Uma análise comparativa entre os protocolos *CANopen*, *DeviceNet* e *Smart Distributed System*

Thiago Mendes da Silva [1]
Marco Aurélio Spohn [1]
Adriano Sanick Padilha [1]

Resumo: O uso de protocolos de comunicação em redes industriais permite maior controle dos dados e Qualidade de Serviço (QoS) em diversas aplicações. Esse artigo contempla uma análise comparativa de três protocolos de alto nível - CANopen, DeviceNet e Smart Distributed System (SDS) - baseados no padrão Controller Area Network (CAN). Como estudo de caso, realiza-se uma avaliação abrangendo métricas de desempenho que possibilitam acompanhar como cada protocolo responde às aplicações alvo. Aborda-se, em conjunto, alguns fundamentos de redes industriais, bem como a arquitetura e funcionamento dos protocolos analisados.

***Palavras-chave*:** Protocolos de comunicação. *Controller Area Network*.

Abstract: *The use of communication protocols in industrial networks enables greater control of data and Quality of Service (QoS) in various applications. This article includes an analysis of three high level protocols - CANopen, DeviceNet and Smart Distributed System (SDS) - based on the Controller Area Network (CAN) standard. We have carried out a performance analysis taking into account some representative performance metrics for assessing how each protocol behaves for some target applications. We also address some fundamentals of industrial networks, as well as the architecture and operation of the analyzed protocols.*

***Keywords*:** *Communication protocols. Controller Area Network.*

1 Introdução

Redes de controle industriais têm requisitos específicos e métodos de operação diferenciados das redes comerciais e, uma diferença essencial, é que redes de controle industriais são conectadas em equipamentos físicos (*e.g.*, um motor, uma esteira, uma caldeira) para controlar e monitorar ações e condições do mundo real. Disso resulta um conjunto de requisitos de Qualidade de Serviço (*Quality-of-Service* - QoS) diferente das redes comerciais, uma vez que a rede deve ser determinística e trabalhar em tempo real.

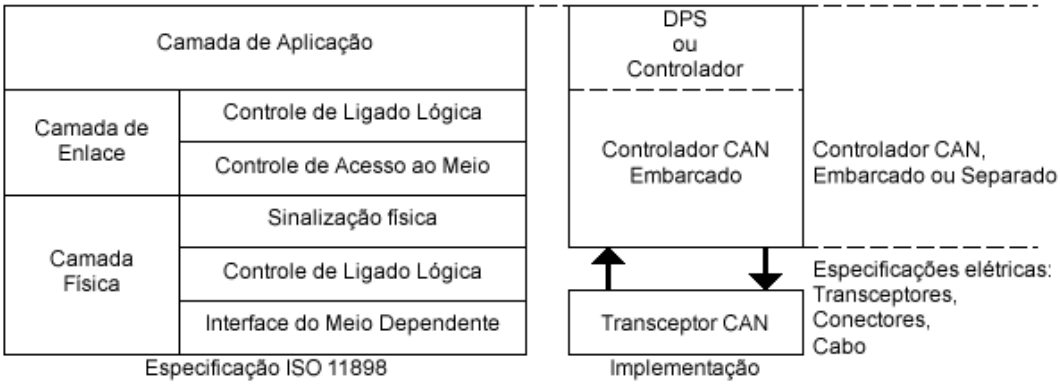
Uma das diferenças entre as redes industriais e as redes comercias é a arquitetura. Em redes industriais a arquitetura consiste em três ou quatro níveis: a conexão de instrumentos aos controladores acontece em um nível, a conexão entre os controladores em outro nível, a interface homem-máquina ocorre em outro nível e a coleta de dados e a comunicação externa acontece em outro nível.

A segurança é outro ponto que difere as duas redes. Devido ao fato de as redes de controle industriais estarem conectadas diretamente a maquinários industriais, falhas no sistema podem causar danos nos equipamentos, causando perda de produção e até perda de vidas. A celeridade com que os dados precisam ser transmitidos, processados e respondidos nas aplicações industriais estabelecem um tempo limite, chegando próximo ao "instantâneo". Uma regra é que o tempo de resposta deveria ser menor que o tempo de amostragem dos dados reunidos. O que diferencia das redes comerciais, que por sua vez, não necessitam de um tempo de resposta delimitado. E se necessitam, usualmente utilizam um tempo da ordem de centenas de milissegundos, ou mais.

[1]Universidade Federal da Fronteira Sul (UFFS) - Curso de Ciência da Computação - Chapecó/SC
{th.thiago.mendes@gmail.com},{marco.spohn,padilha@uffs.edu.br}

http://dx.doi.org/10.5335/rbca.2016.XXX

Figura 1: Arquitetura padrão do CAN



Em uma rede industrial, não só os dados precisam ser transmitidos em tempo real, mas também precisam ser transmitidos de forma determinística. Isso significa que deve ser possível prever quando uma resposta a uma transmissão será recebida. Para isso, a latência de um sinal deve ser delimitada e ter baixa variância. Usualmente, nos níveis mais baixos da rede, há poucos dados a serem transmitidos, geralmente resultante de uma única medição pelo sistema.

Transmissões feitas em redes industriais necessitam tanto de transmissões periódicas quanto de transmissões aperiódicas, como mudanças de estados ou alarmes. O período usado para coleta e transmissão dos dados podem variar de dispositivo para dispositivo de acordo com a necessidade de controle de cada um, além de eventos aperiódicos que podem ocorrer a qualquer momento. Para facilitar as transmissões, relógios e contenção de barramento são implementados nos níveis mais baixos das redes industriais para assegurar que todos os dados serão transmitidos em tempo hábil.

Nas redes industriais há a necessidade de determinar o momento em que ocorreram transmissões e a ordem dos eventos, especialmente em casos de transmissões aperiódicas. Isso é conseguido usando carimbos de tempo, que são marcadores que guardam informações de quando um evento aconteceu, e relógios sincronizados.

