph”, conforme apresentado na Figura 30.2.

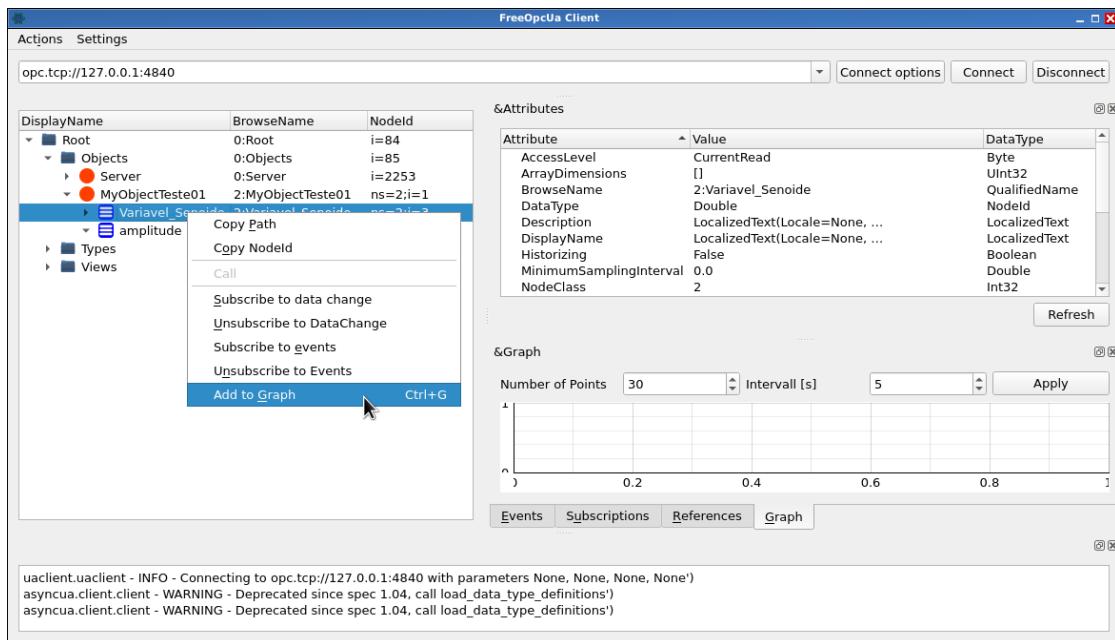


Figura 30.2: Ativação da exibição do gráfico de uma variável.

30.3 Sistema montado no Node-RED

A Figura 30.3 apresenta o sistema montado no Node-RED para interação com o servidor OPC-UA. A primeira linha de blocos é responsável por permitir que o usuário selecione a amplitude da senoide gerada pelo servidor. A segunda linha de blocos é responsável por receber as amostras da senoide gerada pelo servidor.

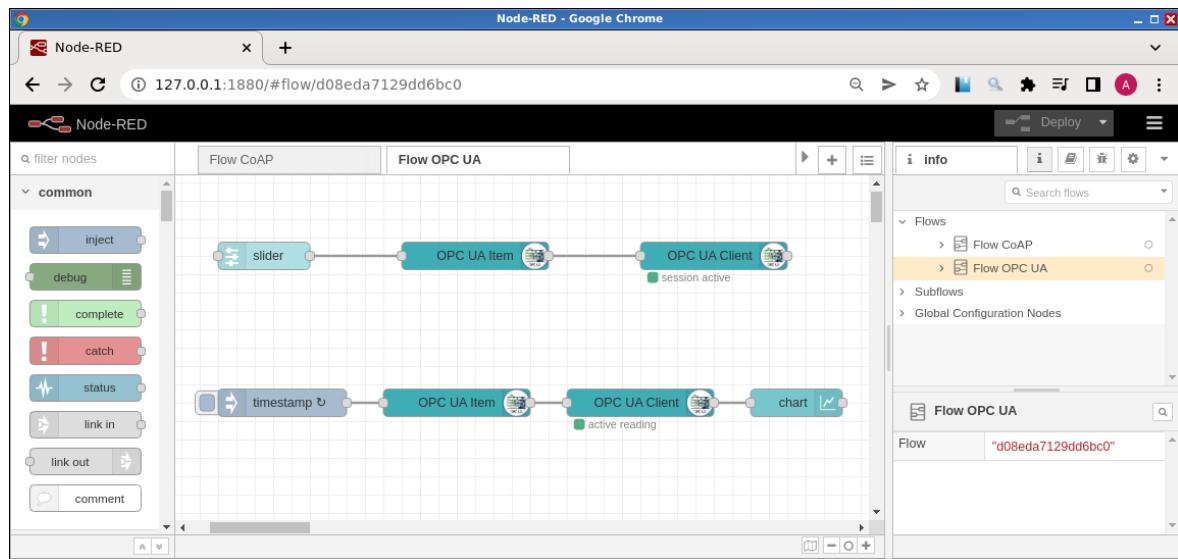


Figura 30.3: Sistema montado no Node-RED para interação com o servidor OPC UA.

30.3.1 Configuração dos blocos para envio da amplitude para o servidor OPC UA

Esta seção apresenta a configuração dos blocos responsáveis pela seleção da amplitude pelo cliente e envio para o servidor OPC UA.

30.3.1.1 Configuração do Slider

O slider foi configurado para variar de 0 a 6 com salto 1, conforme apresentado na Figura 30.4. A configuração do campo “group” contém o nome do dashboard que vai receber o slider.

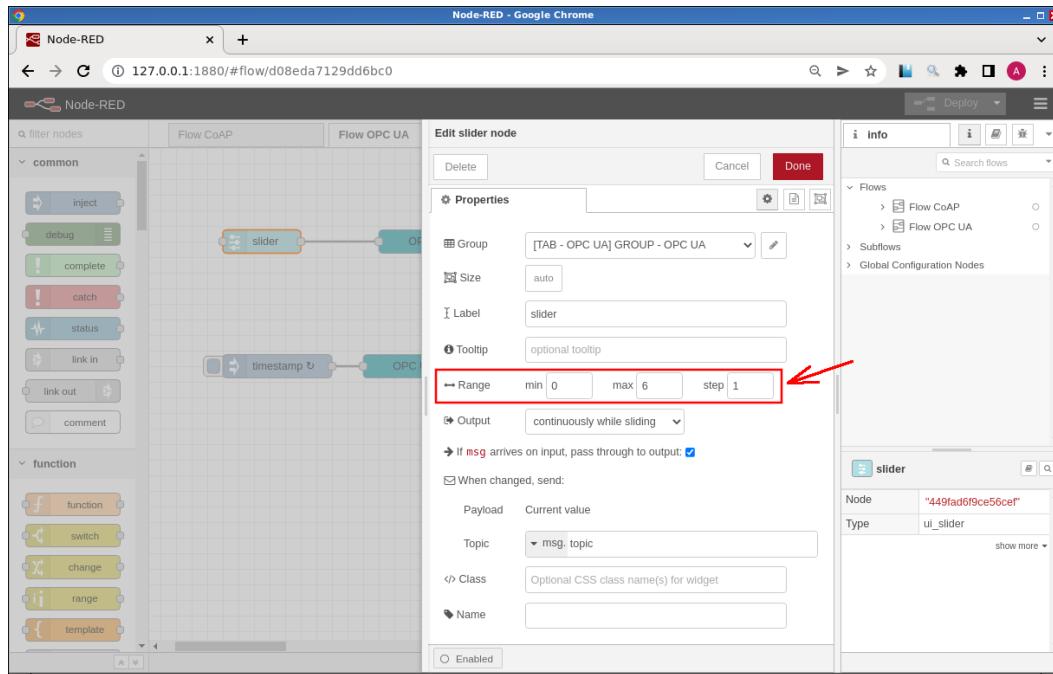


Figura 30.4: Configuração do slider para seleção da amplitude da senoide gerada pelo servidor OPC UA.

30.3.1.2 OPC UA Item para escrita da variável “amplitude”

A Figura 30.5 apresenta a configuração do bloco “OPC UA Item” para escrita da variável “amplitude”. O valor para configuração do campo “Item”, ns=2; i=2, foi obtido pelo FreeOpcUa Client, na coluna “NodeId”, conforme pode ser visto na Figura 30.1.

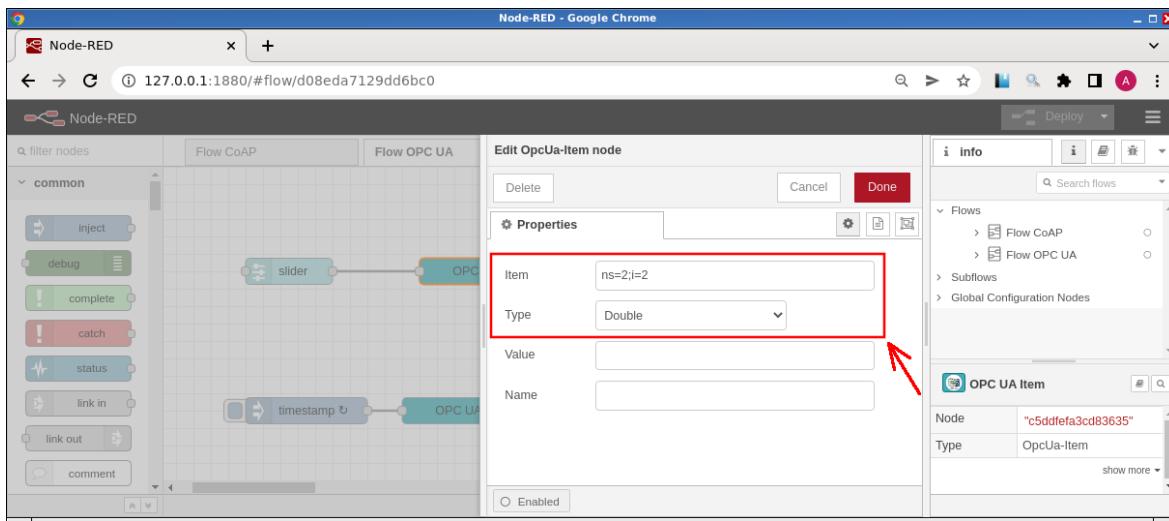


Figura 30.5: Configuração do bloco “OPC UA Item” para escrita da variável “amplitude”.

30.3.1.3 OPC UA Client para escrita da variável “amplitude”

A Figura 30.6 apresenta a configuração do bloco “OPC UA Client” para escrita da variável “amplitude”.

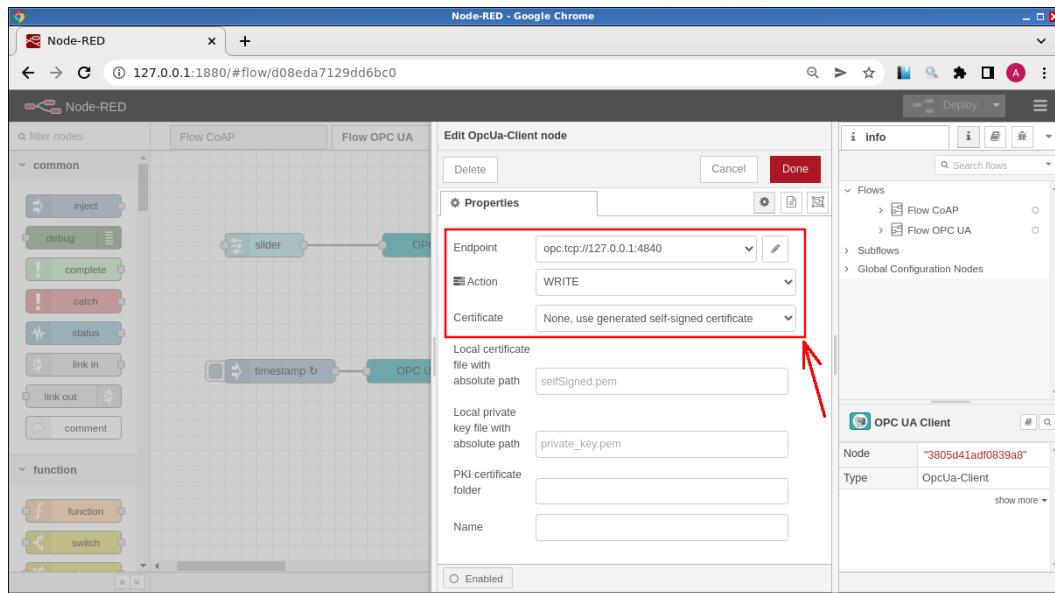


Figura 30.6: Configuração do bloco “OPC UA Client” para escrita da variável “amplitude”.

30.3.2 Configuração dos blocos para leitura da variável “Variavel_Senoide” do servidor OPC UA

Esta seção apresenta a configuração dos blocos para leitura da variável “senoide” gerada pelo servidor OPC UA.

30.3.2.1 Bloco “inject” (timestamp)

A Figura 30.7 apresenta a configuração do bloco “inject”, responsável por disparar a leitura da variável “senoide” com uma frequência de 1 leitura por segundo. Após a configuração do bloco para repetição, o seu nome muda de “inject” para “timestamp”.

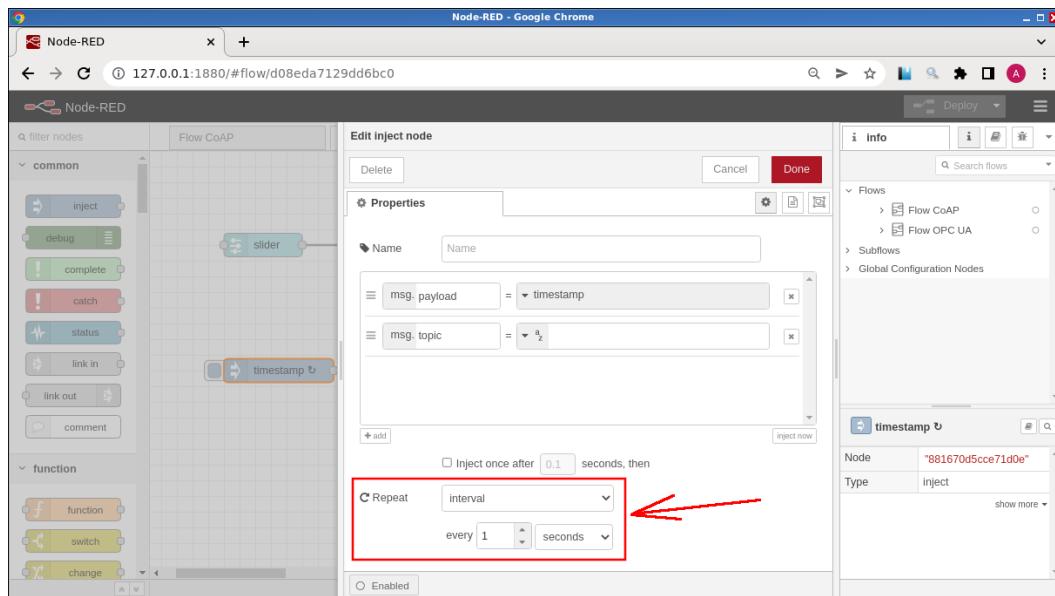


Figura 30.7: Configuração do bloco “inject” para comando da leitura da variável “senoide” a cada 1 segundo.

30.3.2.2 OPC UA Item para leitura da variável “Variavel_Senoide”

A Figura 30.8 apresenta a configuração do bloco “OPC UA Item” para leitura da variável “senoide”. O valor para configuração do campo “Item”, $ns=2; i=3$, foi obtido pelo FreeOpcUa Client, na coluna “NodeId”, conforme pode ser visto na Figura 30.1.

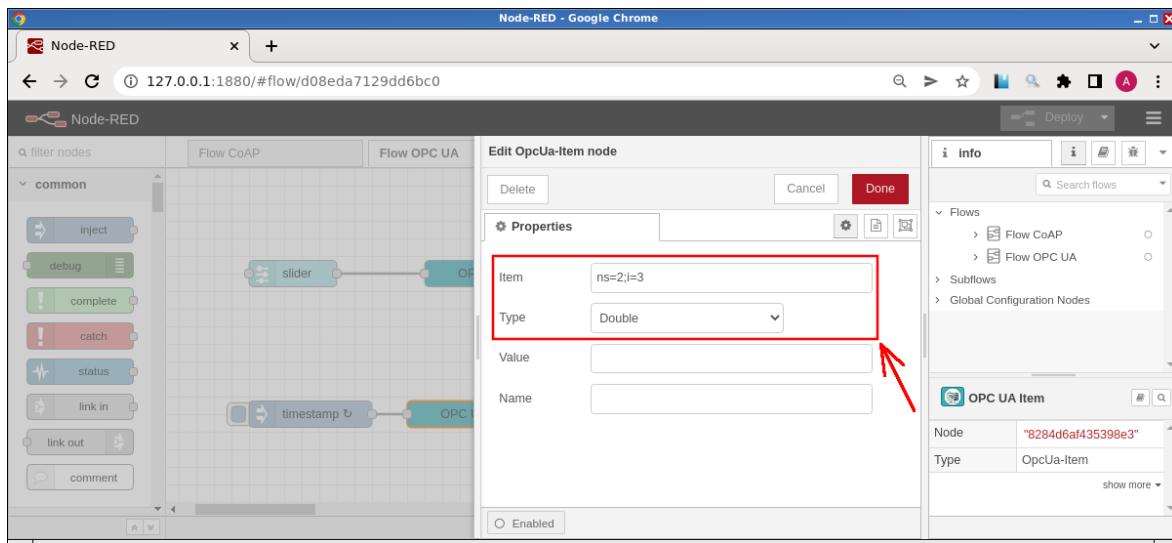


Figura 30.8: Configuração do bloco “OPC UA Item” para leitura da variável “senoide”.

30.3.2.3 OPC UA Client para leitura da variável “Variavel_Senoide”

A Figura 30.9 apresenta a configuração do bloco “OPC UA Client” para leitura da variável “Variavel_Senoide”.

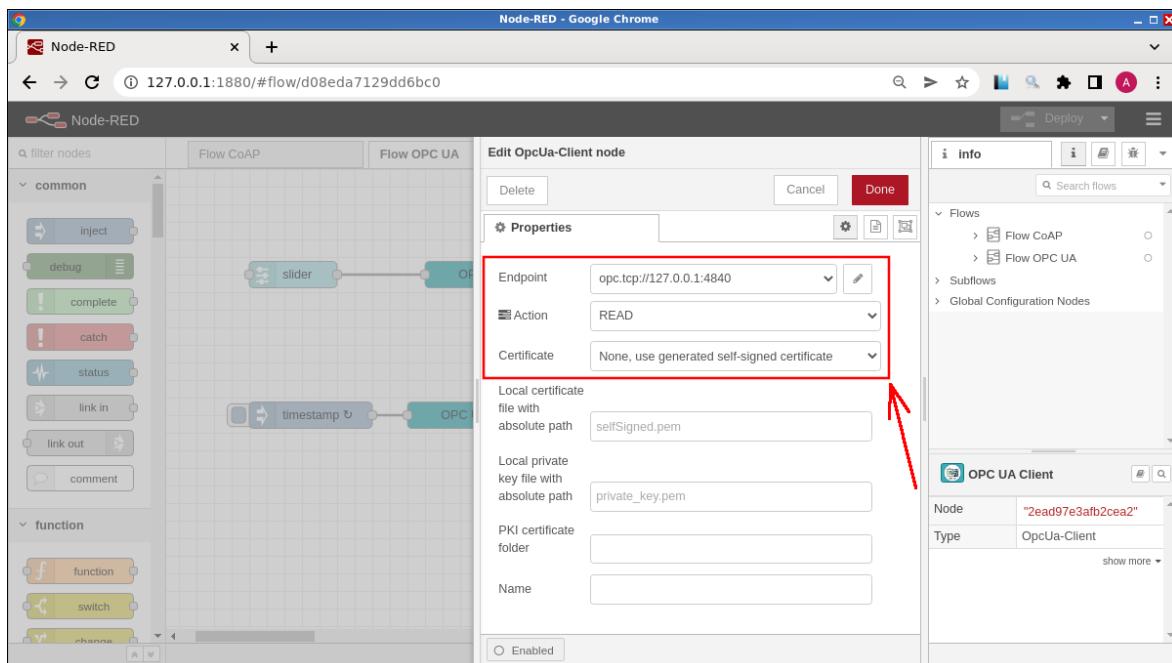


Figura 30.9: Configuração do bloco “OPC UA Client” para leitura da variável “Variavel_Senoide”.

30.3.3 Configuração do bloco “chart”

A configuração do bloco “chart” foi feita para exibir um minuto da senoide gerada pelo servidor OPC UA, conforme apresentado na Figura 30.10.

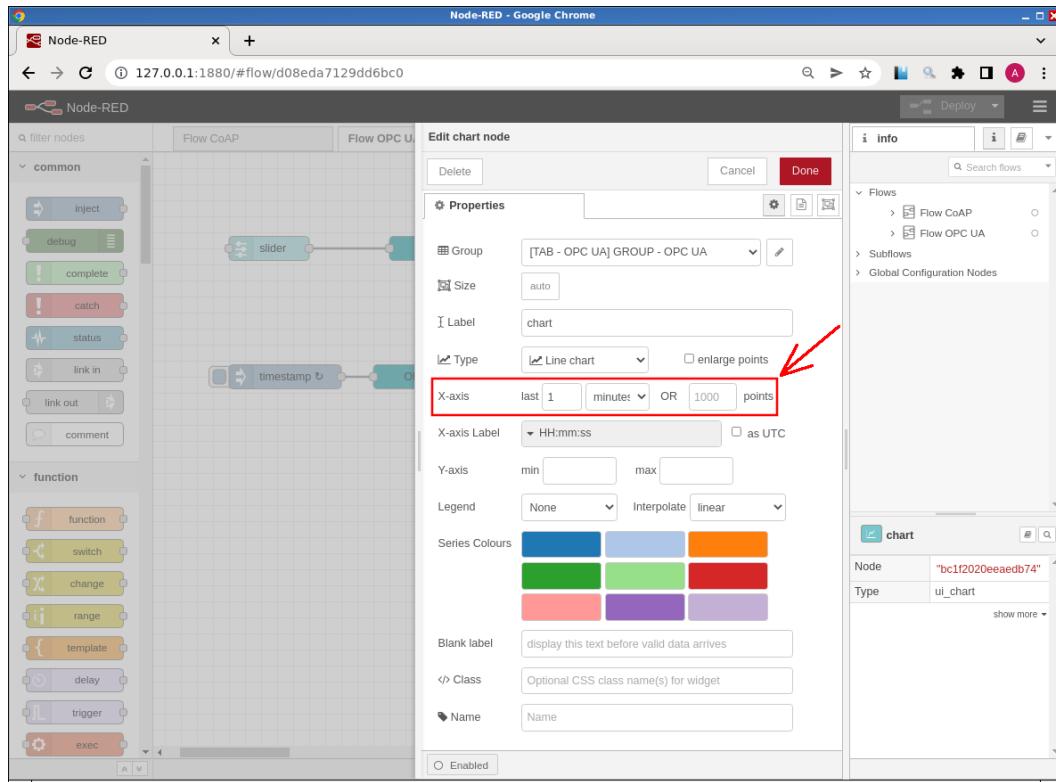


Figura 30.10: Configuração do bloco “chart”.

30.4 Sistema executando

A Figura 30.11 apresenta o dashboard montado executando, acessando o servidor OPC UA.

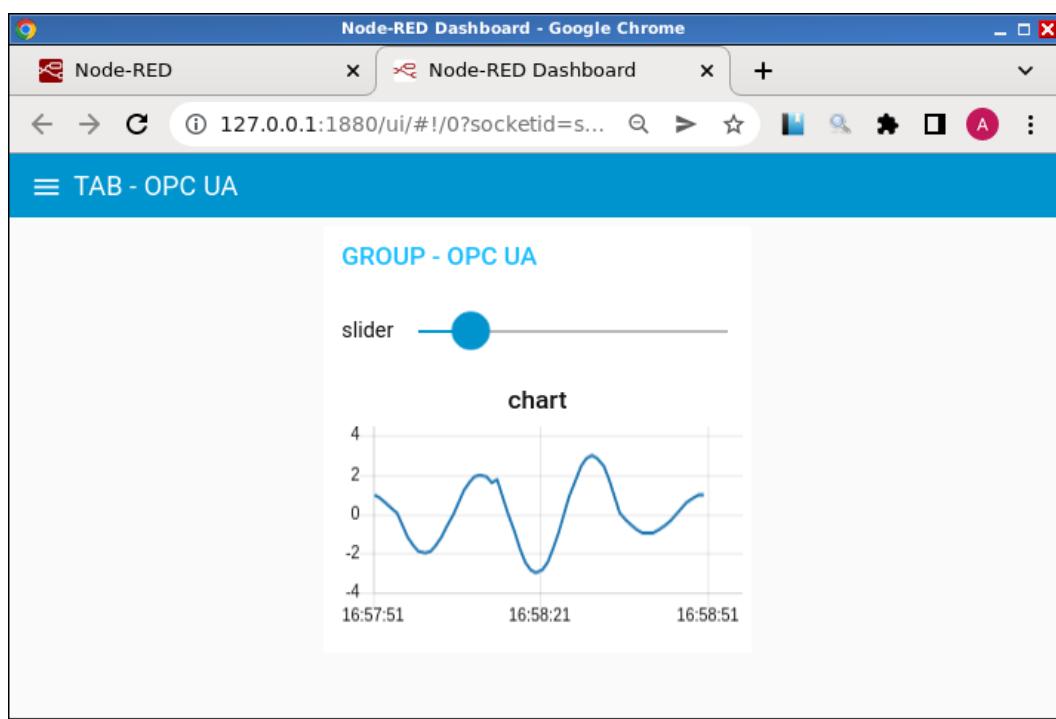


Figura 30.11: Dashboard acessando o servidor OPC UA.

Parte VI

Experimentos e considerações práticas sobre MQTT

Capítulo 31

Introdução ao padrão MQTT

A internet das coisas (IoT) corresponde a uma rede de elementos que podem ser sensores e dispositivos inteligentes. Os sensores são responsáveis pela coleta de dados nos diversos pontos da rede. Os dispositivos inteligentes podem ser, por exemplo, sistemas de monitoramento e segurança doméstica ou dispositivos multimídia. Telefones celulares e outros dispositivos de comunicação sem fio também podem fazer parte de sistemas de IoT, atuando, por exemplo, no monitoramento de pacientes para coleta de dados durante todo o dia em aplicações da área médica, o que não pode ser feito de forma centralizada MAILCHI 2021.

Dentre os dispositivos que implementam tecnologias de IoT, pode-se destacar algumas categorias. Os dispositivos para residências “inteligentes”, desenvolvidos para a automatização de funções domésticas, como o controle de iluminação, temperatura e trancas. Os sensores industriais, que coletam informações de equipamentos, contribuindo para a identificação de falhas e o gerenciamento das manutenções. Os carros inteligentes, que podem fornecer informações sobre o trânsito e ultimamente caminham em direção à autonomia. Dispositivos de monitoramento para aplicações de saúde e controle de atividades físicas, como relógios inteligentes (smart watches), que monitoram constantemente o estado de um indivíduo enviando os dados para um outro equipamento ou profissional da área de saúde VXCHNGE 2020.

Na implementação de sistemas de IoT, um protocolo que se destaca é o MQTT. O MQTT é um padrão para transmitir dados entre um dispositivo IoT e um servidor. Originalmente desenvolvido em 1999 por Andy Stanford-Clark e Arlen Nipper para monitorar oleodutos e gasodutos em conexões remotas por satélite, o MQTT se tornou o padrão IoT de fato para conectar todo tipo de dispositivos IoT (HIVEMQ, 2020). A sigla MQTT significa “MQ Telemetry Transport”, mas anteriormente era conhecida como “Message Queuing Telemetry Transport” STEVES-INTERNET-GUIDE 2021a.

O MQTT é um protocolo de publicação/assinatura (publish/subscribe) leve e requer uma largura de banda mínima para conectar um dispositivo IoT. Ao contrário do paradigma de solicitação/resposta (request/response) do HTTP, o MQTT é orientado por eventos e permite que as mensagens sejam enviadas aos clientes. Esse tipo de arquitetura separa os clientes uns dos outros para permitir uma solução altamente escalável sem dependências entre produtores e consumidores de dados HIVEMQ 2020.

O MQTT é um protocolo de mensagens, ou seja, foi projetado para transferir mensagens e usa um modelo de publicação e assinatura (publish/subscribe). Este modelo permite enviar mensagens para 0,1 ou vários clientes STEVES-INTERNET-GUIDE 2021b.

Pode-se fazer uma analogia do modelo publish/subscribe com canais de TV. Uma emissora de TV transmite um programa usando um canal específico e um espectador sintoniza nesse canal para ver a transmissão. Não há conexão direta entre a emissora e o telespectador. No MQTT, um publicador publica mensagens em um tópico e um assinante deve se inscrever nesse tópico para visualizar a mensagem STEVES-INTERNET-GUIDE 2021b.

Para gerenciar a comunicação o MQTT requer o uso de um Broker (intermediário) central, conforme mostrado na Figura 31.1.

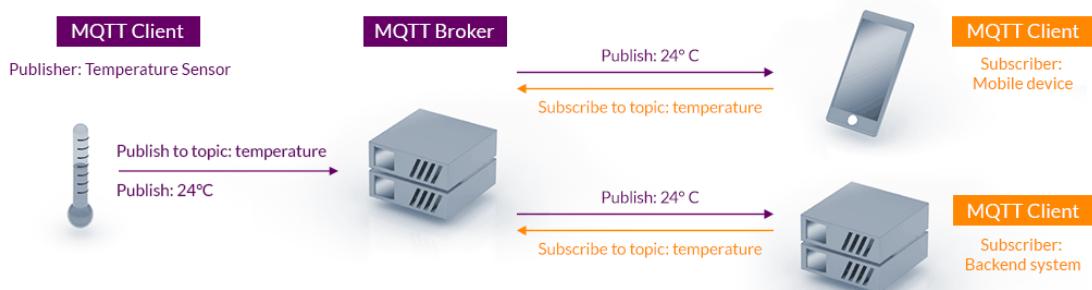


Figura 31.1: Arquitetura Publish/Subscribe do MQTT MQTT.ORG 2021.

Capítulo 32

Exemplo de transmissão de variáveis com MQTT (com broker na nuvem)

Esse capítulo apresenta um exemplo de um cliente publicador (publisher) e um cliente assinante (subscriber). Esse exemplo segue o tutorial “MQTT Beginners Guide” Medium 2020.

Os exemplos estão escritos em linguagem Python. Para a execução do código deve-se instalar a biblioteca “paho-mqtt”, com um dos comandos a seguir (conforme sua versão do python):

Para o Python padrão do Debian (acredito que de outras distribuições do Linux) existe o comando indicado na página (<https://pypi.org/project/paho-mqtt/>):

```
pip install paho-mqtt
```

Para o Anaconda existem os comandos (escolher apenas um) presentes no manual (<https://anaconda.org/conda-forge/paho-mqtt>):

```
conda install -c conda-forge paho-mqtt
conda install -c conda-forge/label/cf202003 paho-mqtt
```

Para teste basta executar o publisher, aguardar a exibição das mensagens de envio da variável para o broker, e na sequência executar o subscriber.

32.1 Cliente publicador (publisher)

Conteúdo do arquivo “01_MQTT_publisher1.py”

```
1 # Alex Brandão Rossow
2 # Adaptado do exemplo disponível em:
3 # https://medium.com/python-point/mqtt-basics-with-python-examples-7c758e605d4
4
5 import paho.mqtt.client as mqtt
6 from random import randrange
7 import time
8
9
10 ##### Configurações para na nuvem #####
11 #mqttBroker ="mqtt.eclipse.org" # Usado no tutorial original (não funciona)
12 #mqttBroker ="mqtt.eclipseprojects.io" # Testado em ABR2021 (funciona)
13
14 ##### Configurações para um broker local (e.g Mosquitto) #####
15 #mqttBroker ="0.0.0.0" # Acessa o broker em qualquer IP (Localhost
16 # ou interface de rede)
17 mqttBroker ="127.0.0.1" # Acessa o broker no IP de localhost
18
19
20 #mqttBroker ="192.168.1.9" # Acessa o broker em um IP específico de uma
21 # interface de rede (e.g. Ethernet ou Wi-Fi)
22 # Também pode acessar um broker na mesma LAN
23 # (IP válido na LAN) ou broker na nuvem
24 # (IP válido na internet)
25
26
27 client = mqtt.Client("Temperature_Outside")
28 client.connect(mqttBroker)
29
30 while True:
31     randNumber = randrange(10)
32     client.publish("TEMPERATURE_Alex", randNumber)
33     print("Just published " + str(randNumber) + " to topic TEMPERATURE_Alex")
34     time.sleep(1)
```

32.2 Cliente assinante (subscriber)

Conteúdo do arquivo “02_MQTT_subscribe.py”

```
1 # Alex Brandão Rossow
```

```
2 # Adaptado do exemplo disponível em:  
3 # https://medium.com/python-point/mqtt-basics-with-python-examples-7c758e605d4  
4  
5 import paho.mqtt.client as mqtt  
6 import time  
7  
8 def on_message(client, userdata, message):  
9     print("received message: " ,str(message.payload.decode("utf-8")))  
10  
11  
12 ##### Configurações para na nuvem #####  
13 #mqttBroker ="mqtt.eclipse.org" # Usado no tutorial original (não funciona)  
14 #mqttBroker ="mqtt.eclipseprojects.io" # Testado em ABR2021 (funciona)  
15  
16 ##### Configurações para um broker local (e.g Mosquitto) #####  
17 #mqttBroker ="0.0.0.0" # Acessa o broker em qualquer IP (Localhost  
18 # ou interface de rede)  
19 mqttBroker ="127.0.0.1" # Acessa o broker no IP de localhost  
20  
21  
22 #mqttBroker ="192.168.1.9" # Acessa o broker em um IP específico de uma  
23 # interface de rede (e.g. Ethernet ou Wi-Fi)  
24 # Também pode acessar um broker na mesma LAN  
25 # (IP válido na LAN) ou broker na nuvem  
26 # (IP válido na internet)  
27  
28  
29 client = mqtt.Client("Smartphone")  
30 client.connect(mqttBroker)  
31  
32 client.loop_start()  
33  
34 client.subscribe("TEMPERATURE_Alex")  
35 client.on_message=on_message  
36  
37 time.sleep(30)  
38 client.loop_stop()
```

32.2.1 Resultado do teste

A Figura 32.1 apresenta os programas rodando, utilizando o bloker remoto (mqtt.eclipseprojects.io). O terminal da esquerda apresenta a execução do *publisher* e o terminal da direita apresenta a execução do *subscriber*.

