

Stefany Gouvêa Vanzeler

**DESENVOLVIMENTO DE UM SISTEMA
DE MONITORAMENTO CAN E
IMPLEMENTAÇÃO DE UM PROTÓTIPO
PARA INTERFACEAMENTO COM O
BARRAMENTO CAN**

Manaus
2015

Stefany Gouvêa Vanzeler

**DESENVOLVIMENTO DE UM SISTEMA
DE MONITORAMENTO CAN E
IMPLEMENTAÇÃO DE UM PROTÓTIPO
PARA INTERFACEAMENTO COM O
BARRAMENTO CAN**

Trabalho de Conclusão de Curso submetido
à Coordenação do curso de Engenharia de
Controle e Automação da Universidade do
Estado do Amazonas como parte dos requi-
sitos necessários para a obtenção do grau de
Engenheiro.

Orientador Me. Angilberto Muniz Ferreira Sobri-
nho

Manaus
2015

Stefany Gouvêa Vanzeler

**DESENVOLVIMENTO DE UM SISTEMA DE
MONITORAMENTO CAN E IMPLEMENTAÇÃO DE
UM PROTÓTIPO PARA INTERFACEAMENTO
COM O BARRAMENTO CAN**

Trabalho de Conclusão de Curso submetido
à Coordenação do curso de Engenharia de
Controle e Automação da Universidade do
Estado do Amazonas como parte dos requi-
sitos necessários para a obtenção do grau de
Engenheiro.

Aprovado em 19 de junho de 2015.

BANCA EXAMINADORA

Me. Angilberto Muniz Ferreira
Sobrinho
Orientador

Dr. Walter Andrés Vermehren
Valenzuela
Presidente da Banca

Me. Reinel Beltran
Convidado 2

Dr. Daniel Guzman
Convidado 3

Manaus
2015

Agradecimentos

Primeiro gostaria de agradecer a Deus, por ter me abençoado e me dado discernimento nesses anos de faculdade.

Um agradecimento especial aos meus pais, Terezinha e Geraldo, pela educação que me deram e por sempre me apoiarem.

Um segundo agradecimento especial à família Rauscher, família Mendes e família TWRMN, que me apoiaram nos meus estudos.

Agradeço também à Universidade do Estado do Amazonas e sua equipe administrativa, à coordenação de Engenharia de Controle e Automação e todos os seus professores, nos quais muitos se tornaram verdadeiros amigos nessa jornada. Agradeço especialmente ao meu orientador, Msc. Angilberto Muniz Ferreira Sobrinho por toda paciencia e dedicacao.

Por fim, e de maneira muito significativa, agradeço aos amigos que criei nessa jornada, e que levarei pra vida inteira: Anderson, Cristiano, Luiz, Luanda, Patrick e Nayana.

Resumo

O presente trabalho tem como objetivo o desenvolvimento de um sistema de monitoramento de barramento CAN, aonde possa-se observar o fluxo de dados entre as ECUs (unidades eletrônicas de controle) e se transmitir mensagens, por meio da implementação de um protótipo, que funcione como um *gateway* entre os protocolos TCP/IP e CAN em um sistema linux embarcado com arquitetura ARM de baixo custo e um computador pessoal com sistema operacional *Windows*, utilizando-se as linguagens C/C++ e C# a partir do conceito de sockets, servindo como modelo para integração futura de dispositivos de campo com a rede *DeviceNet* do laboratório de automacao industrial da EST (Escola Superior de Tecnologia). Para tanto foi feita revisao bibliográfica conferindo embasamento teórico para realização do trabalho e uma comparação entre os minicomputadores *Raspberry PI B+* e *Beaglebone Black*, definindo-se critérios para a escolha de um deles para a implementacao do protótipo. O modelo e arquitetura do sistema foram definidos de acordo com os requisitos do projeto, servindo como base para a implementacao dos módulos de software e hardware, uma rede CAN com duas ECUs (uma com o *Beaglebone Black* e outra com o *Funduino UNO*) foi construída para testes e validacao do funcionamento do sistema de monitoramento e do protótipo. Após validacao um manual do usuário foi criado explicando o funcionamento do sistema desenvolvido.

Palavras-chaves: Controller Area Network. TCP/IP. Gateway. Monitoramento.

Abstract

This bachelor thesis aims to develop a CAN bus monitoring system, in which we can observe the flow of data among the ECUs (electronic control units) and transmit messages by the implementation of a prototype that works as a gateway between the TCP / IP and CAN protocols in a Linux embedded system in a low-cost ARM based platform and a Windows personal computer, using the C / C ++ and C # languages based on the sockets concept, working as model for the future integration of field devices to the DeviceNet network located at the industrial automation laboratory at EST (Amazonas State Technology School). In this regard, a literature revision and a comparison between the Raspberry PI B + and Beaglebone Black minicomputers were done, defining criteria for choosing one of them to implement the prototype. The model and system architecture were defined according to the project requirements, serving as a basis for the implementation of the software and hardware modules, a CAN network with two ECUs (one with Beaglebone Black and another with Funduino UNO) was built for tests and validation of the monitoring system and the prototype. After validation, a user's manual was created explaining the operation of the developed system.

Key-words:Controller Area Network. TCP/IP. Gateway. Monitoring System.

Lista de ilustrações

Figura 1 – Metodologia	14
Figura 2 – Pirâmide da Automação	15
Figura 3 – Níveis de redes	16
Figura 4 – Topologias ponto-a-ponto: estrela, anel, malha regular, malha irregular e árvore	18
Figura 5 – Topologias das redes em difusão: barramento, satélite e anel	18
Figura 6 – Comunicação serial ponto-a-ponto	19
Figura 7 – Comunicação serial mestre - escravo	20
Figura 8 – Comunicação serial multimestre	20
Figura 9 – Produtor-Consumidor	21
Figura 10 – Modelo de Referência OSI	23
Figura 11 – Lógica AND	27
Figura 12 – Nível lógico	27
Figura 13 – Frame de dados versão padrão e estendida	28
Figura 14 – <i>Frame</i> de Erro e de Sobrecarga	30
Figura 16 – Rede CAN	32
Figura 15 – Mecanismos de Detecção de Erros	32
Figura 17 – Meio físico do DeviceNet	36
Figura 18 – Suíte de Protocolos TCP/IP	37
Figura 19 – Fluxo de Dados no Modelo TCP/IP	38
Figura 20 – Frame de Ethernet IEEE 802.3	39
Figura 21 – Cabecalho do Datagrama/Pacote IPV4	41
Figura 22 – Cabecalho do Segmento TCP	43
Figura 23 – Modelo de Hardware	47
Figura 24 – Modelo de Software	47
Figura 25 – Beaglebone Black	49
Figura 26 – Diagrama de Blocos do AM335x	49
Figura 27 – BeagleBone Serial Cape	51
Figura 28 – Circuito CAN do <i>BeagleBone Cape</i>	51
Figura 29 – Buffer de Dados CAN	52
Figura 30 – Funduino UNO	52
Figura 31 – Apto CAN Bus Shield	53
Figura 32 – Arquitetura de Hardware	54
Figura 33 – Arquitetura de Software	55
Figura 34 – Máquina de estados <i>Socketcand</i> *	57
Figura 35 – PuTTy Interface	57

Figura 36 – PuTTy Terminal	58
Figura 37 – WinSCP	58
Figura 38 – TightVNC Viewer	59
Figura 39 – Conexão Remota SSH Usando <i>Eclipse Luna</i>	60
Figura 40 – Configuração do IPv4 do computador local	61
Figura 41 – Servidor DHCP	61
Figura 42 – Rede CAN	64
Figura 43 – Configuração da <i>Socketcan*</i>	65
Figura 44 – Monitoramento CAN BUS	66
Figura 45 – Filtro no estado BCM	67
Figura 46 – Estado RAW	68
Figura 47 – Envio cíclico da mensagem 002	69
Figura 48 – Led azul acende com a mensagem cíclica 002 4 00 00 00 01 ou com o pressionar do botão	69

Lista de tabelas

Tabela 1 – Classificacao das redes quanto à proximidade geográfica	17
Tabela 2 – Modelo OSI	22
Tabela 3 – Redes CAN	24
Tabela 4 – Arquitetura CAN	25
Tabela 5 – Arquitetura DeviceNet	35
Tabela 6 – Arquitetura Ethernet	39
Tabela 7 – Chamadas de sistema da Berkeley Socket	45
Tabela 8 – Raspeberry PI B+ VS Beaglebone Black	48
Tabela 9 – Adaptadores para Barramento CAN Beaglebone Black	50
Tabela 10 – Adaptadores para Barramento CAN Raspberry PI B+	50
Tabela 11 – Atuadores Rede CAN	63
Tabela 12 – Sensores Rede CAN	63
Tabela 13 – Mensagens CAN	64

Sumário

1	INTRODUÇÃO	11
1.1	Formulação do problema	11
1.2	Justificativa	12
1.3	Motivação	12
1.4	Objetivos	12
1.5	Metodologia	13
1.6	Organização do trabalho	14
2	REFERENCIAL TEÓRICO	15
2.1	Classificação das Redes em Relação à Pirâmide de Automação	15
2.2	Classificação das Redes de Computadores Quanto à Proximidade Geográfica	17
2.3	Classificação das Redes Quanto à Topologia	17
2.3.1	Topologias com Canais Ponto-a-Ponto	18
2.3.2	Topologias das Redes em Difusão	18
2.4	Tipos de Comunicação Serial	19
2.4.1	Ponto-a-Ponto	19
2.4.2	Mestre-Escravo	19
2.4.3	Multimestre	20
2.4.4	Produtor-Consumidor	20
2.5	Modelo OSI	21
2.6	Barramento CAN	23
2.6.1	Definição	23
2.6.2	Características	24
2.6.3	A Arquitetura CAN	25
2.6.4	Camada Física	26
2.6.5	Camada de Enlace	28
2.7	Rede DeviceNet	33
2.7.1	Conceito	33
2.7.2	Características	34
2.7.3	A Arquitetura DeviceNet	35
2.8	Suite de Protocolos TCP/IP (<i>Transmission Control Protocol / Internet Protocol</i>)	36
2.8.1	Camada de Interface de Rede	38
2.8.2	Camada de Internet	40
2.8.2.1	IP (Internet Protocol)	40

2.8.2.2	ARP (Address Resolution Protocol)	42
2.8.2.3	ICMP (Internet Control Message Protocol)	42
2.8.3	Camada de Transporte	42
2.9	Interfaces para Programas de Aplicação (APIs)	44
 3	MATERIAIS E MÉTODOS	46
3.1	Modelo do Sistema	46
3.1.1	Especificação dos Requerimentos do Sistema	46
3.1.2	Modelo de Hardware	47
3.1.3	Modelo de Software	47
3.2	Arquitetura do Sistema	48
3.2.1	Comparativo entre as Arquiteturas Raspberry Pi e Beaglebone Black .	48
3.2.2	Unidade Eletrônica de Controle Complementar	52
3.2.3	Arquitetura de Hardware	53
3.2.4	Arquitetura de Software	54
3.3	Implementação	57
3.3.1	Ferramentas de Conexão Remota	57
3.3.2	Configuração da Rede Ethernet TCP/IP	60
3.3.3	Configuração da Rede CAN	61
3.3.4	Montagem da Rede CAN e TCP/IP	63
3.3.5	Configuração da Socketcand*	64
 4	RESULTADOS E DISCUSSÕES	66
4.1	Sistema de Monitoramento CAN	66
4.1.1	Manual do Usuário do Módulo Monitoramento CAN BUS	66
4.2	Discussão dos Resultados	68
 5	CONCLUSÕES	70
5.1	Dificuldades encontradas	70
5.2	Trabalhos futuros	70
 REFERÊNCIAS		71
 APÊNDICE A – ALGORITMO IMPLEMENTADO NO FUN-		
DUINO UNO (TESTESTATUS)		74
 APÊNDICE B – ALGORITMO IMPLEMENTADO NO PRO-		
TÓTIPO (TESTE)		79

1 INTRODUÇÃO

As redes industriais surgiram com a necessidade da integração dos equipamentos e dispositivos em todos os níveis de automação. Elas vêm sendo cada vez mais utilizadas devido ao desenvolvimento da informática e a constante redução dos custos de componentes microcontrolados, o que possibilitou o desenvolvimento dos sistemas com comunicação serial em larga escala aplicados às redes de comunicação e ao desenvolvimento de vários protocolos de comunicação digital. Elas apresentam como grande vantagem à redução significativa de cabos de controle e seus acessórios que interligam os diversos dispositivos presentes em um sistema automatizado para transmitir informações, que podem ser de Input/Output (I/O), parâmetros e diagnósticos, proporcionando assim aumento da eficiência, qualidade e segurança no sistema produtivo, além de maior facilidade na instalação, manutenção e configuração dos equipamentos, fornecimento de diagnósticos rápidos e redução de custos.

O barramento Controller Area Network (CAN) é um sistema de barramento serial projetado para facilitar a comunicação entre dispositivos, como sensores e atuadores, que diretamente interagem com o processo. Ele foi desenvolvido pela empresa alemã Robert BOSCH e disponibilizado em meados dos anos 80 baseado na ISO 11898, voltado inicialmente para a indústria automobilística. Em decorrência das suas características, normalização e baixo custo de produção, tornou-se amplamente utilizado na indústria, (WANG; HE, 2013).

O presente projeto tem como objetivo o desenvolvimento de um sistema de monitoramento de barramento CAN, aonde possa-se observar o fluxo de dados entre as ECUs (unidades eletrônicas de controle) e se transmitir mensagens, por meio da implementação de um protótipo, que funcione como um *gateway* entre os protocolos TCP/IP e CAN em um sistema linux embarcado com arquitetura ARM de baixo custo e um computador pessoal com sistema operacional Windows, utilizando-se as linguagens C/C++ e C# a partir do conceito de sockets, servindo como modelo para integração futura de dispositivos de campo com a rede DeviceNet do laboratório de automação industrial da EST (Escola Superior de Tecnologia).

1.1 Formulação do problema

A inexistência de um dispositivo compatível com o barramento CAN, o qual permita simplificar a rede de comunicação do laboratório de automação industrial da EST, que possui múltiplas interfaces e protocolos incompatíveis, e que sirva de base para a comunicação futura entre o CLP DeviceNet com novos dispositivos.

1.2 Justificativa

O desenvolvimento do sistema de monitoramento CAN e implantação do protótipo podem servir como base para novos projetos nas áreas de: redes indústrias, sistemas distribuídos, controle, automação, supervisão de processos e sistemas supervisórios. Além de poder ser utilizado em aulas práticas no laboratório de automação industrial da EST.

Academicamente, o presente trabalho contribui para a consolidação e aprofundamento nos conteúdos das disciplinas estudadas na EST de Redes Industriais de Comunicação, Redes de Computadores I e II, Microprocessadores e Microcontroladores, Sistemas Microprocessados, Interface entre Usuários e Sistemas Computacionais, Sistemas Supervisório e Automação e Supervisão de Processos.

1.3 Motivação

A utilização da arquitetura CAN torna-se uma alternativa econômica de rede de dados para muitas aplicações industriais por prover um acondicionamento entre sensores e atuadores e toda a distribuição de controle em tempo real de dados através do barramento, além de possuir um sistema de gerenciamento de erros confiável. Aliado a esses fatos, é importante também existir uma interface entre a rede CAN e as outras redes, proporcionando o seu controle e monitoramento remoto, (ZHANG; FENG; GUO, 2013). A rede Ethernet TCP/IP tem sido amplamente também utilizada na indústria, por poder carregar grande quantidade de dados e ser adequada para o nível de gerenciamento na automação industrial, porém não sendo adequada para sistemas em tempo real. Assim surge-se a necessidade da implementação de uma *gateway* entre esses dois protocolos no laboratório de automação industrial da EST e o desenvolvimento de um sistema de monitoramento de barramento CAN, (SARGUNAPRIYA; MANI; AMUDHA, 2014), promovendo uma interface educativa e de controle da rede CAN.

1.4 Objetivos

Desenvolver um sistema de monitoramento de barramento CAN e implementar um protótipo, que funcione como um *gateway* entre os protocolos TCP/IP e CAN em um sistema linux embarcado com arquitetura ARM de baixo custo, servindo de interface entre o sistema de monitoramento em um computador pessoal com sistema operacional Windows e o barramento CAN. Este objetivo geral será alcançado quando todos seguintes objetivos específicos forem atingidos:

- Realizar revisão bibliográfica que confere embasamento teórico para realização do trabalho;

- Fazer um comparativo entre as arquiteturas Raspberry Pi B+ e Beaglebone Black e escolher uma dessas plataformas ARM para a implementação do protótipo;
- Determinar o controlador CAN e receptor CAN;
- Implementar um *gateway* entre os protocolos CAN e TCP/IP;
- Desenvolver um sistema de monitoramento para barramento CAN com interface TCP/IP;
- Montar uma rede CAN para testes;
- Testar o recebimento e envio de mensagens por meio do protótipo, monitorando em tempo real o barramento CAN da rede montada para testes com o sistema de monitoramento criado.

1.5 Metodologia

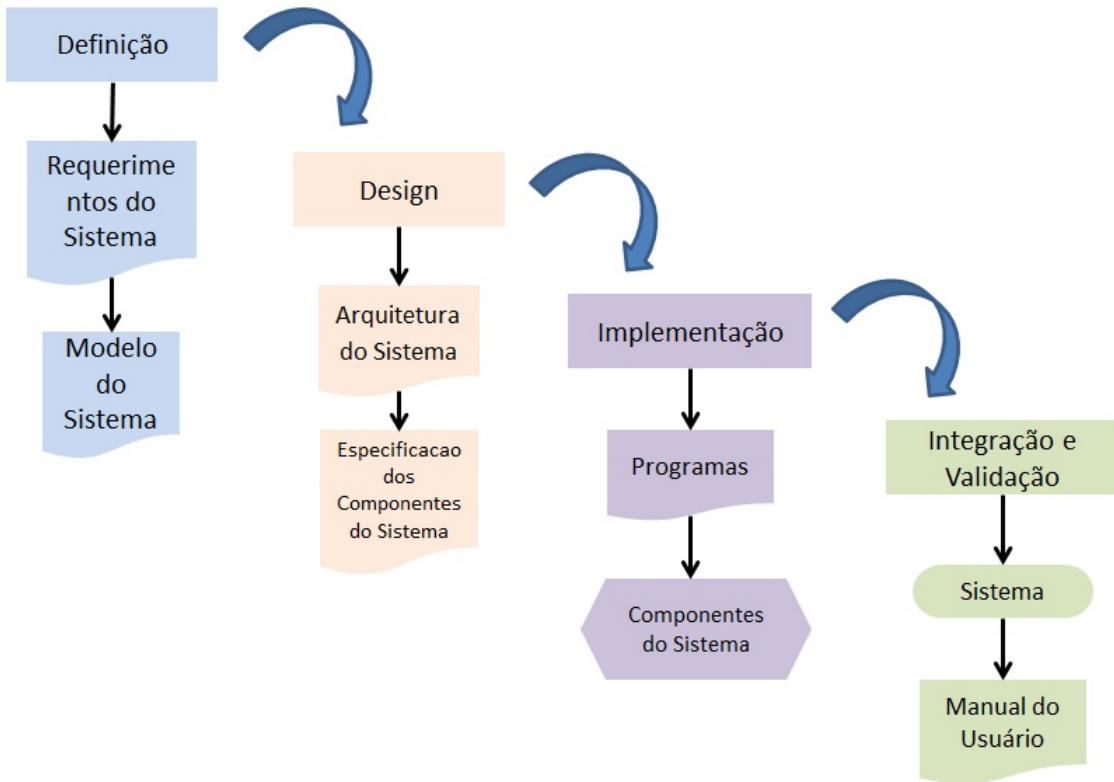
Inicialmente foi realizada uma revisão bibliográfica conferindo embasamento teórico para realização do trabalho, em seguida o projeto foi dividido em 4 fases: definição, design, implementação e integração e validação, figura 1.

Na primeira fase foram definidos os requisitos do sistema a partir dos objetivos e criado um modelo geral do sistema.

Na fase de design foi realizado um comparativo entre os minicomputadores *Raspberry Pi* + e *Beaglebone*, adotando-se critérios para a escolha de uma dessas plataformas para a implementação do protótipo, a compra de materiais e montagem da arquitetura de hardware e especificação dos componentes.

Após o design foi feita a implementação e de cada componente. Por último foi feita a integração desses componentes e validação do sistema desenvolvido, sendo confeccionado um manual do usuário, explicando o funcionamento do sistema de monitoramento CAN com o protótipo.

Figura 1 – Metodologia



fonte: Autora

1.6 Organização do trabalho

Este trabalho está dividido em cinco capítulos para seu melhor entendimento.

O Capítulo 1 contém uma breve introdução sobre o assunto a ser explanado no trabalho, sua motivação, justificativa para estudar as áreas mencionadas e um resumo da metodologia empregada no desenvolvimento do projeto.

O Capítulo 2 contém um referencial teórico apresentando os tópicos relevantes por meio da visão de autores de obras consultadas e de trabalhos de pesquisa voltados para teorias e tecnologias aplicadas nesta monografia.

O Capítulo 3 detalha os materiais e técnicas empregados no desenvolvimento do sistema de monitoramento CAN e a implementação do protótipo, citando os materiais utilizados.

O Capítulo 4 contém a análise dos resultados obtidos após execução da metodologia e o manual de usuário do sistema de monitoramento CAN.

O Capítulo 5 contém as conclusões que podem ser afirmadas após execução do projeto, fazendo análise de qualidade dos resultados obtidos e os confronta com os objetivos propostos pelo projeto.

Ao final do trabalho estão incluídos o Apêndice (A) e Anexo (A) com código fonte de alguns módulos de software produzidos.

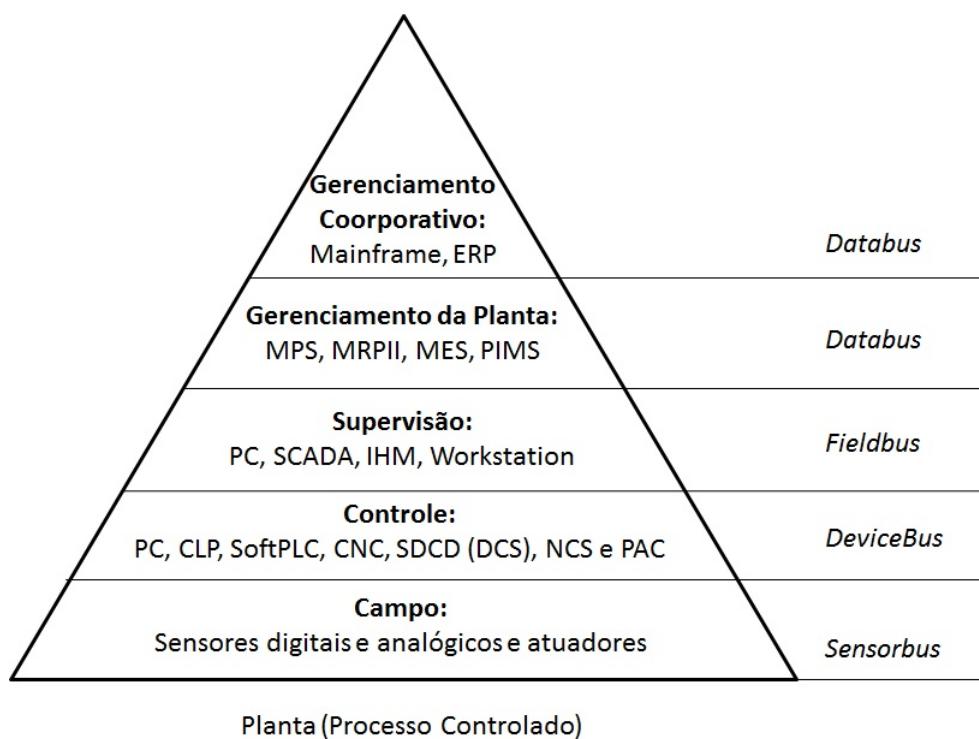
2 REFERENCIAL TEÓRICO

Neste capítulo será realizada uma breve abordagem teórica dos seguintes assuntos, que são necessários para o desenvolvimento do presente trabalho: classificação das redes em relação a pirâmide de automação, classificação das redes de computadores quanto à proximidade geográfica, classificação das redes quanto à topologia, tipos de comunicação serial, modelo OSI, barramento CAN, Rede DeviceNet, suite de protocolos TCP/IP (*Transmission Control Protocol / Internet Protocol*) e interfaces para programas de aplicação (APIs).

