Um injetor de falhas de comunicação para implementações do protocolo EtherCAT

Luiz Gustavo A. Gomes[1], João de Moraes[1], Taisy S. Weber[1], Sérgio L. Cechin[1], João Netto[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

{lgagomes, joao.moraes, taisy, cechin, netto }@inf.ufrgs.br

Abstract. *Computational systems are frequently used in many industrial areas by communicating geographically distant devices. Ethernet-based protocols, like EtherCAT, are commonly adopted for this communication to take place. By having a specification that does not dictate how a new EtherCAT device must be implemented, for a new product, a new coding must be made and validated. This work introduces a communication fault injector for the EtherCAT protocol, aiming the process of validation of new implementations against the occurrence of failures. In this article, we present a functional injector capable of injecting faults into commercial products that implement EtherCAT.*

Resumo. *Sistemas computacionais são frequentemente usados em diferentes áreas da indústria, comunicando dispositivos geograficamente distantes. Protocolos baseados em Ethernet, como o EtherCAT, são comumente adotados para que essa comunicação seja realizada. Por ter uma especificação que não dita como um novo dispositivo EtherCAT deve ser implementado para um novo produto, uma nova codificação deve ser feita e validada. Este trabalho introduz um injetor de falhas de comunicação próprio para o protocolo EtherCAT, visando o processo de validação de novas implementações frente à ocorrência de falhas. Neste artigo nós apresentamos um injetor funcional capaz de injetar falhas em produtos comerciais que implementam EtherCAT.*

1. Introdução

Com sistemas de controle digital se comportando como uma grande rede interconectada e o custo reduzido de equipamentos com interface Ethernet, surgiu um interesse econômico em introduzir sistemas de comunicação baseadas em Ethernet no domínio industrial [Neumann 2007]. Portanto, soluções em comunicação baseadas em Ethernet foram desenvolvidas especialmente para atender requisitos industriais como tempo e sincronismo.

Uma destas soluções é o protocolo EtherCAT ["EtherCAT Technology Group | HOME" 2016], planejado para dar suporte ao processamento de informações, monitoramento e controle de sistemas de controle para diversos setores industriais de maneira simples. Como o padrão EtherCAT não define uma implementação padrão pré-certificada [Zhou e Hu 2011] para qualquer novo dispositivo, o protocolo deve ser implementado pelos desenvolvedores e sua validação feita pelo órgão certificador responsável (*EtherCAT Technology Group*). Ferramentas que possam auxiliar desenvolvedores a testarem seus códigos diante da ocorrência de falhas existem, porém, são ferramentas comerciais ou são ferramentas que não cobrem o modelo de falhas definido pela especificação. Com o auxílio de uma ferramenta de injeção de falhas, o

processo de validação torna-se mais rápido, visto que a implementação já terá sido testada na ocorrência de falhas antes de ser submetida ao órgão certificador.

Com base neste cenário, o objetivo deste trabalho é apresentar um injetor de falhas próprio para EtherCAT, para validar os mecanismos de tolerância a falhas de implementações deste protocolo. Tal injetor faz parte de uma arquitetura de validação que futuramente será utilizada pelo grupo de pesquisa no desenvolvimento de uma nova linha de produtos. A partir dos resultados obtidos e apresentados aqui, é possível afirmar que foi desenvolvida uma ferramenta inovadora, especialmente feita para o processo de validação de implementações de EtherCAT, customizável e que pode ser estendida a outros protocolos.

Injeção de falhas é um dos pilares deste trabalho devido à sua importância no processo de validação de sistemas, onde mecanismos de tolerância a falhas são aprimorados de modo a garantir a dependabilidade deste sistema em sua fase operacional [Arlat et al. 1990].

O resultado deste artigo é uma continuação do trabalho anterior do grupo de pesquisa [Gomes et al. 2016], onde detalhes iniciais do projeto são apresentados, bem como o injetor resultante que serviu como prova de conceito e pavimentou o caminho para este trabalho alcançar o estágio atual.

O artigo apresenta um breve resumo do protocolo EtherCAT, trabalhos relacionados, experimentos realizados, decisões de projeto e resultados alcançados.

2. Protocolo EtherCAT

EtherCAT é um protocolo industrial baseado em Ethernet desenvolvido para oferecer alta performance em tempo real, baixo *jitter* e uma alta utilização da banda disponível. É largamente utilizado no controle de servo motores, coleta de dados de forma sincronizada e na área de automação em geral [Zhou e Hu 2011].

Uma rede EtherCAT é composta de um mestre EtherCAT e até 65535 escravos que podem estar conectados sem qualquer restrição de topologia (linha, árvore, anel, estrela ou qualquer combinação destas), permitindo que quaisquer dois dispositivos estejam distantes em até 100 metros entre si.

Para permitir a comunicação entre o mestre e os escravos, EtherCAT encapsula seus dados em telegramas EtherCAT, que são transportados dentro de um *frame* Ethernet padrão, como mostra a Figura 1. Através do *EtherType* (0x88A4) [EtherCAT Technology Group 2015] é possível saber quando dados EtherCAT estão contidos dentro do *frame*. Cada telegrama EtherCAT é subdividido em um cabeçalho EtherCAT e um ou mais datagramas EtherCAT [EtherCAT Technology Group 2014], que indicam qual tipo de acesso o mestre deseja executar (leitura / escrita) e quais escravos devem realizar uma determinada tarefa [EtherCAT Technology Group 2015].