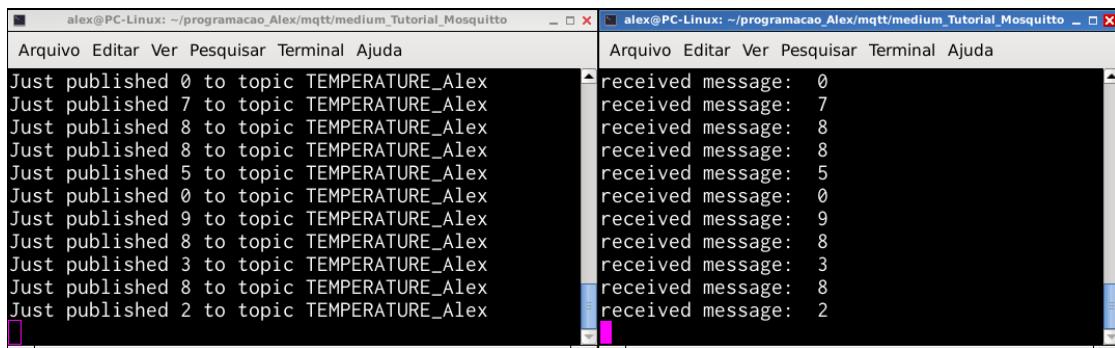


Figura 32.1: Teste com o MQTT. Na esquerda o publisher e na direita o subscriber.

32.3 Uso do MQTT Explorer como subscriber

O subscriber do exemplo deste capítulo pode ser substituído pelo programa MQTT Explorer.

O MQTT Explorer é um programa gratuito, disponível no seguinte endereço:

<http://mqtt-explorer.com/>

Esse programa conecta-se ao broker e apresenta todos os tópicos publicados por esse broker.

A Figura 32.2 apresenta a configuração básica para acesso ao broker, executando na nuvem, utilizado neste exemplo. Por padrão, ao se criar uma nova conexão a configuração padrão faz o cliente se inscrever em todos os tópicos do Broker, isso pode fazer MQTT Explorer ficar lento ou travar.

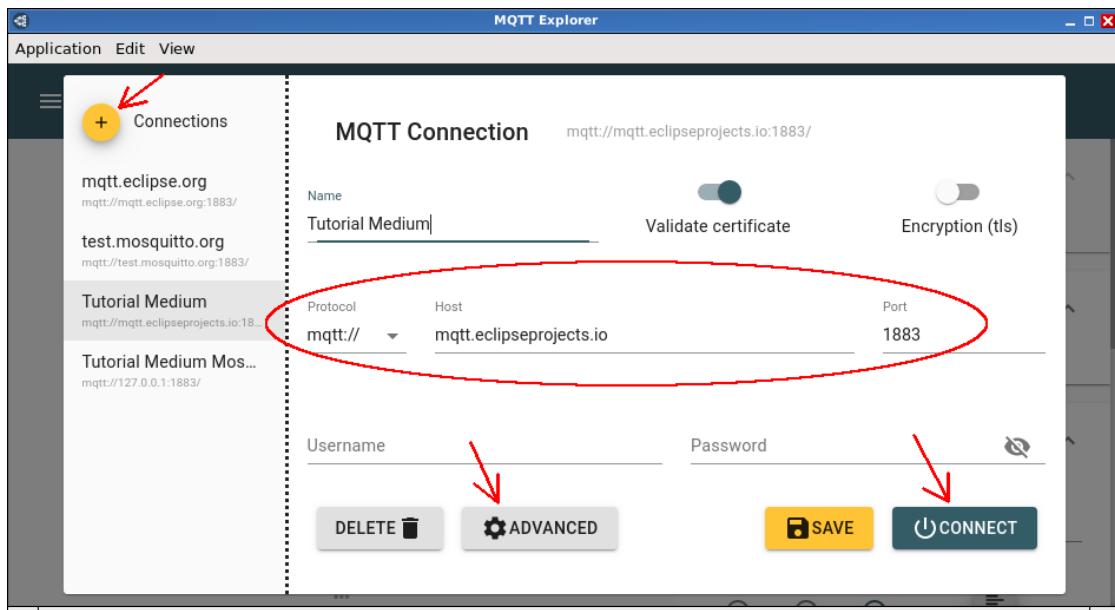


Figura 32.2: Configuração de conexão no MQTT Explorer.

Clicando no botão “ADVANCED” é acessada a lista de tópicos para inscrição. Por padrão são usados códigos definidos pelo padrão MQTT para inscrição em todos os tópicos do broker. A Figura 32.3 apresenta a tela de configuração avançada, com a configuração padrão para inscrição em todos os tópicos.

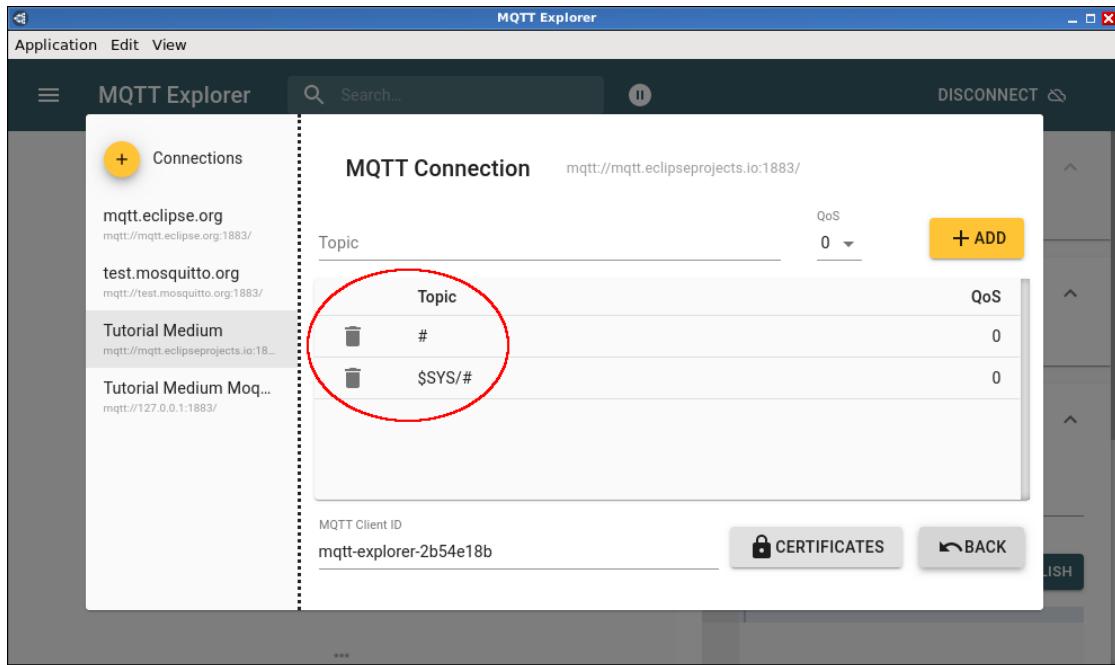


Figura 32.3: Configuração padrão de uma conexão. Com esses códigos (definidos no padrão MQTT) o MQTT Explorer lista todos os tópicos do Broker.

A Figura 32.4 apresenta o MQTT Explorer já conectado, apresentando a variável enviada para o broker pelo código do publisher. Observa-se que o MQTT Explorer está listando todos tópicos do broker (o que pode fazer o MQTT Explorer travar).

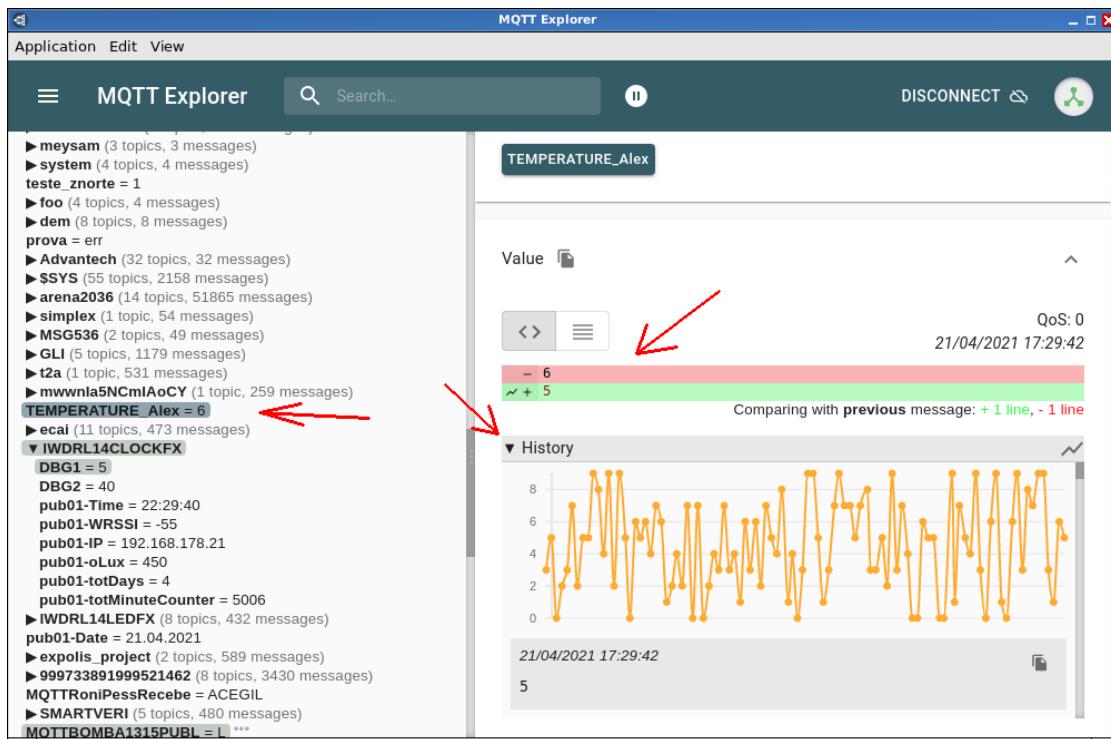


Figura 32.4: MQTT Explorer conectado ao broker, atuando como subscriber.

Para se inscrever em uma lista específica de tópicos, deve-se eliminar as inscrições coringa (apresentadas na Figura 32.3) e inserir os tópicos desejados. Isso elimina o problema do MQTT Explorer travar pelo excesso de tópicos. A Figura 32.5 apresenta a configuração de um tópico específico, acessada pelo botão de configuração “ADVANCED”.

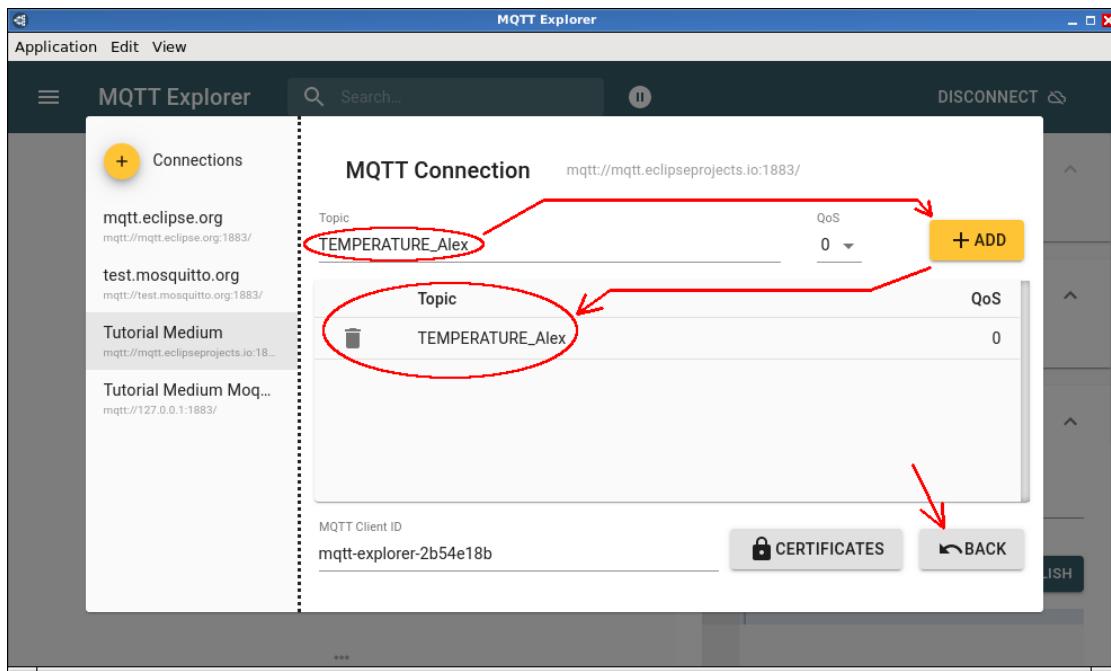


Figura 32.5: Configuração de um tópico específico para inscrição.

A Figura 32.6 apresenta tela do MQTT Explorer conectado ao broker com a configuração para inscrição em um único tópico.

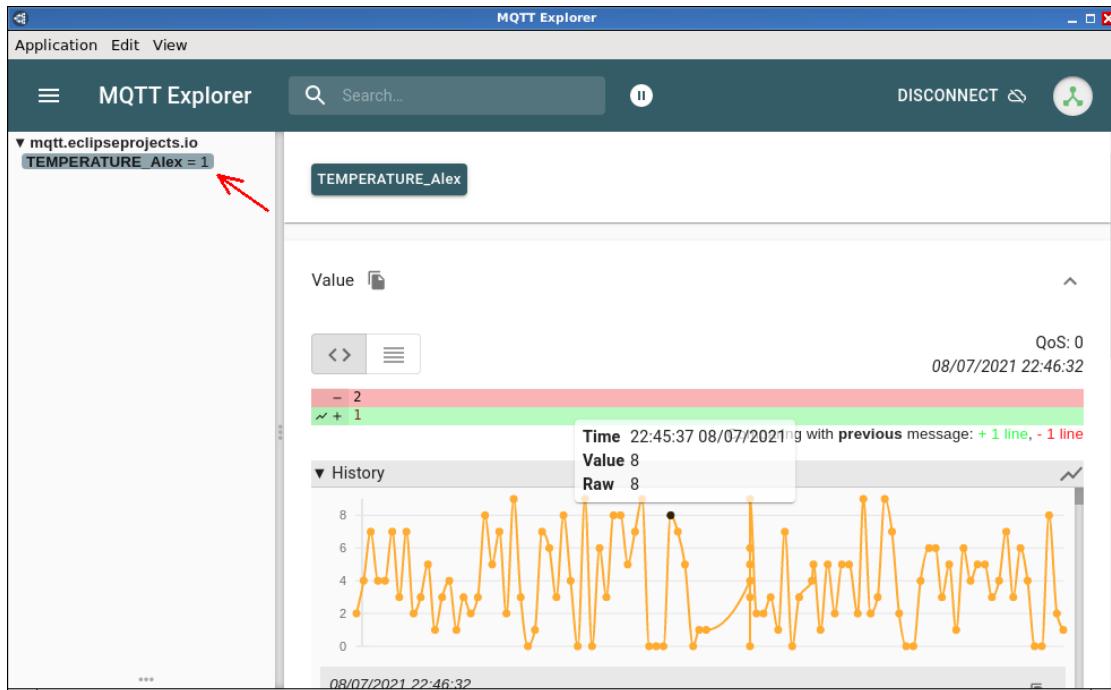


Figura 32.6: MQTT Explorer configuração para inscrição a um único tópico.

Capítulo 33

Eclipse Mosquitto (broker MQTT *open source*)

Esse capítulo apresenta o broker Eclipse Mosquitto, sua instalação e seu uso.

33.1 Introdução

O **Eclipse Mosquitto** é um broker *open source* (EPL/EDL licensed) que implementa o protocolo MQTT versões 5.0, 3.1.1 e 3.1. É um programa leve adequado para execução neste computadores do tipo *single board* até servidores Eclipse Mosquitto™ 2021.

33.2 Instalação e execução

Esta seção apresentará o processo de instalação e execução do Mosquitto no Linux. Esta seção não tem por objetivo apresentar detalhes sobre a instalação em diferentes sistemas operacionais ou fazer uma revisão sobre a execução de processos no Linux. Para mais detalhes e orientações sobre a instalação em outros sistemas (incluindo o Windows) existem diversos sites. As páginas do “Steve’s Internet Guide”, na sequência, apresentam boas informações.

- Mosquitto MQTT Broker (<http://www.steves-internet-guide.com/mosquitto-broker/>),
- How to Install The Mosquitto MQTT Broker on Linux (<http://www.steves-internet-guide.com/install-mosquitto-linux/>),

- How to Install The Mosquitto MQTT Broker on Windows (<http://www.steves-internet-guide.com/install-mosquitto-broker/>).

33.2.1 Instalação

A instalação no Debian (Ubuntu e derivados) é feita com o comando:

```
sudo apt-get install mosquitto
```

33.2.2 Iniciando e interrompendo a execução do Mosquitto

Para **iniciar** o broker Mosquitto execute o comando:

```
sudo service mosquitto start
```

Para **interromper** o serviço do broker execute:

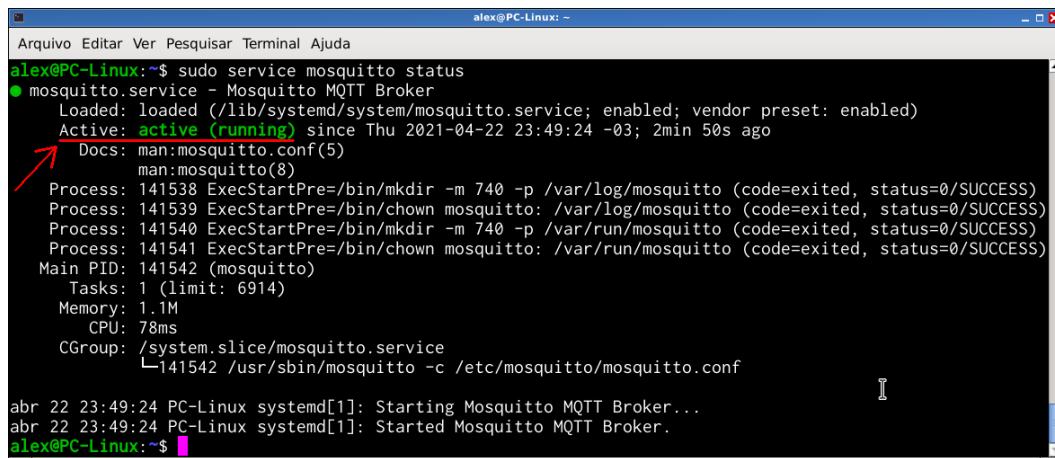
```
sudo service mosquitto stop
```

33.2.3 Obtenção de informações sobre o Mosquitto

Para visualizar o estado do Mosquitto execute:

```
sudo service mosquitto status
```

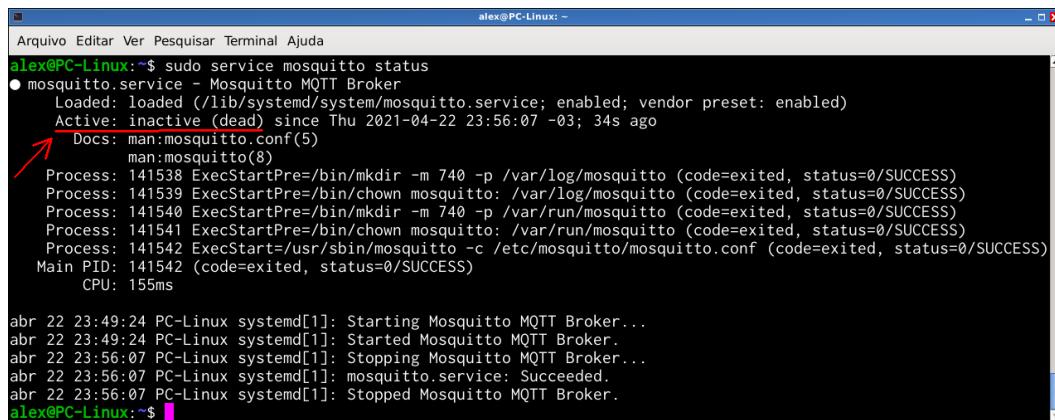
Se o Mosquitto estiver executando deve-se observar uma indicação semelhante à da Figura 33.1 e se não estiver executando observa-se a mensagem da Figura 33.2



```
alex@PC-Linux:~$ sudo service mosquitto status
● mosquitto.service - Mosquitto MQTT Broker
  Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
  Active: active (running) since Thu 2021-04-22 23:49:24 -03; 2min 50s ago
    Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
  Process: 141538 ExecStartPre=/bin/mkdir -m 740 -p /var/log/mosquitto (code=exited, status=0/SUCCESS)
  Process: 141539 ExecStartPre=/bin/chown mosquitto: /var/log/mosquitto (code=exited, status=0/SUCCESS)
  Process: 141540 ExecStartPre=/bin/mkdir -m 740 -p /var/run/mosquitto (code=exited, status=0/SUCCESS)
  Process: 141541 ExecStartPre=/bin/chown mosquitto: /var/run/mosquitto (code=exited, status=0/SUCCESS)
 Main PID: 141542 (mosquitto)
   Tasks: 1 (limit: 6914)
     Memory: 1.1M
        CPU: 78ms
      CGroup: /system.slice/mosquitto.service
              └─141542 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

abr 22 23:49:24 PC-Linux systemd[1]: Starting Mosquitto MQTT Broker...
abr 22 23:49:24 PC-Linux systemd[1]: Started Mosquitto MQTT Broker.
alex@PC-Linux:~$
```

Figura 33.1: Mosquitto executando.



```
alex@PC-Linux:~$ sudo service mosquitto status
● mosquitto.service - Mosquitto MQTT Broker
  Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
  Active: inactive (dead) since Thu 2021-04-22 23:56:07 -03; 34s ago
    Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
  Process: 141538 ExecStartPre=/bin/mkdir -m 740 -p /var/log/mosquitto (code=exited, status=0/SUCCESS)
  Process: 141539 ExecStartPre=/bin/chown mosquitto: /var/log/mosquitto (code=exited, status=0/SUCCESS)
  Process: 141540 ExecStartPre=/bin/mkdir -m 740 -p /var/run/mosquitto (code=exited, status=0/SUCCESS)
  Process: 141541 ExecStartPre=/bin/chown mosquitto: /var/run/mosquitto (code=exited, status=0/SUCCESS)
  Process: 141542 ExecStart=/usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf (code=exited, status=0/SUCCESS)
 Main PID: 141542 (code=exited, status=0/SUCCESS)
    CPU: 155ms

abr 22 23:49:24 PC-Linux systemd[1]: Starting Mosquitto MQTT Broker...
abr 22 23:49:24 PC-Linux systemd[1]: Started Mosquitto MQTT Broker.
abr 22 23:56:07 PC-Linux systemd[1]: Stopping Mosquitto MQTT Broker...
abr 22 23:56:07 PC-Linux systemd[1]: mosquitto.service: Succeeded.
abr 22 23:56:07 PC-Linux systemd[1]: Stopped Mosquitto MQTT Broker.
alex@PC-Linux:~$
```

Figura 33.2: Mosquitto não executando.

Para saber se o Mosquitto está escutando na porta padrão (1883), o que deve acontecer se estiver ativo, execute:

```
netstat -an | grep 1883
```

A indicação de que o Mosquitto está escutando nas portas pesquisadas é feita pelas linhas apontada na Figura 33.3



```
alex@PC-Linux:~$ netstat -an | grep 1883
tcp        0      0 127.0.0.1:1883          0.0.0.0:*          OUÇA
tcp6       0      0 ::1:1883               ::*:             OUÇA
unix  3     [ ]         STREAM     CONECTADO    21883    /run/user/1000/bus
alex@PC-Linux:~$
```

Figura 33.3: Indicação do broker Mosquitto escutando na porta 1883.

33.3 Teste do Mosquitto com MQTT Explorer

Para o teste pode ser feita a substituição do broker remoto pelo broker local (127.0.0.1) no programa do publisher, na Seção 32.1, e do subscriber, na Seção 32.2.

Executando os programas com a alteração do broker deve-se obter o mesmo resultado da Figura 32.1.

A Figura 33.4 apresenta a configuração da conexão para o broker local no MQTT Explorer. E a Figura 33.5 apresenta o MQTT Explorer já conectado ao broker local.

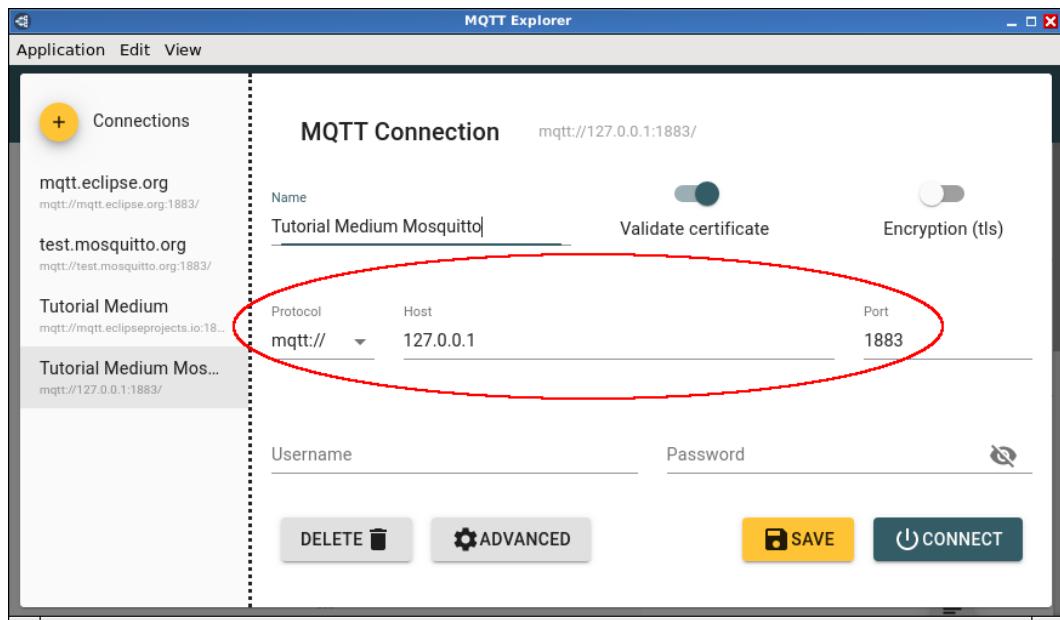


Figura 33.4: Configuração do MQTT Explorer para conexão ao broker Mosquitto.

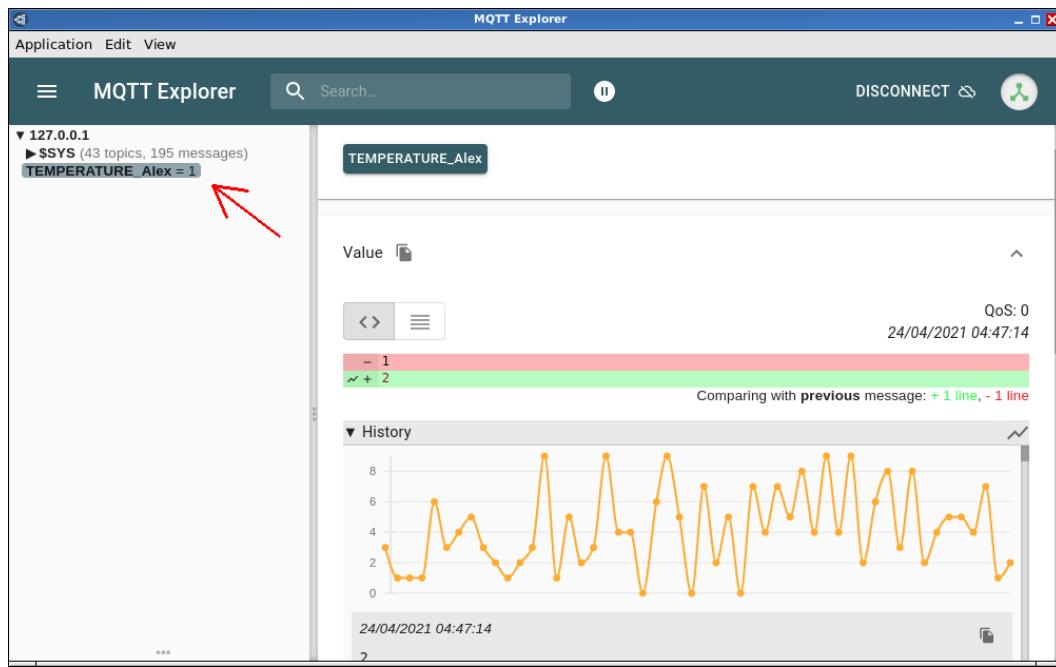


Figura 33.5: MQTT Explorer conectado ao broker local, o Mosquitto.

Capítulo 34

Configuração do Mosquitto para acesso externo

Com a execução do Mosquitto sem configuração, observou-se que o broker ficou disponível apenas para programas executando no mesmo computador que o Mosquitto. Não foi possível, por exemplo acessar o broker por um celular conectado à mesma LAN que o computador.

34.1 Configuração do Mosquitto

Para a liberação do acesso ao broker é necessário criar um arquivo de configuração. No Debian (acredito que é o mesmo para Ubuntu e derivados) o arquivo de configuração deve ser terminado com “**.conf**” e estar localizado no diretório “**/etc/mosquitto/conf.d**”. Como exemplo, o meu arquivo de configuração ficou com o seguinte caminho:

```
/etc/mosquitto/conf.d/alex_Mosquitto.conf
```

Esse arquivo possui o seguinte conteúdo (somente as linhas não comentadas são necessárias):

```
### Para o Mosquitto escutar em qualquer IP  
listener 1883
```

```
### Para o Mosquito escutar em um IP específico (e.g. uma interface de rede)
```

```
#listener 1883 192.168.1.9

### Sem essa linha o Mosquitto não aceita conexões externas
allow_anonymous true

#cafile <path to ca file>
#certfile <path to server cert>
#keyfile <path to server key>
#require_certificate false
```

Nessa configuração, o Mosquito atende em qualquer IP do computador (o mesmo que a configuração 0.0.0.0). Isso é importante para que não seja necessária a mudança desse arquivo sempre que o computador receber um novo endereço de IP (por um DHCP, por exemplo).

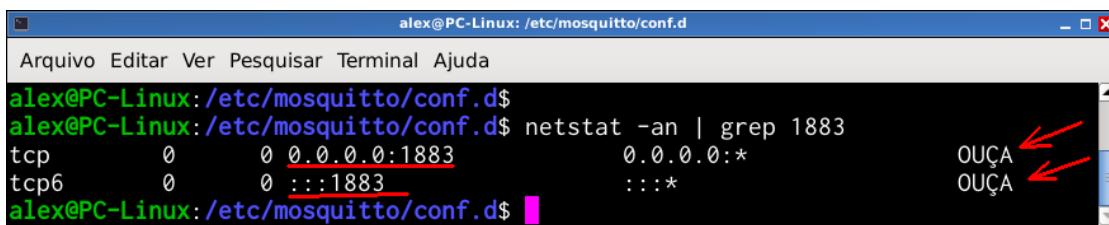
Após a criação ou alteração do arquivo de configuração, deve-se reiniciar o Mosquitto, com os comandos:

```
sudo service mosquitto stop
sudo service mosquitto start
```

Após o reinício, pode-se observar que o Mosquitto está escutando por meio de qualquer IP do computador. Para isso execute:

```
netstat -an | grep 1883
```

A Figura 34.1 apresenta o resultado do comando, com a indicação de que o broker Mosquitto está aceitando conexões em qualquer IP do computador (a indicação 0.0.0.0).



```
alex@PC-Linux:/etc/mosquitto/conf.d$ netstat -an | grep 1883
tcp        0      0 0.0.0.0:1883          0.0.0.0:*
tcp6       0      0 ::1883              ::*:*
```

Figura 34.1: Endereço e portas para acesso ao broker Mosquitto. A configuração 0.0.0.0 indica que o broker atende em qualquer IP do computador (localhost ou interface de rede).