2.1 Classificação das Redes em Relação à Pirâmide de Automação

Segundo UEL (2010) e Pereira (2013) a pirâmide da automação industrial é um diagrama que representa, de forma hierárquica, os diferentes níveis de controle e trabalho em automação industrial em cinco níveis. Paralelamente aos níveis da automação, as redes podem ser classificadas em *sensorbus*, *devicebus*, *fieldbus* e *databus*, de acordo com o volume de dados a ser transmitido, tempo de resposta e distância a ser percorrida como mostrado na figura 2.

Figura 2 – Pirâmide da Automação



Fonte: Autora

Nível de Campo: É o nível das máquinas, dispositivos e componentes (chão-de-fábrica), constituído por sensores e atuadores.

Nível de Controle: Nível onde se encontram os equipamentos que executam o controle automático da planta, ou seja, controladores digitais, dinâmicos e lógicos, e de algum tipo de supervisão associada ao processo. Aqui se encontram concentradores de informações sobre o Nível 1 e as Interfaces Homem-Máquina (IHM).

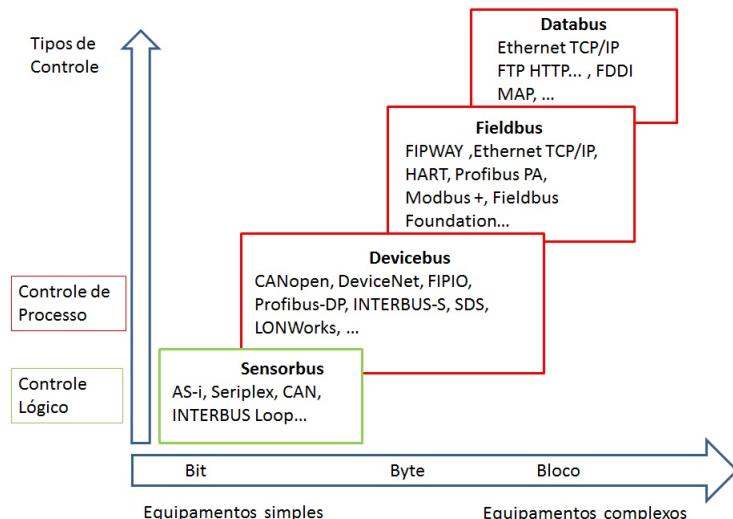
Nível de Supervisão: Permite a supervisão e controle do processo produtivo da planta. Constituído por banco de dados, com informação sobre índices de qualidade da produção, relatórios e estatísticas de processo, índices de produtividade, algoritmos de otimização da operação produtiva, sendo constituído por Sistemas de Supervisão e Aquisição de Dados (SCADA), interface homem-máquina (IHM) e otimizadores de processo dentro do conceito de APC (*Advanced Process Control*).

Nível de Gerenciamento da Planta: Nível da programação e planejamento da produção, realizando o controle e a logística dos suprimentos, como por exemplo: MES (*Manufacturing Execution System*), PIMS (*Process Information Management System*), APS(*Advanced Planning and Scheduling*), LIMS (*Lab Information System*), Sistemas de Manutenção (*MaintenanceManagement System*), Sistema de Gestão de Ativos (*Asset Management System*).

Nível de Gerenciamento Corporativo: Administração dos recursos da empresa. Neste nível encontram-se os softwares para gestão de vendas e financeira.

A figura 3 ilustra os tipos de redes atuais de acordo com os tipos de controle segundo Schneider Electric (2002) e ??).

Figura 3 – Níveis de redes



Fonte: Autora

Sensorbus: comunica com sensores analógicos e discretos e atuadores. As mensagens de dados possuem tamanho de alguns bits, a frequência de comunicação dura de dezenas de milisegundos sobre dezenas de metros com concepção determinística.

Devicebus: faz a comunicação com periféricos controlados. As mensagens de dados são de bytes ou words, a frequência de comunicação tem uma duração de dezenas de milisegundos sobre centenas de metros com concepção determinística.

Fieldbus: faz a integração entre unidades inteligentes. As Mensagens de dados são de words ou blocos, a frequência de comunicação tem uma duração de centenas de milisegundos sobre centenas de metros.

Databus: efetua transferência maciça de dados entre equipamentos. As mensagens de dados são de blocos, a frequência de comunicação tem uma duração de segundos ou minutos sobre grandes distâncias (LAN / WAN / Internet).

2.2 Classificação das Redes de Computadores Quanto à Proximidade Geográfica

As redes podem ser classificadas de acordo com seu tamanho. O seu tamanho pode ser expressa pela área geográfica que ocupam e o número de computadores que fazem parte da rede. Assim podemos classificar as redes em: PAN (*personal area network*), LAN (*local area network*) , MAN (*metropolitan area network*) e WAN (*wide area network*) até chegar à rede das redes, a internet, de acordo com a Tabela 1.

Tabela 1 – Classificação das redes quanto à proximidade geográfica

Distância entre hosts	Área de Abrangência	Classificação
1m	Metro quadrado	PAN
10m	Quarto	
100 m	Prédio	LAN
1 Km	Campus	
10 Km	Cidade	MAN
100 Km	País	
1.000 Km	Continente	WAN
10.000 Km	Planeta	
Fonte: (TANENBAUM, 2003)		Internet

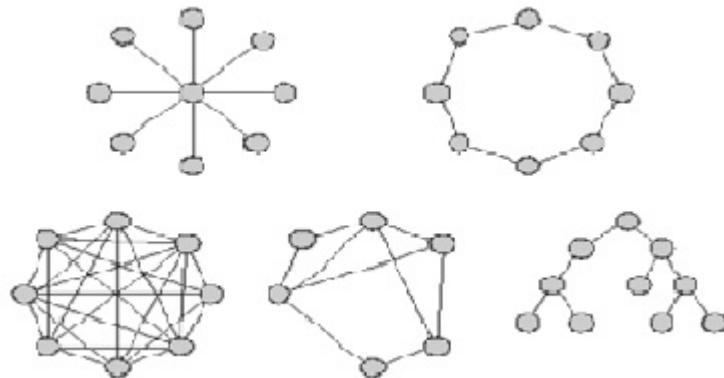
2.3 Classificação das Redes Quanto à Topologia

Topologia da rede é a definição da maneira como as diferentes estações serão interligadas. Estão relacionadas à forma como o canal de comunicação será alocado, ou seja, através de canais ponto-a-ponto ou canais de difusão, (TANENBAUM, 2003).

2.3.1 Topologias com Canais Ponto-a-Ponto

A rede é composta de diversas linhas de comunicação, cada linha sendo associada à conexão de um par de estações⁴. Também conhecida com *store-and-forward*. As redes ponto-a-ponto podem ser concebidas segundo diferentes topologias, simétricas (redes locais) e assimétricas (redes de longa distância).

Figura 4 – Topologias ponto-a-ponto: estrela, anel, malha regular, malha irregular e árvore



Fonte: (BASTOS, 2012)

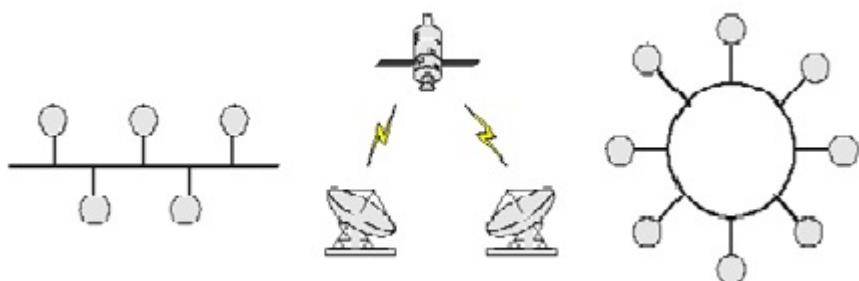
2.3.2 Topologias das Redes em Difusão

Redes em difusão são caracterizadas pelo compartilhamento por todas as estações, de um único canal de comunicação. Neste caso, as mensagens enviadas por uma estação são recebidas por todas as demais conectadas ao suporte de transmissão, sendo que um campo de endereço contido na mensagem permite identificar o destinatário.

Nas redes em difusão existe a possibilidade de uma estação enviar uma mensagem às demais estações da rede, utilizando um código de endereçamento especial, denominado *broadcasting*.

Na transmissão para um subconjunto de estações, o código de endereçamento especial é denominado *multicasting*.

Figura 5 – Topologias das redes em difusão: barramento, satélite e anel



Fonte: (BASTOS, 2012)

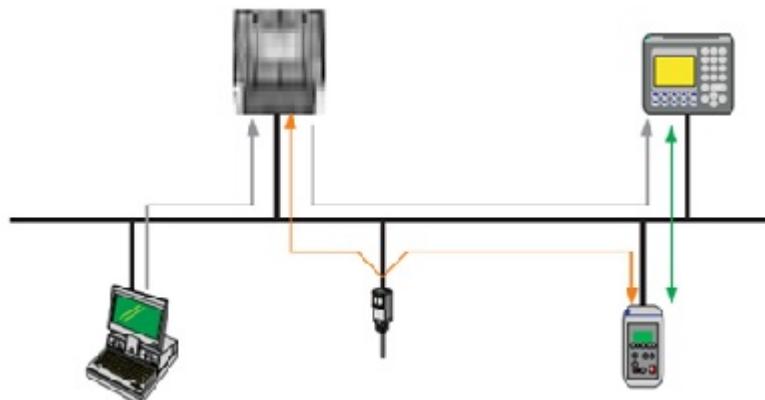
2.4 Tipos de Comunicação Serial

O tipo de comunicação define a conexão entre os equipamentos e a maneira como é feita a troca das informações no que se diz respeito ao caminho percorrido pelos dados, (ROSSIT, 2002).

2.4.1 Ponto-a-Ponto

Ponto-a-ponto (*Peer-to-peer*) é uma arquitetura de redes onde cada um dos pontos ou nós da rede (host) funciona tanto como cliente quanto como servidor, permitindo compartilhamentos de serviços e dados sem a necessidade de um servidor central. Sendo amplamente empregada em equipamentos autônomos, que normalmente realizam suas tarefas sozinhos, mas necessitam de configuração ou dados para manipulação, como exemplo podemos citar: um computador e o mouse. No exemplo abaixo, a comunicação ponto a ponto é utilizada por um sensor que envia dados para um controlador e um analisador.

Figura 6 – Comunicação serial ponto-a-ponto

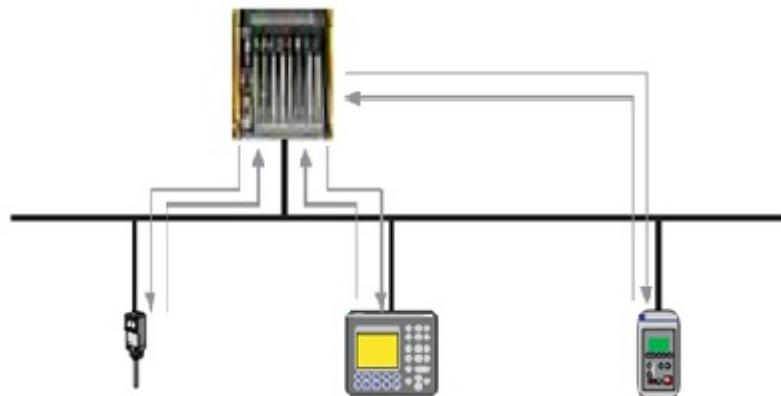


Fonte: (ROSSIT, 2002)

2.4.2 Mestre-Escravo

A comunicação mestre - escravo é amplamente utilizada, possui um mestre para gerenciar a comunicação, e tem como função solicitar e receber os dados e comandos. Os outros participantes da rede conhecidos como escravos, que nunca iniciam uma comunicação e respondem com dados para o mestre, que mantém uma lista de todos os escravos presentes na rede e rotineiramente solicita para cada escravo a troca de dados. O termo mestre é compatível com os termos cliente e *scanner*, o termo escravo é compatível com os termos servidor e adaptador.

Figura 7 – Comunicação serial mestre - escravo

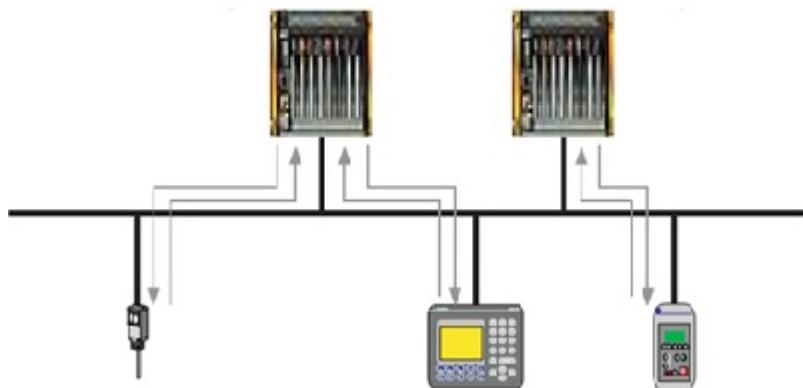


Fonte: (ROSSIT, 2002)

2.4.3 Multimestre

Oferece a possibilidade de mais de um mestre utilizar o mesmo meio físico, mas na prática poucos protocolos permitem a troca de dados de um escravo para os dois mestres, sendo comum neste tipo de configuração que cada mestre possua seu conjunto de escravos.

Figura 8 – Comunicação serial multimestre

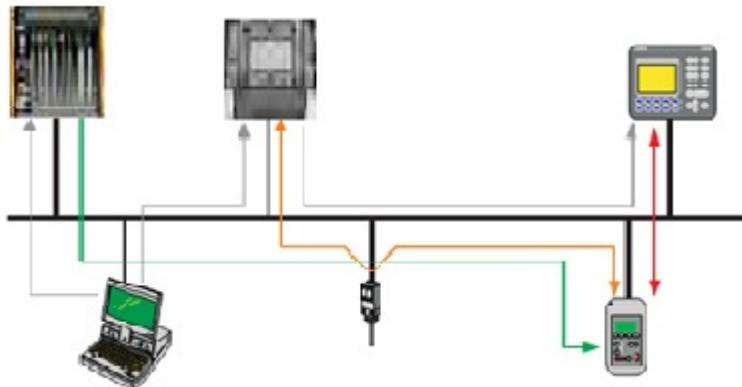


Fonte: (ROSSIT, 2002)

2.4.4 Produtor-Consumidor

As redes produtor-consumidor suportam os três métodos de comunicação expostos anteriormente: ponto-a-ponto, mestre-escravo e multimestre. Do ponto de vista prático, esta forma de comunicação é mais flexível, pois dependendo da natureza da informação a ser trocada pode-se optar pela forma mais adequada, otimizando o barramento no que diz respeito ao tráfego. A rede *DeviceNet* é um exemplo.

Figura 9 – Produtor-Consumidor



Fonte: (ROSSIT, 2002)

2.5 Modelo OSI

Para reduzir a complexidade do projeto, a maioria das redes é organizada como uma pilha de camadas ou níveis, colocadas umas sobre as outras. O número de camadas, o nome, o conteúdo e a função de cada camada diferem de uma rede para outra. A ideia fundamental é que um determinado item de software ou hardware (camada inferior) fornece um serviço a seus usuários (camada superior), mas mantém ocultos os detalhes de seu estado interno e de seus algoritmos.

A camada n de uma máquina se comunica com a camada n de outra máquina. Coletivamente, as regras e convenções usadas nesse diálogo são conhecidas como o protocolo da camada n . Assim um serviço é um conjunto de primitivas (operações) que uma camada oferece à camada situada acima dela e o protocolo é um conjunto de regras que controla o formato e o significado dos pacotes ou mensagens que são trocadas pelas entidades pares contidas em uma camada (TANENBAUM, 2003).

Entre cada par de camadas adjacentes existe uma interface. A interface define as operações e os serviços que a camada inferior tem a oferecer à camada que se encontra acima dela.

Um conjunto de camadas e protocolos é chamado arquitetura de rede. Uma lista de protocolos usados por um determinado sistema, um protocolo por camada, é chamada pilha de protocolos (TANENBAUM, 2003).

O modelo OSI se baseia em uma proposta desenvolvida pela ISO (*International Standards Organization*) como um primeiro passo em direção à padronização internacional dos protocolos empregados nas diversas camadas. O modelo é chamado Modelo de Referência ISO OSI (*Open Systems Interconnection*), pois ele trata da interconexão de sistemas abertos — ou seja, sistemas que estão abertos à comunicação com outros sistemas (TANENBAUM, 2003).

O modelo OSI define sete camadas segundo ??) conforme descrito na tabela 2: camada física, camada de enlace, camada de rede, camada de transporte, camada de sessão, camada de apresentação e camada de aplicação.

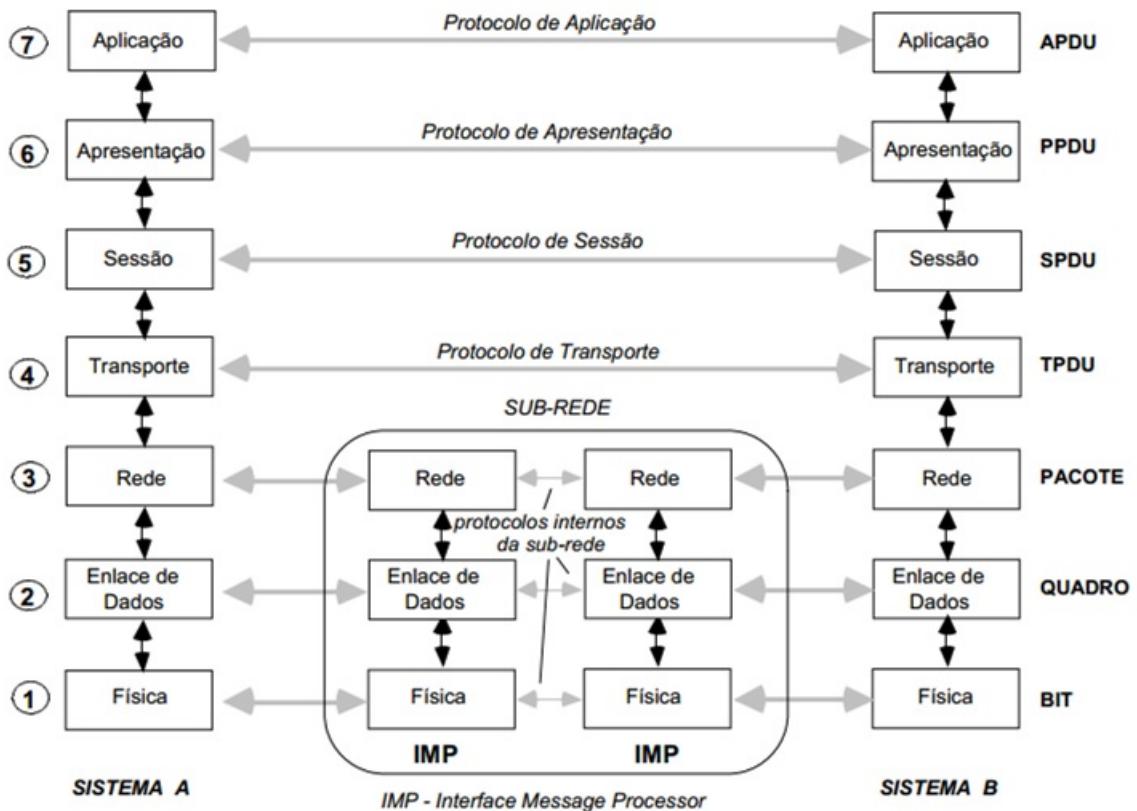
Tabela 2 – Modelo OSI

	Camada OSI	Funções	Exemplos
7	Aplicação	Interface entre rede e software de aplicação.	Telnet, HTTP, WWW-Browsers
6	Apresentação	Como os dados são apresentados, Formatação de dados e conversão de caracteres e códigos, criptografia.	JPEG, ASCII
5	Sessão	Negociação e estabelecimento de conexão com outro nó; Administração e sincronização de diálogos entre dois processos de aplicação.	OS, Aplicações de acesso.
4	Transporte	Entrega de dados com ou sem confiabilidade; Multiplexação; Controle de fluxo.	TCP, UDP, SPX
3	Rede	Endereçamento lógico, o qual roteadores utilizam para determinação da rota dos dados; Controle de fluxo.	IP, IPX
2	Enlace	Combinação de bits em bytes, e bytes em quadros (frames); Acesso ao meio usando o endereço MAC; Sincronização; Controle de fluxo; Confiabilidade de dados.	Ethernet 802.3 / 802.2
1	Física	Transmissão de bits entre os dispositivos; Especificação de voltagem para representar 0 e 1; Velocidade de transmissão; Número de pinos do conector; Diâmetro dos condutores.	EIA/TIA-232, V.35

Fonte: Autora

Na realidade, os dados não são transferidos diretamente da camada n de uma máquina para a camada n de outra máquina. Em vez disso, cada camada transfere os dados e as informações de controle para a camada imediatamente abaixo dela, até ser alcançada a camada mais baixa. Abaixo da camada 1 encontra-se o meio físico através do qual se dá a comunicação propriamente dita. Na figura 10, a comunicação virtual é mostrada por linhas pontilhadas e a comunicação física por linhas contínuas (TANENBAUM, 2003).

Figura 10 – Modelo de Referência OSI



Fonte: (STEMMER, 2001)

2.6 Barramento CAN

2.6.1 Definição

O Barramento CAN (*Controller Area Network*) é um sistema de barramento serial projetado para facilitar a comunicação entre dispositivos, como sensores e atuadores, que diretamente interagem com o processo, sendo por isso considerado um barramento de campo. Ele foi desenvolvido pela empresa alemã Robert BOSCH e disponibilizado em meados dos anos 80, baseado na ISO 11898, devido a crescente necessidade de se interligar os componentes eletrônicos dos veículos modernos. Em decorrência da sua normalização e baixo custo de produção, tornou-se amplamente utilizado na indústria. segundo Froehlich, Souza e Bratti (2008) o CAN consiste basicamente de um padrão de hardware com diferentes tipos de quadros (*frames*), regras de decisão para a transmissão de mensagens e métodos para detecção e correção de erros. Existem três principais tipos de redes CAN em uso, conforme tabela 3, (Vector Informatik GmbH, 2014).

Tabela 3 – Redes CAN

Nomenclatura	Padrão	Taxa Máxima	Identificador
CAN baixa velocidade	ISO 11898-3	125kps	11 bits
Versão 2.0A Alta velocidade (formato padrão)	ISO 11898-2	1 Mbps CAN FD > 1 Mbps	11 bits
Versão 2.0B Alta velocidade (formato estendido)	ISO 11898-2	1 Mbps CAN FD > 1 Mbps	29 bits

Fonte: Autora

CAN com taxa de transmissão variável (*Flexible Data-Rate*, CAN FD) é uma extensão para o protocolo CAN original, conforme especificado na norma ISO 11898-1, que responde a necessidade do aumento de largura de banda nas redes automotivas.

Uma rede pode padrão está limitada a 1 Mbps, com uma carga máxima de 8 bytes por quadro. CAN FD aumenta a taxa efetiva de dados, permitindo campos de dados mais longos - até 64 bytes por quadro - sem alterar a camada física CAN. CAN FD também mantém normal a arbitragem do barramento CAN, aumentando a taxa de transmissão de bits somente a partir do final do processo de arbitragem e retorno à taxa de transmissão inicial depois do delimitador do CRC no quadro, antes que os nós receptores no barramento transmitam o *acknowledgement* bit (confirmação de recebimento do quadro).

2.6.2 Características

Segundo Institute of Industrial Automation and Software Engineering (2015) e Santos (2002) as características de uma rede CAN são enumeradas:

- Topologia linear
- Atribuição de prioridade às mensagens;
- Multimestre, com arbitragem de barramento não destrutiva
- Método de detecção de erros sofisticado e mecanismos de gerenciamento de erros
- Consistência dos dados
- Curto tempos de latência
- Hardware padrão.
- Flexibilidade de configuração
- Regime *multicast* devido ao filtro de mensagens
- Distinção entre erros temporários e erros permanentes dos nós
- Baixo tempo de recuperação de erro

- Considerável imunidade ao ruído;
- Simplicidade;
- Retransmissão automática de mensagens “em espera” logo que o barramento esteja livre;

2.6.3 A Arquitetura CAN

A arquitetura de rede CAN define duas camadas das sete do modelo OSI, a camada de enlace (*data-link*) e a camada física (*physical-layer*) conforme Tabela 4, (Vector Informatik GmbH, 2014).

