UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

RODRIGO JAUREGUY DOBLER

FITT: Fault Injection Test Tool
Uma Ferramenta de Injeção de Falhas para Validar Protocolos de Comunicação Seguros

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Álvaro Freitas Moreira

Porto Alegre, Junho de 2016

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Dobler, Rodrigo Jaureguy

FITT: Fault Injection Test Tool - Uma Ferramenta de Injeção de Falhas para Validar Protocolos de Comunicação Seguros / Rodrigo Jaureguy Dobler. – 2016.

15 f.:il.

Orientador: Álvaro Freitas Moreira.

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2016.

1. Injeção de falhas. 2. Protocolos de comunicação seguros. 3. Validação de mecanismos de tolerância a falhas. 4. IEC61508. 5. Testes. I. Moreira, Álvaro Freitas. II. Título.

AGRADECIMENTOS

Primeiramente eu quero agradecer ao professor orientador Álvaro Freitas Moreira pela oportunidade de poder cursar o mestrado no programa de Pós Graduação da UFRGS.

Eu quero agradecer especialmente ao professor Sérgio Luis Cechin por toda a ajuda e incentivo no desenvolvimento deste trabalho do mestrado. Eu também quero agradecer aos professores Taisy Weber e João Netto, juntamente com o professor Sérgio Luis Cechin, pela convivência nos últimos anos, durante participação do projeto RIO-SIL. Foi a partir desse projeto que surgiu a ideia de desenvolver um injetor de falhas que permitisse validar os protocolos de comunicação seguros, em especial o *PROFIsafe*, utilizado neste projeto. Agradeço também pelas correções e argumentações durante a escrita de artigos, o que me ajudou a ter uma visão mais crítica e objetiva no momento de redigir estes artigos.

A seguir quero expressar minha gratidão para os meus amigos e ex-colegas de laboratório William Vidal, Leandro Silva e Lucas Neubert, pela convivência durante estes últimos anos. Certamente as nossas conversas, relatos de situações engraçadas vivenciadas, contos de piadas e muitas risadas, tornaram o nosso tempo no laboratório muito mais agradável durante os intervalos de nossas tarefas.

Eu também quero agradecer aos meus grandes amigos Fernando Macedo, Giovano Camaratta, Maurício Condessa e Jorel Settin. A convivência com vocês durante a graduação do curso de Engenharia da Computação foi muito boa. Hoje percebo que manter o contato com vocês é essencial para toda a vida.

Por fim eu quero agradecer de forma especial aos meus pais, Oscar e Helena, por terem me incentivado a ir até final do trabalho do mestrado, mesmo quando eu pensei em desistir várias vezes pelos vários problemas encontrados para corrigir a implementação do protocolo *PROFIsafe* utilizado nos testes deste trabalho. Mas antes disso, sou grato pelo caráter que eu tenho, moldado através da educação e valores deles recebidos.

RESUMO

Protocolos de comunicação seguros são essenciais em ambientes de automação industrial, onde falhas não detectadas na comunicação de dispositivos podem provocar danos irreparáveis à vida ou ao meio-ambiente. Esses protocolos seguros devem ser desenvolvidos de acordo com alguma norma de segurança, como a *IEC 61508*. Segundo ela, faz parte do processo de implementação destes protocolos, a escolha de técnicas adequadas de validação, entre elas a injeção de falhas, a qual deve considerar um modelo de falhas apropriado ao ambiente de operação do protocolo.

Geralmente, esses ambientes são caracterizados pela existência de diversas formas de interferência elétrica e eletromagnética, as quais podem causar falhas nos sistemas eletrônicos existentes. Nos sistemas de comunicação de dados, isto pode levar a destruição do sinal de dados e causar estados de operação equivocados nos dispositivos. Assim, é preciso utilizar uma técnica de injeção de falhas que permita simular os tipos de erros de comunicação que podem ocorrer nos ambientes industriais. Dessa forma, será possível verificar o comportamento dos mecanismos de tolerância falhas na presença de falhas e assegurar o seu correto funcionamento.

Para esta finalidade, este trabalho apresenta o desenvolvimento do injetor de falhas *FITT* para validação de protocolos de comunicação seguros. Esta ferramenta foi desenvolvida para ser utilizada com o sistema operacional *Linux*. O injetor faz uso do *PF_RING*, um módulo para o *Kernel* do *Linux*, que é responsável por realizar a comunicação direta entre as interfaces de rede e o injetor de falhas. Assim os pacotes não precisam passar pelas estruturas do *Kernel* do *Linux*, evitando que atrasos adicionais sejam inseridos no processo de recebimento e envio de mensagens.

As funções de falhas desenvolvidas seguem o modelo de falhas de comunicação descrito na norma *IEC 61508*. Esse modelo é composto pelos erros de repetição, perda, inserção, sequência incorreta, endereçamento, corrupção de dados, atraso, mascaramento e falhas de memória em switches.

Palavras-chave: Injeção de Falhas, Protocolos de Comunicação Seguros, Validação, Detecção de Erros, Mecanismos de Tolerância a Falhas, IEC61508, Testes.

FITT: A Fault Injection Tool to Validate Safety Communication Protocols

ABSTRACT

Safe communication protocols are essential in industrial automation environments, where undetected failures in the communication of devices can cause irreparable damage to life or to the environment. These safe protocols must be developed according to some safety standard, like *IEC 61508*. According to it, part of the process of implementing these protocols is to select appropriate techniques for validation, including the fault injection, which should consider an appropriate fault model for the operating environment of the protocol.

Generally, these environments are characterized by the existence of various forms of electric and electromagnetic interference, which can cause failures in existing electronic systems. In data communication systems, this can lead to the destruction of the data signal and cause erroneous operation states in the devices. Thus, it is necessary to use a fault injection technique that allows simulating the types of communication errors that may occur in industrial environments. So, it will be possible to verify the behavior of the fault tolerance mechanisms in the presence of failures and ensure its correct functioning.

For this purpose, this work presents the development of FITT fault injector for validation of safety communication protocols. This tool was developed to be used with Linux operating system. The fault injector makes use of PF_RING, a module for the Linux Kernel and that is responsible to perform the direct communication between the network interfaces and the fault injector. Thus the packages do not need to go through the Linux Kernel structures, avoiding additional delays to be inserted into the process of receiving and sending messages.

The developed fault injection functions follow the communication fault model described in the IEC61508 standard, composed by the errors of repetition, loss, insertion, incorrect sequence, addressing, data corruption, delay, masking and memory failures within switches. The fault injection tests applied with this model allow to properly validate the fault tolerance mechanisms of safety protocols.

Keywords: ABNT. Fault Injection, Safety Communication Protocols, Validation, Errors Detection, Fault Tolerance Mechanisms, IEC61508, Tests.

LISTA DE FIGURAS

Figura 3.1 – Exemplo de uma Aplicação Prática do PROFIsafe....33
Figura 3.2 – Interface entre a Aplicação Segura e o F-Host Driver....34
Figura 3.3 – Interface entre a Aplicação do F-Device e o F-Device Driver....36
Figura 3.4 – Estrutura do Safety PDU com 12 bytes de dados....37
Figura 3.5 – Estrutura do Buffer_CRC para o cálculo do CRC2....38
Figura 3.6 – Estrutura do Control Byte....38
Figura 3.7 – Estrutura do Status Byte....39
Figura 3.8 – Diagrama de Estados do F-Host....40
Figura 3.9 – Diagrama de Estados do F-Device....42
Figura 5.1 – Abordagem de Injeção de Falhas de acordo com a Intrusividade....51
Figura 5.2 – Abordagens de Injeção de Falhas dos Injetores Pesquisados....52
Figura 5.3 – Componentes do Injetor de Falhas....53
Figura 5.4 – Interface da janela principal do Injetor de Falhas FITT....55
Figura 5.5 – Ligação entre o F-Host, Injetor de Falhas e o F-Device....57
Figura 5.6 – Interface da Função de Erro de Endereçamento....58
Figura 5.7 – Interface da Função de Corrupção de Dados....59
Figura 5.8 – Interface da Função de Atraso de Pacotes....61
Figura 5.9 – Interface da Função de Sequência Incorreta....62
Figura 5.10 – Interface da Função de Inserção de Mensagens....63
Figura 5.11 – Interface da Função de Descarte de Mensagens....65
Figura 5.12 – Interface da Função de Repetição de Mensagens....66
Figura 5.13 – Interface da Função de Execução Normal....67
Figura 6.1 – Inicialização do F-Device....70
Figura 6.2 – Inicialização do F-Host....70
Figura 6.3 – Condições necessárias para o F-Device entrar no ciclo normal de operação....71
Figura 6.4 – Condição necessária para o F-Host entrar no ciclo normal de operação....72
Figura 6.5 – Detecção de erro de CRC pelo F-Device....73
Figura 6.6 – A variável FV_activated_S e o Control Bit activate_FV recebem o valor “1” após aviso de erro de CRC pelo F-Device....73
Figura 6.7 – Toggle Bits sincronizados entre os dois dispositivos PROFIsafe....74
Figura 6.8 – Após o recebimento de duas mensagens sem falhas do F-Host, o CE_CRC do F-Device não reporta mais falhas....74
Figura 6.9 – F-Device retorna ao estado normal de operação e depois desliga o Status Bit FV_activated que será enviado ao F-Host....75
Figura 6.10 – F-Host desliga os bits FV_activated_S e activated_FV e retorna ao seu estado normal de operação....75
Figura 6.11 – Detecção de erro de CRC pelo F-Device....76
Figura 6.12 – A variável FV_activated_S e o Control Bit activate_FV recebem o valor “1” após aviso de erro de CRC pelo F-Device....77
Figura 6.13 – Toggle Bits sincronizados entre os dois dispositivos PROFIsafe....77
Figura 6.14 – Após o recebimento de duas mensagens sem falhas do F-Host, o CE_CRC do F-Device não reporta mais falhas ao F-Host....78
Figura 6.15 – F-Device retorna ao estado normal de operação e depois desliga o Status Bit FV_activated que será enviado ao F-Host....78
Figura 6.16 – F-Host desliga os bits FV_activated_S e activated_FV e retorna ao seu estado normal de operação....79
Figura 6.17 – Detecção de erro de Timeout pelo F-Device....80

Figura 6.18 – O F-Host vai para o estado 8 para tratar a falha de erro de CRC detectada.....80
Figura 6.19 – F-Device não aceita mensagem do F-Host por falta de sincronia no Toggle_h 81
Figura 6.20 – A mensagem que foi atrasada não é aceita no F-Device porque a variável ok_nr_de_cycles<2 81
Figura 6.21 – Ack do F-Device descartado pelo F-Host 82
Figura 6.22 – Mensagem descartada do F-Host para o F-Device 82
Figura 6.23 – Ack do F-Device descartado pelo F-Host 82
Figura 6.24 – Toggle Bits sincronizados entre os dois dispositivos PROFIsafe.....83
Figura 6.25 – Após o recebimento de duas mensagens sem falhas do F-Host, o WD_timeout do F-Device não reporta mais falhas ao F-Host 83
Figura 6.26 – F-Device retorna ao estado normal de operação e depois desliga o Status Bit FV_activated que será enviado ao F-Host 84
Figura 6.27 – F-Host desliga os bits FV_activated_S e activated_FV e retorna ao seu estado normal de operação.....84
Figura 6.28 – Detecção de erro de CRC pelo F-Device.....85
Figura 6.29 – A variável FV_activated_S e o Control Bit activate_FV recebem o valor "1" após aviso de erro de CRC pelo F-Device.....86
Figura 6.30 – Mensagem descartada do F-Host para o F-Device 86
Figura 6.31 – Ack do F-Device descartado pelo F-Host 87
Figura 6.32 – Toggle Bits sincronizados entre os dois dispositivos PROFIsafe.....87
Figura 6.33 – Após o recebimento de duas mensagens sem falhas do F-Host, o CE_CRC do F-Device não reporta mais falhas ao F-Host.....88
Figura 6.34 – F-Device retorna ao estado normal de operação e depois desliga o Status Bit FV_activated que será enviado ao F-Host.....88
Figura 6.35 – F-Host desliga os bits FV_activated_S e activated_FV e retorna ao seu estado normal de operação.....89
Figura 6.36 – O F-Device detecta a mensagem inserida através de erro de CRC 90
Figura 6.37 – A variável FV_activated_S e o Control Bit activate_FV recebem o valor "1" após aviso de erro de CRC pelo F-Device.....90
Figura 6.38 – Toggle Bits sincronizados entre os dois dispositivos PROFIsafe.....91
Figura 6.39 – Após o recebimento de duas mensagens sem falhas do F-Host, o CE_CRC do F-Device não reporta mais falhas ao F-Host.....91
Figura 6.40 – F-Device retorna ao estado normal de operação e depois desliga o Status Bit FV_activated que será enviado ao F-Host.....92
Figura 6.41 – F-Host desliga os bits FV_activated_S e activated_FV e retorna ao seu estado normal de operação.....92
Figura 6.42 – Detecção de erro de Timeout pelo F-Device 93
Figura 6.43 – O F-Host vai para o estado 8 para tratar a falha de erro de CRC detectada 94
Figura 6.44 – F-Device não aceita mensagem do F-Host por falta de sincronia no Toggle_h 94
Figura 6.45 – F-Host não aceita mensagem do F-Device por que os sinais R_cons_nr e cons_nr_R são diferentes.....95
Figura 6.46 – Toggle Bits sincronizados entre os dois dispositivos PROFIsafe 95
Figura 6.47 – Após o recebimento de duas mensagens sem falhas do F-Host, o WD_timeout do F-Device não reporta mais falhas ao F-Host 96
Figura 6.48 – F-Device retorna ao estado normal de operação e depois desliga o Status Bit FV_activated que será enviado ao F-Host 96
Figura 6.49 – F-Host desliga os bits FV_activated_S e activated_FV e retorna ao seu estado normal de operação.....97
Figura 6.50 – O F-Device detecta a mensagem duplicada através de erro de CRC 98

Figura 6.51 – A variável FV_activated_S e o Control Bit activate_FV recebem o valor “1” após aviso de erro de CRC pelo F-Device 98
Figura 6.52 – Toggle Bits sincronizados entre os dois dispositivos PROFIsafe 99
Figura 6.53 – Após o recebimento de duas mensagens sem falhas do F-Host, o CE_CRC do F-Device não reporta mais falhas ao F-Host 99
Figura 6.54 – F-Device retorna ao estado normal de operação e depois desliga o Status Bit FV_activated que será enviado ao F-Host 100
Figura 6.55 – F-Host desliga os bits FV_activated_S e activated_FV e retorna ao seu estado normal de operação 101
Figura 7.1 – Entrada de dados inválida do usuário 103
Figura 7.2 – Limitação relacionada ao buffer de uma posição e tempo 1 < tempo 2 105

LISTA DE TABELAS

Tabela 2.1 - Principais métodos de aplicação de falhas para as técnicas de injeção de falhas por hardware e por software 26
Tabela 2.2 – Características dos métodos de injeção de falhas para as técnicas de injeção de falhas por hardware e por software 27
Tabela 3.1 – Medidas de segurança adotadas pelo PROFIsafe para detectar os vários tipos de erros de transmissão 32
Tabela 6.1 – Comparação dos tempos de intrusividade causados pelo injetor em duas configurações diferentes de computadores 102

LISTA DE ABREVIATURAS E SIGLAS

AMD	Advanced Micro Devices
CAN	Controller Area Network
CPU	Central Unit Processing
CRC	Cyclic Redundancy Check
DNA	Direct NIC Access
FPGA	Field-Programmable Gate Array
FSCP	Functional Safety Communication Profiles
GB	GigaByte
GSD	General Station Description
IEC	International Electrotechnical Commission
IP	Internet Protocol
ISO	International Organization for Standardization
LTS	Long Term Support
MTBF	Mean Time Between Failures
MTTF	Mean Time To Failure
NAPI	New API
PFD	Probability of Dangerous Failure on Demand
PFH	Average Frequency of Dangerous Failure
RAM	Random Access Memory
SIL	Safety Integrity Level
SWIFI	Software-implemented fault injection
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UML	Unified Modeling Language
VHDL	Very High-speed integrated circuit Hardware Description

Sumário

1 INTRODUÇÃO 15
1.1 Motivação 15
1.2 Objetivos 16
1.3 Organização do Texto 17
2 INJEÇÃO DE FALHAS 19
2.1 Conceito 19
2.2 Objetivos da Injeção de Falhas 20
2.2.1 Validação 20
2.2.2 Auxílio em Projeto 21
2.3 A Questão da Intrusividade 21
2.4 Técnicas de Injeção de Falhas 21
2.4.1 Injeção de Falhas por Simulação 22
2.4.2 Injeção de Falhas por Hardware 22
2.4.3 Injeção de Falhas por Software 23
2.4.3.1 Problemas Encontrados em Ferramentas SWIFI 24
2.4.4 Injeção de Falhas Mista 25
2.5 Comparativo das Técnicas de Injeção de Falhas por Hardware e Software 25
3 PROTOCOLO DE COMUNICAÇÃO SEGURO PROFISAFE 28
3.1 Norma de Segurança Funcional IEC61508 28
3.2 Conceito Geral e Principais Características do PROFIsafe 29
3.3 Modelo de Falhas 30
3.4 Mecanismos de Tolerância a Falhas 31
3.5 Exemplo de Utilização do Protocolo PROFIsafe 32
3.6 Bits de Interação com a Aplicação 34
3.7 F-Parameters 36
3.8 Mensagem PROFIsafe 37
3.9 Funcionamento das Máquinas de Estados do F-Host e do F-Device 39
3.9.1 Máquina de Estados do F-Host 40
3.9.2 Máquina de Estados do F-Device 41
4 TRABALHOS RELACIONADOS 43
4.1 sfiCAN 43
4.2 Fault Injection Framework for PROFIBUS 44
4.3 FIRMAMENT 45
4.4 F-Host Test Tool 46
4.5 Simmcast-FT 46
4.6 Xception 47
4.7 Análise dos Injetores de Falhas Pesquisados 49
5 INJETOR DE FALHAS 51
5.1 Definição da Abordagem de Injeção de Falhas 51
5.2 Arquitetura do Injetor de Falhas 53
5.2.1 PF_RING 53
5.2.2 Interface Gráfica 54
5.2.3 Injetor de Falhas 55
5.3 Funcionamento do Injetor de Falhas 56
5.4 Funções de Injeção de Falhas 57
5.4.1 Função de Erro de Endereçamento 57
5.4.2 Função de Corrupção de Dados 59

5.4.3 Função de Atraso de Pacotes 60
5.4.4 Função de Sequência Incorreta 61
5.4.5 Função de Inserção de Mensagens 63
5.4.6 Função de Descarte de Pacotes 64
5.4.7 Função de Repetição de Mensagens 65
5.4.8 Função de Execução Normal 67
6 TESTES REALIZADOS COM O PROTOCOLO PROFISAFE 68
6.1 Ambiente de Testes 68
6.2 Comentários Sobre os Testes 68
6.3 Testes Realizados 69
6.3.1 Teste de Execução Normal 69
6.3.2 Teste de Erro de Endereçamento 72
6.3.3 Teste de Corrupção de Dados 76
6.3.4 Teste de Atraso de Pacotes 79
6.3.5 Teste de Sequência Incorreta 85
6.3.6 Teste de Inserção de Mensagens 89
6.3.7 Teste de Descarte de Pacotes 93
6.3.8 Teste de Repetição de Mensagens 97
6.4 Análise da Intrusividade Causada pelo Injetor 101
7 LIMITAÇÕES DO INJETOR DE FALHAS 103
7.1 Validação dos Dados de Entrada do Usuário 103
7.2 Limitação do Tamanho do Buffer e da escolha dos Tempos de Injeção de Falhas 104
7.3 Teste de Transições Específicas na Máquina de Estados do F-Device 105
7.3.1 Transição T31 106
7.3.2 Transição T32 106
8 CONSIDERAÇÕES FINAIS 107
8.1 Conclusões 107
8.2 Trabalhos Futuros 108
REFERÊNCIAS 110

1 INTRODUÇÃO

Este capítulo apresenta as motivações para o desenvolvimento do injetor de falhas apresentado neste trabalho. Também são mostrados os objetivos pretendidos para o injetor de falhas, de forma que ele seja uma ferramenta de fácil utilização para o usuário, que permita validar adequadamente os mecanismos de tolerância a falhas do protocolo *PROFIsafe* e que no futuro possa ser atualizado para testar outros protocolos de comunicação seguros. No final é apresentada a forma de organização do texto.

1.1 Motivação

A automação industrial e de processos baseia-se na utilização de barramentos de comunicação, os quais permitem a interligação de funções e sistemas altamente descentralizados nas plantas industriais [Thomesse 2005]. Em processos que envolvem níveis maiores de criticidade, é preciso que essa interligação seja feita de maneira confiável e segura [Thomesse 2005].

Para que a comunicação entre esses sistemas seja feita de forma segura, foram criados os protocolos de comunicação seguros [Neumann 2007], como por exemplo, o *Safety over EtherCAT* [EtherCAT FSoE 2015], o *openSafety* [openSafety 2015] e o *PROFIsafe* [PROFIsafe 2010]. Esses protocolos são especificados e desenvolvidos para atender aos requisitos de segurança da norma de Segurança Funcional *IEC 61508* [IEC 61508]. A norma estabelece critérios e requisitos para se desenvolver e determinar o nível de integridade de uma função de segurança. Estas funções são usadas para atingir ou manter um estado seguro quando um evento perigoso acontece, como por exemplo, vazamento de óleo e gás e/ou princípio de incêndio.

Funções de segurança para aplicações críticas de automação na indústria de energia, química e de óleo e gás, devem utilizar um protocolo de comunicação seguro. Nessas indústrias existe uma alta variedade de equipamentos, os quais geram diversas formas de interferência elétrica e eletromagnética. Em sistemas de comunicação, isto pode levar a destruição do sinal de dados e gerar estados errôneos em transceptores e outros circuitos eletrônicos [Kim 2000]. Assim, o uso de um protocolo seguro torna-se essencial para adicionar segurança na comunicação desses dispositivos.

Como forma de avaliar os mecanismos de gerenciamento de falhas do sistema alvo, pode-se utilizar a técnica de injeção de falhas [Carreira 1998]. Esta técnica é utilizada para

validar a dependabilidade de um sistema, sendo baseada na realização de experimentos controlados, onde a observação do comportamento do sistema na presença de falhas é explicitamente induzida pela introdução deliberada de falhas dentro desse sistema [YU 2005]. A avaliação da dependabilidade é necessária para verificar a conformidade do comportamento do sistema de acordo com a sua especificação [YU 2005].

Nas pesquisas realizadas foram encontrados muitos injetores de falhas com diferentes propósitos e metodologias de testes empregadas. No entanto, não encontramos nenhuma ferramenta que fosse capaz de injetar falhas de forma a simular os tipos de falhas descritos na norma *IEC 61508* e, assim, avaliar toda a implementação dos mecanismos de tolerância a falhas dos protocolos de comunicação seguros.

Por essa razão, a motivação para este trabalho foi desenvolver o injetor de falhas de comunicação *FITT* (*Fault Injection Test Tool*) para permitir a validação dos mecanismos de tolerância a falhas dos protocolos de comunicação seguros. O injetor de falhas será desenvolvido para testar o protocolo *PROFIsafe* de acordo com os requisitos do projeto *RIO-SIL*, o qual é financiado pela *FINEP* dentro da Rede Temática de Eletrônica Embarcada para Equipamentos (Rede E3). O protocolo foi escolhido por interesse da empresa parceira no projeto, para ser utilizado no desenvolvimento e produção de módulos de entradas e saídas digitais para sistemas instrumentados de segurança.

O uso do *FITT* é de grande importância para o projeto, pois o código do *PROFIsafe* pode ser obtido através da implementação conforme a sua especificação ou através da aquisição da sua pilha de comunicação. Em ambos os casos é necessário integrar o código do *PROFIsafe* ao código dos protocolos de transporte *PROFIBUS* [PROFIBUS 2015] ou *PROFINET* [PROFINET 2015] e também adaptá-lo ao *hardware* utilizado. No entanto, o processo de desenvolvimento, integração ou adaptação pode não ser feito corretamente, levando ao surgimento de erros durante a operação do protocolo. Isso pode gerar um mau funcionamento nos dispositivos de segurança e até causar graves acidentes. Dessa forma é necessário que o código do protocolo *PROFIsafe* seja verificado com o injetor desenvolvido para garantir o correto funcionamento do sistema.

1.2 Objetivos

O injetor de falhas *FITT* foi idealizado para validar a implementação de protocolos de comunicação seguros, utilizados na comunicação de equipamentos de segurança de

automação industrial. Para atender esse requisito, o injetor precisa ser capaz de simular os mesmos tipos de erros de comunicação que podem ocorrer nesses ambientes. Esses tipos de falhas estão descritos na norma de segurança funcional *IEC 61508*, utilizada no desenvolvimento desses tipos de protocolos.

Além disso, o injetor precisa apresentar uma baixa intrusividade, para causar a menor interferência possível no sistema sob teste. Dessa forma as atividades de injeção de falhas não irão interferir com os requisitos de tempo do protocolo e nem do sistema alvo, não afetando os resultados dos testes.

O injetor também tem por objetivo oferecer uma interface gráfica amigável para o usuário, facilitando a configuração do cenário de injeção de falhas. A visualização do relatório dos testes de injeção de falhas a partir da interface também é uma possibilidade interessante, pois agiliza a análise dos resultados.

Outra característica importante a ser oferecida pela ferramenta é a possibilidade de permitir a especificação de novas funções de injeção de falhas. Assim, o injetor poderá evoluir ao longo do tempo, com a adição de funções para testar outros protocolos de comunicação seguros.

Também se pretende desenvolver o injetor de falhas de forma que o custo final não seja muito elevado, para tornar viável a sua utilização no projeto *RIO-SIL* apresentado anteriormente. Nesse sentido, para facilitar o desenvolvimento, pretende-se utilizar a biblioteca *PF_RING*, a qual quando carregada no *Kernel* do *Linux*, realiza a troca de pacotes diretamente entre a aplicação e as interfaces de rede.

Essa biblioteca, além de ser encarregada de controlar a comunicação, oferece a vantagem de não inserir atrasos adicionais no envio e recebimento dos pacotes, pois eles não precisam passar pelas estruturas do *Kernel* para chegar até a aplicação.