O mesmo comando lista as conexões existentes, caso já exista algum cliente MQTT conectado ao broker. A Figura 34.2 apresenta a situação onde o broker está sendo acessado pelo publisher da Seção 32.1, pelo MQTT Explorer e por um telefone celular (com aplicativo MQTIZER).

```
alex@PC-Linux:/etc/mosquitto/conf.d$ netstat -an | grep 1883
tcp        0      0 0.0.0.0:1883          0.0.0.*:*
tcp        0      0 127.0.0.1:1883        127.0.0.1:50738
tcp        4      0 127.0.0.1:39491       127.0.0.1:1883
tcp        0      0 127.0.0.1:50738       127.0.0.1:1883
tcp        0      0 127.0.0.1:1883        127.0.0.1:39491
tcp        0     21 192.168.1.9:1883      192.168.1.2:54290
tcp6       0      0 ::1:1883            ::*:*
alex@PC-Linux:/etc/mosquitto/conf.d$
```

IP do computador na LAN (e porta usada no broker)

IP do celular na LAN (e porta usada no celular)

OUÇA

ESTABELECIDA

ESTABELECIDA

ESTABELECIDA

ESTABELECIDA

ESTABELECIDA

ESTABELECIDA

OUÇA

Figura 34.2: Lista de clientes atendidos pelo broker Mosquitto.

Na figura, observa-se que o atendimento às aplicações locais se dá com o IP 127.0.0.1 (localhost). Para o celular (destacado pelas setas e comentários verdes) o atendimento se dá pela interface de rede, no caso a placa Wi-Fi.

34.2 Teste de conexão ao broker Mosquitto a partir de um telefone celular

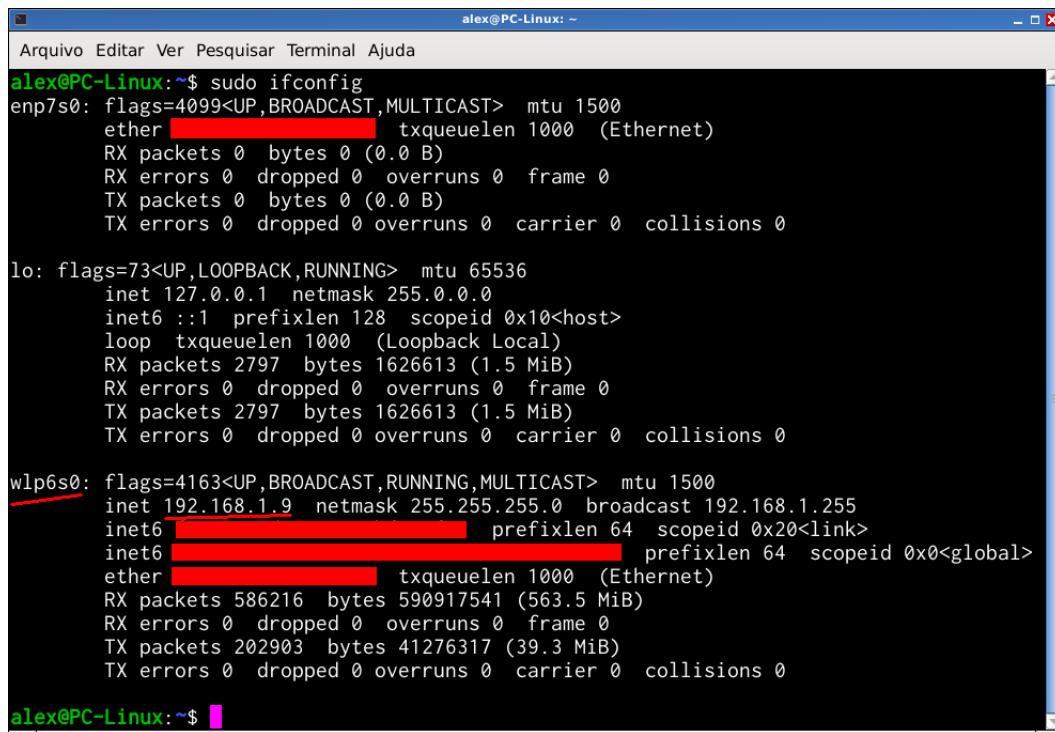
Esta seção apresenta o acesso ao broker Mosquito, executando no computador, a partir de um telefone celular com aplicativo “MQTIZER”. Não fiz uma busca complexa por clientes MQTT para celular, portanto, podem existir outros aplicativos com recursos interessantes. Para teste de funcionamento do broker o MQTIZER atende ao necessário.

34.2.1 Descobrindo o IP do broker

O primeiro passo para conexão por um dispositivo remoto é a identificação do IP do computador que executa o broker na LAN usada. No Linux isso é feito com o comando:

```
sudo ifconfig
```

No Windows o comando equivalente é o “ipconfig”.



```
alex@PC-Linux:~$ sudo ifconfig
enp7s0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
      ether [REDACTED] txqueuelen 1000 (Ethernet)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
      inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
          loop txqueuelen 1000 (Loopback Local)
            RX packets 2797 bytes 1626613 (1.5 MiB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 2797 bytes 1626613 (1.5 MiB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlp6s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 192.168.1.9 netmask 255.255.255.0 broadcast 192.168.1.255
        inet6 [REDACTED] prefixlen 64 scopeid 0x20<link>
        inet6 [REDACTED] prefixlen 64 scopeid 0x0<global>
      ether [REDACTED] txqueuelen 1000 (Ethernet)
        RX packets 586216 bytes 590917541 (563.5 MiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 202903 bytes 41276317 (39.3 MiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

alex@PC-Linux:~$
```

Figura 34.3: Saída do comando “sudo ifconfig”.

A Figura 34.3 apresenta o resultado do ifconfig. Esse comando apresenta informações de todas as interfaces de rede. No caso apresentado a interface de conexão com a LAN é a “wlp6s0”, que corresponde à placa Wi-Fi. O IP do computador é o “192.168.1.9”.

34.2.2 Uso do MQTIZER

Esta seção apresenta os passos para conexão ao broker e assinatura do tópico publicado pelo publisher da Seção 32.1.

A Figura 34.4 apresenta a tela inicial do MQTIZER, destacando o botão para configurar um novo broker.

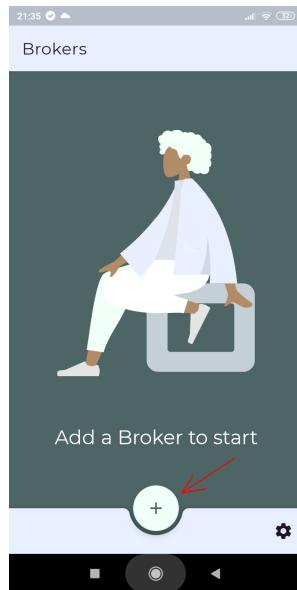


Figura 34.4: Tela inicial do MQTIZER.

A Figura 34.5 apresenta a tela de preenchimento das informações do broker. O campo “Nick Name” pode receber qualquer nome. O campo “Host Name” recebe o IP ou endereço web do broker. A porta é aquela configurada no broker, o valor padrão é esse da figura.

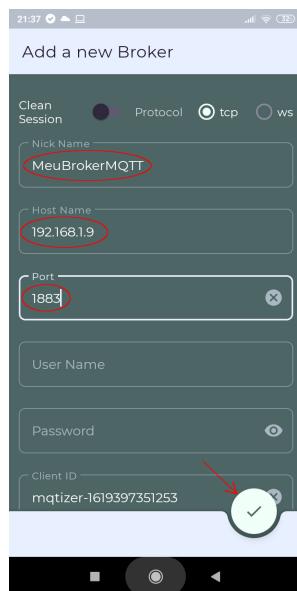


Figura 34.5: Configuração do broker a ser acessado.

A Figura 34.6 apresenta o broker já configurado. Para conectar ao broker deve-se clicar sobre seu nome.

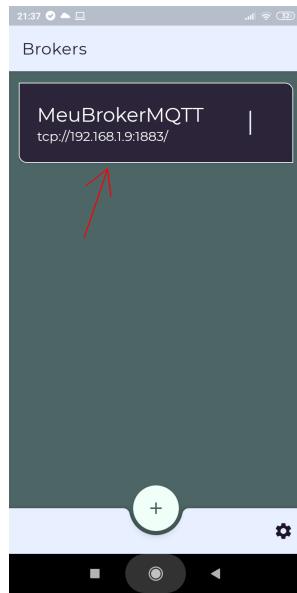


Figura 34.6: Broker configurado para conexão.

Após conectar ao broker pela primeira vez deve-se acessar a aba “TOPICS”, digitar o nome do tópico desejado (diferente do MQTT Explorer, o MQTIZER não lista os tópicos existentes no broker) e clicar no botão “ADD TOPIC”. Esses passos estão representados na Figura 34.7.

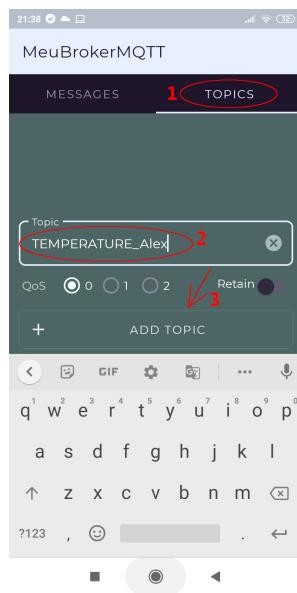


Figura 34.7: Adicionando um tópico ao subscriber após a conexão com o broker.

Com o tópico adicionado (ou vários tópicos), deve-se acionar ativar o botão “Subscribe” para que o MQTIZER receba os valores das variáveis ativas (substritas). A Figura 34.8 apresenta tela com o tópico configurado.

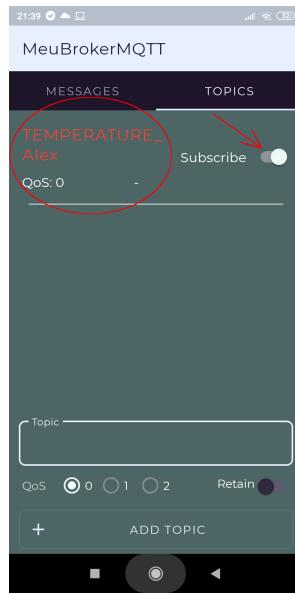


Figura 34.8: Tópico adicionado, com opção de ativar ou não o acesso pelo cliente (subscriber).

A Figura 34.9 apresenta a aba “Messages” mostrando as mensagens recebidas no tópico subscrito pelo cliente (neste caso somente um tópico).



Figura 34.9: Mensagens recebidas pelo cliente subscriber.

Capítulo 35

Exemplo de aplicação do padrão MQTT (testes com o Node-RED)

Este capítulo apresenta o acesso a um cliente MQTT a partir do Node-RED.

O programa possui dois tópicos acessíveis por MQTT, descritas a seguir:

- amplitude_MQTT: Valor passado pelo usuário para definição da amplitude de uma onda senoidal criada pelo programa. Para esta variável o Node-RED atua como publisher e o programa em python atua como subscriber.
- Variavel_Senoide: O valor da variável senoidal criada pelo programa. A amplitude é configurada pelo usuário. Os demais parâmetros (período de amostragem e frequência) são configurados no código. Para esta variável o programa em python atua como publisher e o Node-RED atua como subscriber.

35.1 Código executado

```
1 # Alex Brandão Rossow
2
3 import paho.mqtt.client as mqtt
4 from random import randrange
5 import time
6 import math
7
8 from threading import Thread # Import para operar processos paralelos
9
10
11 ##### Configurações para na nuvem #####
12 #mqttBroker ="mqtt.eclipse.org" # Usado no tutorial original (não funciona)
13 #mqttBroker ="mqtt.eclipseprojects.io" # Testado em ABR2021 (funciona)
14
15 ##### Configurações para um broker local (e.g Mosquitto) #####
16 #mqttBroker ="0.0.0.0" # Acessa o broker em qualquer IP (Localhost
17 # ou interface de rede)
18 mqttBroker ="127.0.0.1" # Acessa o broker no IP de localhost
19
20
21 mqttBroker ="192.168.1.9" # Acessa o broker em um IP específico de uma
22 # interface de rede (e.g. Ethernet ou Wi-Fi)
23 # Também pode acessar um broker na mesma LAN
24 # (IP válido na LAN) ou broker na nuvem
25 # (IP válido na internet)
26
27
28 # Variáveis globais
29 amplitude = 1.0;
30
31
32
33 ##### Subscribers #####
34 # Função responsável por receber o valor da amplitude
35 # pelo modo subscriber do MQTT
36 def on_message(cliente, userdata, message):
37     global amplitude
38     print("Mensagem recebida: " ,str(message.payload.decode("utf-8")))
39     amplitude = int(message.payload)
40 ##### Publishers #####
41
42
43
44 ##### Publishers #####
```

```
45 # Gera uma senoide e publica no broker MQTT
46 def variavel_Senoide_MQTT_Task():
47
48     global mqttBroker
49     global amplitude
50
51     clienteEnvio = mqtt.Client("ClienteEnvioMQTT")
52     clienteEnvio.connect(mqttBroker)
53
54     # O período de amostragem mínimo exibido pelo
55     # opcua-client é 1 segundo.
56     print(">>> Entrou em variavel_Senoide_MQTT_Task")
57     t = 0; # tempo
58     ts = 1; # período de amostragem em segundos
59     f = 0.05; # frequência em Hz
60     #pi = 3.14;
61     periodo = 1/f; # período da senoide em segundos
62     senoide = 0;
63
64     try:
65         while True:
66             time.sleep(ts)
67             t += ts;
68             senoide = amplitude*(math.sin(2*math.pi*f*t));
69             if (t>=periodo):
70                 t = 0;
71             try:
72                 clienteEnvio.publish("Variavel_Senoide", senoide)
73                 print("Publicou " + str(senoide) + " no tópico Variavel_Senoide")
74             except ValueError:
75                 print("Erro ao escrever a variável!")
76         finally:
77             return;
78
79     time.sleep(1)
80 ##########
81
82
83
84 def iniciaMQTT():
85
86     global mqttBroker
87
88     # Inicia a thread que tem a função de publisher MQTT
89     t1 = Thread(target=variavel_Senoide_MQTT_Task, args=())
90
```

```
91 #####
92 # Cliente subscriber para receber o valor de amplitude do Node-RED
93 #####
94 clienteRecepcao = mqtt.Client("ClienteRecepcaoMQTT")
95 clienteRecepcao.connect(mqttBroker)
96
97 clienteRecepcao.loop_start()
98
99 clienteRecepcao.subscribe("amplitude_MQTT")
100 clienteRecepcao.on_message=on_message
101
102 #####
103 #time.sleep(30)
104 #client.loop_stop()
105 #####
106 #####
107
108 # Inicia as tasks
109 #t1.start()
110 t1.start()
111
112 # Espera todas as tasks finalizarem
113 #t1.join()
114 t1.join()
115
116
117 if __name__ == "__main__":
118     iniciaMQTT();
```

35.2 Sistema montado no Node-RED

A Figura 35.1 apresenta o sistema simulado no Node-RED.

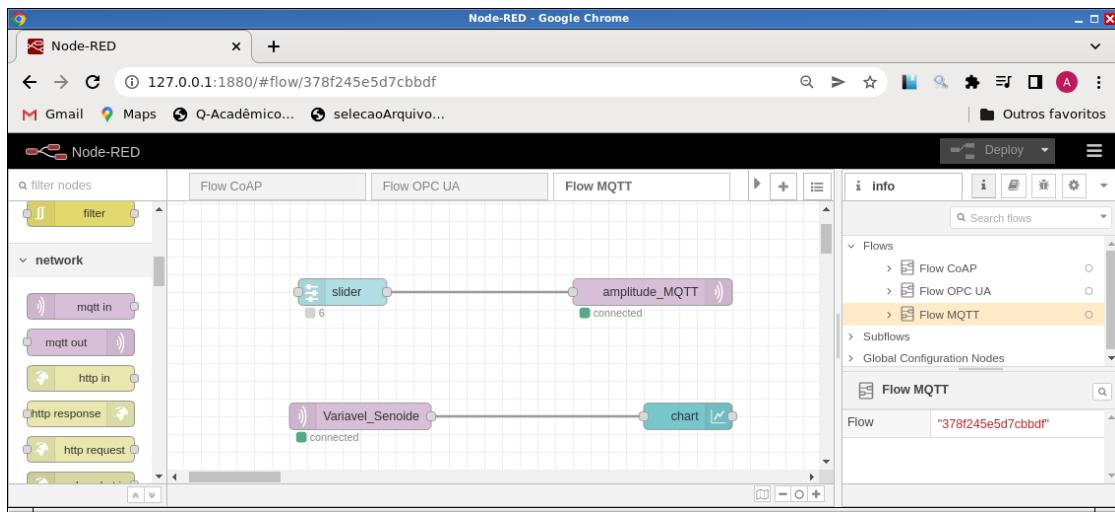


Figura 35.1: Montagem do sistema no Node-RED.

35.2.1 Configuração dos blocos

Esta seção apresenta a configuração dos blocos utilizados no sistema.

35.2.1.1 Configuração do slider

O slider foi configurado para variar de 0 a 10 com salto 1, conforme apresentado na Figura 35.2.

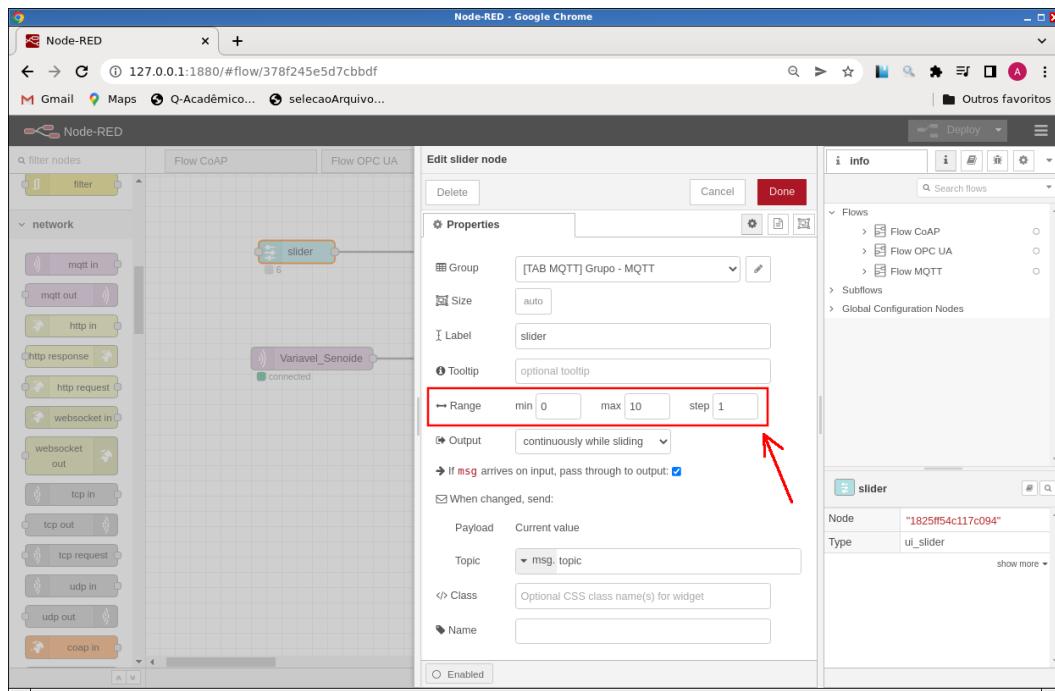


Figura 35.2: Configuração do slider.

35.2.1.2 Configuração do bloco “mqtt out”

O bloco “mqtt out” é utilizado para o Node-RED atuar como publisher de um tópico. A Figura 35.3 apresenta a configuração do bloco “mqtt out” utilizado no sistema para o envio do valor de amplitude do Node-RED para o programa em python.

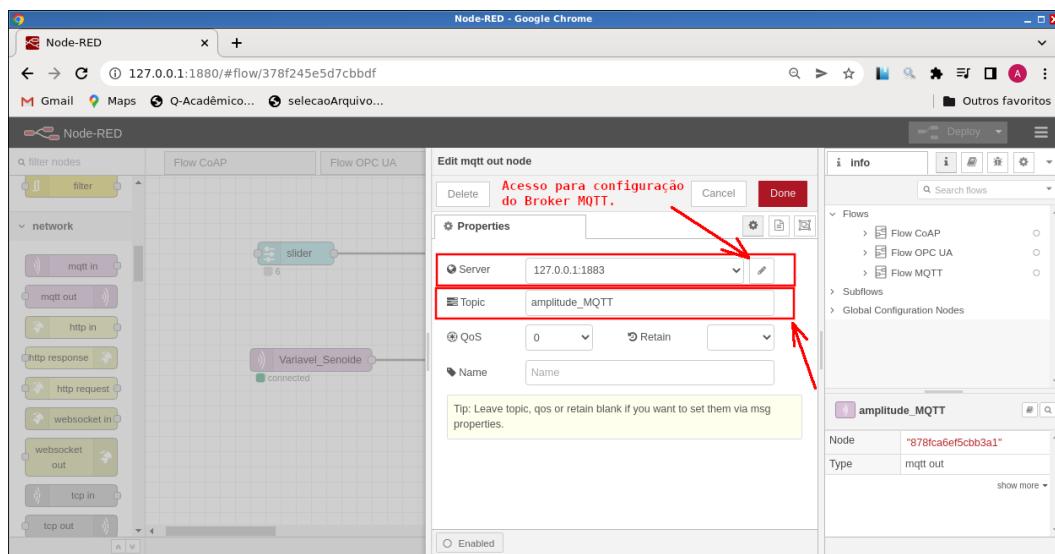


Figura 35.3: Configuração do bloco mqtt out.

35.2.1.3 Configuração do Broker MQTT

A configuração dos Brokers pode ser acessada pelos blocos “mqtt in” ou “mqtt out”. A Figura 35.4 apresenta a configuração utilizada neste exemplo. Foi utilizado um broker local, servidor Mosquitto, executando no mesmo computador que o Node-RED (endereço 127.0.0.1).

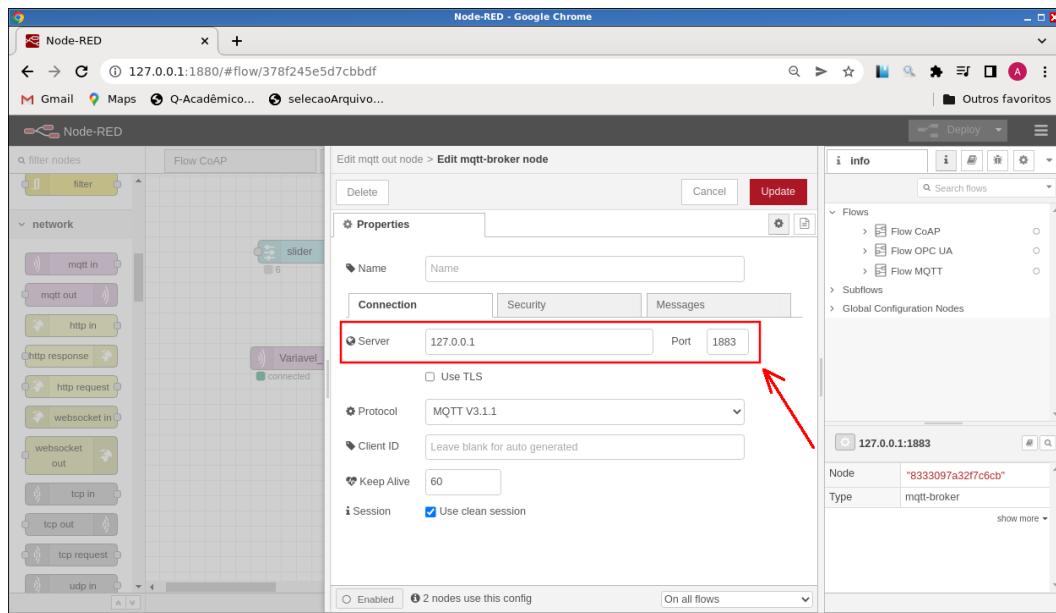


Figura 35.4: Configuração do Broker MQTT.

35.2.1.4 Configuração do bloco “mqtt in”

O bloco “mqtt in” é utilizado para o Node-RED atuar como subscriber de um varável MQTT. A Figura 35.5 apresenta a configuração do bloco “mqtt in” utilizado no sistema para receber os valores da senoide gerada pelo programa em python.

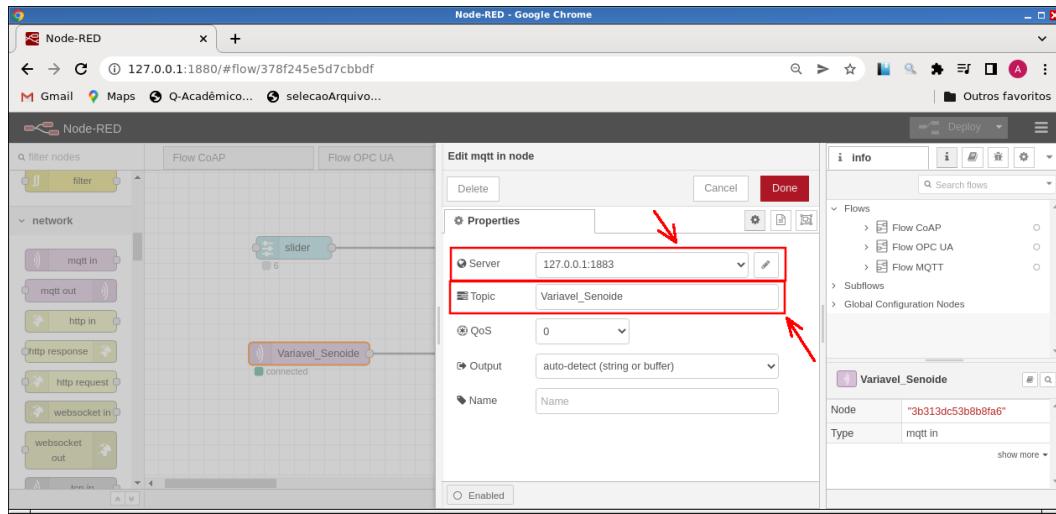


Figura 35.5: Configuração do bloco “mqtt in”.

35.3 Sistema executando

A Figura 35.6 apresenta o dashboard montado executando, acessando o programa da Seção 35.1.

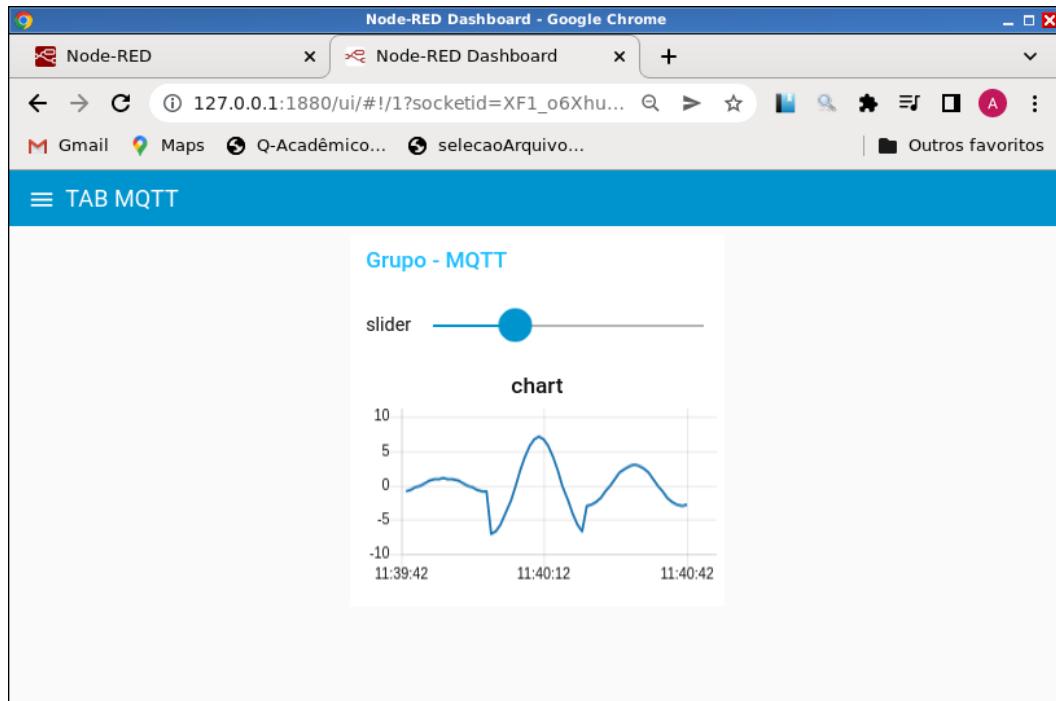


Figura 35.6: Sistema executando.

Parte VII

**Experimentos e considerações práticas
sobre CoAP (*Constrained Application
Protocol*)**

Capítulo 36

Introdução ao padrão CoAP

O padrão CoAP (*Constrained Application Protocol*) é um protocolo de transferência da web especializado para uso com nós e redes restritas na Internet das Coisas (Coap Technology 2021).

O protocolo é projetado para aplicações máquina a máquina (*Machine to Machine*, M2M), como energia inteligente e automação predial (Coap Technology 2021).

Modelo REST para pequenos dispositivos

Como o HTTP, o CoAP é baseado no modelo REST extremamente bem-sucedido: os servidores disponibilizam recursos em uma URL e os clientes acessam esses recursos usando métodos como GET, PUT, POST e DELETE (Coap Technology 2021).

Transferência de habilidades existentes

Do ponto de vista do desenvolvedor, o CoAP se parece muito com o HTTP. Obter um valor de um sensor não é muito diferente de obter um valor de uma API da web (Coap Technology 2021).

Pronto para integração

Como o HTTP e o CoAP compartilham o modelo REST, eles podem ser facilmente conectados usando proxies de protocolo cruzado agnósticos de aplicativos. Um cliente Web pode nem perceber que acabou de acessar um recurso de sensor (Coap Technology 2021).

Escolha seu modelo de dados

Como o HTTP, o CoAP pode transportar diferentes tipos de carga útil e pode identificar qual tipo de carga útil está sendo usado. CoAP integra-se com XML, JSON, CBOR ou qualquer formato de dados de sua escolha (Coap Technology 2021).

Feito para bilhões de nós

A Internet das Coisas precisará de bilhões de nós, muitos dos quais deverão ser baratos. O

CoAP foi projetado para funcionar em microcontroladores com apenas 10 KiB de RAM e 100 KiB de espaço de código (RFC 7228) (Coap Technology 2021).

Mantenha o desperdício sob controle

O CoAP foi projetado para usar o mínimo de recursos, tanto no dispositivo quanto na rede. Em vez de uma pilha de transporte complexa, ele funciona com UDP no IP. Um cabeçalho fixo de 4 bytes e uma codificação compacta de opções permitem pequenas mensagens que causam pouca ou nenhuma fragmentação na camada de link. Muitos servidores podem operar de forma completamente sem estado (Coap Technology 2021).

Descobrimento integrado

O diretório de recursos CoAP fornece uma maneira de descobrir as propriedades dos nós em sua rede (Coap Technology 2021).

Protocolo bem desenhado

O CoAP foi desenvolvido como um documento de padrões da Internet, RFC 7252. O protocolo foi projetado para durar décadas. Questões difíceis, como controle de congestionamento, não foram varridas para debaixo do tapete, mas foram tratadas usando o que há de mais moderno (Coap Technology 2021).

Seguro

A Internet das Coisas não pode se espalhar enquanto puder ser explorada por hackers à vontade. O CoAP não apenas elogia a segurança da boca para fora; na verdade, fornece uma segurança forte. A escolha padrão dos parâmetros DTLS do CoAP é equivalente a chaves RSA de 3072 bits, mas ainda funciona bem nos nós menores (Coap Technology 2021).

Capítulo 37

CoAP vs MQTT

Os padrões de comunicação CoAP e MQTT são adequados para aplicações de IoT. Como ambos estão disponíveis entre os exemplos disponibilizados pelo ESP-IDF, que é a ferramenta de programação dos microcontroladores ESP, torna-se fácil a implementação de comunicação nesses padrões.

Embora sejam dois protocolos adequados para IoT, os padrões CoAP e MQTT apresentam diferenças significativas. É importante entender essas diferenças para realizar a escolha adequada. Também podem existir situações onde esses protocolos se complementem, cada um atuando uma parte de um sistema mais complexo.

A Figura 37.1 apresenta uma comparação entre o CoAP e o MQTT.

	CoAP	MQTT
Communication kind	Request-response model	Publish-subscribe model
Protocol type	P2P/one-to-one communication protocol	Many-to-many protocol
Messaging mode	Asynchronous and Synchronous	Only Asynchronous
Transport layer protocol	UDP	TCP/IP
RESTful-based	✓	✗
Message labeling	✓	✗

Figura 37.1: Comparação entre os padrões CoAP e MQTT (Nabto 2021).

37.1 Considerações sobre as diferenças entre o CoAP e o MQTT

A estrutura de comunicação do MQTT torna esse padrão mais adequado para situações onde vários dispositivos podem estar interessado em um fonte de informação ou um equipamento pode receber comandos de diversos nós. Para o entendimento da forma de operação do MQTT deve-se lembrar que é um padrão que depende de um nó central, o Broker. Com o Broker atuando na nuvem esse padrão torna muito fácil a transmissão de variáveis através da internet. A seguir é apresentada uma lista de aplicações onde considero que o MQTT seria adequado.

Exemplos de aplicações onde o padrão **MQTT** poderia ser usado:

- Uma estação de coleta de dados, como uma estação meteorológica, acessada por vários interessados nas informações coletadas.
- Um sistema de rastreamento com vários interessados na informação de posição dos veículos, como os aplicativos que informam a localização dos ônibus em uma cidade.
- Equipamentos de automação residencial que devam ser acessados por vários dispositivos, como celulares e ou assistentes (como Alexa, Siri ou Google Assistente).
- Equipamentos de automação industrial ou comercial que devam gerar informações úteis para vários dispositivos ou aplicações, como módulos de programas ERP (*Enterprise Resource Planning*).
- Um aplicativo de IHM, executando em um celular ou tablet, usado para executar comandos remotamente através da internet, como no monitoramento de variáveis e comando de válvulas e bombas de uma subestação de tratamento de água (o tempo de resposta não é crítico, sendo aceito um atraso da ordem de alguns segundos).