Há protocolos que garantem a qualidade de serviço que as redes industriais requerem. Esses protocolos aplicam diversas técnicas para manter os dados consistentes, não sofrer interferências externas e garantir a transmissão dos dados. Dentre os protocolos de redes industriais mais representativos estão: CAN, EtherNet/IP, PROFIBUS, PROFINET, INTERBUS, Foundation Fieldbus [1]. O protocolo CAN atua nas duas primeiras camadas da rede (física e enlace) e dispõe de um conjunto de recursos que atendem os requisitos de uma rede industrial. Principais recursos são: rápida transmissão e consistência de dados, mecanismos de detecção de erros, robustez, além de oferecer uma boa relação preço/desempenho. Baseados no protocolo CAN, existem três protocolos de referência que atuam na camada de aplicação [2].

Esse artigo apresenta uma análise comparativa de três protocolos de alto nível - CANopen [3], DeviceNet [4] e Smart Distributed System (SDS) [5] - baseados no padrão *Controller Area Network* (CAN). Como estudo de caso, realiza-se uma avaliação contemplando métricas de desempenho que possibilitam observar como cada protocolo responde às aplicações alvo. Aborda-se, em conjunto, alguns fundamentos de redes industriais, bem como a arquitetura e funcionamento dos protocolos analisados.

2 *Controller Area Network* - CAN

Controller Area Network (CAN) [6] contempla um protocolo de comunicação serial, definido no padrão ISO 11898, com arquitetura (ilustrada na Figura 1) multi-mestre baseada em *broadcast*. O protocolo oferece uma boa relação preço/desempenho, permite transmissão de dados de até 1 Mbits/s e pode ser implementado em sistemas de tempo real; adicionalmente, adota mecanismos robustos de detecção de erros. O CAN também oferece a possibilidade de substituir componentes da rede durante a operação do sistema.

Figura 2: CAN 2.0A com identificador de 11 bits

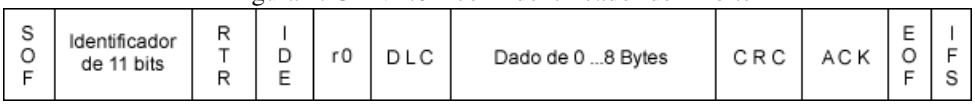
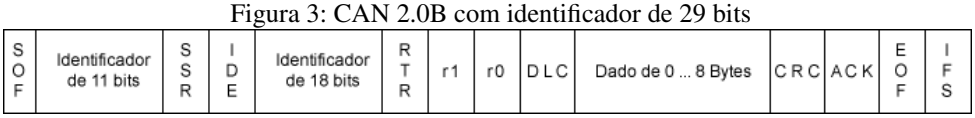


Figura 3: CAN 2.0B com identificador de 29 bits



A arquitetura definida no ISO 11898 abrange as camadas física e de enlace. O CAN utiliza *Carrier Sense Multiple Access* (CSMA) com detecção de colisão e arbitragem na prioridade da mensagem (CD+AMP): as colisões são resolvidas através da arbitragem de lógica binária, com base na prioridade pré-programada de cada mensagem (a mensagem com a maior prioridade vence o acesso ao barramento) [7, 8]. Dispositivos CAN possuem identificadores de 11 bits no padrão ISO 11898 (também chamado de CAN 2.0A) e identificadores de 29 bits na emenda definida no padrão ISO 11989 (também referenciado como CAN 2.0B) [7, 6].

Os pacotes de dados no CAN possuem os seguintes campos (ilustração na Figura 2):

- SOF: Início de quadro. Único bit dominante. Um bit marca o início da mensagem e é usado para sincronizar no barramento após ficar ocioso.
- Identificador: Valor que estabelece a prioridade da mensagem, quanto menor o valor maior a prioridade da mensagem.
- RTR: Único bit dominante - nível lógico alto - para pedido de transmissão remota. Todos os nós recebem o pedido, porém o campo Identificador determina o nó específico. A resposta também chega em todos os nós.
- IDE: Único bit dominante que significa que o campo Identificador não segue o padrão estendido.
- r0: Bit reservado.
- DLC: Dado de 4 bits que contêm o número em bytes do dado que está sendo transmitido.
- Dado: Dado da aplicação a ser transmitido que pode chegar até 64 bits.
- CRC: Teste de redundância cíclica de 16 bits para detecção de erro.
- ACK: Todo nó que receber a mensagem livre de erros substitui o bit recessivo desse campo por um bit dominante. Caso o receptor perceber algum erro, ele deixa o bit como está e descarta a mensagem. O nó transmissor repete a mensagem após a arbitragem das prioridades. ACK possui 2 bits, um para escrita e outro como delimitador.
- EOF: 7 bits que indicam o final do quadro (mensagem) e verificam o erro de bit *stuffing*.
- IFS: 7 bits que indicam o tempo necessário para o controlador mover o quadro (mensagem) para a memória (aplicação).

O CAN estendido (2.0B) possui os mesmos campos do CAN 2.0A; porém, há algumas adições que são:

- SSR: Bit único que substitui o RTR na posição da mensagem como uma área reservada no formato estendido.
- IDE: Bit recessivo que indica a extensão do Identificador com acréscimo de 18 bits.
- r1: Bit reservado adicional.

A rede CAN utiliza um meio de transmissão onde é avaliada a diferença de tensão entre dois fios (i.e., CANH e CANL). O estado recessivo é mantido por dois resistores de *pull-up* conectados a esses dois fios. Assim, sem que seja transmitida nenhuma informação, a rede receberia um número constante de bits recessivos. Portanto, torna-se necessário um estado lógico diferente para marcar o começo de uma transmissão [7, 8, 6].

O acesso ao barramento é um evento dirigido e ocorre de maneira aleatória; mas, se dois nós tentarem acessar o barramento simultaneamente, o acesso é implementado através da arbitragem lógica não destrutiva, fazendo com que o nó vencedor continue transmitindo a mensagem, sem destruí-la ou corrompê-la [7, 8].

A atribuição de prioridade às mensagens é o recurso que torna o CAN particularmente atrativo para sistemas em tempo real. Quanto menor for o valor do identificador, maior é a prioridade da mensagem. Caso dois nós tentarem acessar o barramento simultaneamente, caso um deles enviar zero (dominante) e o outro enviar um (recessivo), o primeiro nó ganha acesso ao barramento e completa a mensagem: bits dominantes sempre sobrescrevem bits recessivos no barramento CAN [7].