A ferramenta vai poder ser utilizada em qualquer computador com o sistema operacional *Linux* a partir da versão 2.6.32, requisito devido ao *PF_RING*. Ela vai utilizar duas interfaces de rede para realizar a comunicação nos dois sentidos entre os dispositivos mestre e escravo a serem testados.

1.3 Organização do Texto

Neste trabalho é apresentado o desenvolvimento do injetor de falhas *FITT*, no qual é feito um estudo de conceitos, técnicas e arquiteturas necessários para permitir o entendimento

das escolhas e decisões tomadas no projeto.

No capítulo 2 é apresentado o conceito de injeção de falhas e das suas principais técnicas, mostrando qual delas é mais adequada para simular qual tipo de falhas. Já o capítulo 3 descreve as principais características do funcionamento do protocolo de comunicação seguro *PROFIsafe*, escolhido para ser utilizado nos testes com o injetor de falhas desenvolvido. Nessa seção também serão apresentadas as principais características da norma de Segurança Funcional *IEC 61508*, utilizada como uma das normas base para o desenvolvimento de protocolos de comunicação seguros.

Ainda nesse mesmo capítulo, são mostradas as máquinas de estados completas dos dispositivos *F-Host* e *F-Device*, permitindo analisar o funcionamento de cada uma delas através das transições entre os estados. Essas transições são realizadas de acordo com condições relativas às trocas de mensagens e recebimento das mesmas com ou sem erros de comunicação. Esse conhecimento é necessário para entender os testes realizados.

Por sua vez, o capítulo 4 trata sobre os trabalhos relacionados, onde foram pesquisadas diversas ferramentas com diferentes propósitos e técnicas de testes utilizadas. Elas foram escolhidas porque possuem uma ou mais características consideradas úteis para o desenvolvimento do *FITT*.

O capítulo 5 trata do desenvolvimento do injetor de falhas *FITT*, abordando os principais aspectos e questões utilizadas no seu desenvolvimento além de suas funcionalidades.

Já o capítulo 6 apresenta os testes realizados com o *FITT* e os resultados obtidos, mostrando que o injetor serve ao propósito de validar a implementação dos mecanismos de tolerância a falhas do protocolo de comunicação seguro *PROFIsafe.*

No capítulo 7 são discutidas as limitações da ferramenta injetora de falhas desenvolvida. No capítulo final são apresentadas as conclusões com os resultados obtidos neste trabalho, os possíveis trabalhos futuros para melhorar o *FITT* e propostas de testes a serem realizados. Além disso, também são propostas modificações a serem feitas nos dispositivos mestre e escravo *PROFIsafe* para a realização de testes mais complexos.

2 INJEÇÃO DE FALHAS

Este capítulo apresenta o conceito da técnica de injeção de falhas juntamente com os seus principais objetivos. Também é apresentada a questão da intrusividade, muito importante quando se realiza testes com sistemas de comunicação de tempo real. Em seguida, são apresentadas as principais técnicas de injeção de falhas utilizadas atualmente. Por fim, é feito um comparativo das duas principais técnicas de injeção de falhas, com a intenção de verificar qual delas melhor se adapta ao desenvolvimento do injetor de falhas deste trabalho.

2.1 Conceito

A injeção de falhas é a técnica utilizada para validar a dependabilidade de um sistema, sendo baseada na realização de experimentos controlados, onde a observação do comportamento do sistema na presença de falhas é explicitamente induzida pela introdução deliberada de falhas dentro desse sistema [YU et al 2005]. A avaliação da dependabilidade é um processo necessário para verificar a conformidade do comportamento do sistema de acordo com a sua especificação [LAPRIE et al 1995].

Esta técnica tem sido tradicionalmente utilizada para estimar os parâmetros de cobertura e latência para modelos analíticos de dependabilidade. Contudo, ela pode também avaliar outras medidas como confiabilidade, disponibilidade, *MTTF* (*Mean Time To Failure*) e *MTBF* (*Mean Time Between Failures*) [CLARCK et al 1995].

Vários experimentos de classificação de falhas analisaram como as falhas injetadas afetam o desempenho de um serviço de um sistema de computador [CLARCK et al 1995, YU et al 2005]. Outros estudos de injeção de falhas investigaram a propagação de erros do local de uma falha para outros componentes do sistema. Finalmente, pesquisadores observaram uma correlação entre a dependabilidade do sistema e a sua carga computacional ou características do código da aplicação. Tais relações de carga de trabalho são frequentemente exploradas utilizando-se injeção de falhas [CLARCK et al 1995].

A injeção de falhas foi aplicada inicialmente aos sistemas centralizados, especialmente as arquiteturas de computadores tolerantes a falhas do início dos anos 1970 [YU et al 2005]. Depois, ela foi utilizada quase que exclusivamente pela indústria, para medir os parâmetros de cobertura e de latência de sistemas altamente confiáveis. A academia começou a utilizar esta técnica depois da metade dos anos 1980 para conduzir pesquisas de testes de sistemas. Os primeiros trabalhos foram concentrados em entender o processo de propagação de erros e

analisar a eficiência de novos mecanismos de injeção de falhas. A seguir, a injeção de falhas foi aplicada em sistemas distribuídos e mais recentemente na *Internet*, podendo também atuar sobre as várias camadas de um sistema de computador, variando desde o *hardware* até o *software*. [BENSO et al 2003].

2.2 Objetivos da Injeção de Falhas

A técnica de injeção de falhas faz parte do processo de validação global de um projeto, onde ela tenta determinar se a resposta de um sistema na presença de um espaço definido de falhas coincide com a sua especificação. Geralmente, as falhas são inseridas em pontos específicos e estados do sistema determinados por uma análise inicial. Os desenvolvedores de um injetor de falhas conhecem muito bem o projeto, projetando os casos de teste, incluindo os tipos de falhas, pontos de teste, tempo de injeção e estado, com base em um critério estrutural e geralmente de uma maneira determinística [YU et al 2005]. Nesse contexto, os dois principais objetivos desta técnica são a validação e o auxílio em projeto [ARLAT et al 1989].

2.2.1 Validação

O papel da injeção de falhas está relacionado ao conceito de cobertura, isto é, a validação da validação, ou seja, como atingir confiança em métodos e mecanismos usados para construir a confiança no sistema [ARLAT et al 1989]. Nesse contexto, existem duas questões principais. A primeira diz respeito à validação dos procedimentos de verificação (conjunto de testes), os quais são utilizados para revelar falhas durante todas as fases do processo de desenvolvimento. Já a segunda refere-se à validação dos mecanismos de tolerância a falhas, objetivando atingir a dependabilidade do sistema na fase operacional.

A injeção de falhas participa de dois aspectos no processo de validação [ARLAT et al 1989]. O primeiro é a remoção de falhas através de uma análise qualitativa com o objetivo de verificar a adequação dos procedimentos de verificação e dos mecanismos de tolerância a falhas com as suposições de falhas consideradas. Já o segundo trata da previsão de falhas, onde são avaliadas a eficiência dos procedimentos de teste (cobertura de testes) e os parâmetros de tempo dos mecanismos de tolerância a falhas (fator de cobertura [BOURICIUS et al 1969], dormência de falhas, latência de erros [AVIZIENIS et al 1986], etc).

2.2.2 Auxilio em Projeto

A Injeção de falhas pode ser aplicada nas várias etapas do processo de desenvolvimento e, em particular, os resultados negativos das fases de remoção de falhas podem ser utilizados para iniciar ciclos de *feedback* para melhorar procedimentos de testes e os mecanismos de tolerância a falhas [ARLAT et al 1989, YU et al 2005].

Por esse motivo, a injeção de falhas pode ser considerada como um auxílio ao projeto, dada a sua aplicabilidade nas fases iniciais. Outro objetivo importante desta técnica como auxiliar de projeto refere-se à elaboração de dicionários de falhas, os quais podem ser utilizados para desenvolver procedimentos de manutenção [ARLAT et al 1989].

2.3 A Questão da Intrusividade

Intrusividade é a interferência causada pelo injetor ao realizar a injeção de falhas sobre o sistema alvo [DREBES 2006]. Essa interferência não pode ser evitada, porque de alguma forma o sistema precisa ser modificado para a realização de testes e, assim, pode-se apenas tentar minimizá-la. Ela é classificada em espacial e temporal.

A intrusividade espacial ocorre pela modificação do código da aplicação para a inserção de código de mecanismos de injeção de falhas, o que pode inserir *bugs* através do código e também pode alterar o fluxo de execução da aplicação.

Já a intrusividade temporal corresponde ao aumento do tempo de execução da aplicação pelo acréscimo das atividades do injetor, as quais são executadas junto com a aplicação alvo. Isto é muito importante em sistemas de tempo real, onde a previsibilidade do tempo pode ser alterada pela sobrecarga de um mecanismo de injeção de falhas [DREBES 2006]. Também é necessário garantir que erros não intencionais e erros de temporização, não sejam introduzidos durante o experimento.

2.4 Técnicas de Injeção de Falhas

A injeção de falhas acelera a ocorrência de falhas e erros, o que a torna um método atrativo para testar os mecanismos de tolerância a falhas em relação a um conjunto específico de falhas com as quais esses mecanismos são projetados para lidar [ARLAT et al 2003].

Atualmente existem quatro tipos de técnicas de injeção de falhas, os quais são apresentados a seguir, juntamente com as suas principais características.

2.4.1 Injeção de Falhas Por Simulação

A injeção de falhas por simulação é geralmente utilizada nas fases iniciais do projeto de um sistema, onde falhas são aplicadas em um modelo de simulação de um sistema alvo como forma de avaliar a sua dependabilidade e a efetividade do projeto dos seus mecanismos de tolerância a falhas [CARREIRA et al 1998, HSUEH et al 1997, ARLAT et al 1999]. Modelos de simulação são geralmente desenvolvidos utilizando-se uma linguagem de descrição de *hardware*, como *VHDL* ou *Verilog* [YU et al 2005].

Este método assume que as falhas e erros ocorrem de acordo com uma distribuição pré-determinada, o que permite controlar o tempo, o tipo de falha e o componente afetado no modelo com maior ou menor precisão, dependendo do nível de abstração do simulador [CARREIRA et al 1998]. Assim, se o modelo possuir detalhes suficientes, qualquer sinal pode ser corrompido da forma desejada, com os resultados da corrupção sendo facilmente observáveis, independentemente da localização do sinal no modelo. [YU et al 2005].

Uma das principais vantagens desta técnica [CARREIRA et al 1998], é que ela permite que os fabricantes de sistemas a utilizem mais cedo nos seus projetos para a realização de testes. A simulação também permite injetar falhas muito precisas e coletar informações detalhadas dos seus efeitos, proporcionando alta controlabilidade e observabilidade. Ela também oferece total controle sobre os modelos de falhas e mecanismos de injeção, permitindo modelar falhas transientes e permanentes [YU et al 2005].

A principal desvantagem [CARREIRA et al 1998] desta abordagem é a necessidade de construção de um modelo de simulação preciso, o que pode consumir muito tempo no caso de sistemas complexos. Outro ponto negativo é que ela requer parâmetros de entrada precisos, os quais são difíceis de prover, pois mudanças no projeto e na tecnologia frequentemente complicam a utilização de medições anteriores [HSUEH et al 1997].

2.4.2 Injeção de Falhas Por Hardware

A injeção de falhas implementada em *hardware* [CARREIRA et al 1998] corresponde às técnicas utilizadas para injetar falhas físicas na camada de *hardware* do sistema alvo.

Dependendo do tipo e da localização, os métodos de injeção de falhas em *hardware* podem ser divididos em com e sem contato [HSUEH et al 1997].

O primeiro é a injeção de falhas em *hardware* com contato, onde o injetor tem contato físico direto com o sistema alvo, produzindo alterações de forma externa na voltagem ou corrente dos *chips* alvos. Como exemplo, podem-se citar sondas em nível de pinos e de *sockets*. Este método proporciona um bom controle sobre os tempos e locais das falhas aplicadas, com pouca ou nenhuma perturbação do sistema alvo.

Já o segundo tipo é a injeção de falhas em *hardware* sem contato, na qual o injetor não tem contato físico direto com o sistema alvo. Assim, uma fonte externa produz algum fenômeno físico, como radiação de íons carregados ou interferência eletromagnética, para originar correntes falsas dentro do *chip*. Porém, com este método torna-se difícil precisar o tempo e o local de uma injeção de falhas, porque não se pode controlar o exato momento de uma emissão de íons ou da criação de um campo magnético.

Essa técnica tem a vantagem de injetar falhas físicas reais [CARREIRA et al 1998] e sem a necessidade de qualquer alteração no *software* que está sendo executado. Além disso, o *hardware* também aciona falhas e monitora o seu impacto, proporcionando alta resolução temporal e uma pequena perturbação [HSUEH et al 1997].

No entanto, essa abordagem apresenta algumas desvantagens [CARREIRA et al 1998], [ARLAT et al 2003], como por exemplo, a dificuldade no desenvolvimento de um ambiente de suporte, a realização de experimentos e a necessidade de se utilizar instrumentação especial de *hardware* para o sistema alvo. Esses fatores acarretam em um alto custo de recursos e de tempo para realizar os testes.

2.4.3 Injeção de Falhas Por Software

A técnica de *SWIFI* (*Software-implemented fault injection*) [CARREIRA et al 1998] consiste em utilizar *software* para modificar o estado do *hardware* ou *software* do sistema, fazendo com que o mesmo se comporte como se uma falha real tivesse ocorrido. De acordo com [ARLAT et al 2003], a *SWIFI* é geralmente realizada pela alteração de conteúdos da memória ou de registradores, baseada em modelos especificados para emular as consequências de falhas de *hardware* ou para injetar falhas de *software*. Ela é ideal para ser utilizada no teste de programas ou mecanismos de tolerância a falhas implementados em *software* [CLARK et al 1995], [ARLAT et al 2003].

A *SWIFI* [CARREIRA et al 1998] apresenta vantagens como ser uma alternativa de baixo custo, pois não requer *hardware* especial, sendo assim menos complexa e cara, e requer menos esforço de desenvolvimento do que as outras técnicas. Essa abordagem pode ser também expandida para novas classes de falhas e não causa qualquer problema de interferência física ou risco de danificar o sistema alvo como em uma injeção de falhas em *hardware*. Além disso, ela é a técnica ideal a ser utilizada para alvejar sistemas operacionais e aplicações, o que é difícil de fazer com a injeção de falhas por *hardware* [HSUEH et al 1997].

Os métodos de injeção de falhas por *software* podem ser classificados, conforme o momento em que as falhas são injetadas, em durante o tempo de compilação ou durante a execução [HSUEH et al 1997]. Para injetar falhas em tempo de compilação, as instruções do programa devem ser modificadas antes que o programa seja carregado e executado. Ao invés de injetar falhas dentro do *hardware* do sistema alvo, este método injeta erros no código fonte ou no código *assembly* do programa para emular o efeito de falhas de *hardware*, *software* e transientes. Por outro lado, durante a execução, um mecanismo é necessário para acionar a injeção de falhas. O mecanismo de acionamento das falhas geralmente inclui um cronômetro de *time-out*, que expira em um tempo pré-determinado, acionando a injeção de falhas.

2.4.3.1 Problemas Encontrados em Ferramentas SWIFI

Apesar dessa técnica de injeção de falhas oferecer muitas vantagens, as ferramentas de *SWIFI* ainda possuem muitos problemas [CARREIRA et al 1998], [HSUEH et al 1997] para realizar a injeção de falhas. Alguns desses problemas são descritos a seguir, os quais se procurou evitar durante o desenvolvimento do injetor de falhas proposto nesse trabalho.

Um dos principais problemas é o impacto causado por essas ferramentas de injeção de falhas no sistema alvo. Isso ocorre porque todo ou parte do código da ferramenta precisa ser executado no sistema alvo, isto é, ele se torna parte da carga de trabalho do alvo, ou porque o processador alvo pode necessitar executar em modo de rastreamento, perdendo desempenho.

Já outra dificuldade é em relação à faixa restrita dos disparos de falhas. As falhas são geralmente injetadas pela corrupção da imagem de memória da aplicação, pela inserção de armadilhas ou pela substituição de um conjunto de instruções por outro. Esses métodos estão relacionados à execução de instruções e não à manipulação de dados.

Outro ponto de adversidade é o monitoramento do sistema, necessário para detectar a ativação de falhas ou para coletar informação relevante do impacto de falhas. No entanto, ele é encontrado em poucas ferramentas. O monitoramento pode ser implementado através de

instrumentação extra e requer *hardware* complexo adicional, pela utilização de *softwares* monitores ou pela inserção de instruções de “armadilha” nos locais adequados. Esses dois últimos causam grande sobrecarga de execução e não alcançam um monitoramento detalhado.

A precisão da *SWIFI*, a sua capacidade de emular falhas reais, também é um problema, pois ela é considerada como sendo muito reduzida. Em qualquer abordagem de injeção de falhas é preciso garantir que erros produzidos por injeção sejam tão próximos quanto possível de erros produzidos por falhas reais. O objetivo final da injeção de falhas é a validação dos mecanismos de gerenciamento de falhas. Sendo assim, os conjuntos de erros produzidos por injeções mais exatas irão validar esses mecanismos de uma forma mais precisa. Em geral, níveis mais baixos de abstração provêm uma maior precisão a um custo mais alto e níveis mais altos de abstração provêm vários níveis de precisão a um custo mais baixo.

Essa técnica também possui limitações em relação à precisão, pois as ferramentas de *SWIFI* não atuam diretamente sobre alguns recursos do processador e estruturas, como as linhas de controle de barramento e dispositivos periféricos. Assim nenhuma falha injetada por *SWIFI* pode causar erros nos tempos de controle do barramento de baixo nível, podendo apenas emular as suas consequências através de erros de dados no barramento.

2.4.4 Injeção de Falhas Mista

Na injeção de falhas mistas, é utilizada qualquer combinação das técnicas de injeção de falhas por simulação, *hardware* e *software* [CARREIRA et al 1998]. O mais comum é utilizar a *SWIFI* juntamente com algum *hardware* adicional, para ajudar no processo de injeção de falhas ou rastrear a ativação e propagação das falhas no sistema alvo. Outra possibilidade é utilizar a *SWIFI* com simulação para aproveitar a velocidade do processador alvo e a precisão dos modelos de falhas de baixo nível. Nesse caso também é possível avaliar as propriedades de dependabilidade de um processador [CARREIRA et al 1998].

2.5 Comparativo das Técnicas de Injeção de Falhas por Hardware e Software

A seguir é apresentado um comparativo entre as técnicas de injeção de falhas por *hardware* e *software*, porque elas são as técnicas mais utilizadas para verificar a dependabilidade de sistemas computacionais. A comparação é feita apresentando-se as principais características relacionadas ao uso dos métodos de injeção de falhas por *software* e

hardware no desenvolvimento de injetores de falhas.

A tabela 2.1 resume os métodos de injeção de falhas mais comumente estudados e utilizados para as técnicas de falhas por *hardware* e *software*.

Tabela 2.1 - Principais métodos de aplicação de falhas para as técnicas de injeção de falhas por *hardware* e por *software*

Hardware	**Software**
Circuito aberto	Corrupção dos dados armazenados em registradores, disco e memória.
Bridging	
Bit-flip	Corrupção de dados de comunicação em barramentos e redes de comunicação.
Correntes falsas	
Surto de Tensão	Manifestação de defeitos de *software* no nível de máquina e nos níveis mais altos.
Stuck-at	

Fonte: HSUEH et al 1997 (pg. 3).

Analisando-se essa tabela, percebe-se que os métodos de injeção de falhas por *hardware* são utilizados no nível mais baixo, atuando diretamente sobre o *hardware* do sistema. Já os métodos de injeção de falhas por *software* atuam no nível mais alto, no software do sistema ou nos pacotes de dados transmitidos nos canais de comunicação.

Já a tabela 2.2 compara diversas características envolvidas na utilização dos diversos métodos de injeção de falhas por *hardware* e *software*. Essa comparação é importante para o projeto do injetor de falhas desenvolvido neste trabalho, pois nele foram considerados vários fatores, como, por exemplo, o custo de desenvolvimento, a perturbação causada no sistema alvo, a capacidade de monitoramento dos efeitos das falhas, a controlabilidade dos testes, o disparo das falhas e a repetibilidade do processo.

Observando-se a tabela 2.2, percebe-se que o contraste entre os métodos de injeção de falhas por *hardware* e por *software* está principalmente nos pontos de injeção de falhas que eles podem acessar, no seu custo e no nível de perturbação causado no sistema alvo.

Os métodos de injeção de falhas por *hardware* podem injetar falhas dentro dos pinos de um *chip* ou em componentes internos, tais como circuitos combinacionais e registradores que não são endereçáveis por *software* [HSUEH et al 1997]. Já os métodos de injeção de falhas por *software* são convenientes para produzir mudanças diretamente no estado de *software*, como alterar o conteúdo da memória e de registradores [HSUEH et al 1997].

Assim, métodos de *hardware* são utilizados para avaliar a detecção de erros em baixo nível e em mecanismos de mascaramento. Já os métodos de *software* são mais adequados para

testar mecanismos de alto nível. Esses métodos de *software* possuem um menor custo, mas eles também podem causar uma interferência maior porque pode ser preciso integrá-los no sistema alvo [HSUEH et al 1997].

Tabela 2.2 - Características dos métodos de injeção de falhas para as técnicas de injeção de falhas por *hardware* e por *software*

	Hardware		**Software**	
	Com Contato	**Sem Contato**	**Compilação**	**Execução**
Custo	Alto	Alto	Baixo	Baixo
Perturbação	Nenhum	Nenhum	Baixo	Alto
Risco de danos	Alto	Baixo	Nenhum	Nenhum
Monitoramento e Resolução de tempo	Alto	Alto	Alto	Baixo
Acessibilidade dos pontos de injeção de falhas	Pinos do chip	Interno ao chip	Registrador Memória Software	Registrador Memória Controlador de I/O
Controlabilidade	Alto	Baixo	Alto	Alto
Disparo de Falhas	Sim	Não	Sim	Sim
Repetibilidade	Alta	Baixo	Alto	Alto

Fonte: HSUEH et al 1997 (pg. 7).

Neste trabalho pretende-se desenvolver um injetor de falhas que atue sobre as mensagens de dados do protocolo *PROFIsafe* para avaliar os seus mecanismos de tolerância a falhas (mecanismos de alto nível). Também se tem por objetivo desenvolver um injetor de falhas de custo não muito elevado, tornando viável a sua utilização no projeto *RIO-SIL* e em outros projetos do ponto de vista econômico.

Outro ponto importante para o injetor é a controlabilidade do processo de injeção de falhas, pois se deseja saber quando, onde e o tipo de falha que estará sendo injetada em um determinado momento.

Por essas razões citadas acima e pelas análises apresentadas anteriormente sobre as técnicas de injeção de falhas por *software* e por *hardware*, constatou-se que a *SWIFI* é a mais adequada para o desenvolvimento do injetor de falhas apresentado neste trabalho.

3 PROTOCOLO DE COMUNICAÇÃO SEGURO PROFISAFE

Nesta seção, é feita uma descrição geral dos principais aspectos da norma de segurança funcional *IEC 61508*, na qual a especificação do protocolo *PROFIsafe* é baseada. A seguir são apresentados o conceito geral, as principais características, mecanismos de detecção de falhas, os principais componentes e o funcionamento geral do protocolo.

3.1 Norma de Segurança Funcional IEC61508

A norma *IEC 61508* [IEC 61508] trata da Segurança Funcional de Sistemas Elétricos, Eletrônicos e Eletrônicos Programáveis (E/E/EP) e serve como base para outras normas relacionadas com segurança. Desta forma, a base para a especificação dos protocolos de comunicação seguros também está contida nela. O principal objetivo desta norma é orientar o trabalho de equipes técnicas envolvidas no desenvolvimento de produtos e aplicações de segurança. No caso dos sistemas relacionados à segurança, a norma cobre todas as partes do sistema necessárias para executar a função de segurança.

Para classificar os sistemas, a norma especifica quatro níveis de desempenho para uma função de segurança, os quais são chamados de Níveis de Integridade de Segurança, *SIL* (*Safety Integrity Level*). O nível mais baixo corresponde ao *SIL* 1 enquanto que o mais alto corresponde ao *SIL* 4. Os requisitos necessários para se atingir um determinado *SIL* são mais rigorosos à medida em que se aumenta o *SIL*, indicando uma menor probabilidade de ocorrência de falhas perigosas – aquelas que podem levar o sistema a apresentar defeito.

Quando a função de segurança depende da comunicação de dados, deve-se estimar a taxa de erros residuais deste processo de comunicação e considerá-la na determinação das taxas de falhas daquela função de segurança. A norma define função de segurança *como uma função a ser implementada por um sistema E/E/EP relacionado com a segurança ou com outras medidas de redução de riscos, que se destina a alcançar ou manter um estado seguro para o equipamento controlado, em relação a um evento perigoso específico*.

A taxa de falha *é um parâmetro de confiabilidade de um componente ou sistema que representa a probabilidade desse componente ou sistema vir a falhar em um determinado tempo "t"*. Já a taxa de erros residuais *corresponde à razão entre a quantidade de dados recebidos incorretamente e não detectados ou não corrigidos pelo equipamento de controle de erro, para o total de dados correspondentes enviados.*

Para isso, a norma especifica que devem ser considerados erros de transmissão como repetições, perda de mensagens, mensagens inseridas, pacotes entregues fora de ordem, mensagens com dados corrompidos, atrasos no recebimento de pacotes, e mascaramento que corresponde a não identificação correta do emissor da mensagem.

Alternativamente, pode-se utilizar a abordagem do *Black Channel*. Trata-se da implementação de um protocolo seguro que implementa os mecanismos para a detecção dos erros de comunicação originados nos dispositivos e protocolos usados no seu transporte [IEC 61508]. Dessa forma, o sistema poderá ser colocado em um estado seguro, caso ocorra algum erro de comunicação. A principal vantagem do uso do *Black Channel* é que partes do canal de comunicação não precisam ser projetadas ou validadas de acordo com a *IEC 61508*. Essas partes formam o *Black Channel*.

3.2 Conceito Geral e Principais Características do PROFIsafe

É um protocolo de comunicação seguro popular na indústria, com uma base instalada em torno de quatro milhões de dispositivos até o final do ano de 2015. Foi desenvolvido pela *Profibus & Profinet Internacional* [PI 2015] para ser utilizado com as redes *PROFIBUS* e *PROFINET*. O uso do protocolo na comunicação de equipamentos e sistemas relacionados à segurança desenvolvidos para estas redes permite a esses componentes serem utilizados no desenvolvimento de funções de segurança.