Utilizando o protocolo CAN, a ligação entre o nível físico (nível 1 OSI) e o de aplicação (nível 7 OSI) é feita utilizando vários protocolos emergentes ou através de software desenvolvido pelo usuário. O exemplo de um protocolo, baseado em CAN, segundo Neto (2010) padrão industrial ao nível de aplicação é o DeviceNet da Allen-Bradley o qual é utilizado para ligar em rede CLPs (controladores lógicos programáveis) e sensores inteligentes.

Tabela 4 – Arquitetura CAN

Modelo OSI		Padrão ISO		Implementação	
7	Aplicação		CAN Kingdom, NMEA 2000, DeviceNet, ISO 11783		Microcontrolador
:			CANOpen, SAE J1939, DIN 9684, TTCAN, SDS		
2	Enlace	LLC		Controlador	Controlador
		MAC	ISO 11988-1	CAN	CAN FD
1	Física	PLS		Transceptor CAN	
		PMA	ISO 11898-2	Meio de transmissão	
		PMS	ISO 11988-3		
		MDI	CIA-DS102	Conectores	

Fonte: Autora

LLC (*Logical Link Control*): gerencia o controle e notificação de sobrecarga (*overload*), o filtro de mensagens e funções de gestão de recuperação.

MAC (*Medium Access Control*): fornece mecanismos de controle de acesso ao canal, o que torna possível vários nós se comunicarem dentro de uma rede de acesso múltiplo em um barramento: encapsulamento de dados / desencapsulamento, detecção e controle de erros, bit *stuffing/destuffing* e funções de serialização e desserialização.

PLS (*Physical Layer Signalling*): responsável pela codificação e decodificação de bits, *bit-timing* (duração de um bit) e sincronização.

PMA (*Physical Medium Attachement*): determina as características do transceptor.

PMS (*physical Medium Specification*): descreve juntamente com a PMA as camadas can de alta velocidade (*high-speed CAN*) e can de baixa velocidade (*low-speed CAN*), diferindo em voltagem e taxas de transmissão.

MDI (*Medium Dependent Interface*): descreve o meio de transmissão juntamente com a PMS, além das conexões.

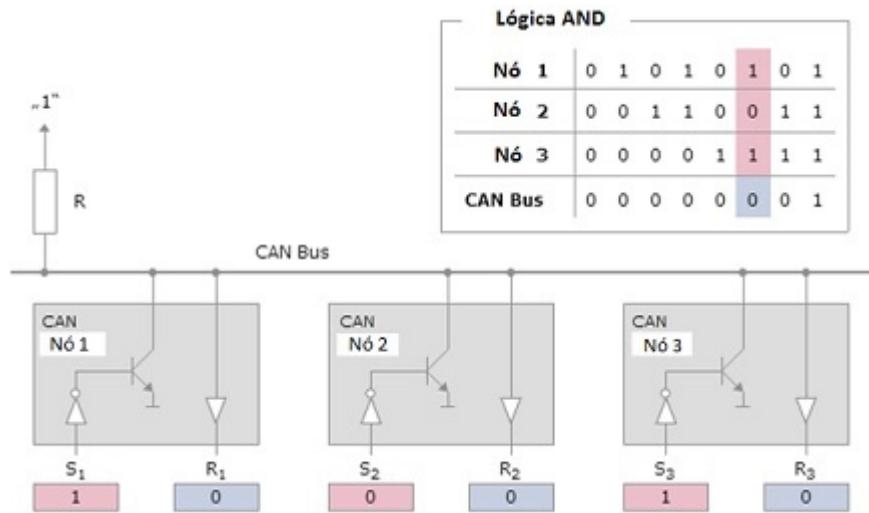
CIA-DS 102 (*Can in Automation*): não existe nenhuma norma para a subcamada MDI da camada física. CiA DS-102 recomenda apenas o uso de conectores muito específicos (SUB-D9), bem como a descrição da pinagem do conectores.

2.6.4 Camada Física

No padrão internacional ISO a camada física é definida como segue, (Institute of Industrial Automation and Software Engineering, 2015):

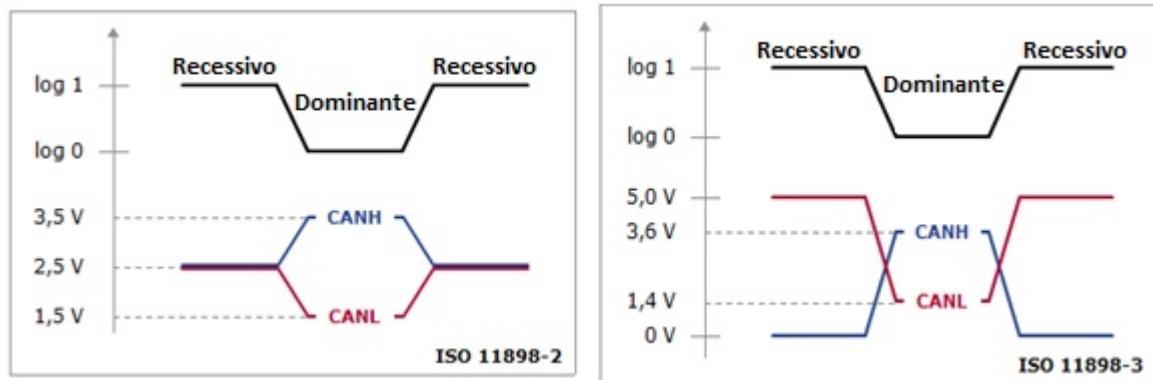
- Meio de transmissão: normalmente um par de fios trancados para rejeição a ruído, um do CAN *high* e outro do CAN *low*, formando uma diferença de potencial e com uma impedância de 120Ω se a rede for de alta velocidade.
- Topologia de barramento: topologia linear ou uma topologia física em estrela com resistores na extremidade de cada barramento.
- Número máximo de participantes (quantidade de nós): dependente do barramento. O comprimento máximo do barramento depende da taxa de transmissão dados. O número de nós que podem existir numa única rede é, teoricamente, ilimitado. No entanto, o número de identificadores e a capacidade dos transceptores existentes impõem restrições. Assim, dependendo do tipo de transceptor, até 32 nós por rede é normal, existindo, no entanto, transceptores que permitem ligar pelo menos 110 nós por rede.
- Codificação de bit: NRZ (*Non Return to Zero*), onde cada bit (0 ou 1) é transmitido por um valor de tensão específico e constante, diferenciando nível dominante (nível lógico zero) e recessivo (nível lógico um) com um mecanismo de fiação AND como mostrado na figura 11. De acordo com a ISO 11898-2 (CAN alta velocidade), atribui-se valor lógico "1" para uma tensão diferencial *diff* típica de 0 Volt ($-1 \text{ Volt} \leq \text{diff} \leq 0,5 \text{ Volts}$) e valor lógico "0" para uma tensão diferencial *diff* típica de 2 volts ($0,9 \text{ Volt} \leq \text{diff} \leq 5 \text{ Volts}$). Caso $0,5 \text{ Volts} < \text{diff} < 0,9 \text{ Volts}$, o valor lido no barramento será considerado como uma falha. Em contrapartida a ISO 11898-3 (CAN baixa velocidade) atribui uma tensão diferencial típica de 5 volts para a lógica "1", e uma típica tensão diferencial de 2 volts corresponde a lógica "0". A figura 12 retrata esses cenários.

Figura 11 – Lógica AND



Fonte: https://elearning.vector.com/index.php?seite=vl_can_introduction_en&root=378422&wbt_ls_kapitel_id=489560&wbt_ls_seite_id=489584&d=yes

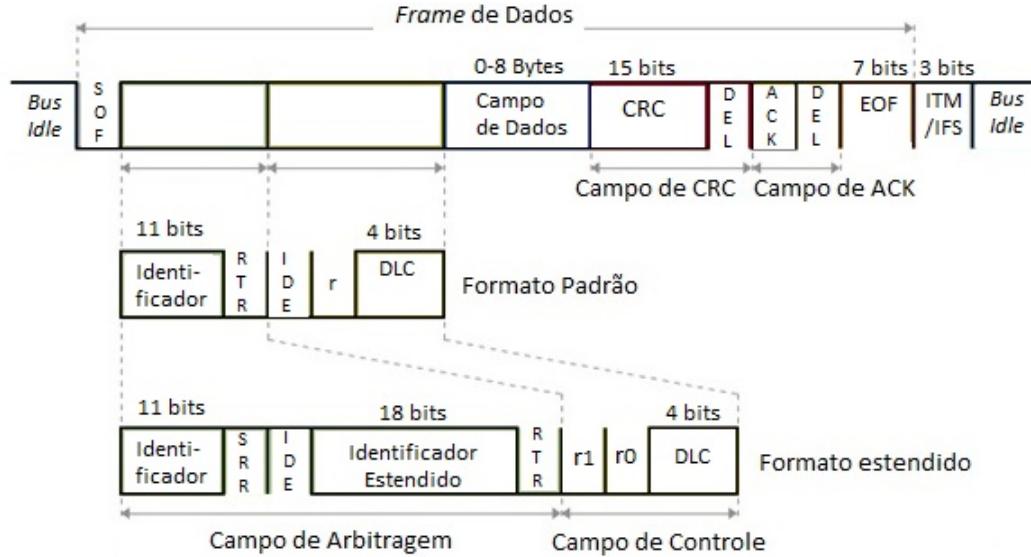
Figura 12 – Nível lógico



Fonte: https://elearning.vector.com/index.php?seite=vl_can_introduction_en&root=378422&wbt_ls_kapitel_id=489560&wbt_ls_seite_id=489580&d=yes

- Sincronização dos bits: pós-sincronização contínua do fluxo de bits transmitidos utilizando transições. Devido ao fato da NRZ-codificação ser utilizada, longos períodos sem transições podem ocorrer, o que deve ser gerenciado usando-se formatos de mensagens adequadas.
- Taxa de dados: para assegurar uma segura sincronização de bits o intervalo de tempo de bit tem que ser maior do que o dobro do máximo atraso do grupo. O que significa que quanto maior o barramento menor a taxa de transmissão de dados. A taxa máxima de transmissão especificada é de 1 Mbit/s, correspondendo este valor a sistemas com comprimento de barramento até 40 m.
-

Figura 13 – Frame de dados versão padrão e estendida



Fonte: Autora

2.6.5 Camada de Enlace

A principal tarefa da camada de enlace é transmitir e receber dados da rede. Assim todos os dados devem ser agrupados em unidades transportáveis (mensagens), o acesso ao barramento deve ser definido entre os múltiplos participantes prontos para enviar as mensagens, e a transmissão dos dados deve utilizar mecanismos de detecção e correção de erro.

CAN transmite mensagens usando quadros (*data frames*). Existem quatro tipos de *frames*, (TSOU, 2013):

1. *Frame* de Dados: é transmitida de um transmissor para receptores.
2. *Frame* Remota (RTR): um nó pode solicitar uma *frame* de dados com o mesmo identificador.
3. *Frame* de Erro: é transmitida por qualquer nó quando ele detecta um erro no barramento.
4. *Frame* de Sobrecarga (*overload*): é usada para dar atraso extra para *frame* de dados ou remota e a *frame* posterior.

A *frame* de dados no formato padrão e no formato estendido possuem a mesma estrutura, se diferenciando apenas nos campos de arbitragem e de controle conforme figura 13.

A *frame* remota é idêntica a *frame* de dados, com exceção da inexistência do campo de dados.

Segundo Tsou (2013), Institute of Industrial Automation and Software Engineering (2015), Santos (2002) e Vector Informatik GmbH (2014) os significados destes campos são:

SOF – O bit dominante *Start of Frame* (início de quadro) marca o início de uma mensagem e é usado para sincronizar todos os nós. Ele marca uma mudança de estado lógico do barramento de 1 para 0.

Identificador – O identificador no formato padrão possui 11 bits e no estendido 29 bits e estabelece a prioridade da mensagem em vez de definir o endereço do receptor. Quanto menor o valor binário, maior é a prioridade. O identificador também é utilizado pelo controlador CAN para filtrar quais mensagens devem ser processadas pela unidade de controle.

RTR – O bit *Remote Transmission Request* (requesicao de transmissao remota) é dominante nas *frame* de dados e recessivo nas *frames* remotas.

IDE – O bit *Extended Identifier* (identificador de extensão) dominante significa que o identificador no formato padrão está sendo transmitido (apenas 11 bits). Um bit recessivo indica que há mais 18 bits após o IDE no campo do identificador estendido, caracterizando o formado estendido de *frame*.

r, r0 e r1 – Bits reservados, em estado dominante.

DLC – Estes 4 bits (*Data Length Code*) informam quantos bytes de dados compõem o campo de dados da *frame*, que podem ser no máximo 8 bytes. É importante destacar que embora o campo de dados da *frame* remota seja inexistente, o seu valor DLC permanece igual ao da *frame* de dados respectiva.

Campo de dados – o campo de dados suporta até 64 bits (8 bytes).

CRC – O *Cyclic Redundancy Check* contém o *checksum* para detecção de erros.

Campo de CRC – Contém o CRC mais um bit delimitador que é sempre recessivo, indicando o fim do CRC.

ACK – Se o CRC calculado por um nó receptor for igual ao CRC recebido na *frame* de dados enviada por um nó transmissor, o bit *Acknowledgement* muda seu valor lógico de 1 para 0 no barramento, indicando que a mensagem foi recebida sem erro por pelo menos um dos nós receptores. Assim o valor lógico do ACK como recessivo no barramento indica erro no recebimento da mensagens por todos os nós receptores ou ainda que o nó de origem encontra-se sozinho no barramento.

Campo de ACK - Constituído pelo ACK mais um bit delimitador recessivo, o qual garante o tempo necessário para que todos os nós ouvintes da rede respondam com o bit ACK.

EOF – *End-of-Frame* são 7 bits recessivos indicando o final da *frame*.

ITM ou IFS – Estes 3 bits *Intermission* (intermissão) ou *Inter Frame Space* (espaço entre frames) contém o tempo necessário para o controlador mover corretamente os dados recebidos para um *buffer* de recepção de mensagem.

SRR – O bit *Substitute Remote Request* é transmitido em *frames* no formato estendido na posição que o bit RTR ocupa em *frames* no formato padrão e é recessivo.

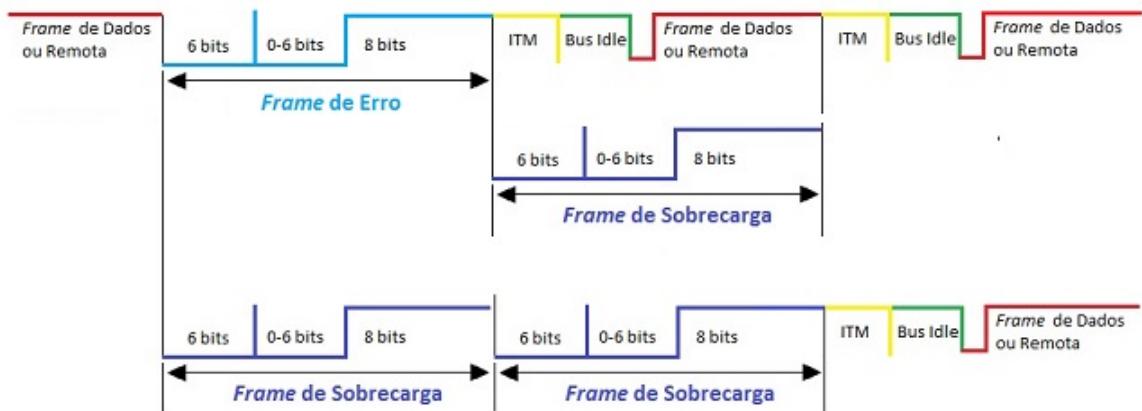
Bus Idle – Representa o estado em que o barramento encontra-se livre para o envio de mensagens, possuindo valor lógico recessivo.

Após 11 consecutivos recessivos bits (DEL + EOF + ITM/IFS ou ITM/IFS+ delimitador de erro/sobrecarga) o barramento é considerado livre para envio de novas mensagens, permanecendo em estado recessivo (*bus idle*) até nova transmissão de mensagem.

Os *layouts* de uma *frame* de erro e de sobrecarga são iguais, diferindo porém significativamente das *frames* de dados e remota. Elas são compostas por apenas duas partes: a *flag* de erro/sobrecarga e o delimitador de erro/sobrecarga (8 bits recessivos) conforme o caso, como mostrado na figura 14. Tanto a *flag* de erro quanto de sobrecarga podem ser divididas em duas partes, a *flag* de erro/sobrecarga primária (6 bits dominantes) e a *flag* de erro/sobrecarga secundária (0-6 bits dominantes), devido a sobreposição das *flags* de erro/sobrecarga dos vários nós receptores do barramento.

Uma *frame* de sobrecarga pode ser enviada após uma *frame* de dados ou remota ou depois de uma *frame* de erro ou de sobrecarga, como mostrado na figura 14. A *flag* de sobrecarga consiste em seis bits “dominantes”. Esta *flag* destrói o formato fixo do campo de Intermissão (ITM/IFS) da *frame* anterior, pelo que todos os outros nós também detectam uma condição de sobrecarga e iniciam a transmissão de uma *flag* de sobrecarga, ocorrendo assim uma sobreposição de *flags* (*flag* secundária), seguida do delimitador de sobrecarga.

Figura 14 – *Frame* de Erro e de Sobrecarga



Fonte: Autora

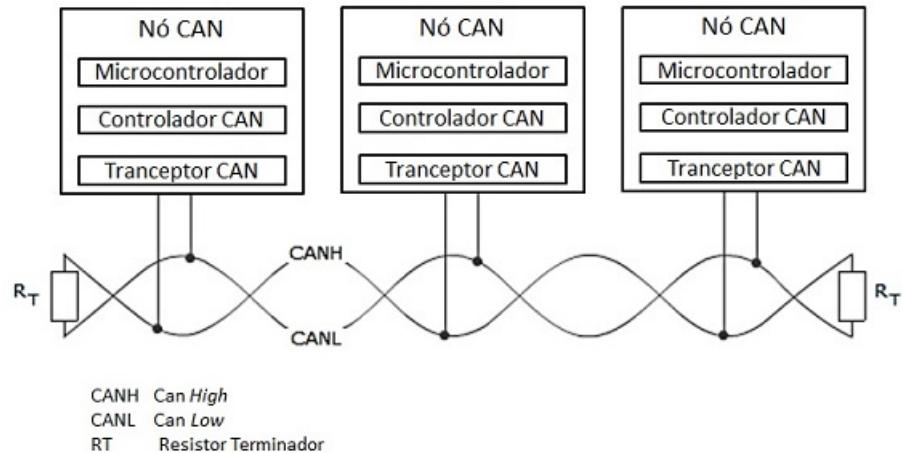
Uma *frame* de erro é transmitida quando o cheque de CRC falha por pelo menos um nó receptor ou quando um nó encontra-se sozinho. Após um nó iniciar a transmissão de uma

frame de erro os outros nós percebem a falha na transmissão da mensagem, pois a *frame* de erro viola o formato de uma *frame* de dados ou remota, assim todos os nós enviam em seguida uma *frame* de erro, acarretando na sobreposição de suas *flags* de erro. Além disso o nó emissor sempre compara o valor transmitido (T_x) com o valor no barramento (R_x), identificando dessa forma falhas de transmissão também.

Para a detecção de erros o protocolo CAN implementa seis mecanismos não mutuamente exclusivos identificados na figura 15, (Vector Informatik GmbH, 2014):

- **Monitoramento de bits pelo transmissor:** O transmissor compara em tempo real o valor transmitido (T_x) com o valor lido no barramento (R_x);
- **Confirmação do ACK bit pelo transmissor:** O transmissor aguarda o recebimento do bit ACK como dominante;
- **Verificação da regra de inserção do bit (*bit stuffing*) pelos receptores:** Após cinco bits iguais consecutivos no barramento, o próximo deve apresentar valor inverso;
- **Verificação do formato da *frame* pelo receptores:** Após o primeiro nó receptor iniciar o envio de uma *frame* de erro, o campo EOF (7 bits recessivos) será sobreescrito no barramento pela *flag* de erro primária do nó que identificou o erro (6 bits dominantes), caracterizando assim uma violação no formato da *frame* de dados ou remota;
- **Verificação da sequência CRC de 15 bits pelo receptores:** O CRC calculado por uma expressão polinomial com os dados obtidos pelo receptor desde o identificador da *frame* até o final do campo de dados deve ser igual ao CRC enviado pelo transmissor.
- **Monitoramento do ACK bit pelos receptores:** Se um dos nós enviar o bit ACK como recessivo (1) e outro nó enviar como dominante (0), o valor predominante do ACK no barramento permanecerá como dominante, porém após o bit delimitador do campo ACK, o receptor que não recebeu corretamente a mensagem iniciará o envio de uma *frame* de error.

Figura 16 – Rede CAN



Fonte: Autora

Figura 15 – Mecanismos de Detecção de Erros



Fonte: Autora

Para que haja uma comunicação entre os nós, sendo cada nó uma ECU, unidade eletrônica de controle (*Electronic Control Unit*), é necessário que cada um deles seja munido dos seguintes componentes: unidade de controle, controlador CAN e transceptor CAN.

Cada ECU leva dentro dela um processador (unidade de controle) que tem a função de receber os sinais dos sensores e processá-los. Caso seja uma informação importante para outras unidades, esta informação é transmitida até um controlador CAN que, por sua vez, tem a função de transformar essa informação em uma mensagem (*frame* de dados). Em seguida esta informação é transmitida para o transceptor, que tem como função codificar esta mensagem e transformá-las em pulsos elétricos (bits) que serão lançados ao barramento.

A figura 16 ilustra uma rede CAN com três nós, ou seja, três ECUs.

A *frame* de dados ou remota é transmitida a todos os nós e a todas as redes de interconexão, inclusive para as unidades que não necessitam do sinal em questão. A *Gateway* retransmite a *frame* para as outras redes de interconexão (redes CAN com

diferentes taxas de transmissão por exemplo).

Uma *frame* transmite uma específica mensagem que é caracterizada pelo seu identificador. Todo participante checa o identificador da mensagem recebida para analisar se esta é relevante ou não através do controlador CAN. O princípio de filtro de aceitação de mensagem também possibilita que mensagens sejam aceitas por todos ou por múltiplos nós ao mesmo tempo (*multicasting*), o que é importante para processos de sincronização.

Quanto menor o identificador da mensagem maior a sua prioridade. Esse mecanismo é utilizado para controlar o acesso ao barramento, o qual ocorre localmente de acordo com o método CSMA/CA with NDA (*Carrier Sense Multiple Access / Collision Detection with Non-Destructive Arbitration*). Isto significa que todos os módulos (nós) verificam o estado do barramento, analisando se outro módulo está ou não enviando mensagens com maior prioridade. Caso isto seja percebido, o módulo cuja mensagem tiver menor prioridade cessará sua transmissão e o de maior prioridade continuará enviando sua mensagem deste ponto, sem ter que reiniciá-la. Isto é feito comparando-se o valor lógico presente no barramento com o valor enviado por cada unidade, (GUIMARÃES, 2003).

Todas as ECUs que estiverem conectadas a este mesmo barramento receberão essa mensagem através do transceptor (através dos dois fios – CAN *high* e *low*), que no momento de recepção da mensagem tem o papel de decodificar a mensagem (de voltagem diferencial para bits) e repassá-la para o controlador CAN. Este por sua vez receberá a mensagem e filtrará a informação que está contida nela. Se a informação for útil e realmente importante para o processador daquela unidade, o controlador CAN irá repassar a informação para o processador que analisará a informação e tomará as devidas providências.

2.7 Rede DeviceNet

2.7.1 Conceito

A rede *DeviceNet* é uma rede de baixo nível que proporciona comunicações utilizando o mesmo meio físico entre equipamentos, desde os mais simples (como sensores e atuadores), até aos mais complexos, como Controladores Lógicos Programáveis (CLP) e microcomputadores. Ela possui o protocolo aberto, tendo um número expressivo de fornecedores de equipamentos que adotaram o protocolo.