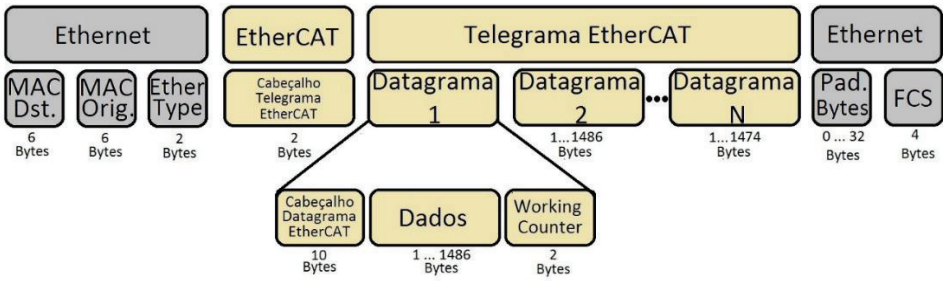

Figura 1. Estrutura de um *frame* Ethernet com dados EtherCAT

O início do ciclo de comunicação EtherCAT se dá com o mestre enviando um telegrama que passa por todos os nós escravos da rede. Cada escravo lê e insere seus respectivos dados no seu datagrama designado enquanto este está em trânsito. Como os *frames* não são colocados em um *buffer* para leitura ou escrita de dados, este é atrasado apenas pelos tempos de propagação do *hardware*. O último escravo a receber o *frame* detecta que não possui escravos subsequentes a quem possa repassá-lo e envia este *frame* no sentido contrário, em direção ao mestre, utilizando a característica *full duplex* da tecnologia Ethernet [EtherCAT Technology Group 2015]. Do ponto de vista de criação de *frames,* apenas o mestre pode criar e enviar *frames,* enquanto os escravos apenas podem realizar leituras e escritas nestes *frames* [EtherCAT Technology Group 2015].

2.1. Modelo de falhas de comunicação

Erros de comunicação podem ser causados pelos mais variados motivos, desde falhas de hardware até interferências do ambiente externo. As falhas consideradas na especificação do EtherCAT serão apresentadas a seguir.

Falhas de perda de pacotes ocorrem quando um ou mais pacotes não conseguem alcançar seu destino. Podem ser causadas, por exemplo, por congestionamento do enlace, fazendo com que o protocolo descarte pacotes. Entre outras causas, pode-se citar baixo desempenho de equipamentos físicos (roteadores, *switches*, etc.), mau funcionamento do *software* utilizado e rompimento do meio físico de transmissão.

Falhas por corrupção de dados ocorrem quando a mensagem que sai do dispositivo de origem é diferente da mensagem que chega ao dispositivo de destino. Pode ser causada por falhas em elementos físicos do sistema ou por erros de *software*.

Atraso de mensagens é observado quando restrições temporais não são respeitadas, isto é, o tempo esperado para um pacote ir de sua origem até seu destino é maior ou menor do que o esperado. No caso de EtherCAT apenas pacotes atrasados são considerados pela especificação, pois pacotes que chegam antes do tempo previsto não representam qualquer ameaça ao funcionamento normal.

Circulação de *frame*(s) indefinidamente pelo sistema acontece no caso de ruptura do enlace ou mau funcionamento de alguma porta do dispositivo escravo, fazendo com que uma topologia em linha seja dividida em duas. Um *frame* viajando através dos dispositivos pode começar a circular, ou seja, ser transportado apenas entre os nós de um lado da ruptura sem nunca alcançar o último escravo ou o mestre. Tal situação pode ser observada a seguir. Na Figura 2 (a) é exibido o caminho normal que um pacote percorre. Ao ocorrer uma falha de enlace entre os escravos 1 e 2 (Figura 2 (b)), estes fecham suas portas 1 e 0 respectivamente e *frames* à direita do escravo 2 seguirão circulando indefinidamente, pois estão impossibilitados de alcançar o nó mestre. Como não existe

um contador de *hops* em um telegrama ou datagrama EtherCAT, esta situação representa um risco real ao funcionamento da rede.

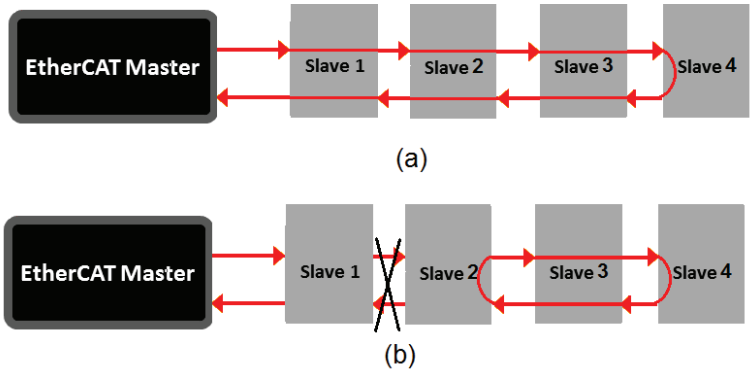

Figura 2. (a) Caminho normal percorrido pelos *frames*. (b) Caminho percorrido pelos *frames* após uma falha de enlace

2.2. Mecanismos de detecção e correção de falhas

A seguir são descritos os métodos do protocolo para lidar com as falhas apresentadas anteriormente.

A primeira medida é o *Frame Check Sequence* (FCS) presente no *frame* Ethernet. Este é verificado tanto pelo mestre quanto pelos escravos para determinar se um *frame* foi recebido corretamente. Como vários escravos podem alterar o conteúdo do *frame* durante o percurso, seu FCS é calculado por cada nó escravo tanto na sua recepção quanto na sua retransmissão. Se um erro de *checksum* é encontrado, o escravo não repara o FCS, mas sinaliza o mestre incrementando um contador de erros interno, de modo que o ponto da falha seja precisamente localizado na topologia.

O *Working Counter* é o último campo de um datagrama EtherCAT. Possui um valor esperado calculado pelo mestre e é incrementado pelos escravos a cada leitura ou escrita bem-sucedida realizada no datagrama, ou seja, se ao menos um *byte* ou um *bit* em todo o datagrama foi lido ou escrito com sucesso. Ao comparar o *Working Counter* com o número esperado de escravos que deveriam acessar dados, o mestre pode saber quantos escravos processaram seus dados correspondentes. Se o *Working Counter* presente no datagrama vindo dos escravos possui um valor diferente do esperado, o mestre não envia os dados deste datagrama para a aplicação [EtherCAT Technology Group 2015].