A especificação do *PROFIsafe* [PROFIsafe 2010] está em conformidade com a norma *IEC 61508*, atendendo os requisitos para sistemas com até *SIL* 3 e *FSCP 3* (*Functional Safety Communication Profile – 3*) da *IEC 61784-3* [IEC 61784-3]. Porém, quando o protocolo é utilizado no desenvolvimento de dispositivos de segurança para o setor de óleo e gás que precisam ser certificados para *SIL* 3, todos os componentes destes dispositivos, inclusive o *PROFIsafe* precisam ser certificados pela Agência de Inspeção Técnica alemã *TÜV SÜD*.

O *PROFIsafe* reduz a probabilidade de erros nos dados transmitidos entre um *F-Host* (controlador seguro) e um *F-Device* (dispositivos com segurança integrada) para o nível exigido pela norma. Ele também permite que dispositivos relacionados à segurança possam se comunicar paralelamente com dispositivos não seguros no mesmo barramento. Contudo, é preciso garantir que os dois tipos de processamento ocorram de forma logicamente isolada.

O protocolo corresponde a uma camada acima do protocolo de transporte e, dessa forma, não influencia e nem perturba os serviços das camadas subjacentes. Além disso, ele é o

mais independente possível dos canais físicos de transmissão, sejam eles fios de cobre, fibras ópticas, *wireless* ou *backplanes* (grupo de conectores ligados de maneira a formar um barramento). Este princípio é chamado de *Black Channel*.

O mecanismo do *Black Channel* torna desnecessária a avaliação de segurança dos elementos que o compõe: *backplanes* individuais, caminhos de transmissão e redes *PROFIBUS* e *PROFINET*. Essa "proteção" de todo o caminho, desde a geração do sinal seguro em um dispositivo remoto de entradas e saídas até o seu destino onde será processado em controlador seguro (*F-Host*), é obtida através dos mecanismos de proteção implementados pelo protocolo seguro [PROFIsafe 2010].

3.3 Modelo de Falhas

O Modelo de Falhas considerado na especificação do *PROFIsafe* é aquele previsto na norma *IEC 61508*. Esse modelo é composto pelas falhas de repetição, perda, inserção, sequência incorreta, endereçamento, corrupção de dados, atraso, mascaramento e falhas de memória em *switches*.

A falha de repetição de mensagem ocorre quando um dispositivo defeituoso faz com que mensagens de segurança velhas e obsoletas sejam repetidas no momento errado, o que pode causar uma perturbação perigosa no receptor. Por exemplo, uma porta de segurança pode ser reportada como fechada quando na verdade ela já foi aberta.

Já a falha de perda de pacote ocorre quando um dispositivo em estado de mau funcionamento descarta uma mensagem de segurança aleatoriamente, como por exemplo, um pedido para uma parada de segurança. Por outro lado, a falha de inserção acontece devido a um mau funcionamento em um dispositivo de barramento, o qual insere por conta própria uma mensagem de segurança. Como exemplo, o dispositivo pode inserir uma mensagem para cancelar uma parada de segurança operacional.

Já a falha de sequência incorreta é causada por um dispositivo com defeito que modifica a ordem de sequência das mensagens de segurança. Por exemplo, antes de iniciar uma parada de segurança operacional, é aconselhável reduzir com segurança a velocidade de rotação de um equipamento. Se a ordem dessas mensagens for trocada, a máquina ainda estará funcionando ao invés de estar desligada.

A falha de endereçamento, por sua vez, é causada por um dispositivo em mau funcionamento que modifica o endereço de destino das mensagens de segurança. Nesse

estado de falha, o emissor envia uma mensagem para o receptor, o qual descarta a mensagem por ela ter um endereço de destino diferente do dele.

A corrupção de dados ocorre devido a um mau funcionamento em um dispositivo de barramento ou no *link* de transmissão que perturba mensagens seguras. Nesse tipo de falha, o emissor envia um dado correto para o receptor e este recebe um dado corrompido.

Por outro lado, o atraso de mensagens ocorre quando uma troca de mensagens causa uma situação que leva à execução atrasada de um serviço. Em geral, essas situações ocorrem quando a troca operacional de dados excede a capacidade do *link* de comunicação e as mensagens gastam mais tempo para circularem do que o previsto.

Já o mascaramento acontece por causa de um dispositivo de barramento defeituoso que faz com que mensagens com e sem relevância para a segurança sejam misturadas.

Finalmente, as falhas de memória que podem ocorrer em *switches* são de mensagens perdidas, atrasadas, duplicadas, enviadas para o destinatário errado ou que tenham o seu conteúdo corrompido. Além disso, um *switch* pode seguir enviando continuamente mensagens armazenadas mesmo quando o emissor já tiver sido desligado.

3.4 Mecanismos de Tolerância a Falhas

Com o objetivo de detectar os erros de comunicação descritos no modelo de falhas, o protocolo possui as seguintes funcionalidades [PROFIsafe 2010]: número sequencial de mensagens, controle de temporização, identificador único entre emissor e receptor e *CRC* (*Cyclic Redundancy Check*). A tabela 3.1 relaciona os tipos de falhas com o(s) respectivo(s) mecanismo(s) para sua detecção.

O número de sequência (consecutivo) é uma técnica através da qual um receptor pode identificar se recebeu ou não as mensagens na ordem correta. Ele possui várias funções, como detectar repetição não intencional, perda de mensagens, inserção de pacotes, sequência incorreta de mensagens e erros de retransmissão de mensagens armazenadas em *switches*.

Já o controle de temporização é um mecanismo utilizado para detectar atrasos não permitidos na comunicação. Dessa forma, as mensagens só podem chegar aos seus destinos dentro de uma faixa pré-determinada de tempos. Ele possui as funções de detectar falhas de perda de mensagens, inserções em mensagens, atraso inaceitável e mascaramento.

Por outro lado, o identificador único de conexão (*codename*) é utilizado para detectar mensagens mal encaminhadas entre transmissor e receptor. Esse identificador é conhecido

unicamente pelos dois dispositivos envolvidos em uma comunicação e é estabelecido a cada conexão. Ele possui a função de detectar falhas em mensagens com endereçamento incorreto entre segmentos da rede, mascaramento e inserções de mensagens.

Tabela 3.1 - Medidas de segurança adotadas pelo *PROFIsafe* para detectar os vários tipos de erros de transmissão

Erros de Comunicação	Medidas de Segurança			
	Número Consecutivo	Controle de Temporização	Identificador Único	CRC
Repetição não Intencional	X			
Perda	X	X		
Inserção	X	X	X	
Sequência Incorreta	X			
Corrupção				X
Atraso Inaceitável		X		
Endereçamento			X	
Mascaramento		X	X	X
Erros de Memória em Switches	X			

Fonte: *PROFIsafe System Description* (2010, p. 30).

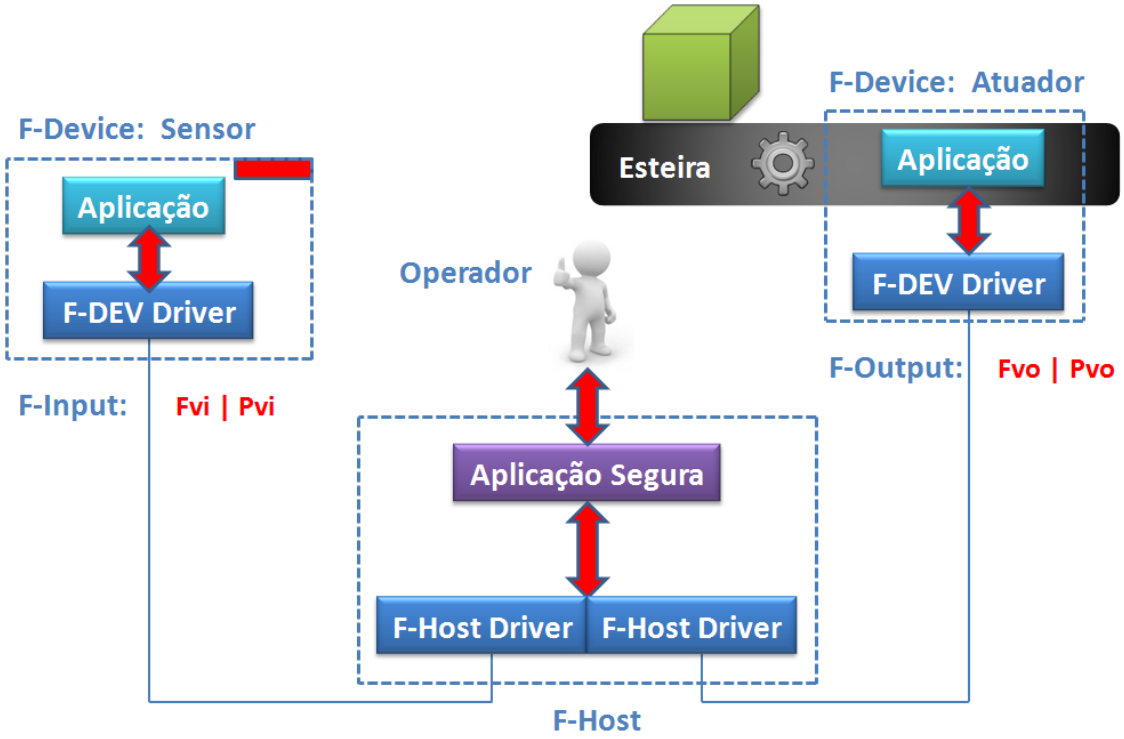
Por fim, o *CRC*, checagem de redundância cíclica, é utilizado para detectar *bits* de dados corrompidos nas mensagens durante as transmissões de dados ou por algum módulo seguro defeituoso que alterou indevidamente informações nas mensagens seguras. No *PROFIsafe*, o *CRC* também é utilizado para detecção de mascaramento de mensagens.

3.5 Exemplo de Utilização do Protocolo PROFIsafe

A figura 3.1 ilustra um exemplo de uma aplicação do *PROFIsafe* na automação de processos industriais [PROFIsafe 2010]. Neste exemplo um operador em uma planta industrial monitora o funcionamento de um sensor e de uma esteira através de uma aplicação de segurança. Essa aplicação opera no *F-Host*, dispositivo mestre *PROFIsafe*, que controla os dois dispositivos escravos (*F-Devices*).

Um desses dispositivos é um sensor seguro, responsável pelos dados de entrada (*F-Input*), o qual pode, por exemplo, detectar vazamento de gás, aumento de temperatura, princípio de incêndio, etc. Já o outro dispositivo escravo controla um atuador, o motor, de uma esteira em uma linha de montagem, encarregado de executar os dados de saída (*F-Output*) enviados pelo *F-Host*, como por exemplo, girar a esteira a uma velocidade específica.

Figura 3.1 - Exemplo de uma Aplicação Prática do *PROFIsafe*



Fonte: Modificado a partir da *PROFIsafe System Description* (2010, p. 39,41).

O sensor monitora continuamente o ambiente da fábrica, e, a cada instante especificado, avisa a aplicação segura no *F-Host* sobre o estado de ameaças no ambiente de operação através dos dados de operação de processo. Enquanto o sensor não identifica qualquer problema, ele envia dados de operação normal (*PVi*) através do seu *F-Device driver* para a aplicação segura. Esta, por sua vez, interpreta os valores recebidos e determina que a esteira deva seguir girando na velocidade informada pelos seus dados de saída (*PVo*) repassados ao *F-Device driver* através do *F-Host*.

Contudo, quando um problema ocorre, como, por exemplo, detecção de um princípio de incêndio pelo sensor, a aplicação desse *F-Device* informa a aplicação segura no *F-Host*, que o *F-Host driver* vai enviar dados de entrada seguros para ela (*FVi*) para cada entrada recebida do *F-Device* sensor enquanto houver um problema. Ao receber essa informação, a aplicação no *F-Host* envia para o *F-Device* atuador, um dado de saída seguro de operação (*FVo*) para o motor da esteira, informando que ele deve passar a operar em um estado seguro, provavelmente uma parada de segurança.

O sistema fica nesse estado seguro até que os operadores da fábrica resolvam os problemas e reiniciem a operação através da aplicação segura no *F-Host*. Se a comunicação do *F-Device* do motor da esteira falhar, ela irá automaticamente para o estado seguro,

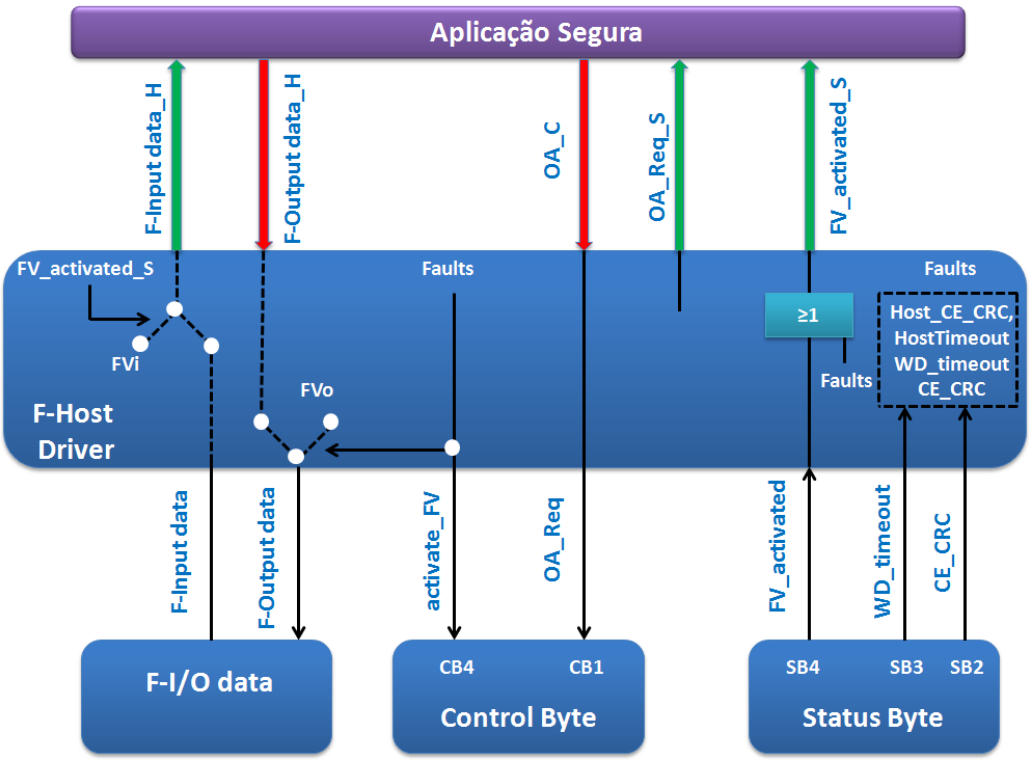
independente da aplicação segura ou do operador. Da mesma forma, se a comunicação do *F-Device* do sensor falhar, a aplicação segura vai determinar que o *F-Device* do motor da esteira vá para o estado seguro.

3.6 Bits de Interação com a Aplicação

A seguir são apresentados os sinais utilizados pela aplicação segura para controlar os dispositivos *F-Host* e *F-Device* que foram implementados neste trabalho [PROFIsafe 2010]. Esses sinais são responsáveis por enviar comandos de controle para os dispositivos *F-Device* ou para reportar o estado de operação desses dispositivos.

Na figura 3.2 a ocorrência de falhas (*Faults*) é detectada através das variáveis *Host_CE_CRC=1* (*F-Host* detectou erro de *CRC* em um *ack* recebido do *F-Device*), *HostTimeout=1* (*F-Host* detectou *Timeout*), *WD_timeout=1* (*F-Device* detectou *Timeout*) ou *CE_CRC=1* (*F-Device* detectou erro de *CRC* em uma mensagem recebida do *F-Host*).

Figura 3.2 - Interface entre a Aplicação Segura e o *F-Host Driver*



Fonte: *PROFIsafe System Description* (2010, p. 39).

Sempre que ocorrer uma falha, a variável *activate_FV* que corresponde ao *bit* 4 do *Control Byte* recebe o valor "1", o *F-Host* passa a enviar dados seguros de saída (*Fvo*) para o

F-Device, indicando que este dispositivo trabalhe em um estado seguro. Caso contrário o dado de saída é o valor normal do processo (*PVo*).

O sinal *FV_activated_S* do *F-Host* recebe o valor "1" quando ocorrer uma falha (*Faults*) ou quando o *bit* 4 do *Status Byte* recebido do *F-Device* possuir o valor "1". Nesse caso, esse sinal indica que o *F-Host driver* está enviando valores seguros de entrada (*FVi*) para a aplicação segura no *F-Host* para cada dado de entrada do *F-Device,* se ele for um dispositivo de entrada. Se o *F-Device* for um dispositivo de saída, essa variável indica que o *F-Host* está enviando dados seguros de saída (*FVo*) para o *F-Device*. Caso o valor dessa variável seja "0", ela indica para a aplicação segura que o *F-Device* está enviando dados normais de operação de processo (*PVi*) para ela ou recebendo dados normais de saída de operação de processo (*PVo*) do *F-Host driver*.

Já o sinal *OA_C* (*Operator Acknowledgment*), quando possui o valor "1", permite que o usuário reinicie a função de segurança depois de uma reação a uma falha (loop de controle de segurança) através da aplicação segura no *F-Host*. O sinal *OA_Req_S* indica um pedido de confirmação antes de reiniciar a função de segurança.

O *F_input_data_h* sinaliza que a aplicação do *F-Host* recebe valores de processo (*PVi*) do *F-Device* caso tudo esteja normal, ou valores seguros (*FVi*) em caso de falha. Já o *F_output_data_h* indica que a aplicação do *F-Host* envia valores de processo (*PVo*) para o *F-Device* caso tudo esteja normal, ou valores seguros (*FVo*) em caso de falha.

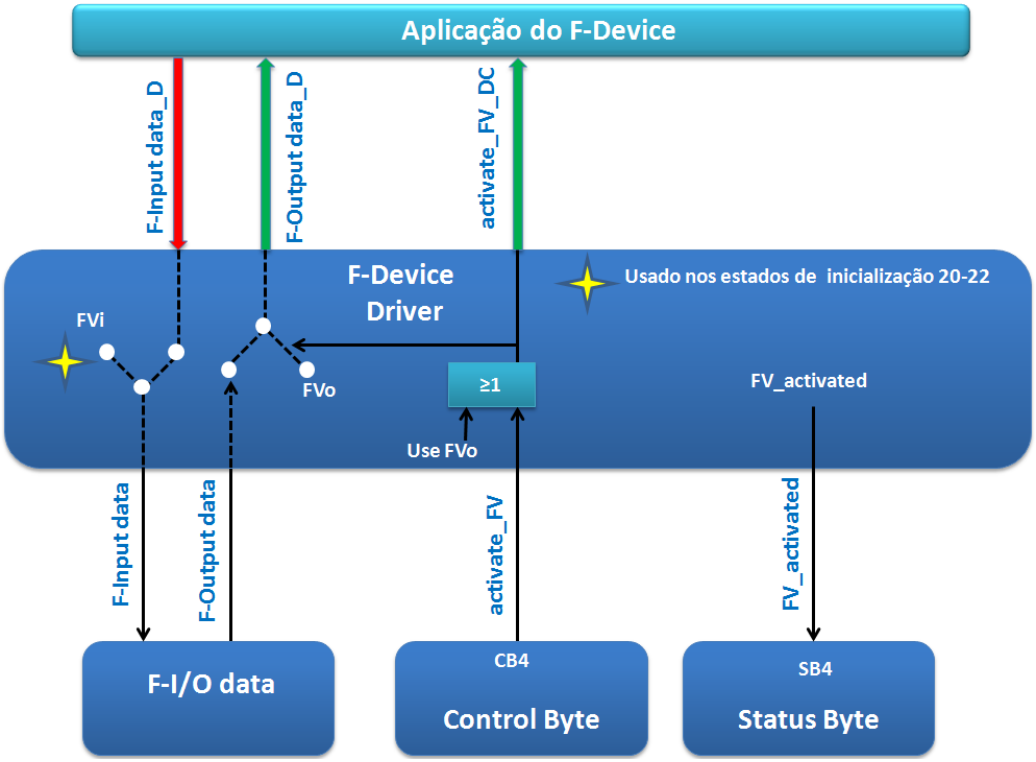
Na figura 3.3, o sinal *activate_FV_DC* recebe o seu valor do *bit* 4 do *Control Byte* (*activate_FV*) ou da variável *use_FVo*, que recebe o valor "1" quando ocorre uma falha no *F-Device*. Quando o *activate_FV_DC* possui o valor "1", ele indica para a aplicação do *F-Device* que devem ser utilizados dados de saída seguros (*FVo*). Caso contrário os dados de saída serão os valores normais de operação do processo (*PVo*).

A variável *FV_activated* corresponde ao *bit* 4 do *Status Byte* e possui o valor "1" na inicialização do sistema e sempre que ocorrer uma falha na comunicação. Quando ela tem o valor "1", ela indica que o *F-Device driver* vai entregar dados seguros de saída (*FVo*) para a aplicação no *F-Device* se ele for um dispositivo de saída. Se o *F-Device* for um dispositivo de entrada, ela indica que a aplicação segura do *F-Host* está recebendo valores seguros de entrada (*FVi*) do *F-Host driver* (através do sinal *FV_activated_S*).

O *F_input_data_D* indica que a aplicação do *F-Device* (*F_input* no caso) fornece valores de processo *PVi* em estado normal de operação ou *FVi* nos estados de inicialização. Já o *F_output_data_D* sinaliza que a aplicação do *F-Device* (*F_output* no caso) recebe valores

seguros em casos de falha *FVo*, ou valores de processo, *PVo*, gerados pela aplicação do *F-HOST* durante operação normal.

Figura 3.3 - Interface entre a Aplicação do *F-Device* e o *F-Device Driver*



Fonte: *PROFIsafe System Description* (2010, p. 41).

3.7 F-Parameters

Os *F-Parameters* são parâmetros seguros enviados do *F-Host* para o *F-Device* durante a inicialização do sistema [PROFIsafe 2010]. Eles contêm informações para a camada do protocolo *PROFIsafe* ser configurada de acordo com as necessidades específicas dos dispositivos e para verificar a correção de atribuições. Os *F-Parameters* são definidos nos arquivos *GSD* (*General Station Description*) e os seus valores são protegidos por uma assinatura *CRC* para evitar corrupção de dados nas mídias de armazenamento.

Após a transferência, os dispositivos *F-Host* e *F-Device* utilizam os mesmos *F-Parameters* para calcular o *CRC1*, com um tamanho de 2 *bytes*, e que será utilizado posteriormente no cálculo do *CRC2* enviado em cada *ack* e mensagem. Essa assinatura *CRC2* é utilizada para verificar se ocorreram erros nas mensagens ou *acks* enviados e recebidos. Por essa razão, os parâmetros seguros precisam ser idênticos entre cada par *F-Host* e *F-Device* para gerar o mesmo valor de *CRC1*. Os principais parâmetros seguros são o *F_S/D_Address*,

F_WD_Time, *F_SIL*, *F_iPar_CRC*, *F_Par_CRC*. A lista completa dos *F-Parameters* é apresentada na página 70 da especificação do protocolo *PROFIsafe* [PROFIsafe 2010].

3.8 Mensagem PROFIsafe

Para a transmissão de informações entre os dispositivos mestre e escravo, o protocolo oferece duas opções para o tamanho dos dados seguros a serem enviados dentro do *Safety PDU* (pacote *PROFIsafe*), os quais requerem diferentes proteções de *CRC* [PROFIsafe 2010]. Na primeira opção são utilizados 12 *bytes* de dados com um *CRC2* de 24*bits*, como é apresentado na figura 3.4, sendo que eles são usados na automação industrial que trabalha com entrada e saída de *bits*.

Figura 3.4 - Estrutura do *Safety PDU* com 12 *bytes* de dados

Fonte: *PROFIsafe System Description* (2010, p. 43).

Já para a segunda opção são utilizados 123 *bytes* de dados com um *CRC2* de 32*bits* e eles são utilizados na automação de processos, a qual trabalha com valores maiores de entrada e saída (números em ponto flutuante).

Neste trabalho foi utilizada uma implementação da versão V2 do Protocolo *PROFIsafe* com 12 *bytes* de dados no *safety PDU*. Nessa versão, diferentemente da V1, o número consecutivo não é transmitido em cada *safety PDU*. Ao invés disso, são utilizados contadores locais de 24*bit*s para o número consecutivo tanto no *F-Device* (*Vconsnr_d*) quanto no *F-Host* (*Vconsnr_h*) e um *Toggle Bit* dentro do *Control/Status Byte* para incrementar esses contadores sincronamente. A verificação para exatidão e sincronismo dos dois contadores independentes é feita pela inclusão dos números consecutivos no cálculo do *CRC2*.

O *CRC2* é enviado em cada *safety PDU* para verificação da integridade dos dados transmitidos. Ele é computado sobre um vetor de 18 *bytes* (*Buffer_CRC*), formado por *CRC1*, o *byte* de *Control/Status*, os dados a serem transmitidos e o número consecutivo. Nesse vetor, apenas a ordem original do número consecutivo não é mantida, pois ele deve estar invertido. Ou seja, primeiro vem o *byte* menos significativo, depois o *byte* do meio e por fim o *byte* mais

significativo da variável original de 3 *bytes*. Esse vetor é ilustrado na figura 3.5 a seguir.

Figura 3.5 - Estrutura do *Buffer_CRC* para o cálculo do *CRC2*

CRC1	Control/Status Byte	Dados	Nº Consecutivo
(2 bytes)	(1 byte)	(12 bytes)	(3 bytes)

Fonte: *PROFIsafe System Description* (2010, p. 43).

O *Control/Status Byte* possui o tamanho de 1 *byte* em ambas as opções de quantidades de dados a serem enviados. O *Control byte*, apresentado na figura 3.6, contém os comandos de controle enviados do *F-Host* para o *F-Device*.

Figura 3.6 - Estrutura do *Control Byte*

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
res	res	Toggle Bit	Fail-safe values (FV) to be activated	Use F_WD_Time_2 (secondary watchdog)	Reset Vconsnr_d	Operator acknowledge requested	iParameter assignment deblocked
–	–	Toggle_h	activate_FV	Use_TO2	R_cons_nr	OA_Req	iPar_EN

Fonte: *PROFIsafe System Description* (2010, p. 45).