Apresentado em 1994 originalmente pela *Allen-Bradley*, o *DeviceNet* teve sua tecnologia transferida para a ODVA em 1995. A ODVA (*Open DeviceNet Vendor Association*) é uma organização sem fins lucrativos composta por centenas de empresas ao redor do mundo que mantém, divulga e promove o *DeviceNet* e outras redes baseadas no protocolo CIP (*Common Industrial Protocol*).

A rede *DeviceNet* é classificada no nível de rede chamada *devicebus*, cuja características principais são: alta velocidade, comunicação a nível de byte englobando comunicação com

equipamentos discretos e analógicos e alto poder de diagnóstico dos dispositivos da rede, (CAETANO, 2009).

Essa tecnologia é um padrão aberto de automação com objetivo de transportar 2 tipos principais de informação:

- dados cíclicos de sensores e atuadores, diretamente relacionados ao controle e,
- dados acíclicos indiretamente relacionados ao controle, como configuração e diagnóstico.

Os dados cíclicos representam informações trocadas periodicamente entre o equipamento de campo e o controlador. Por outro lado, os acíclicos são informações trocadas eventualmente durante configuração ou diagnóstico do equipamento de campo.

2.7.2 Características

Segundo CAETANO (2009) e Rossit (2002) as características de uma rede DeviceNet são enumeradas:

- Topologia em barramento, linear ou árvore.
- Comunicação serial produtor-consumidor.
- Permite o uso de repetidores, *bridges*, roteadores e *gateways*.
- Suporta até 64 nós, incluindo o mestre, endereçados de 0 a 63 (MAC ID).
- Cabo com 2 pares: um para alimentação de 24V e outro para comunicação.
- Inserção e remoção à quente, sem perturbar a rede.
- Suporte para equipamentos alimentados pela rede em 24V ou com fonte própria.
- Uso de conectores abertos ou selados.
- Proteção contra inversão de ligações e curto-circuito.
- Alta capacidade de corrente na rede (até 16 A).
- Uso de fontes de alimentação de prateleira.
- Diversas fontes podem ser usadas na mesma rede atendendo às necessidades da aplicação em termos de carga e comprimento dos cabos.
- Taxa de comunicação selecionável : 125,250 e 500 kbps.
- Comunicação baseada em conexões de E/S e modelo de pergunta e resposta.

- Diagnóstico de cada equipamento e da rede.
- Transporte eficiente de dados de controle discretos e analógicos.
- Detecção de endereço duplicado na rede.
- Mecanismo de comunicação extremamente robusto a interferências eletromagnéticas.

2.7.3 A Arquitetura DeviceNet

A camada de enlace da rede *DeviceNet* segundo ODVA (2002) é baseada no protocolo CAN e a camada de aplicação no protocolo CIP, que define uma arquitetura baseada em objetos e conexões entre eles e é independente das outras duas camadas. Essas camadas são mostradas na Tabela 5.

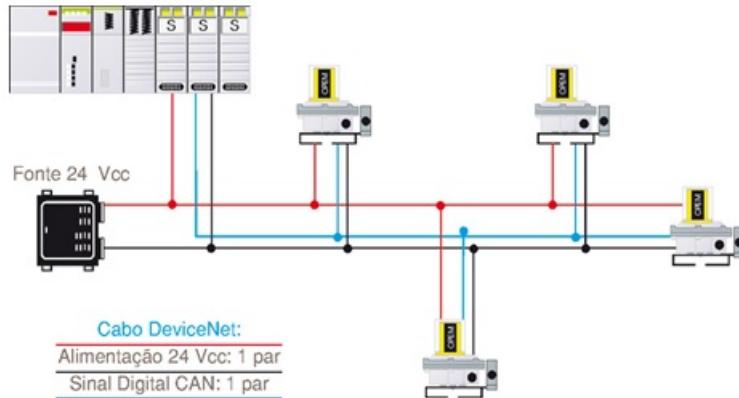
Tabela 5 – Arquitetura DeviceNet

Modelo OSI	Padrão DeviceNet	
Aplicação	CIP	Biblioteca de objetos de aplicação
Apresentação		Serviços de gerenciamento, mensagens de entrada e saída
Sessão		Roteamento de Mensagens, gerenciamento de conexão
Transporte		DeviceNet transporte
Rede		CAN CSMA/NBA
Enlace		DeviceNet camada física
Física		

Fonte: Autora

A rede *DeviceNet* identifica a camada física mas não usa a mesma interface de camada física como a ISO 11898, mas está baseado no protocolo de CAN Bus. *DeviceNet* provê isolamento óptico para proteção adicional e não usa conectores de 9- pinos tipo subD. Ela aborda três taxas de *bauds*: 125, 250 e 50048 Kbaud (500 metros) com até 64 dispositivos no barramento diferencial. Além disso o cabo leva 24 volts para alimentação nos dispositivos, como mostrado na figura 17.

Figura 17 – Meio físico do DeviceNet



Fonte: <http://www.mecatronicaatual.com.br/educacao/1138-rede-devicenet>

A rede pode ter 64 equipamentos ativos, que utilizam o barramento para se comunicar, endereçados de 0 a 63, significando 64 equipamentos com comunicação CAN ligados ao mesmo meio físico, onde sugere-se a utilização de no máximo 61 equipamentos, deixando os seguintes endereços livres ao se fazer um novo projeto:

- 0 para o *scanner*;
- 62 para a interface entre microcomputador e rede;
- 63 para novos equipamentos que venham a ser incluídos, considerando-se que segundo os padrões *DeviceNet*, os equipamentos novos saem de fábrica com o endereço 63.

A camada de aplicação é fundamentada no protocolo CIP, que define uma arquitetura baseada em objetos e conexões entre eles e é independente das outras camadas.

Segundo Smar (2012) o CIP tem dois objetivos principais:

- Transporte de dados de controle dos dispositivos de I/O.
- Transporte de informações de configuração e diagnóstico do sistema sendo controlado.

Um nó *DeviceNet* é então modelado por um conjunto de objetos CIP, os quais encapsulam dados e serviços e determinam assim seu comportamento.

2.8 Suite de Protocolos TCP/IP (Transmission Control Protocol / Internet Protocol)

O TCP/IP (Protocolo de Controle de Transmissão/Protocolo Internet) foi desenvolvido em 1969 pelo U.S. *Department of Defense Advanced Research Projects Agency*, como um recurso para um projeto experimental chamado de ARPANET (*Advanced Research Project Agency Network*) para preencher a necessidade de comunicação entre uma grande

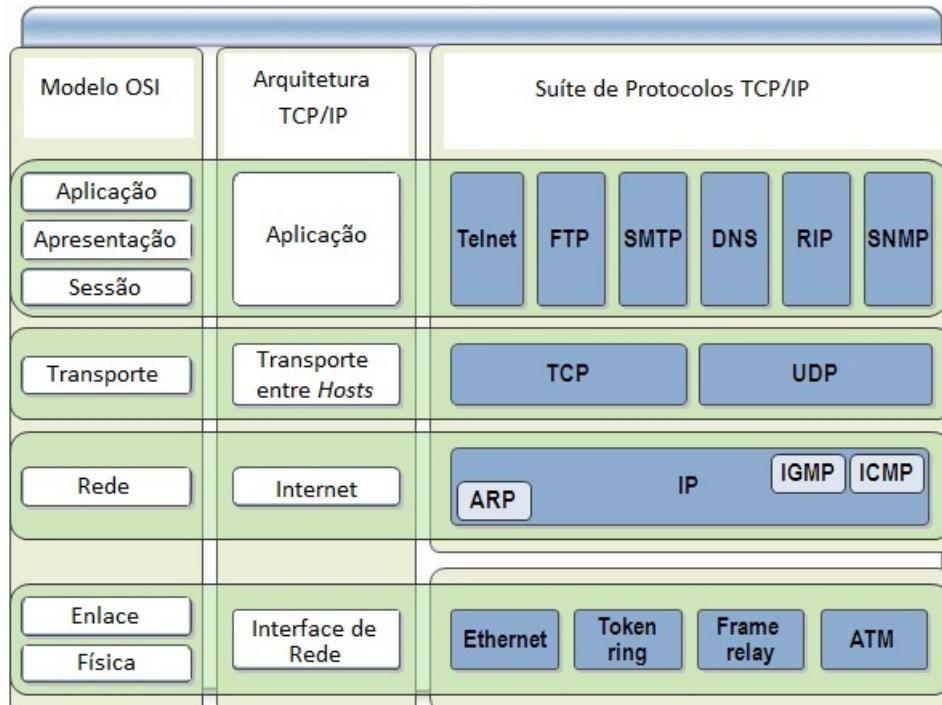
quantidade de sistemas de computadores e várias organizações militares dispersas. O objetivo do projeto era disponibilizar *links* (vínculos) de comunicação com altas velocidades utilizando redes de comutação de pacotes, (LUIZ, 2007).

O TCP/IP não é apenas um protocolo, mas uma suíte ou grupo de protocolos que se tornou padrão na indústria por oferecer comunicação em ambientes heterogêneos, tais como sistemas operacionais UNIX, Windows, MAC OS, minicomputadores e até mainframes, (LUIZ, 2007).

Hoje o TCP/IP se refere a uma suíte de protocolos utilizados na Internet, a rede das redes. Este conjunto padrão de protocolos especifica como computadores se comunicam e fornece as convenções para a conexão e rota no tráfego da Internet através de conexões estabelecidas por roteadores, (LUIZ, 2007).

Os protocolos TCP/IP são organizados em quatro camadas: camada de interface de rede, internet, transporte e aplicação. Essas camadas e sua relação com o modelo OSI podem ser comparadas na figura 18.

Figura 18 – Suíte de Protocolos TCP/IP



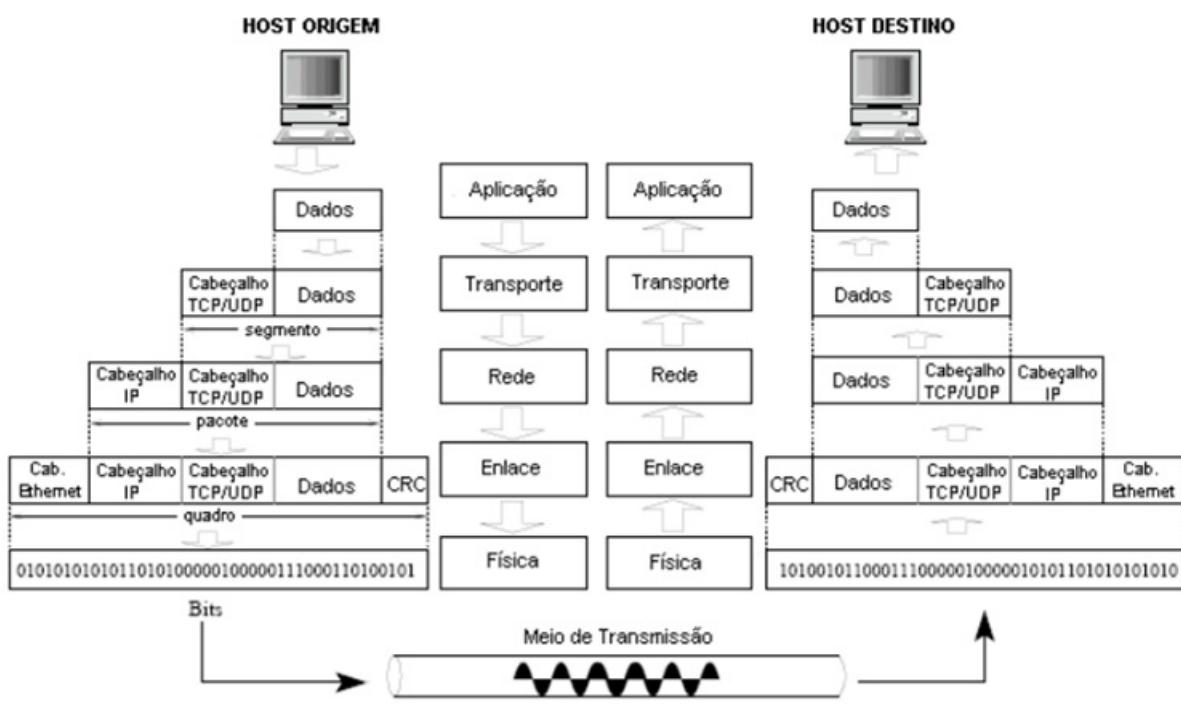
Fonte: <https://tech4it.files.wordpress.com/2010/05/screenshot110.gif>

O termo *host* é utilizado para se referir a qualquer parte de hardware que pode ser endereçada. Isto inclui estações de trabalho e servidores, como também roteadores. *Hosts* são identificados pelo endereço físico de suas placas de rede (MAC address).

No *host* de origem, os dados a serem transmitidos são gerados pela camada de aplicação e passados para as camadas inferiores, até chegar ao meio de transmissão. Os dados são transmitidos em forma de PDUs (*Protocol Data Unit*, “protocolo de unidade de dados”) através das camadas. Cada camada define seu próprio PDU, nominalmente, Segmento

(*Segment*) para a camada de transporte, Pacote (ou datagrama, *packet*) para a camada de rede e Quadro (*frame*) para a camada de enlace. Os PDUs contém os dados recebidos da camada acima, juntamente com as informações de controle do protocolo em uso (cabeçalho). O processo de inserção de cabeçalho é denominado de encapsulamento e o de retirada de desencapsulamento. Assim durante uma transmissão, os dados são encapsulados de acordo com o nível de camada correspondente no *host* de origem até atingirem o nível físico e serem transmitidos ao *host* de destino que irá desencapsular estes dados até chegar ao nível de aplicação, como mostrado na figura 19, (HAMERSKI, 2008).

Figura 19 – Fluxo de Dados no Modelo TCP/IP



Fonte:(HAMERSKI, 2008)

No TCP/IP, além do endereço físico existem outros dois níveis de endereçamento: Nomes de Domínios e Endereços de IP. Os nomes de domínio são utilizados em ambiente TCP/IP através de um serviço denominado *DNS Domain Name Server* (Servidor de Nome de Domínio). O DNS oferece um esquema de nomes hierárquico para os hosts TCP/IP. Esse esquema permite às organizações dividirem logicamente as suas redes e delegar autoridade aos administradores de rede em cada uma das áreas, (LUIZ, 2007).

2.8.1 Camada de Interface de Rede

A camada de interface de rede é a camada de mais baixo nível dentro do modelo. Ela é responsável por colocar e retirar quadros (*frames*, pacotes) no meio físico.

Ethernet é um padrão de rede que compõem a suíte TCP/IP e que define as camadas física e de enlace do modelo OSI, conforme Tabela 6.

Tabela 6 – Arquitetura Ethernet

Modelo OSI		Padrão Ethernet	Implementação
2	Enlace	Controle do Link Lógico (LLC) Controle de Acesso ao Meio (MAC)	IEEE 802.2 IEEE 802.3
	1 Física	Meio de transmissão e conectores	10Base2, 10BaseT, 10Base5, 100BaseT4, 100BaseTX

Fonte: Autora

Uma ethernet de rede de área local (LAN) é essencialmente uma de rede de difusão (*broadcasting* ou *multicasting* com o uso de *switches*) com lógica de barramento; embora na implementação física possa utilizar *hubs* ou *switches* (topologia física de estrela), (W&T, 1999).

O mecanismo utilizado para controlar o acesso ao meio padronizado pela IEEE 802.3 é o CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*) por meio da codificação Manchester. Assim quando mais de uma estação está pronta para emitir mensagens com o meio livre, elas emitem os respectivos frames (quadros), os quais acabam colidindo. A primeira estação que detectar a colisão interrompe imediatamente sua transmissão, reiniciando o processo de transmissão depois de um período de tempo, de forma a tornar improvável que uma nova colisão aconteça, (PINHEIRO, 2010).

O endereço de ethernet (MAC *address*) é gravado no adaptador de ethernet (cartão de rede, roteador, etc) pelo fabricante, sendo fixo e não podendo ser mudado. É composto por 6 bytes representados normalmente na forma hexadecimal, sendo os 3 primeiros bytes usados para identificação do fabricante e os próximos 3 para o número sequencial da placa em questão, por exemplo: 00-C0-3D-00-27-8B, (W&T, 1999).

O frame (quadro) de ethernet é dividido em campos como mostrado na figura 20.

Figura 20 – Frame de Ethernet IEEE 802.3

Tamanho do campo em bytes							
7	1	6	6	2	46-1500	4	
Preâmbulo	S O F	Destino Endereço	Origem Endereço	Tamanho	Cabeçalho e dados 802.2	FCS	

Fonte: <http://tiigr1214nunomartins.blogspot.com.br/p/quando-o-router-inicia.html>

Os principais campos podem ser descritos da seguinte maneira:

Preâmbulo (Preamble): Sequência alternada de 1 e 0 informando às estações receptoras que um frame está começando.

SOF (Start-of-Frame): Campo de um byte que termina com 2 bits “1” consecutivos que servem para sincronizar a parte de recepção de frame de todas as estações da LAN.

Endereço de destino: Contém o endereço MAC do destinatário;

Endereço de origem: Contém o endereço MAC do remetente;

Tamanho: indica o tamanho em Bytes do campo de dados;

Cabeçalho e dados: É importante notar que o *frame* IEEE 802.3 não seria capaz de identificar o protocolo da camada superior (de rede). Por esse motivo, o *frame* IEEE 802.3 encapsula um *frame* 802.2 LLC em seu campo de dados, que contém os cabeçalhos das camadas superiores), além disso esse campo também carrega os dados (mensagem) a ser transportada e ele deve ter tamanho mínimo de 46 bytes e máximo de 1500 bytes. Se o dado no *frame* for insuficiente para preencher o mínimo de 64bytes (somados do endereço de destino até o campo FCS), bytes de preenchimento são inseridos para garantir este número mínimo de bytes;

FCS (Frame Check Sequence): Contém o CRC com funcionamento análogo ao CRC do protocolo CAN.

2.8.2 Camada de Internet

A camada de Internet é responsável pelas funções de endereçamento, empacotamento e roteamento. São definidos três protocolos principais nesta camada: IP (*Internet Protocol*), ARP (*Address Resolution Protocol*) e o ICMP (*Internet Control Message Protocol*).

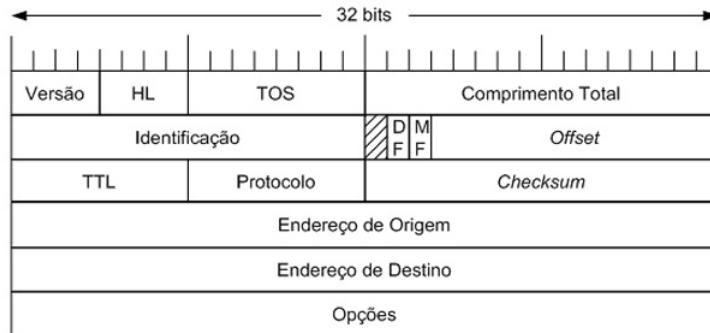
2.8.2.1 IP (*Internet Protocol*)

O IP é responsável pelo endereçamento de informação dos *host* de origem e destino e possível roteamento entre as respectivas redes, se diferentes. Este roteamento é executado através de endereços de IP, os quais são compostos de 4 octetos, que identificam logicamente origem e destino da informação, constituído por duas partes: identificador de rede ou net id (*network identification*, endereço de rede) e o identificador de nó ou *host* id (*host identification*, endereço de nó), (HAMERSKI, 2008).

Este protocolo, usando a parte identificador de rede do endereço, normalmente designada pelos seus primeiros bytes, pode definir a melhor rota através de uma tabela de roteamento mantida e atualizada pelos roteadores.

Este protocolo recebe os dados da camada de transporte na forma de segmentos. Ocorre então o processo de fragmentação e os conjuntos de dados passam a ser chamar datagramas ou pacotes. Um datagrama IP pode ser dividido em: cabeçalho IP e dados. Os dados (tamanho livre, definido pelo tipo de rede) contém o dado que está sendo comunicado em si vem logo após o cabeçalho. O cabeçalho IP possui os campos retratados na figura 21.

Figura 21 – Cabecalho do Datagrama/Pacote IPV4



Fonte: <http://www.techtudo.com.br/artigos/noticia/2012/05/o-que-e-ip.html>

Versão (4 bits): Indica a versão do protocolo sendo utilizada, o que determina o formato do cabeçalho.

Comprimento do cabeçalho (4 bits): HL (*header lenght*) comprimento do cabeçalho, medido em palavras de 32 bits.

Tipo de serviço (8 bits): TOS (*type of service*) informa ao roteador como o datagrama deve ser tratado e identifica algumas funções de qualidade de serviço, como prioridade, atraso, vazão e confiabilidade.

Comprimento total (16 bits): Este campo proporciona o comprimento do datagrama medido em bytes, incluindo cabeçalho e dados.;

Identificação (16 bits): O campo de identificação contém um único inteiro que identifica o datagrama, é um campo muito importante porque quando uma *gateway* fragmenta um datagrama, ela copia a maioria dos campos do cabeçalho do datagrama em cada fragmento, então a identificação também deve ser copiada, com o propósito de que o destino saiba quais fragmentos pertencem a quais datagramas.

Flag (3 bits): Controla a fragmentação, constituído por um bit reservado valor lógico zero, um bit DF (*don't fragment*) que determina se o pacote pode ser fragmentado (=0), ou não (=1) e um bit MF (*more fragments*) que indica se o fragmento é o último (=0) de um pacote ou se existem mais (=1).

(Fragment) offset (13 bits): Especifica o início do datagrama original dos dados que estão sendo transportados no fragmento. É medido em unidades de 8 bytes.

Time to Live (8 bits): TTL informa o número de roteadores que podem redirecionar o datagrama. O valor é decrementado até zero a cada roteador, quando então é descartado;

Protocolo (8bits): diz respeito a qual protocolo deverá receber o datagrama na próxima camada, identificando qual protocolo que está ocupando a área de dados. Exemplos: ICMP=1, TCP=6, UDP=17;

CRC do cabeçalho (16 bits): O *checksum* seleciona qual processo será utilizado na detecção de erros (verificação de redundância cíclica ou checagem na sequencia de quadros). Ele assegura a integridade do cabeçalho IP (apenas). É gerado na origem e conferido a cada roteador. Após decrementar o valor do TTL o roteador gera um novo valor e reenvia o pacote.

Opções (variável de 0 a 320 bits): É um campo opcional, se existir contém informações adicionais relacionadas a roteamento, rotas, horário ou segurança.

Endereços de origem e destino (ambos com 32bits): É o endereço de IP que caracteriza por completo toda informação sobre endereçamento , necessária ao processo de roteamento.

Na passagem da camada de internet para a camada de interface de rede o pacote/datagrama IP é dividido em novos pacotes IP, fragmentos do original, ocorrendo novamente a fragmentação pelos roteadores, aonde cada novo pacote IP cabe em um único quadro/*frame* de ethernet. No destino final o processo é revertido, ocorrendo a remontagem do datagrama original.

2.8.2.2 ARP (Address Resolution Protocol)

ARP é responsável por mapear um endereço IP para um endereço tipo MAC, através de uma tabela que associa a localização lógica de uma determinada máquina com a sua estrutura física de comunicação, (HAMERSKI, 2008).

2.8.2.3 ICMP (Internet Control Messsage Protocol)

ICMP é responsável por enviar mensagens e relatar erros relacionados a entrega de um pacote, (HAMERSKI, 2008).

2.8.3 Camada de Transporte

O TCP é orientado à conexão, ou seja, estabelece um caminho negociado entre os *hosts* de origem e destino antes de enviar os dados, possuindo mecanismos de confirmação de envio e recebimento de informações.