Escravos contam com dois *watchdogs* para monitorar comunicações malsucedidas: enquanto um monitora acessos de escrita feitos *no* escravo, outro monitora leituras e escritas bem-sucedidas feitas *pelo* escravo. Caso não ocorra nenhuma comunicação dentro do tempo previsto, o respectivo *watchdog* tem seu contador incrementado e seus sinais de saída não são entregues [EtherCAT Technology Group 2014].

Para lidar com *frames* que podem circular indefinidamente pelo sistema, existe o *Circulating Bit,* que é um *bit* para informar aos dispositivos da topologia quando um *frame* está circulando pela topologia na ocorrência de uma falha de enlace que cause uma situação semelhante à da Figura 2 (b). Para prevenir isto, um escravo sem um enlace em sua porta 0 fará o seguinte: se o *Circulating Bit* de um datagrama EtherCAT é 0, muda

seu valor para 1 e o envia adiante. Se o *Circulating Bit* é 1, o *frame* não é processado e sim descartado [EtherCAT Technology Group 2014].

3. Trabalhos relacionados

A seguir é apresentado um conjunto, não exaustivo, de ferramentas voltadas à verificação e teste em geral de implementações do protocolo EtherCAT. Por ser um protocolo aberto e sem uma implementação padrão para todos os dispositivos, ferramentas deste tipo são desejáveis para se avaliar a conformidade de uma implementação frente à especificação. As ferramentas descritas a seguir foram escolhidas por servirem, mesmo que parcialmente, a este propósito.

beStorm [“beSTORM® Software Security Testing Tool” 2016] é uma ferramenta voltada para avaliação de segurança que realiza uma análise exaustiva em busca de novas vulnerabilidades em aplicações que utilizam algum protocolo de rede. Ao gerar, automaticamente, ataques, *beSTORM* testa as aplicações quanto a sua segurança (*security*) antes que estes atinjam o mercado.

EC-STA [“EC-STA [EtherCAT Slave Test Application]” [S.d.]] é um *framework* desenvolvido para realização de testes para verificação do funcionamento durante e após o desenvolvimento de escravos EtherCAT, como alteração de estado e operações de escrita e leitura. Possui como componente principal uma aplicação baseada em .NET e escrita em linguagem C#. Também é fornecido o código fonte completo, permitindo ao usuário adicionar suas próprias funções. Seu funcionamento se dá configurando um computador com um mestre EtherCAT (fornecido separadamente) e conectando este ao escravo que será avaliado.

EtherCAT Conformance Test Tool[“EtherCAT Technology Group | EtherCAT Conformance Test Tool (CTT)” 2016] é uma ferramenta desenvolvida pelo próprio grupo responsável por manter e gerenciar o protocolo, (*EtherCAT Technology Group* – ETG) para realizar testes em dispositivos escravos EtherCAT a fim de verificar sua conformidade. Nesta ferramenta, *frames* EtheCAT são enviados ao dispositivo sob teste através de uma porta Ethernet padrão de modo a estimulá-lo. Por ter suporte e ser licenciada pelo grupo responsável pelo protocolo EtherCAT, é utilizada pelo órgão verificador como última etapa dos processos de validação e certificação e é tida como mandatória para que produtos que implementam EtherCAT possam ser certificados e assim liberados para atingir o mercado.

O uso da ferramenta *beStorm* para a verificação do protocolo EtherCAT é válido, mas por ser focada apenas em encontrar falhas de segurança, pode não ser mandatória quando se procura testar o correto funcionamento de uma dada implementação. Além disso, por ser uma ferramenta do tipo “caixa-preta”, não é sabido quais destes testes de segurança podem ser utilizados para identificar erros na implementação.

As ferramentas *EC-STA* e *EtherCAT Conformance Test Tool* são capazes de testar completamente uma implementação do protocolo EtherCAT de acordo com a especificação. Porém além de serem ferramentas pagas, são voltadas para o produto em seu estágio final de desenvolvimento, quando se está prestes a ser lançado no mercado, podendo não se adequar a testes realizados durante o processo de desenvolvimento. Além disso, os testes são realizados seguindo uma abordagem simples, através de estímulos nos dispositivos e comparação das respostas com resultados esperados, não atuando nas mensagens de comunicação.

4. O injetor de falhas TANATTOS

4.1. Visão geral

A empresa parceira no projeto pretende desenvolver uma nova linha de produtos, que implementam o protocolo EtherCAT. Para tal, é necessário que este novo equipamento passe por uma bateria de testes de campo em condições normais de operação e em condições de falhas antes de ser enviado para o processo de validação e certificação do equipamento pelo órgão responsável.

O uso de um injetor de falhas que ponha o produto à prova irá auxiliar a equipe na identificação de eventuais falhas de implementação. Com isso as chances de que novas implementações do protocolo EtherCAT sejam certificadas aumentará, pois torna possível identificar combinações de eventos que possam levar a comportamentos não desejados [Yu et al. 2005].

O injetor de falhas TANATTOS (*Tool for fAult iNjection on ethercAT implementaTiOnS*), foi desenvolvido para que implementações do protocolo EtherCAT possam ser testadas à medida que vão sendo construídas. Além disto, tal ferramenta foi planejada para contar com uma baixa intrusividade, ou seja, mínima interferência causada pelo ato de injetar as falhas. A Figura 3 exibe a janela principal do injetor, que conta com botões relativos às funções de injeção de falhas, abrir os arquivos de *log* pós-injeção das falhas e arquivo de ajuda, bem como informações referentes ao desenvolvedor.