O *Bit* 0 é o sinal *iPar_EN* e é configurado pela aplicação segura executando no *F-Host* no caso de um pedido de parametrização, ou seja, quando o *F-Device* precisa de novos *iParameters*. Já o sinal do *Bit* 1, *OA_Req*, é controlado pelo *F-Host driver* e corresponde a variável "*OA_Req_S*". Esse sinal não é relacionado à segurança e deve ser utilizado pelo *F-Device* para indicar localmente um pedido para *operator acknowledgement* (*OA_C*).

Por outro lado, o *Bit* 2 do sinal *R_cons_nr* é assinalado quando o *F-Host* detecta um erro de comunicação, por ele mesmo ou pelo *Status Byte*. Como consequência o contador do número consecutivo no *F-Device* deve receber o valor "0". Este *bit* deve receber o valor "0" depois que o erro já tiver sido corrigido. O *Bit* 3 é configurado quando o *F-Host* é informado de que está acontecendo uma atualização de configuração durante o funcionamento de um dispositivo seguro ou mesmo uma manutenção de um sistema tolerante a falhas.

Já o *Bit* 4 corresponde ao sinal "*activate_FV*" e pode ser usado para forçar as saídas de um *F-Device* para serem do tipo seguras. O *Bit* 5 representa o *Toggle_h*, o qual é utilizado para incrementar o número consecutivo no *F-Device*. Finalmente, os *Bits* 6 e 7 do *Control Byte* são reservados para futuras atualizações do protocolo.

O *Status Byte*, mostrado na figura 3.7 a seguir, é enviado do *F-Device* para o *F-Host* e contém o estado atual do dispositivo escravo.

Figura 3.7 - Estrutura do *Status Byte*

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
res	Vconsnr_d has been reset	Toggle Bit	Fail-safe values (FV) activated	Communi-cation fault: WD-timeout	Communi-cation fault: CRC	Failure exists in F-Device or F-Module	F-Device has new iParameters values assigned
–	Cons_nr_R	Toggle_d	FV_activated	WD_timeout	CE_CRC	Device_Fault	iPar_OK

Fonte: *PROFIsafe System Description* (2010, p. 44).

O *Bit* 0 do *Status Byte* corresponde ao sinal "*iPar_OK*" e é utilizado quando novos valores para os parâmetros da tecnologia são atribuídos. Já o *Bit* 1 é o sinal "*Device_Fault*", o qual é utilizado para sinalizar quando um dispositivo não é capaz de garantir a integridade de segurança dos dados a serem transmitidos.

Por outro lado, o *Bit* 2 representa o sinal "*CE_CRC*" e ele é usado quando o *F-Device* reconhece uma falha na comunicação segura, isto é, o número consecutivo está errado (detectado através do *CRC2*) ou a integridade dos dados foi violada (erro de *CRC*). O *Bit* 3 corresponde ao sinal "*WD_timeout*", e ele é assinalado sempre que o tempo de *watch dog* decorrer no *F-Device* durante a espera de mensagens seguras.

Já o *Bit* 4 do *Status Byte* é o sinal "*FV_activated*"e ele recebe o valor "1" durante a inicialização do protocolo e nos casos de qualquer erro de comunicação. O *Bit* 5 representa o sinal do *Toggle_d* e ele é utilizado para incrementar o número consecutivo no *F-Host.*

O *Bit* 6, "*Cons_nr_R*", possui o valor "1" sempre que o *F-Device* atribuir o valor "0" para o contador do seu número consecutivo. Por fim, o *Bit* 7 é reservado e não é utilizado.

Finalmente, os dados contidos no pacote são os dados atuais sendo transmitidos entre os dispositivos mestre e escravo *PROFIsafe*. Esses mesmos dados transmitidos no *Safety PDU* são utilizados no cálculo do *CRC2* no *Buffer_CRC*.

3.9 Funcionamento das Máquinas de Estados do F-Host e do F-Device

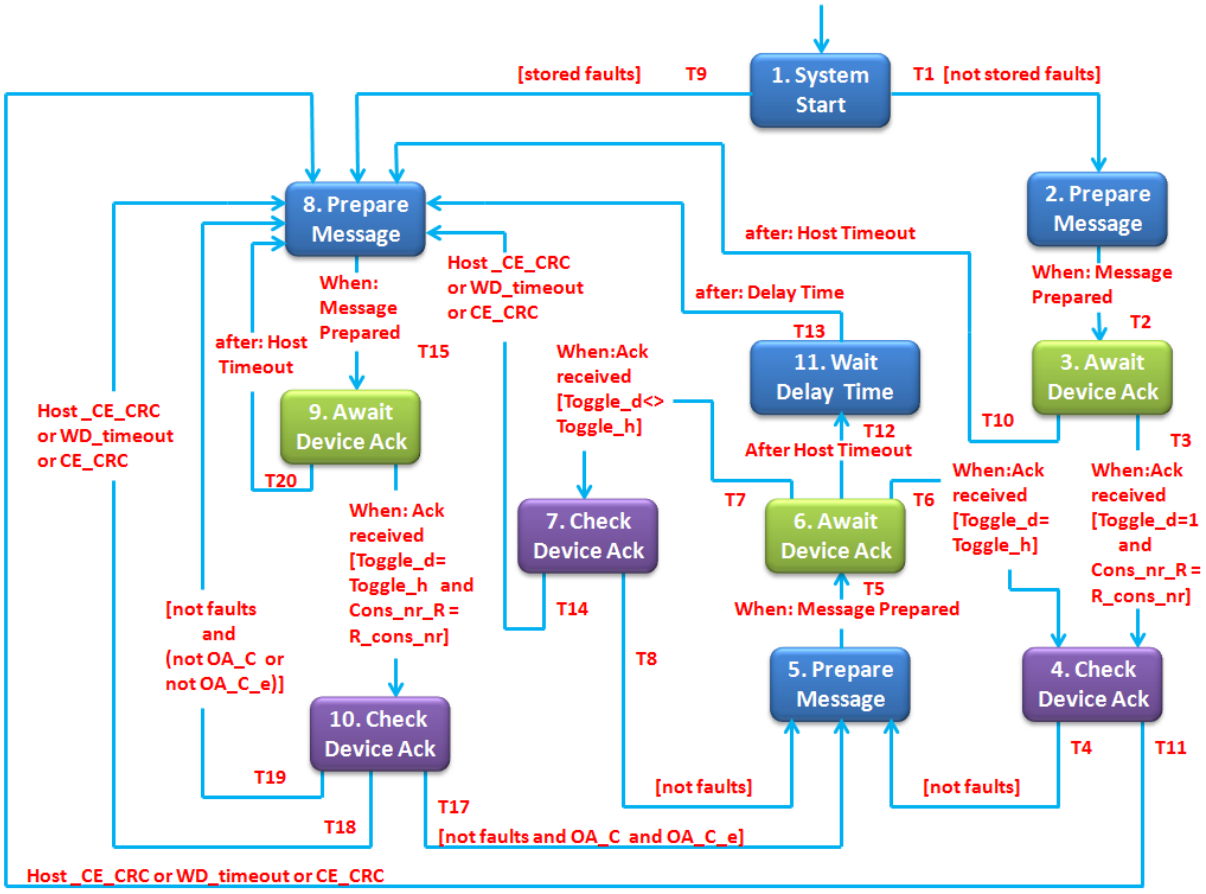
A seguir é feita uma descrição geral do funcionamento das máquinas de estados dos dispositivos *F-Host* e *F-Device*. A descrição completa e detalhada é encontrada na especificação do *PROFIsafe* [PROFIsafe 2010]. As transições entre os estados ocorrem em função do envio e recebimento de mensagens e *acks* entre os dispositivos, do resultado da

verificação de erros de *CRC* nessas mensagens e *acks*, dos *bits* do *Control* e *Status Bytes* e da ocorrência ou não de *timeout* nos dispositivos *PROFIsafe*.

3.9.1 Máquina de Estados do *F-Host*

A máquina de estados do dispositivo mestre *PROFIsafe*, o *F-Host*, é apresentada na figura 3.8. Na inicialização do *F-Host*, os valores iniciais do *safety PDU* e as variáveis são todas inicializadas com o valor "0". A exceção é o número consecutivo, o qual recebe como valor inicial o número 0xFFFFF0. A variável *HostTimeout* é responsável por detectar um *timeout* local enquanto o *F-Host* espera por um *ack* do *F-Device*. Já a variável *Host_CE_CRC* pode reconhecer ou não um erro de *CRC* enquanto analisa o *safety PDU* recebido.

Figura 3.8 - Diagrama de Estados do *F-Host*



Fonte: *PROFIsafe System Description* (2010, p. 45).

Por outro lado, o sinal *Device_Fault* indica que o *F-Device* reportou uma falha ao *F-Host* através do *Status Bit 1*=1. O sinal *CE_CRC* avisa que o *F-Device* reportou uma falha de *CRC* para o *F-Host* através do *Status Bit 2*=1. Já o sinal *WD_timeout* representa um aviso

quando o *F-Device* reporta uma falha de *timeout* para o *F-Host* por meio do *Status Bit 3*=1. A variável *not faults* possui o valor "1" se nenhum dos *bits Host_CE_CRC*, *CE_CRC* ou *WD_timeout* possui valor "1". Já a variável *stored faults* vai possuir o valor "1" se qualquer dos *bits Host_CE_CRC*, *HostTimeout*, *CE_CRC* ou *WD_timeout* possui valor "1".

Depois de ser inicializado, o *F-Host* envia uma mensagem para o *F-Device* e ele permanece em estado de espera pelo *ack* do *F-Device*. Quando o *F-Host* receber esse *ack* do *F-Device*, ele o verificará com relação à ocorrência de erros. Se não ocorrer nenhum erro de comunicação, o *F-Host* entra no seu ciclo normal de funcionamento (estados 5, 6 e 4), no qual ele permanece trocando mensagens com o *F-Device*.

Nos estados 3, 6 e 9 de "Espera por um *ack* do *F-Device*", se ocorrer *timeout* no *F-Host* na espera por um *ack* do *F-Device*, o *F-Host* entra nos estados 8, 9 e 10 para tratamento de falha. Já nos estados 4, 7 e 10 de "Verificação do *Ack* do *F-Device*", se ocorrer uma falha, o *F-Host* também realiza a transição para os estados 8, 9 e 10 para iniciar o tratamento de uma falha. A partir dessa situação de tratamento de falha, o *F-Host* só pode voltar ao seu ciclo normal de operação a partir do estado 10, quando não houver mais falha e o operador executar o *operator acknowledgement* a partir da aplicação segura no *F-Host*.

3.9.2 Máquina de Estados do *F-Device*

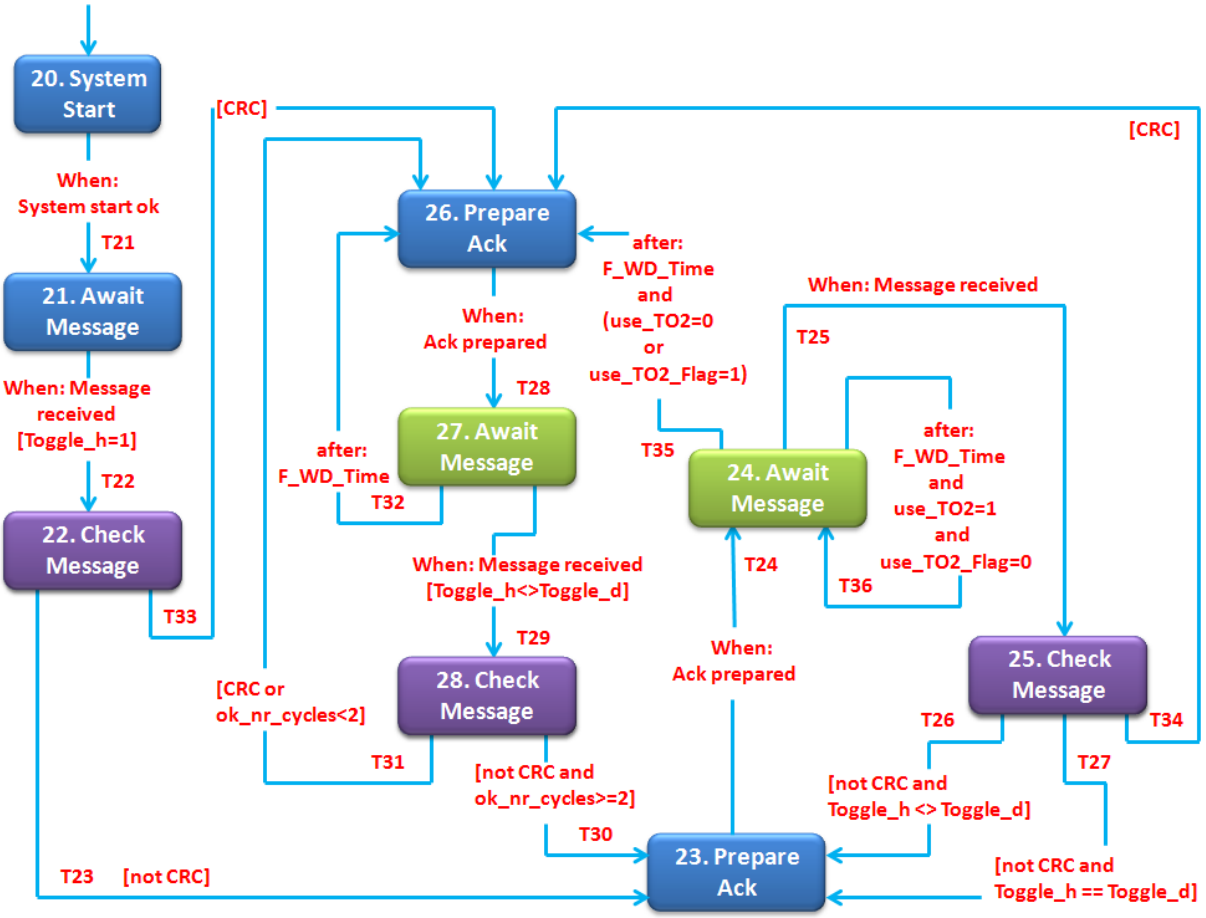
A figura 3.9 apresenta a máquina de estados do dispositivo escravo *PROFIsafe*, o *F-Device*, juntamente com os termos utilizados nela. Na inicialização do *F-Device*, os valores iniciais do *safety PDU* e as variáveis são todas inicializadas com ao valor "0". As exceções são o número consecutivo, que é inicializado com 0xFFFFF0 e o sinal *FV_activated*, que recebe o valor "1". A variável *HostTimeout* é responsável por detectar um *timeout* local enquanto o *F-Host* espera por um *ack* do *F-Device*. Já a variável *Host_CE_CRC* pode reconhecer ou não um erro de *CRC* enquanto o *F-Host* analisa o *ack* recebido do *F-Device*.

A notação *UML [Toggle_h == Toggle_d]* é utilizada para realizar uma transição entre dois estados. Ela significa que o *Bit Toggle_h* não mudou de valor (*not Toggled*). Em sentido oposto, a notação *[Toggle_h <> Toggle_d]* indica que o *Bit Toggle_h* mudou de valor (*Toggled*). Por sua vez, a notação *[CRC]* significa que o *F-Device* reconheceu uma falha de *CRC* (na comunicação e/ou erro de número consecutivo).

O sinal *F_WD_Time* representa o tempo de *watchdog* definido pelo parâmetro seguro *F_WD_Time*. Já o sinal *use_TO2* (3º *bit* do *Control Byte*) estabelece o uso de um segundo tempo de *watchdog*, o *F_WD_Time2*. O sinal *use_TO2_Flag* é uma *flag* auxiliar associada ao

uso do *F_WD_Time2*. Já o termo *Ack* representa o *acknowledgement safety PDU* do *F-Device*. Por fim, o termo *Message received* corresponde a um novo *safety PDU* recebido, que não tenha todos os seus valores iguais a "0".

Figura 3.9 - Diagrama de Estados do *F-Device*



Fonte: *PROFIsafe System Description* (2010, p. 53).

Quando o *F-Device* é inicializado, ele fica esperando por uma mensagem do *F-Host*. Quando o *F-Device* receber essa mensagem, se não ocorreu erro de *CRC*, o *F-Device* entra no seu ciclo normal de funcionamento (estados 23, 24 e 25), no qual ele permanece trocando mensagens com o *F-Host*. Nos estados 24, e 27 de "Espera por Mensagem", se ocorrer *timeout* no *F-Device* na espera por uma mensagem do *F-Host*, o *F-Device* entra nos estados 26, 27 e 28 para tratamento de falha.

Já nos estados 22, 25 e 28 de "Verificação de Mensagem", se ocorrer erro de *CRC*, o *F-Device* também realiza a transição para o estado 26 para tratamento de falha. A partir dessa situação de tratamento de falha, o *F-Device* só pode voltar ao seu ciclo normal de funcionamento a partir do estado 28, quando não houver erro de *CRC* e a variável *ok_nr_cycles* possuir um valor maior ou igual a 2 (*ok_nr_cycles>=2*).

4 TRABALHOS RELACIONADOS

Esta seção apresenta um conjunto de ferramentas de injeção de falhas desenvolvidas utilizando-se os diferentes tipos de abordagens de injeção de falhas. Estas ferramentas possuem diferentes propósitos, como, por exemplo, testar diferentes tipos de protocolos de comunicação, sistemas distribuídos, validar e avaliar sistemas e componentes de computadores ou interfaces de comunicação segura de dispositivos seguros através da aplicação de falhas. Estes injetores foram escolhidos, dentre outros pesquisados, porque eles apresentam uma ou mais características, limitações ou aspectos positivos, consideradas relevantes, as quais foram levadas em consideração para a especificação e o desenvolvimento do injetor de falhas apresentado neste trabalho.

4.1 sfiCAN

O *sfiCAN, star-based physical fault injector for CAN,* [Gessner 2011] é um injetor de falhas físicas para o protocolo *CAN*, que tem por objetivo testar o comportamento dos nodos de uma rede *CAN* na presença de erros de canal, em particular dos controladores de nodos *CAN* e do *software* executando neles.

O módulo de injeção de falhas foi implementado em um *hub,* juntamente com um módulo *CAN* e um módulo acoplador, utilizando-se a linguagem *VHDL*. O módulo acoplador tem por função implementar a função *wired-AND* através da realização do *AND* lógico dos sinais de *uplink* e *downlink*. Já o módulo *CAN* tem por objetivo observar o sinal de saída do módulo acoplador e indicar para o módulo de injeção de falhas um conjunto de sinais sinalizando qual *bit* dentro de qual campo de *bits* está sendo transmitido no momento.

O injetor de falhas permite criar e armazenar simultaneamente diferentes configurações de injeção de falhas, as quais são definidas através de um arquivo texto contendo a especificação do cenário de falhas. Essa especificação possui vários parâmetros de configuração, que indicam o tipo de falha (*stuck-at ou bit-flip*), quando e onde injetá-las.

Para realizar a injeção de falhas, a topologia de barramento *CAN* é substituída por uma topologia em estrela, cujo elemento central é o *hub* mencionado anteriormente. Ao *hub*, são ligados um computador, o controlador e os nodos *CAN*. O propósito de ligar o computador ao *hub* é para facilitar a configuração do módulo de injeção de falhas. O controlador e os nodos *CAN* são conectados ao *hub* por meio de *links* dedicados separados em *uplink* e *downlink*, aonde as falhas de comunicação são injetadas durante os testes.

O uso de uma topologia em estrela com um *hub* central, cujos *links* são separados em *uplink* e *downlink*, permite que o *sfiCAN* injete falhas com alta resolução espacial; ao passo que o módulo *CAN* dentro do *hub*, que mantém registro de qual *bit* dentro de qual campo de *bits* está atualmente sendo enviado, permite que a injeção de falhas seja realizada com grande resolução temporal. Além disso, o injetor causa um pequeno impacto no sistema, inserindo um pequeno atraso, que pode ser considerado como um atraso adicional de transmissão.

4.2 Fault Injection Framework for PROFIBUS

É um *framework* de injeção de falhas desenvolvido para avaliar a dependabilidade do protocolo *PROFIBUS* [Carvalho 2005]. O principal objetivo dos testes é analisar os efeitos de falhas no comportamento do *PROFIBUS*, isto é, na ativação dos mecanismos tolerantes a falhas e o seu impacto em parâmetros de desempenho.

A plataforma de *hardware* utilizada nos testes é composta por três elementos principais: nodos de comunicação *PROFIBUS-DP*, um módulo injetor de falhas e um módulo monitor. Os nodos de comunicação são parte da infraestrutura de comunicação enquanto que os módulos monitor e injetor são parte da infraestrutura de injeção de falhas.

O injetor utiliza uma abordagem física de injeção de falhas, na qual um nível elétrico é imposto em um local específico, emulando os efeitos de falhas. As falhas inseridas podem ser tanto assíncronas como síncronas. Já o módulo monitor possui a função de registrar todos os eventos relevantes no barramento, verificando a ativação e os efeitos das falhas.

Para satisfazer os requisitos de tempo real e a controlabilidade (tempo, localização e valor) do processo de injeção de falhas, o injetor foi desenvolvido com uma arquitetura de dois níveis. O nível inferior é composto por uma sonda que injeta as falhas físicas. Já o nível superior é composto por uma unidade de controle de injeção de falhas, responsável por coordenar os experimentos de injeção de falhas.

Nos testes, foram aplicadas falhas físicas de *bit-flips*, que correspondem à inversão do nível lógico e são utilizadas para corromper a informação transportada no sinal digital, emulando a ocorrência de falhas transientes. As falhas podem ser inseridas em dois níveis. No nível de barramento são injetadas falhas para corromper os quadros *PROFIBUS*, simulando a ocorrência de fatores externos como interferência eletromagnética. No nível físico são introduzidas falhas para emular um mau funcionamento do transceptor, na sua função de receber e transmitir mensagens.

Ao final de cada experimento, todos os dados do processo de injeção de falhas são enviados pela rede *Ethernet* para um computador onde são armazenados. Isso possibilita a análise dos resultados, permitindo o cálculo dos respectivos parâmetros de dependabilidade.

4.3 FIRMAMENT

FIRMAMENT, Fault Injection Relocatable Module for Advanced Manipulation and Evaluation of Network Transports, [Drebes 2006] [Siqueira 2009] é um injetor de falhas desenvolvido em software (*SWIFI*), que permite a especificação de cenários complexos de falhas de comunicação, com baixa intrusividade. Ele tem por objetivo permitir o teste de protocolos de comunicação, de aplicações de rede e de sistemas distribuídos, aonde o uso de implementações reais pode evidenciar deficiências na implementação que podem escapar à técnica de simulação.

O *FIRMAMENT* corresponde a um módulo integrado no *kernel* do *Linux* que permite realizar a manipulação de pacotes através da utilização de pontos específicos, ou ganchos do *Netfilter* (RUSSEL 2013). Por utilizar as interfaces de programação de alto nível do *Netfilter*, o injetor é independente da arquitetura da máquina.

A realização da injeção de falhas para validar protocolos de nível superior em ambientes de comunicação heterogênea é feita sobre os pacotes *IP* (protocolos *IPv4* e *IPv6*), o que permite emular as características de falhas dos protocolos inferiores. Isso ocorre porque o protocolo *IP*, devido a sua característica (serviço de *datagrama*, sem retransmissão) simplesmente reflete as limitações dos níveis anteriores, permitindo se aproximar do comportamento de falhas reais nos níveis de acesso à rede.

A configuração do cenário de falhas é feita através do uso de *faultlets*, que são aplicações que emulam o comportamento de falhas. Um *faultlet* é especificado através de um conjunto de 31 instruções na linguagem *FIRMASM* e executado sobre cada pacote que cruza os fluxos de entrada ou saída dos protocolos *IPV4* e *IPV6*, podendo inspecionar e modificar o conteúdo do pacote, ou ainda descartá-lo, duplicá-lo ou atrasá-lo.

Os *faultlets* são executados pela máquina virtual *FIRMVM*, responsável pela interpretação e processamento das instruções que especificam os cenários de falhas. A máquina virtual trabalha sobre quatro fluxos de pacotes, associados aos processamentos de entradas e saídas dos protocolos *IPV4* e *IPV6*.

4.4 F-Host Test Tool

A ferramenta *F-Host Test Tool* [Mühlhause 2007] não é uma ferramenta de injeção de falhas, ela foi desenvolvida para ser utilizada em testes de conformidade das interfaces de comunicação segura dos dispositivos mestres *PROFIsafe* (*F-Hosts*). Nesse tipo de teste, a ferramenta gera estímulos para o *F-Host* através do canal de comunicação e compara as reações recebidas com o resultado esperado de acordo com a norma.

Para verificar o dispositivo *F-Host* é necessário estabelecer uma conexão com ele. Para isso foi desenvolvido um *hardware* específico, o *PROFILgate*, que possui uma interface *PROFIBUS DP* e uma interface *PROFINET IO* para comunicação com o *F-Host*.

A arquitetura de teste baseia-se na norma *ISO* 9646 (ISO/IEC 1994) e, dessa forma, ela deve apresentar uma funcionalidade de teste superior e outra inferior. O testador superior é executado no dispositivo *F-Host*, diretamente acima da camada de comunicação *PROFIsafe*, que deve ser verificada. Assim, uma pequena parte do ambiente de testes precisa ser implementada no dispositivo a ser verificado. O testador inferior é implementado dentro da ferramenta de testes e utiliza a comunicação segura do canal *PROFIsafe*.

A Ferramenta *F-Host Test Tool* consiste de dois módulos principais: o *Testframe Client* e o *Testframe Server*. O cliente possui uma interface gráfica com o usuário, onde os projetos de teste podem ser configurados, administrados e os resultados avaliados. Já o servidor provém características específicas da comunicação, como um adaptador específico de teste *PROFIsafe*, que fornece acesso ao *hardware*, a execução dos casos de teste, registro dos estímulos e das suas reações e também a geração do veredito do teste.

4.5 Simmcast-FT

O *Simmcast-FT* [Barcelos 2005] é um *framework* de injeção de falhas por simulação que tem por objetivo avaliar a dependabilidade e o desempenho de sistemas distribuídos tolerantes a falhas. Ele também permite o teste com implementações parciais de sistemas (protótipos), possibilitando *feedbacks* importantes durante o processo de desenvolvimento.

O simulador emprega um modelo de eventos discretos baseado em processos, no qual componentes são combinados e estendidos para criar novos ambientes de simulação. Os componentes básicos do *framework* são: Nodo, *Thread*, Caminho, Mensagem ou Pacote, Grupo e Rede. As falhas, no entanto, são previstas apenas em Nodos, Caminhos e Grupos.