Existem quatro tipos de mensagens que podem ser transmitidas no CAN [7]:

1. Mensagem de dado: o tipo de mensagem mais comum, que inclui o campo de arbitragem, campo de dado, campo RCR e o campo ACK.
2. Mensagem remota: o propósito da mensagem remota é solicitar dado de outro nó da rede. A mensagem remota é similar a mensagem de dado, com a diferença que o campo RTR contém um bit recessivo e o campo de dados é vazio.
3. Mensagem de erro: mensagem especial que viola o formato da mensagem CAN. É transmitida quando um nó detecta um erro em uma mensagem. Assim, o transmissor original da mensagem a retransmite.
4. Mensagem de sobrecarga: similar à mensagem de erro no quesito formato, sendo transmitida quando um nó fica ocupado. É usada principalmente para fornecer um atraso extra entre mensagens.

Uma mensagem é considerada sem erro quando o último bit no campo EOF é recebido como bit recessivo livre de erro. Um bit dominante no campo EOF faz o transmissor reenviar a mensagem. O protocolo CAN possui cinco métodos de checagem de erro: três em nível de mensagem e dois em nível de bit. Se uma mensagem falhar em um desses métodos, ela não é aceita e uma mensagem de erro é gerada pelos nós receptores e, como o CAN trabalha em *broadcast*, qualquer nó pode gerar uma mensagem de erro quando perceber algo errado com a mensagem, forçando o transmissor a reenviar a mensagem até que ela seja aceita [7, 8, 6].

3 Protocolo CANopen

CANopen é um protocolo de comunicação de alto nível, baseado no protocolo CAN, e padroniza a comunicação entre dispositivos e aplicações da rede. O CANopen trata do endereçamento, roteamento, confiabilidade fim-a-fim, sincronização, padronização e representação dos dados. A camada de aplicação é responsável pela descrição de como configurar, transferir e sincronizar os dispositivos da rede [9].

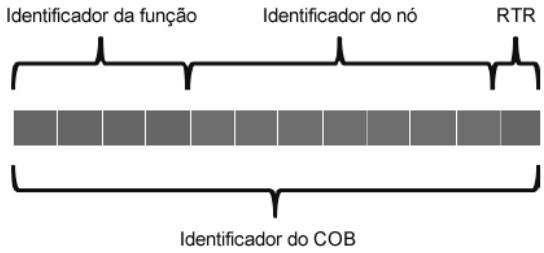
A rede CANopen deve possuir um mestre, que é responsável por gerenciar a rede. Uma rede pode possuir até 127 escravos (nós). Todo nó da rede possui uma lista de objetos denominada de dicionário de objetos, que contém objetos de comunicação (COB) responsáveis pela comunicação entre dispositivos da rede. Esses objetos são: Objeto de serviço de dados, responsável pelo acesso direto ao dicionário de objetos de um dispositivo da rede; Objeto de processamento de dados, usado para acessar os dados de um dispositivo; Objeto de emergência, responsável pelo envio de mensagens para indicar a ocorrência de erros no dispositivo; Objeto de sincronização, que permite a um dispositivo enviar uma mensagem de sincronização para toda a rede, periodicamente; Objeto de gerenciamento de rede, para o mestre da rede gerenciar os seus serviços de controle do dispositivo e serviço de controle de erros nos nós da rede [9, 3].

O protocolo CANopen utiliza a mensagem padrão do CAN com o campo Identificador dividido em duas partes [3, 10]: a primeira, representada por quatro bits, é usada para a identificação da função; a segunda, com sete bits, é usada para identificação do nó. A união dessas duas partes mais o campo RTR resulta no Identificador de Objetos de Comunicação (COB-ID), ilustrado na Figura 5.

Figura 4: Mensagem completa do protocolo CANopen

SOF	Identificador da função	Identificador do nó	*RTR*	Reservado RB1,RB0	*DLC*	DADO	*CRC*	*ACK*	*EOF*	NOME DO CAMPO
1	4	7	*1*	2	4	8x8	*16*	*2*	*7*	BITS/CAMPO
---	{0 ... 16}	{0 ... 127}	{0 ... 1}	---	{0 ... 7}	{0 ... 255}x8	---	---	---	FAIXA DE VALOR

Figura 5: Campo Identificador (em bits) modificado pelo protocolo CANopen



4 Protocolo DeviceNet

O protocolo DeviceNet atua nas camadas superiores do protocolo CAN, adotando um modelo produtor-consumidor ele suporta múltiplos modos de comunicação. Utilizando o método de prioridades de mensagens, ele possui métodos de detecção de endereços duplicados e isolamento de nós em caso de erros críticos. O sistema opera com a arquitetura distribuída ponto a ponto, mas também pode ser configurado para operar numa arquitetura mestre-escravo. A rede suporta até 64 dispositivos, endereçados de 0 a 63 [4].

As mensagens numa rede DeviceNet são definidas em dois tipos, I/O (dados de processo) e *explicit* (configuração e parametrização), cada um deles é adequado a um tipo de dado [4, 2]. Os principais tipos de mensagens I/O são: *Polled*, mensagem que o mestre envia solicitando dados para cada um de seus escravos; *Bit-strobe*, mensagem de oito bytes de dados enviada pelo mestre. onde cada bit desses 8 bytes identifica um escravo que, por sua vez, responde ao mestre de acordo com o que foi programado; *Change of state*, mensagem que é enviada somente quando há uma troca de estado ou quando um período de tempo previamente programado é atingido; *Cyclic*, que se tratam de mensagens trocadas em intervalos regulares, independente de haver mudanças de estado ou não em um dispositivo.

Mensagens *explicit* são de uso geral e não prioritário, utilizadas principalmente em tarefas assíncronas tais como configuração e parametrização do equipamento. Uma rede DeviceNet utiliza somente dois tipos de mensagens padronizadas pelo protocolo CAN: a mensagem de dados e a mensagem de erro [4].