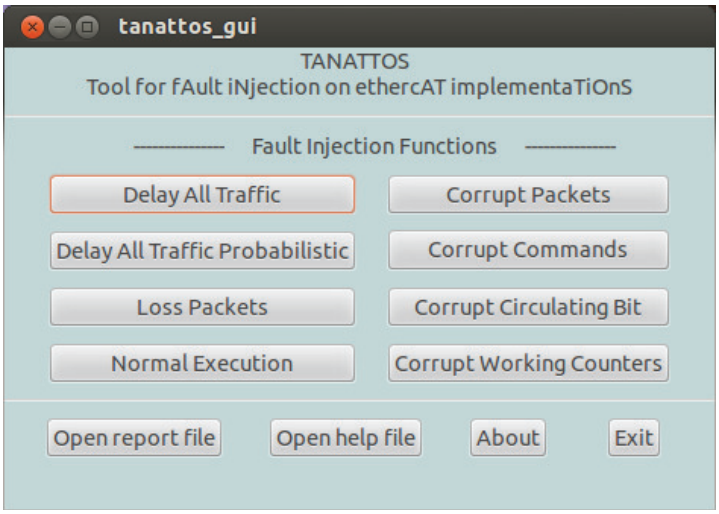

Figura 3. Janela principal do injetor TANATTOS

TANATTOS age simulando um ataque do tipo *man-in-the-middle*, ou seja, interceptando pacotes trocados entre dois dispositivos (Figura 4 (b)), sendo assim um componente independente do sistema. Do ponto de vista dos dispositivos mestre e escravo a comunicação flui diretamente sem interferências (Figura 4 (a)), quando na verdade esta comunicação passa através do injetor. Para isto, o injetor é executado em um computador exclusivo que conta com duas interfaces de rede.

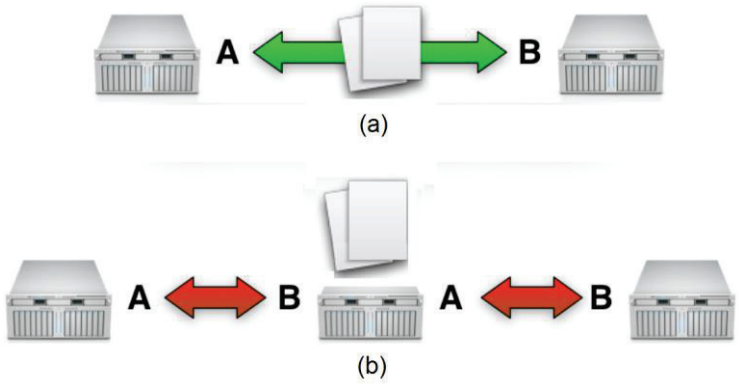

Figura 4. (a) Como os dispositivos enxergam a comunicação. (b) Como a comunicação ocorre de fato

Com relação à estrutura do injetor TANATTOS, este é dividido em três componentes principais: uma biblioteca para captura de pacotes em alta velocidade, chamada PF_RING [“PF_RING” 2011], uma aplicação que implementa as funções de injeção de falhas e gera um arquivo de *log* com dados referentes ao teste aplicado e uma interface gráfica que envia parâmetros para esta aplicação, com base no tipo de falha a ser injetada, como ilustrado pela Figura 5.

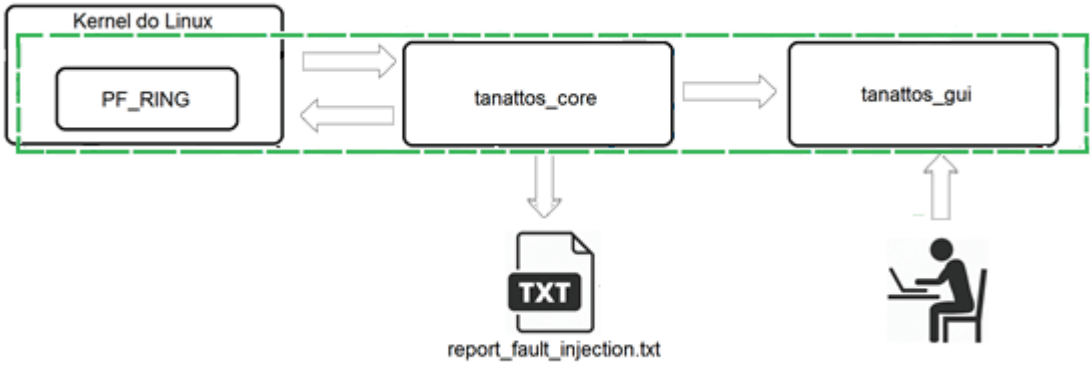

Figura 5. Organização interna do injetor TANATTOS

PF_RING é uma biblioteca para captura e envio de pacotes em alta velocidade, a qual torna possível a análise e manipulação de pacotes e tráfego ativo de rede. Essa biblioteca possui um módulo que deve ser carregado no *Kernel* do *Linux*, permitindo que os pacotes sejam trocados diretamente entre a interface de rede e aplicações do usuário sem utilizar mecanismos do *Kernel* do *Linux*. Neste injetor utiliza-se o modo DNA (*Direct NIC Access*) do PF_RING, que mapeia a memória e registradores das interfaces de rede para o espaço do usuário sem utilizar a *Linux Network API* (*Linux NAPI*), evitando perdas de performance significativas.

4.2. Funções de injeção de falhas

Com base no modelo de falhas levantado a partir da especificação do protocolo, foram criadas funções de injeção de falhas que exercitam os mecanismos de detecção e correção de falhas já descritos. Assim, o injetor TANATTOS conta com as funções de atraso de pacotes, atraso de pacotes intermitente, perda de pacotes, corrupção de pacotes, corrupção

de comandos de escrita e leitura, corrupção do *Circulating Bit*, corrupção do *Working Counter* e uma função de execução normal.