O modelo de falhas oferecido pela ferramenta é orientado a sistemas distribuídos e compreende os seguintes tipos de falhas: colapso, omissão, temporização, sintática e semântica. A falha de Colapso ocorre quando um componente para silenciosamente de funcionar. Já a de omissão é quando um componente omite resultados, de forma completa ou parcial. Por outro lado, a falha de temporização corresponde a um funcionamento com um tempo arbitrário. A falha sintática reflete um comportamento incorreto e detectável. Finalmente, a falha semântica compreende um comportamento correto com sentido incorreto.

Os comportamentos de falhas nos componentes são definidos pelo usuário e eles indicam quais os tipos de falhas que um componente pode sofrer durante o experimento de simulação e também quando as falhas estarão ativas ou inativas. Isso é feito através do uso de regras de ativação e de desativação, chamadas respectivamente de *on* e *off*. As falhas aplicadas durante a simulação podem ser efêmera, temporária ou permanente. No primeiro caso, a falha atua sobre o componente (ações) e depois é cancelada. Já no segundo caso, uma falha atua por um tempo não nulo, mas menor do que o tempo total da simulação. No terceiro caso, o componente não se recupera do tipo de falha.

A ferramenta apresenta como pontos positivos, a alcançabilidade, que permite que um conjunto extenso de falhas seja injetado de forma a atingir todos os componentes da rede. Outro ponto importante é a alta controlabilidade, pois é possível especificar precisamente os tipos de falhas e quando elas devem ocorrer. A ferramenta também possui uma baixa intrusividade, pelo fato do processo de injeção de falhas por simulação não apresentar sobrecarga temporal e nem desvio da lógica de execução normal do sistema alvo (sem alteração do sistema alvo). Além disso, é possível medir a latência de erros e o tempo de recuperação dos sistemas distribuídos.

4.6 Xception

Xception [Maia 2005] é um injetor de falhas que permite realizar tanto verificações quanto validações precisas e flexíveis e avaliações de sistemas e componentes de computadores, com particular ênfase em componentes de *software*. O injetor corresponde a um *framework* abrangente para testes experimentais e avaliação de mecanismos e recursos de dependabilidade. Isso é possível pela inclusão de vários métodos de injeção de falhas para vários sistemas alvos e um conjunto de módulos para ajudar a execução de experimentos de teste e análise dos resultados.

A ferramenta possui uma metodologia híbrida de injeção de falhas ao invés de utilizar apenas a metodologia *SWIFI.* A ideia é utilizar os recursos avançados de *debug* e de monitoramento de desempenho existentes nos processadores modernos, para injetar falhas mais realistas por *software* e monitorar a ativação dessas falhas e o seu impacto no comportamento do sistema alvo. Para fazer isso, o injetor utiliza os registradores de *breakpoint* dos processadores para definir os gatilhos de falhas, inclusive aqueles relacionados à manipulação de dados. A injeção de falhas corresponde normalmente à execução de uma pequena rotina de exceção.

Depois que uma falha é injetada, o *hardware* de monitoramento de desempenho dentro do processador é usado para obter informação detalhada do comportamento do sistema. Assim é possível saber se uma célula de memória foi acessada depois da falha ou se alguma rotina de recuperação de erro foi executada, por exemplo. Além disso, como o *Xception* opera muito próximo do *hardware*, é possível injetar falhas em qualquer processo rodando no sistema alvo, inclusive no *kernel.*

Essa abordagem híbrida do *framework* permite emular falhas físicas transientes em processadores, memória principal e outros dispositivos que podem ser alcançados por *software*. Os gatilhos de falha baseados nos registradores de *breakpoint* permitem a injeção de falhas em praticamente todas as circunstâncias de execução de código, manipulação de dados ou tempo. Os tipos de falhas injetadas correspondem à manipulação de *bits* (modelo de falhas por *hardware*).

O injetor apresenta uma interface gráfica chamada de *EME* (*Experiment Manager Environment),* a qual fornece acesso a todas as ferramentas de injeção de falhas e oferece uma visão consistente para o usuário. A *EME* é responsável pela definição das falhas, execução e controle dos experimentos, coleta e análise dos resultados. Ela ainda possui os recursos *EFD*, para facilitar a definição de falhas, e o *Xtract*, para avaliar detalhadamente os resultados, incluindo um rastreamento detalhado dos efeitos das falhas.

A arquitetura de injeção de falhas do *Xception* é parecida com o modelo servidor-cliente. De um lado, um computador com a ferramenta faz o controle/gerenciamento do experimento e do outro, um sistema alvo, aonde um núcleo de injeção de falhas e elementos de monitoramento são executados e são responsáveis por inserir e monitorar as falhas. A conexão entre eles é feita utilizando os protocolos *TCP-IP*. O fato de o sistema alvo ter sido modificado para a inclusão do núcleo de injeção de falhas e dos elementos de monitoramento, causa certa intrusividade, ainda que baixa.

4.7 Análise dos Injetores de Falhas Pesquisados

Nenhuma das ferramentas de injeção de falhas apresentadas anteriormente pode ser utilizada para validar de forma segura a implementação dos mecanismos de tolerância a falhas do protocolo *PROFIsafe*. Isso ocorre porque essas ferramentas não possuem recursos suficientes para simular todos os tipos de falhas descritos na norma *IEC 61508* e, dessa maneira, avaliar todos os mecanismos de detecção de falhas do referido protocolo.

Os injetores de falhas físicas *sfiCAN* e *Framework* para o *PROFIBUS* simulam falhas transientes de *hardware* através da modificação do valor de um *bit* ou de uma sequência de *bits* nos pacotes transmitidos. Esses dois tipos de falhas podem causar apenas erros de *CRC* e de sequência incorreta nas mensagens, simulando apenas dois dos tipos de falhas que o protocolo se propõe a detectar. Já o injetor de falhas *Firmament* embora apresente mais recursos para injetar falhas nas mensagens, não consegue simular o modelo completo de falhas especificado na norma *IEC 61508*. Com ele seria possível injetar falhas de descarte, duplicação, atraso, sequência incorreta e corrupção de mensagens, mas não erros de endereçamento, mascaramento e inserção de mensagens na comunicação.

A ferramenta *F-Host Test Tool* permite testar apenas o dispositivo mestre *PROFIsafe* (*F-Host*) através do envio de estímulos e comparação das respostas recebidas com resultados esperados e já conhecidos anteriormente. Ele não atua nas mensagens de comunicação e, por essa razão, não pode ser usado para verificar o funcionamento dos mecanismos de tolerância a falhas do protocolo. Do mesmo modo, o injetor de falhas por simulação *Simmcast-FT* também não prevê a injeção de falhas em mensagens individuais porque elas são de caráter temporário e não fazem parte da configuração de um experimento de simulação para sistemas distribuídos. Assim, da mesma forma que a ferramenta *F-Host Test Tool*, o *Simmcast-FT* também não permite injetar falhas nas mensagens do *PROFIsafe.*

Finalmente, a ferramenta *Xception* avalia a dependabilidade de mecanismos e recursos físicos de sistemas de computadores e de componentes de *software*. As falhas inseridas por *software* emulam falhas transientes de *hardware* através da inversão de valores de *bits*. Novamente, este tipo de falha causaria apenas erros de *CRC* e de sequência incorreta nas mensagens do protocolo *PROFIsafe*.

Embora essas ferramentas de teste não permitam validar todos ou mesmo nenhum dos mecanismos de tolerância a falhas do protocolo, o seu estudo possibilitou conhecer algumas características interessantes empregadas nos seus projetos. Essas características, encontradas

em uma ou mais ferramentas, foram consideradas importantes e utilizadas para definir a arquitetura do injetor de falhas desenvolvido neste trabalho.

Como exemplo dessas características, tem-se a utilização de uma abordagem externa para a realização dos testes de injeção de falhas de forma a minimizar a questão da intrusividade espacial e temporal, causando, dessa forma, um pequeno impacto nos sistemas sob teste. Isso é importante, pois quanto menor for a intrusividade causada pela ferramenta injetora, menor será a sua interferência sobre os resultados dos testes realizados.

Outro ponto importante é a controlabilidade do teste que alguns dos injetores oferecem, a qual permite saber o tipo, quando e onde uma falha será injetada. Essa característica é essencial, pois permitirá saber exatamente qual pacote sofreu a injeção de falhas, o tipo de falha aplicada e o momento em que isso aconteceu. Essa informação é valiosa para acompanhar a ativação dos mecanismos de tolerância a falhas relacionados ao tipo de falha que foi inserida.

Além disso, algumas ferramentas armazenam os dados do processo de injeção de falhas para posterior análise. O armazenamento de um relatório é indispensável para um engenheiro de testes, pois o relatório permite que o engenheiro analise os resultados do processo de injeção de falhas, verificando o comportamento do protocolo de acordo com as falhas injetadas. Possuir um histórico com os resultados dos testes realizados durante o desenvolvimento de um protocolo de comunicação ou de dispositivos que façam uso de um protocolo de comunicação é útil para se controlar as alterações feitas e também realizar comparações e análises no decorrer do processo de desenvolvimento.

Por fim, a última característica relevante encontrada em alguns dos injetores é a utilização de uma interface gráfica para configuração do cenário de falhas, a visualização e análise dos resultados obtidos. O uso de uma interface é importante, pois torna mais fácil e amigável a interação dos usuários com a ferramenta. Por exemplo, com o uso de uma interface gráfica, podem-se oferecer diversas opções selecionáveis para se configurar um teste de injeção de falhas, requerendo poucas entradas de dados do usuário, o que reduz bastante a possibilidade de inserção de erros durante a configuração dos testes.

5 INJETOR DE FALHAS

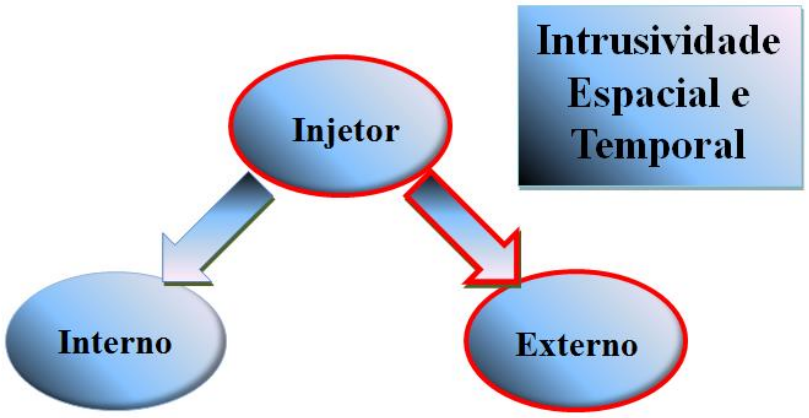
Esta seção apresenta o processo de desenvolvimento do injetor de falhas de comunicação *FITT (Fault Injection Test Tool)*. Primeiramente foi definida a abordagem de injeção de falhas feita a partir de uma análise das ferramentas injetoras pesquisadas, juntamente com um estudo sobre a questão da intrusividade espacial e temporal. Em seguida é mostrada a especificação da arquitetura do injetor de falhas *FITT*. Depois, é apresentado o funcionamento interno do injetor de falhas, juntamente com a descrição das funções de injeção de falhas e a utilização da sua interface gráfica para a configuração de testes.

5.1 Definição da Abordagem de Injeção de Falhas

A definição da abordagem de injeção de falhas do injetor desenvolvido nesse trabalho levou em consideração a questão da intrusividade espacial e temporal. Já a forma de implementação da ferramenta considerou como foram desenvolvidos os injetores pesquisados.

A figura 5.1 mostra as duas opções de abordagens consideradas para a implementação de um injetor de falhas levando-se em conta a questão da intrusividade espacial e temporal. De acordo com ela, a abordagem externa é a mais indicada para injetar falhas, pois ela evita alterações e execução de código adicional no sistema alvo.

Figura 5.1 - Abordagem de Injeção de Falhas de acordo com a Intrusividade



Fonte: Rodrigo Dobler.

Os injetores de falhas para os protocolos de comunicação *CAN* e *PROFIBUS*, apresentam uma abordagem externa para minimizar a intrusividade espacial, e utilizam *hardware* específico para satisfazer os requisitos de tempo real dos respectivos protocolos.

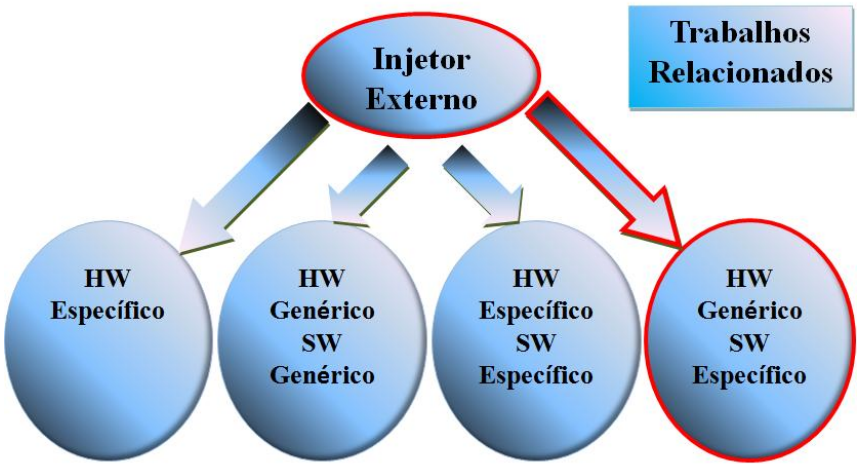
O injetor de falhas *Firmament* pode ser configurado internamente ou externamente ao sistema de teste em um computador com *Linux* (*hardware* genérico), podendo gerar uma pequena intrusividade temporal. Ele é um *software* genérico para injeção de falhas, podendo testar protocolos de comunicação, sistemas distribuídos e aplicações de rede.

Já a *F-Host Test Tool* é uma ferramenta com *software* e *hardware* específicos para permitir a comunicação com o dispositivo a ser testado. Este dispositivo também precisa ser modificado para a implementação de uma pequena parte do ambiente de testes nele, o que causa certa intrusividade espacial e temporal, mesmo que pequena.

Por outro lado, o *Simmcast-FT* é um injetor por simulação para o teste de sistemas distribuídos. Ele utiliza *software* específico e *hardware* genérico, pois pode ser utilizado em qualquer computador. Finalmente, a ferramenta *Xception* utiliza *software* específico com uma abordagem híbrida de injeção de falhas (*hardware e software*) para avaliar componentes e sistemas de computadores (*hardware* genérico), principalmente componentes de *software*.

A figura 5.2 a seguir mostra as diferentes possibilidades de implementação encontradas nos injetores de falhas dos trabalhos relacionados. Elas foram agrupadas para analisar a opção mais adequada ao propósito da ferramenta desenvolvida nesse trabalho.

Figura 5.2 - Abordagens de Injeção de Falhas dos Injetores Pesquisados



Fonte: Rodrigo Dobler.

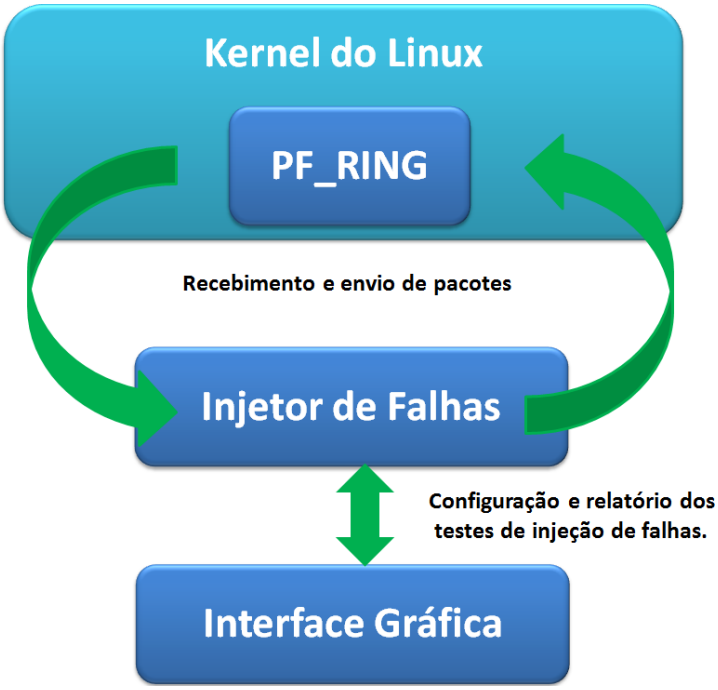
A partir da análise feita, optou-se por uma abordagem externa com *hardware* genérico e *software* específico (o injetor). A técnica de *SWIFI* foi escolhida devido ao menor custo, maior facilidade de desenvolvimento [Carreira 1998] e por esta técnica ser mais adequada para se trabalhar com estruturas de dados de mais alto nível do sistema [Drebes 2006]. Ela também é a mais adequada para avaliar mecanismos de tolerância a falhas implementados em *software* [CLARK et al 1995, ARLAT et al 2003]. Além disso, ela permite que o injetor de

falhas evolua futuramente através da inserção de novas classes de falhas para testar outros protocolos [Carreira 1998].

5.2 Arquitetura do Injetor de Falhas

O *FITT* foi desenvolvido utilizando-se a linguagem de programação *C*, para ser utilizado em um computador de *hardware* genérico e com sistema operacional *Linux* a partir da versão *2.6.32* do *kernel*. O injetor é composto por três partes: o *PF_RING* [PF_RING 2015], a interface gráfica e a aplicação do injetor de falhas com as funções de falha, mostrados na figura 5.3. Além disso, o injetor faz uso de duas placas de rede (interfaces *DNA1* e *DNA2*) compatíveis com o *PF_RING*.

Figura 5.3 - Componentes do Injetor de Falhas



Fonte: Rodrigo Dobler.

5.2.1 PF_RING

O *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*, o qual pode operar em dois modos diferentes: o *PF_RING Vanilla* e o *PF_RING DNA* (*Direct NIC Access*).

No modo *Vanilla*, o *PF_RING* captura os pacotes dos adaptadores de rede por meio do

NAPI do *Linux*. O *NAPI* (*New API*) copia os pacotes da placa de rede para o *buffer* circular do *PF_RING* e então a aplicação no espaço do usuário lê os pacotes desse anel. Neste cenário, existem dois captadores, a aplicação e o *NAPI*, o que consome processamento da *CPU* para captar os pacotes. Nesse modo, o *PF_RING* pode apenas receber e não transmitir pacotes.

Já o modo *DNA* é uma forma de mapear a memória e os registradores da placa de rede diretamente para o espaço do usuário. Assim, os pacotes são lidos/enviados diretamente da/para a interface de rede por uma aplicação sem utilizar o mecanismo de captura *NAPI* do *Linux*. Isso reduz a intrusividade, pois o processamento da *CPU* é usado apenas para consumir pacotes e não para movê-los para fora ou para dentro do adaptador de rede. No *DNA* ambas as operações de recebimento e de envio de pacotes nas interfaces de rede são suportadas.

Para permitir a transmissão de pacotes diretamente entre o adaptador de rede e a aplicação executando no espaço do usuário (em cópia zero), sobre o *DNA*, foi criada a biblioteca *libzero*. Ela permite a distribuição de pacotes através de *threads* e processos e o encaminhamento eficiente de pacotes através de interfaces de redes pela utilização de *drivers* específicos para certos tipos de placas de rede, os quais enviam/recebem os pacotes diretamente para/de uma aplicação no espaço do usuário. A *libzero* possui dois componentes que operam sobre os pacotes no modo de cópia-zero: o *DNA Cluster* e o *DNA Bounce*.

O *DNA Cluster* permite que várias aplicações estejam conectadas a ele e que elas recebam e enviem os pacotes em “cópia zero”. Já o *DNA Bounce* permite que uma aplicação envie e receba pacotes entre duas interfaces em “cópia-zero”, com uma latência baixa de 3.45 microssegundos. Ele é mais fácil de usar e foi projetado para uma troca de pacotes em alta velocidade entre as interfaces e uma aplicação.

Neste trabalho foi utilizado o *PF_RING* modo de operação *DNA* com o componente *DNA Bounce*. Essa escolha foi feita porque o modo *DNA* suporta o recebimento e envio de pacotes entre a ferramenta injetora e as interfaces de rede e porque o componente *DNA Bounce* é mais rápido e mais simples de usar do que o *DNA Cluster*.

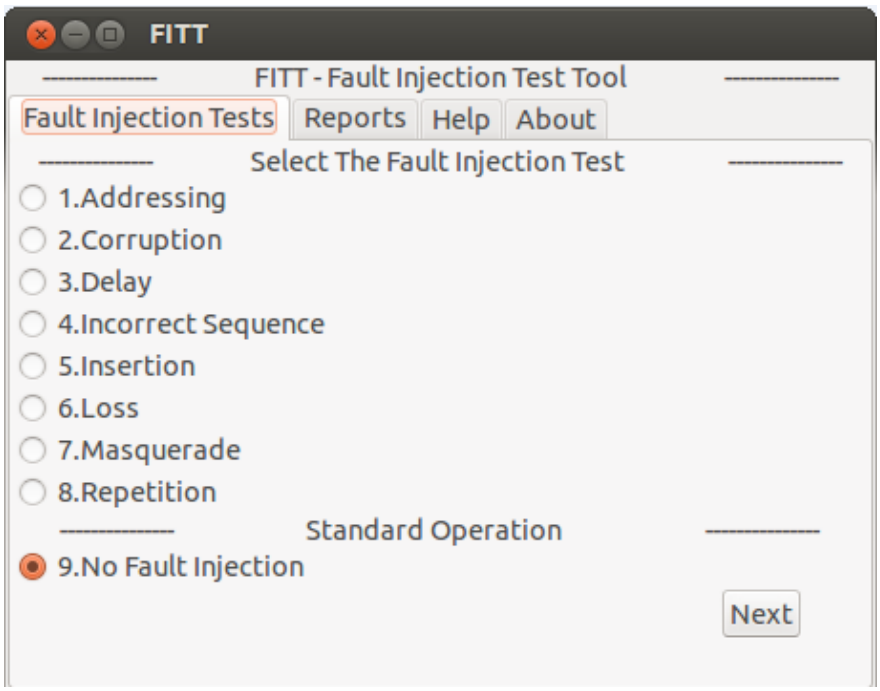
5.2.2 Interface Gráfica

A interface gráfica foi desenvolvida com a ferramenta *Glade* [Glade 2015] e foi projetada para ser amigável, tornando mais fácil e agradável a experiência ao se usar o *FITT*. Ela contém as opções de utilização da ferramenta pra o usuário e é apresentada na figura 5.4.

A primeira aba da tela principal apresenta um *menu* para a escolha da função de injeção de falhas desejada a ser utilizada nos testes. A segunda aba oferece a possibilidade de

visualização do relatório dos testes. Já a terceira aba apresenta uma ajuda para o usuário, permitindo ao mesmo visualizar as instruções de uso da ferramenta. Finalmente, a última aba apenas mostra informações sobre o desenvolvedor da aplicação.

Figura 5.4 - Interface da janela principal do Injetor de Falhas *FITT*



Fonte: Janela Principal da Interface gráfica do Injetor de Falhas *FITT*.

5.2.3 Injetor de Falhas

O injetor de falhas é uma aplicação que contém as funções e procedimentos necessários para injetar falhas nos pacotes transmitidos na rede. A ferramenta recebe os pacotes de dados do *PF_RING* e aplica a injeção de falhas de acordo com o cenário de falhas especificado pelo usuário através da interface gráfica. O Injetor utiliza as suas funções de injeção de falhas para simular os diversos tipos de falhas que podem ocorrer durante a transmissão dos pacotes em um ambiente industrial, em conformidade com a norma *IEC 61508* e com a especificação do *PROFIsafe*.

Depois que o processo de injeção de falhas foi realizado, o usuário pode visualizar um relatório com informações deste processo. A visualização pode ser feita tanto na interface gráfica como diretamente no arquivo texto gerado. Nesse relatório são apresentados o número de pacotes recebidos e enviados em cada interface de rede, o conteúdo de cada pacote recebido e enviado, o tipo de teste escolhido e o número do pacote em que as falhas estão sendo injetadas. Além disso, é sinalizado quando a injeção de falhas irá ocorrer, o tempo de

intrusividade causado pelo teste aplicado e é mostrado o conteúdo do pacote antes e depois da falha ter sido injetada. Estas informações são úteis para realizar análises e comparações de resultados sobre o comportamento do protocolo na presença de falhas.

5.3 Funcionamento do Injetor de Falhas

O *FITT* trabalha com pacotes *UDP/IP* sobre *Ethernet*, tipo 0x0800, para a transmissão dos pacotes da implementação do protocolo *PROFIsafe* utilizada neste trabalho. Essa identificação do tipo de pacote é utilizada junto com o número da porta, 5100, usada na comunicação entre os dispositivos *F-Host* e *F-Device* para criar um filtro de pacotes. Com este filtro, apenas os pacotes *PROFIsafe* são enviados ao injetor de falhas, sendo que todos os outros tipos de pacotes são automaticamente enviados de uma interface de rede à outra.

Os *sockets* criados para a comunicação entre os dispositivos *F-Host* e *F-Device*, fazem uma verificação de *CRC* nos pacotes *UDP* transmitidos e recebidos. Por esse motivo, o injetor precisa recalcular o *checksum* do pacote *UDP*, inserindo o novo valor no lugar do antigo, sempre que ele injetar uma falha que modifique o conteúdo da mensagem *PROFIsafe* sendo transmitida dentro do pacote *UDP*.

Já na sua estrutura principal, o injetor de falhas *FITT* utiliza *threads* para controlar os diferentes mecanismos necessários ao seu funcionamento. Dependendo da configuração do cenário de falhas, o injetor poderá ter três ou quatro *threads* em execução simultânea. A primeira e a segunda *threads* controlam o envio/recebimento de pacotes, de forma independente, nas interfaces de rede *DNA* do *PF_RING*. Essas duas *threads* poderão estar associadas a qualquer uma das interfaces *DNA* de acordo com a configuração feita pelo usuário. O utilizador também define como as aplicações *F-Host* e *F-Device* vão estar ligadas ao injetor de falhas, podendo estar conectadas através de qualquer uma das interfaces de rede *DNA*. Uma descrição geral dessa ligação é apresentada na figura 5.5. A injeção de falhas pode ser aplicada em ambos os sentidos da comunicação entre o *F-Host* e o *F-Device*, mas em apenas um sentido por vez.