O protocolo DeviceNet requer que uma conexão seja estabelecida antes de haver troca de dados, o estabelecimento de conexões é feito com mensagem do tipo *explicit*, e a seguir é usado o tipo de mensagem I/O para a troca de dados, as informações necessárias para que uma conexão seja estabelecida são armazenadas em um arquivo de configuração presente em cada nó da rede [4].

5 Protocolo *Smart Distributed System* - SDS

O protocolo *Smart Distributed System* (SDS) é baseado no protocolo CAN e possui uma arquitetura em três camadas: a camada física, de enlace e a de aplicação. Essa redução de camadas torna o SDS adequado para aplicações de controle de tempo real [5]. Os serviços oferecidos pela camada de aplicação são: **Leitura** de um atributo de um Objeto Embarcado da rede; **Escrita** de um atributo de um Objeto Embarcado; **Evento**, para reportar um evento especificado para um Objeto Embarcado; **Ação**, usado para executar ações especificadas para um Objeto Embarcado; **Mudança de Estado** para *ON/OFF*, usado para reportar uma mudança de estado de um dispositivo lógico; **Escrita *ON/OFF***, para ativar ou desativar um dispositivo lógico da rede; **Conexão**, usado para

Figura 6: Visão geral do endereçamento dos componentes físicos

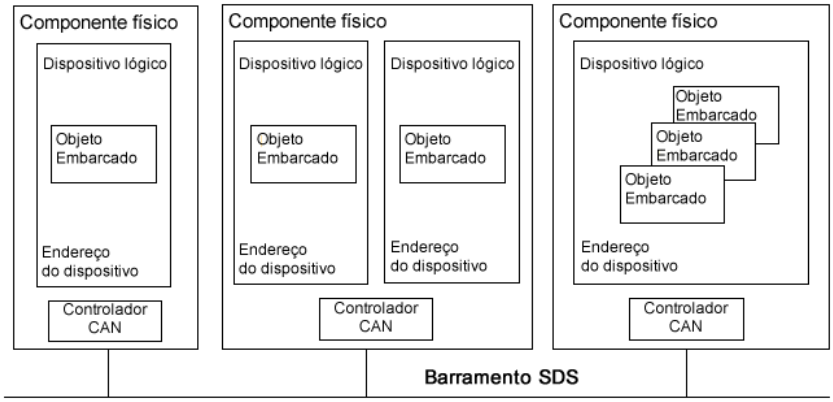


Figura 7: Mensagem SDS: formato curto

	Cabeçalho SDS									
	Cabeçalho CAN							CAN Footer		
SOF	Dir/Pri	Endereço lógico	Tipo de serviço	*RTR*	Reservado RB1,RB0	*DLC*	*CRC*	*ACK*	*EOF*	NOME DO CAMPO
1	1	7	3	*1*	2	4	*16*	*2*	*7*	BITS/CAMPO
---	{0,1}	{0 ... 125}	{0 ... 7}	{0}	---	{0}	---	---	---	FAIXA DE VALOR

estabelecer uma conexão ponto a ponto entre dois dispositivos; **Canal**, para conexão *multicast* e conexão ponto a ponto bidirecional para a comunicação entre dois ou mais dispositivos da rede [5].

Uma rede SDS é capaz de suportar até 126 dispositivos, endereçados de 0 a 125. As informações necessárias para a execução dos serviços de aplicação são armazenadas em cada dispositivo que, por sua vez, pode filtrar ou ignorar mensagens de um ou vários endereços. A rede possui Componentes Físicos, que podem ter um ou mais Dispositivos e, cada um destes, pode ter um ou mais Objeto Embarcado, como ilustrado na Figura 6.

O SDS utiliza o quadro de mensagem do CAN padrão. A partir da mensagem do CAN, o SDS classifica dois tipos de mensagens usadas na rede: Mensagem Curta e Mensagem Longa. Um quadro do protocolo SDS se diferencia do CAN padrão dividindo o campo Identificador de 11 bits em três partes. Se o bit mais significativo do campo Direção/Prioridade for setado em 1, significa que o campo de endereço contém o endereço de origem. Se for setado em 0, significa que o campo de endereço contém o endereço de destino. Consequentemente, mensagens que endereçam o campo com o endereço de destino tem maior prioridade [5]. Os próximos sete bits do Identificador são para o Endereçamento Lógico; assim, uma rede pode suportar até 125 nós (os nós 126 e 127 são reservados pela especificação CAN). Os últimos três bits do campo Identificador são para o Tipo de Serviço que, no tipo de mensagem Curta, são: Mudança de Estado para *ON/OFF* e Escrita *ON/OFF*. O campo RTR não é usado e o campo DLC indica o tamanho do dado transmitido. Para mensagens do tipo curto esse campo é sempre zerado.

Mensagens Longas, ao contrário das curtas, contêm dados [5]. O campo Direção/Prioridade é definido de acordo com o tipo de serviço presente na mensagem. Para serviços do tipo Leitura, Escrita, Ação, Evento e Conexão, o comportamento do campo Direção/Prioridade é idêntico ao descrito na mensagem do tipo Curta. Para o serviço Canal, o endereço lógico sempre será o endereço de origem e o bit do campo Direção/Prioridade é usado como um segundo nível de prioridade: zero indica alta prioridade e um define baixa prioridade. O campo DLC é usado para indicar o tamanho do dado da mensagem em bytes. O primeiro byte de dados da mensagem é dividido em duas partes: Especificadores de Serviço, com três bits, e Identificador de Objeto Embarcado (EOID), com cinco bits.