O protocolo fragmenta os dados da camada de aplicação por meio dos roteadores em segmentos e os transmite para a camada de internet. No *host* destino, o protocolo TCP

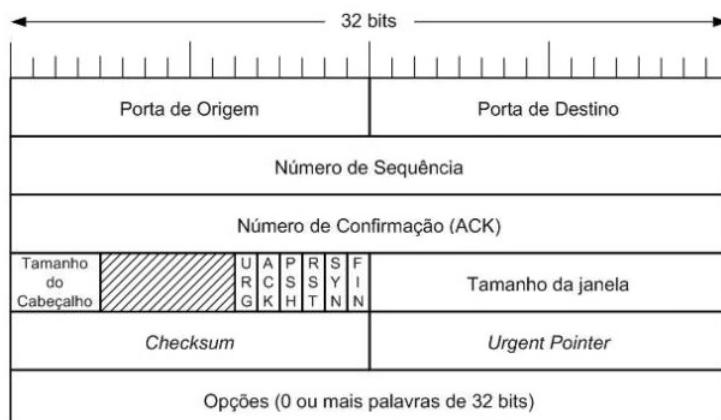
remonta os segmentos. O TCP cuida também do controle de fluxo, impedindo que um transmissor rápido sobrecarregue um receptor lento com um volume de mensagens muito grande, (HAMERSKI, 2008).

Todas as conexões TCP são *full-duplex* e ponto a ponto. *Full-duplex* quer dizer que o tráfego pode ser feito em ambas as direções ao mesmo tempo. Ponto a ponto quer dizer que cada conexão possui exatamente dois pontos terminais.

O serviço TCP é obtido quando tanto o transmissor quanto o receptor criam pontos terminais, denominados *sockets*. Cada *socket* tem um endereço que consiste no endereço IP do *host* mais um número de 16 bits local para esse *host*, chamado porta. Porta é o nome usado pelo TCP para um TSAP (*transport service access point*). Para que uma conexão funcione, é necessário que uma conexão seja explicitamente estabelecida entre uma *socket* da máquina transmissora e uma *socket* da máquina receptora.

Assim como o IP, o TCP é dividido em dados e cabeçalho. A estrutura do cabeçalho TCP é mostrada na figura 22.

Figura 22 – Cabecalho do Segmento TCP



Fonte: <https://rapazes.wordpress.com/2010/05/27/camada-de-transporte-tcpip/>

Porta de origem e destino (16 bits): usa um par de 16-bit para endereçamento de portas de origem e destino.

Número de sequencia: Os próximos 32 bits representam o número de sequência, que indica o segmento na rede e também é utilizado para reordenar o segmento na chegada ao destino.

Número de confirmação: O campo número de confirmação (ACK) é sempre um valor acima do número de sequência do segmento TCP recebido.

Tamanho do cabeçalho: Indica o número de quatro octetos presentes dentro do cabeçalho TCP.

URG: Se esta flag estiver em 1 o pacote deve ser tratado urgentemente.

ACK: Se esta flag estiver em 1 o pacote é um aviso de recepção.

PSH (push): Se esta flag estiver em 1, indica que o receptor dos dados deve entregar os dados à aplicação mediante sua chegada, em vez de armazená-los até que um *buffer* completo tenha sido recebido;.

RST: Se esta flag estiver em 1, a conexão é reiniciada.

SYN: É usada para sincronizar o início de uma conexão.

FIN: Se esta flag estiver em 1, a conexão é interrompida.

Tamanho da janela: É um contador de 16 bits que é usado para indicar o número de pacotes que o receptor pode receber em seu *buffer* em um determinado período de tempo.

Checksum: É usado para assegurar a integridade do segmento.

Urgent Pointer: Indica o número de sequência do octeto final quando a flag “*urgent*” está setada.

Opções (tamanho variável): Opções diversas.

O protocolo TCP, em um nível superior, é responsável por direcionar os dados para a aplicação correta. Isso é feito através dos campos portas origem e destino no cabeçalho TCP. O valor presente nesse campo é relacionado dinamicamente pelas aplicações. A exceção são as 1024 primeiras portas, previamente definidas, que só podem ser usadas por aplicações específicas.

2.9 Interfaces para Programas de Aplicação (APIs)

APIs (*Application Program Interfaces*) são interfaces para programadores usando diretamente os serviços da camada de transporte. Seu objetivo é facilitar a implementação de programas de aplicação que fazem o uso de serviços de comunicação da rede. APIs são usualmente oferecidas para alguns sistemas operacionais (por exemplo, *Unix*, *Linux*, *Windows*, etc) e em algumas linguagens (por exemplo, C, C++, C#, Java, etc), (STEMMER, 2001).

Uma das APIs bem difundida é a *Berkeley Socket*, API em linguagem C com TCP em UNIX BSD (*UNIX Berkeley Software Distribution*) e compatíveis. Os serviços são acessíveis por meio das chamadas de sistema (*system calls*) como nos exemplos da Tabela 7.

A interface entre um programa de aplicação e os protocolos de comunicação em um sistema operacional é conhecida como API. *Sockets* proveem uma implementação da

abstração SAP (*service access point*) na camada de transporte na suíte TCP/IP, a qual faz parte do BSD Unix.

Tabela 7 – Chamadas de sistema da Berkeley Socket

Chamada de Sistema	Função
socket	Cria uma <i>socket</i> (TSAP)
bind	Associa um nome ASCII a um <i>socket</i> já existente
listen	Cria uma fila de espera para requisições de serviço
accept	Remove um pedido de conexão de uma fila ou espera por um novo pedido
connect	Estabelece uma conexão com outra <i>socket</i>
shutdown	Encerra uma conexão entre <i>sockets</i>
send	Envia mensagem por conexão existente
recv	Recebe mensagens por conexão existente
select	Testa um grupo de <i>sockets</i> para ver se tem mensagens

Fonte: (STEMMER, 2001)

A *Socketcan* prove uma interface de *socket* para o espaço de aplicações do usuário construída baseada na camada de rede do Linux.

3 MATERIAIS E MÉTODOS

Neste capítulo serão abordados, de maneira detalhada, cada etapa da metodologia utilizada neste projeto, e quais foram os materiais utilizados em cada etapa.

3.1 Modelo do Sistema

3.1.1 Especificação dos Requerimentos do Sistema

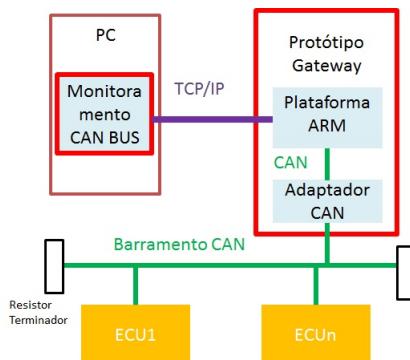
Os requerimentos do sistema são o desdobramento dos objetivos do trabalho:

- A implementação do *gateway* TCP/IP-CAN por meio do protótipo deve ser feita em uma plataforma com arquitetura ARM de baixo custo;
- Caso a plataforma ARM escolhida venha desprovida de controlador e transceptor CAN, um adaptador CAN deverá ser construído ou comprado;
- O protótipo deve ser capaz de receber todas as mensagens CAN do barramento da rede e retransmí-las via TCP/IP para um computador pessoal por uma conexão Ethernet;
- O protótipo deve ser capaz de receber comandos do sistema de monitoramento de barramento CAN desenvolvido em um computador pessoal e a partir desses comandos, transmitir e filtrar mensagens;
- O sistema de monitoramento funcionará como cliente e o protótipo como servidor da rede Ethernet TCP/IP.
- O sistema de monitoramento deve ser capaz de receber todos os dados do servidor e exibi-los ao usuário.
- O protótipo funcionará como uma ECU na rede CAN;
- Uma rede CAN deve ser montada para testes, com no mínimo duas ECUs, considerando-se o protótipo;
- A integração e validação do sistema serão alcançadas com a inserção do protótipo na rede CAN montada e monitoramento em tempo real do barramento CAN com o sistema de monitoramento desenvolvido.

3.1.2 Modelo de Hardware

Baseado nos requerimentos do sistema, o modelo de hardware do sistema foi projetado como na figura 23, sendo constituído por uma unidade de software denominada “Monitoramento CAN BUS” - representando o sistema de monitoramento de barramento CAN - em um PC e o protótipo, formado por uma unidade de software (*gateway*) e hardware constituída por uma plataforma com processador ARM e uma unidade de hardware adicional para fazer o interfaceamento com o barramento CAN.

Figura 23 – Modelo de Hardware

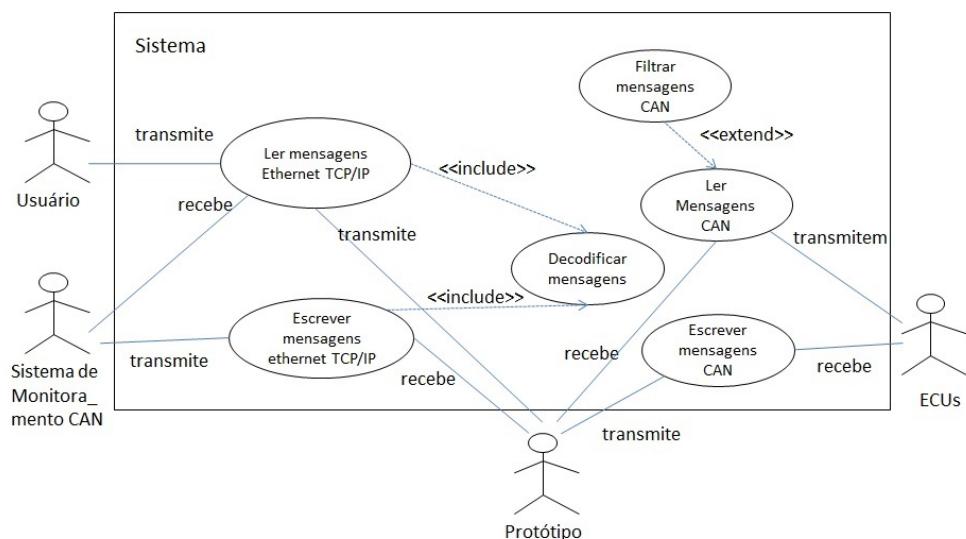


Fonte: Autora

3.1.3 Modelo de Software

Baseado nos requerimentos do sistema, o diagrama de caso da figura 24 ilustra a relação entre o usuário, o sistema de monitoramento de barramento CAN, o protótipo e outras ECUs conectadas à rede CAN de testes, além de representar os objetivos de cada um dentro do sistema, configurando-se assim como modelo de software para o sistema.

Figura 24 – Modelo de Software



Fonte: Autora

3.2 Arquitetura do Sistema

3.2.1 Comparativo entre as Arquiteturas Raspberry Pi e Beaglebone Black

Foi realizada uma comparação como mostrado na tabela 8, entre as arquiteturas *Raspberry PI B+* e *Beaglebone Black*. Ambos são computadores de baixo custo baseado em *Linux* e na arquitetura ARM do tamanho de um cartão de crédito. O *Raspberry PI B+* é desenvolvido pela fundação de caridade *Raspberry Pi* no Reino Unido, cujo principal objetivo é promover o ensino em ciência da computação básica em escolas enquanto que o *Beaglebone Black* é *open hardware*, sendo suportada por uma comunidade de desenvolvedores e produzido por terceiros, (RASPBERRYPI, 2015), (BEAGLEBOARD, 2015).

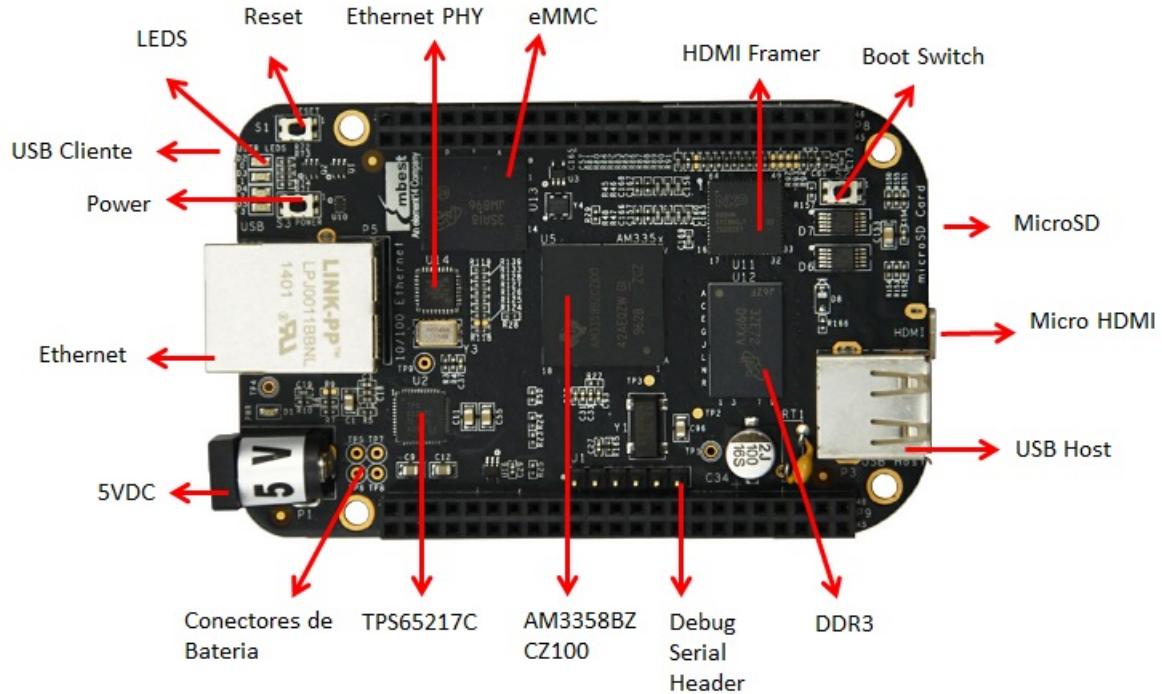
Tabela 8 – Raspeberry PI B+ VS Beaglebone Black

	Raspberry PI B+	Beaglebone Black
Número de GPIOs	40 máximo	69 máximo
Serial I/O	UART, SPI, I2C, USB	CAN, I2C, SPI, UART, USB
Processador	<i>Broadcom BCM2835</i> , 700 MHz	<i>Sitara AM3358BZCZ100</i> , 1GHz
Memória SDRAM	512MB , 400MHz	512MB DDR3L, 800MHZ
Memória Flash	Micro SD slot	4GB, 8bit Embedded MMC / Micro SD slot
Portas USB 2.0	4	1
Gráfico	HDMI, <i>Dual Core VideoCore IV® Multimedia Co-Processor</i>	16b HDMI, 1280x1024 (MAX) SGX530 3D
Arquitetura ARM	ARM1176JZFS (ARMv6 instruction set)	Cortex A8 (ARMv7 instruction set)
Preço	\$32.65 (14/10/2015, Amazon)	\$55.00 (14/10/2015, Amazon)
Ethernet	10/100Mbps RJ45	10/100Mbps RJ45
Tipo de Hardware	<i>Closed Hardware</i>	<i>Open Hardware</i>

Fonte: Autora

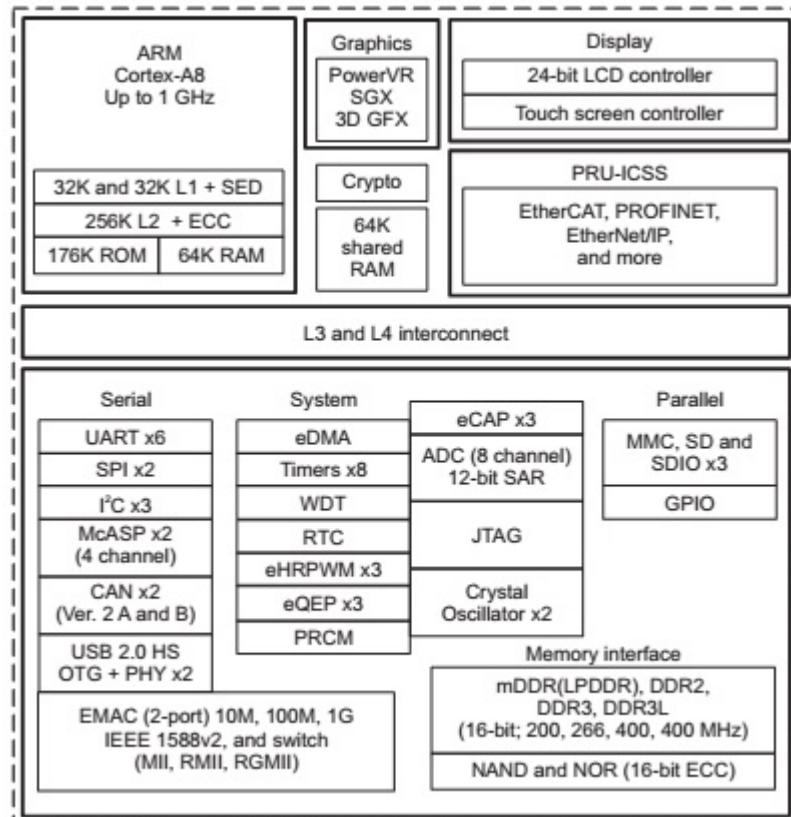
Após comparação entre essas plataformas, optou-se pelo *BeagleBone Black rev C - Element14 Commercial Edition* (ELEMENT14, 2015), para o para desenvolvimento do protótipo, figura 25, que apesar de ter um preço um pouco mais alto, apresenta maior velocidade de processamento, arquitetura ARM Cortex mais recente, maior número de GPIOs, e já apresenta incorporado no SoC (*System on a chip*) *Sitara AM3358BZCZ100* o controlador CAN, figura 26.

Figura 25 – Beaglebone Black



Fonte: Autora

Figura 26 – Diagrama de Blocos do AM335x

Fonte: <http://www.ti.com/lit/ds/symlink/am3358.pdf>

As tabelas 9 e 10 apresentam os principais adaptadores CAN para o *Raspberry PI B+* e *Beaglebone Black* atualmente existentes no mercado. Importante salientar que não foi encontrado nenhum fornecedor nacional.

Tabela 9 – Adaptadores para Barramento CAN Beaglebone Black

Beaglebone Black	Preço (05/11/2015)	Referências
GNUBLIN Module-CAN (Alemanha)	9,95 € + 19,95 € + correio (só entrega na Europa)	http://shop.embedded-projects.net/embedded-projects-gmbh/gnublin-module-gnubo-fr-beagle-bone.html
		http://shop.embedded-projects.net/embedded-projects-gmbh/gnublin-module-can-3.3v-transceiver.html
TT3201 CAN Cape (Itália)	87,00 € + correio	http://www.towertech.it/en/products/hardware/tt3201-can-cape/
BeagleBone Serial Cape	\$54.90 (correio incluso)	http://www.logicsupply.com/cbb-serial/

Fonte: Autora

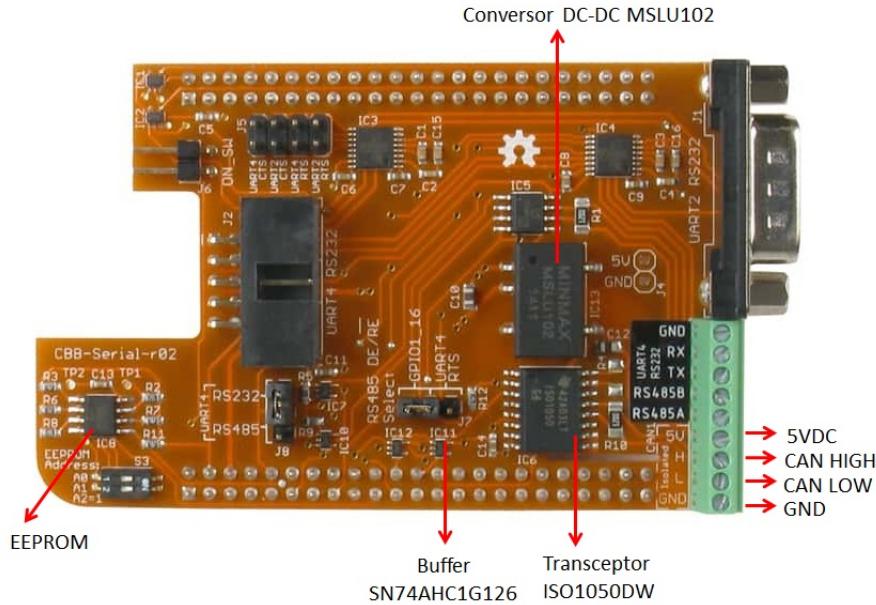
Tabela 10 – Adaptadores para Barramento CAN Raspberry PI B+

Raspberry PI B+	Preço (05/11/2015)	Referências
PICAN skpang (Reino Unido)	£32.28 Inc VAT £26.90 Ex VAT + £8.25 (correio)	http://skpang.co.uk/catalog/pican-canbus-board-for-raspberry-pi-p-1196.html
Canberry V 2.1 (Itália)	45€ + correio	http://www.industrialberry.com/canberry-v-2-1-isolated/
CAN Bus Shield for Raspberry Pi	75.00 € +7.00 € (correio)	https://www.cooking-hacks.com/can-bus-shield-for-raspberry-pi

Fonte: Autora

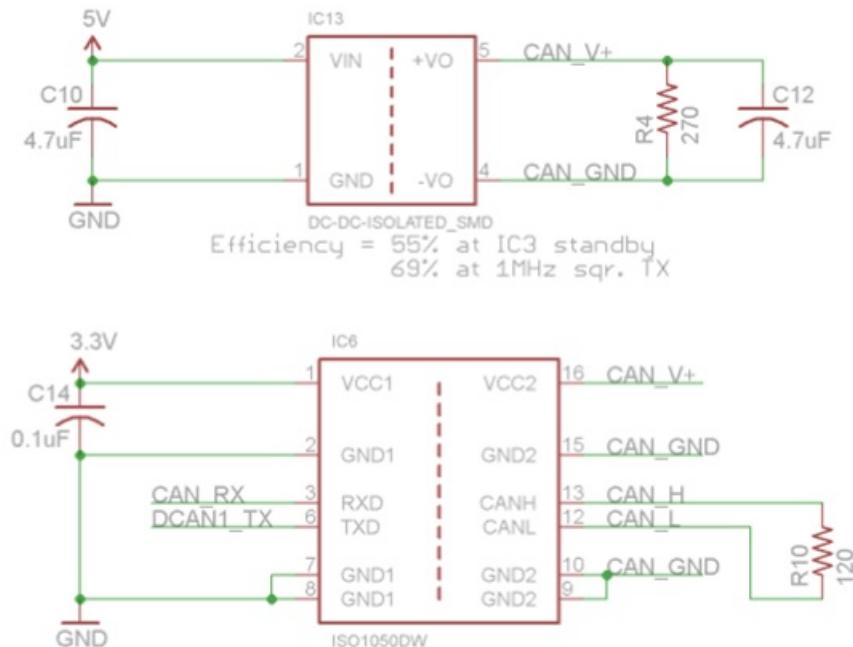
Entre os adaptadores CAN para o *Beaglebone Black*, optou-se pelo *BeagleBone Serial CAN RS485 RS232 Cape (BEAGLEBONE...)*, figura 27, que apesar de mais caro que o *GNUBLIN Module-CAN*, permite simultaneamente o uso de até duas portas RS232 e interface para o barramento CAN ou a operação simultânea de uma porta RS232, uma interface RS485 e uma para o barramento CAN, enquanto que o *GNUBLIN Module-CAN* garante interface apenas para o barramento CAN além de fazer entrega apenas na Europa, tendo em vista a futura expansão do protótipo para interfaceamento com outras redes no laboratório de automação da EST.

Figura 27 – BeagleBone Serial Cape



Fonte: <http://www.logicsupply.com/cbb-serial/>

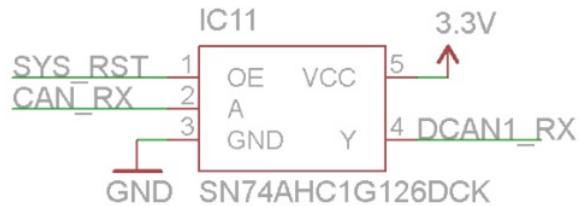
O *BeagleBone Serial Cape* é *open hardware* e utiliza o transceptor galvânico isolado ISO1050DW (TEXAS INSTRUMENTS, 2015), que atende às especificações da ISO11898-2, com conexões para uma fonte de 5V DC, GND, CAN HIGH e LOW. O resistor R4 garante mínimo carregamento exigido pelo conversor DC-DC MSLU102 (MINMAX, 2013), figura 28.

Figura 28 – Circuito CAN do *BeagleBone Cape*

Fonte: <https://github.com/lgxlogic/CBB-Serial>

Ele também possui uma memória EEPROM-I2CSMD para carregar o *Beaglebone Black* com a configuração apropriada e um *buffer*, SN74AHC1G126, para a recepção de dados CAN .