A terceira *thread* é usada para contar o tempo de injeção de falhas definido pelo usuário (chamado de tempo 1). Após decorrer esse tempo, esta *thread* sinaliza ao injetor de falhas que o próximo pacote que chegar é o escolhido para passar pelo processo de injeção de falhas. O ciclo de contagem de tempo reinicia e continua até que o programa seja encerrado.

O uso de uma quarta *thread* é requerido apenas para as funções de falhas de atraso,

repetição e inserção de mensagens. Para estas funções, a *thread* número 4 controla o tempo que deve decorrer (conhecido como tempo 2) antes que o pacote de dados resultante do processo de injeção de falhas seja enviado por ela no canal de comunicação.

Figura 5.5 - Ligação entre o *F-Host*, Injetor de Falhas e o *F-Device*

Fonte: Rodrigo Dobler.

Durante o processo de injeção de falhas, as informações referentes ao processo vão sendo armazenadas pelo injetor de falhas em um relatório, o qual é salvo no formato de arquivo texto. A configuração do cenário de injeção de falhas é feita através da interface gráfica apresentada anteriormente. As opções escolhidas pelo usuário e os dados informados por ele são passados por parâmetros ao código da aplicação injetora de falhas. Nessa configuração atual dos testes é permitido escolher apenas um tipo de falha, de forma a verificar individualmente os mecanismos de tolerância a falhas do *PROFIsafe*.

5.4 Funções de Injeção de Falhas

Para avaliar de forma eficiente o comportamento do protocolo *PROFIsafe* em relação à ocorrência de falhas, foram desenvolvidas funções de falhas conforme o modelo de falhas da norma *IEC 61508*. As funções de falhas criadas foram as de erro de endereçamento, corrupção, atraso, sequência incorreta, inserção, perda e repetição de mensagens.

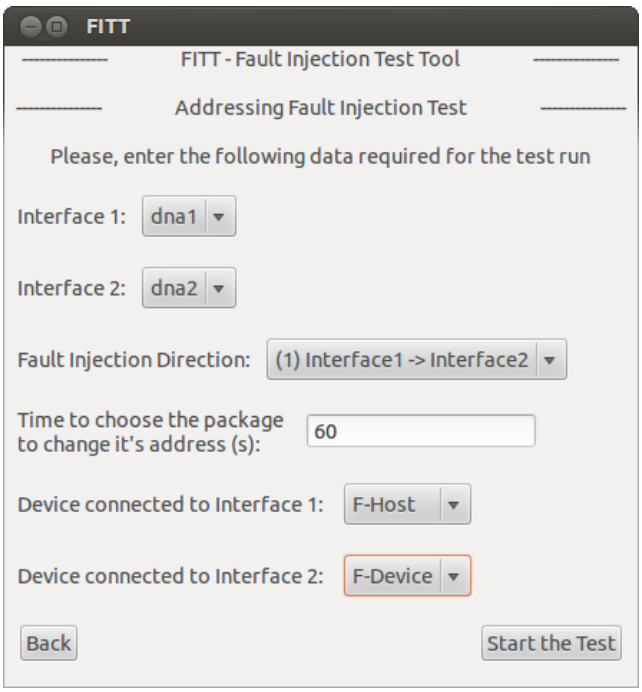
5.4.1 Função de Erro de Endereçamento

A função de injeção de falhas por erro de endereçamento tem por objetivo enviar uma mensagem segura para o destinatário errado no canal de comunicação, simulando um dispositivo *PROFIsafe* defeituoso operando em uma rede industrial. A interface gráfica dessa função é apresentada na figura 5.6.

Para configurar o teste, o utilizador precisa informar as duas interfaces de rede *DNA* correspondentes às interfaces de rede 1 e 2 do injetor. A seguir, é preciso definir o sentido da

injeção de falhas e o tempo de escolha do pacote que terá o seu endereço de destino alterado (tempo 1). Para concluir, é necessário informar qual dos dispositivos *F-Host* e *F-Device* está ligado em cada uma dessas duas interfaces. Ao se iniciar a transmissão de pacotes entre os dispositivos *F-Host* e *F-Device*, um contador de tempo na *thread* 3 é iniciado, sendo incrementado até atingir o valor em segundos especificado no tempo 1. Nesse momento, o injetor sinaliza que o próximo pacote que for recebido, no sentido especificado para a injeção de falha, terá o seu endereço alterado.

Figura 5.6 - Interface da Função de Erro de Endereçamento



Fonte: Janela de Configuração do Teste de Erro de Endereçamento.

No *PROFIsafe*, para alterar o endereço de destino de uma mensagem, é preciso modificar o *F-Parameter codename*. A implementação do protocolo *PROFIsafe* utilizada neste trabalho foi simplificada, definindo-se diretamente um valor para o *CRC1* e não calculado ele com os *F-Parameters*. Como o *codename* é único no relacionamento *F-Host* e *F-Device*, se ele for alterado, vai resultar em um *CRC1* diferente. Dessa forma, a mudança do endereço de destino do pacote escolhido é feita através da geração de um novo valor randômico para o *CRC1* e diferente do valor original. Esse novo valor do *CRC1* é utilizado juntamente com o C*ontrol/Status Byte*, mais os dados do pacote escolhido e o mesmo número de sequência, para gerar um novo valor de *CRC2*, o qual é inserido no pacote escolhido.

Como o conteúdo da mensagem *PROFIsafe* foi alterado, é preciso recalcular o *UDP Checksum* para que o pacote não seja descartado ao chegar no dispositivo de destino. Isso

acontece porque os dispositivos *F-Host* e *F-Device* calculam o *UDP Checksum* ao enviarem uma mensagem/*ack* e recalculam-no ao receberem essa mesma mensagem/*ack* para verificar a consistência dos dados emitidos e recebidos. Para que o descarte não aconteça, esta função faz um novo cálculo do *UDP Checksum* da mensagem alterada, e insere o novo valor no pacote escolhido, antes de ele ser enviado para o seu dispositivo de destino com o endereço trocado.

Quando os próximos pacotes chegarem, eles serão enviados normalmente ao seu destino, até que o contador de tempo atinja de novo o tempo especificado e o ciclo de injeção de falhas se repita na direção especificada.

5.4.2 Função de Corrupção de Dados

Na função de corrupção de mensagens é oferecida a possibilidade de alterar um *bit* ou todos os *bits* do *byte* escolhido pelo usuário. A alteração é feita através da inversão do valor do(s) *bit*(s) de um dos 16 *bytes* que compõe o *Safety PDU*. Desta forma, poderá ser alterado qualquer um dos *bytes* que o compõe (campos de *CRC*2, dados e *Control/Status Byte*). Isso garante que o conteúdo da mensagem enviada será alterado, simulando a ocorrência de um erro no barramento de comunicação ou um mau funcionamento em um dispositivo durante a transmissão de dados. A janela de configuração desta função é mostrada na figura 5.7.

Figura 5.7 - Interface da Função de Corrupção de Dados

Fonte: Janela de Configuração do Teste de Corrupção de Dados.

Este é o teste mais complexo dentre todos os desenvolvidos e o que requer a maior quantidade de informações fornecidas pelo operador. Neste teste é preciso selecionar as duas interfaces de rede *DNA* correspondentes às interfaces de rede 1 e 2 do injetor. Depois, é necessário informar o sentido da injeção de falhas e o tempo de escolha do pacote que terá os seus dados corrompidos (tempo 1). A seguir, é preciso informar o tipo do teste (corrupção de um *byte* ou de apenas um *bit*), o número do *byte* do *safety PDU* que vai ser corrompido e a posição do *bit* dentro do *byte* (esse dado só é utilizado no teste de corrupção de 1 *bit*). Para concluir, é necessário especificar qual dos dispositivos *F-Host* e *F-Device* está ligado em cada uma dessas duas interfaces.

Quando a transmissão de pacotes entre o *F-Host* e o *F-Device* começa, inicia-se o contador de tempo na *thread* 3, sendo o mesmo incrementado até atingir o valor contido no tempo 1. Nesse momento, o injetor sinaliza que o próximo pacote que for recebido, no sentido especificado para a injeção de falha, terá os seus dados corrompidos. Se o tipo de teste selecionado for o de corrupção de *byte*, o *byte* escolhido do *safety PDU* terá todos os seus *bits* invertidos. Por outro lado, se o teste especificado for o de corrupção de *bit*, o *bit* escolhido do *byte* selecionado do *safety PDU* será invertido. O *UDP Checksum* é recalculado e inserido na mensagem escolhida pelo mesmo motivo explicado na função de erro de endereçamento e depois o pacote é enviado para o seu destino.

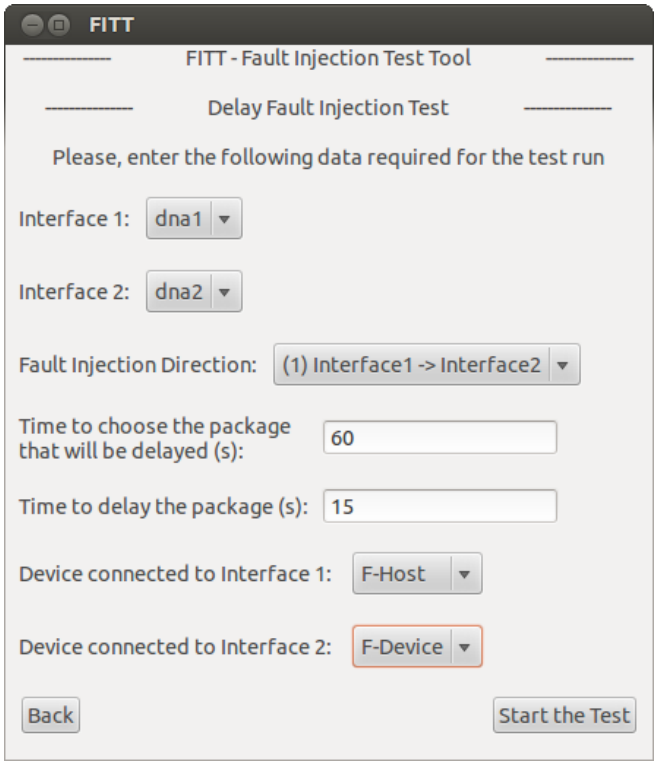
Da mesma forma, quando os próximos pacotes chegarem, eles serão encaminhados normalmente ao seu destino, até que o contador na *thread* 3 atinja o tempo determinado e o ciclo de injeção de falhas recomece na direção especificada.

5.4.3 Função de Atraso de Pacotes

Já a função de atraso tem por objetivo atrasar as mensagens selecionadas. O teste reproduz a situação em que a troca de dados excede a capacidade do *link* de comunicação ou quando um dispositivo causa uma sobrecarga ao simular mensagens seguras incorretas, fazendo com que um serviço que esteja ligado às mensagens seja atrasado ou prevenido. A janela de configuração do teste é mostrada na figura 5.8.

O usuário configura o teste informando qual das interfaces *DNA* corresponde a qual das interfaces de rede 1 e 2 utilizadas pelo injetor. Depois, ele deve especificar o sentido da injeção de falhas, o tempo de escolha do pacote que será atrasado (tempo 1) e o tempo de espera para atrasar o pacote selecionado (tempo 2). Por fim, também é preciso definir qual dos dispositivos *F-Host* e *F-Device* está conectado em cada uma dessas duas interfaces de rede.

Figura 5.8 - Interface da Função de Atraso de Pacotes



Fonte: Janela de Configuração do Teste de Atraso de Pacotes.

No instante em que a troca de mensagens entre os dispositivos mestre e escravo começa, o injetor inicia o contador de tempo da *thread* 3, o qual é atualizado até atingir o valor em segundos especificado no tempo 1. Nesse momento, o injetor estabelece que o próximo pacote que for recebido no sentido especificado para a injeção de falhas, será atrasado. O pacote escolhido é então copiado e fica guardado em um *buffer*, sob controle de outra função associada a *thread* 4 criada para este fim.

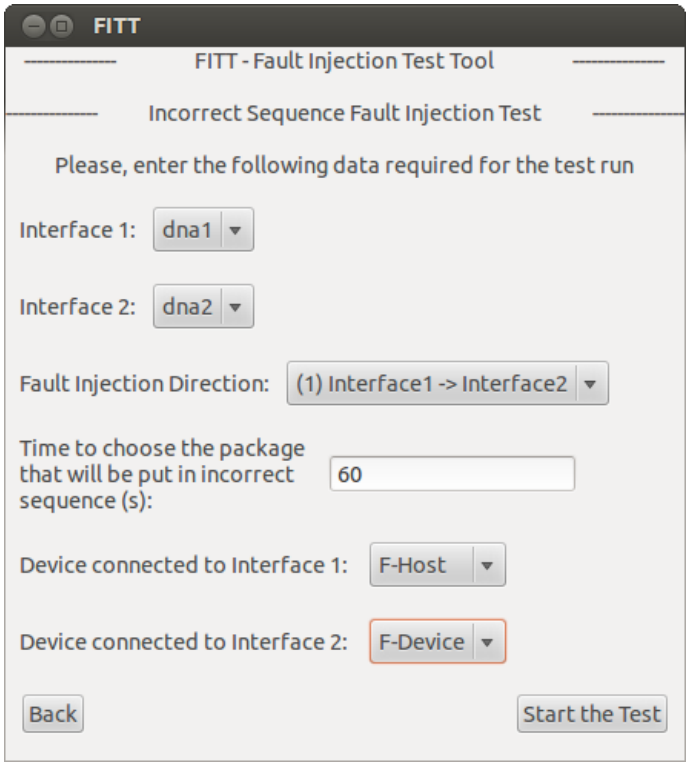
O pacote fica em espera até que o tempo para enviá-lo (tempo 2) tenha sido alcançado no contador de tempo da *thread* 4. Assim, que o tempo 2 foi atingido, o pacote atrasado é enviado ao seu destino. Após a injeção de falhas, a troca de mensagens segue normalmente, até que o contador de tempo na *thread* 3 atinja novamente o tempo definido e o ciclo de injeção de falhas seja reiniciado.

5.4.4 Função de Sequência Incorreta

Na sequência incorreta, a ordem das mensagens é trocada durante o seu envio, através da troca do número de sequência da mensagem escolhida antes de ela ser enviada ao seu destino. Esta função simula o comportamento de um dispositivo que, por não estar

funcionando corretamente, troca a ordem de envio das mensagens seguras. A figura 5.9 mostra a interface gráfica do teste de sequência incorreta.

Figura 5.9 - Interface da Função de Sequência Incorreta



Fonte: Janela de Configuração do Teste de Sequência Incorreta.

O usuário deve configurar o teste informando qual das interfaces *DNA* representa qual das interfaces de rede 1 e 2 usadas pelo injetor. A seguir, ele precisa definir o sentido da injeção de falhas e o tempo de escolha do pacote que será colocado em sequência incorreta (tempo 1). Finalmente é necessário informar qual dos dispositivos *F-Host* e *F-Device* está associado a cada uma dessas duas interfaces.

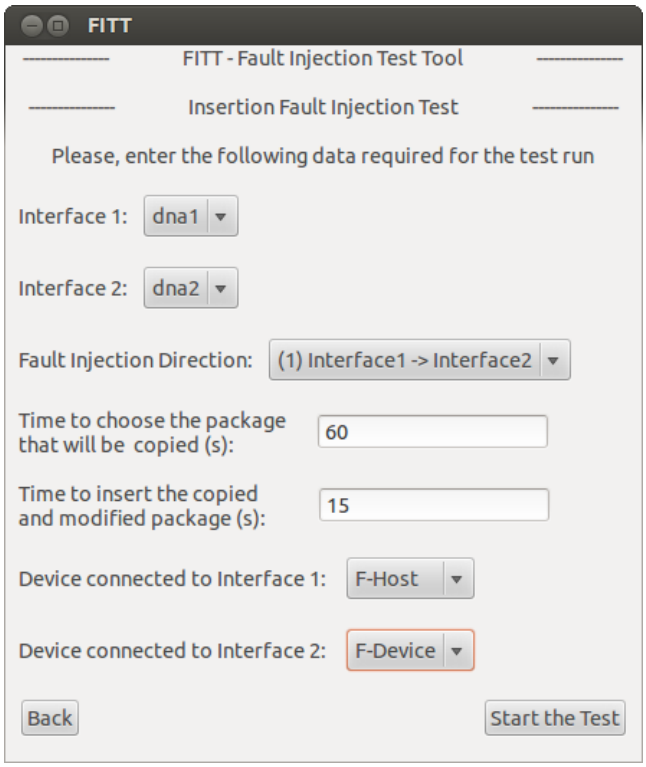
No momento em que a troca de pacotes entre os dispositivos *PROFIsafe* começa, é iniciado o contador de tempo da *thread* 3, o qual é incrementado até igualar o valor especificado no tempo 1. Nesse instante, o injetor indica que o próximo pacote recebido, no sentido especificado para a injeção de falhas, terá a sua ordem alterada. Como a versão do protocolo *PROFIsafe* utilizada nestes testes é a V2, o número de sequência não é transmitido nos pacotes, ele fica nos dispositivos e por isso ele é chamado de virtual. O que é transmitido em cada mensagem é o *Toggle Bit* dentro do *Status/Control Byte*, o qual indica um incremento dos contadores locais. Sempre que o *Toggle Bit* inverter de valor (de 0 para 1 ou de 1 para 0) as mensagens estarão sendo recebidas na ordem certa, e os contadores no *F-Host* e no *F-Device* são incrementados.

Assim, para trocar a ordem da mensagem selecionada, esta função de injeção de falhas inverte o valor do *Toggle Bit* contido nela e depois envia essa mensagem para o seu destino. Quando essa mensagem for recebida, o dispositivo verificará que o valor do *Toggle Bit* que ele tem armazenado é o mesmo do *Toggle Bit* recebido na mensagem. Por esse motivo, o contador do número consecutivo do dispositivo não é incrementado, a mensagem é considerada como estando fora de ordem e não é aceita. Depois da injeção de falhas os próximos pacotes que chegarem serão enviados normalmente ao seu destino, até que o tempo 1 especificado tenha decorrido e o ciclo de injeção de falhas seja reiniciado.

5.4.5 Função de Inserção de Mensagens

A inserção de mensagens tem por objetivo inserir uma mensagem segura que não pertence à comunicação entre o *F-Host* e o *F-Device*. O teste simula as situações em que um dispositivo com defeito insere mensagens seguras no barramento de comunicação. A janela de configuração desta função é apresentada na figura 5.10.

Figura 5.10 - Interface da Função de Inserção de Mensagens



Fonte: Janela de Configuração do Teste de Inserção de Mensagens.

O teste é configurado através da informação sobre qual das interfaces *DNA* está associada a qual das interfaces de rede 1 e 2 utilizadas pela ferramenta injetora de falhas.

Depois, é preciso informar a direção da injeção de falhas, o tempo de escolha do pacote que será modificado e inserido na comunicação (tempo 1) e o tempo de espera para enviar o pacote a ser inserido (tempo 2). Por fim, também é necessário informar qual dos dispositivos mestre e escravo está conectado em cada uma dessas duas interfaces de rede do injetor.

No momento em que começa a transmissão de mensagens entre os dispositivos *PROFIsafe*, o injetor inicializa o contador de tempo na *thread* 3, que é atualizado até atingir o valor em segundos especificado no contador do tempo 1. Nesse ponto, a ferramenta indica que o próximo pacote recebido no sentido especificado para injeção de falha, será copiado. O pacote escolhido é então copiado e em seguida enviado ao seu destino. Já o pacote que foi copiado e está guardado em um *buffer* vai ser modificado por outra função associada a *thread* 4. Essa função então calcula um novo *CRC1* randômico e diferente do original e um novo número de sequência randômico e diferente do original.

O propósito de se usar um *CRC1* diferente é simular a troca do *F-Parameter Codename*, trocando o relacionamento único entre *F-Host* e *F-Device*. Já a troca do número de sequência diz respeito ao fato de que somente os dois dispositivos do par de comunicação têm controle do valor atual deste número. Como a mensagem é para ser de fora da comunicação *F-Host* e *F-Device*, o número de sequência será diferente também. Esses novos valores gerados para o *CRC1* e o número de sequência são usados com o C*ontrol/Status Byte* e com os dados do pacote escolhido para gerar um novo valor de *CRC2*, o qual é inserido no pacote que foi copiado. Após essa alteração do *CRC2*, o pacote fica em espera até que o tempo para enviá-lo (tempo 2) tenha decorrido no contador de tempo da *thread* 4. Assim, que o tempo 2 é atingido, o pacote modificado é enviado no canal de comunicação. Após a injeção de falhas, a troca de mensagens segue normalmente, até que o contador na *thread* 3 atinja novamente o tempo especificado e o ciclo de injeção de falhas seja reiniciado.

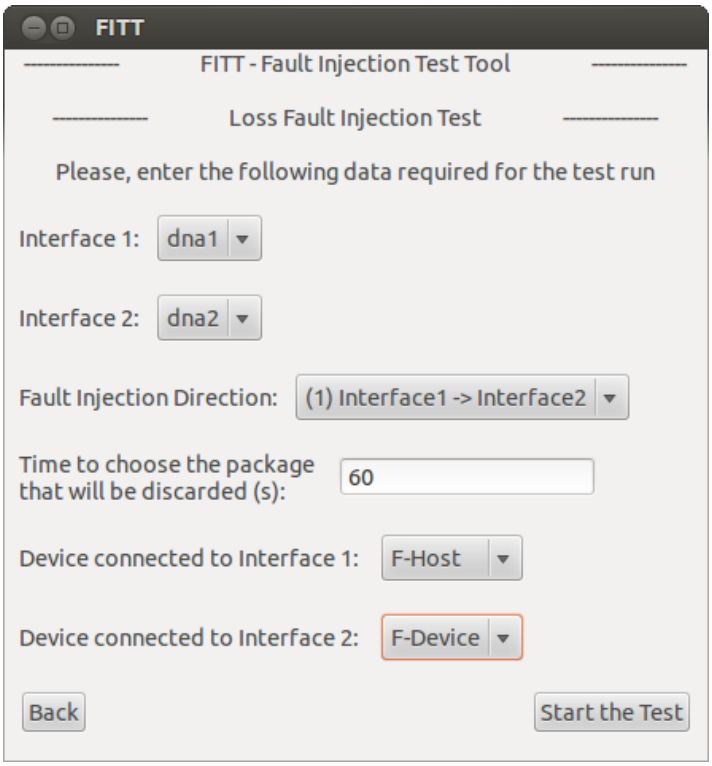
5.4.6 Função de Descarte de Pacotes

Na função de perda de mensagens, o injetor é configurado de forma a descartar (não enviar) as mensagens que forem selecionadas entre as que estiverem sendo recebidas em um dos sentidos da comunicação entre o *F-Host* e o *F-Device*. Este teste emula um dispositivo de barramento defeituoso que apaga mensagens seguras aleatoriamente. A sua janela de configuração é apresentada na figura 5.11.

A configuração do teste é feita pela definição de qual das interfaces *DNA* corresponde a qual das interfaces de rede 1 e 2 utilizadas pelo injetor. Além disso, é necessário selecionar

o sentido da injeção de falhas e o tempo de escolha do pacote que será descartado (tempo 1). Por fim, também é preciso informar qual dos dispositivos *PROFIsafe* está ligado em cada uma dessas duas interfaces.

Figura 5.11 - Interface da Função de Descarte de Mensagens



Fonte: Janela de Configuração do Teste de Descarte de Mensagens.

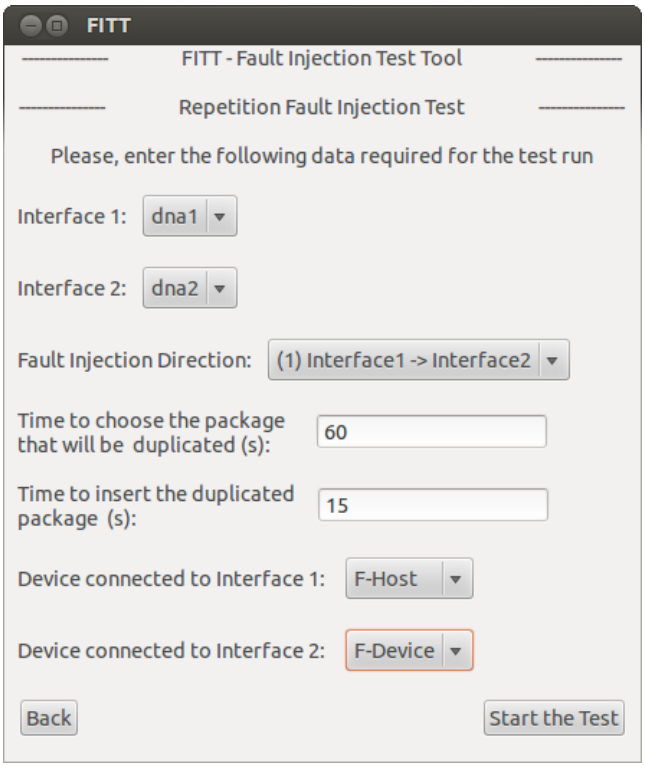
Quando a troca de pacotes entre os dispositivos mestre e escravo começa, é iniciado um contador de tempo da *thread* 3, o qual é incrementado até atingir o valor contido no tempo 1. Nesse momento, o injetor estabelece que o próximo pacote que chegar no sentido especificado para injeção de falha, será descartado. O descarte do pacote selecionado é feito simplesmente não enviando ele para o seu destino. Quando os próximos pacotes chegarem, eles serão enviados normalmente ao seu destino, até que o contador atinja novamente o tempo especificado e o ciclo de injeção de falhas se repita.

5.4.7 Função de Repetição de Mensagens

Finalmente, a função de repetição de mensagens tem por objetivo emular a recepção em duplicata de mensagens antigas e obsoletas. A função reproduz o mau funcionamento de um dispositivo que faz com que mensagens antigas e obsoletas sejam repetidas em um tempo equivocado, o que pode causar distúrbios perigosos ao receptor. A janela da função de

repetição é mostrada a seguir, na figura 5.12.

Figura 5.12 - Interface da Função de Repetição de Mensagens



Fonte: Janela de Configuração do Teste de Repetição de Mensagens.

O operador de testes precisa configurar o teste selecionando as interfaces *DNA* que correspondem as interfaces de rede 1 e 2 usadas pelo injetor. A seguir, ele precisa definir o sentido da injeção de falhas, o tempo de escolha do pacote que será duplicado (tempo 1) e o tempo de espera para inserir o pacote duplicado (tempo 2). No final, também é necessário estabelecer qual dos dispositivos está associado a qual das interfaces de rede do injetor.