O campo Especificadores de Serviço consiste em dois subcampos: Pedido/Resposta (dois bits) e Indicador

Figura 8: Mensagem SDS: formato longo

	Cabeçalho SDS									DADO SDS				
	Campo de arbitragem do CAN				Controle CAN		DADO CAN				CAN Footer			
SOF	Dir/Pri	Endereço lógico	Tipo de serviço	*RTR*	Reservado RB1,RB0	*DLC*	Especificador de serviço	EOID	Parâmetros de serviço	DADO	*CRC*	*ACK*	*EOF*	NOME DO CAMPO
1	1	7	3	*1*	2	4	3	5	8	8x6	*16*	*2*	*7*	BITS/CAMPO
---	{0,1}	{0 ... 125}	{0 ... 7}	{0}	---	{0 ... 7}	{0 ... 7}	{0 ... 31}	{0 ... 255}	{0 ... 255}x6	---	---	---	FAIXA DE VALOR

de Fragmentação (um bit). O subcampo Pedido/Resposta especifica se a mensagem é pedido, resposta bem-sucedida, resposta de erro ou esperar resposta. O subcampo Indicador de Fragmentação indica se a mensagem é fragmentada (valor um) ou não fragmentada (valor zero). O campo EOID indica endereços de objetos embarcados específicos em um dispositivo da rede SDS. Um dispositivo pode ter um ou mais objetos embarcados, como ilustarado na Figura 6. O segundo byte de dados da mensagem representa o campo Parâmetros de Serviço. Quando o serviço da mensagem é Leitura ou Escrita, o campo representa o identificador do atributo de um objeto embarcado a ser lido ou modificado. Se o serviço da mensagem for Ação ou Evento, o campo representa o identificador da Ação ou Evento, especificados em cada nó da rede. Com essas modificações na mensagem padrão do protocolo CAN, a mensagem possui 6 bytes de dados disponíveis e, caso a mensagem for fragmentada, a mensagem possuirá quatro bytes de dados, uma vez que dois bytes representarão o número de fragmentações e o tamanho total da mensagem, respectivamente.

6 Configuração dos protocolos

Nesta sessão serão abordadas as configurações escolhidas para cada protocolo apresentado anteriormente. Como já foi mencionado, o protocolo CAN atua nas camadas física e de enlace. Os protocolos CANopen, DeviceNet e SDS atuam acima dessas camadas, oferecendo serviços de endereçamento, sincronização, confiabilidade fim-a-fim, padronização e representação dos dados.

6.1 Configuração do protocolo CANopen

O CANopen possui somente a configuração mestre-escravo. O mestre envia uma mensagem de requisição para um escravo, uma interrupção é gerada no nó destino e uma mensagem de resposta é enviada ao mestre da rede. Esse processo é repetido para todos os nós da rede de forma sequencial. Todo o processo está dentro de um laço infinito, conforme descrito nos Algoritmos 1 e 2.

Algoritmo 1: Algoritmo de endereçamento e comunicação do CANopen - **Mestre**

```
1 enquanto verdadeiro faça
2     para cada nó ativo na rede faça
3         enderecoTx ← nos[noAtual] + dicionario[processamentoDados];
4         enviaMensagem(enderecoTx, dado);
5         repita
6             recebeMensagem(enderecoRx, dado);
7             se enderecoRx = enderecoTx então
8                 aceitaDado(dado);
9         até aceitar resposta;
```

6.2 Configuração do protocolo DeviceNet

A configuração escolhida para o protocolo DeviceNet foi a mestre-escravo, para equiparar à configuração dos outros protocolos. Utilizando mensagens do tipo *Polled*, o protocolo DeviceNet se comporta da seguinte forma:

Algoritmo 2: Algoritmo de endereçamento e comunicação do CANopen - **Escravo**

```
1  Interrupçao verdadeiro faça
2      recebeMensagem(enderecoRx, dado);
3      se mensagem recebida então
4          se (enderecoRx − ID) = dicionario[processamentoDados] então
5              enderecoTx ← enderecoRx;
6              enviaMensagem(enderecoTx, dado);
7  início
8      Configura mascara para aceitar qualquer padrão de mensagem;
9      Configura filtro para receber mensagem somente do mestre;
10     Configura pinos do microcontrolador;
11     enquanto verdadeiro faça
12         dado ← leSensor();
13         tratamentoDado(dado);
```

o mestre envia uma mensagem de requisição de dados para um ou mais nós da rede; cada nó da rede verifica se é um alvo do mestre; caso afirmativo, o nó envia uma mensagem de resposta. Posteriormente, o mestre envia uma mensagem de confirmação, fazendo com que o escravo não envie mais a mensagem até receber uma nova mensagem de pedido. O processo continua nesse laço enquanto a rede estiver em funcionamento, como descrito nos Algoritmos 3 e 4.

Algoritmo 3: Algoritmo de endereçamento e comunicação do DeviceNet - **Mestre**

```
1  enquanto verdadeiro faça
2      enderecoTx ← idMestre;
3      enviaMensagem(enderecoTx, alvo);
4      controle ← contagem(alvo);
5      nosAux ← nos;
6      enquanto controle > 0 faça
7          recebeMensagem(enderecoRx, dado);
8          se mensagem recebida então
9              para cada nó ativo na rede faça
10                 se nosAux[noAtual] = enderecoRx então
11                     nosAux[noAtual] ← 0;
12                     controle − −;
13                     dadoConfirma[noAtual] ← confirmado
                       enviaMensagem(enderecoTx, dadoConfirma);
```

6.3 Configuração do protocolo SDS

Similar ao CANopen, o protocolo SDS tem a configuração mestre-escravo. O mestre envia uma mensagem para um escravo, espera a resposta da mensagem, e envia uma outra mensagem para outro escravo da rede. O processo é realizado até o mestre se comunicar com todos os escravos da rede. O que diferencia o SDS do CANopen é um bit no endereçamento chamado Direção, sendo atribuído valor um a esse bit sempre que o mestre envia uma mensagem de requisição de dados, e valor zero sempre que um escravo for se comunicar com o seu mestre (i.e., responder ao mestre ou enviar uma mensagem de evento). Esse bit no endereçamento é importante porque força as mensagens escravo-mestre terem maior prioridade que as mensagens mestre-escravo. O comportamento do protocolo SDS é descrito nos Algoritmos 5 e 6.