Figura 29 – Buffer de Dados CAN

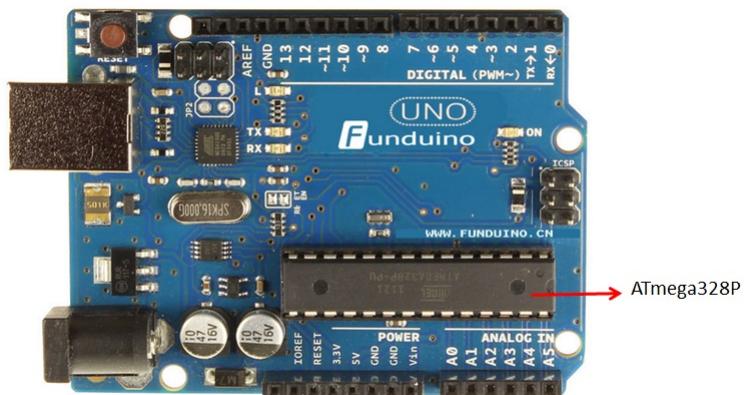


Fonte: <https://github.com/lgxlogic/CBB-Serial>

3.2.2 Unidade Eletrônica de Controle Complementar

Para montagem da rede CAN de teste foi escolhido o *Funduino UNO* como unidade microcontroladora, o qual é uma placa compatível com o Arduino UNO, devido a facilidade de operação e aquisição e grande quantidade e variedade de fontes de consulta. O Arduino UNO é uma plataforma de prototipagem eletrônica de hardware livre e de placa única criada na Itália, projetada com o microcontrolador ATmega328P com suporte de entrada/saída embutido, uma linguagem de programação típica com um ambiente de desenvolvimento, podendo ser também programado em C/C++, (ARDUINO, 2015).

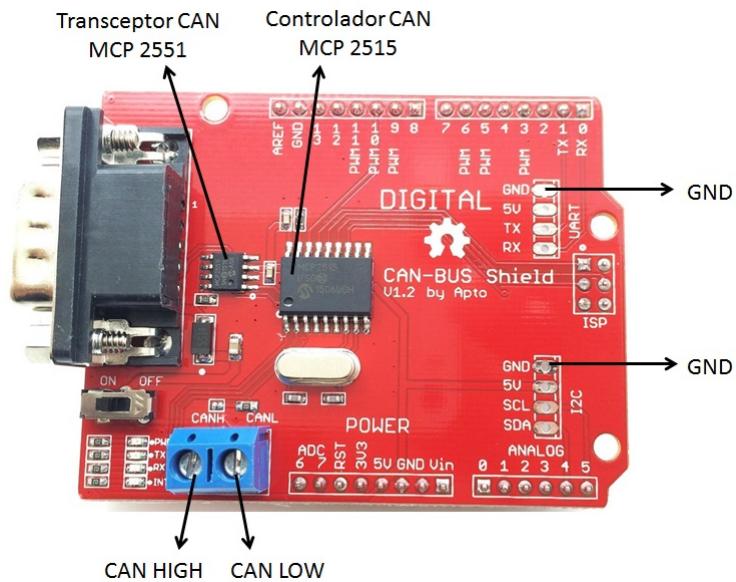
Figura 30 – Funduino UNO



Fonte: <http://www.arduino123.com/a/peijian/arduinokaifaban/2013/0729/26370.html>

O *Funduino UNO* não possui interface para o barramento CAN, assim foi necessário a compra de um adaptador CAN também. O adaptador escolhido para o *Funduino UNO* foi o *CAN BUS shield* da APTO, mostrado na figura 31.

Figura 31 – Apto CAN Bus Shield



Fonte: <http://www.amazon.de/Apto%C2%AE-CAN-BUS-Shield-f%C3%BCr-Arduino-Schwarz/dp/B00XQ4GHG2>

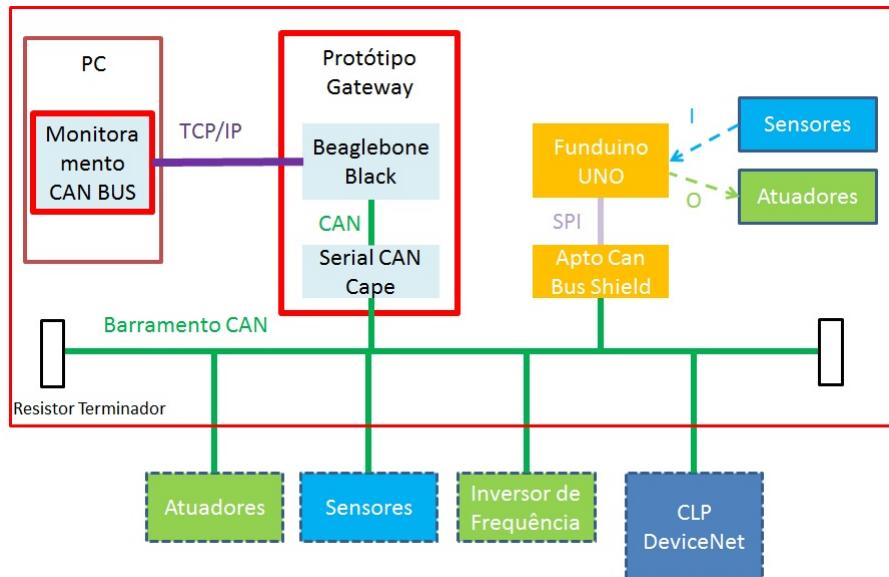
O *CAN BUS shield* da Apto implementa o transceptor MCP 2551 e o controlador CAN MCP 2515, promovendo interface para o CAN versão 2.0 B velocidade alta, com taxa de transmissão de até 1Mb/s, interface SPI de até 10 MHz e dois *buffers* para recepção de mensagens.

3.2.3 Arquitetura de Hardware

A arquitetura de hardware é composta por uma rede Ethernet TCP/IP entre o protótipo e a unidade de monitoramento de barramento CAN, e uma rede CAN entre o protótipo e uma ECU formada pelo *Funduino UNO* e a *shield CAN* conectados à sensores e atuadores, criada para aplicação de testes e validação do sistema desenvolvido, como na figura 32.

A rede CAN montada serve como uma rede modelo, aonde novas ECUs podem ser adicionadas, como o CLP *DeviceNet* do laboratório de automação industrial da EST.

Figura 32 – Arquitetura de Hardware



Fonte: Autora

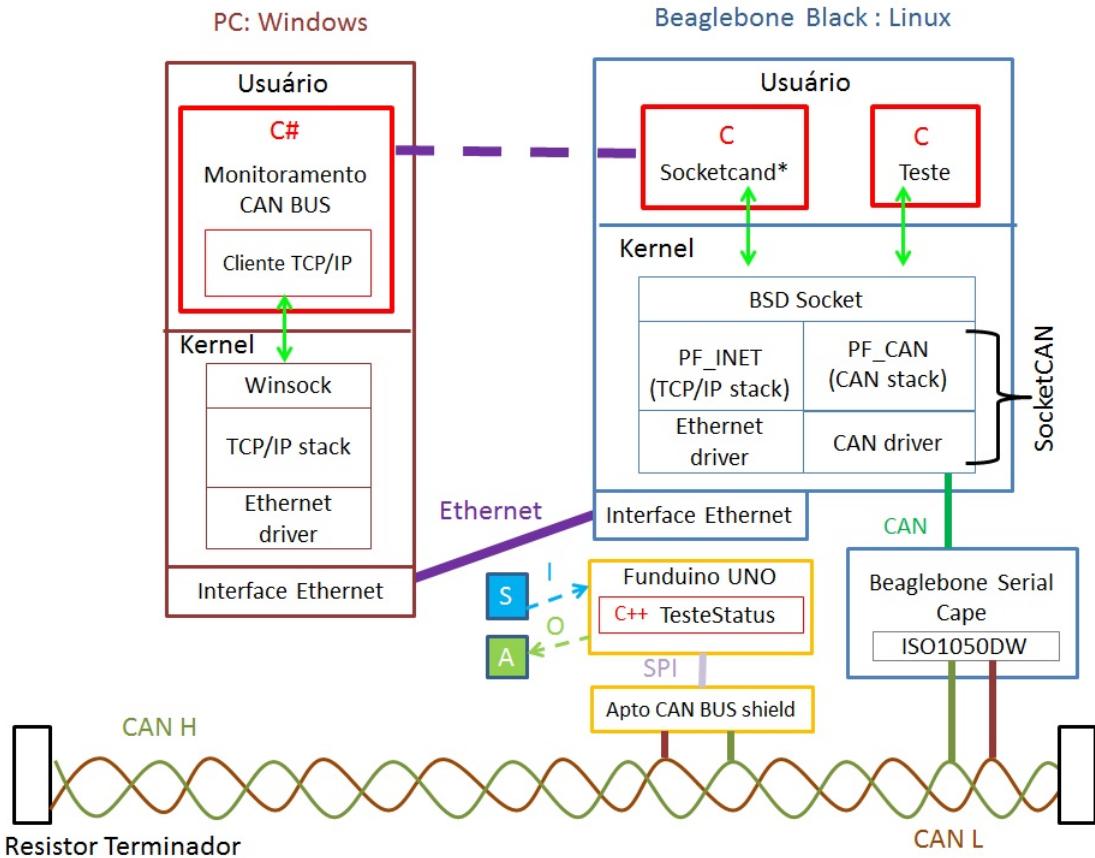
3.2.4 Arquitetura de Software

A arquitetura de software é subdividida em 4 módulos:

- O módulo Monitoramento CAN BUS, um módulo cliente TCP/IP em C#, responsável por receber e transmitir dados do/para o módulo *Socketcan**, com interface gráfica por meio da qual o usuário pode interagir com o barramento CAN;
- O módulo *Socketcan** em C que funciona como uma *gateway* entre os protocolos TCP/IP e CAN, baseado no *open source Socketcan*, (MEIER, 2015);
- O módulo Teste em C que interpreta as mensagens CAN recebidas pelo protótipo da ECU do *Funduino UNO* e que também é responsável pelo envio de mensagens CAN do protótipo para a ECU do *Funduino UNO* de acordo com a lógica da rede CAN criada, baseado na biblioteca, (SEEED-STUDIO, 2015);
- O módulo TesteStatus em C++ que faz a leitura dos sinais recebidos dos sensores e escrita dos sinais enviados aos atuadores pelo *Funduino UNO* à ele conectados, de acordo com a lógica da rede CAN criada, além de codificar esses dados em mensagens CAN e transmiti-los ao protótipo. Esse módulo também é responsável pela recepção e tratamento das mensagens CAN que chegam na ECU do *Funduino UNO*.

A figura 33 mostra a arquitetura de software do sistema, assim como a comunicação entre desses módulos é realizada dentro da arquitetura de hardware.

Figura 33 – Arquitetura de Software



Fonte: Autora

Os módulos *Socketcand** e *Teste* são baseados na *SocketCAN*, que é um conjunto *open source* de *drivers CAN* e *stack* de rede CAN contribuído pelo grupo de pesquisa *Wolkswagen* ao *kernel* do sistema operacional Linux, formalmente conhecido como *LLCF (Low Level CAN Framework)*, (ELINUX, 2015).

Tradicionalmente CAN *drivers* para Linux são baseados no modelo de dispositivos de caracteres, isto é envia/recebe um fluxo de caracteres (bytes, octetos), sem considerar qualquer estrutura de blocos, não sendo endereçável e não dispondo de qualquer operação de posicionamento somente, assim permitindo normalmente somente o envio e recebimento de dados direto do controlador CAN e o acesso do dispositivo apenas por um processo de cada vez. Além disso, cada um desses *drivers* apresentam uma interface diferente para a aplicação, dificultando a portabilidade. O conceito de *SocketCAN* por outro lado utiliza o modelo de dispositivos de rede, o que permite que múltiplas aplicações acessem um dispositivo CAN simultaneamente. Assim, uma única aplicação é capaz de acessar múltiplas redes CAN em paralelo, (WIKIPEDIA, 2015).

A *SocketCAN* estende o conceito das *Berkeley Socket APIs* no *Linux*, introduzindo uma nova família de protocolos, o *PF_CAN*, que coexiste com outras famílias de protocolo como o *PF_INET* para o protocolo de internet. Assim a comunicação com o barramento CAN é feita analogamente ao uso do protocolo de internet via *sockets*. Os componentes

fundamentais da *SocketCAN* são os *drivers* dos dispositivos de rede para diferentes controladores CAN e a implementação da família de protocolos CAN (CAN stack).

Um *driver* de dispositivo para hardware de controlador CAN se registra com a camada de rede do Linux como um dispositivo de rede, de forma que frames CAN do controlador possam ser passadas para a camada de rede e para o *stack* CAN e vice-versa. Assim, o módulo de protocolo da família CAN provê uma API para os módulos de protocolo de transporte para se registrar, de maneira que qualquer protocolos de transporte podem ser carregados e descarregados dinamicamente. Uma aplicação desejando se comunicar utilizando um protocolo específico, por exemplo o ISO-TP, precisa somente selecionar esse protocolo quando abrindo a *socket*, (HARTKOPP; THUERMANN; KIZKA JAN UND GRANDEGGER, 2002?).

A *SocketCAN* implementa dois protocolos CAN, o *raw* e o *broadcast manager* (BCM).

Após a conexão da *socket raw*, algumas configurações iniciais são preestabelecidas (*defaults*), podendo ser previamente à conexão serem reconfiguradas:

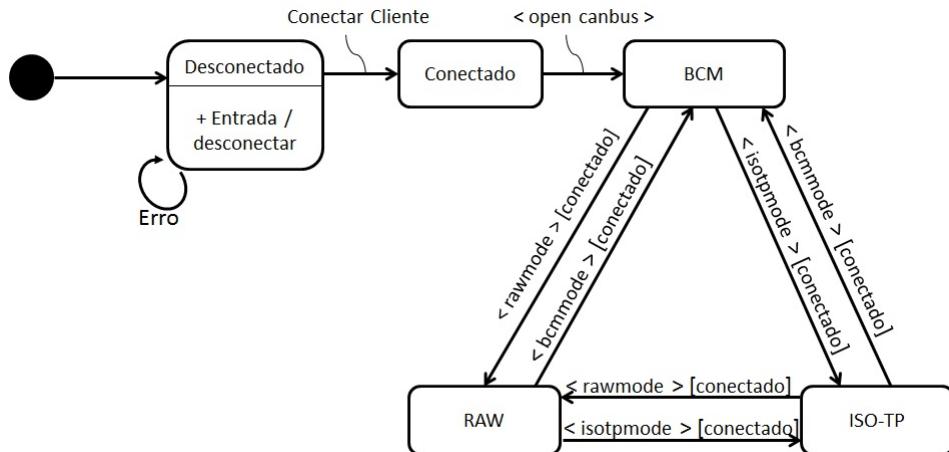
- Os filtros são configurados para apenas um filtro recebendo todas as *frames* de dados, com exceção das *frames* de erro e das *frames* enviadas pela mesma ECU;
- O *loopback* das *frames* enviadas é habilitado.

O protocolo de gerenciamento de transmissão *broadcast* (BCM) fornece uma interface de configuração baseada em comandos para filtrar e enviar mensagens CAN no *kernel* do Linux. Filtros podem ser usados para mostrar mensagens frequentes, detectar eventos, como mudança no conteúdo e tamanho das mensagens. Periódicas tarefas de transmissão de *frames* CAN ou uma sequência de *frames* podem ser criadas e modificadas no tempo de execução (*runtime*); ambos os conteúdos de mensagem e o intervalo de transmissão podem ser alterados.

*Socketcan** faz uso do protocolo ISO-TP, que é a ISO 15765-2, padrão internacional para o envio de pacotes de dados CAN, cobrindo as camadas de rede e transporte do Modelo OSI. Um canal ISO-TP consiste de dois identificadores CAN exclusivos, um para transmitir dados e outro para receber dados.

A *Socketcan** funciona de acordo com a máquina de estados da figura 34, possuindo cinco estados:

- Desconectado: o cliente não está conectado ao protótipo pela *socketcan**;
- Conectado: o cliente está conectado ao protótipo pela *Socketcan**;
- BCM: apresenta as funcionalidades do protocolo BCM da *SocketCAN*;
- RAW: apresenta as funcionalidades do protocolo RAW da *SocketCAN*;
- ISO-TP: permite a configuração do canal ISO-TP.

Figura 34 – Máquina de estados *Socketcan**^{*}

Fonte: Autora

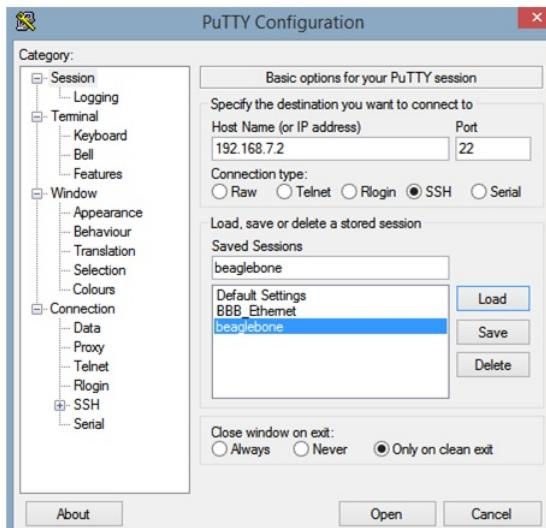
3.3 Implementação

3.3.1 Ferramentas de Conexão Remota

O desenvolvimentos dos softwares embarcados foram desenvolvidos por meio de conexão remota entre o *Beaglebone Black* e um computador pessoal, utilizando-se as ferramentas *PuTTy*, *WinSCP*, *TightVNC Viewer* e *Eclipse Luna*. O IP do *beaglebone black* por default é o 192.168.7.2.

PuTTy é um cliente SSH e *telnet*, desenvolvido por Simon Tathan para a plataforma *Windows*. Ele é *open source* software, disponível com código fonte e desenvolvido por um grupo de voluntários, (PUTTY, 2015).

Figura 35 – PuTTy Interface



Fonte: Autora

Figura 36 – PuTTy Terminal

```

Using username "root".
Debian GNU/Linux 7

BeagleBoard.org BeagleBone Debian Image 2014-04-23

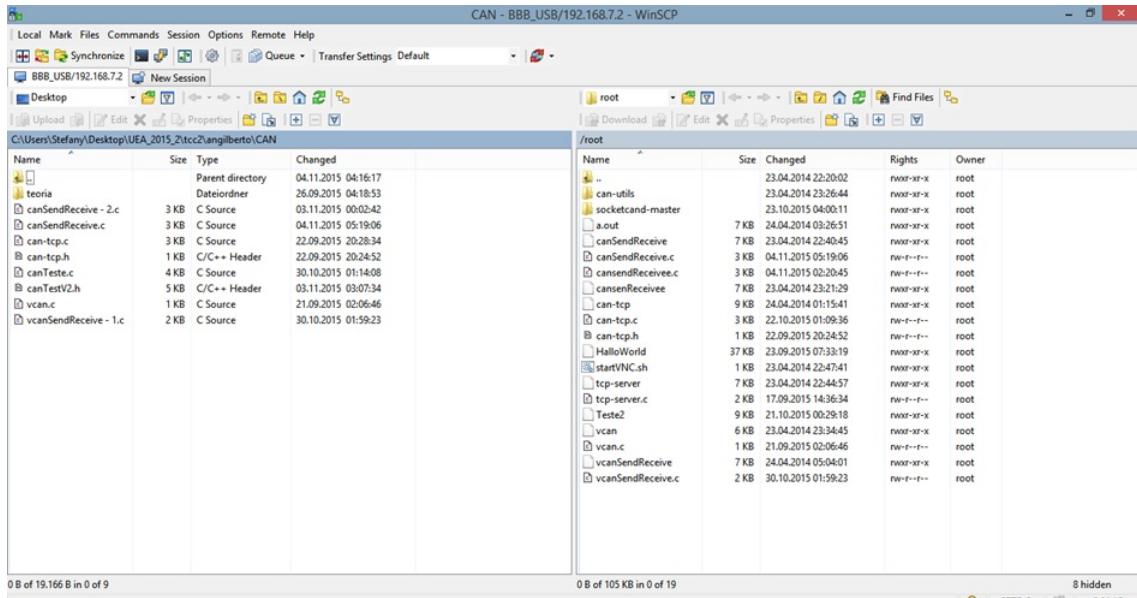
Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian
Last login: Wed Apr 23 20:20:32 2014 from vaio.local
root@beaglebone:~# date 111213402015
Thu Nov 12 13:40:00 UTC 2015
root@beaglebone:~#

```

Fonte: Autora

WinSCP é uma aplicação de software livre. Ele é um cliente SFTP gráfico para *Windows* que utiliza SSH. Sua principal função é facilitar a transferência segura de arquivos entre dois sistemas de computador, um local e remoto que oferece serviços de SSH, (WINSSCP, 2015).

Figura 37 – WinSCP

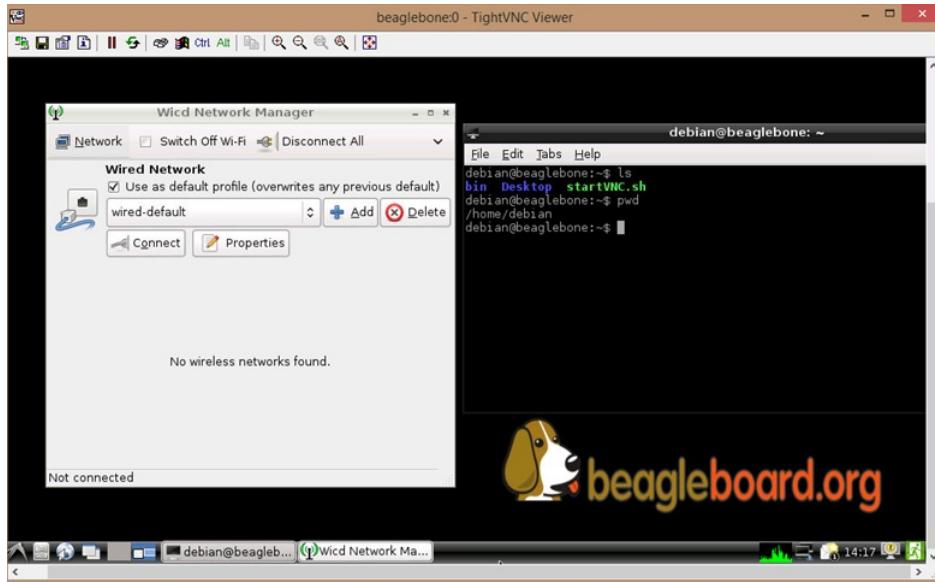


Fonte: Autora

Na fase de implementação dos softwares, os módulos desenvolvidos eram escritos em bloco de texto (*Notepad*) no computador local, transferidos para o *Beaglebone Black* via *WinSCP*, compilados com o *GCC (GNU compiler collection)* e executados via terminal *PuTTy* do *Linux*.

O *TightVNC Viewer* é um software de área de trabalho remota, o que permite o ambiente de um computador ser executado remotamente em um servidor e exibido em um dispositivo cliente separado, (TIGHTVNC, 2015).