As funções de atraso de pacotes mantêm os pacotes recebidos por uma das interfaces de rede em um *buffer* (próprio do PF_RING) por um tempo especificado pelo usuário. A diferença entre as funções de atraso de pacotes e atraso de pacotes intermitente é que a função de atraso comum atrasa todo o fluxo de dados, enquanto a função de atraso intermitente simula uma situação onde a comunicação entre dois dispositivos alterna entre períodos de funcionamento normal e períodos onde o fluxo de pacotes inteiro é atrasado. Estes testes possuem dois objetivos principais: o primeiro é avaliar o correto funcionamento dos *watchdogs* do escravo ao reproduzir uma situação de "silêncio" na comunicação por um período de tempo maior que o contador interno dos *watchdogs*. O segundo objetivo é obter uma métrica da margem de operacionalidade do dispositivo mestre através da seguinte questão: "*se é sabido que um dispositivo escravo introduz um atraso de X milissegundos na comunicação do sistema como um todo, qual o maior atraso que um mestre pode suportar antes de entrar em colapso?* ".

Na função de perda de pacotes, o injetor não reenvia os pacotes que estiverem sendo recebidos em uma das interfaces de rede. Este cenário simula um barramento ou dispositivo com problemas de conexão, onde pacotes são perdidos durante a comunicação e tem como finalidade avaliar a robustez do dispositivo.

As funções de corrupção de pacotes e comandos de leitura / escrita permitem respectivamente alterar o conteúdo de um *byte* ou *bit* específico de um pacote ou os dados oriundos de uma leitura ou escrita. Estas têm por objetivo forçar erros de *checksum* e verificar o correto funcionamento das funções de cálculo de CRC da camada de Enlace do escravo e do contador interno responsável por armazenar o número de erros deste tipo.

Já a função de corrupção do *Working Counter* é a única que não tem como alvo testar a funcionalidade de um dispositivo escravo e sim observar como o mestre se comporta ao receber um valor inesperado de *Working Counter*.

A função de corrupção do *Circulating Bit* busca alterar o valor do *Circulating Bit* dos datagramas EtherCAT contidos dentro de cada pacote. Com isso, alterar seu valor de 0 (padrão) para 1 fará com que o pacote não seja processado e sim descartado ao chegar no dispositivo escravo.

Finalmente, a função de execução normal oferece a opção de uma comunicação sem falhas. Esta função tem o objetivo de verificar se o ambiente de testes montado está configurado de maneira correta, ou seja, se a comunicação entre dois dispositivos está funcionando adequadamente antes que os testes de injeção de falhas possam ser iniciados.

4.3. Metodologia de injeção de falhas

O método com que as falhas são injetadas busca chegar o mais próximo de um cenário real, onde falhas podem acontecer a qualquer momento. Deste modo, dificilmente um teste retornará os mesmos resultados caso seja executado várias vezes com os mesmos parâmetros de entrada. Para isto, foi adotado um método onde os pacotes que irão sofrer a injeção de falhas são escolhidos aleatoriamente seguindo uma distribuição uniforme.

Para se injetar uma falha, primeiro seleciona-se o tipo de falha e em seguida devem ser escolhidos alguns parâmetros. Dentre estes parâmetros três são comuns a todas as funções: as duas interfaces de rede correspondentes aos dispositivos mestre e escravo

(interfaces DNA) e uma direção para injetar a falha, seja no sentido mestre para escravo ou escravo para mestre. Dependendo do tipo de falha a ser injetada, outros parâmetros específicos devem ser informados, como chance de cada pacote sofrer a falha, um *bit* ou *byte* a ser corrompido ou tempo de atraso.

Após a definição da função de falhas escolhida e do preenchimento dos seus respectivos parâmetros, o experimento está pronto para ser iniciado. Com exceção do teste de atraso de pacotes, o usuário informa um valor entre 0,0 e 1,0, que corresponde à chance percentual de cada pacote sofrer a falha (0,0 sendo equivalente a 0% e 1,0 a 100%). Assim que um pacote chega à interface definida, o injetor calcula um valor entre 0,0 e 1,0 de maneira aleatória, seguindo uma distribuição uniforme. Caso este valor seja menor que o valor informado pelo usuário, o pacote sofrerá a injeção de falhas, caso contrário, o pacote será encaminhado normalmente.

Uma característica existente no modo de operação de TANATTOS é que apenas um tipo de falha pode ser injetada por vez, não sendo permitido que diferentes testes sejam executados sequencialmente de forma automática. Para tal é necessário que após a execução de um teste outro seja iniciado de forma manual.

4.4. Geração de *log* de injeção de falhas

Após o encerramento da seção de injeção de falhas, informações pertinentes à função de injeção de falhas escolhida serão salvas em um arquivo de texto, podendo ser acessado pelo botão correspondente na janela principal do injetor, sendo exibido através do editor de texto padrão do sistema (*gedit* por exemplo).

Neste relatório são exibidos a data e hora de início do teste, a quantidade de pacotes recebidos e enviados em cada interface e direção bem como uma indicação de qual direção foi escolhida para a injeção de falhas e a quantidade de falhas que foram injetadas. Nesse caso, para os testes de atraso de pacotes é informado quantos pacotes foram atrasados e o tempo de atraso. Para o teste de corrupção de pacotes também é exibida a quantidade (absoluta e percentual) de pacotes que foram corrompidos. Para os testes de corrupção de dados de comandos, corrupção de *Circulating Bit* e corrupção de *Working Counter* é exibida a quantidade de datagramas EtherCAT afetados e a média de datagramas por pacote. Para o teste de execução normal é exibida apenas a quantidade de pacotes transmitidos entre as interfaces.