O padrão CoAP se assemelha aos padrões industriais tradicionais, como o Modbus, onde um mestre interage com os escravos. Esse tipo de comunicação pode ser mais adequado para sistemas onde a agilidade da comunicação seja mais importante que a flexibilidade do MQTT. Para aplicações onde a comunicação envolve apenas dois nós, o CoAP apresenta a conveniência de não necessitar de um nó intermediário, como o Broker usado no padrão MQTT. A seguir é apresentada uma lista de aplicações onde considero que o CoAP seria adequado.

Exemplos de aplicações onde o padrão **CoAP** poderia ser usado:

- Um CLP ou sistema supervisório que acessa equipamentos de campo para receber valores e enviar comandos (se o tempo de resposta não for crítico pode ser usado o MQTT).
- Dois microcontroladores que devam interagir diretamente na operação de uma determinada planta.
- Um aplicativo de IHM, executando em um celular ou tablet, que deva atuar como controle remoto de um determinado dispositivo, como no controle de um veículo ou drone (o tempo de resposta é crítico para o controle).

Capítulo 38

CoAP User Agent Copper (extensão para Google Chrome)

Este capítulo apresenta instruções para instalação e uso do Copper, que é um cliente gráfico para o CoAP.

38.1 Instalação do Copper

Para facilitar o processo o processo será apresentado como uma sequência de passos. As instruções de instalação podem ser encontradas na página oficial do projeto: <https://github.com/mkovatsc/Copper4Cr>.

Passo 1:

Salvar o projeto a partir do GitHub com o seguinte comando:

```
git clone https://github.com/mkovatsc/Copper4Cr.git
```

Passo 2:

Executar um dos comandos a seguir, disponibilizados na pasta salva no passo anterior, conforme o sistema operacional usado:

install.bat (Windows) ou install.sh (linux).

Passo 3:

Acessar o gerenciador de extensões do Google Chrome, conforme apresentado na Figura 38.1.

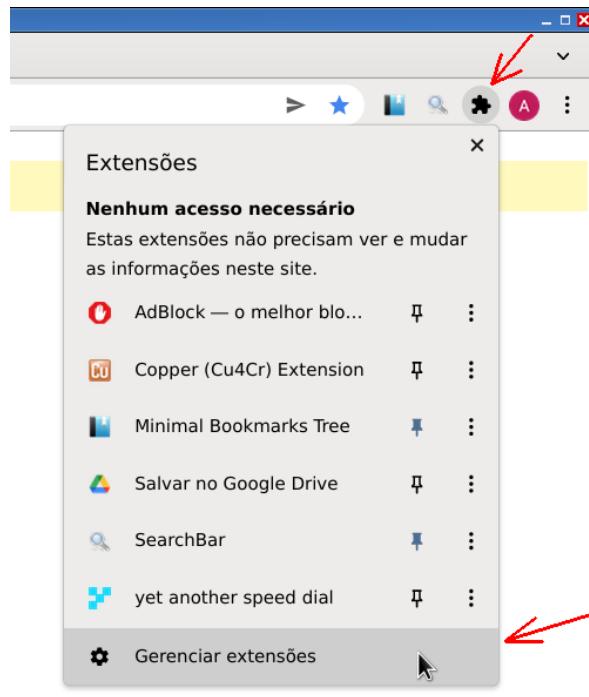


Figura 38.1: Acesso ao gerenciador de extensões do Google Chrome.

Passo 4:

Habilitar a opção “Modo do desenvolvedor”, clicar no botão “Carregar sem compactação” e selecionar o diretório “extension” (disponível no diretório do projeto salvo do GitHub) e clicar no botão “Selecionar”. Repetir o processo selecionando o diretório “app”. Os botões e diretórios envolvidos neste passo são apresentados na Figura 38.2.

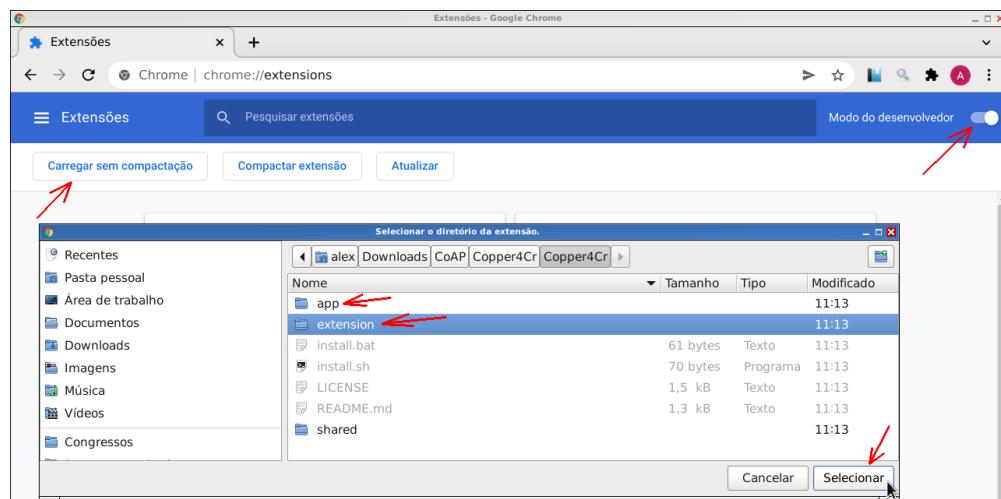


Figura 38.2: Carregando a extensão e o aplicativo Copper no Google Chrome.

Passo 5:

Atualizar o código “**appId**” no arquivo “**ClientPortChrome.js**” com o código obtido no Chrome. O arquivo “**ClientPortChrome.js**” está localizado no diretório “**endpoint**” (no projeto obtido do GitHub). A localização das informações envolvidas neste passo são apresentadas na Figura 38.3.

Atenção: Neste ponto da instalação devem existir duas ocorrência do Copper na lista de extensões do Chrome, o código copiado deve ser o da ocorrência na lista “apps do Chrome”, conforme apresentado na Figura 38.3.

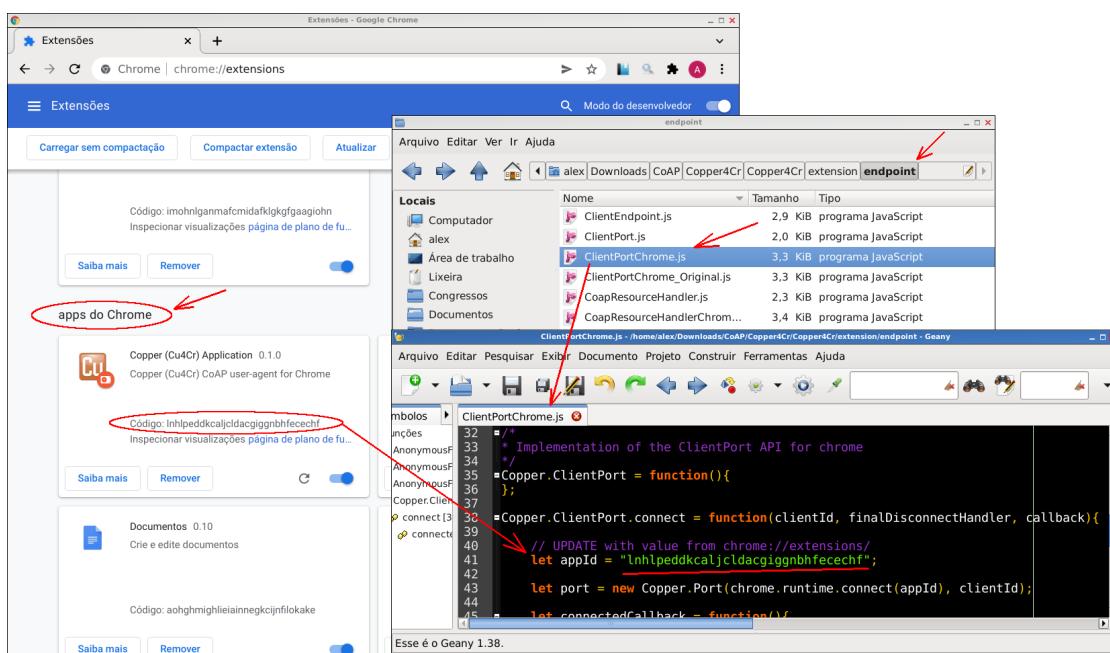


Figura 38.3: Configuração do código da aplicação no arquivo “ClientPortChrome.js”.

38.2 Execução do Copper

Após a instalação, o botão para início do Copper fica disponível na lista de extensões do Chrome, conforme apresentado na Figura 38.4.

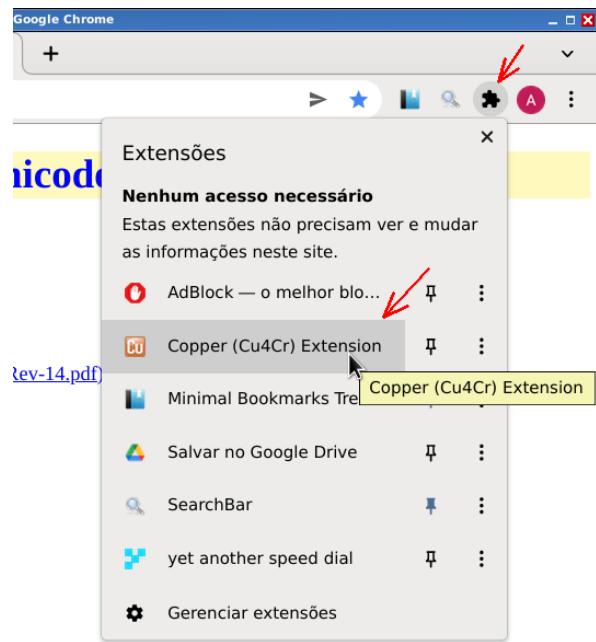


Figura 38.4: Botão para início do Copper.

A Figura 38.5 apresenta a tela inicial do Copper.



Figura 38.5: Tela inicial do Copper.

Capítulo 39

Exemplo de aplicação com padrão CoAP (implementações)

Este capítulo apresenta a implementação de um servidor e de alguns clientes CoAP. Para melhor organização, os testes com esses códigos são apresentados no Capítulo 40.

Os exemplos utilizam a biblioteca **aiocoap**. As instruções de instalação dessa biblioteca podem ser encontradas em:

<https://aiocoap.readthedocs.io/en/latest/installation.html>.

Em resumo, deve-se executar o seguinte comando (digite as aspas se necessário):

```
pip3 install --upgrade "aiocoap[all]"
```

Este capítulo utiliza como referência o tutorial disponível em:

<https://aiocoap.readthedocs.io/en/latest/examples.html>

39.1 Servidor CoAP

Esta seção apresenta o servidor CoAP testado.

Conteúdo do arquivo 01_server_CoAP.py:

```
1 # Arquivo: 01_server_CoAP.py
2 # Alex Brandão Rossow
3
4 # Servidor CoAP
5
6 # Exemplo de referência:
7 # https://aiocoap.readthedocs.io/en/0.4/examples.html
8
9 # Foram adicionados novos nós ao servidor tratando chamadas
10 # GET PUT e POST, conforme detalhado nos comentários inseridos
11 # com as novas funções do servidor.
12
13 import datetime
14 import logging
15
16 import asyncio
17
18 import aiocoap.resource as resource
19 import aiocoap
20
21
22 class BlockResource(resource.Resource):
23     """Example resource which supports the GET and PUT methods. It sends large
24     responses, which trigger blockwise transfer."""
25
26     def __init__(self):
27         super().__init__()
28         self.set_content(b"This is the resource's default content. It is padded \"\
29                         b\"with numbers to be large enough to trigger blockwise \"\
30                         b\"transfer.\n")
31
32     def set_content(self, content):
33         self.content = content
34         while len(self.content) <= 1024:
35             self.content = self.content + b"0123456789\n"
36
37     async def render_get(self, request):
38         return aiocoap.Message(payload=self.content)
39
40     async def render_put(self, request):
```

```
41     print('PUT payload: %s' % request.payload)
42     self.set_content(request.payload)
43     return aiocoap.Message(code=aiocoap.CHANGED, payload=self.content)
44
45
46
47
48 class SeparateLargeResource(resource.Resource):
49     """Example resource which supports the GET method. It uses asyncio.sleep to
50     simulate a long-running operation, and thus forces the protocol to send
51     empty ACK first."""
52
53     def get_link_description(self):
54         # Publish additional data in .well-known/core
55         return dict(**super().get_link_description(), title="A large resource")
56
57     async def render_get(self, request):
58         await asyncio.sleep(3)
59
60         payload = "Three rings for the elven kings under the sky, seven rings \"\
61             \"for dwarven lords in their halls of stone, nine rings for \"\
62             \"mortal men doomed to die, one ring for the dark lord on his \"\
63             \"dark throne.".encode('ascii')
64         return aiocoap.Message(payload=payload)
65
66
67
68
69 class TimeResource(resource.ObservableResource):
70     """Example resource that can be observed. The 'notify' method keeps
71     scheduling itself, and calls 'update_state' to trigger sending
72     notifications."""
73
74     def __init__(self):
75         super().__init__()
76
77         self.handle = None
78
79     def notify(self):
80         self.updated_state()
81         self.reschedule()
82
83     def reschedule(self):
84         self.handle = asyncio.get_event_loop().call_later(5, self.notify)
85
86     def update_observation_count(self, count):
```

```
87     if count and self.handle is None:
88         print("Starting the clock")
89         self.reschedule()
90     if count == 0 and self.handle:
91         print("Stopping the clock")
92         self.handle.cancel()
93         self.handle = None
94
95     async def render_get(self, request):
96         payload = datetime.datetime.now().\
97             strftime("%Y-%m-%d %H:%M").encode('ascii')
98         return aiocoap.Message(payload=payload)
99
100 class WhoAmI(resource.Resource):
101     async def render_get(self, request):
102         text = ["Used protocol: %s." % request.remote.scheme]
103
104         text.append("Request came from %s." % request.remote.hostinfo)
105         text.append("The server address used %s." % request.remote.hostinfo_local)
106
107         claims = list(request.remote.authenticated_claims)
108         if claims:
109             text.append("Authenticated claims of the client: %s." % ", ".join(repr(c)
110                                         for c in claims))
111         else:
112             text.append("No claims authenticated.")
113
114         return aiocoap.Message(content_format=0,
115                                payload="\n".join(text).encode('utf8'))
116
117 # logging setup
118
119 logging.basicConfig(level=logging.INFO)
120 logging.getLogger("coap-server").setLevel(logging.DEBUG)
121
122
123
124 ######
125 # Classes definidas por Alex #####
126 #####
127
128 class Variavel1_Resource(resource.Resource):
129     # Ao receber um comando GET do cliente executa as seguintes ações:
130     # 1) Apresenta uma mensagem solicitando que seja digitado um valor,
131     # 2) Envia o valor para o cliente
```

```
132
133     # Função executada quando o Cliente executa um "GET"
134     async def render_get(self, request):
135         text = []
136         print("Digite o valor da variável myVarTeste1:")
137         try:
138             myVarTeste1 = int(input())
139             text.append(str(myVarTeste1))
140         except ValueError:
141             print("Não é um valor válido!")
142             text.append("Nao eh um valor valido")
143
144         return aiocoap.Message(content_format=0,
145                               payload="".join(text).encode('utf8'))
146
147
148     # Função executada quando o Cliente executa um "PUT"
149     async def render_put(self, request):
150         try:
151             myVarTeste1 = int(request.payload)
152             print("\n")
153             print("O valor recebido pelo método PUT é um inteiro.")
154             print(f'Valor recebido pelo método PUT: {myVarTeste1} \n')
155         except ValueError:
156             print("\n")
157             print("O valor recebido pelo método PUT não é um inteiro.")
158             print(f'Valor recebido pelo método PUT: {request.payload} \n')
159         return aiocoap.Message(code=aiocoap.CHANGED, payload=self.content)
160
161
162
163
164 class Variavel2_Resource(resource.Resource):
165     # Ao receber um comando GET do cliente executa as seguintes ações:
166     # 1) soma 1 à variável "minha_variavel2"
167     # 2) se o valor de "minha_variavel2" atingir 10 esta ézerada
168     # 3) Envia o valor "minha_variavel2" para o cliente.
169
170
171     # Função executada quando o Cliente executa um "GET"
172     async def render_get(self, request):
173         global minha_variavel2
174         text = []
175         print("Digite o valor da variável myVarTeste1:")
176         minha_variavel2 += 1;
177         if (minha_variavel2 == 10):
```

```
178         minha_variavel2 = 0;
179         minha_variavel2_String = str(minha_variavel2);
180
181     return aiocoap.Message(content_format=0,
182                            payload=minha_variavel2_String.encode('utf8'))
183
184
185
186
187 class Variavel3_Resource(resource.Resource):
188     # Ao receber um comando PUT do cliente executa as seguintes ações:
189     # 1) Verifica se o valor recebido é um número ou um texto
190     # 2) Imprime no terminal a indicação se é um número ou um texto
191     # 3) Imprime o valor recebido
192     # 4) Se for um texto apresenta o efeito do ".decode('ascii')"
193
194
195     # Função executada quando o Cliente executa um "PUT"
196     async def render_put(self, request):
197         try:
198             myVarTeste1 = int(request.payload)
199             print("\n")
200             print("O valor recebido pelo método PUT é um inteiro.")
201             print(f'Valor recebido pelo método PUT: {myVarTeste1} \n')
202         except ValueError:
203             print("\n")
204             print("O valor recebido pelo método PUT não é um inteiro.")
205             print(f'Valor recebido pelo método PUT sem decode: {request.payload}')
206             valor_Recebido_decode = request.payload.decode('ascii')
207             print(f'Valor recebido pelo método PUT com decode: {valor_Recebido_decode}')
208             print("\n")
209
210
211
212
213 class VariavelUsuarioResource(resource.Resource):
214     # Função executada quando o Cliente executa um "GET"
215     async def render_get(self, request):
216         texto = "Executou o GET da variável criada pelo usuário"
217         return aiocoap.Message(content_format=0,
218                                payload="".join(texto).encode('utf8'))
219
220     # Função executada quando o Cliente executa um "PUT"
221     async def render_put(self, request):
222         texto = "Executou o PUT da variável criada pelo usuário"
```

```
223     return aiocoap.Message(content_format=0,
224                             payload="".join(texto).encode('utf8'))
225
226
227
228
229 class Variavel4_Resource(resource.Resource):
230     # Ao receber um comando "POST" do cliente executa as seguintes funções:
231     # 1) Armazena o valor recebido como uma string
232     # 2) Cria um novo nó com o nome recebido, os recursos da nova
233     # variável estão implementadas na classe "VariavelUsuarioResource"
234     #
235     # Aqui foi testado o "PUT" ao invés do "POST", funcionou normalmente.
236     async def render_post(self, request):
237     #async def render_put(self, request):
238
239         # Sem o "decode" o valor recebido fica com códigos da mensagem
240         #valorRecebido = request.payload
241         valorRecebido = request.payload.decode('ascii')
242
243         print("\n")
244         print(f'Valor recebido pelo método POST: {valorRecebido} \n')
245
246         # adiciona uma nova variável com o nome passado pelo Cliente
247         root.add_resource(['Variavel-4', valorRecebido], VariavelUsuarioResource())
248
249         texto = "Executou o POST da variavel Variavel-2 \n"
250         return aiocoap.Message(content_format=0,
251                             payload="".join(texto).encode('utf8'))
252 #####
253 #####
254
255
256
257 async def main():
258
259     # As variáveis "global" podem ser acessadas a partir
260     # das classes de tratamento dos nós da rede CoAP.
261     global minha_variavel2
262     minha_variavel2 = 0
263
264     # Resource tree creation
265     global root
266     root = resource.Site()
267
268
```

```
269     root.add_resource(['.well-known', 'core'],
270                         resource.WKCResource(root.get_resources_as_linkheader))
271     root.add_resource(['time'], TimeResource())
272     root.add_resource(['other', 'block'], BlockResource())
273     root.add_resource(['other', 'separate'], SeparateLargeResource())
274     root.add_resource(['whoami'], WhoAmI())
275
276 ######
277 # Adicionado por Alex
278 # - O primeiro parâmetro é o nome do nó criado na rede CoAP.
279 # - O segundo parâmetro é a classe responsável por tratar
280 # as chamadas (GET, PUT, POST ou DELETE)
281     root.add_resource(['Variavel-1'], Variavel1_Resource())
282     root.add_resource(['Variavel-2'], Variavel2_Resource())
283     root.add_resource(['Variavel-3'], Variavel3_Resource())
284     root.add_resource(['Variavel-4'], Variavel4_Resource())
285 #####
286
287     await aiocoap.Context.create_server_context(root)
288
289     # Run forever
290     await asyncio.get_running_loop().create_future()
291
292
293 if __name__ == "__main__":
294     asyncio.run(main())
```

39.2 Cliente CoAP executando a função GET

Esta seção apresenta um cliente CoAP executando a função GET.

Conteúdo do arquivo 02_client_CoAP_GET_Original.py:

```
1 # Arquivo: 02_client_CoAP_GET_Original.py
2 # Alex Brandão Rossow
3
4 # Cliente CoAP
5 # Sólicita por GET o valor do nó "time" implementado no servidor
6
7 # Fonte:
8 # https://aiocoap.readthedocs.io/en/0.4/examples.html
9
10
11 import logging
12 import asyncio
13
14 from aiocoap import *
15
16 logging.basicConfig(level=logging.INFO)
17
18 async def main():
19     protocol = await Context.create_client_context()
20
21     request = Message(code=GET, uri='coap://localhost/time')
22
23     try:
24         response = await protocol.request(request).response
25     except Exception as e:
26         print('Failed to fetch resource:')
27         print(e)
28     else:
29         print('Result: %s\n%s' %(response.code, response.payload))
30
31
32 if __name__ == "__main__":
33     asyncio.get_event_loop().run_until_complete(main())
```

39.3 Cliente CoAP executando a função PUT

Esta seção apresenta um cliente CoAP executando a função PUT.

Conteúdo do arquivo 03_client_CoAP_PUT_Original.py:

```
1 # Arquivo: 03_client_CoAP_PUT_Original.py
2 # Alex Brandão Rossow
3
4 # Cliente CoAP
5 # Escreve por PUT um texto no nó "other/block" implementado no servidor
6 # O texto é uma frase curta ("The quick brown...") repetida 30 vezes
7
8
9 # No servidor a classe "BlockResource", responsável por atender o
10 # acesso ao nó "other/block" realiza as seguintes funções
11 # (pale função "render_put"):
12 # 1) Recebe o conteúdo enviado pelo cliente pela função "PUT" e
13 # imprime o conteúdo (no prompt do servidor).
14 # 2) Retorna o mesmo conteúdo recebido de volta para o cliente.
15
16 # Fonte:
17 # https://aiocoap.readthedocs.io/en/0.4/examples.html
18
19
20 import logging
21 import asyncio
22
23 from aiocoap import *
24
25 logging.basicConfig(level=logging.INFO)
26
27 async def main():
28     """Perform a single PUT request to localhost on the default port, URI
29     "/other/block". The request is sent 2 seconds after initialization.
30
31     The payload is bigger than 1kB, and thus sent as several blocks."""
32
33     context = await Context.create_client_context()
34
35     await asyncio.sleep(2)
36
37     payload = b"The quick brown fox jumps over the lazy dog.\n" * 30
38     request = Message(code=PUT, payload=payload, uri="coap://localhost/other/block")
39
40     response = await context.request(request).response
```

```
41     print('Result: %s\n%r'%(response.code, response.payload))
42
43
44 #####
45 # Adicionado por Alex
46 # Foi necessário o "decode('ascii')" para executar as quebras
47 # de linha comandadas pelos "\n" inseridos na mensagem
48 respostaASCII = request.payload.decode('ascii')
49 print("")
50 print(f'Reposta ASCII \n{respostaASCII}')
51 #####
52
53
54
55
56 if __name__ == "__main__":
57     asyncio.get_event_loop().run_until_complete(main())
```

Capítulo 40

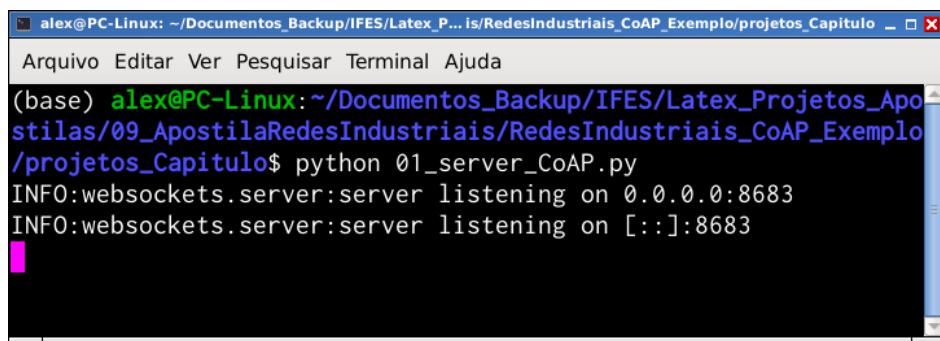
Exemplo de aplicação com padrão CoAP (testes)

Este capítulo apresenta alguns testes realizados com o servidor e com os clientes CoAP implementados no Capítulo 39

40.1 Execução do servidor CoAP

Esta seção apresenta a execução do servidor CoAP implementado na Seção 39.1.

A Figura 40.1 apresenta o servidor iniciado em um terminal, com a indicação do IP e da porta usadas para a comunicação.

A screenshot of a Linux terminal window titled "alex@PC-Linux: ~/Documentos_Backup/IFES/Latex_Projetos_Apostilas/09_ApostilaRedes Industriais/Redes Industriais_CoAP_Exemplo/projetos_Capitulo - Terminal". The window contains the following text:

```
Arquivo Editar Ver Pesquisar Terminal Ajuda
(base) alex@PC-Linux:~/Documentos_Backup/IFES/Latex_Projetos_Apostilas/09_ApostilaRedes Industriais/Redes Industriais_CoAP_Exemplo/projetos_Capitulo$ python 01_server_CoAP.py
INFO:websocket.server:server listening on 0.0.0.0:8683
INFO:websocket.server:server listening on [::]:8683
```

The terminal window has a standard blue title bar and a black body with white text. The scroll bar on the right side is partially visible.

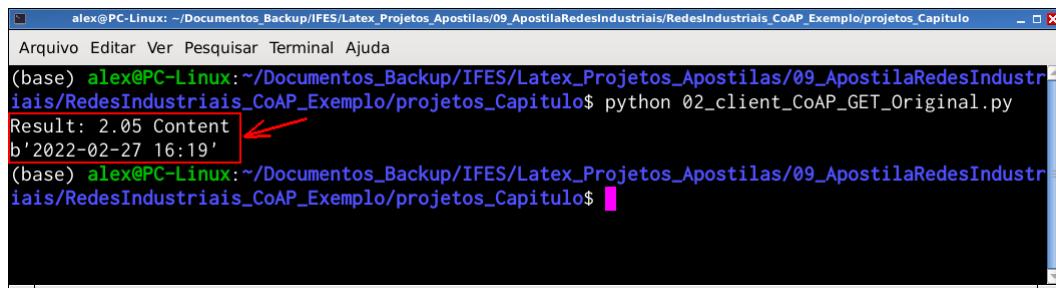
Figura 40.1: Servidor CoAP iniciado.

Nota: O IP “0.0.0.0” indica que o servidor é acessível por qualquer interface do computador, como pelo endereço de localhost (127.0.0.1) ou pelo IP da interface de rede (ethernet ou Wifi).

40.2 Execução do cliente CoAP com implementação da função GET

Essa seção apresenta a execução do cliente implementado na Seção 39.2, após o inicio do servidor, apresentado na Seção 40.1

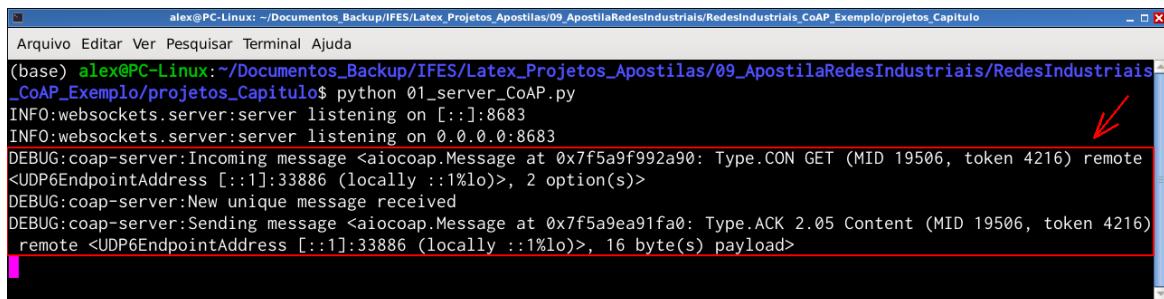
A Figura 40.2 apresenta o terminal onde foi executado o cliente CoAP. A resposta recebida do servidor é apresentada em destaque com um retângulo vermelho. Neste exemplo a função GET do nó acessado pelo cliente (nó “time”) retorna a data e a hora atual.



```
alex@PC-Linux: ~/Documentos_Backup/IFES/Latex_Projetos_Apostilas/09_ApostilaRedesIndustriais/RedesIndustriais_CoAP_Exemplo/projetos_Capitulo
Arquivo Editar Ver Pesquisar Terminal Ajuda
(base) alex@PC-Linux:~/Documentos_Backup/IFES/Latex_Projetos_Apostilas/09_ApostilaRedesIndustriais/RedesIndustriais_CoAP_Exemplo/projetos_Capitulo$ python 02_client_CoAP_GET_Original.py
Result: 2.05 Content
b'2022-02-27 16:19'
(base) alex@PC-Linux:~/Documentos_Backup/IFES/Latex_Projetos_Apostilas/09_ApostilaRedesIndustriais/RedesIndustriais_CoAP_Exemplo/projetos_Capitulo$
```

Figura 40.2: Cliente CoAP executando a função GET do nó "time".

A Figura 40.3 apresenta a terminal onde foi iniciado o servidor. Observa-se em destaque algumas informações geradas de forma automática quando o servidor atendeu à requisição do cliente, que acessou a função GET do nó “time”.



```
alex@PC-Linux: ~/Documentos_Backup/IFES/Latex_Projetos_Apostilas/09_ApostilaRedesIndustriais/RedesIndustriais_CoAP_Exemplo/projetos_Capitulo
Arquivo Editar Ver Pesquisar Terminal Ajuda
(base) alex@PC-Linux:~/Documentos_Backup/IFES/Latex_Projetos_Apostilas/09_ApostilaRedesIndustriais/RedesIndustriais_CoAP_Exemplo/projetos_Capitulo$ python 01_server_CoAP.py
INFO:websocket.server:server listening on [::]:8683
INFO:websocket.server:server listening on 0.0.0.0:8683
DEBUG:coop-server:Incoming message <aiocoap.Message at 0x7f5a9f992a90: Type.CON GET (MID 19506, token 4216) remote <UDP6EndpointAddress [::1]:33886 (locally ::1%lo)>, 2 option(s)>
DEBUG:coop-server>New unique message received
DEBUG:coop-server:Sending message <aiocoap.Message at 0x7f5a9ea91fa0: Type.ACK 2.05 Content (MID 19506, token 4216) remote <UDP6EndpointAddress [::1]:33886 (locally ::1%lo)>, 16 byte(s) payload>
```

Figura 40.3: Exemplo de mensagens geradas de forma automática no terminal do servidor CoAP ao responder a uma requisição.

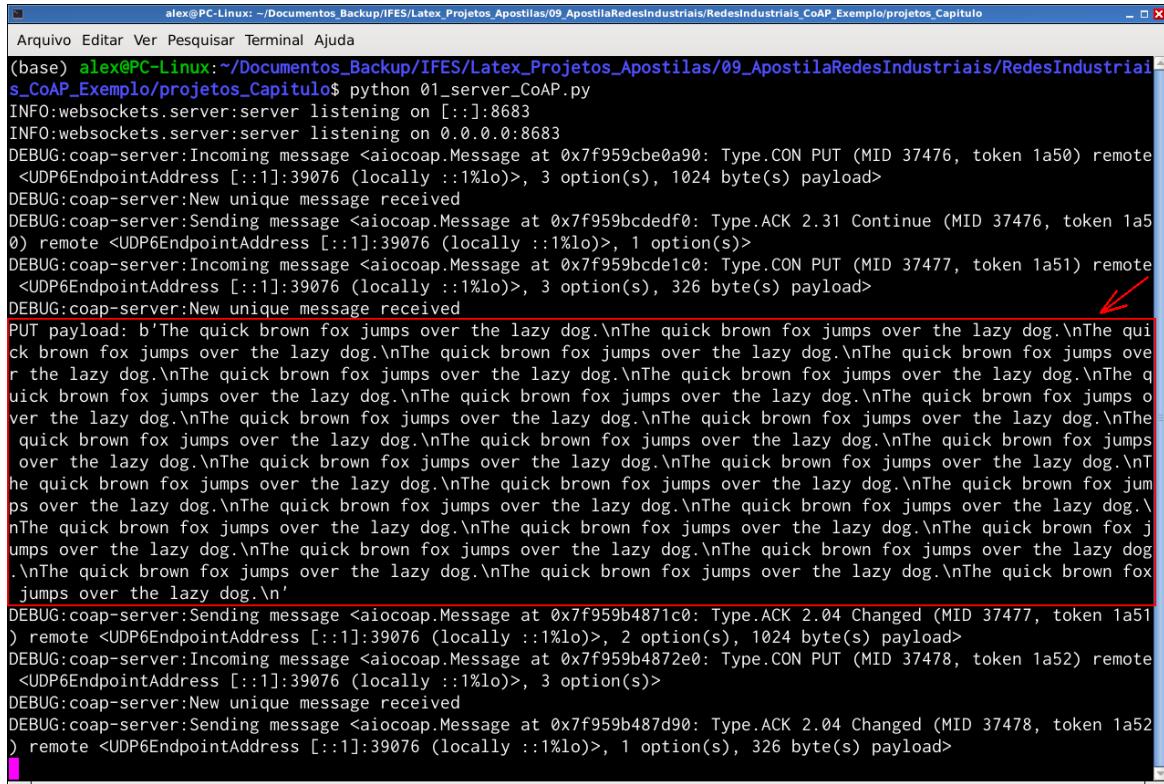
40.3 Execução do cliente CoAP com implementação da função PUT

Essa seção apresenta a execução do cliente implementado na Seção 39.3, após o inicio do servidor, apresentado na Seção 40.1

A Figura 40.4 apresenta o terminal do cliente CoAP executando o comando PUT do nó “other/block”. Neste exemplo o cliente envia uma mensagem longa para o servidor (*The quick brown fox jumps over the lazy dog.* 30X). A função implementada no servidor retorna a mesma mensagem de volta para o cliente. Por fim o cliente imprime a mensagem recebida do servidor. Na figura é destacada a impressão da mensagem no formato recebido do servidor e a mensagem convertida para ASCII, onde o código “\n” executa a função de quebra de linha desejada.