Assim que a comunicação para troca de mensagens entre os dispositivos *PROFIsafe* é iniciada, o injetor inicia o contador de tempo da *thread* 3, que é atualizado até atingir o valor em segundos contido no tempo 1. Nesse instante, o injetor sinaliza que o próximo pacote que chegar ao sentido especificado para injeção de falha, será duplicado.

O pacote escolhido é copiado e em seguida enviado ao seu destino. O pacote que foi duplicado fica guardado em um *buffer*, sob controle de outra função associada a *thread* 4, criada para este propósito. O pacote fica em espera até que o tempo para enviá-lo (tempo 2) tenha sido atingido no contador de tempo desta *thread*. Assim, que o tempo 2 foi atingido, o pacote duplicado é enviado ao seu destino. Após a injeção de falhas, a troca de mensagens segue normalmente, até que o tempo 1 tenha decorrido novamente e o ciclo de injeção de falhas seja reiniciado.

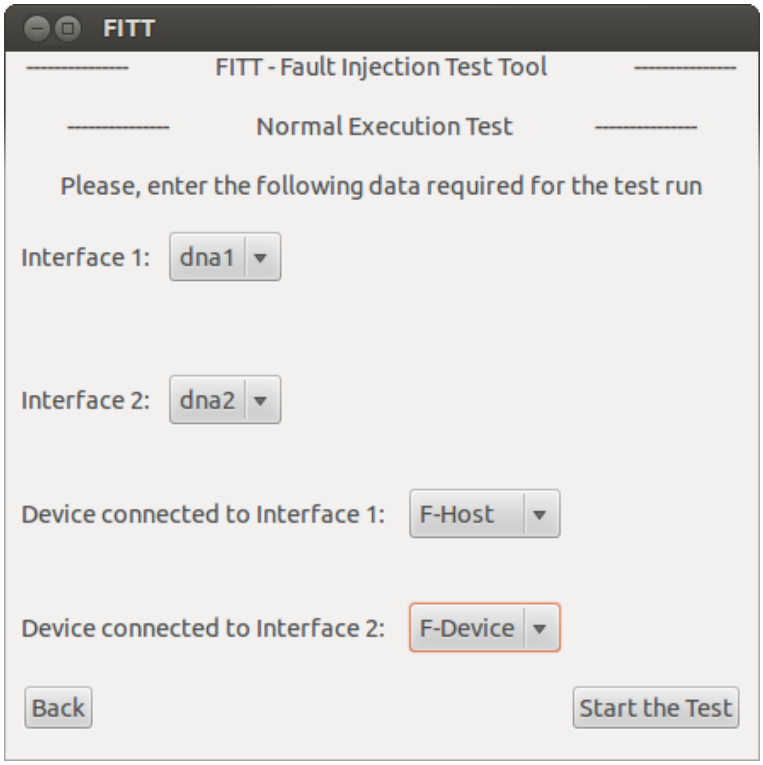
5.4.8 Função de Execução Normal

Além dos tipos de falhas apresentados acima, a ferramenta oferece uma opção de execução normal (sem falhas). Esta opção é a que fica selecionada por padrão na janela principal do programa. Este teste é o mais simples de todos e, por isso, requer a menor quantidade de informações do usuário, como é mostrado na figura 5.13.

Nela, o usuário precisa informar apenas qual das interfaces *DNA* corresponde a qual das interfaces de rede 1 e 2 usadas pelo injetor e qual dos dispositivos *F-Host* e *F-Device* está ligado em cada uma dessas duas interfaces.

O propósito dessa opção é de verificar se o ambiente de testes foi configurado corretamente, ou seja, se a comunicação entre os dispositivos mestre e escravo, passando pelo do injetor, está funcionando adequadamente para a realização dos testes de injeção de falhas. Nessa opção, as mensagens trocadas entre os dispositivos são transmitidas sem qualquer interferência do injetor de falhas.

Figura 5.13 - Interface da Função de Execução Normal



Fonte: Janela de Configuração do Teste de Execução Normal.

6 TESTES REALIZADOS COM O PROTOCOLO PROFISAFE

Neste capítulo são apresentados os testes de injeção de falhas realizados para validar a implementação dos dispositivos *PROFIsafe F-Host* e *F-Device*. Nesses testes, os mecanismos de tolerância a falhas do protocolo de comunicação seguro *PROFIsafe* serão avaliados utilizando as funções de falhas do injetor de falhas *FITT*.

6.1 Ambiente de Testes

Os testes foram realizados utilizando-se duas configurações diferentes de computadores. Na primeira delas foram utilizados dois notebooks *Intel C*ore i5 *dual core*, um da primeira geração, modelo *560M 2,66 GHz* com *4GB* de memória *RAM* onde o *F-Device* foi executado e outro da segunda geração, modelo *2410M 2,3 GHz* com *4GB* de memória *RAM* onde o *F-Host* foi instalado. O injetor de falhas foi configurado em um computador com processador *AMD Phenom II X4 975 3.6 GHz* com *4GB* de memória *RAM*. O sistema operacional utilizado foi o *Ubuntu* 12.04.5 *LTS 32bit*.

Já na segunda configuração para os testes foram utilizados três computadores com processador *Core i7 modelo 3770 3.4 GHz* com *16 GB* de memória *RAM* cada. Nos dois computadores onde o *F-Host* e o *F-Device* foram instalados, foi utilizado o sistema operacional *Ubuntu 14.04.3 LTS* de *64bit*. Por outro lado o *FITT* foi executado com o sistema operacional *Ubuntu 12.04.5 LTS* de *64bit*.

A utilização da versão *Ubuntu 12.04.5 LTS* foi necessária porque até o momento dos testes não existia suporte dos *drivers igb-DNA* do *PF_RING* para a versão 3.13 do *Kernel* do *Linux* utilizada no *Ubuntu 14.04.3 LTS*.

Adicionalmente, no computador em que o injetor de falhas foi instalado, foram utilizadas duas placas de rede *Intel Gigabit* modelo *e1000e 82574L* que são compatíveis com o modo *DNA* do *PF_RING* e são responsáveis por interligar os computadores com as implementações dos dispositivos *PROFIsafe* ao injetor de falhas.

6.2 Comentários Sobre os Testes

O início de todos os testes segue o comportamento apresentado no teste de execução normal. Nas figuras a serem mostradas, a cor verde nos campos da planilha corresponde a mudanças no estado e no valor de vaiáveis do *F-Device* e a cor azul indica alterações no

estado e nas variáveis do *F-Host*. Nas figuras do *F-Device*, a linha com campos na cor azul indica o momento em que o *F-Host* recebeu o *ack* do *F-Device*. Reciprocamente, nas figuras do *F-Host*, a linha com campos na cor verde indica o momento em que o *F-Device* recebeu a mensagem enviada pelo *F-Host*. Nas figuras existem dois campos chamados *CRC2 Host* e *CRC2 Dev*. Quando o campo *CRC2 Host* estiver preenchido e o *CRC2 Dev* estiver vazio, indica que o *F-Host* enviou uma nova mensagem ao *F-Device*. Da mesma forma, quando o campo *CRC2 Dev* está preenchido e o *CRC2 Host está* vazio significa que o *F-Device* enviou um novo *ack*. As linhas que mostram os dois valores de *CRC2* juntos, indicam que os dispositivos estão verificando a mensagem ou *ack* quanto à ocorrência ou não de erros.

6.3 Testes Realizados

A seguir são apresentados os testes realizados com o injetor de falhas *FITT* e o protocolo de comunicação seguro *PROFIsafe*. Em todos os testes foi utilizada a mesma configuração de tempo, com 60s como o tempo de escolha do pacote a sofrer o processo de injeção de falhas (tempo 1), exceto no teste de execução normal que não utiliza temporização. Nos testes de atraso, inserção de pacotes e repetição foi definido 15s como o tempo 2 para enviar o pacote atrasado, duplicado ou a ser inserido na comunicação.

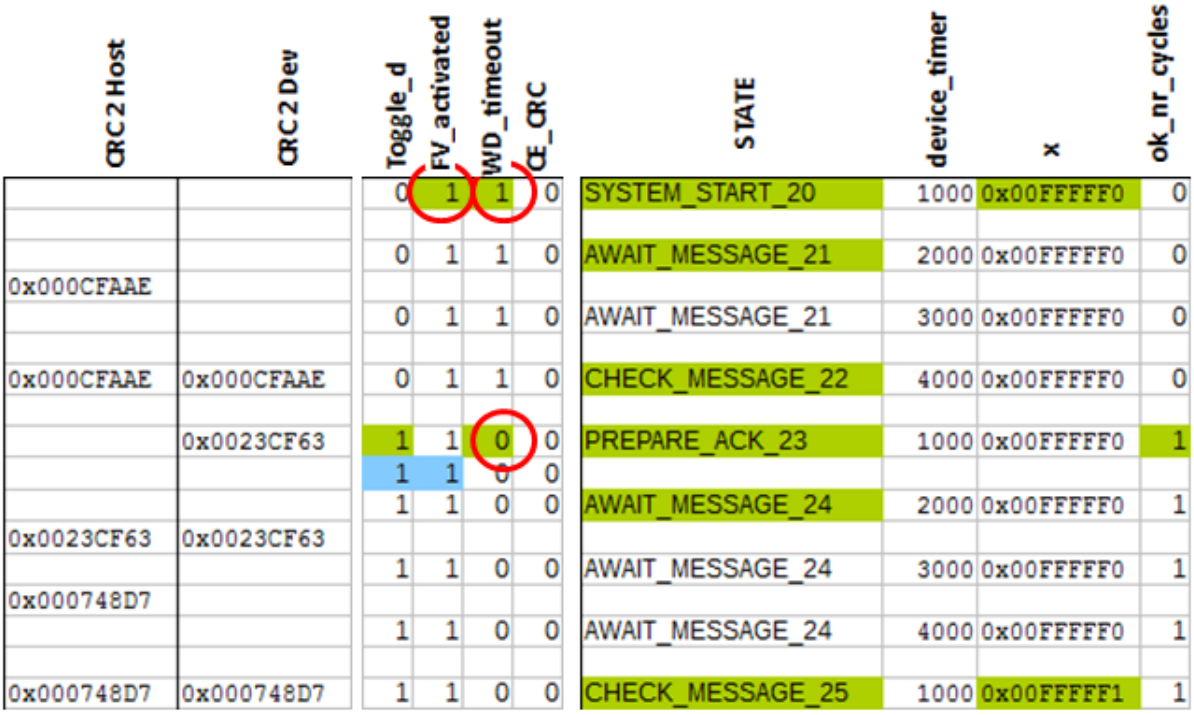
Os valores para os tempos 1 e 2 foram definidos de forma a permitir que a injeção de falhas seja realizada e que as máquinas de estados dos dispositivos *F-Host* e *F-Device* possam se recuperar das falhas de comunicação inseridas. Os tempos especificados não são os tempos mínimos que permitem que essa situação aconteça e foram especificados conforme o funcionamento da implementação utilizada do protocolo *PROFIsafe*.

6.3.1 Teste de Execução Normal

Nesse teste, a comunicação dos dispositivos *PROFIsafe* não é perturbada pelo injetor de falhas. O *F-Device* é inicializado com *FV_activated=1, WD_timeout=1 e* o *F-Host* com *activate_FV =1, FV_activated_S =1* , como mostram as figuras 6.1 e 6.2. Essas variáveis indicam que o *F-Host* deve fornecer dados de saída seguros (*FVo*) para a aplicação no *F-Device* se ele for um atuador. Por definição o *F-Device* inicia fornecendo dados seguros de entrada (*FVi*) para a aplicação no *F-Host* se ele for um sensor. Além disso, os dispositivos iniciam em seu ciclo normal de funcionamento (5, 6 e 4 no *F-Host*; 23, 24 e 25 no *F-Device*).

O *bit WD_timeout* recebe o valor “0” na primeira mensagem sem erro recebida do *F-Host*.

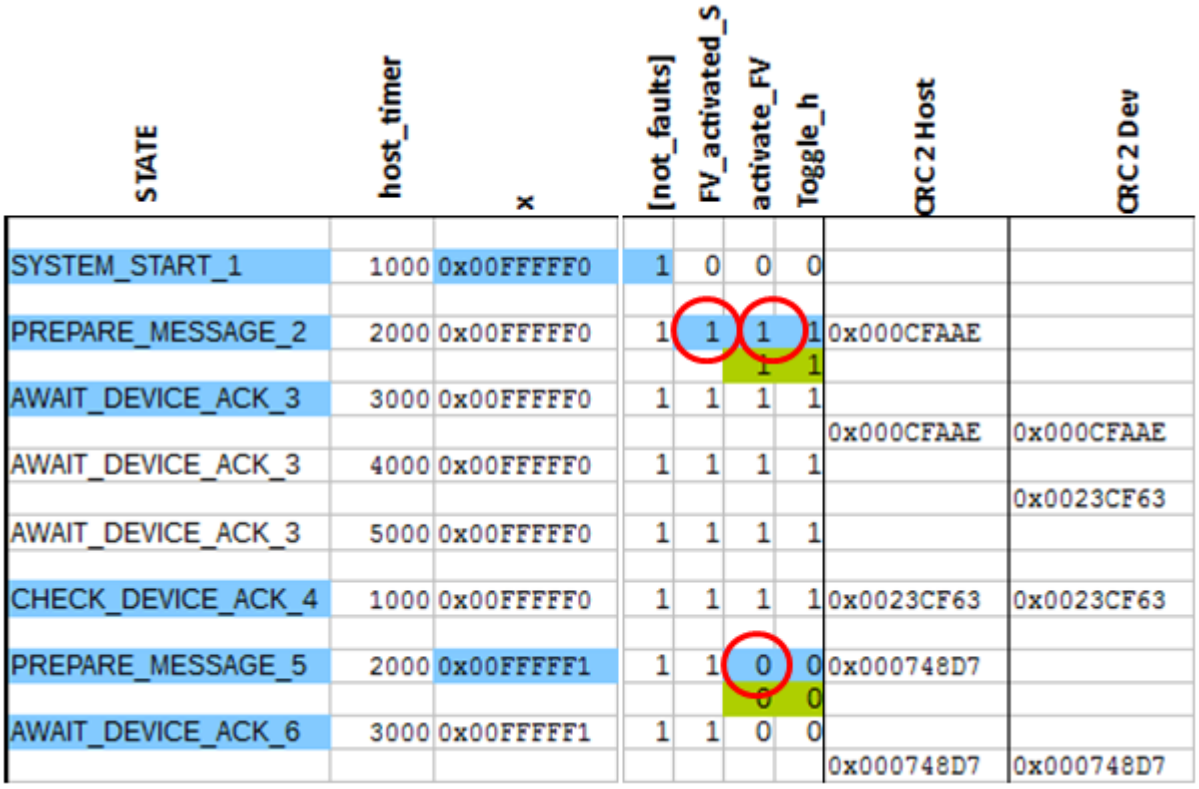
Figura 6.1 – Inicialização do *F-Device*



CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	WD_timeout	CE_CRC	STATE	device_timer	x	ok_nr_cycles
		0	1	1	0	SYSTEM_START_20	1000	0x00FFFFF0	0
		0	1	1	0	AWAIT_MESSAGE_21	2000	0x00FFFFF0	0
0x000CFAAE									
		0	1	1	0	AWAIT_MESSAGE_21	3000	0x00FFFFF0	0
0x000CFAAE	0x000CFAAE	0	1	1	0	CHECK_MESSAGE_22	4000	0x00FFFFF0	0
	0x0023CF63	1	1	0	0	PREPARE_ACK_23	1000	0x00FFFFF0	1
		1	1	0	0				
		1	1	0	0	AWAIT_MESSAGE_24	2000	0x00FFFFF0	1
0x0023CF63	0x0023CF63								
		1	1	0	0	AWAIT_MESSAGE_24	3000	0x00FFFFF0	1
0x000748D7									
		1	1	0	0	AWAIT_MESSAGE_24	4000	0x00FFFFF0	1
0x000748D7	0x000748D7	1	1	0	0	CHECK_MESSAGE_25	1000	0x00FFFFF1	1

Fonte: Planilha Gerada com os Resultados do Teste de Execução Normal.

Figura 6.2 – Inicialização do *F-Host*



STATE	host_timer	x	[not_faults]	FV_activated_S	activate_FV	Toggle_h	CRC 2 Host	CRC 2 Dev
SYSTEM_START_1	1000	0x00FFFFF0	1	0	0	0		
PREPARE_MESSAGE_2	2000	0x00FFFFF0	1	1	1	1	0x000CFAAE	
					1	1		
AWAIT_DEVICE_ACK_3	3000	0x00FFFFF0	1	1	1	1		
							0x000CFAAE	0x000CFAAE
AWAIT_DEVICE_ACK_3	4000	0x00FFFFF0	1	1	1	1		
								0x0023CF63
AWAIT_DEVICE_ACK_3	5000	0x00FFFFF0	1	1	1	1		
CHECK_DEVICE_ACK_4	1000	0x00FFFFF0	1	1	1	1	0x0023CF63	0x0023CF63
PREPARE_MESSAGE_5	2000	0x00FFFFF1	1	1	0	0	0x000748D7	
					0	0		
AWAIT_DEVICE_ACK_6	3000	0x00FFFFF1	1	1	0	0		
							0x000748D7	0x000748D7

Fonte: Planilha Gerada com os Resultados do Teste de Execução Normal.

Os *bits* T*oggle_d* e T*oggle_h*, assim como os números de sequência dos dispositivos, estão sincronizados para um funcionamento normal, ou seja, *Toggle_h == Toggle_d* quando o *F-Host* recebe um *ack* do *F-Device* e *Toggle_h >< Toggle_d* quando o *F-Device* recebe uma mensagem do *F-Host*.

Após o *F-Host* receber o primeiro *ack* bem sucedido, a variável *activate_FV* recebe o valor "0", como pode visto na figura 6.2. Na próxima mensagem envida pelo *F-Host*, esta variável com o valor "0" vai indicar ao *F-Device* que ele deve voltar a trabalhar com dados normais de operação de saída *(PVo)* se ele for um dispositivo de saída.

Depois de quatro recebimentos normais de mensagens no *F-Device*, a variável *ok_nr_cycles* recebe o valor "4", como pode ser visto na figura 6.3, sendo esta a condição necessária para que o *Status Bit FV_activated* receba o valor "0". Apenas a partir do próximo *ack* enviado para o *F-Host*, é que o *F-Device* avisa o *F-Host* que não existem mais variáveis de segurança ativas e que a partir daquele momento, o *F-Device* vai enviar os valores normais de operação (*PVo*) recebidos do *F-Host* para a aplicação no *F-Device* se ele for um dispositivo de saída.

Figura 6.3 – Condições necessárias para o *F-Device* entrar no ciclo normal de operação

CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	WD_timeout	STATE	device_timer	x	ok_nr_cycles
0x00D28E75								
		1	1	0	AWAIT_MESSAGE_24	5000	0x00FFFFF2	3
0x00D28E75	0x00D28E75	1	1	0	CHECK_MESSAGE_25	1000	0x00FFFFF3	3
	0x00FDBBB8	0	0	0	PREPARE_ACK_23	2000	0x00FFFFF3	4
		0	0	0				
		0	0	0	AWAIT_MESSAGE_24	3000	0x00FFFFF3	4
0x00FDBBB8	0x00FDBBB8							
		0	0	0	AWAIT_MESSAGE_24	4000	0x00FFFFF3	4
0x00EA27F5								

Fonte: Planilha Gerada com os Resultados do Teste de Execução Normal.

Quando o *F-Host* recebe o *ack* do *F-Device* com o *Status Bit FV_activated* igual a "0", ele atribui o valor "0" para a *FV_activated_S*. Assim, a aplicação segura no *F-Host* sabe que o *F-Device* está enviando dados normais de operação de entrada (*PVi*) para ela se ele for um dispositivo de entrada ou que ele está recebendo dados normais de operação de saída (*PVo*) do *F-Host driver* se ele for um dispositivo de saída. Isso é mostrado na figura 6.4 a seguir.

Figura 6.4 – Condição necessária para o *F-Host* entrar no ciclo normal de operação

STATE	host_timer	x	[not_faults]	FV_activated_S	activate_FV	Toggle_h	CRC 2 Host	CRC 2 Dev
PREPARE_MESSAGE_5	2000	0x00FFFFF3	1	1	0	0	0x00D28E75	
					0	0		
AWAIT_DEVICE_ACK_6	3000	0x00FFFFF3	1	1	0	0		
							0x00D28E75	0x00D28E75
AWAIT_DEVICE_ACK_6	4000	0x00FFFFF3	1	1	0	0		
								0x00FDBBB8
AWAIT_DEVICE_ACK_6	5000	0x00FFFFF3	1	1	0	0		
CHECK_DEVICE_ACK_4	1000	0x00FFFFF3	1	1	0	0	0x00FDBBB8	0x00FDBBB8
PREPARE_MESSAGE_5	2000	0x00FFFFF4	1	0	0	1	0x00EA27F5	

Fonte: Planilha Gerada com os Resultados do Teste de Execução Normal.

A partir desse momento, todas as variáveis de segurança estão com valores nulos e assim, os dispositivos seguem os seus respectivos ciclos normais de operação, pois nenhuma falha será inserida na comunicação entre eles.

6.3.2 Teste de Erro de Endereçamento

Nesse teste, uma mensagem escolhida na direção do *F-Host* para o *F-Device* tem o seu endereço de destino trocado, o que gera um valor diferente para o *CRC2* contido na mensagem. O *F-Device* detecta a corrupção da mensagem através da checagem do *CRC*, ativando os *Status Bits CE_CRC* e *FV_activated* e também passa a trabalhar com valores seguros para os dados de saída (*FVo*) que são entregues à aplicação do *F-Device* se ele for um dispositivo de saída. Ao detectar o erro, o dispositivo *F-Device* vai para o ciclo de estados 26, 27 e 28 para tratamento de falha, como é apresentado na figura 6.5.

O *F-Host* vai receber um *ack* do *F-Device* com os *Status Bits CE_CRC="1"* indicando que o *F-Device* detectou um erro de *CRC e FV_activated="1"* sinalizando que o *F-Device* está em um estado seguro de operação, trabalhando com dados seguros se ele for um dispositivo de saída. Então o *F-Host* atribui o valor "1" para a variável *FV_activated_S* e para o *Control Bit activate_FV* e ele entra no ciclo de estados 8, 9 e 10 para tratamento de falha, como é mostrado na figura 6.6. A variável *FV_activated_S* com o valor "1" vai indicar

para a aplicação segura que o *F-Host driver* está enviando dados seguros (*FVo*) para o *F-Device* se ele for um atuador ou que ela está recebendo valores seguros do *F-Host driver* (*FVi*) se o *F-Device* for um sensor, por exemplo.

Figura 6.5 – Detecção de erro de *CRC* pelo *F-Device*

CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	CE_CRC	STATE	device_timer	x	ok_nr_cycles
0x001013D9								
		1	0	0	AWAIT_MESSAGE_24	5000	0x00FFFFFC	4
0x00C58014	0x001013D9	1	0	0	CHECK_MESSAGE_25	1000	0x00FFFFFD	4
	0x0017DACA	0	1	1	PREPARE_ACK_26	1000	0x00FFFFFD	0
		0	1	1				
		0	1	1	AWAIT_MESSAGE_27	2000	0x00FFFFFD	0

Fonte: Planilha Gerada com os Resultados do Teste de Erro de Endereçamento.

Figura 6.6 – A variável *FV_activated_S* e o *Control Bit activate_FV* recebem o valor "1" após aviso de erro de *CRC* pelo *F-Device*

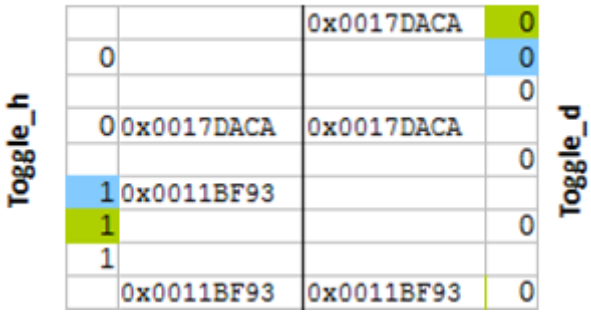
STATE	host_timer	x	[not_faults]	FV_activated_S	activate_FV	Toggle_h	CRC 2 Host	CRC 2 Dev
AWAIT_DEVICE_ACK_6	5000	0x00FFFFFD	1	0	0	0		
CHECK_DEVICE_ACK_4	1000	0x00FFFFFD	0	0	0	0	0x0017DACA	0x0017DACA
PREPARE_MESSAGE_8	2000	0x00000000	0	1	1	1	0x0011BF93	
					1	1		
AWAIT_DEVICE_ACK_9	3000	0x00000000	0	1	1	1		

Fonte: Planilha Gerada com os Resultados do Teste de Erro de Endereçamento.

Os dispositivos *F-Host* e *F-Device* seguem trocando mensagens sem descartá-las porque o *Toggle_h* e o *Toggle_d* estão sincronizados, como é apresentado na figura 6.7, ou seja, *Toggle_h == Toggle_d* quando o *F-Host* recebe um *ack* do *F-Device* e *Toggle_h >< Toggle_d* quando o *F-Device* recebe uma mensagem do *F-Host*.

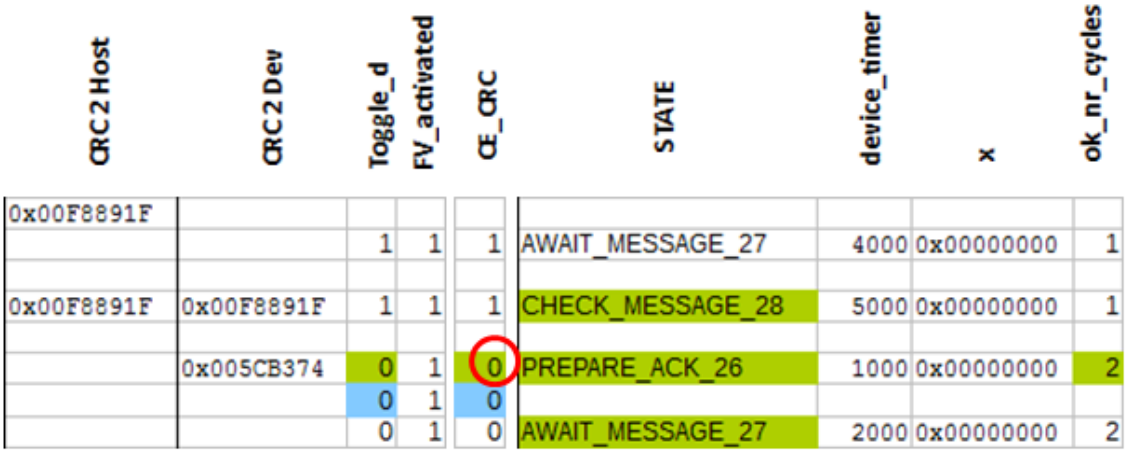
Após duas mensagens recebidas com sucesso, como é mostrado na figura 6.8, o *CE_CRC* do *F-Device* recebe o valor "0", assim o *F-Host* reconhece que não há mais falhas informadas pelo *F-Device*, alterando o valor da sua variável *not_faults* para "1".