Nos testes que envolvem corrupção dos pacotes (corrupção, *Circulating Bit, Working Counter* e dados de comandos), quando um pacote é alterado pela respectiva função, é exibida em uma tela de terminal o conteúdo do pacote antes e depois da falha

5. Ambiente experimental e testes realizados

O ambiente experimental apresentado na seção 4.1, conta com dispositivos que executam o protocolo EtherCAT cedidos pela empresa parceira no projeto: um mestre EtherCAT que é monitorado em tempo real por um computador separado, um escravo EtherCAT que recebe comandos vindos do mestre e um segundo computador que executa o injetor, sendo colocado entre o mestre e o escravo, interceptando os pacotes trocados. O fato dos dispositivos cedidos já serem previamente certificados serviu como meio de validar o injetor, pois estes dispositivos já foram certificados.

O computador onde o injetor foi executado conta com processador *Intel Core* i7-3770 3.4 GHz com 16 GB de memória RAM, com sistema operacional *Ubuntu* 12.04

LTS 64 *bits* e duas placas de rede *Intel Gigabit* modelo e1000e 82574L que são responsáveis por interligar os dispositivos mestre e escravo ao injetor de falhas. Já o computador que monitorava o mestre EtherCAT conta com um processador *Intel Core* 2 *Duo* E4000 2.4 GHz, 4 GB de memória RAM e sistema operacional *Windows*® 7 32 bits.

Antes que o sistema fosse fisicamente montado, utilizou-se uma ferramenta de configuração do *hardware* a ser utilizado que define: uma aplicação que é a carga de trabalho e quais dispositivos seriam usados, como seriam conectados e qual seria sua disposição física nos barramentos, de modo que fosse possível o monitoramento via *software*. A Figura 6 exibe uma captura de tela que mostra a modelagem final do sistema, com os dois barramentos (barramento do mestre à esquerda e do escravo à direita) conectados e a disposição física dos dispositivos a serem usados.

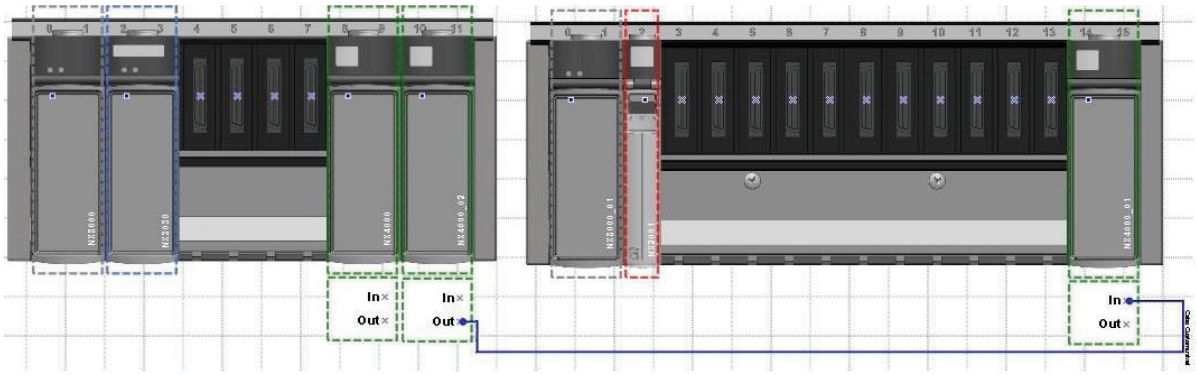

Figura 6. Arquitetura do sistema modelada via *software*

Em seguida, o ambiente de testes final foi montado seguindo o modelo desenvolvido em software. A Figura 7 mostra um esquemático do sistema, exemplificando as ligações entre todos os equipamentos. Tais equipamentos são: dispositivos “A” são fontes de alimentação 30 W e 24 Vdc, uma para cada barramento. O dispositivo “B” é o módulo escravo EtherCAT. Dispositivos “C” são responsáveis por transportar os pacotes de e para o mestre e o escravo, por isto devem ser dois. Estes dispositivos “C” serão conectados cada um a uma das placas de rede do computador executando o injetor de falhas. O dispositivo “E” é um controlador programável que age como mestre. Dispositivo “D” serve para monitorar o fluxo de pacotes que transita pelo mestre e está conectado ao computador monitor para fins de depuração, onde o *sniffer Wireshark* [“Wireshark · Go Deep.” 2017] é executado. Com o auxílio do *Wireshark* é possível verificar se os testes que envolvem corrupção de dados estão surtindo efeito. Isto se faz comparando o conteúdo do pacote que sai do mestre com o conteúdo do pacote corrompido que sai do injetor em direção ao escravo, por exemplo.

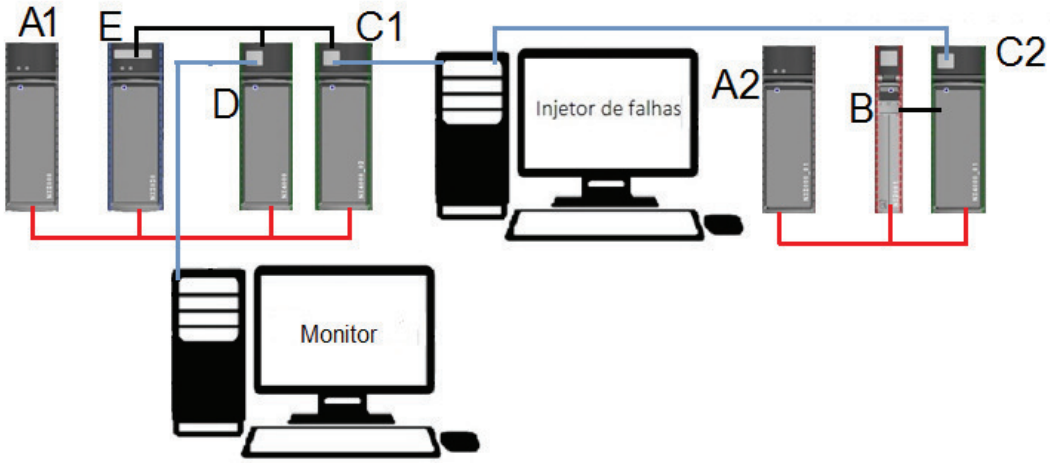

Figura 7. Arquitetura do ambiente utilizado durante os experimentos

O *software* de monitoramento conta com variáveis de diagnóstico referentes ao comportamento do mestre: *byWHSBBusErrors* que é um contador de falhas no barramento variando de 0 a 255; *adwModulePresenceStatus*, que é um *array* de DWORDS representando o *status* de presença dos dispositivos de entrada e saída declarados nos barramentos individualmente; *adwRackIOErrorStatus,* que representa erros nos dispositivos dos bastidores e *byHotSwapAndStartupStatus,* que informa através de códigos de erro, situações anormais no barramento responsáveis pela parada da aplicação.