Figura 40.4: Cliente CoAP executando o comando PUT do nó “other/blck” e apresentando a resposta retornada pelo servidor.

A Figura 40.5 apresenta o terminal do servidor apresentando o resultado do comando PUT do nó “other/block”, executado pelo cliente. Observa-se que a função implementada no servidor imprime no terminal o conteúdo recebido do cliente (*The quick brown fox jumps over the lazy dog. 30X*).



A terminal window titled "alex@PC-Linux: ~/Documentos_Backup/IFES/Latex_Projetos_Apostilas/09_ApostilaRedesIndustriais/RedesIndustriais_CoAP_Exemplo/projetos_Capitulo". The window shows the following text:

```
alex@PC-Linux: ~/Documentos_Backup/IFES/Latex_Projetos_Apostilas/09_ApostilaRedesIndustriais/RedesIndustriais_CoAP_Exemplo/projetos_Capitulo
Arquivo Editar Ver Pesquisar Terminal Ajuda
(base) alex@PC-Linux:~/Documentos_Backup/IFES/Latex_Projetos_Apostilas/09_ApostilaRedesIndustriais/RedesIndustriais_CoAP_Exemplo/projetos_Capitulo$ python 01_server_CoAP.py
INFO:websockets.server:server listening on [::]:8683
INFO:websockets.server:server listening on 0.0.0.0:8683
DEBUG:coap-server:Incoming message <aiocoap.Message at 0x7f959cbe0a90: Type.CON PUT (MID 37476, token 1a50) remote <UDP6EndpointAddress [::1]:39076 (locally ::1%lo)>, 3 option(s), 1024 byte(s) payload>
DEBUG:coap-server>New unique message received
DEBUG:coap-server:Sending message <aiocoap.Message at 0x7f959bcdedf0: Type.ACK 2.31 Continue (MID 37476, token 1a50) remote <UDP6EndpointAddress [::1]:39076 (locally ::1%lo)>, 1 option(s)>
DEBUG:coap-server:Incoming message <aiocoap.Message at 0x7f959bcde1c0: Type.CON PUT (MID 37477, token 1a51) remote <UDP6EndpointAddress [::1]:39076 (locally ::1%lo)>, 3 option(s), 326 byte(s) payload>
DEBUG:coap-server>New unique message received
PUT payload: b'The quick brown fox jumps over the lazy dog.\nThe quick brown fox jumps over the lazy dog.\n'
DEBUG:coap-server:Sending message <aiocoap.Message at 0x7f959b4871c0: Type.ACK 2.04 Changed (MID 37477, token 1a51) remote <UDP6EndpointAddress [::1]:39076 (locally ::1%lo)>, 2 option(s), 1024 byte(s) payload>
DEBUG:coap-server:Incoming message <aiocoap.Message at 0x7f959b4872e0: Type.CON PUT (MID 37478, token 1a52) remote <UDP6EndpointAddress [::1]:39076 (locally ::1%lo)>, 3 option(s)>
DEBUG:coap-server>New unique message received
DEBUG:coap-server:Sending message <aiocoap.Message at 0x7f959b487d90: Type.ACK 2.04 Changed (MID 37478, token 1a52) remote <UDP6EndpointAddress [::1]:39076 (locally ::1%lo)>, 1 option(s), 326 byte(s) payload>
```

Figura 40.5: Terminal do servidor CoAP atendendo ao comando PUT no nó “other/block”.

Capítulo 41

Exemplo de aplicação com padrão CoAP (testes com o Copper)

Este capítulo apresenta exemplos de acesso ao servidor implementado na Seção 39.1 utilizando o cliente Copper, apresentado no Capítulo 38.

A Figura 41.1 apresenta o terminal após o início do servidos apresentado na Seção 39.1. Esse servidor será usado em todos os testes com o cliente Copper.

A screenshot of a Linux terminal window titled "alex@PC-Linux: ~/Documentos_Backup/IFES/Latex_Projetos_Apostilas/Redes Industriais/Redes Industriais CoAP_Exemplo/projetos_Capitulo_01_server_CoAP.py". The terminal shows the command "python 01_server_CoAP.py" being run. The output indicates that the server is listening on "0.0.0.0:8683" and "[::]:8683".

```
alex@PC-Linux: ~/Documentos_Backup/IFES/Latex_Projetos_Apostilas/Redes Industriais/Redes Industriais CoAP_Exemplo/projetos_Capitulo_01_server_CoAP.py
(base) alex@PC-Linux: ~/Documentos_Backup/IFES/Latex_Projetos_Apostilas/Redes Industriais/Redes Industriais CoAP_Exemplo/projetos_Capitulo$ python 01_server_CoAP.py
INFO:websockets.server:server listening on 0.0.0.0:8683
INFO:websockets.server:server listening on [::]:8683
```

Figura 41.1: Servidor CoAP iniciado.

A Figura 41.2 apresenta a configuração do endereço do servidor CoAP para acesso pelo Copper. Observa-se na Figura 41.1 que o servidor está escutando no endereço “0.0.0.0” e na porta “8683”. O endereço “0.0.0.0” corresponde à estar ouvindo em todas interfaces de rede do computador, incluindo o endereço de localhost “127.0.0.1”. A porta não é informada para conexão pelo Copper (descoberta empírica).



Figura 41.2: Configuração do endereço do servidor CoAP para conexão pelo Copper.

A Figura 41.3 apresenta a tela após a conexão do Copper ao servidor. Observa-se na barra do lado esquerdo todos os elementos inseridos no servidor, no programa da Seção 39.1. Para atualizar essa lista deve-se clicar no botão “Discover”.

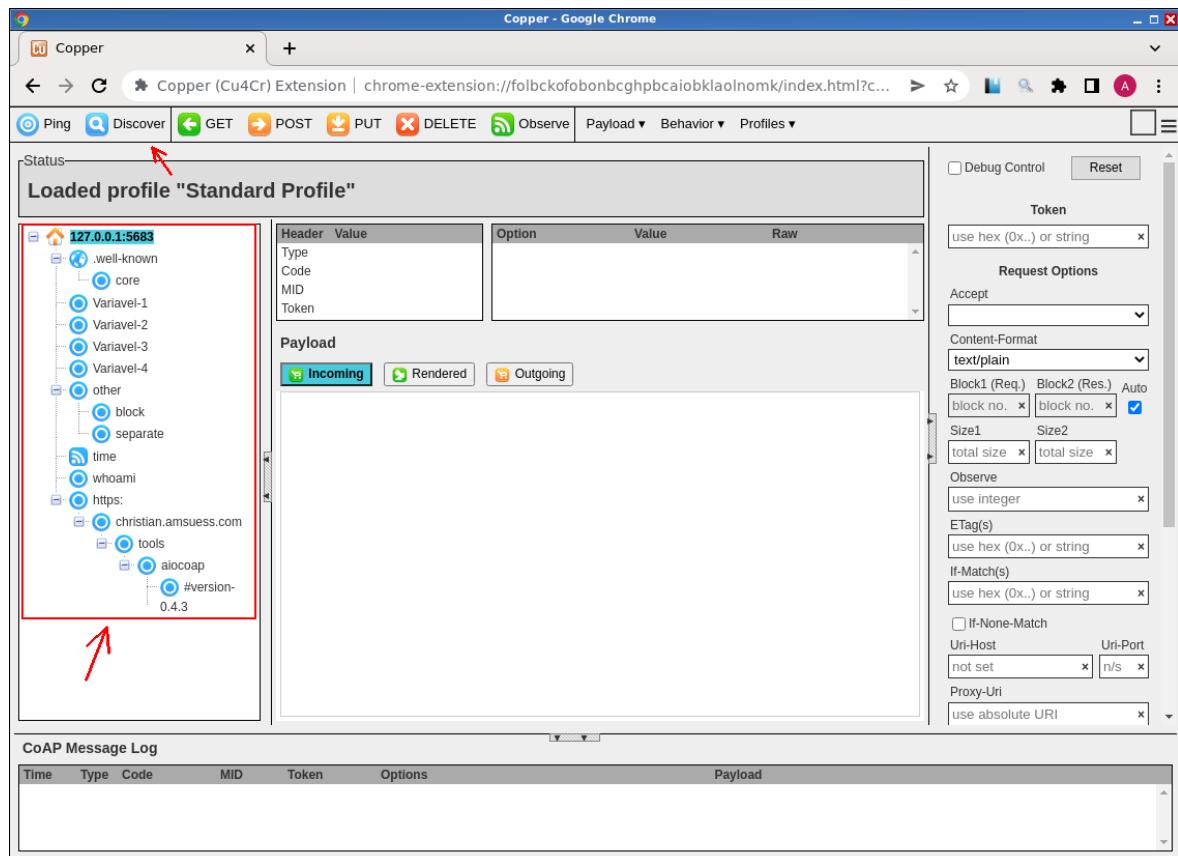


Figura 41.3: Tela do Copper após a conexão ao servidor CoAP.

41.1 Acesso ao nó “time”, método GET

A Figura 41.4 apresenta o resultado do acesso ao método GET do nó “time”. A execução de um comando é feito com os seguintes passos:

1. Seleção do nó desejado na barra da esquerda.
2. Execução do comando desejado, neste caso o comando GET.
3. Observação do resultado na aba “Incomming”.

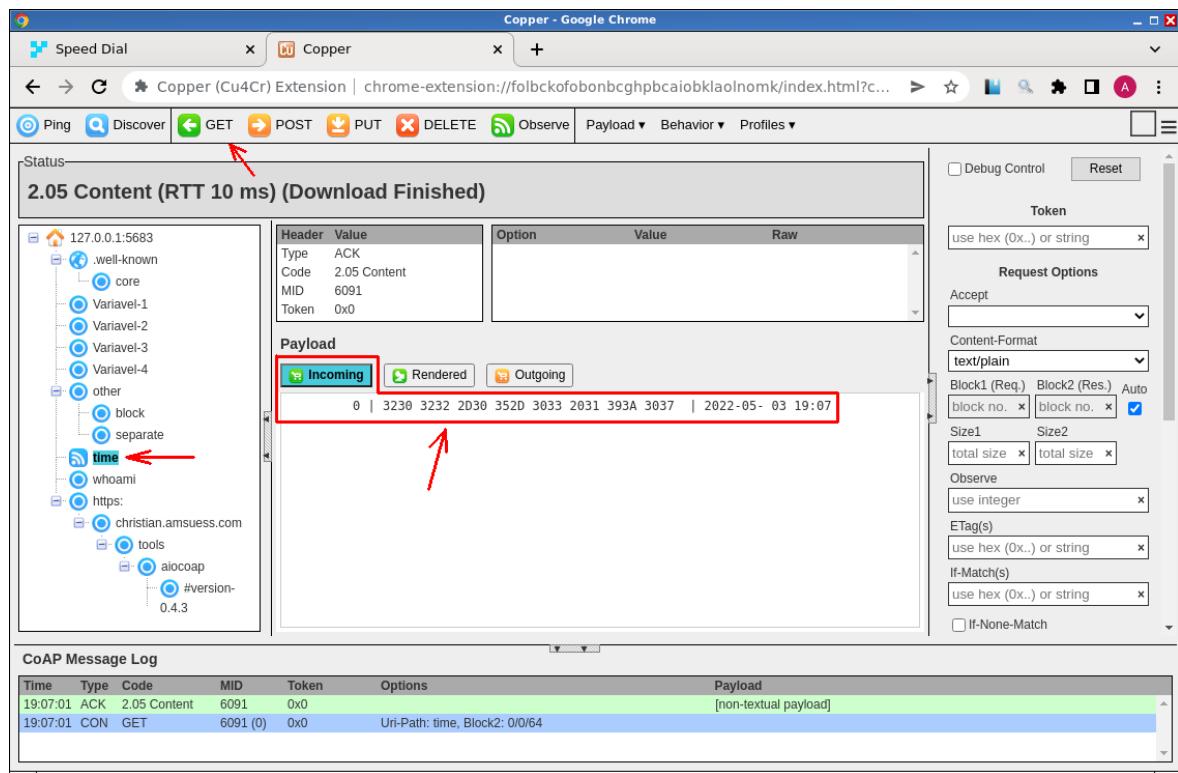


Figura 41.4: Acesso ao método GET do nó “time”.

A Figura 41.5 apresenta as mensagens exibidas no terminal do servidor ao responder à requisição do cliente Copper.

```

alex@PC-Linux: ~/Documentos_Backup/IFES/Latex_Projetos_Apostilas/09_ApostilaRedesIndustriais/RedesIndustriais_CoAP_Exemplo/projetos_Capitulo
Arquivo Editar Ver Pesquisar Terminal Ajuda
(base) alex@PC-Linux:~/Documentos_Backup/IFES/Latex_Projetos_Apostilas/09_ApostilaRedesIndustriais/RedesIndustriais_CoAP_Exemplo/projetos_Capitulo$ python 01_server_CoAP.py
INFO: websockets.server:server listening on 0.0.0.0:8683
INFO: websockets.server:server listening on [::]:8683
DEBUG: coap-server: Incoming message <aiocoap.Message at 0x7fd0f74c4a00: Type.CON GET (MID 6091, empty token)
remote <UDP6EndpointAddress 127.0.0.1:50484 (locally 127.0.0.1%lo)>, 2 option(s)>
DEBUG: coap-server: New unique message received
DEBUG: coap-server: Sending message <aiocoap.Message at 0x7fd0f65d1eb0: Type.ACK 2.05 Content (MID 6091, empty token)
remote <UDP6EndpointAddress 127.0.0.1:50484 (locally 127.0.0.1%lo)>, 16 byte(s) payload>

```

Figura 41.5: Mensagens no terminal do servidor ao responder à requisição do cliente.

41.2 Acesso ao nó “other/block”, método PUT

A Figura 41.6 apresenta a composição da mensagem para envio do cliente Copper para o servidor CoAP. A mensagens será encaminhada para o nó “other/block”, com o comando PUT.

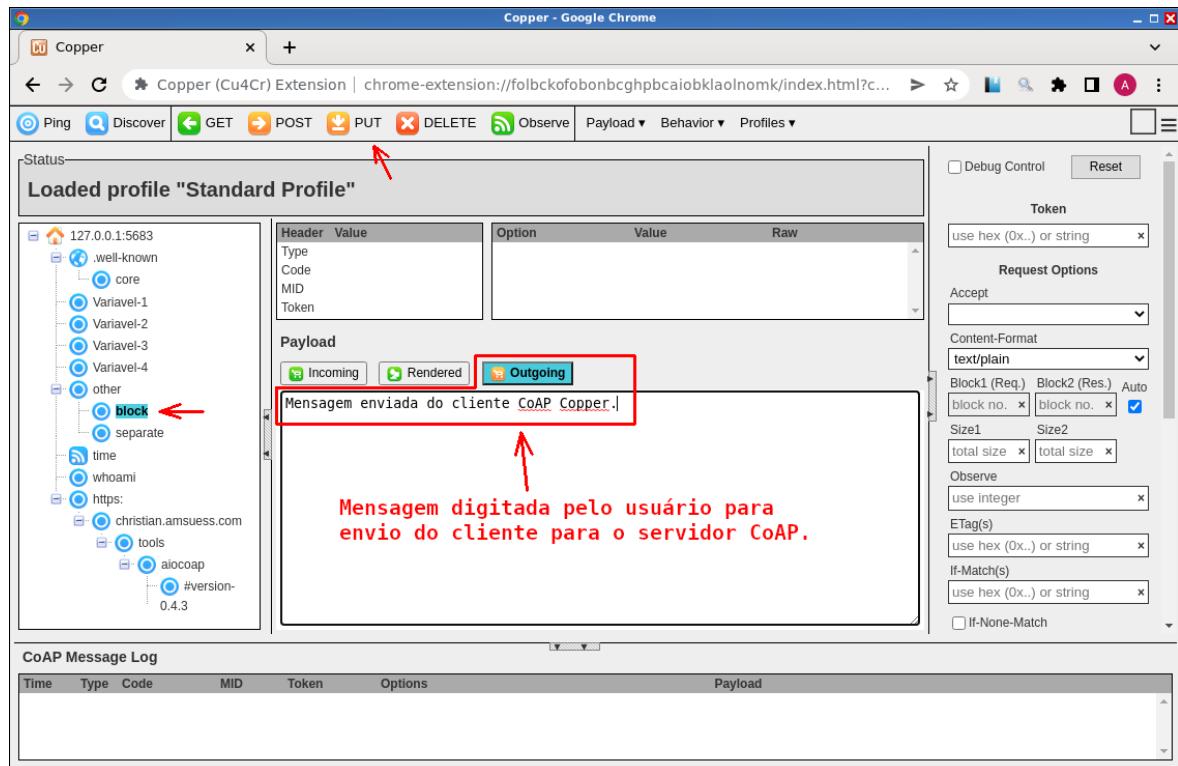


Figura 41.6: Composição da mensagem para envio do cliente para o servidor pelo comando PUT.

A ?? apresenta a resposta retornada pelo servidor após a execução do comando PUT, pelo

cliente Copper, no nó “other/block”. Observa-se que a resposta contém a mensagem original enviada pelo cliente acrescida de uma grande quantidade de números. Esse acréscimo de números para envio da resposta está implementado e documentado na classe “BlockResource” que é quem trata dos chamados do nó “other/block” no servidor implementado na Seção 39.1.

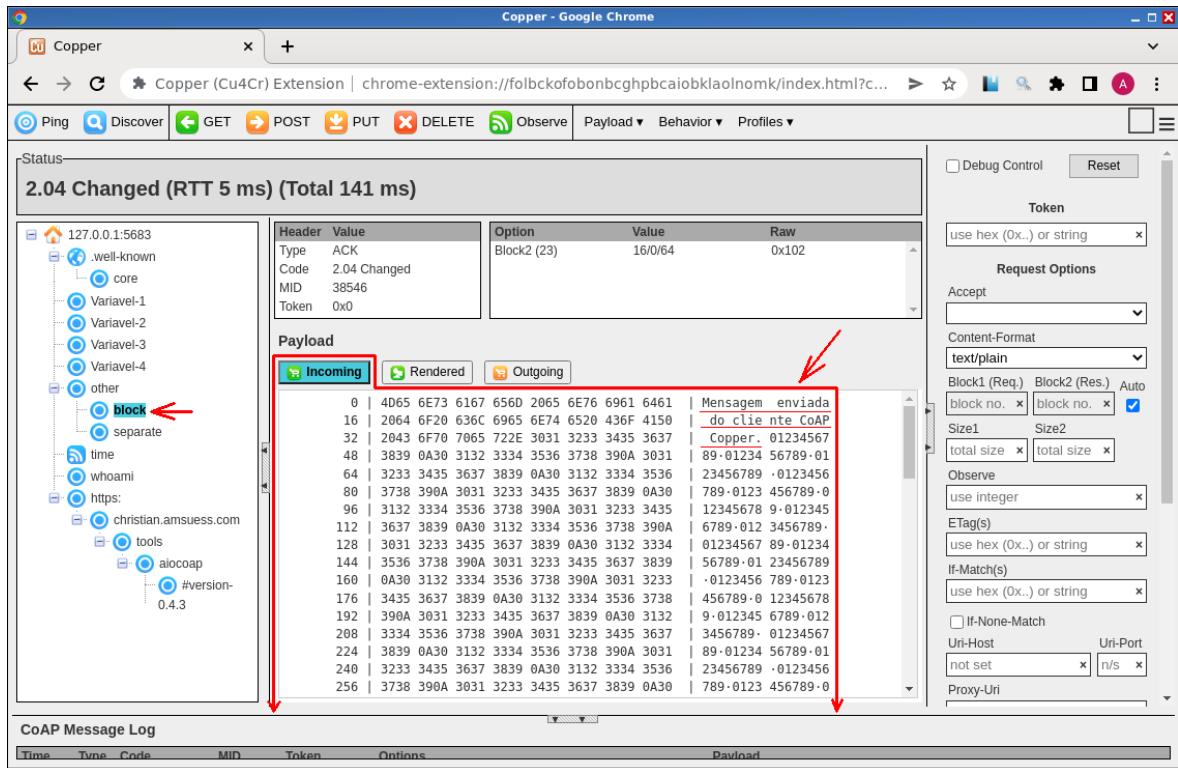


Figura 41.7: Resposta do servidor à execução do comando PUT do nó “other/block”.

41.3 Acesso ao nó “variável-4”, método POST

A Figura 41.8 apresenta a preparação para execução do comando POST na variável “Variável-4”. O comando POST do nó “Variável-4” foi programado na classe “Variavel4_Resource”, no servidor da Seção 39.1, para adicionar um novo nó dentro de “Variável-4”.

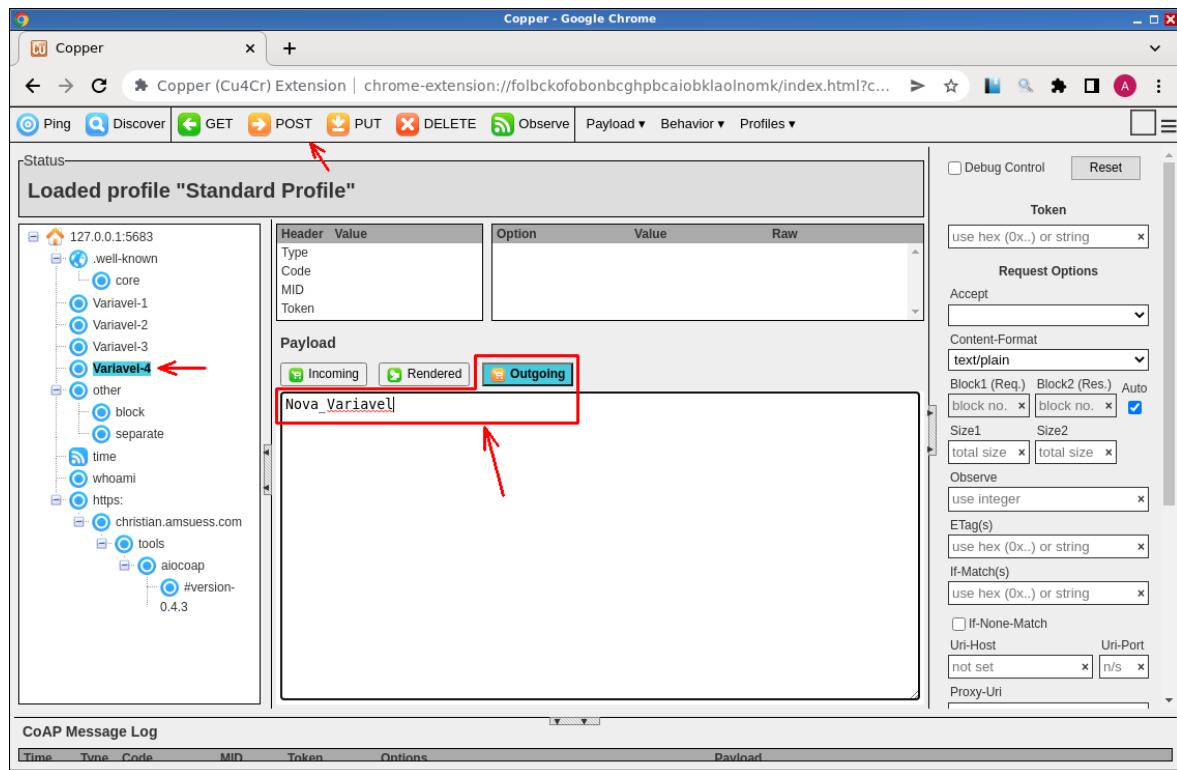


Figura 41.8: Preparação para execução do comando POST no nó “Variavel-4”.

A Figura 41.9 apresenta o resultado da execução do comando POST do nó “Variavel-4”. A função responsável pela criação do novo nó foi implementada na classe “Variavel4_Resource”, no servidor da Seção 39.1. Para a atualização da árvore com os nós foi usado o botão “Discover”.

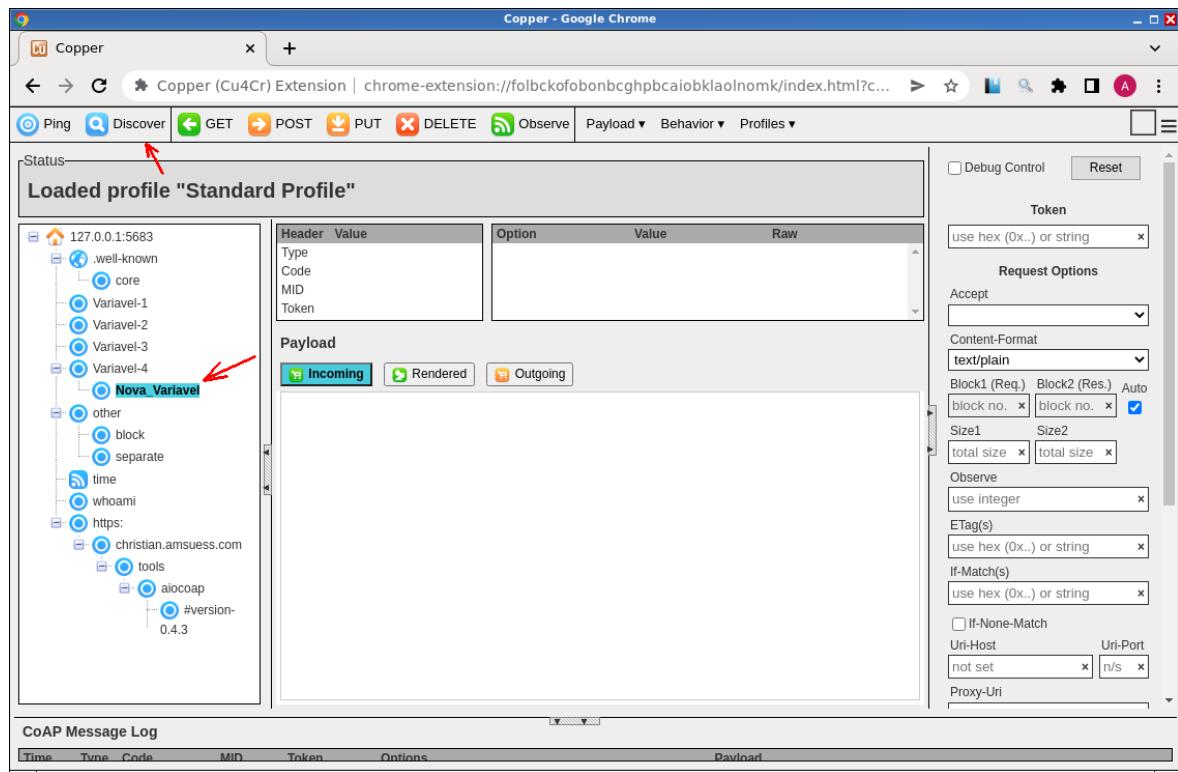


Figura 41.9: Resultado do comando POST no nó “Variavel-4”.

Capítulo 42

Exemplo de aplicação com padrão CoAP (testes com o Node-RED)

Este capítulo apresenta o acesso ao servidor CoAP da Seção 39.1 a partir do Node-RED.

42.1 Acesso ao nó “time”, função GET

A Figura 42.1 apresenta o flow criado no Node-RED para realização do teste. O flow apresenta os seguintes blocos:

- **Button** (grupo **Dashboard**): usado para disparar o comando GET;
- **coap request**: responsável por acessar o servidor;
- **text** (grupo **Dashboard**): usado para exibir a resposta do servidor CoAP.

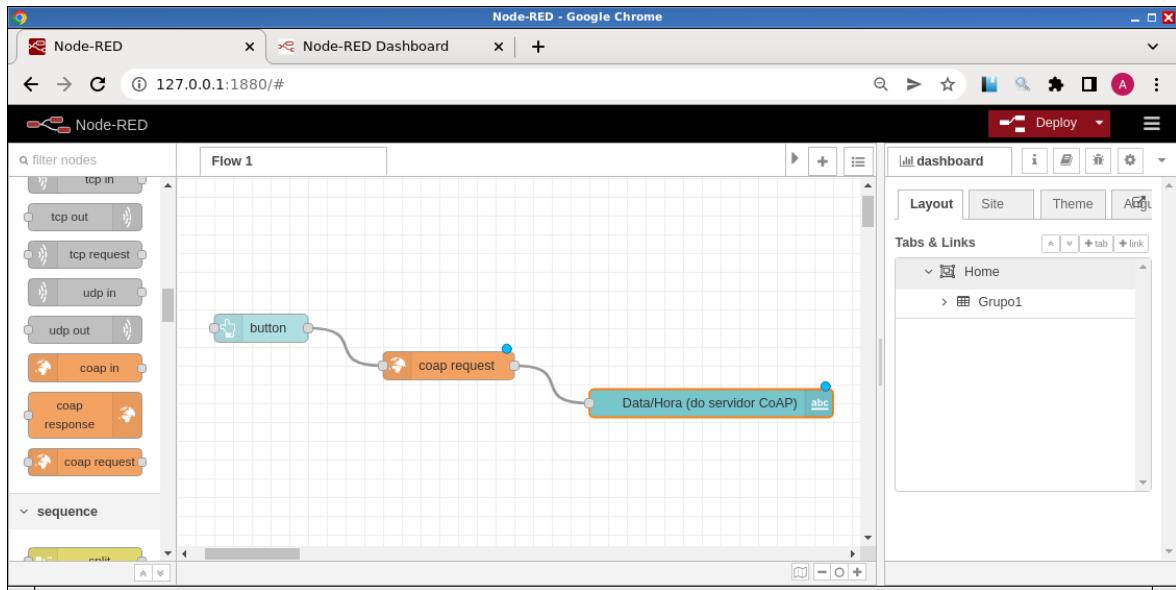


Figura 42.1: Flow do Node-RED para acesso ao servidor CoAP.

A Figura 42.2 apresenta a configuração do bloco “coap request” para acesso ao nó “time” e execução da função GET.

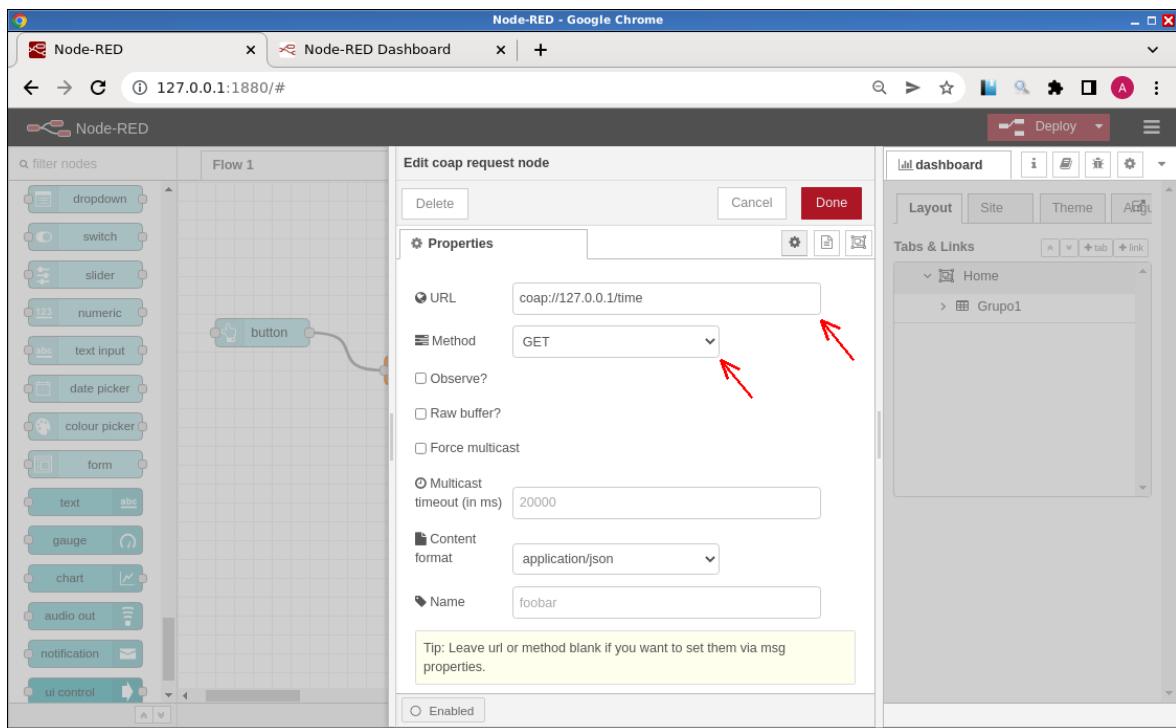


Figura 42.2: Configuração do bloco “coap request”.