Figura 38 – TightVNC Viewer

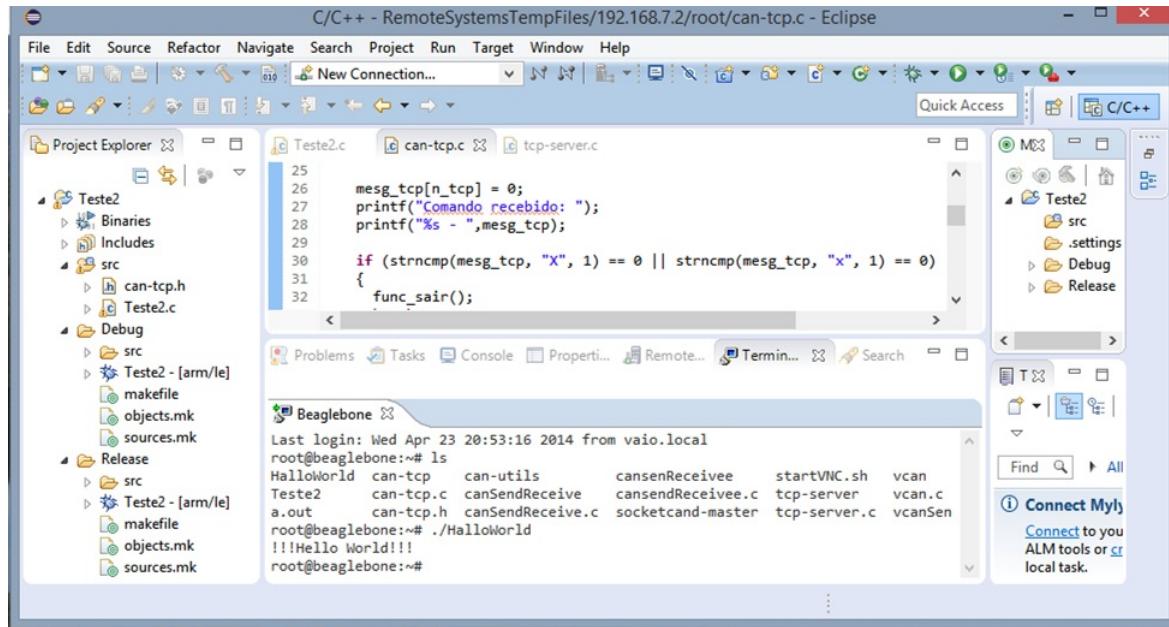


Fonte: Autora

O *Eclipse* é uma plataforma *cross-compiler*, sendo possível criar executáveis para uma plataforma diferente da plataforma em que o compilador é executável. Assim ele foi utilizado para desenvolvimento e debug dos módulos de software e posterior transferência dos arquivos pela conexão remota SSH para o *Beaglebone Black*.

O *Eclipse Luna* é a versão de junho de 2014 da plataforma *Eclipse*, sendo sucedida pela *Mars* de junho de 2015. A conexão e implementação remota utilizando-se o *Eclipse Luna* foi feita como explicado no tutorial em (LTD, 2015), utilizando-se o compilador *gcc-linaro-arm-linux-gnueabihf-4.9-2014.09_win32.zip*, o “make.exe” e as bibliotecas “libiconv2.dll” e “libintl3.dll” contidas no *make-3.81-bin.zip* e *make-3.81-dep.zip* respectivamente do *GnuWin* disponíveis em (GNUWIN32, 2006).

Figura 39 – Conexão Remota SSH Usando Eclipse Luna



Fonte: Autora

3.3.2 Configuração da Rede Ethernet TCP/IP

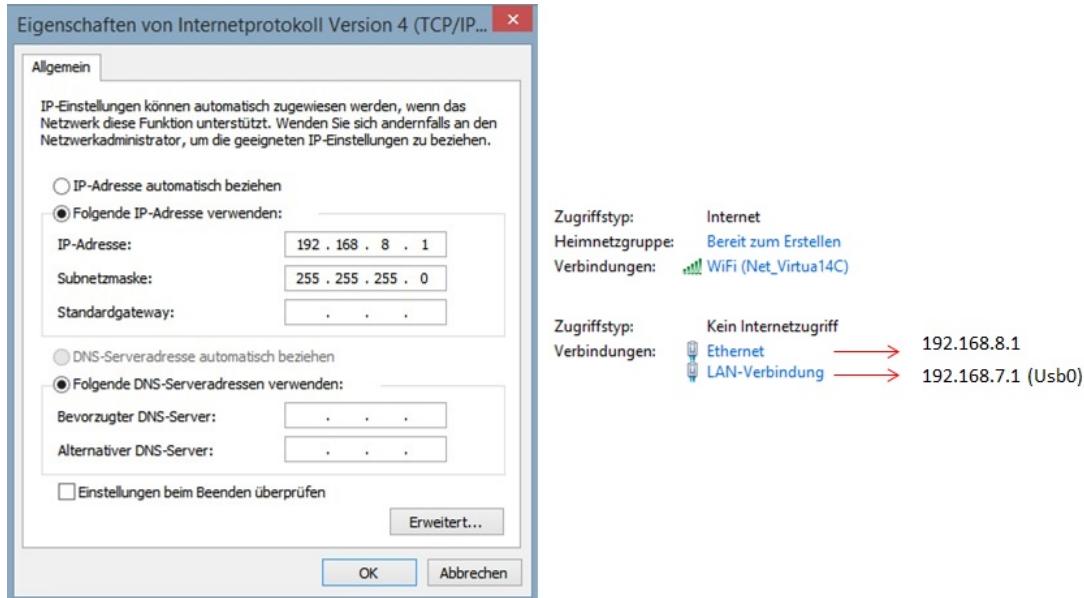
A conexão Ethernet TCP/IP entre o *Beaglebone Black* e o computador pessoal pode ser feita pela porta `usb0` que encapsula uma conexão Ethernet, com as seguintes configurações por *default*:

- IP estático 192.168.7.2
- *netmask* 255.255.255.0
- *network* 192.168.7.0
- *gateway* 192.168.7.1

Tal configuração pode ser visualizada com o comando `cat /etc/network/interfaces` por meio de um terminal remoto. Dessa forma, quando acessamos o *Beaglebone Black* por uma conexão remota SSH, o computador cliente recebe o IP 192.168.7.1.

Outra forma de conexão Ethernet entre o *Beaglebone black* e um computador cliente é por meio de um cabo Ethernet, configurando-se o IP do computador local como estático e como interface aonde um servidor DHCP irá ouvir, para se atribuir um IP pré-definido ao *Beaglebone black*. Foram escolhidos arbitrariamente os IP 192.168.8.1 para o computador local e o IP 192.168.8.200 para o *Beaglebone black*.

Figura 40 – Configuração do IPv4 do computador local



Fonte: Autora

O *Open DHCP Server* foi utilizado como servidor DHCP. Ele é um *open source Freeware* para *Windows* e *Linux*, (DHIR, 2015).

Figura 41 – Servidor DHCP

Active Leases			
Mac Address	IP	Lease Expiry	Hostname (first 20 chars)
78:a5:04:e6:73:8d	192.168.8.200	13-Nov-15 09:29:12	beaglebone

Fonte: Autora

3.3.3 Configuração da Rede CAN

A *SocketCAN* apresenta suporte para *driver CAN* virtual, que pode ser manipulado no *Linux* pelos comandos descritos abaixo em um terminal, (HARTKOPP; THUERMANN; KIZKA JAN UND GRANDEGGER, 2002?):

- modprobe vcan, para poder acessar a interface virtual CAN;
- ip link add type vcan, para criar uma interface CAN virtual de rede;
- ip link add dev vcan0 type vcan, para criar uma interface CAN virtual de rede com o nome 'vcan0';
- ip link set up vcan0, para ativar a interface "vcan0";

- ip link del vcan0, para remover a interface “vcan0” opcao 1;
- ip link set down vcan0, para parar a interface “vcan0” opcao 2.

Exemplo de sequência de comandos para a criação e carregamento do CAN virtual:

1. modprobe can;
2. modprobe can_raw;
3. modprobe vcan;
4. sudo ip link add dev vcan0 type vcan;
5. sudo ip link set up vcan0;
6. ip link show vcan0.

Para acessar o CAN, o *Beaglebone Black* precisa estar necessariamente conectado com um adaptador CAN, neste caso a *Serial CAN Cape*, e também precisa de uma fonte 5VDC de alimentação externa.

O dispositivo CAN deve ser configurado via interface *netlink*.

Exemplo de comando para criação e ativação de uma rede CAN com nome ‘can0’ e taxa de transmissão de dados igual a 500kb/s:

- ip link set can0 up type can bitrate 500000.

Para começar ou terminar uma rede CAN o comando abaixo é utilizado:

- ip link set canX up/down, com “*up*” para começar e o “*down*” para terminar.

Ainda um pacote de ferramentas *open software*, o *Can-utils*, (SOCKETCAN..., 2015), pode ser utilizado, oferecendo alguns utilitários do espaço do usuário para o subsistema *Linux SocketCAN*, como geração de *frames* e monitoramento no terminal.

Exemplos de alguns comandos do *Can-utils*:

- cangen can0/vcan0, para gerar *frames* aleatoriamente na rede CAN ou com o CAN virtual;
- candump can0/vcan0; para monitorar a rede CAN ou o CAN virtual com o terminal.

3.3.4 Montagem da Rede CAN e TCP/IP

Para validação e testes do sistema uma rede CAN entre o protótipo e a ECU do *Funduino UNO* foi criada. Os atuadores e sensores da rede foram ligados ao *Funduino UNO* de acordo com as tabelas 11 e 12.

Tabela 11 – Atuadores Rede CAN

Atuadores	Estado		<i>Funduino</i> Portas	Sigla	Imagen
	OFF	ON			
Botão	0	1	4	Bt	
Led azul	0	1	8	LAzul	
Led vermelho	0	1	2	LVerm	
Led verde	0	1	3	LVerd	
Buzina	0	1	5	Bz	

Fonte: Autora

Tabela 12 – Sensores Rede CAN

Sensores	Grandeza	<i>Funduino</i> Portas		Sigla	Imagen
Sensor ultrassônico HC-SR04	distância	6, 7	d		
Fotocélula 1K Ohm	luminosidade	A2	l		
TMP36	temperatura	A1	t		

Fonte: Autora

Em contraste com a rede Ethernet TCP / IP, a rede CAN possui um meio físico que funciona em regime *broadcast*, não apresentando nenhum endereçamento na camada MAC. O identificador CAN é utilizado para arbitração no barramento CAN. Assim quando se projeta uma rede CAN os identificadores devem ser mapeados para serem mandados para uma específica ECU, dessa forma o identificador pode ser visto como um tipo de fonte de endereçamento.

Para a rede CAN construída foram definidas duas mensagens CAN, cada uma com 4 bytes de dados, conforme a tabela 13. O *Funduino UNO* adquire os sinais dos sensores

e transporta essa informação na mensagem de identificador 0x01, por sua vez o protótipo recebe essa mensagem e responde à ECU do *Funduino UNO* com a mensagens de identificador 0x02, passando novos parâmetros para o *Funduino UNO*.

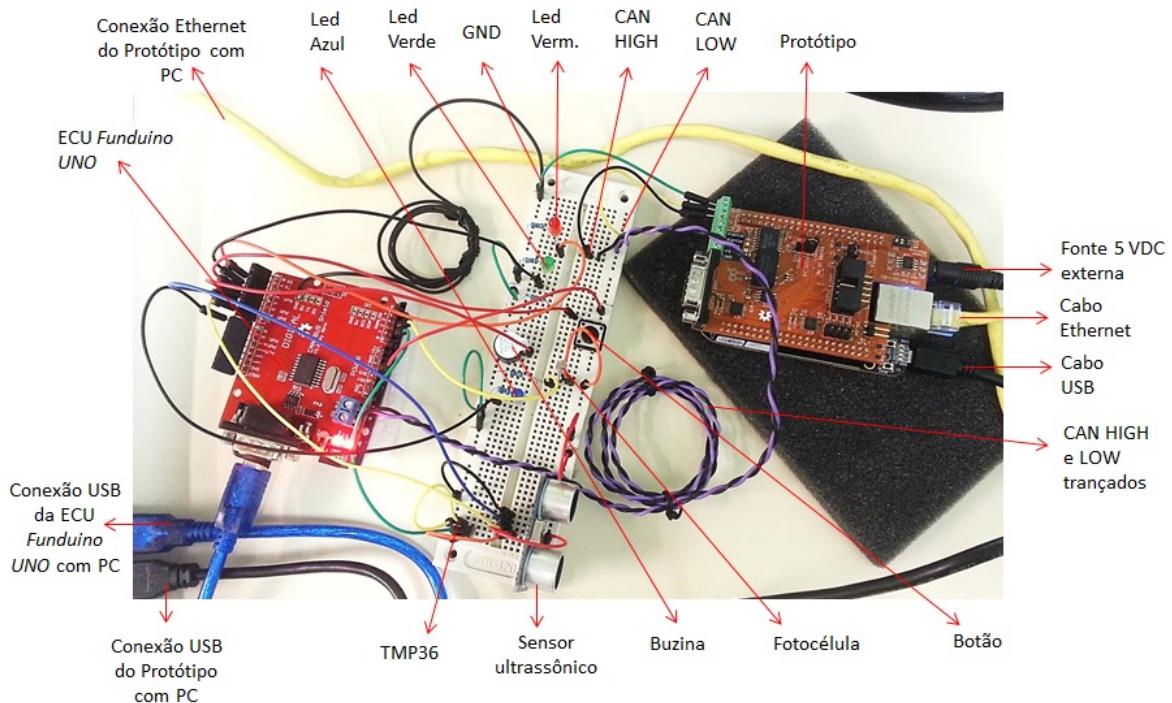
Tabela 13 – Mensagens CAN

Transmissor	Id	DLC	1 Byte	2 Byte	3 Byte	4 Byte
ECU <i>Funduino Uno</i>	0x01	4	“d”	“t”	“l”	“Bt”
Protótipo	0x02	4	se “d” > 0x38, “Bz” = 1, senão “Bz” = 0	se “t” > 0x2A, “LVerm” = 1, senão “LVerd” = 0	se “l” > 0xD5, “LVerd” = 1, senão “LVerd” = 0	se “Bt” = 1, “LAzul” = 1, senão “LAzul” = 0

Fonte: Autora

Após definição da estrutura física e lógica da rede CAN, ela foi montada conforme apresentado na figura 42.

Figura 42 – Rede CAN



Fonte: Autora

3.3.5 Configuração da Socketcan*

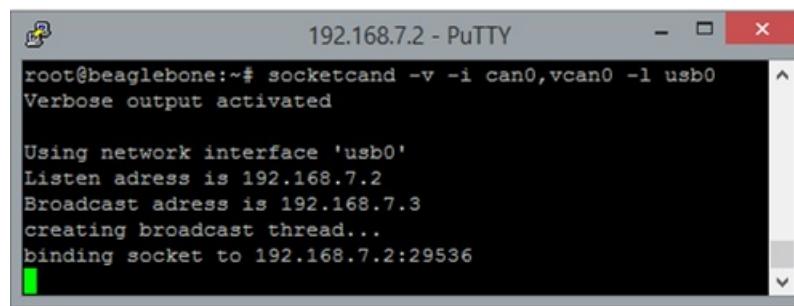
A *Socketcan** é inicializada no protótipo a partir do comando:

- *socketcan [-v | -verbose] [-i interfaces | -interfaces] [-p porta | -porta] [-l IP | -interface para se ouvir]*

Principais opções:

- -v ativa *verbose*;
- -i lista de interfaces separada por vírgulas que a *Socketcand** fornece acesso (por exemplo: -i can0,vcan1);
- -p muda a porta que o servidor *Socketcand** está ouvindo, (*default* 29536);
- -l escolhe a interface para a conexão da *Socketcand**, por exemplo eth0 ou usb0;
- -n desativa o UDP *broadcast beacon* para a porta 42000, que fornece informações sobre todas as redes CAN conectadas à *Socketcand** (por exemplo vcan0 e can0) acessíveis pelo do servidor BCM.

Figura 43 – Configuração da *Socketcand**



```
root@beaglebone:~# socketcand -v -i can0,vcan0 -l usb0
Verbose output activated

Using network interface 'usb0'
Listen address is 192.168.7.2
Broadcast address is 192.168.7.3
creating broadcast thread...
binding socket to 192.168.7.2:29536
```

Fonte: Autora

4 RESULTADOS E DISCUSSÕES

Neste capítulo serão abordados e discutidos todos os resultados obtidos no projeto, a partir da metodologia abordada no capítulo anterior.

4.1 Sistema de Monitoramento CAN

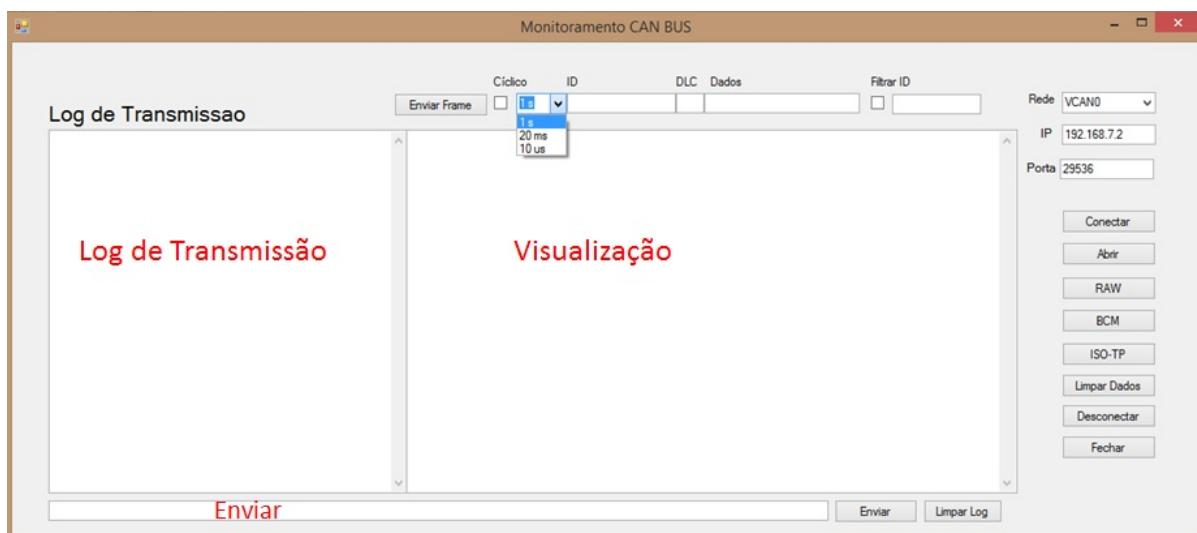
Depois do desenvolvimento dos módulos de software e hardware, esses componentes foram integrados formando o sistema final e foram submetidos em conjunto a testes para aprimoramento e validação do sistema como um todo.

Assim após montagem da rede CAN, configuração da redes Ethernet e CAN e da *Socketcan** e execução dos módulos Teste no *Beaglebone Black* e *TesteStatus* no *Funduino UNO*, o Sistema de monitoramento CAN (módulo Monitoramento CAN BUS), foi testado em conjunto com os outros módulos desenvolvidos e capaz de receber, transmitir e filtrar as mensagens CAN por meio do protótipo implementado, sendo assim alcançados os objetivos do trabalho.

4.1.1 Manual do Usuário do Módulo Monitoramento CAN BUS

Após a execução deste módulo, a interface gráfica da figura 44 é exibida ao usuário, por meio da qual ele pode interagir com o barramento CAN.

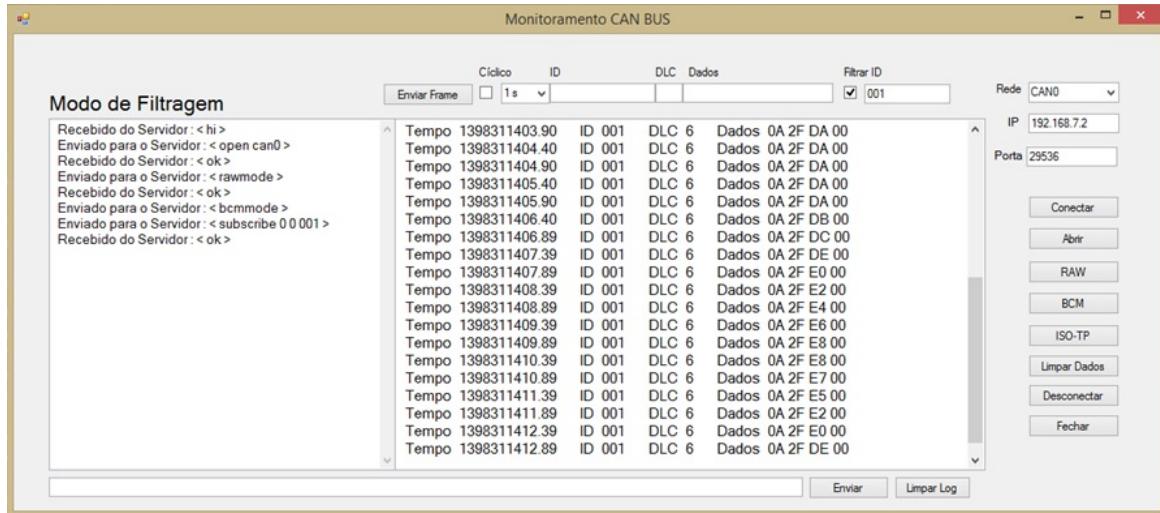
Figura 44 – Monitoramento CAN BUS



Fonte: Autora

O Monitoramento CAN BUS funciona de acordo com a máquina de estados da *Socketcan**. Dessa forma para visualização, transmissão e filtro de mensagens, o usuário deve

Figura 45 – Filtro no estado BCM



Fonte: Autora

primeiro conectar com o servidor, que neste caso é o protótipo, inserindo o IP do servidor no campo IP, que por *default* é o 192.168.7.2 (*Beaglebone Black*), e a porta da *socket* no campo Porta, sendo por *default* igual à porta *default* da *Socketcand** (29536).

Depois da configuração dos campos IP e Porta, a conexão com o protótipo é estabelecida pelo botão Conectar. Se a conexão TCP/IP for bem sucedida, o servidor manda uma mensagem <hi> para o cliente, podendo esta ser visualizada no campo Log de Transmissão. Em seguida o usuário deve abrir uma rede CAN selecionando uma das opções do menu Rede (vcan0 ou can0) e apertando o botão Abrir, considerando que a rede selecionada já foi previamente configurada no protótipo via interface *netlink* como explicado no Capítulo 3. Se o cliente tiver acesso ao barramento escolhido, o servidor enviará uma mensagem <ok> e a *socketcand** entra no estado BCM, senão uma mensagem de erro será enviada para o cliente, podendo esta ser visualizada no campo Log de Transmissão e a conexão é terminada.

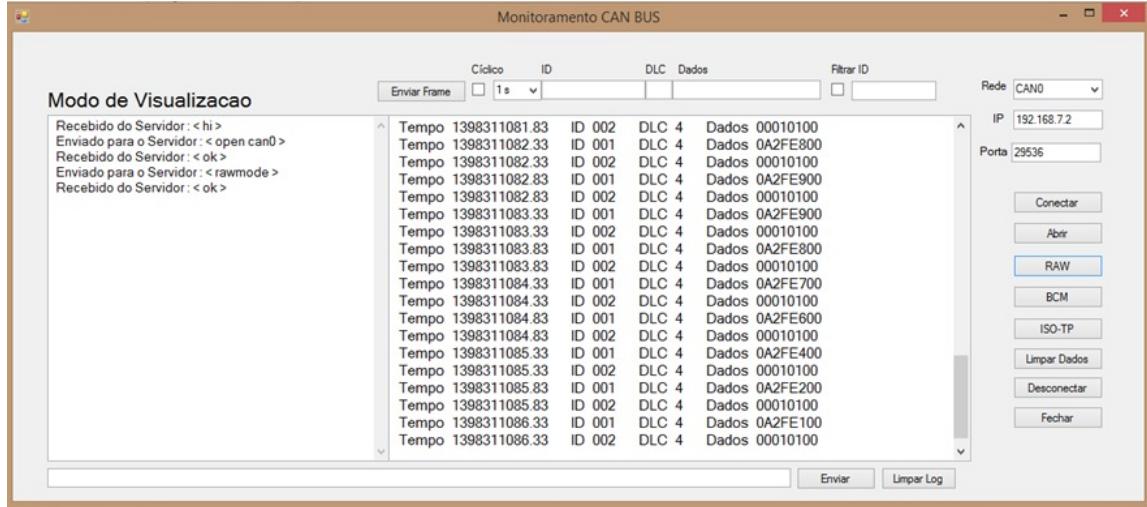
No estado BCM, o usuário pode enviar mensagens cíclicas e acíclicas e filtrar mensagens.

Para enviar mensagens cíclicas, basta preencher os campos ID, DLC e Dados com valores válidos, selecionar um intervalo no menu Cíclico (1s, 20ms, 10µs) e marcar a *checkbox* do mesmo campo. Desmarcando a *checkbox* as mensagens deixam de ser enviadas. Para enviar mensagens acíclicas no estado BCM, basta preencher os campos ID, DLC e Dados com valores válidos e apertar o botão Enviar Frame.