Inicialmente foi aplicada a função de execução normal para saber o estado normal das variáveis apresentadas anteriormente, ou seja, qual valor é exibido por elas nas condições normais de funcionamento. Assim, são obtidos os valores correspondentes a uma execução sem falhas, que serão usados como referência para testes futuros. A Figura 8 mostra tais valores, em especial NORMAL_OPERATING_STATE para *byHotSwapAndStartupStatus*, que indica operação normal do sistema, os valores 1024 e 16388 para *adwModulePresenceStatus*, que indicam que os dispositivos funcionam corretamente e os valores 0 para *adwRackIOErrorStatus* indicando uma operação sem erros. Neste trabalho, por uma questão de espaço, serão apresentados e discutidos apenas os testes de perda e corrupção de pacotes.

WHSB	T_DIAG_WHSB	
byHotSwapAndStartupStatus	EN_HOT_SWAP	NORMAL_OPERATING_STATE
adwRackIOErrorStatus	ARRAY [0..31] OF DWORD	
adwModulePresenceStatus	ARRAY [0..31] OF DWORD	
byWHSBBusErrors	BYTE	0
adwModulePresenceStatus	ARRAY [0..31] OF DWORD	
adwModulePresenceStatus[0]	DWORD	1024
adwModulePresenceStatus[1]	DWORD	16388
adwRackIOErrorStatus	ARRAY [0..31] OF DWORD	
adwRackIOErrorStatus[0]	DWORD	0
adwRackIOErrorStatus[1]	DWORD	0

Figura 8. Variáveis utilizadas para monitoramemento e seus valores padrão

Para o teste de perda de pacotes, o injetor foi configurado a descartar 25% dos pacotes na direção do mestre para o escravo. Este valor foi escolhido porque testes preliminares mostraram que probabilidades menores do que 25% não foram capazes de gerar erros na operação normal dos dispositivos.

Após o descarte de pacotes ocorrido, a variável *byHotSwapAndStartupStatus* apresentou o *status* APPL_STOP_MODULES_NOT_READY, que, segundo o manual dos dispositivos, indica que a aplicação se encontra em modo *Stop* e todas as consistências foram realizadas com sucesso, mas os dispositivos não estão aptos para a partida do sistema. Isto é um indicativo que a perda de pacotes foi significativa a ponto de os dispositivos mestre e escravo não conseguirem executar os procedimentos necessários para o início do funcionamento. A variável *adwRackIOErrorStatus* por sua vez, apresentava os valores 1024 e 16388, diferentes dos valores 0 obtidos pela execução normal, indicando problemas devido à perda de pacotes. Os dados coletados podem ser vistos na Figura 9.

WHSB	T_DIAG_WHSB	
byHotSwapAndStartupStatus	EN_HOT_SWAP	APPL_STOP_MODULES_NOT_READY
adwRackIOErrorStatus	ARRAY [0..31] OF DWORD	
adwRackIOErrorStatus[0]	DWORD	1024
adwRackIOErrorStatus[1]	DWORD	16388

Figura 9. Variáveis *byHotSwapAndStartupStatus* e *adwRackIOErrorStatus* após a ocorrência de perda de pacotes

No teste de corrupção do *Working Counter*, cada datagrama EtherCAT tinha 40% de chance de sofrer uma alteração em seu respectivo *Working Counter* na direção mestre para escravo. A Figura 10 exibe a tela de configuração deste teste.

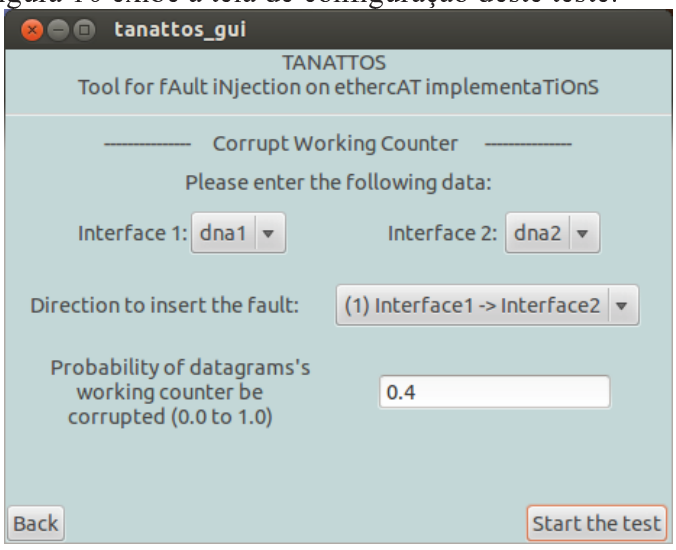

Figura 10. Configuração do teste de corrupção de pacotes

As variáveis *byHotSwapAndStartupStatus* e *byWHSBBusErrors* apresentam valores diferentes do encontrado pelo teste de execução normal, indicando que problemas devido à alteração de dados dos pacotes sem alterar o campo FCS foram detectados. Como mostrado pela Figura 11, *byHotSwapAndStartupStatus* apresenta o *status* APPL_STOP_MODULES_NOT_READY e *byWHSBBusErrors* agora contabiliza 215 erros.