A Figura 42.3 apresenta o dashboard executando. Cada vez que o botão é pressionado a

informação de data/hora é atualizada a partir do servidor CoAP implementado na Seção 39.1.

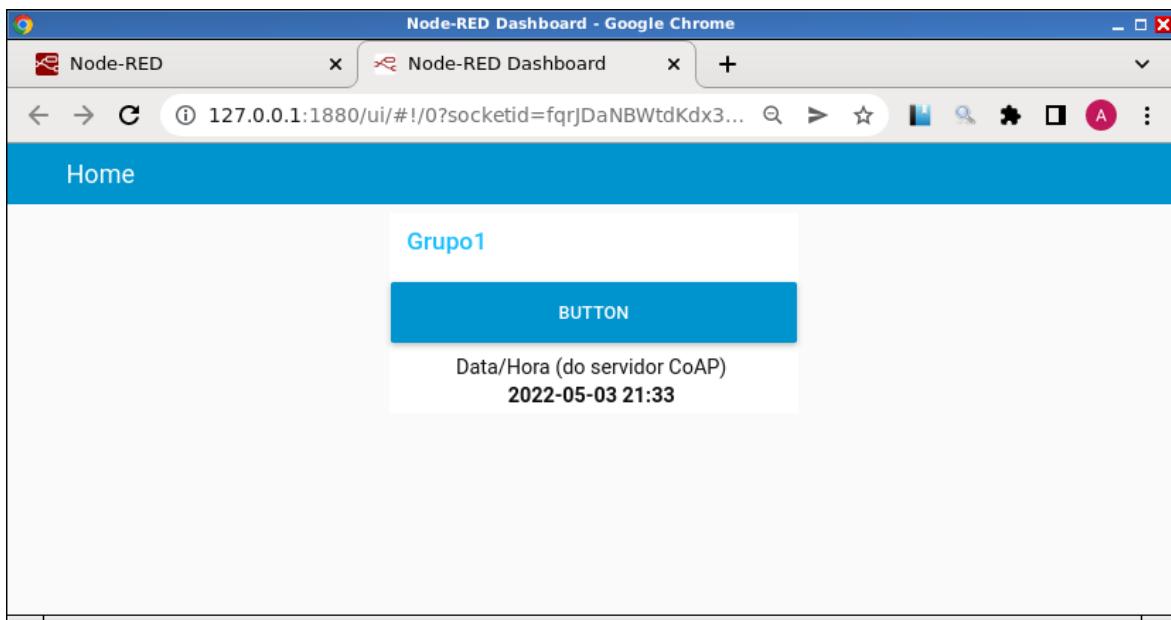


Figura 42.3: Dashboard apresentando as informações de data/hora obtidas do servidor CoAP.

Parte VIII

Experimentos e considerações práticas sobre Bluetooth

Capítulo 43

Introdução

Este capítulo apresenta considerações práticas sobre o Bluetooth.

Capítulo 44

Programação com Soquete Bluetooth (*Bluetooth Socket*) usando Python PyBluez

Este capítulo apresenta uma introdução à programação Bluetooth com a biblioteca PyBluez, um módulo Python Bluetooth. A programação de soquete Bluetooth é semelhante à programação de soquete usada em conexões TCP/IP.

O tutorial de referência usado neste capítulo pode ser encontrado no seguinte endereço:

<https://roboticsknowledgebase.com/wiki/networking/bluetooth-sockets/>.

44.1 Endereços e dispositivos Bluetooth

Cada chip Bluetooth fabricado é impresso com um endereço exclusivo de 48 bits, conhecido como endereço Bluetooth ou endereço do dispositivo. É de natureza idêntica aos endereços MAC de Ethernet e é usado para identificar um determinado dispositivo Bluetooth Robotics Knowledgebase 2017.

A seguir algumas definições importantes:

- **Nome do dispositivo (*Device Name*):** É um nome legível por humanos para um dispositivo bluetooth e é mostrado ao usuário em vez do endereço Bluetooth do dispositivo. Isso geralmente é configurável pelo usuário e pode ser o mesmo para vários dispositivos. É responsabilidade do programa cliente traduzir o nome para os endereços bluetooth correspondentes. O TCP/IP tem servidores de nomes para fazer isso (servidores

DNS), mas no caso do Bluetooth, o programa cliente deve aproveitar a proximidade de dispositivos com nomes fornecidos pelo usuário Robotics Knowledgebase 2017.

- **Protocolo de transporte:** Na arquitetura de rede em camadas, o protocolo de transporte é a camada responsável por criar pacotes de dados e garantir a sincronização, robustez e prevenção de perda de dados. Na programação da Internet, os protocolos de transporte usados são **TCP** ou **UDP**, enquanto no caso do Bluetooth os protocolos de transporte mais usados são **RFCOMM** e **L2CAP** Robotics Knowledgebase 2017.
- **Números de porta (Port Numbers):** Uma vez que o endereço numérico e o protocolo de transporte são conhecidos, o mais importante na programação do Bluetooth é escolher um número de porta para comunicação. Sem os números das portas, não seria possível que vários aplicativos no mesmo host utilizassem o mesmo protocolo de transporte. Este conceito é usado na Programação da Internet (aplicações sobre TCP/IP), como por exemplo as portas usadas para HTTP, FTP, SMTP, IMAP, POP3, Modbus TCP, MQTT, CoAP, UPC UA, etc. No protocolo de transporte **L2CAP**, as portas são conhecidas como Multiplexadores de Serviço de Protocolo (*Protocol Service Multiplexers*). O **RFCOMM** possui canais numerados de 1 a 30 para uso. No Bluetooth, o número de portas é muito limitado em comparação com a programação da Internet. Portanto, no Bluetooth, as portas são atribuídas em tempo de execução, dependendo dos requisitos do aplicativo. Os clientes sabem qual descrição de porta estão procurando usando um número de 128 bits chamado *Universally Unique Identifier* (UUID) em tempo de design, usando algo chamado *Service Discovery Protocol* Robotics Knowledgebase 2017.
- **perfis Bluetooth (Bluetooth Profiles):** Devido à natureza de curto alcance do Bluetooth, perfis Bluetooth são necessários para tarefas específicas. Existem perfis separados para troca de informações de localização física, um perfil para uso de impressoras próximas e um perfil para fazer chamadas telefônicas usando modens próximos. Também existe um perfil para reduzir a programação Bluetooth à programação da Internet Robotics Knowledgebase 2017.

44.2 Camada de Transporte no Bluetooth

Existem diferentes protocolos de camada de transporte no Bluetooth, dois dos mais importantes são o RFCOMM e o L2CAP, apresentados a seguir.

- **RFCOMM:** O protocolo RFCOMM é semelhante ao TCP e fornece os mesmos serviços e recursos. É bastante simples de usar em muitos cenários e é usado principalmente

em aplicativos ponto a ponto. Se uma parte dos dados não for entregue dentro de um período fixo de tempo, a conexão será encerrada. No final do aplicativo, uma vez que o número da porta tenha sido definido, ele é semelhante à programação da porta serial e todas as mesmas práticas podem ser usadas como com qualquer dispositivo serial. Esta é a parte mais intuitiva e fácil da programação de soquetes Robotics Knowledgebase 2017.

- **L2CAP:** Este protocolo de transporte é usado principalmente em situações onde cada pacote não é crucial, mas a velocidade de entrega é necessária. É semelhante ao protocolo UDP e é usado por sua simplicidade. Ele envia pacotes de dados confiáveis de comprimento máximo fixo e é bastante personalizável para vários níveis de confiabilidade. Existem três políticas que um aplicativo pode usar:
 1. nunca retransmitir,
 2. retransmitir até a falha total de conexão (o padrão),
 3. descarte um pacote e prosseguir para os dados enfileirados.

44.3 Visualização da pilha de protocolos do Bluetooth

A seguir são apresentadas algumas figuras que fornecem um visualização do posicionamento dos diferentes protocolos na pilha do Bluetooth. Observa-se que são apresentados os padrões do Bluetooth tradicional (*Bluetooth basic rate/enhanced data rate* - BR/EDR) e a versão *Bluetooth Low Energy* (Bluetooth LE ,BLE).

Esta seção não irá detalhar todos protocolos envolvidos. A ideia é somente fornecer uma visualização da localização dos padrões necessários para os exemplos implementados nesta apostila. Os detalhes dos diferentes protocolos podem ser encontrados nas fontes da figuras.

A Figura 44.1 apresenta a arquitetura da pilha de protocolos do Bluetooth.

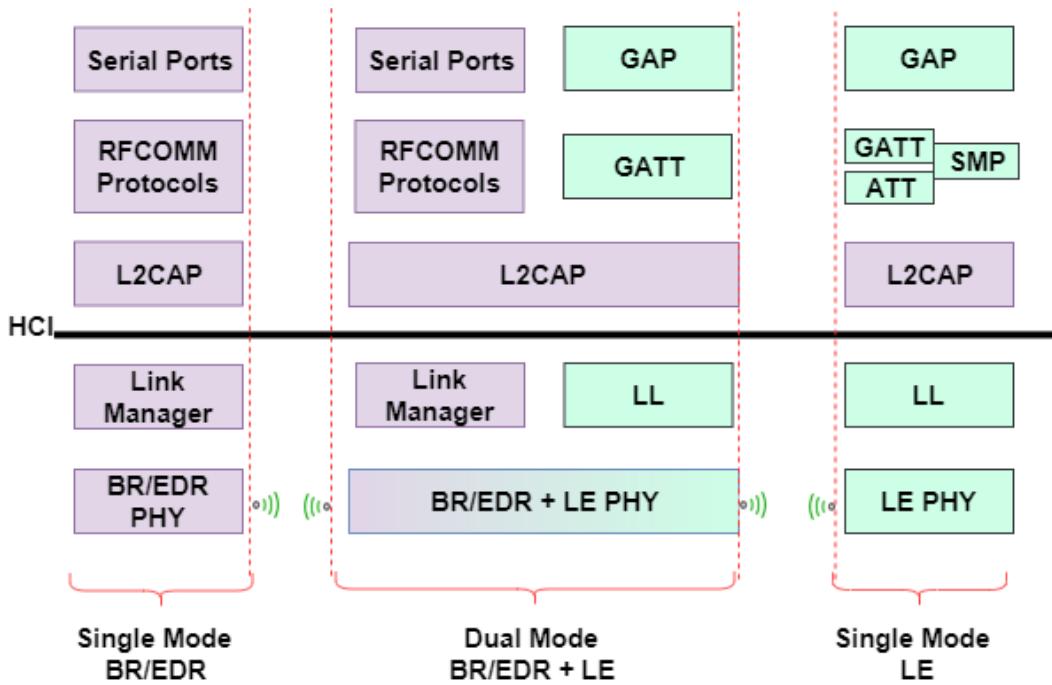


Figura 44.1: Pilha de protocolos do Bluetooth (tradicional e BLE) MathWorks 2021.

Os dispositivos Bluetooth podem ser de um dos dois tipos a seguir MathWorks 2021:

- **Single mode** - Suporta um perfil BR/EDR ou LE.
- **Dual mode** - Suporta os perfis BR/EDR e LE.

O **HCI** fornece uma interface de comando para o rádio BR/EDR, controlador de banda base e gerenciador de link. É uma interface padrão única para acessar os recursos da banda base do Bluetooth, o status do hardware e os registros de controle MathWorks 2021.

Na Figura 44.2 observase- que as portas seriais acima do padrão RFCOMM consistem no padrão **Serial Port Profile (SPP)**.

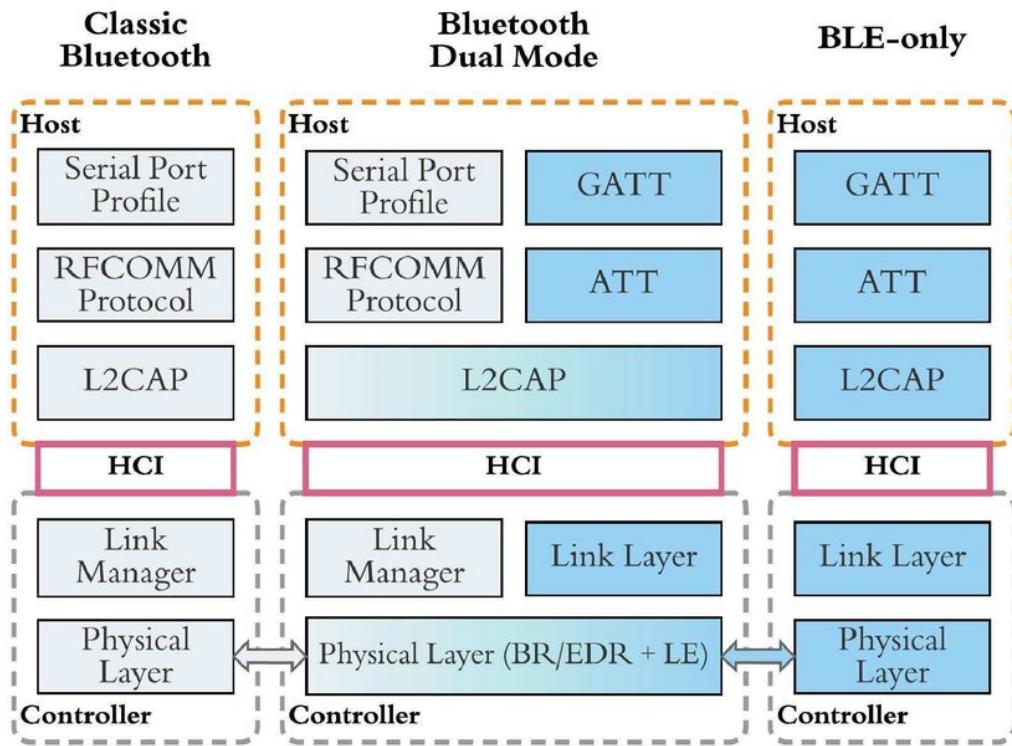


Figura 44.2: Pilha de protocolos Bluetooth Yang et al. 2020.

Capítulo 45

Bluetooth RFCOMM

Esse capítulo apresenta o padrão de comunicação RFCOMM do Bluetooth.

45.1 Bluetooth protocol stack

A principal função do Bluetooth é sua pilha de protocolos (*Bluetooth protocol stack*). Ela define e fornece diferentes tipos de camadas e funcionalidades. O Bluetooth pode executar diferentes aplicativos em diferentes pilhas de protocolo, mas cada uma dessas pilhas de protocolo usa o mesmo link Bluetooth e camadas físicas. O diagrama mostra uma pilha de protocolo Bluetooth completa. Mostra a relação entre os protocolos que utilizam os serviços de outros protocolos quando há uma carga útil a ser transferida no ar. De qualquer forma, os protocolos têm muitas outras relações entre os outros protocolos - por exemplo, alguns protocolos (L2CAP, TCS Binary) usam o LMP para controlar o gerenciador de link. Este capítulo descreve como usar o Bluetooth Framework com o protocolo Classic Bluetooth RFCOMM Soft Service - BTFramework 2022.

Para Bluetooth Low Energy, consulte o artigo "Estrutura do Bluetooth e Bluetooth Low Energy GATT"(<https://www.btf framework.com/gatt.htm>).

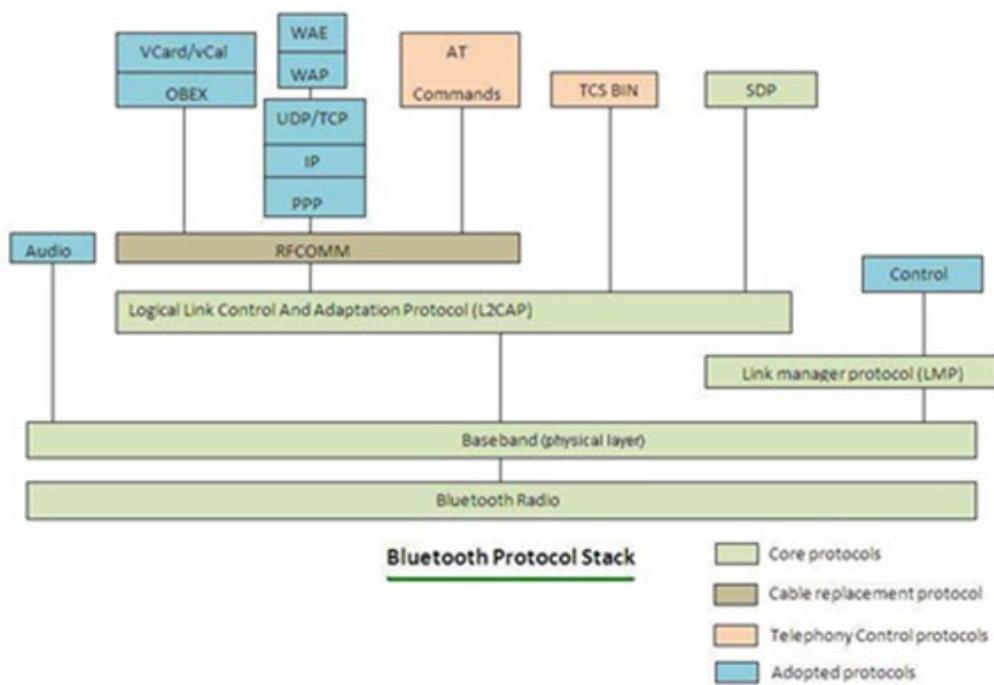


Figura 45.1: Pilha de protocolos do Bluetooth Soft Service - BTFramework 2022.

45.2 O Framework Bluetooth e o RFCOMM

O protocolo Bluetooth RFCOMM (comunicação por radiofrequência) é um conjunto simples de protocolos de transporte, feito sobre o protocolo L2CAP, fornecendo portas seriais RS-232 emuladas Soft Service - BTFramework 2022.

Geralmente se conhece RFCOMM como Perfil de Porta Serial (SPP). Mas isso não é verdade. O Perfil de porta serial é apenas um dos muitos perfis baseados no protocolo RFCOMM Bluetooth. Object Push Profile (OPP), File Transfer Profile (FTP) e muitos outros perfis Bluetooth funcionam acima do RFCOMM (use o RFCOMM como protocolo de transporte) Soft Service - BTFramework 2022.

O Bluetooth Framework oferece suporte nativo ao protocolo Bluetooth Classic RFCOMM nos modos cliente e servidor. Isso significa que o Bluetooth Framework pode atuar como cliente RFCOMM (conecta-se a outros dispositivos habilitados para Bluetooth) ou como servidor RFCOMM (aceita conexão de outros dispositivos habilitados para Bluetooth). Isso também significa que você não precisa criar e/ou usar nenhuma porta COM virtual para se comunicar com dispositivos habilitados para Bluetooth Soft Service - BTFramework 2022.

Capítulo 46

Configuração do Bluetooth Serial Port Profile (SPP) no Linux/Raspberry Pi

Este capítulo apresenta os passos para habilitação da comunicação serial pelo Bluetooth usando o padrão RFCOMM. O tutorial seguido destina-se ao Raspberry Pi, que é um sistema baseado em Debian. Os testes realizados no desenvolvimento deste capítulo foram feitos no Debian. Portanto, acredita-se que esses mesmos passos possam ser seguidos em distribuições baseadas em Debian ou nos seus derivados, como o Ubuntu e o Mint. Com as devidas adaptações, que devem ser encontradas na internet, esse conteúdo deve ser útil para distribuições Linux que não sejam baseadas no Debian.

Antes das configurações apresentadas neste capítulo o aplicativo do celular usado no teste (Serial Bluetooth Terminal, disponibilizado por Kai Morich, na Play Store) acusava que o socket não estava disponível, mesmo com o celular e o computador pareados.

São apresentados somente os passos que foram testados e suficientes para realização de teste. O tutorial completo pode ser encontrado no seguinte endereço:

<https://scribbles.net/setting-up-bluetooth-serial-port-profile-on-raspberry-pi/>.

46.1 Configuração para habilitação da comunicação serial

A seguir os passos executados para habilitar a comunicação serial através do Bluetooth.

Passo 1:

Abrir o arquivo “dbus-org.bluez.service” com algum editor de texto simples, como o “nano” em ambiente de texto ou o “geany” em ambiente gráfico.

Comando:

```
sudo nano /etc/systemd/system/dbus-org.bluez.service
```

ou

```
sudo geany /etc/systemd/system/dbus-org.bluez.service
```

Passo 2:

Procure a linha iniciada com ?ExecStart? e dicsione a flag de compatibilidade “-C” no fim da linha. A linha deve ficar conforme apresentado a seguir:

```
ExecStart=/usr/lib/bluetooth/bluetoothd -C
```

Passo 3:

Adicione a seguinte linha imediatamente após a linha iniciada por “ExecStart”, então salve e feche o arquivo.

```
ExecStartPost=/usr/bin/sdptool add SP
```

Passo 4:

Recarregue o arquivo de configuração, com o seguinte comando:

```
sudo systemctl daemon-reload
```

Passo 5:

Reinic peace o serviço com o seguinte comando:

```
sudo systemctl restart bluetooth.service
```

Nota: Após a execução dos passos anteriores a comunicação funcionou. Porém na busca da solução já tinha sido feita uma configuração anterior, que não se sabe se foi necessária ou não. Recomendo que esse passo extra seja realizado somente se os passos anteriores não funcionarem.

Passo Extra (testar se os passos anteriores não funcionarem):

Essa solução foi obtida no seguinte endereço:

<https://stackoverflow.com/questions/14618277/rfcomm-without-pairing-using-pybluez-on-debian/14827036#14827036>.

Passo Extra (a):

Desabilite o plugin “pnat” editando o arquivo `/etc/bluetooth/main.conf` adicionando a seguinte linha:

```
DisablePlugins = pnat
```

Passo Extra (b):

Reinic peace o bluetoothd com o seguinte comando:

```
sudo invoke-rc.d bluetooth restart
```

Capítulo 47

Exemplo de comunicação com Bluetooth RFCOMM

O exemplo apresentado neste capítulo foi obtido no seguinte endereço:

<https://people.csail.mit.edu/albert/bluez-intro/x232.html>

Para implementação dos exemplos apresentados neste capítulo é importante conferir se há a necessidade de realização das configurações apresentadas no Capítulo 46.

47.1 Instalação da biblioteca “PyBluez” no Python

Para a execução dos exemplos deste capítulo deve-se instalar o pacote “PyBluez”. É esse pacote que vai permitir a execução do comando `import bluetooth`.

A página da biblioteca “PyBluez” pode ser acessada pelo seguinte link:

<https://pypi.org/project/PyBluez/>

A instalação do pacote “PyBluez” pode ser feita com o seguinte comando

```
pip install PyBluez
```

47.2 RFCOMM Server

Conteúdo do arquivo 01_rfcomm-server.py:

```
1 # Alex Brandão Rossow
2
3 # Servidor Bluetooth RFCOMM
4
5 # Exemplo de referência:
6 # https://people.csail.mit.edu/albert/bluez-intro/x232.html
7
8 import bluetooth
9
10 server_sock=bluetooth.BluetoothSocket( bluetooth.RFCOMM )
11
12 port = 1
13 server_sock.bind((" ",port))
14 server_sock.listen(1)
15
16
17 client_sock,address = server_sock.accept()
18 print ("Accepted connection from ",address)
19
20 # Recebe uma mensagem do cliente
21 data = client_sock.recv(1024)
22 print ("received [%s]" % data)
23
24 # Envia uma mensagem para o cliente
25 client_sock.send("Msg Computador para Celular")
26
27
28 client_sock.close()
29 server_sock.close()
```

47.3 RFCOMM Client

Conteúdo do arquivo 02_rfcomm-client.py:

```
1 import bluetooth
2
3 # Endereço do celular
4 bd_addr = "18:01:F1:46:3A:BE"
5
6
```

```
7 # Endereço do computador
8 #bd_addr = "50:63:13:DF:95:05"
9
10 port = 1
11
12 sock=bluetooth.BluetoothSocket( bluetooth.RFCOMM )
13 sock.connect((bd_addr, port))
14
15 sock.send("hello!!")
16
17 sock.close()
```

47.4 Habilitação do Bluetooth no Computador

Nesta seção apresentada a forma de habilitação do Bluetooth em um ambiente gráfico “XFCE” com a interface Bluetooth “blueman-applet”. Outros ambientes gráficos e outros gerenciadores devem ter opções semelhantes.

Para deixar o computador visível para conexão Bluetooth, deve-se acessar a janela de configuração “Adaptadores” por meio do ícone do Blueman-Applet, conforme apresentado na Figura 47.1.

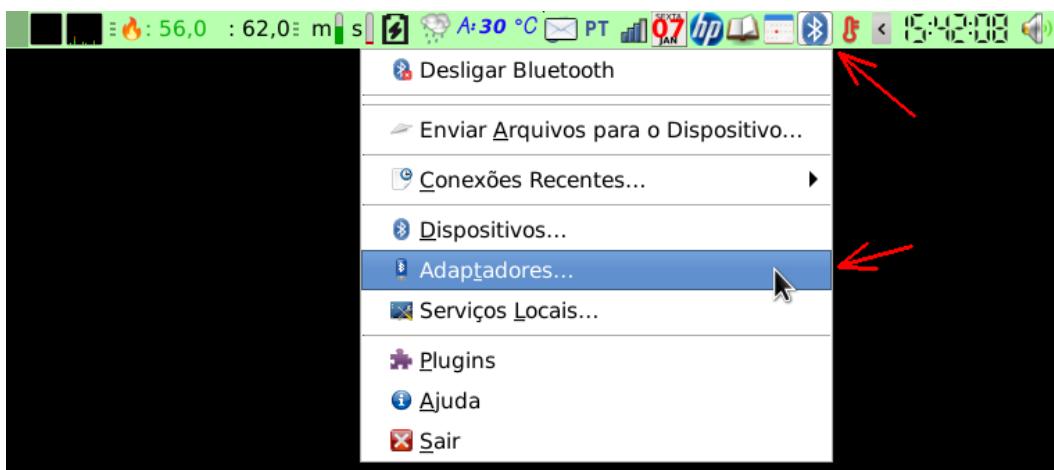


Figura 47.1: Acesso à janela “Adaptadores”.

Na janela “Adaptadores”, deve-se habilitar para o dispositivo ficar visível para conexões Bluetooth, além de se configurar o nome com o qual o dispositivo será encontrado na rede, o “Nome Amigável”, conforme apresentado na Figura 47.2.



Figura 47.2: Ativação da visibilidade e configuração do nome do computador na rede Bluetooth.

47.5 Comunicação do “Serial Bluetooth Terminal” com o servidor programado em Python

Nesta seção é apresentada a transmissão de uma mensagem a partir do aplicativo para celular Android “Serial Bluetooth Terminal” para o servidor Python apresentado na Seção 47.2.

A Figura 47.3 apresenta a tela inicial do “Serial Bluetooth Terminal”, destacando o botão para acesso aos dispositivos.



Figura 47.3: Tela inicial do “Serial Bluetooth Terminal”.

A Figura 47.4 apresenta a tela de acesso aos itens do aplicativo, destacando o botão de acesso aos dispositivos.

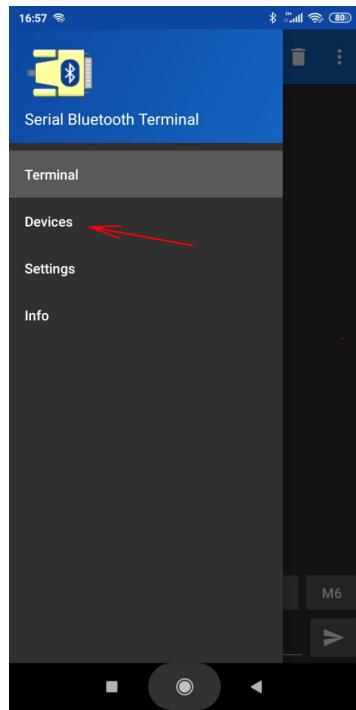


Figura 47.4: Acesso aos itens do “Serial Bluetooth Terminal”.

A Figura 47.5 apresenta a tela de dispositivos, sem nenhuma dispositivo disponível, destacando o botão para acesso à tela de pareamento Bluetooth do Android.

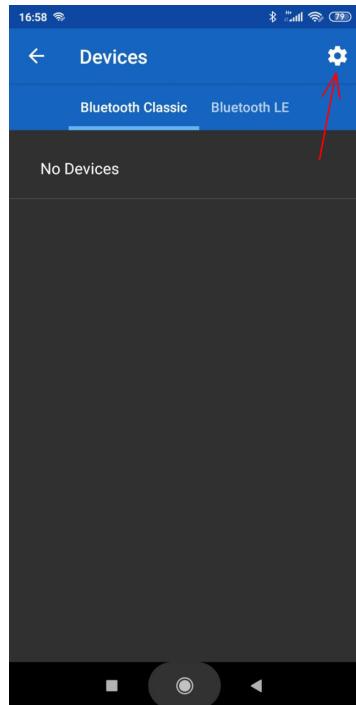


Figura 47.5: Tela de dispositivos do “Serial Bluetooth Terminal”. Em destaque o botão para a tela de pareamento Bluetooth do Android.

A Figura 47.6 apresenta a tela de pareamento Bluetooth do Android, destacando o botão para atualizar a lista de dispositivos visíveis e apresentando o nome do computador na rede Bluetooth (sobre o qual deve-se clicar para realizar o pareamento).

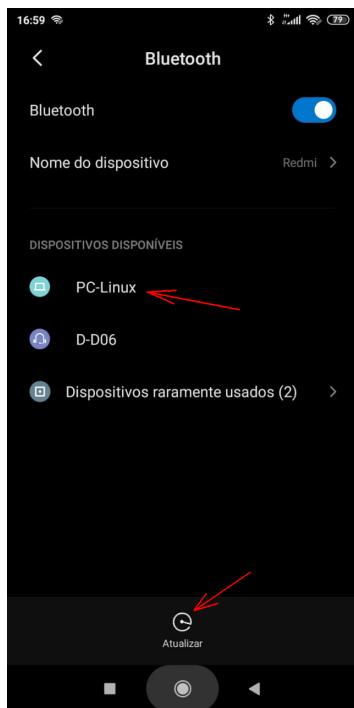


Figura 47.6: Tela de pareamento Bluetooth do Android.

Durante o pareamento podem ser apresentadas telas com opções para confirmação do pareamento. A opção de confirmação do celular é apresentada na Figura 47.7 e a opção de confirmação do computador é apresentada na Figura 47.8.

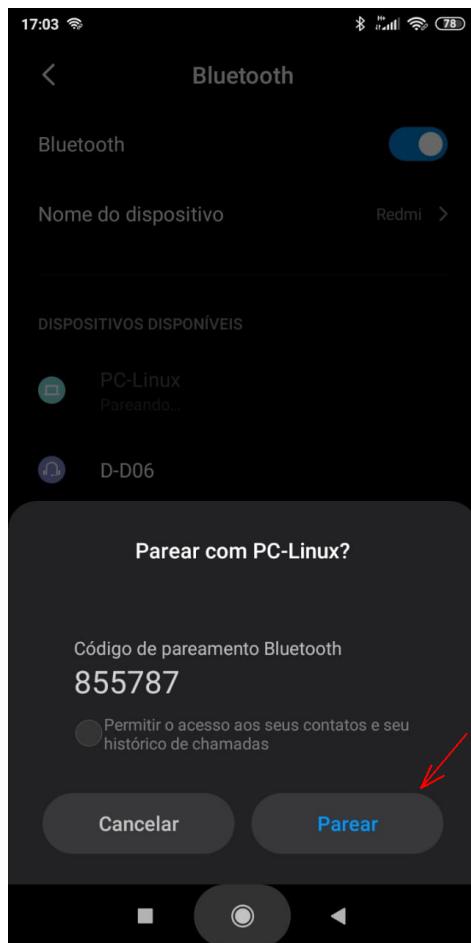


Figura 47.7: Confirmação do pareamento no celular.



Figura 47.8: Janela de confirmação do pareamento no computador (padrão de mensagem do XFCE).

A Figura 47.9 apresenta a tela de dispositivos do Android após o pareamento, destacando o botão para retorno ao “Serial Bluetooth Terminal”.

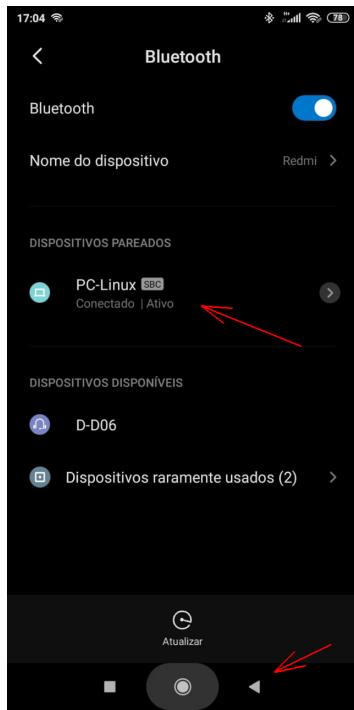


Figura 47.9: Tela de dispositivos do Android após o pareamento.