Algoritmo 4: Algoritmo de endereçamento e comunicação do DeviceNet - **Escravo**

```
1  Interrupçao verdadeiro faça
2      recebeMensagem(enderecoRx, dadoRx); se dadoRx[ID] = verdadeiro então
3          enderecoTx ← ID;
4          enviaMensagem(enderecoTx, dadoTx);
5          confirmacao ← falso;
6          enquanto confirmacao = falso faça
7              recebeMensagem(enderecoRx, dadoRx);
8              se dadoRx[ID] = confirmado então
9                  confirmacao ← verdadeiro;
10             senão
11                 enviaMensagem(enderecoTx, dadoTx);
12 início
13     Configura mascara para aceitar qualquer padrão de mensagem;
14     Configura filtro para receber mensagem somente do mestre;
15     Configura pinos do microcontrolador;
16     enquanto verdadeiro faça
17         dadoTx ← leSensor();
18         tratamentoDado(dadoTx);
```

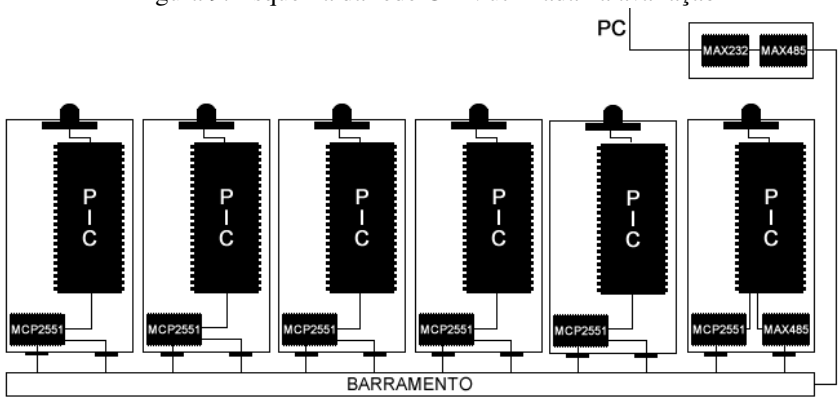
Algoritmo 5: Algoritmo de endereçamento e comunicação do SDS - **Mestre**

```
1  enquanto verdadeiro faça
2      para cada nó ativo na rede faça
3          enderecoTx ← nos[noAtual] + funcao[lerDados];
4          enderecoTx ← endecoTx + Direcao;
5          enviaMensagem(enderecoTx, dado);
6          repita
7              recebeMensagem(enderecoRx, dado);
8              se enderecoRx = enderecoTx então
9                  aceitaDado(dado);
10         até aceitar resposta;
```

Algoritmo 6: Algoritmo de endereçamento e comunicação do SDS - **Escravo**

```
1  Interrupçao verdadeiro faça
2      recebeMensagem(enderecoRx, dado);
3      se mensagem recebida então
4          se ((enderecoRx − ID) − Direcao) = funcao[lerDados] então
5              enderecoTx ← enderecoRx;
6              enviaMensagem(enderecoTx, dado);
7  início
8      Configura mascara para aceitar qualquer padrão de mensagem;
9      Configura filtro para receber mensagem somente do mestre;
10     Configura pinos do microcontrolador;
11     enquanto verdadeiro faça
12         dado ← leSensor();
13         tratamentoDado(dado);
```

Figura 9: Esquema da rede CAN utilizada na avaliação



7 Ambiente de testes e aplicação

Nessa seção serão descritos uma rede física, que é o ambiente de testes dos protocolos, e a aplicação que simula um ambiente de monitoramento de sensores. Para realizar os testes dos protocolos foi implementada uma rede de sensores com seis nós. Cada nó da rede possui um microcontrolador modelo dsPIC30F4013, um *transceiver* CAN que promove a ligação do controlador CAN com o barramento físico, modelo MCP2551 (ambos da *Microchip Technology*^{TM}) [11, 12]. Adicionalmente, utiliza-se um potenciômetro para emular a entrada de dados pelos sensores.

Apenas um dos seis nós possui um MAX485 (desenvolvido pela *Maxim Integrated*^{TM} [13]) que é um *transceiver* para prover a comunicação na interface RS-485. O MAX485 é conectado a outro, e esse segundo é conectado a um MAX232 (desenvolvido pela *Texas Instruments*^{TM} [14]) que é um circuito integrado capaz de converter sinais de uma porta serial para a sinalização compatível com os microcontroladores. O MAX232 está presente no nó mestre e tem como finalidade a comunicação da rede CAN com um PC. O microcontrolador foi configurado para trabalhar com um *clock* de 20 Mhz, e tem suporte a um módulo CAN e dois módulos UART, para comunicação serial. O microcontrolador tem 1024 bytes de memória EEPROM e 40 pinos, dos quais 30 são de I/O. A Figura 9 ilustra o esquema da rede de sensores.

A aplicação usada para realizar os testes simula um cenário onde o mestre, para saber o valor de cada sensor da rede, envia uma mensagem de requisição aos seus escravos. Os nós escravos realizam a leitura do valor de saída do potenciômetro e fazem o tratamento do dado. Ao notar uma troca de sinal no pino do microcontrolador responsável pela leitura do barramento CAN, uma interrupção é gerada. Essa interrupção é responsável por enviar o valor, após feito o tratamento do valor absoluto, para o mestre da rede.

8 Avaliação de desempenho

O desempenho de sistemas distribuídos pode ser influenciado por alterações nos seus parâmetros de configuração como taxa de transmissão dos dados, tamanho das mensagens, prioridade das mensagens e período de amostragem dos dispositivos [15]. Para balizar a avaliação de desempenho, utilizou-se as seguintes métricas:

1. *Turnaround Time*: o tempo necessário para satisfazer um pedido (as medições serão realizadas a partir do momento que o mestre envia uma mensagem de requisição até o momento que ele recebe a resposta do último nó da rede).
2. *Overhead*: medida da carga de dados auxiliares necessários à transmissão de dados de interesse real da aplicação (medidas nos sentidos mestre-escravo e escravo-mestre).
3. Ocupação do Barramento: percentual de ocupação do barramento.
4. *Throughput*: vazão, refletindo a quantidade de dados transmitidos em cada processo de requisição.