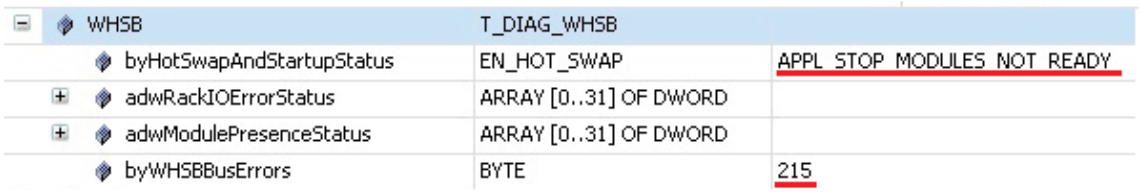

Figura 11. Variáveis *byHotSwapAndStartupStatus* e *adwRackIOErrorStatus* após corrupção de pacotes

O comportamento apresentado pelos dispositivos e refletido através das variáveis de diagnóstico, mostra que a comunicação entre eles foi de fato perturbada pelas falhas injetadas. Dessa maneira, os testes aplicados bem como os resultados obtidos serviram o propósito de validar a ferramenta desenvolvida.

6. Conclusão

Este trabalho descreveu o projeto e implementação do injetor de falhas TANATTOS, sendo próprio para o protocolo EtherCAT. Este injetor permite que implementações do protocolo sejam testadas mediante falhas durante seu processo de desenvolvimento, antes do processo de validação final realizado pelo órgão responsável. Para tanto, não é exigida nenhuma modificação no código-alvo nem uma carga de trabalho específica.

Por ser localizado fora dos dispositivos mestre e escravo, TANATTOS apresenta uma natureza minimamente intrusiva, agindo diretamente nos pacotes trocados entre estes. Além disto, a rápida operação do injetor sobre os pacotes é alcançada através do uso da biblioteca PF_RING em seu modo DNA.

TANATTOS pode ser utilizado para medir a cobertura de falhas de uma dada estratégia de tolerância a falhas implementada no sistema-alvo, porém seu objetivo principal é ajudar desenvolvedores a avaliar se as implementações dos métodos de tolerância a falhas do protocolo EtherCAT atendem aos requisitos impostos pela especificação.

Por ser uma ferramenta em *software*, TANATTOS é flexível o bastante para ser adaptado para protocolos diferentes de EtherCAT. De fato, as funções de atraso e perda de pacotes podem ser reaproveitadas. Outros tipos de falhas, porém, podem precisar de mudanças para atender às características específicas destes protocolos.

Em seu estado atual TANATTOS possui todas as funções de injeção de falhas anteriormente descritas já implementadas, porém a função de corrupção de pacotes não pode ser testada de forma adequada. É necessário que um erro forçado de CRC seja detectado e para isto é preciso que: (1) as interfaces de rede transmitam o campo de FCS dos pacotes para as camadas superiores e (2) que pacotes com erro de CRC não sejam descartados. Assim, não é desejável que tais pacotes sejam descartados pelas interfaces ou que o CRC seja recalculado, mascarando assim a falha injetada. Uma possível solução usando o utilitário *ethtool* presente na distribuição *Ubuntu* do *Linux* permitia que as condições (1) e (2) fossem atendidas, mas os comandos utilizados no *ethtool* para isto acabaram causando conflitos com o PF_RING.

Agradecimentos

Trabalho desenvolvido no Projeto SDCD - FAURGS/UFRGS/BNDES – (13.2.0646.1 - projeto 8108).

7. Referências

Arlat, J. et al. (1990). “Fault injection for dependability validation: A methodology and some applications”, *IEEE Transactions on Software Engineering*, v. 16, n. 2, p. 166–182.

beSTORM® Software Security Testing Tool (2016). Disponível em: <http://www.beyondsecurity.com/bestorm.html>, Acesso em 26 Out. 2016.

EC-STA [EtherCAT Slave Test Application] ([S.d.]). Disponível em: <http://www.acontis.com/eng/products/ethercat/ec-sta/index.php>, Acesso em 26 Out. 2016.

EtherCAT Technology Group (2014). “EtherCAT Slave Controller – Hardware Data Sheet Section 1”.

EtherCAT Technology Group (2015). “EtherCAT – The Ethernet Fieldbus”, . EtherCAT Technology Group. Disponível em: <https://www.ethercat.org/en/downloads.html>.

EtherCAT Technology Group | EtherCAT Conformance Test Tool (CTT) (2016). Disponível em: <https://www.ethercat.org/en/ctt.htm>, Acesso em 27 Out. 2016.

EtherCAT Technology Group | HOME (2016). Disponível em: <https://www.ethercat.org/default.htm>, Acesso em 30 Mai. 2016.

Gomes, L. G. et al. (2016). “Injeção de falhas de comunicação sobre implementações do protocolo EtherCAT”, *14ª Escola Regional de Redes de Computadores*, p. 53–60.

Neumann, P. (2007). “Communication in industrial automation—What is going on?”, *Control Engineering Practice*, v. 15, n. 11, p. 1332–1347.

PF_RING (4 ago 2011). Disponível em: <http://www.ntop.org/products/packet-capture/pf_ring/>, Acesso em 18 Out. 2016.

Wireshark · Go Deep. (2017). Disponível em: <https://www.wireshark.org/>, Acesso em 27 Mar. 2017.

Yu, Y. et al. (2005). “A state of research review on fault injection techniques and a case study”, Em *Annual Reliability and Maintainability Symposium, 2005. Proceedings.* IEEE.

Zhou, T. e Hu, J. (2011). “Design and realization of EtherCAT master”, Em *Electronic and Mechanical Engineering and Information Technology*. IEEE.