A Figura 47.10 apresenta a tela de dispositivos do “Serial Bluetooth Terminal” após a conexão, destacando o botão para retorno ao terminal.

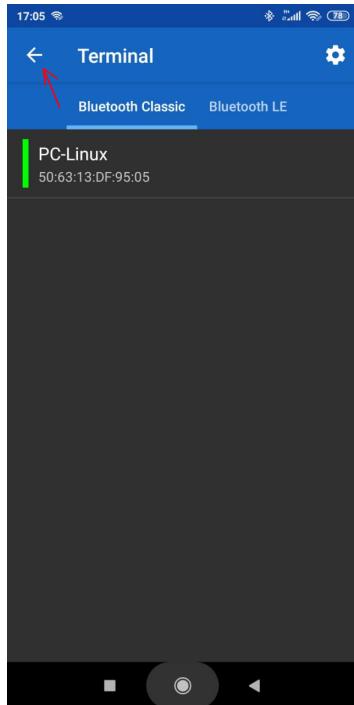


Figura 47.10: Tela de dispositivos do “Serial Bluetooth Terminal” após o pareamento.

Após o pareamento o “Serial Bluetooth Terminal” pode se conectar ao computador através de um socket. Sem o servidor executando no computador, é observado a mensagem de erro da Figura 47.11 na tentativa de conexão (através do botão destacado na figura).



Figura 47.11: Falha de conexão pela inexistência de um socket disponível. Aqui o servidor Bluetooth não está executando no computador.

Nota: Neste teste não foi necessário fazer indicação de porta para conexão por socket (como é feito em conexões TCP).

A Figura 47.12 apresenta a janela do terminal com o servidor da Seção 47.2 iniciado.

A screenshot of a Linux terminal window titled "Terminal". The window has a standard title bar with buttons for "Arquivo", "Editar", "Ver", "Pesquisar", "Terminal", and "Ajuda". The command line shows the user's path: "(base) alex@PC-Linux: ~/Documentos_Backup/IFES/Latex_Projetos_Apostilas/09_ApostilaRedesIndustriais/RedesIndustriais_Bluetooth_RFCOMM_exemplo/projeto_capitulo\$". Below the path, the user has typed the command "python 01_rfcomm-server.py" and is waiting for the output. A red arrow points to the command line where the script name is entered.
(base) alex@PC-Linux: ~/Documentos_Backup/IFES/Latex_Projetos_Apostilas/09_ApostilaRedesIndustriais/RedesIndustriais_Bluetooth_RFCOMM_exemplo/projeto_capitulo\$ python 01_rfcomm-server.py

Figura 47.12: Servidor Bluetooth RFCOMM executando.

A Figura 47.13 apresenta a conexão realizada com sucesso pelo “Serial Bluetooth Terminal” após o início do servidor RFCOMM no computador.



Figura 47.13: Conexão Bluetooth RFCOMM realizada com sucesso.

A Figura 47.14 apresenta uma mensagem pronta para ser enviada do “Serial Bluetooth Terminal” para o servidor executando no computador.

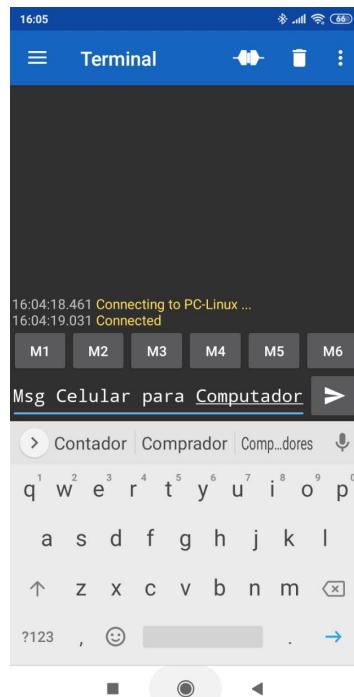


Figura 47.14: Mensagem pronta para ser enviada.

A Figura 47.15 apresenta a confirmação após o envio da mensagem (indicação da mensagem enviada em azul), também apresenta a mensagem enviada pelo servidor programado em Python (mensagem em verde). Por fim a figura apresenta a informação de que a conexão foi perdida, o que é normal, pelo fato das últimas linhas do servidor estarem configuradas para fechar o socket e o servidor.



Figura 47.15: Confirmação da mensagem enviada e aviso de conexão perdida.

A Figura 47.16 apresenta a janela com o terminal executando o servidor Bluetooth RF-COMM, onde é possível ver a mensagem enviada pelo “Serial Bluetooth Terminal” a partir de um telefone celular.

A screenshot of a Linux terminal window titled 'Terminal'. The window has a blue header bar with the title 'Terminal' and some icons. The main area is a dark terminal window showing the following text:
alex@PC-Linux: ~/Documentos_Backup/IFES/Latex_Projetos_Apostilas/09_ApostilaRedesIndustriais/Redes Industriais_Bluetooth_RFCOMM_exemplo/projeto_capitulo\$ python 01_rfcomm-server.py
Accepted connection from ('18:01:F1:46:3A:BE', 1)
received [b'Msg Celular para Computador\r\n']
alex@PC-Linux: ~/Documentos_Backup/IFES/Latex_Projetos_Apostilas/09_ApostilaRedesIndustriais/Redes Industriais_Bluetooth_RFCOMM_exemplo/projeto_capitulo\$

Figura 47.16: Mensagem recebida pelo servidor Bluetooth RFCOMM.

Capítulo 48

Abertura de uma porta serial Bluetooth RFCOMM pelo comando “rfcomm watch”

Este capítulo segue o tutorial disponível em:

<https://scribbles.net/setting-up-bluetooth-serial-port-profile-on-raspberry-pi/>.

A primeira parte deste tutorial, correspondente à configuração que deve ser realizada, está disponível no Capítulo 46.

Atenção

Na atualização de pacotes do Linux é possível que a configuração apresentada no Capítulo 46 seja perdida. Se as passos descritos neste capítulo deixarem de funcionar, verifique se a configuração do arquivo “/etc/systemd/system/dbus-org.bluez.service” não foi desfeita pela atualização de pacotes do Linux.

Essa parte do tutorial corresponde à abertura da porta serial sobre Bluetooth com RF-COMM e a realização da comunicação do “Serial Bluetooth Terminal” com uma ferramenta de comunicação serial executando no computador. No exemplo original a ferramenta de comunicação usada foi o “Minicom”. Nesta capítulo foi usado o HTerm no lugar do Minicom.

48.1 Pareamento

O pareamento pode ser feito conforme apresentado na Figura 47.6.

Caso não se tenha uma ferramenta gráfica no computador para o gerenciamento do pareamento Bluetooth, pode-se utilizar os seguintes passos:

Passo 1: Iniciar o bluetoothctl com o seguinte comando:

```
bluetoothctl
```

Passo 2: Tornar o computador descobrível, com o seguinte comando, executado no terminal do bluetoothctl:

```
discoverable on
```

A Figura 48.1 apresenta o terminal com o iniciado.

```
alex@PC-Linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
(base) alex@PC-Linux: ~$ bluetoothctl
Agent registered
[bluetooth]# discoverable on
Changing discoverable on succeeded
[bluetooth]#
```

Figura 48.1: Terminal com o bluetoothctl iniciado.

A Figura 48.2 apresenta o terminal do bluetoothctl após o pareamento com o telefone celular.

```
alex@PC-Linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
(base) alex@PC-Linux: ~$ bluetoothctl
Agent registered
[bluetooth]# discoverable on
Changing discoverable on succeeded
[CHG] Device 18:01:F1:46:3A:BE Connected: yes
[Redmi]#
```

Figura 48.2: Terminal do bluetoothctl após o pareamento com o telefone celular.

Pressione `Ctrl D` para sair do bluetoothctl

Instruções sobre o bluetoothctl podem ser encontradas no seguinte artigo:

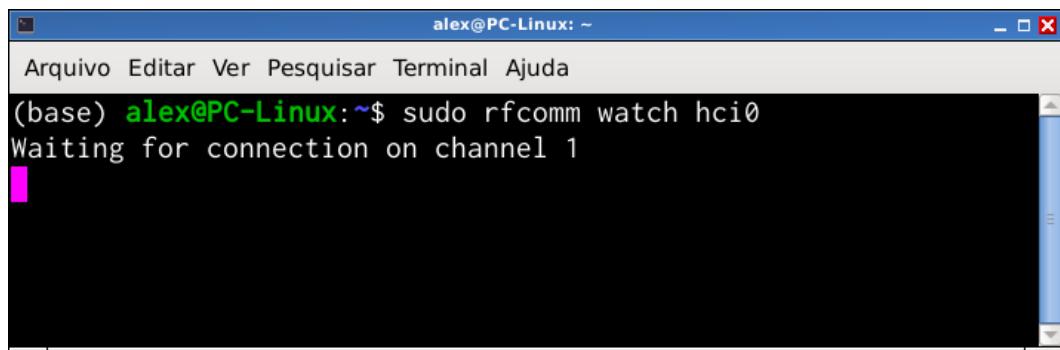
<https://www.linux-magazine.com/Issues/2017/197/Command-Line-bluetoothctl>

48.2 Abertura da porta serial

Para a escuta do computador por um canal RFCOMM é iniciada com o seguinte comando:

```
sudo rfcomm watch hci0
```

Conforme apresentado na Figura 48.3.



```
alex@PC-Linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
(base) alex@PC-Linux:~$ sudo rfcomm watch hci0
Waiting for connection on channel 1
```

Figura 48.3: Escuta por um canal Bluetooth RFCOMM (executando com Debian Linux).

Com o computador na escuta e os dispositivos já pareados, o celular pode realizar a conexão no canal RFCOMM aberto pelo computador, conforme apresentado na Figura 48.4

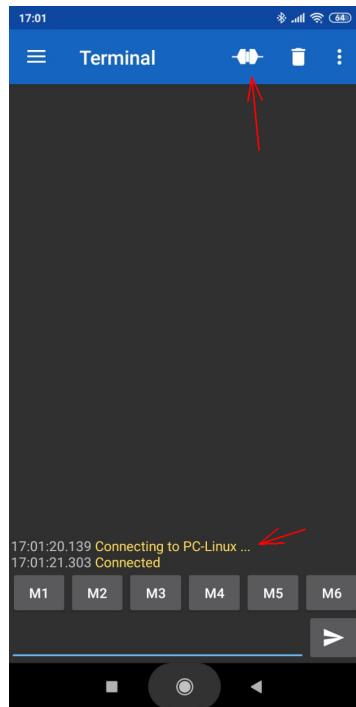


Figura 48.4: Conexão, a partir de um celular, ao canal RFCOMM aberto pelo computador.

Após a conexão o terminal onde foi iniciada a escuta indica a realização da conexão, conforme apresentado na Figura 48.5.

```
alex@PC-Linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
(base) alex@PC-Linux:~$ sudo rfcomm watch hci0
Waiting for connection on channel 1
Connection from 18:01:F1:46:3A:BE to /dev/rfcomm0
Press CTRL-C for hangup
```

A screenshot of a Linux terminal window titled "alex@PC-Linux: ~". The window has a standard window title bar with icons for minimize, maximize, and close. The menu bar includes "Arquivo", "Editar", "Ver", "Pesquisar", "Terminal", and "Ajuda". The main terminal area displays the command "(base) alex@PC-Linux:~\$ sudo rfcomm watch hci0" followed by the output "Waiting for connection on channel 1", "Connection from 18:01:F1:46:3A:BE to /dev/rfcomm0", and "Press CTRL-C for hangup". The terminal uses a dark background with white text. A vertical scroll bar is visible on the right side of the window.

Figura 48.5: Indicação da conexão ao canal RFCOMM aberto pelo computador.

48.3 Comunicação serial

A comunicação desta seção pode ser feita com qualquer ferramenta genérica de comunicação serial, como o Minicon ou o HTerm. Esta seção apresenta a comunicação pelo HTerm.

Foi necessário executar o HTerm com privilégio de root usando “sudo”.

48.3.1 Conexão pelo HTerm

Após a execução do comando `sudo rfcomm watch hci0` e da conexão pelo aplicativo “Serial Bluetooth Terminal”, pode-se realizar a conexão através da seguinte porta:

`/dev/rfcomm0`

No teste realizado, a taxa de transmissão configurada no HTerm não precisou ser ajustada para um valor específico, a comunicação funcionou com todos valores testados.

A Figura 48.6 apresenta o HTerm conectado à porta `/dev/rfcomm0`.

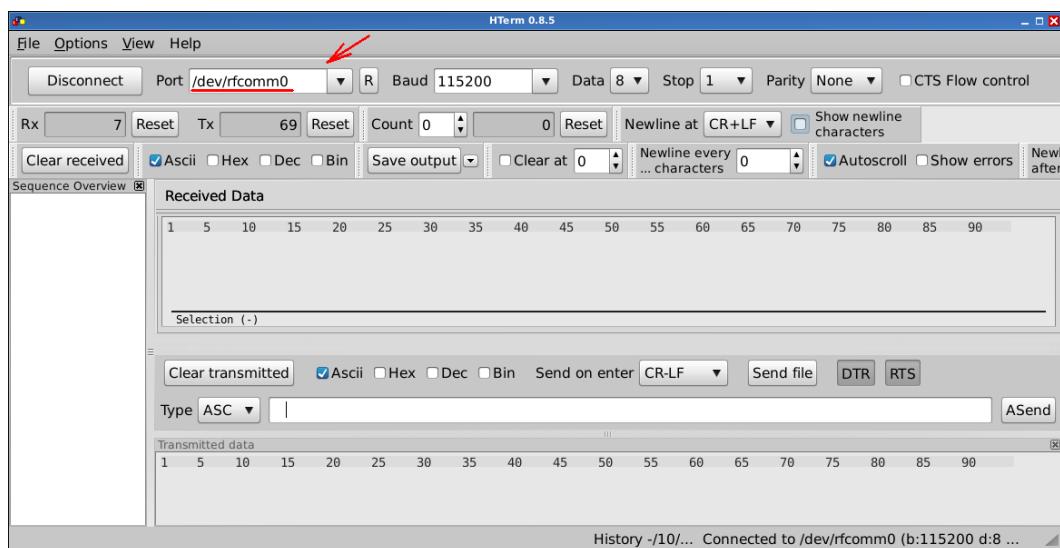


Figura 48.6: HTerm conectado à porta `/dev/rfcomm0`.

Após a conexão o HTerm (computador) pode ser usado para trocar mensagens com o Serial Bluetooth Terminal (celular).

Parte IX

Experimentos e considerações práticas sobre I²C

Capítulo 49

Exemplo de projeto com Arduino e componentes I²C

O padrão de comunicação I²C define um barramento. Portanto, vários componentes podem ser ligados compartilhando as mesmas linhas de dados (SDA) e relógio (SCK ou SCL).

Este capítulo apresenta um projeto que ilustra este potencial, compartilhando o barramento I²C com dois modelos de displays e três módulos de memória.

O requisito para compartilhamento do barramento é que os componentes tenham endereços diferentes. Portanto, o uso de componentes repetidos depende da capacidade de configuração para mudança do endereço. Os módulos de memória do exemplo deste capítulo apresentam três pinos que permitem a configuração endereços distintos, permitindo até 8 módulos. Para os displays usados, deve-se consultar o datasheet para verificação dessa possibilidade (não sei se é possível).

49.1 Circuito simulado

A Figura 49.1 apresenta o circuito simulado.

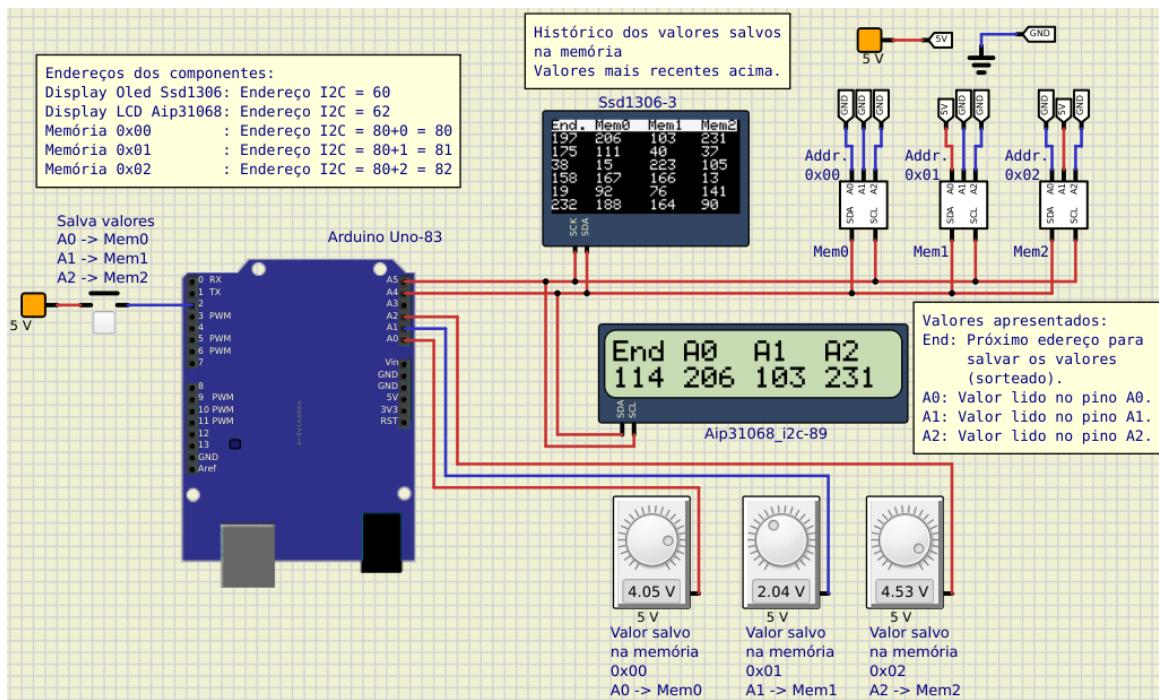


Figura 49.1: Projeto com Arduino e componentes I2C em um único barramento.

49.2 Código executado

```
1 // Alex Brandão Rossow
2
3 #include <Wire.h> // Para uso do I2C
4 #include <Adafruit_GFX.h>
5 #include <Adafruit_SSD1306.h>
6 #include<LiquidCrystal_AIP31068_I2C.h> // Biblioteca do display LCD
7
8
9
10 // #####
11 // Configuração dos pinos #####
12 // #####
13 int salva_Pin = 2;
14 int analogic0_Pin = A0;
15 int analogic1_Pin = A1;
16 int analogic2_Pin = A2;
17 int analogic3_Pin = A3;
18 // #####
19 // #####
20
21
22
23 // #####
24 // Configuração do display I2C OLED Ssd1306 #####
25 // #####
26 #define SCREEN_WIDTH 128 // OLED display width, in pixels
27 #define SCREEN_HEIGHT 64 // OLED display height, in pixels
28
29 // Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
30 // The pins for I2C are defined by the Wire-library.
31 // On an arduino UNO: A4(SDA), A5(SCL)
32 // On an arduino MEGA 2560: 20(SDA), 21(SCL)
33 // On an arduino LEONARDO: 2(SDA), 3(SCL), ...
34 #define OLED_RESET -1 // Reset pin # (or -1 if sharing Arduino reset pin)
35
36 // Endereço I2C do display, ver o datasheet para obter esse valor.
37 // No SimulIDE o endereço padrão é 60(=0x3C).
38 #define SCREEN_ADDRESS 0x3C
39 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
40 // #####
41 // #####
42
43
44
```

```
45 // #####  
46 // Configuração do display I2C LED Aip31068 #####  
47 // #####  
48 // Endereço I2C do display LCD ver o datasheet para obter esse valor.  
49 // No SimulIDE o endereço padrão é 62(=0x3E).  
50 LiquidCrystal_AIP31068_I2C lcd(0x3E,16,2); //End I2C, #colunas, #linhas  
51 // #####  
52 // #####  
53  
54  
55  
56 // #####  
57 // Configuração para acesso à memória I2C #####  
58 // #####  
59 // byte my_data; // dado escrito/lido da memória RAM I2C  
60 // byte my_address; // endereço para escrita/leitura da memória RAM I2C  
61 // #####  
62 // #####  
63  
64  
65  
66 // #####  
67 // Variáveis #####  
68 // #####  
69 byte valorAnalogico0; // recebe o valor lido no pino A0  
70 byte valorAnalogico1; // recebe o valor lido no pino A1  
71 byte valorAnalogico2; // recebe o valor lido no pino A1  
72  
73 byte valorLidoMem0; // valor lido da memória Mem0  
74 byte valorLidoMem1; // valor lido da memória Mem1  
75 byte valorLidoMem2; // valor lido da memória Mem2  
76  
77  
78 byte enderecoSorteado; // Endereço sorteado  
79 const int dimUltimosEnd = 7; // Dimensão do histórico de endereço  
80  
81 // vetor com os últimos endereços sorteados (exibidos no display OLED)  
82 byte ultimosEnderecos[dimUltimosEnd] = {0, 0, 0, 0, 0, 0, 0};  
83  
84 // controle para reduzir o número de escritas no display  
85 int contadorControleDisplay = 0;  
86  
87 // Estado antigo do pino "salva_Pin", para detectar borda (P)  
88 int salva_Pin_Old=0;  
89 // #####  
90 // #####
```

```
91
92
93
94 void setup()
95 {
96     Serial.begin(9600); // Inicia a comunicação serial
97     lcd.init(); // Inicia o display LCD
98     Wire.begin(); // Inicia a comunicação I2C
99
100    // ##### SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally #####
101    // Configuração do display I2C OLED #####
102    // #####
103    // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
104    if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
105        Serial.println(F("SSD1306 allocation failed"));
106        for(;;); // Don't proceed, loop forever
107    }
108    // Show initial display buffer contents on the screen --
109    // the library initializes this with an Adafruit splash screen.
110    display.display();
111    delay(2000); // Pause for 2 seconds
112
113    // Clear the buffer
114    display.clearDisplay();
115
116    // Draw a single pixel in white
117    display.drawPixel(10, 10, SSD1306_WHITE);
118    // #####
119    // #####
120
121    pinMode(salva_Pin, INPUT); // Configura o pino do botão "Salva valores"
122
123    sorteia(); // Executa o primeiro sorteio de endereço
124 }
125
126
127
128 void loop()
129 {
130     Serial.println("Iniciou o loop");
131
132     // Lê os pinos analógicos
133     valorAnalogico0 = map(analogRead(analogic0_Pin), 0, 1023, 0, 255);
134     valorAnalogico1 = map(analogRead(analogic1_Pin), 0, 1023, 0, 255);
135     valorAnalogico2 = map(analogRead(analogic2_Pin), 0, 1023, 0, 255);
136 }
```

```
137 // Reduz a frequência de atualização dos displays para
138 // deixar a simulação mais leve (não é reduzido no delay
139 // do loop para não comprometer a resposta do botão)
140 if (contadorControleDisplay == 30)
141 {
142     escreveDisplayLCD();
143     escreveDisplayOLED();
144     contadorControleDisplay = 0;
145 }
146
147
148
149 // #####
150 // Habilitar essas mensagens somente para debugar
151 // o programa, pois deixam a simulação lenta.
152 //Serial.println("valorAnalogico0 = ");
153 //Serial.println(valorAnalogico0);
154
155 //Serial.println("valorAnalogico1 = ");
156 //Serial.println(valorAnalogico1);
157
158 //Serial.println("valorAnalogico2 = ");
159 //Serial.println(valorAnalogico2);
160
161 //Serial.println("valorAnalogico3 = ");
162 //Serial.println(valorAnalogico3);
163 // #####
164
165
166 if (digitalRead(salva_Pin) && !salva_Pin_Old) // Borda (P)
167 {
168     // Escreve nas memórias.
169     // Parâmetros: Endereço I2C do módulo, Endereço do dado, Valor
170     writeI2CByte(80, ultimosEnderecos[0], valorAnalogico0); //Mem0
171     writeI2CByte(81, ultimosEnderecos[0], valorAnalogico1); //Mem1
172     writeI2CByte(82, ultimosEnderecos[0], valorAnalogico2); //Mem2
173     sorteia();
174 }
175
176 //Estado do botão salvo para detectar a borda de subida (P)
177 salva_Pin_Old = digitalRead(salva_Pin);
178
179 contadorControleDisplay +=1;
180 delay(10);
181 }
182 }
```

```
183
184
185 // ######
186 // Sortei um endereço de memória para salvar os valores. #####
187 // O mesmo endereço sorteado é usado nos três módulos de #####
188 // memória I2C. #####
189 void sorteia()
190 {
191     enderecoSorteado = random(0,255);
192
193     // Desloca o vetor com os últimos endereços sorteados
194     for(int i=dimUltimosEnd-1; i>=0; i--){
195         ultimosEnderecos[i+1]=ultimosEnderecos[i];
196     }
197     ultimosEnderecos[0] = enderecoSorteado;
198 }
199 // #####
200
201
202 // #####
203 // Escreve no display OLED Ssd1306 #####
204 void escreveDisplayOLED(void) {
205     display.clearDisplay();
206
207     display.setTextSize(1); // Normal 1:1 pixel scale
208
209     display.setTextColor(SSD1306_BLACK, SSD1306_WHITE); // Draw 'inverse' text
210     display.setCursor(0, 0); // Inicia no canto superio esquerdo (coluna, linha)
211     display.cp437(true); // Use full 256 char 'Code Page 437' font
212
213     display.write("End. Mem0 Mem1 Mem2\n");
214     display.write("\n");
215
216     display.setTextColor(SSD1306_WHITE); // Draw white text
217
218     // Varre o histórico de valores salvos nas
219     // memórias para exibição
220     for (int i=1; i<dimUltimosEnd; i++)
221     {
222         valorLidoMem0 = readI2CByte(80, ultimosEnderecos[i]); // Lê Mem0
223         valorLidoMem1 = readI2CByte(81, ultimosEnderecos[i]); // Lê Mem1
224         valorLidoMem2 = readI2CByte(82, ultimosEnderecos[i]); // Lê Mem2
225
226         display.setCursor(0, 9*i); //posiciona o cursor (coluna, linha)
227         display.print(ultimosEnderecos[i]);
228 }
```

```
229     display.setCursor(30, 9*i);
230     display.print(valorLidoMem0);
231
232     display.setCursor(67, 9*i);
233     display.print(valorLidoMem1);
234
235     display.setCursor(102, 9*i);
236     display.print(valorLidoMem2);
237 }
238
239 display.display();
240 }
241 // #####
242
243
244 // #####
245 // Escreve no display LCD Aip31068 #####
246 void escreveDisplayLCD(void) {
247     lcd.clear();
248
249     lcd.setCursor(0,0); // Posiciona o cursor na coluna 0, linha 0
250     lcd.print("End");
251
252     lcd.setCursor(4,0); // Posiciona o cursor na coluna 4, linha 0
253     lcd.print("A0");
254
255     lcd.setCursor(8,0); // Posiciona o cursor na coluna 8, linha 0
256     lcd.print("A1");
257
258     lcd.setCursor(12,0); // Posiciona o cursor na coluna 12, linha 0
259     lcd.print("A2");
260
261
262     lcd.setCursor(0,1);
263     lcd.print(ultimosEnderecos[0]); // Posiciona o cursor na coluna 0, linha 1
264
265     lcd.setCursor(4,1);
266     lcd.print(valorAnalogico0); // Posiciona o cursor na coluna 4, linha 1
267
268     lcd.setCursor(8,1);
269     lcd.print(valorAnalogico1); // Posiciona o cursor na coluna 8, linha 1
270
271     lcd.setCursor(12,1);
272     lcd.print(valorAnalogico2); // Posiciona o cursor na coluna 12, linha 1
273 }
274 // #####
```

```
275
276
277 // ##### Escreve um byte em um dos módulos de memória #####
278 // Escreve um byte em um dos módulos de memória #####
279 // Parâmetros:
280 // ADDR : Endereço I2C do módulo
281 // data_addr: Endereço do byte ser usado no módulo
282 // data : Valor a ser escrito
283 void writeI2CByte(byte ADDR, byte data_addr, byte data){
284     Wire.beginTransmission(ADDR);
285     Wire.write(data_addr);
286     Wire.write(data);
287     Wire.endTransmission();
288 }
289 // #####
290
291
292 // ##### Lê um byte em um dos módulos de memória #####
293 // Lê um byte em um dos módulos de memória #####
294 byte readI2CByte(byte ADDR, byte data_addr){
295     byte data = NULL;
296     Wire.beginTransmission(ADDR);
297     Wire.write(data_addr);
298     Wire.endTransmission();
299     Wire.requestFrom(ADDR, 1); //retrieve 1 returned byte
300     delay(1);
301     if(Wire.available())
302     {
303         data = Wire.read();
304     }
305     return data;
306 }
307 // #####
```

Parte X

Exercícios

Capítulo 50

Exercícios

Exercícios referentes ao conteúdo da apostila.

50.1 Exercícios sobre integração dos níveis de uma organização

1. Qual dos níveis da estrutura da piramide de automação apresenta o menor volume de dados transmitido por equipamento (considerando um mesmo intervalo de tempo)?
2. Qual dos níveis da estrutura da piramide de automação apresenta o maior volume de dados transmitido por equipamento (considerando um mesmo intervalo de tempo)?
3. Qual dos níveis da estrutura da piramide de automação apresenta menor taxa de transmissão?
4. Qual dos níveis da estrutura da piramide de automação apresenta maior taxa de transmissão?
5. Qual dos níveis da estrutura da piramide de automação apresenta requisitos de menor tempo de resposta (menor tempo de transmissão)?
6. Cite três protocolos que atuam na interligação de componentes do nível dos dispositivos de campo (e.g. sensores e atuadores) com componentes no nível de controle (e.g. CLP e SDCD).
7. Cite três protocolos adequados para levar informações até o nível gerencial.

50.2 Exercícios sobre Histórico das Redes industriais

1. Considerando as rede Modbus, Profibus e Foundation Fieldbus pesquise o ano de lançamento, a empresa ou grupo responsável pela criação e a participação atual desses padrões na indústria (percentual de instrumentos já instalados e novos instrumentos vendidos).

50.3 Exercícios sobre Alcance das Redes

1. Qual o alcance típico de uma LAN?
2. Cite um protocolo empregado na implementação de um LAN.
3. Qual o alcance típico de uma MAN?
4. Cite um protocolo empregado na implementação de um MAN.
5. Qual o alcance típico de uma WAN?

50.4 Exercícios sobre Topologia de Redes

1. Qual topologia de rede se caracterizada por conectar apenas dois dispositivos?
2. Qual a topologia de rede, normalmente usada no nível de sensores, é implementada pela ligação dos dispositivos em cadeia ou pela derivação de um cabo principal por meio de uma caixa de junção?
3. Qual a topologia de rede apresenta como vantagem a presença de um caminho redundante para transmissão dos dados?

50.5 Exercícios sobre Sistemas de Comunicação

1. O que caracteriza um sistema de comunicação simplex?
2. O que caracteriza um sistema de comunicação half-duplex?
3. O que caracteriza um sistema de comunicação full-duplex?
4. O que caracteriza um sistema de comunicação assíncrono?

5. O que caracteriza um sistema de comunicação síncrono?
6. Indique quais padrões de comunicação listados a seguir são síncronos e quais são assíncronos: RS-232, I2C, CAN, RS-485, SPI.
7. Indique quais padrões de comunicação listados a seguir apresentam capacidade de comunicação half-duplex e quais apresentam capacidade full-duplex: RS-232, I2C, CAN, RS-485 a dois fios, RS-485 a quatro fios, SPI.
8. Indique quais padrões de comunicação listados a seguir permitem apenas a comunicação ponto a ponto (dois dispositivos apenas) e quais permitem a conexão de vários dispositivos: RS-232, I2C, CAN, RS-485, SPI.
9. Indique quais padrões de comunicação listados a seguir empregam dois condutores dedicados para o canal de dados (definido como canal balanceado ou diferencial): RS-232, I2C, CAN, RS-485, SPI.
10. Para o padrão RS-485, a transmissão a uma distância de 300m pode ser realizada com qual velocidade máxima aproximada (velocidade definida pelo padrão RS-485)?

50.6 Exercícios sobre os modelos ISO/OSI e TCP/IP

1. Em qual camada do modelo OSI é definida a forma de codificação em bits (através do sinal elétrico ou de radiofrequência)?
2. Qual a camada do modelo OSI responsável por gerenciar o acesso ao meio de transmissão em uma rede?
3. Qual alternativa apresenta a camada do modelo OSI responsável pela comunicação fim a fim (entre programas ou dispositivos na origem e no destino).
4. Qual protocolo da camada de transporte utilizado na internet garante uma comunicação livre de erros (solicita o reenvio de pacotes perdidos ou corrompidos)?
5. Qual protocolo da camada de transporte utilizado na internet tem foco na velocidade, sendo adequado para serviços como VoIP?
6. O protocolo TCP é orientado a conexão?
7. O protocolo UDP é orientado a conexão?
8. Qual protocolo de camada física utilizado pelo ZigBee?

9. Qual protocolo de camada física utilizado pelo Profibus PA?
10. Qual protocolo de camada física utilizado pelo DeviceNet?
11. Qual protocolo de camada física utilizado pelo Profibus DP?
12. Qual protocolo de camada física utilizado pelo Foundation Fieldbus H1?
13. Cite três protocolos que empregam o padrão IEEE802.15.4 como camada física.
14. Cite dois protocolos que empregam o padrão IEC61158-2 como camada física.
15. HTTP, FTP, IMAP e SMTP são exemplos de protocolos que atuam em qual camada do modelo TCP/IP?
16. Qual elemento de endereçamento do protocolo TCP?
17. O protocolo HART permite a ligação de vários dispositivos em um mesmo cabeamento? Em caso afirmativo, qual o node dessa configuração?
18. O protocolo ASi permite a transmissão de valores analógico? Em caso afirmativo, quantas mensagens são necessárias para transmissão de um valor analógico de 16 bits?