Para filtrar mensagens o usuário precisa preencher o campo Filtrar ID com um ID válido e marcar a *checkbox* correspondente, figura . Desmarcando essa *checkbox* as mensagens deixam de ser filtradas.

Para entrar no estado RAW, o usuário precisa apertar o botão RAW. Este estado permite visualizar todas as *frames* de dados recebidas por *default*, figura 46, com exceção das *frames* de erro e das *frames* enviadas pela mesma ECU como explicado no Capítulo 3.

Figura 46 – Estado RAW



Fonte: Autora

No estado RAW é possível enviar mensagens acíclicas pelos mesmos procedimentos anteriormente explicados para mensagens acíclicas BCM.

Para entrar no estado ISO-TP, basta apertar o botão ISO-TP.

O botão Limpar Dados limpa todas as *frames* recebidas do campo Visualização.

O botão Desconectar termina a conexão TCP/IP com o servidor.

O botão Fechar, fecha o programa.

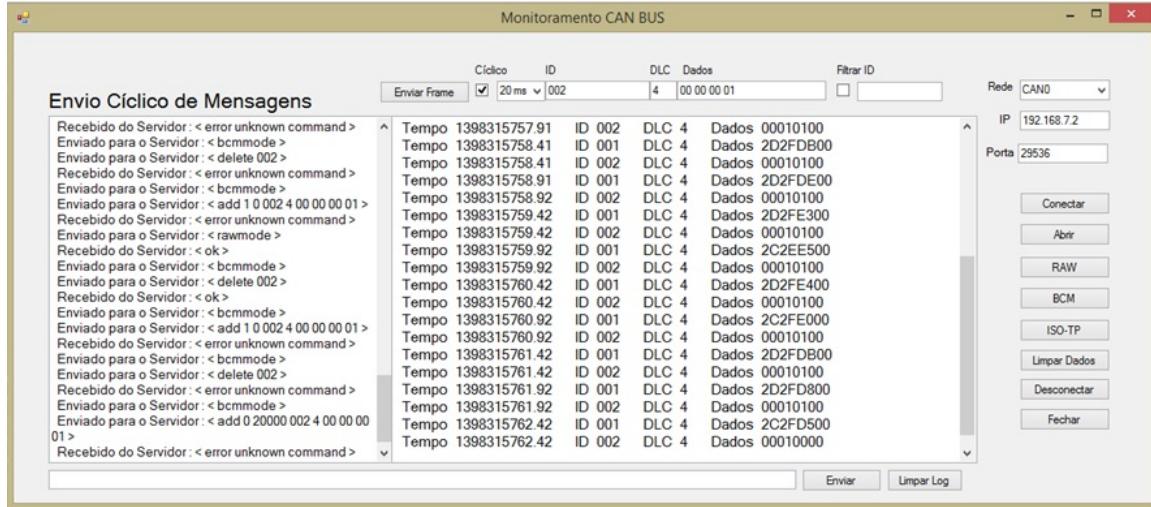
O botão Enviar, envia os comandos inseridos no campo Enviar via TCP/IP do módulo Monitoramento CAN BUS para a *Socketcan**.

O botão Limpar Log, limpa todos os dados do campo Log de Transmissão.

4.2 Discussão dos Resultados

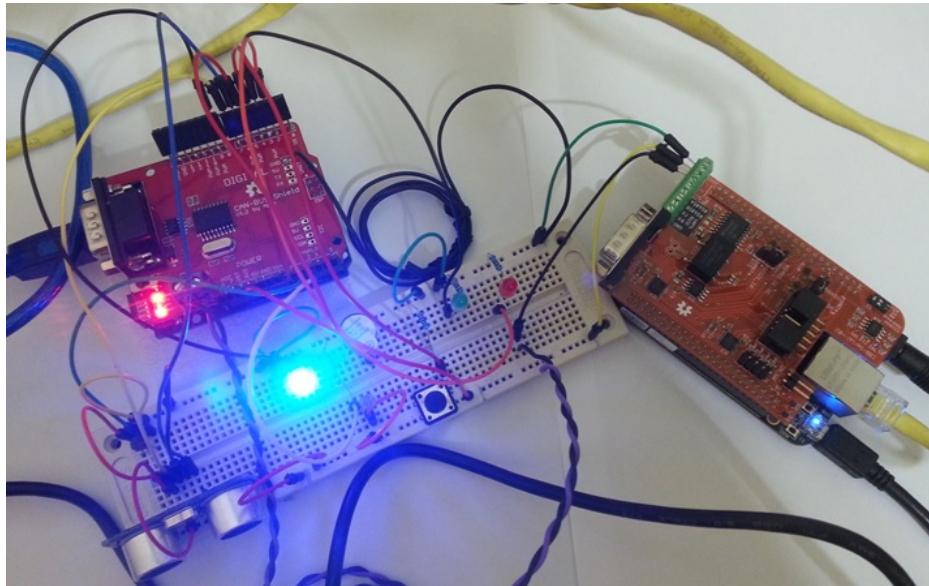
Ficou comprovado que a *Socketcan** integrada aos outros módulos permite o monitoramento de diferentes redes CAN por meio de apenas uma conexão TCP/IP, baseando-se na *SocketCAN*. As *frames* de dados enviadas pela ECU do *Funduino UNO* e respondidas pelo protótipo de acordo com a lógica da rede CAN montada e explicada no Capítulo 3 puderam ser visualizadas no sistema de monitoramento CAN, assim como a própria resposta dos atuadores ligados ao *Funduino UNO* às *frames* recebidas, tanto do usuário, quanto das ECUs da rede, validaram o sistema desenvolvido. As *frames* de dados também puderam ser filtradas pelo protótipo e visualizadas no sistema de monitoramento CAN.

Figura 47 – Envio cíclico da mensagem 002



Fonte: Autora

Figura 48 – Led azul acende com a mensagem cíclica 002 4 00 00 00 01 ou com o pressionar do botão



Fonte: Autora

5 CONCLUSÕES

Entre o campo de controle indústria moderna, sistema de controle inferior são a maior parte tecnologia de bus de campo utilizado. A maioria dos sistemas de controlo e gestão de camada superior Considere um computador com uma interface Ethernet como um terminal. No domínio da indústria controle, CAN (Controller Area Network), tecnologia de ônibus tem sido amplamente reconhecida e amplamente utilizados na automação industrial, devido ao seu elevado desempenho e confiabilidade e tem sido amplamente aplicada no navio, equipamento médico, inteligente edifício e outros campos.

5.1 Dificuldades encontradas

A principal dificuldade encontrada foi a compra dos materiais, que são importados.

5.2 Trabalhos futuros

Tem-se como trabalhos futuros

- Conectar o protótipo como servidor na internet;
- Adicionar ECUs à rede, como sensores inteligentes e o CLP DeviceNET do laboratório de automação industrial da EST;
- Criar um sistema supervisório para a rede CAN utilizando o protótipo;
- Implementar um módulo dentro do sistema de monitoramento para adicionar ECUs virtuais na rede CAN;
- Implementar outras *gateways* utilizando o protótipo.

Referências

- ARDUINO. *What is Arduino?* 2015. Disponível em: <<https://www.arduino.cc/en/Guide/Introduction>>. 52
- BASTOS, M. P. *Redes de Computadores: Aula 02.* 2012. 18
- BEAGLEBOARD. *BeagleBone Black.* 2015. Disponível em: <<http://beagleboard.org/black>>. 48
- BEAGLEBONE Serial Cape. Disponível em: <<http://www.logicsupply.com/cbb-serial/>>. 50
- CAETANO, B. C. S. K. *CONFIGURAÇÃO E PROGRAMAÇÃO DE UMA REDE DEVICENET PARA UM CCM.* Dissertação (Mestrado) — UNIVERSIDADE FEDERAL DE OURO PRETO, 2009. Disponível em: <<http://www.em.ufop.br/cecau/monografias/2009/Bruno%20C.%20S.%20Kfouri%20Caetano.pdf>>. 34
- DHIR, A. *Open DHCP Server.* 2015. Disponível em: <<http://sourceforge.net/projects/dhcpserver/>>. 61
- ELEMENT14. *The element14 BeagleBone Black (Revision C).* 2015. Disponível em: <http://www.element14.com/community/community/designcenter/single-board-computers/next-gen_beaglebone>. 48
- ELINUX. *CAN BUS.* 2015. Disponível em: <http://elinux.org/CAN_Bus>. 55
- FROEHLICH, A. A.; SOUZA, D. T.; BRATTI, D. *Sistemas Automotivos Embarcados.* 2008. Disponível em: <<http://www.lisha.ufsc.br/teaching/os/ine5355-2008-2/work/auto.pdf>>. 23
- GNUWIN32. *GnuWin: Provides native Win32 open source ports and utilities.* 2006. Disponível em: <<http://sourceforge.net/projects/gnuwin32/files/make/3.81/>>. 59
- GUIMARÃES, A. D. A. *Análise da norma ISO11783 e sua utilização na implementação do barramento de um monitor de semeadora.* Dissertação (Mestrado) — USP, 2003. Disponível em: <www.teses.usp.br/teses/disponiveis/.../Dissertacao.pdf>. 33
- HAMERSKI, J. C. *Desenvolvimento de uma Arquitetura parametrizável para processamento da pilha TCP/IP em hardware.* Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, 2008. Disponível em: <<http://www.lume.ufrgs.br/handle/10183/15310>>. 38, 40, 42, 43
- HARTKOPP, O.; THUERMANN, U.; KIZKA JAN UND GRANDEGGER, W. *Readme file for the Controller Area Network Protocol Family (aka SocketCAN).* 2002? Disponível em: <<https://www.kernel.org/doc/Documentation/networking/can.txt>>. 56, 61
- Institute of Industrial Automation and Software Engineering. *Laboratory Course Industrial Automation: Introduction to CAN.* Stuttgart, 2015. Disponível em: <<http://www.ias.uni-stuttgart.de/lehre/praktika/industrialautomation/unterlagen/01-CAN-en.pdf>>. 24, 26, 28

- IRC. *IRC for Beginners. Basics of tcp/ip, switching, routing and firewalling.* [2013?]. Disponível em: <http://www ircbeginner com/ircinfo/ Routing_Article pdf>.
- LTD, G. M. P. *Cross-Compile and Remote Deploy from Windows for BeagleBone using Eclipse and a gcc-Linaro Toolchain.* 2015. Disponível em: <<http://beaglebone globalmultimedia in/>>. 59
- LUIZ, H. *Apostila Fundamentos de Redes e TCP/IP.* 2007. Disponível em: <<http://www.ebah.com.br/content/ABAAABTK8AI/3288-redes>>. 37, 38
- MEIER, J.-N. *Socketcand.* 2015. Disponível em: <<https://github.com/dschanoeh/socketcand>>. 54
- MINMAX. *MINMAX MSLU100 Series.* [S.l.], 2013. Disponível em: <<http://www.cdistore.com/MinMax/datasheets/minmax/MSLU100-R17-130826.pdf>>. 51
- NETO, D. A. Requisitos técnicos para aplicação do protocolo can. *Universidade Metodista de Piracicaba,* 2010. Disponível em: <<http://www.ebah.com.br/content/ABAAAAPckAB/requisitos-tecnicos-aplicacao-can>>. 25
- ODVA. *CIP: How the protocol protects your investment.* 2002. Disponível em: <http://odva.or.kr/popup/2002oct_2.htm>. 35
- Pacheco PACHECO, N. D. O. *REDES INDUSTRIAS DE COMUNICAÇÃO.* [2010?]. Disponível em: <<https://www.passeidireto.com/arquivo/2675073/redes20industriais20i>>.
- PEREIRA, E. B. *Automação Industrial: A pirâmide de automação.* 2013. Disponível em: <http://www.ppgel.net.br/ebento/Disciplinas/automacao/Aula_x23.pdf>. 15
- PINHEIRO, J. M. S. *Padrao Ethernet.* 2010. Disponível em: <https://www.projetoderedes.com.br/aulas/ugb_redes_I/ugb_redes_I_material_de_apoio_02.pdf>. 39
- PUTTY. *PuTTY Download Page.* 2015. Disponível em: <<http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>>. 57
- RASPBERRYPI. *WHAT IS A RASPBERRY PI?* 2015. Disponível em: <<https://www.raspberrypi.org/help/what-is-a-raspberry-pi>>. 48
- ROSSIT, R. *Curso Redes Industriais DeviceNet.* 3. ed. São Paulo, 2002. 19, 20, 21, 34
- SANTOS, A. V. P. *Desenvolvimento de software de aplicação para uma rede CAN e sua interligação com uma rede Ethernet.* Dissertação (Mestrado) — Faculdade de Engenharia da Universidade do Porto, 2002. Disponível em: <<https://web.fe.up.pt/~ee99058/projecto/>>. 24, 28
- SARGUNAPRIYA, N.; MANI, M. J.; AMUDHA, P. Monitoring and control system for industrial parameters using can bus. *International Journal of Engineering Trends and Technology,* 2014. 12
- Schneider Electric. *Industrial communication glossary.* 2002. Disponível em: <http://overpof.free.fr/schneider/CAN%20&%20CANopen/CANopen/Pr%20sentations/Training%20I/Glossaire/En/Com_Glossary_en.pdf>. 16
- SEEED-STUDIO. *CAN BUS Shield.* 2015. Disponível em: <https://github.com/Seeed-Studio/CAN_BUS_Shield>. 54

- SMAR. *Tutorial DeviceNet*. 2012. Disponível em: <<http://www.smar.com/brasil/devicenet>>. 36
- SOCKETCAN userspace utilities and tools. 2015. Disponível em: <<https://github.com/linux-can/can-utils/>>. 62
- STEMMER, M. R. *Das 5331 - sistemas distribuídos e redes de COMPUTADORES para Controle e automação Industrial*. 2001. Disponível em: <http://alvarestech.com/temp/simprebal/Relatorios_Tecnicos-Publicacoes-Dissertacoes/docs/cursos/Aula6-Apostila-Sistemas_Distribuidos_E_Redes_De_Computadores_Para_Controle.pdf>. 23, 44, 45
- TANENBAUM, A. S. *REDES DE COMPUTADORES*. 4^a ediÇÃo. ed. [S.l.]: CAMPUS ELSEVIER, 2003. 17, 21, 22
- TEXAS INSTRUMENTS. *ISO1050 Isolated CAN Transceiver*. [S.l.], 2015. Disponível em: <<http://www.ti.com/lit/ds/symlink/iso1050.pdf>>. 51
- TIGHTVNC. *Download TightVNC*. 2015. Disponível em: <<http://www.tightvnc.com/download.php>>. 58
- TSOU, T.-H. *An Implementation of Controller Area Network Bus Analyzer Using Microblaze and Petalinux*. Dissertação (Mestrado) — Western Michigan University, 2013. 28
- UEL. *Redes Industriais de COMUNICAÇÃO*. 2010. Disponível em: <http://www.ebah.com.br/content/ABAAAAA_3MAG/redes-industriais-comunicacao>. 15
- Vector Informatik GmbH. *CANoe Fundamentals Workshop*. 2014. 23, 25, 28, 31
- WANG, Y.; HE, L. The application of microcontroller mc9s08dz60 in automotive can bus electronic control unit. *The Open Electrical & Electronic Engineering Journal*, 2013. Disponível em: <<http://test.benthamopen.com/contents/pdf/TOEEJ/TOEEJ-7-110.pdf>>. 11
- WIKIPEDIA. *SocketCAN*. 2015. Disponível em: <<https://en.wikipedia.org/wiki/SocketCAN>>. 55
- WINSCP. *WinSCP Downloads*. 2015. Disponível em: <<https://winscp.net/eng/download.php#download2>>. 58
- W&T. *TCP/IP - Ethernet*. [S.l.], 1999. 39
- ZHANG, Y.-p.; FENG, X.-h.; GUO, Y.-x. Ethernet-can protocol conversion module. *Advanced Science and Technology Letters*, 2013. Disponível em: <http://onlinepresent.org/proceedings/vol31_2013/7.pdf>. 12

APÊNDICE A – Algoritmo implementado no Funduino Uno (TesteStatus)

```
#include <mcp_can.h>
#include <SPI.h>

//botao
int taster=4;
int tasterstatus=0;

//leds
int LEDblau=8;
int LEDrot=2;
int LEDgelb=3;
int LedBlauStatus =0;
int LedRotStatus =0;
int LedGelbStatus =0;

//sensor ultrassonico
int trigger=7;
int echo=6;
long dauer=0;
long entfernung=0;

//sensor de luminosidade
int eingang= A2;
int sensorWert = 0;

// buzina
int Pieps=5;
int PiepsStatus = 0;

// sensor de temperatura
int TMP36 = A1;
int temperatur = 0;
int temp[10];
int time= 20;
```

```
void setup()
{
    Serial.begin(115200);

    // init can bus, baudrate: 500k
    if(CAN.begin(CAN_500KBPS) ==CAN_OK)
    {
        Serial.print("Can init ok!!\r\n");

        pinMode(taster, INPUT);
        pinMode(LEDblau, OUTPUT);
        pinMode(LEDrot, OUTPUT);
        pinMode(LEDgelb, OUTPUT);
        pinMode(trigger, OUTPUT);

        //sensor ultrassonico
        pinMode(echo, INPUT);
        pinMode(Pieps,OUTPUT);
    }
    else Serial.print("Can init fail!!\r\n");
}

void temperatura()
{
    temp[0] = map(analogRead(TMP36), 0, 410, -50, 150);
    delay(time);

    temp[1] = map(analogRead(TMP36), 0, 410, -50, 150);
    delay(time);

    temp[2] = map(analogRead(TMP36), 0, 410, -50, 150);
    delay(time);

    temp[3] = map(analogRead(TMP36), 0, 410, -50, 150);
    delay(time);

    temp[4] = map(analogRead(TMP36), 0, 410, -50, 150);
    delay(time);

    temp[5] = map(analogRead(TMP36), 0, 410, -50, 150);
    delay(time);

    temp[6] = map(analogRead(TMP36), 0, 410, -50, 150);
    delay(time);

    temp[7] = map(analogRead(TMP36), 0, 410, -50, 150);
    delay(time);
```

```
temp[8] = map(analogRead(TMP36), 0, 410, -50, 150);
delay(time);
temp[9] = map(analogRead(TMP36), 0, 410, -50, 150);
temperatur=(temp[0]+temp[1]+temp[2]+temp[3]+temp[4]+temp[5] +temp[6]+temp[7]);
void distancia()
{
// medicao de distancia com o sensor ultrassonico
digitalWrite(trigger, LOW);
delay(5);
digitalWrite(trigger, HIGH);
delay(10);
digitalWrite(trigger, LOW);
dauer = pulseIn(echo, HIGH);
entfernung = (dauer/2) / 29.1;
}
void loop()
{
// configuracao de recepcao de mensagens
byte nMsgLen = 0;
byte nMsgBuffer[8];
unsigned char buf[8];
char sString[4];
unsigned char Dta[8];
if(CAN.checkReceive() == CAN_MSGAVAIL)
{
// Read the message buffer
CAN.readMsgBuf(&nMsgLen, &nMsgBuffer[0]);
INT32U nMsgID = CAN.getCanId();
// Print message ID to serial monitor
Serial.print("Message ID: 0x");
if (nMsgID == 0x02)
{
unsigned char a[nMsgLen];
for (int i = 0; i < nMsgLen; i++)
{
for (int nIndex = 0; nIndex < nMsgLen; nIndex++)
{
a[i] = *(&nMsgBuffer[0]+i);
}
```

```
// end for 2
}// end for1

// se distancia > 0x38
if (a[0] == 0x01)
{
    digitalWrite(Pieps, HIGH); // buzina liga
    //delay(1);
}
else if (a[0] == 0x00)
{
    digitalWrite(Pieps, LOW);
}
// se temperatura > 0x2A,
if (a[1] == 0x01)
{
    digitalWrite(LEDrot, HIGH);
    //led vermelho acende
    //delay(1000);
}
else if (a[1] == 0x00)
{
    digitalWrite(LEDrot, LOW);
}
// se luminosidade > 0xD5
if (a[2] == 0x01)
{
    digitalWrite(LEDgelb, HIGH);
    //led verde acende
}
else if (a[2] == 0x00)
{
    digitalWrite(LEDgelb, LOW);
}
// botao = 1 (apertado)
if (a[3] == 0x01)
{
    digitalWrite(LEDbblau, HIGH);
    // led azul acende
```

```
// delay(1000);
}

else if (a[3] == 0x00)
{
    digitalWrite(LEDblau, LOW);
}

} // end if (nMsgID == 0x02)
// imprimir mensagem no serial monitor do Arduino
if (nMsgID < 16) Serial.print("0");
Serial.print(itoa(nMsgID, sString, 16));
Serial.print("\n\r");

// Print data to serial monitor
Serial.print("Data: ");
for (int nIndex = 0; nIndex < nMsgLen; nIndex++)
{
    Serial.print("0x");
    if (nMsgBuffer[nIndex] < 16)
        Serial.print("0");
    Serial.print(itoa(nMsgBuffer[nIndex], sString, 16));
    Serial.print(" ");
}

// end for
Serial.print("\n\r\n\r");
}

// end if
// check de status

// status do botao
tasterstatus=digitalRead(taster);
//distancia
distancia();
// sensor de luminosidade
sensorWert =analogRead(eingang);
// temperatura  temperatura();
// Envio de mensagens
unsigned char stmp_entfernung[4] = {entfernung, temperatur, sensorWert, ta
CAN.sendMsgBuf(0x01, 0, 4, stmp_entfernung); // id padrao
//CAN.sendMsgBuf(0x00, CAN_EXTID, 8, stmp); id padrao extendido
delay(300);

}

// fim
```

APÊNDICE B – Algoritmo implementado no Protótipo (Teste)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <net/if.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <linux/can.h>
#include <linux/can/raw.h>

int main( void )
{
    int s;
    int nbytes;
    struct sockaddr_can addr;
    struct can_frame frameReceive;
    struct can_frame frameSend;
    struct ifreq ifr;
    int PiepsStatus = 0x00;
    int LedRotStatus = 0x00;
    int LedGelbStatus = 0x00;
    int LedBlauStatus = 0x00;

    char *ifname = "can0";
    if ((s = socket(PF_CAN, SOCK_RAW, CAN_RAW)) < 0)
    {
        perror("Error while opening socket");
        return -1;
    }
    strcpy(ifr.ifr_name, ifname);
    ioctl(s, SIOCGIFINDEX, &ifr); /*To determine the interface index an approach
    be used */
    addr.can_family = AF_CAN;
```

```
addr.can_ifindex = ifr.ifr_ifindex;
printf("%s at index %d\n", fname, ifr.ifr_ifindex);
if(bind(s, (struct sockaddr *)&addr, sizeof(addr)) < 0)
{
    perror("Error in socket bind");
    return -2;
}
for(;;)
{
    /* receiving CAN frames*/
    nbytes = read(s, &frameReceive, sizeof(struct can_frame));
    if(nbytes < 0)
    {
        perror("can raw socket read");
        return 1;
    }
    /* paranoid check ... */
    if(nbytes < sizeof(struct can_frame))
    {
        fprintf(stderr, "read: incomplete CAN frame\n");
        return 1;
    }
    if(frameReceive.can_id == 0x01)
    { /* {entfernung, temperatur, sensorWert, tasterstatus} */
        if(frameReceive.data[0] > 0x38)
        {
            PiepsStatus = 1;
        }
        else
        {
            PiepsStatus = 0;
        }
        if(frameReceive.data[1] > 0x2A)
        {
            LedRotStatus = 1;
        }
        else
        {
            LedRotStatus = 0;
        }
        if(frameReceive.data[2] > 0xD5)
```

```
{  
    LedGelbStatus = 1;  
}  
else  
{  
    LedGelbStatus = 0;  
}  
if (frameReceive.data[3] == 0x01)  
{  
    LedBlauStatus = 1;  
}  
else  
{  
    LedBlauStatus = 0;  
}  
/*sending CAN frames*/  
frameSend.can_id = 0x02;  
frameSend.can_dlc = 4;  
frameSend.data[0] = PiepsStatus;  
frameSend.data[1] = LedRotStatus;  
frameSend.data[2] = LedGelbStatus;  
frameSend.data[3] = LedBlauStatus;  
    nbytes = write(s, &frameSend, sizeof(struct can_frame));  
    printf("Wrote %d bytes\n", nbytes);  
}  
}  
}
```