Figura 10: *Turnadound Time* sem tempo de processamento

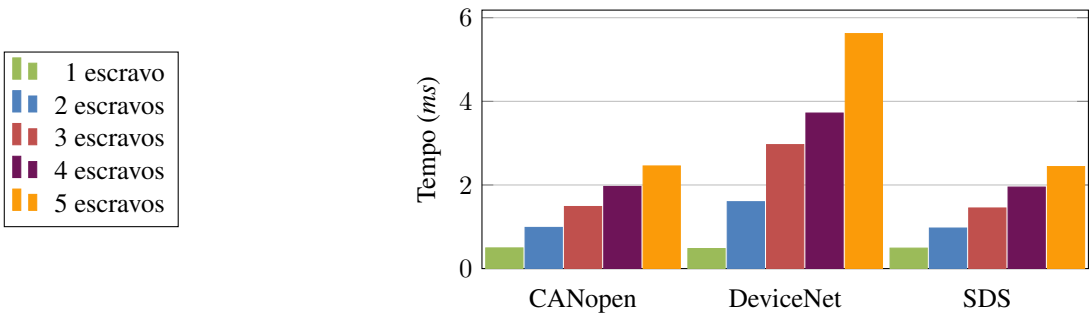
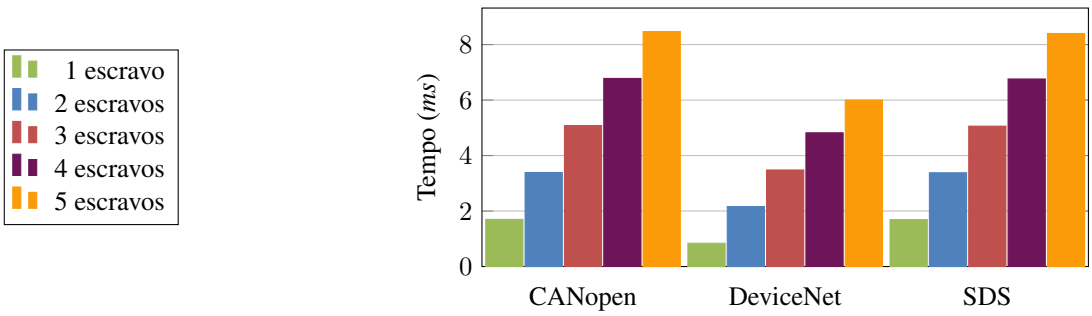


Figura 11: *Turnadound Time* com tempo de processamento



Para se obter os dados do *Turnaround* e Ocupação do Barramento utilizou-se um osciloscópio *tektronix*™(modelo TBS1062). Um pino do microcontrolador foi configurado para receber valor um (sinal lógico alto) sempre que o mestre desse início ao processo de pedido de dados, e receber zero (sinal lógico baixo) durante o processo de envio e recebimento das mensagens. O objetivo foi ter um pico de sinal no pino sempre que o mestre desse início aos pedidos. Um atraso de 5 *ms* foi adicionado entre a mudança de sinal no pino (tempo estabelecido como *Turnaround Time* no osciloscópio).

No primeiro cenário para medir o *Turnaround* (Figura 10), avaliou-se cinco configurações para cada protocolo variando-se de 1 a 5 escravos. Nessa avaliação, os escravos não realizam o tratamento do dado: assim que ocorre uma interrupção, o escravo transmite o dado do sensor para o mestre. Os resultados apresentados no gráfico desconsideram os 5 *ms* de atraso adicionados ao processo.

Os resultados do segundo cenário, agora com cada escravo simulando um tempo de processamento, são apresentados na Figura 11. Para simular o tempo de processamento, cada escravo incrementa uma variável de 0 a 1000. Vale ressaltar que para este cenário os 5 *ms* também já foram subtraídos dos resultados.

É possível observar que no primeiro cenário, onde não há processamento de dados no escravo, os protocolos CANopen e SDS têm resultados melhores que o protocolo DeviceNet. Porém, no segundo cenário, o protocolo DeviceNet tem melhor desempenho em relação aos demais. Isso ocorre em decorrência do DeviceNet permitir o processamento paralelo em seus escravos, uma vez que a mensagem de pedido atinge todos os escravos simultaneamente. Já no caso do CANopen e SDS, o mestre envia uma mensagem individual a cada escravo, aguardando resposta para só após enviar ao próximo escravo.

Conforme descrito anteriormente, o microcontrolador empregado no sistema utiliza o padrão de mensagem CAN 2.0B. Considerando-se o melhor caso, ou seja, sem erros de transmissão, o *Overhead* no sentido mestre-escravo é:

- Protocolo CANopen: 51 bits.
- Protocolo DeviceNet: 116 bits.

Figura 12: Ocupação do barramento (em *ms*) utilizando cinco escravos

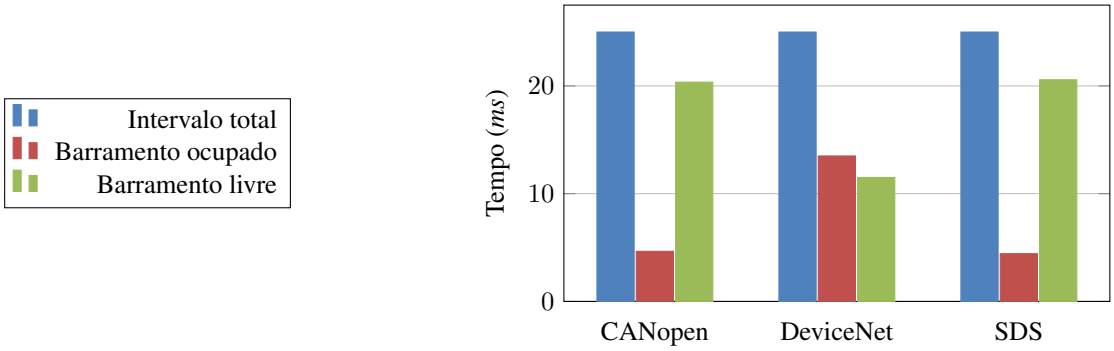


Tabela 1: Vazão em Kbps para cada protocolo

Protocolo	Vazão (Kbps)
CANopen	64,3
DeviceNet	178,3
SDS	64,5

- Protocolo SDS: 51 bits.

A diferença considerável entre o protocolo DeviceNet e os demais se dá porque o protocolo DeviceNet utiliza o campo direcionado aos dados para fazer controle, assim os bits do campo dados somam-se à carga de controle da mensagem. Para mensagens transmitidas no sentido escravo-mestre, o *Overhead* dos três protocolos é equivalente e corresponde a 50 bits.