50.7 Exercícios sobre dispositivos de Redes

1. Quais dispositivos de rede atuam exclusivamente na camada física do modelo OSI?
2. Quais dispositivos de rede atuam na camada de enlace de dados do modelo OSI?
3. Qual dispositivo a ser utilizado para possibilitar a comunicação de um dispositivo Profibus com outros dispositivos DeviceNet?
4. Os roteadores atuam em qual camada do modelo OSI?
5. Qual o protocolo de camada de rede empregado na internet?

50.8 Exercícios sobre meios de comunicação

1. Para uma aplicação Ethernet industrial a combinação de um componente categoria “CAT6A” com um componente categoria “CAT7” gera uma caminho de qual classe?

2. Qual a tecnologia de transmissão mais adequada para a construção de um canal de comunicação para transmissão de uma grande quantidade de dados (da ordem de Gbps) a uma distância de aproximadamente 5km em um ambiente com grande incidência de ruído eletromagnético?
3. Qual padrão de wireless voltado para área industrial permite a transmissão sem limitar a camada de aplicação a um único protocolo?
4. Qual a diferença entre fibra ótica multimodo e monomodo?
5. Qual a função dos resistores de terminação em cabeamento metálico?

50.9 Exercícios sobre Redes de Campo Seriais

1. Qual o protocolo criado com o objetivo de permitir a transmissão de informações digitais sobre o cabeamento de dados analógico existente?
2. Qual o padrão de rede industrial foi criado com enfoque na ligação de dispositivos discretos?
3. Qual padrão de rede foi criado com enfoque na indústria automobilística, e ainda hoje é muito usado nesta área?
4. O padrão DeviceNet utiliza qual padrão de rede nas camadas física e de enlace de dados.
5. Cite duas redes que utilizam padrões de camada física que permitem sua utilização em áreas potencialmente explosivas?
6. Qual padrão de rede possui uma especificação de cabo com formato achatado e que permite a fácil ligação e remoção de dispositivos por meio de conectores vâmpiro?
7. Qual a extensão dos arquivos fornecidos pelos fabricantes de dispositivos PROFIBUS que permite a configuração de equipamentos, como CLPs, para que possam acessar os instrumentos?
8. Qual a extensão dos arquivos fornecidos pelos fabricantes de para o acesso e configuração de seus instrumentos através de ferramentas como o PactWare?
9. Qual padrão de rede industrial adota um padrão de comunicação peer-to-peer com um dispositivo especial chamado LAS (Link Active Scheduler) gerenciando o acesso ao meio de transmissão.

10. RTU, ASCII e TCP são variações de qual protocolo de rede industrial?
11. Dos padrões listados a seguir, qual não adota o padrão de comunicação mestre/escravo? Modbus, Profibus, HART, CAN, ASi.

Apêndice A

Manual do sensor de temperatura e umidade SHT20

Esse apêndice apresenta o manual do sensor de temperatura e umidade SHT20.

Temperature and humidity transmitter SHT20 sensor Modbus RS485

Product Description:

Product adopts industrial-grade chip, high-precision SHT20 temperature and humidity sensors, ensure the products with good reliability, high precision and interchangeability. Adopt RS485 hardware interface (with the lightning protection design), the protocol layer compatible with standard industrial Modbus Rtu protocol.

This product integrating MODBUS protocol with ordinary, users can choose communication protocols, common agreement with automatic upload function(**Connect the RS485 serial interface mode tool by automatically output temperature and humidity**).

Product Highlights:

Industrial products, high progress SHT20 temperature and humidity sensor, the RS485 communication;

Standard MODBUS protocol with ordinary at an organic whole, the user can choose communication protocol;

Baud rate can decide for themselves;

General agreement with automatic upload function, upload speed can decide for themselves.

Product Parameters:

Work voltage: DC4-30 v (highest do not exceed 33 v) .

Most powerful: 0.2 W .

Work environment: Temperature 20 °C - 60 °C, Humidity 0-100.

Control precision: Temperature \pm 0.3°C, Humidity \pm 3%RH.

Output interface: RS485 communication (standard MODBUS protocol and custom ordinary), see note agreement device.

Device address: 1-247 can be set, the default is 1.

Baud rate: 9600(the user can set), 8bits, one stop, no check;

Shape size:60*30*18(mm)

MODBUS PROTOCOL**Modbus Function Code:**

0x03:Read keep register

0x04: Read input register

0x06: Write a single keep register

0x10: Write more keep registers

Register Type	Register Address	Register contents	Number of bytes
Input Register	0x0001	Temperature	2
	0x0002	Humidity	2
Keep Register	0x0101	Device Address	2
	0x0102	Baud Rate 0:9600 1:14400 2:19200	2
	0x0103	Temperature correction(/10) -10.0~10.0	2
	0x0104	Humidity correction(/10) -10.0~10.0	2

Modbus Frame format:

Master send format:

Device Address	Function Code	Starting Address Hi	Starting Address Li	Quantity Hi	Quantity Li	CRC Hi	CRC Li

The response format of slave:

MODBUS COMMAND

Master **read temperature** command frame (0x04):

0x01 0x04 0x00 0x01 0x00 0x01 0x60 0x0A (mensagem Hex no RealTerm)

Device Address	Function Code	Starting Address Hi	Starting Address Li	Quantity Hi	Quantity Li	CRC Hi	CRC Li
0x01	0x04	0x00	0x01	0x00	0x01	0x60	0x0a

The **response data from slave:**

Device Address	Function Code	Num of Bytes	Temp Hi	Temp Li	CRC Hi	CRC Li
0x01	0x04	0x02	0x01	0x31	0x79	0x74

Temperature value=0x131, converted to a decimal 305, the actual temperature value = $305 / 10 = 30.5^{\circ}\text{C}$

Note: the temperature is signed hexadecimal number, temperature value = 0xFF33, converted to a decimal - 205, the actual temperature = -20.5°C ;

Master **read humidity** command frame(0x04)

0x01 0x04 0x00 0x02 0x00 0x01 0x90 0x0A (mensagem Hex no RealTerm)

Device Address	Function Code	Starting Address Hi	Starting Address Li	Quantity Hi	Quantity Li	CRC Hi	CRC Li
0x01	0x04	0x00	0x02	0x00	0x01	0xc1	0xca

0x90 0A

The **response data from slave:**

Device Address	Function Code	Num of Bytes	Humi Hi	Humi Li	CRC Hi	CRC Li
0x01	0x04	0x02	0x02	0x22	0xd1	0xba

Humidity value = 0x222, converted to a decimal 546, actual humidity value = 546/10 = 54.6 %;

Continuous read temperature and humidity command frame

(0x04):

Device Address	Function Code	Starting Address Hi	Starting Address Li	Quantity Hi	Quantity Li	CRC Hi	CRC Li
0x01	0x04	0x00	0x01	0x00	0x02	0x20	0x0b

The response data from slave:

Device Address	Function Code	Num of Bytes	Temp Hi	Temp Li	Humi Hi	Humi Li	CRC Hi	CRC Li
0x01	0x04	0x04	0x01	0x31	0x02	0x22	0x2a	0xce

Read keep register(0x03):

Read device address from the slave :

Device Address	Function Code	Starting Address Hi	Starting Address Li	Quantity Hi	Quantity Li	CRC Hi	CRC Li
0x01	0x03	0x01	0x01	0x00	0x01	0xd4	0x0f

The response data from slave:

Device Address	Function Code	Num of Bytes	Slave Add Hi	Slave Add Li	CRC Hi	CRC Li
0x01	0x03	0x02	0x01	0x02	0x30	0x18

Modify the contents of the registers (0x06):

Modify the slave address register:

Device Address	Function Code	Register Address Hi	Register Address Li	Value Hi	Value Li	CRC Hi	CRC Li
0x01	0x06	0x01	0x01	0x00	0x08	0xd4	0x0f

Modify the slave address: 0x08 = 8

The response data from slave(**And send the same**):

Device Address	Function Code	Register Address Hi	Register Address Li	Value Hi	Value Li	CRC Hi	CRC Li
0x01	0x06	0x01	0x01	0x00	0x08	0xd4	0x0f

Continuously change keep registers (0x10):

Device Address	Function Code	Start Address Hi	Start Address Li	Quantity Hi	Quantity Li	Num of Bytes	Reg1 Hi	Reg1 Li	Reg2 Hi	Reg2 Li	CRC Hi	CRC Li
0x01	0x10	0x01	0x01	0x00	0x02	0x04	0x00	0x20	0x25	0x80	0x25	0x09

Slave address : 0x20 = 32

Baud rate : 0x2580 = 9600

The response data from slave:

Device Address	Function Code	Start Address Hi	Start Address Li	Reg Num Hi	Reg Num Li	CRC Hi	CRC Li
0x01	0x10	0x01	0x01	0x00	0x02	0x11	0xf4

General Protocol

The default baud rates 9600 (the user can set), 8 bits of data, one stop, no check

RS485

CMD	instructions
READ	Report triggered a temperature and humidity <i>(27.4°C,67.7% 温度 27.4°C 湿度 67.7%)</i>
AUTO	Start the temperature and humidity automatically report function <i>(Same as above)</i>
STOP	Stop the temperature and humidity automatically report function
BR:XXXX	Set the baud rate 9600~19200 <i>(BR:9600)</i>
TC:XX.X	Set the temperature calibration (-10.0~10.0) <i>(TC:02.0 温度修正值为 2.0°C)</i>
HC:XX.X	Set the humidity ration (-10.0~10.0) <i>(HC:-05.1 湿度修正值为 -5.1%)</i>
HZ:XXX	Set the temperature and humidity reporting rate (0.5,1,2,5,10) <i>(HZ:2 reporting rate 2Hz)</i>
PARAM	Read the system current Settings

PARAM CMD:

TC:0.0,HC:0.0,BR:9600,HZ:1 ->Temp calibration 0.0, Humi calibration0.0, Baud rate 9600,report rate 1Hz

SLAVE_ADD:1 ->MODBU Slave address 1

Bibliografia

- Advantech (2019). *Advantech 2019 ICG Industrial Communication Brochure*. Disponível em: <https://advcloudfiles.advantech.com/ecatalog/2018/12211319.pdf>. Acesso em 15 de novembro de 2020.
- AfterAcademy (2020). *What are Routers, Hubs, Switches, Bridges?* Disponível em: <https://afteracademy.com/blog/what-are-routers-hubs-switches-bridges>. Acesso em 13 de dezembro de 2020.
- Analog Devices (2020). *RS485 Quick Guide*. Disponível em: <https://www.analog.com/media/en/technical-documentation/product-selector-card/rs485fe.pdf>. Acesso em 24 de novembro de 2020.
- Anybus (2020a). *Anybus Gateway Selector*. Disponível em: <https://www.anybus.com/products/gateway-index>. Acesso em 16 de dezembro de 2020.
- (2020b). *Anybus X-gateway ? DeviceNet Scanner - PROFIBUS Slave*. Disponível em: <https://www.anybus.com/products/gateway-index/anybus-xgateway/detail/anybus-x-gateway-devicenet-scanner---profibus-slave>. Acesso em 16 de dezembro de 2020.
- Arduino® (2021a). *Arduino RS485*. Disponível em: <https://www.arduino.cc/en/Reference/ArduinoRS485>. Acesso em 17 de outubro de 2021.
- (2021b). *Serial Communication*. Disponível em: <https://www.arduino.cc/reference/en/language/functions/communication/serial/>. Acesso em 17 de outubro de 2021.
- (2021c). *SoftwareSerial Library*. Disponível em: <https://www.arduino.cc/en/Reference/softwareSerial>. Acesso em 17 de outubro de 2021.
- AutomationForum.Co (2018). *How to calibrate Fieldbus Transmitters?* Disponível em: <https://automationforum.co/calibrate-fieldbus-transmitters/>. Acesso em 10 de novembro de 2020.
- AutomationWorld (2013). *Industrial Networks: Wired and Wireless*. Disponível em: <https://isa100wci.org/en-US/Documents/Presentations/WirelessEBookOpt-from-Automation-World-Nov-2013.aspx>. Acesso em 02 de fevereiro de 2021.

- B+B SmartWorx (2010). *RS-422 AND RS-485 APPLICATIONS EBOOK - A Practical Guide to Using RS-422 and RS-485 Serial Interfaces*. Disponível em: <https://advantech-bb.com/wp-content/uploads/2014/12/RS-422-RS-485-eBook.pdf>. Acesso em 24 de novembro de 2020.
- Coap Technology (2021). *CoAP - RFC 7252 Constrained Application Protocol*. Disponível em: <https://coap.technology/>. Acesso em 24 de dezembro de 2021.
- Computer Hope (2017). *ARPANET*. Disponível em: <https://www.computerhope.com/jargon/a/arpnet.htm>. Acesso em 18 de novembro de 2020.
- (2019). *Computer networking history*. Disponível em: <https://www.computerhope.com/history/network.htm>. Acesso em 18 de novembro de 2020.
- Contemporary Controls (2020a). *ARCNET Tutorial*. Disponível em: <https://www.ccontrols.com/pdf/Tutorial.pdf>. Acesso em 14 de novembro de 2020.
- (2020b). *CAN Tutorial*. Disponível em: <https://www.ccontrols.com/pdf/CANTutorial.pdf>. Acesso em 25 de novembro de 2020.
- Dhaker, Piyu (2018). *Analog Devices - Introduction to SPI Interface*. Disponível em: <https://www.analog.com/media/en/analog-dialogue/volume-52/number-3/introduction-to-spi-interface.pdf>. Acesso em 24 de novembro de 2020.
- Eclipse Mosquitto™ (2021). *Eclipse Mosquitto: An open source MQTT broker*. Disponível em: <https://mosquitto.org/>. Acesso em 21 de abril de 2021.
- Eletronicnotes (2020). *Ethernet Cable: Types, Performance and Pinout - Cat 5, 5e, 6, 6a, 7, 8*. Disponível em: <https://www.electronics-notes.com/articles/connectivity/ethernet-ieee-802-3/cables-types-pinout-cat-5-5e-6.php>. Acesso em 31 de janeiro de 2021.
- Endress+Hauser (2020). *Foundation Fieldbus Overview*. Disponível em: <https://valveproducts.neles.com/documents/softwarepackages/Fieldbus/BA013e01.pdf>. Acesso em 18 de novembro de 2020.
- Fluke Networks (2020). *A Closer Look at Industrial Ethernet Cables*. Disponível em: <https://www.flukenetworks.com/blog/cabling-chronicles/closer-look-industrial-ethernet-cables>. Acesso em 31 de janeiro de 2021.
- Fossbytes (2020). *What Is VPN (Virtual Private Network)? How Does It Work?* Disponível em: <https://fossbytes.com/vpn-virtual-private-network-works/>. Acesso em 16 de novembro de 2020.
- Fromm (2020). *Considerations When Applying TIA-1005-A to Network Infrastructure*. Disponível em: [https://www.frommelectric.com/ASSETS/DOCUMENTS/CMS/EN/Presentation_Archive/Considerations%20When%20Applying%20TIA1005A%20\[IFMA%202020\].pdf](https://www.frommelectric.com/ASSETS/DOCUMENTS/CMS/EN/Presentation_Archive/Considerations%20When%20Applying%20TIA1005A%20[IFMA%202020].pdf). Acesso em 01 de fevereiro de 2021.
- General Cable - A Brand of Prysmian Group (2021a). *Industrial Communication Protocols Cable*. Disponível em: <https://www.generalcable.com/na/us-can/products->

- solutions/industrial/industrial-automation-cable/industrial-communication-protocols-cable. Acesso em 31 de janeiro de 2021.
- General Cable - A Brand of Prysmian Group (2021b). *Multi-Paired, Shielded/Individually Shielded*. Disponível em: <http://general-cable.dcatalog.com/v/Industrial-Automation-Catalog/?page=41>. Acesso em 31 de janeiro de 2021.
- Helmholz (2016). *The basics of the PROFINET topology*. Disponível em: <https://www.helmholz-benelux.eu/nieuwsbericht/the-basics-of-the-profinet-topology/>. Acesso em 15 de novembro de 2020.
- HIVEMQ (2020). *Getting Started with MQTT*. Disponível em: <https://www.hivemq.com/blog/how-to-get-started-with-mqtt/>. Acesso em 21 de abril de 2021.
- IIMc Hyderabad (2020). *Network Devices (Hub, Repeater, Bridge, Switch, Router and Gateways)*. Disponível em: <http://www.iimchderabad.com/Material/Networking%20and%20Internetworking%20Devices.pdf>. Acesso em 13 de dezembro de 2020.
- Innodisk (2017). *J1939-standard CANbus Solutions*. Disponível em: https://www.innodisk.com/epaper/eDM/US_Whitepaper_Download_J1939_CANbus.html. Acesso em 25 de novembro de 2020.
- Inst Tools (2020a). *4-20mA Junction Box versus Fieldbus (FF) Junction Box*. Disponível em: <https://instrumentationtools.com/how-will-a-field-jb-of-conventional-4-20ma-connection-look-different-from-that-of-ff-jb/>. Acesso em 10 de novembro de 2020.
- (2020b). *Fieldbus vs 4-20mA*. Disponível em: <https://instrumentationtools.com/fieldbus-vs-4-20ma/>. Acesso em 10 de novembro de 2020.
- ISA100 Wireless Compliance Institute (2020). *The Technology Behind the ISA100.11a Standard ? An Exploration*. Disponível em: http://isa100wci.org/Documents/PDF/The-Technology-Behind-ISA100-11a-v-3_pptx.aspx. Acesso em 02 de fevereiro de 2021.
- KenCorner (2018). *Network Devices*. Disponível em: <https://kencorner.com/network-devices/>. Acesso em 13 de dezembro de 2020.
- Kunbus Industrial Communication (2002). *Highway Addressable Remote Transducer*. Disponível em: <https://www.kunbus.com/highway-addressable-remote-transducer.html>. Acesso em 18 de novembro de 2020.
- (2020a). *DeviceNet Basics*. Disponível em: <https://www.kunbus.com/devicenet-basics.html>. Acesso em 18 de novembro de 2020.
- (2020b). *Fieldbus Basics*. Disponível em: <https://www.kunbus.com/fieldbus-basics.html>. Acesso em 10 de novembro de 2020.
- Leoni Group (2020). *Cables for data transmission in industrial automation*. Disponível em: https://publications.leoni.com/fileadmin/process_industry/publications/

- catalogues/cables_for_data_transmission.pdf?1501574588. Acesso em 31 de janeiro de 2021.
- Link Labs (2015). *Bluetooth Vs. Bluetooth Low Energy: What's The Difference?* Disponível em: <https://www.link-labs.com/blog/bluetooth-vs-bluetooth-low-energy>. Acesso em 02 de fevereiro de 2021.
- MAILCHI (2021). *The Internet of Things: Connecting Tomorrow*. Disponível em: <https://mailchi.mp/ssstc.com/iot>. Acesso em 21 de abril de 2020.
- MathWorks (2021). *Bluetooth Protocol Stack*. Disponível em: <https://www.mathworks.com/help/comm/ug/bluetooth-protocol-stack.html>. Acesso em 08 de janeiro de 2022.
- Medium (2020). *MQTT Beginners Guide - The IoT protocol explained with Python*. Disponível em: <https://medium.com/python-point/mqtt-basics-with-python-examples-7c758e605d4>. Acesso em 24 de abril de 2021.
- Microchip (2002). *A CAN Physical Layer Discussion*. Disponível em: <http://ww1.microchip.com/downloads/en/appnotes/00228a.pdf>. Acesso em 25 de novembro de 2020.
- Mikroe (2016). *UART - Serial communication*. Disponível em: <https://www.mikroe.com/blog/uart-serial-communication>. Acesso em 17 de outubro de 2020.
- MQTT.ORG (2021). *MQTT: The Standard for IoT Messaging*. Disponível em: <https://mqtt.org/>. Acesso em 21 de abril de 2021.
- Nabto (2021). *Websocket vs. MQTT vs. CoAP: Which is the Best Protocol?* Disponível em: <https://www.nabto.com/websocket-vs-mqtt-vs-coap/>. Acesso em 24 de dezembro de 2021.
- National Instruments (2014). *FOUNDATION™ Fieldbus Overview*. Disponível em: <https://www.ni.com/pdf/manuals/370729d.pdf>. Acesso em 19 de março de 2021.
- Newbedev (2021). *Difference between UART and RS-232?* Disponível em: <https://newbedev.com/difference-between-uart-and-rs-232>. Acesso em 17 de outubro de 2021.
- Nexans (2005). *Industrial Cabling - Technical Paper*. Disponível em: <https://www.nexans.com/Belgium/files/TPIIndustrialcablingv2.pdf>. Acesso em 01 de fevereiro de 2021.
- NI (2019). *O protocolo Modbus em detalhes*. Disponível em: <https://www.ni.com/pt-br/innovations/white-papers/14/the-modbus-protocol-in-depth.html>. Acesso em 18 de novembro de 2020.
- Nixon, Mark (2012). *Emerson Process Management - A Comparison of WirelessHART and ISA100.11a*. Disponível em: <https://www.emerson.com/documents/automation/white-paper-a-comparison-of-wirelesshart-isa100-11a-en-42598.pdf>. Acesso em 02 de fevereiro de 2020.

- NXP Semiconductors (2009). *MPC5121e Serial Peripheral Interface (SPI)*. Disponível em: <https://www.nxp.com/docs/en/application-note/AN3904.pdf>. Acesso em 24 de novembro de 2020.
- (2014). *I2C-bus specification and user manual - Rev.6*. Disponível em: <https://www.nxp.com/docs/en/user-guide/UM10204.pdf>. Acesso em 23 de novembro de 2020.
- OPC FOUDATION (2021). *What is OPC?* Disponível em: <https://opcfoundation.org/about/what-is-opc/>. Acesso em 22 de abril de 2021.
- OPEN62541 (2021). *Release notes*. Disponível em: <https://open62541.org/>. Acesso em 22 de abril de 2021.
- PCMag (2020). *BROWSE ENCYCLOPEDIA - Router*. Disponível em: <https://www.pcmag.com/encyclopedia/term/router>. Acesso em 14 de dezembro de 2020.
- Pepperl+Fuchs (2011). *WIRELESS TECHNOLOGY WirelessHART, Technical White Paper*. Disponível em: https://files.pepperl-fuchs.com/selector_files/navi/productInfo/doct/tdoct1841a_eng.pdf. Acesso em 02 de fevereiro de 2021.
- Phoenix Contact (2011). *WirelessHART - Product Overview*. Disponível em: https://www.phoenixcontact.com/assets/downloads_ed/global/web_dwl_promotion/52005992_EN_HQ_Wireless_HART_Overview_LoRes.pdf. Acesso em 02 de fevereiro de 2021.
- (2021). *Classes of industrial communication cabling*. Disponível em: [https://www.phoenixcontact.com/online/portal/pi?ldmy&urile=wcm:path:/pien/web/main/products/subcategory_pages/Data_plug-in_connectors_P-20-02/f63ebf2a-e6ce-4ec6-8fcc-5c13c558cee0](https://www.phoenixcontact.com/online/portal/pi?ldmy&urile=wcm:path:/pien/web/main/products/subcategory_pages/Data_plug-in_connectors_P-20-02/f63ebf2a-e6ce-4ec6-8fcc-5c13c558cee0/f63ebf2a-e6ce-4ec6-8fcc-5c13c558cee0). Acesso em 31 de janeiro de 2021.
- PI North America (2019). *Topology Options: Fieldbuses and Industrial Ethernets*. Disponível em: <https://us.profinet.com/topology-options-fieldbus-industrial-ethernet/>. Acesso em 15 de novembro de 2020.
- PROFIBUS Nutzerorganisation (2002). *PROFIBUS Technology and Application*. Disponível em: <https://www.automation.siemens.com/sce-static/learning-training-documents/classic/appendix/iv-field-bus-description-en.pdf>. Acesso em 18 de novembro de 2020.
- PROFINET University (2021). *OPC UA and PROFINET*. Disponível em: <https://profinetuniversity.com/industrial-automation-ethernet/opc-ua-profinet/>. Acesso em 30 de abril de 2021.
- Pyramid Solutions (2018). *What is DeviceNet?* Disponível em: <https://pyramidsolutions.com/network-connectivity/blog-nc/devicenet-explained/>. Acesso em 18 de novembro de 2020.
- Renesas Electronics Corporation (2010). *Serial Peripheral Interface (SPI) and Inter-IC (I2C) (SPI-I2C)*. Disponível em: <https://www.idt.com/us/en/document/apn/serial->

- peripheral-interface-spi-inter-ic-i2c-spii2c. Acesso em 18 de novembro de 2020.
- Robotics Knowledgebase (2017). *Bluetooth Socket Programming using Python PyBluez*. Disponível em: <https://roboticsknowledgebase.com/wiki/networking/bluetooth-sockets/>. Acesso em 08 de janeiro de 2022.
- Rockwell Automation (2009). *Guidance for Selecting Cables for EtherNet/IP Networks*. Disponível em: https://literature.rockwellautomation.com/idc/groups/literature/documents/wp/enet-wp007_en-p.pdf. Acesso em 31 de janeiro de 2021.
- (2018). *Deploying a Fiber Optic Physical Infrastructure within a Converged Plantwide Ethernet Architecture*. Disponível em: https://literature.rockwellautomation.com/idc/groups/literature/documents/td/enet-td003_en-p.pdf. Acesso em 01 de fevereiro de 2021.
- Rossow, Alex Brandão (2018). “Integração de ferramentas de modelagem e simulação de sistemas utilizando o protocolo OPC-UA”. Em: *SODEBRAS Journal* 14.159. <http://www.sodebras.com.br/edicoes/N159.pdf>, pp. 131–135.
- RTA (2020). *An Introduction to Modbus RTU Addressing, Function Codes, and Modbus RTU Networking Overview*. Disponível em: <https://www.rtautomation.com/technologies/modbus-rtu/>. Acesso em 18 de novembro de 2020.
- SAMSON (1999). *Technical Information - Serial Data Transmission*. Disponível em: <https://www.samsongroup.com/document/1153en.pdf>. Acesso em 23 de novembro de 2020.
- Schneider Electric (2000). *POWERLOGIC System Architecture and Application Guide*. Disponível em: https://download.schneider-electric.com/files?p_enDocType=User+guide&p_File_Name=System_Guide.pdf&p_Doc_Ref=System_Guide. Acesso em 14 de novembro de 2020.
- (2006). *Chapter 9 - Industrial Networks*. Disponível em: https://www.se.com/ww/en/download/document/asg_9_industrial_networks_EN/. Acesso em 10 de novembro de 2020.
- (2007). *Wiring of RS485 Communications Networks*. Disponível em: https://www.schneider-electric.cn/library/SCHNEIDER_ELECTRIC/SE_LOCAL/APS/188266_2F4E/16798.pdf. Acesso em 14 de novembro de 2020.
- Sense - Sensores e Instrumentos (2014). *Junction BOX - Catalog*. Disponível em: https://www.sense.com.br/arquivos/produtos/arq1/Junction-Box_PA_e_FF_Brochure_Rev_B.pdf. Acesso em 15 de novembro de 2020.
- Siemens (2016). *Industrial Ethernet switches SCALANCE X-100 media converter - Operating Instructions*. Disponível em: <https://cache.industry.siemens.com/dl/>

- files/594/22446594/att_887804/v1/BA_SCALANCE-X100-MC_76.pdf. Acesso em 15 de novembro de 2020.
- Smar Technology Company (2020a). *Redes Industriais*. Disponível em: <https://www.smar.com.brasil/artigo-tecnico/redes-industriais>. Acesso em 10 de novembro de 2020.
- (2020b). *Wireless - ISA 100*. Disponível em: <https://www.smar.com.brasil/artigo-tecnico/wireless-isa-100>. Acesso em 02 de fevereiro de 2021.
- Soft Service - BTFramework (2022). *Bluetooth Framework and RFCOMM Protocol*. Disponível em: <https://www.btframework.com/rfcomm.htm>. Acesso em 05 de janeiro de 2022.
- Software Testing Help (2020). *LAN Vs WAN Vs MAN: Exact Difference Between Types Of Network*. Disponível em: <https://www.softwaretestinghelp.com/lan-wan-man-networks/>. Acesso em 16 de novembro de 2020.
- SourceForge (vfrolov) (2018). *Null-modem emulator (com0com)*. Disponível em: <http://com0com.sourceforge.net/>. Acesso em 12 de agosto de 2021.
- Sparkfun (2020). *Logic Levels*. Disponível em: <https://learn.sparkfun.com/tutorials/logic-levels/all>. Acesso em 25 de novembro de 2020.
- STEVE'S-INTERNET-GUIDE (2021a). *Beginners Guide To The MQTT Protocol*. Disponível em: <http://www.steves-internet-guide.com/mqtt/>. Acesso em 21 de abril de 2021.
- (2021b). *How MQTT Works - Beginners Guide*. Disponível em: <http://www.steves-internet-guide.com/mqtt-works/>. Acesso em 21 de abril de 2021.
- Teleco (2022). *Redes Wi-Fi I: Espectro de Frequência ISM*. Disponível em: https://www.teleco.com.br/tutoriais/tutorialredeswifi1/pagina_5.asp. Acesso em 21 de maio de 2021.
- Texas Instruments (2002). *Texas Instruments - Interface Circuitos for TIA-EIA-232-F*. Disponível em: https://www.ti.com/lit/an/sl1a037a/sl1a037a.pdf?ts=1606251991492&ref_url=https%253A%252F%252Fwww.google.com%252F. Acesso em 24 de novembro de 2020.
- (2004). *AN-759 Comparing EIA-485 and EIA-422-A Line Drivers and Receivers in Multipoint Applications*. Disponível em: https://www.ti.com/lit/an/snla023a/snla023a.pdf?ts=1606269053693&ref_url=https%253A%252F%252Fwww.google.com%252F. Acesso em 24 de novembro de 2020.
- (2008). *Interface Circuits for TIA/EIA-485 (RS-485)*. Disponível em: https://www.ti.com/lit/an/sl1a036d/sl1a036d.pdf?ts=1606272813462&ref_url=https%253A%252F%252Fwww.google.com%252F. Acesso em 24 de novembro de 2020.
- (2010). *RS-422 and RS-485 Standards Overview and System Configurations*. Disponível em: https://www.ti.com/lit/an/sl1a070d/sl1a070d.pdf?ts=1606272813462&ref_url=https%253A%252F%252Fwww.google.com%252F. Acesso em 24 de novembro de 2020.

- 1606226874998&ref_url=https%253A%252F%252Fwww.google.com%252F. Acesso em 24 de novembro de 2020.
- Texas Instruments (2012). *KeyStone Architecture - Serial Peripheral Interface (SPI)*. Disponível em: https://www.ti.com/lit/ug/sprugp2a/sprugp2a.pdf?ts=1606186667797&ref_url=https%253A%252F%252Fwww.google.com%252F. Acesso em 24 de novembro de 2020.
- (2016). *Introduction to the Controller Area Network (CAN)*. Disponível em: https://www.ti.com/lit/an/sloa101b/sloa101b.pdf?ts=1606244140275&ref_url=https%253A%252F%252Fwww.google.com%252F. Acesso em 25 de novembro de 2020.
- Turck Industrial Automation (2020). *Fieldbus Components for FOUNDATION FIELDBUS*. Disponível em: <https://www.turck.de/att/D301024.pdf>. Acesso em 15 de novembro de 2020.
- Unified Automation (2018). *OPC UA Publish-Subscribe (Pub/Sub) - IoT becomes easier*. Disponível em: <https://www.unified-automation.com/news/news-details/article/opc-ua-publish-subscribe-pubsub-iot-becomes-easier.html>. Acesso em 22 de abril de 2021.
- VXCHNGE (2020). *What Are IoT Devices and What Should You Know about Them?* Disponível em: <https://www.vxchnge.com/blog/what-are-iot-devices>. Acesso em 21 de abril de 2021.
- Weed Instrument (2003). *Industrial Fiber Optic Networking for Factory Automation and Process Control*. Disponível em: <http://www.pantek.fr/telecharge/fibernew.pdf>. Acesso em 01 de fevereiro de 2021.
- Weidmuller (2003). *Data Sheet: Fieldbus-Components*. Disponível em: https://www.auser.fi/wp-content/uploads/Vaylahaaroittimet_esite.pdf. Acesso em 15 de novembro de 2020.
- (2012). *Industrial Ethernet Handbook - A practical guideline*. Disponível em: <http://download.weidmueller.com/asset/download/file//40066>. Acesso em 31 de janeiro de 2021.
- Weis, Olga (2020). *Eltima Software - RS232. Settings and configuration of serial protocol*. Disponível em: <https://www.serial-port-monitor.org/articles/serial-communication/rs232-interface/>. Acesso em 24 de novembro de 2020.
- Yang, Jian et al. (2020). “Beyond Beacons: Emerging Applications and Challenges of BLE”. Em: *ArXiv* abs/1909.11737.
- Yokogawa (2017). *Yokogawa Field Wireless Solution*. Disponível em: https://web-material3.yokogawa.com/BU01W01A13-01EN_004.pdf. Acesso em 02 de fevereiro de 2021.

Yokogawa (2018). *ISA100 Wireless gateway optimized for a small system - Gateway Module FN110 (Modbus Communication for PLC/RTU)*. Disponível em: https://web-material3.yokogawa.com/BU01W03A13-01EN_001.pdf. Acesso em 02 de fevereiro de 2021.