Nos protocolos CANopen e SDS é subtraído um bit, que é o campo RTR, responsável por sinalizar que a mensagem é um pedido. Como mensagens escravo-mestre são mensagens de resposta o bit RTR é desconsiderado. O mesmo acontece com o protocolo DeviceNet, que além de desconsiderar o bit RTR também não soma os bits relacionados aos dados que, nas mensagens de respostas, contêm informação útil à aplicação.

Em relação a ocupação do barramento (Figura 12), os protocolos CANopen e SDS apresentam resultados similares. Em um intervalo de 20 *ms* os protocolos CANopen e SDS mantiveram o barramento ocupado por $4,65$ e $4,43$ milissegundos, respectivamente. Um percentual de $23,25\%$ para o CANopen e $22,15\%$ para o protocolo SDS. No protocolo DeviceNet, em um mesmo intervalo de tempo, o barramento manteve-se ocupado por $13,5$ *ms* (um percentual de $67,5\%$).

A ocupação do barramento no DeviceNet é maior porque ele utiliza mais bits de controle nas suas mensagens e, além da mensagem de pedido enviada pelo mestre, existem também mensagens de confirmação que o mestre envia após receber uma resposta. Nota-se que, enquanto um escravo não receber a mensagem de confirmação, ele continua escrevendo a mensagem de resposta no barramento.

Tendo-se como base os resultados da métrica *Turnaround Time*, calculou-se a vazão da seguinte forma:

$$throughput = \frac{\sum_{n=1}^{5} \frac{bits\_enviados_n + bits\_recebidos_n}{turnaround_n}}{5} \quad (1)$$

Onde: $bits\_enviados_n$ é a quantidade de bits que o mestre envia para o escravo n; $bits\_recebidos_n$ é a quantidade de bits que o mestre recebe como resposta do escravo n; $turnaround_n$ é o resultado da métrica *Turnaround Time* correspondente ao escravo n.

A Tabela 1 apresenta os resultados obtidos para a vazão dos protocolos. Observa-se uma semelhança entre os resultados dos protocolo CANopen e SDS, notando-se uma vantagem do protocolo DeviceNet pois este apresentou melhores resultados no *Turnaround Time* e manteve o barramento ocupado por mais tempo.

9 Conclusões

Em aplicações onde não é necessário tratamento de dados nos escravos (i.e., os escravos só precisam fazer a leitura de um sensor e enviar o valor para o mestre quando requisitados), os protocolos CANopen e SDS se destacam. Além de ter melhores resultados em termos de atraso, eles possuem um percentual menor de ocupação do barramento (por adotar uma estratégia mais simples, empregam menos dados de controle).

No entanto, para aplicações que fazem tratamento de dados nos escravos, o protocolo DeviceNet obteve melhores resultados, apesar de ter mais dados de controle e taxa de ocupação do barramento maior que os outros dois protocolos. Devido à sua estratégia *Polled*, todos os escravos recebem a mensagem de pedido do mestre simultaneamente, tornando o tratamento de dados nos escravos um processo paralelo, enquanto o tratamento de dados nos protocolos CANopen e SDS é sequencial.

Referências

[1] GALLOWAY, B.; HANCKE, G. P. Introduction to industrial control networks. *Communications Surveys & Tutorials, IEEE*, IEEE, v. 15, n. 2, p. 860–880, 2013.

[2] BRADY, J. Networking with devicenet. *The Computer Applications Journal*, 1998. Circuit Cellar Incorporated.

[3] WEG, E. *Comunicação CANopen*. 2010. <http://ecatalog.weg.net/files/wegnet/WEG-plc300-comunicacao-canopen-10000849433-manual-portugues-br.pdf>. [Online; Acesso em: 19 de julho de 2014].

[4] WEG, E. *Comunicação DeviceNet*. 2008. <http://ecatalog.weg.net/files/wegnet/1-2195.pdf>. [Online; Acesso em: 20 de julho de 2014].

[5] HONEYWELL. *SDS specification*. 1995. Honeywell Inc.Microswitch Division, GS 052 103 ... GS 052 107. 1995.

[6] BOSH, R. *CAN specification version 2.0*. 1991. Rober Bousch GmbH, Postfach. 1991.

[7] CORRIGAN, S. *Introduction to the Controller Area Network (CAN)*. Dallas, Texas: [s.n.], 2008. Texas Instruments, Application Report, SLOA101A.

[8] BARBOSA, L. R. G. Rede can. *Escola de Engenharia da UFMG. Universidade Federal de Minas Gerais, Belo Horizonte*, 2003.

[9] BOTERENBROOD, H. *CANopen high-level protocol for CAN-bus*. 2000. NIKHEF, Amsterdam. 2000.

[10] NATIONAL, I. *The Basic of CANopen*. 2013. <http://www.ni.com/white-paper/14162/en/>. [Online; Acesso em: 19 de julho de 2014].

[11] TECHNOLOGY, I. M. *dsPIC30F3014, dsPIC30F4013 Data Sheet*. 2006. <http://ww1.microchip.com/downloads/en/devicedoc/70138c.pdf>. [Online; Acesso em: 26 de novembro de 2015].

[12] TECHNOLOGY, I. M. *MCP2551 - High-Speed CAN Transceiver*. 2010. <http://ww1.microchip.com/downloads/en/DeviceDoc/21667f.pdf>. [Online; Acesso em: 26 de novembro de 2015].

[13] PRODUCTS, I. M. I. *Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers*. 2014. <http://datasheets.maximintegrated.com/en/ds/MAX1487-MAX491.pdf>. [Online; Acesso em: 26 de novembro de 2015].

[14] INSTRUMENTS, I. T. *MAX232x Dual EIA-232 Drivers/Receivers*. 2014. <http://www.ti.com/lit/ds/symlink/max232.pdf>. [Online; Acesso em: 26 de novembro de 2015].

[15] LIAN, F.-L.; MOYNE, W.; TILBURY, D. Network design consideration for distributed control systems. *Control Systems Technology, IEEE Transactions on*, IEEE, v. 10, n. 2, p. 297–307, 2002.