Figura 6.7 – *Toggle Bits* sincronizados entre os dois dispositivos *PROFIsafe*



Fonte: Planilha Gerada com os Resultados do Teste de Erro de Endereçamento.

Figura 6.8 – Após o recebimento de duas mensagens sem falhas do *F-Host*, o *CE_CRC* do *F-Device* não reporta mais falhas



Fonte: Planilha Gerada com os Resultados do Teste de Erro de Endereçamento.

Após três recebimentos normais (ocasionando o incremento da variável *ok_nr_cycles* no *F-Device* para o valor "3"), o *F-Device* retorna ao seu ciclo normal (sem falhas) dos estados 23, 24 e 25.

No quarto recebimento normal das mensagens enviadas pelo *F-Host*, a variável *ok_nr_cycles* recebe o valor "4", sendo a condição necessária para o *Status Bit FV_activated* receber o valor "0", como ilustra a figura 6.9 a seguir. Contudo, o *F-Device* vai seguir trabalhando com valores seguros de saída (*FVo*), caso ele seja um dispositivo de saída, até que ele receba do *F-Host* o *Control Bit activate_FV* com o valor "0".

Quando o *F-Host* receber o *ack* do *F-Device*, ele vai atribuir o valor "0" para o *Control Bit activate_FV* e para a variável *FV_activated_S* porque o *Status Bit FV_activated* tem valor "0". A variável *FV_activated_S* com o valor "0" vai indicar para a aplicação segura que o *F-Host driver* vai enviar dados de operação normal (*PVi*) do *F-Device* para ela se ele

for um dispositivo de entrada ou que o *F-Device* vai receber dados normais de operação (*PVo*) do *F-Host driver* se ele for um dispositivo de saída.

Figura 6.9 – *F-Device* retorna ao estado normal de operação e depois desliga o *Status Bit FV_activated* que será enviado ao *F-Host*

CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	CE_CRC	STATE	device_timer	x	ok_nr_cycles
0x00F23847	0x00F23847	1	1	0	CHECK_MESSAGE_25	1000	0x00000002	3
	0x00F9DB71	0	0	0	PREPARE_ACK_23	2000	0x00000002	4
		0	0	0				
		0	0	0	AWAIT_MESSAGE_24	3000	0x00000002	4
0x00F9DB71	0x00F9DB71							
		0	0	0	AWAIT_MESSAGE_24	4000	0x00000002	4
0x00F1AF6E								
		0	0	0	AWAIT_MESSAGE_24	5000	0x00000002	4
0x00F1AF6E	0x00F1AF6E	0	0	0	CHECK_MESSAGE_25	1000	0x00000003	4

Fonte: Planilha Gerada com os Resultados do Teste de Erro de Endereçamento.

Assim que o operador ativar o *OA_C* (*Operator Acknowledgement*) na aplicação do *F-Host*, ele retorna ao ciclo normal de operação dos estados 5, 6 e 4, como mostra a figura 6.10.

Figura 6.10 – *F-Host* desliga os *bits FV_activated_S* e *activated_FV* e retorna ao seu estado normal de operação

STATE	host_timer	x	[not_faults]	FV_activated_S	OA_C	activate_FV	Toggle_h	CRC 2 Host	CRC 2 Dev
AWAIT_DEVICE_ACK_9	3000	0x00000002	1	1	0	1	0		
								0x00F23847	0x00F23847
AWAIT_DEVICE_ACK_9	4000	0x00000002	1	1	1	1	0		
									0x00F9DB71
AWAIT_DEVICE_ACK_9	5000	0x00000002	1	1	1	1	0		
CHECK_DEVICE_ACK_1	1000	0x00000002	1	1	1	1	0	0x00F9DB71	0x00F9DB71
PREPARE_MESSAGE_5	2000	0x00000003	1	0	0	0	1	0x00F1AF6E	
						0	1		
AWAIT_DEVICE_ACK_6	3000	0x00000003	1	0	0	0	1		
								0x00F1AF6E	0x00F1AF6E
AWAIT_DEVICE_ACK_6	4000	0x00000003	1	0	0	0	1		
									0x00DE9AA3
AWAIT_DEVICE_ACK_6	5000	0x00000003	1	0	0	0	1	0x00DE9AA3	0x00DE9AA3
CHECK_DEVICE_ACK_4	1000	0x00000003	1	0	0	0	1		

Fonte: Planilha Gerada com os Resultados do Teste de Erro de Endereçamento.

No instante em que o *F-Host* enviar uma nova mensagem, o *F-Device* vai receber o *Control Bit activate_FV* com o valor “0”, e assim ele volta a entregar dados normais de operação (*PVo*) para a aplicação do *F-Device* se ele for um dispositivo de saída.

6.3.3 Teste de Corrupção de Dados

Nesse experimento, a mensagem escolhida terá um dos dados do *safety PDU* (*bit* ou *byte*) corrompido. Como ambos os tipos de testes de corrupção (de um *bit* ou de um *byte* inteiro) levam a mesma transição de estados nas máquinas de estados dos dispositivos *F-Host* e *F-Device*, será apresentado o teste de corrupção de um *bit* dos dados seguros do *safety PDU*. Esse teste foi escolhido com o objetivo de mostrar que mesmo uma pequena perturbação nos dados seguros transmitidos nas mensagens (corrupção de apenas 1 *bit*) é detectada pelos mecanismos de tolerância a falhas do protocolo *PROFIsafe*.

A mensagem escolhida no sentido *F-Host* → *F-Device* é corrompida. O *F-Device* detecta a corrupção da mensagem através da checagem do *CRC*, ativando os *Status Bits CE_CRC* e *FV_activated*, passando a trabalhar com valores seguros para os dados de saída (*FVo*) que são entregues à aplicação do *F-Device* se ele for um dispositivo de saída. Ao detectar o erro, o *F-Device* vai para o ciclo de estados 26, 27 e 28 para tratamento de falha. Essa situação é mostrada na figura 6.11.

Figura 6.11 – Detecção de erro de *CRC* pelo *F-Device*

CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	CE_CRC	STATE	device_timer	x	ok_nr_cycles
0x001013D9								
		1	0	0	AWAIT_MESSAGE_24	5000	0x00FFFFFC	4
0x001013D9	0x0051EA69	1	0	0	CHECK_MESSAGE_25	1000	0x00FFFFFD	4
	0x0017DACA	0	1	1	PREPARE_ACK_26	1000	0x00FFFFFD	0
		0	1	1				
		0	1	1	AWAIT_MESSAGE_27	2000	0x00FFFFFD	0

Fonte: Planilha Gerada com os Resultados do Teste de Corrupção.

O *F-Host* vai receber um *ack* do *F-Device* com os *Status Bit CE_CRC=”1”* sinalizando que o *F-Device* detectou um erro de *CRC e FV_activated=”1”* indicando que o *F-Device* está em um estado seguro de operação, trabalhando com dados seguros se ele for

um dispositivo de saída. Então o *F-Host* atribui o valor “1” para a variável *FV_activated_S* e para o *Control Bit activate_FV* e ele entra no ciclo de estados 8, 9 e 10 para tratamento de falha, como é apresentado na figura 6.12. A variável *FV_activated_S* com o valor “1” vai indicar para a aplicação segura que o *F-Host driver* está enviando dados seguros (*FVo*) para o *F-Device* se ele for um atuador ou que ela está recebendo valores seguros do *F-Host driver* (*FVi*) se o *F-Device* for um sensor, por exemplo.

Figura 6.12 – A variável *FV_activated_S* e o *Control Bit activate_FV* recebem o valor “1” após aviso de erro de *CRC* pelo *F-Device*

STATE	host_timer	x	[not_faults]	FV_activated_S	activate_FV	Toggle_h	CRC 2 Host	CRC 2 Dev
AWAIT_DEVICE_ACK_6	5000	0x00FFFFFD	1	0	0	0		
CHECK_DEVICE_ACK_4	1000	0x00FFFFFD	0	0	0	0	0x0017DACA	0x0017DACA
PREPARE_MESSAGE_8	2000	0x00000000	0	1	1	1	0x0011BF93	
					1	1		
AWAIT_DEVICE_ACK_9	3000	0x00000000	0	1	1	1		

Fonte: Planilha Gerada com os Resultados do Teste de Corrupção.

Os dispositivos *F-Host* e *F-Device* seguem trocando mensagens sem descartá-las porque o *Toggle_h* e o *Toggle_d* estão sincronizados, como ilustrado na figura 6.13, ou seja, *Toggle_h == Toggle_d* quando o *F-Host* recebe um *ack* do *F-Device* e *Toggle_h >< Toggle_d* quando o *F-Device* recebe uma mensagem do *F-Host*.

Figura 6.13 – *Toggle Bits* sincronizados entre os dois dispositivos *PROFIsafe*

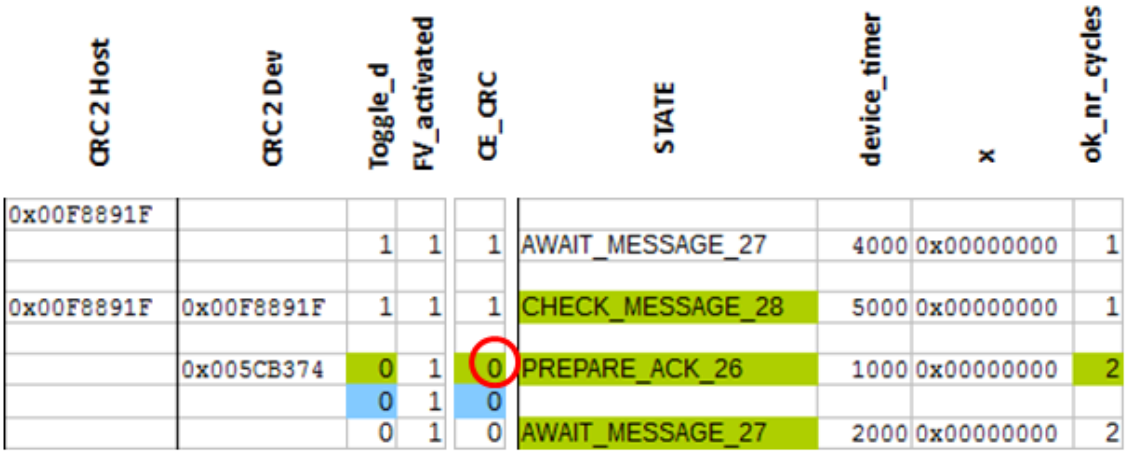
Toggle_h			Toggle_d
		0x0017DACA	0
0			0
			0
0	0x0017DACA	0x0017DACA	
			0
1	0x0011BF93		
1			0
1			
	0x0011BF93	0x0011BF93	0

Fonte: Planilha Gerada com os Resultados do Teste de Corrupção.

Após duas mensagens recebidas com sucesso, como apresentado na figura 6.14, o *CE_CRC* do *F-Device* recebe o valor “0”, assim o *F-Host* reconhece que não há mais falhas

informadas pelo *F-Device*, alterando o valor da sua variável *not_faults* para "1". Após três recebimentos normais (ocasionando o incremento da variável *ok_nr_cycles* no *F-Device* para o valor "3"), o *F-Device* retorna ao seu ciclo normal (sem falhas) dos estados 23, 24 e 25.

Figura 6.14 – Após o recebimento de duas mensagens sem falhas do *F-Host*, o *CE_CRC* do *F-Device* não reporta mais falhas ao *F-Host*

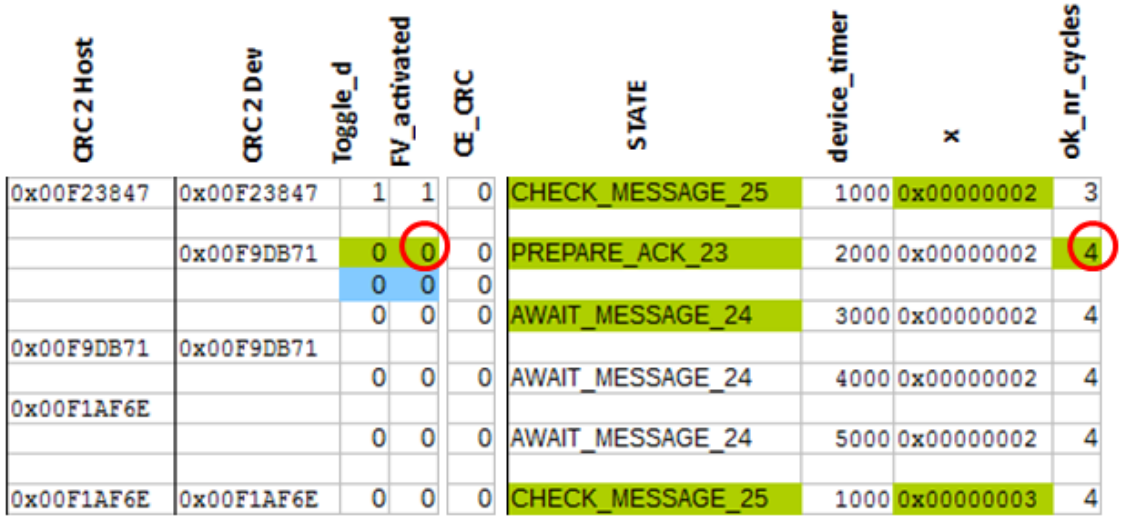


CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	CE_CRC	STATE	device_timer	x	ok_nr_cycles
0x00F8891F								
		1	1	1	AWAIT_MESSAGE_27	4000	0x00000000	1
0x00F8891F	0x00F8891F	1	1	1	CHECK_MESSAGE_28	5000	0x00000000	1
	0x005CB374	0	1	0	PREPARE_ACK_26	1000	0x00000000	2
		0	1	0				
		0	1	0	AWAIT_MESSAGE_27	2000	0x00000000	2

Fonte: Planilha Gerada com os Resultados do Teste de Corrupção.

No quarto recebimento normal das mensagens enviadas pelo *F-Host*, a variável *ok_nr_cycles* recebe o valor "4", sendo a condição necessária para o *Status Bit FV_activated* receber o valor "0". Essa situação é apresentada na figura 6.15. Contudo, o *F-Device* vai seguir trabalhando com valores seguros de saída (*FVo*), caso ele seja um dispositivo de saída, até que ele receba do *F-Host* o *Control Bit activate_FV* com o valor "0".

Figura 6.15 – *F-Device* retorna ao estado normal de operação e depois desliga o *Status Bit FV_activated* que será enviado ao *F-Host*



CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	CE_CRC	STATE	device_timer	x	ok_nr_cycles
0x00F23847	0x00F23847	1	1	0	CHECK_MESSAGE_25	1000	0x00000002	3
	0x00F9DB71	0	0	0	PREPARE_ACK_23	2000	0x00000002	4
		0	0	0				
		0	0	0	AWAIT_MESSAGE_24	3000	0x00000002	4
0x00F9DB71	0x00F9DB71							
		0	0	0	AWAIT_MESSAGE_24	4000	0x00000002	4
0x00F1AF6E								
		0	0	0	AWAIT_MESSAGE_24	5000	0x00000002	4
0x00F1AF6E	0x00F1AF6E	0	0	0	CHECK_MESSAGE_25	1000	0x00000003	4

Fonte: Planilha Gerada com os Resultados do Teste de Corrupção.

Quando o *F-Host* receber o *ack* do *F-Device*, ele vai atribuir o valor “0” para o *Control Bit activate_FV* e para a variável *FV_activated_S* porque o *Status Bit FV_activated* tem valor “0”. A variável *FV_activated_S* com o valor “0” vai indicar para a aplicação segura que o *F-Host driver* vai enviar dados de operação normal (*PVi*) do *F-Device* para ela se ele for um dispositivo de entrada ou que o *F-Device* vai receber dados normais de operação (*PVo*) do *F-Host driver* se ele for um dispositivo de saída. Assim que o operador ativar o *OA_C* (*Operator Acknowledgement*) na aplicação do *F-Host*, este retorna ao ciclo normal de operação dos estados 5, 6 e 4, como é ilustrada na figura 6.16.

Figura 6.16 – *F-Host* desliga os *bits FV_activated_S* e *activated_FV* e retorna ao seu estado normal de operação

STATE	host_timer	x	[not_faults]	FV_activated_S	OA_C	activate_FV	Toggle_h	CRC 2 Host	CRC 2 Dev
AWAIT_DEVICE_ACK_9	3000	0x00000002	1	1	0	1	0		
								0x00F23847	0x00F23847
AWAIT_DEVICE_ACK_9	4000	0x00000002	1	1	0	1	0		
									0x00F9DB71
AWAIT_DEVICE_ACK_9	5000	0x00000002	1	1	1	1	0		
CHECK_DEVICE_ACK_1	1000	0x00000002	1	1	1	1	0	0x00F9DB71	0x00F9DB71
PREPARE_MESSAGE_5	2000	0x00000003	1	0	0	0	1	0x00F1AF6E	
						0	1		
AWAIT_DEVICE_ACK_6	3000	0x00000003	1	0	0	0	1		
								0x00F1AF6E	0x00F1AF6E
AWAIT_DEVICE_ACK_6	4000	0x00000003	1	0	0	0	1		
									0x00DE9AA3
AWAIT_DEVICE_ACK_6	5000	0x00000003	1	0	0	0	1		
CHECK_DEVICE_ACK_4	1000	0x00000003	1	0	0	0	1	0x00DE9AA3	0x00DE9AA3

Fonte: Planilha Gerada com os Resultados do Teste de Corrupção.

No instante em que o *F-Host* enviar uma nova mensagem, o *F-Device* vai receber o *Control Bit activate_FV* com o valor “0”, e assim ele volta a entregar dados normais de operação (*PVo*) para a aplicação do *F-Device* se ele for um dispositivo de saída.

6.3.4 Teste de Atraso de Pacotes

Neste experimento, a mensagem escolhida vinda do *F-Host* para o *F-Device* é atrasada para além do tempo de *timeout* configurado nos dispositivos. O *F-Device* detecta o atraso

através da contagem de *timeout*, como pode ser visto na figura 6.17. Ele então ativa os *Status Bits WD_timeout* e *FV_activated* e também passa a trabalhar com valores seguros para os dados de saída (*FVo*) que são entregues à aplicação do *F-Device* se ele for um dispositivo de saída. Ao detectar o não recebimento da mensagem, o *F-Device* entra no ciclo de estados 26, 27 e 28 para tratamento de falhas.

Figura 6.17 – Detecção de erro de *Timeout* pelo *F-Device*

CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	WD_timeout	STATE	device_timer	x	ok_nr_cycles
0x001013D9								
		1	0	0	AWAIT_MESSAGE_24	5000	0x00FFFFFC	4
		1	0	0	AWAIT_MESSAGE_24	6000	0x00FFFFFC	4
		1	0	0	AWAIT_MESSAGE_24	7000	0x00FFFFFC	4
		1	0	0	AWAIT_MESSAGE_24	8000	0x00FFFFFC	4
		1	0	0	AWAIT_MESSAGE_24	9000	0x00FFFFFC	4
		1	0	0	AWAIT_MESSAGE_24	10000	0x00FFFFFC	4
		1	0	0	AWAIT_MESSAGE_24	11000	0x00FFFFFC	4
	0x00EFD758	1	1	1	PREPARE_ACK_26	1000	0x00FFFFFC	0
		1	1	1				
		1	1	1	AWAIT_MESSAGE_27	2000	0x00FFFFFC	0

Fonte: Planilha Gerada com os Resultados do Teste de Atraso.

Figura 6.18 – O *F-Host* vai para o estado 8 para tratar a falha de erro de *CRC* detectada

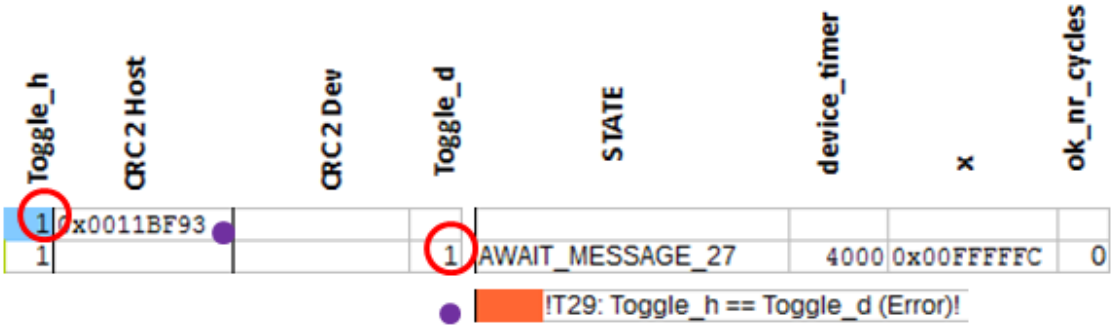
STATE	host_timer	x	[not_faults]	FV_activated_S	activate_FV	Toggle_h	CRC 2 Host	CRC 2 Dev
AWAIT_DEVICE_ACK_6	9000	0x00FFFFFD	1	0	0	0		
								0x00EFD758
AWAIT_DEVICE_ACK_6	10000	0x00FFFFFD	1	0	0	0		
CHECK_DEVICE_ACK_7	11000	0x00FFFFFD	0	0	0	0	0x00E6B80C	0x00EFD758
PREPARE_MESSAGE_8	1000	0x00000000	0	1	1	1	0x0011BF93	
					1	1		
AWAIT_DEVICE_ACK_9	2000	0x00000000	0	1	1	1		

Fonte: Planilha Gerada com os Resultados do Teste de Atraso.

Devido ao não recebimento da mensagem atrasada, o *Toggle_d* do *F-Device* não é incrementado e, assim, o *ack* enviado por ele após o *timeout* é recusado pelo *F-Host*, por erro de *CRC*. O *F-Host* então atribui o valor "1" para a variável *FV_activated_S* e para o *Control Bit activate_FV* e ele entra no ciclo de estados 8, 9 e 10 para tratamento de falha (figura 6.18). A variável *FV_activated_S* com o valor "1" vai indicar para a aplicação segura que o *F-Host driver* está enviando dados seguros (*FVo*) para o *F-Device* se ele for um atuador ou que ela está recebendo valores seguros do *F-Host driver* (*FVi*) se o *F-Device* for um sensor.

A mensagem seguinte do *F-Host* para o *F-Device* é descartada pela falta de sincronia entre os *bits* do *Toggle_d* e do *Toggle*_h, que são iguais, como é ilustrado pela figura 6.19.

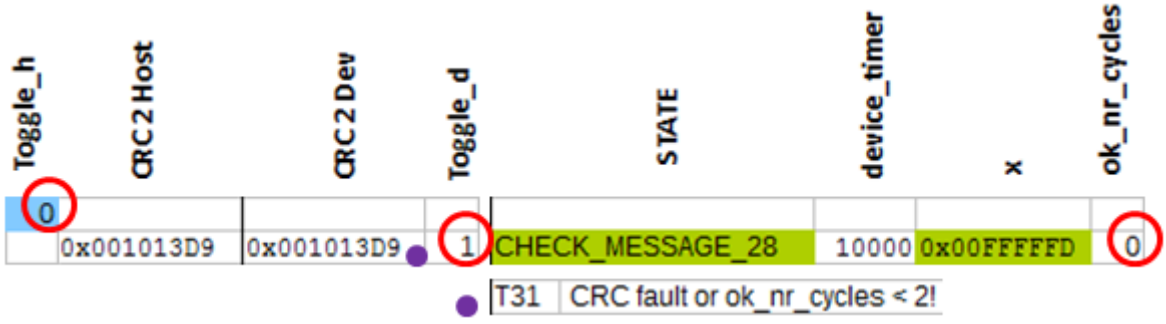
Figura 6.19 – *F-Device* não aceita mensagem do *F-Host* por falta de sincronia no *Toggle_h*



Fonte: Planilha Gerada com os Resultados do Teste de Atraso.

Em seguida, a mensagem atrasada é enviada pelo injetor para o *F-Device*. O *F-Device* não detecta erro de *CRC* porque o seu número consecutivo é incrementado para 0xFFFFFD (o mesmo do *F-Host* quando ele enviou essa mensagem) e o *Toggle_d* possui o valor "1", que é diferente do *Toggle_h* que possui o valor "0" quando essa mensagem foi enviada. No entanto, o *F-Device* continua no ciclo 26, 27 e 28 porque a variável *ok_nr_de_cycles* é menor do que o valor "2" (*ok_nr_de_cycles==0*). Essa situação é mostrada na figura 6.20.

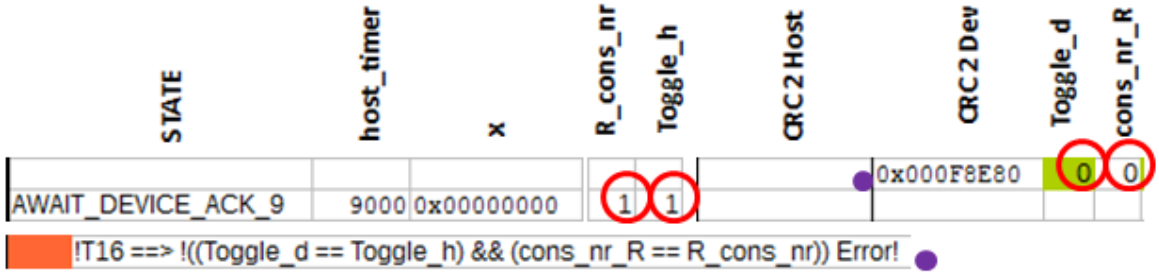
Figura 6.20 – A mensagem que foi atrasada não é aceita no *F-Device* porque a variável *ok_nr_de_cycles<2*



Fonte: Planilha Gerada com os Resultados do Teste de Atraso.

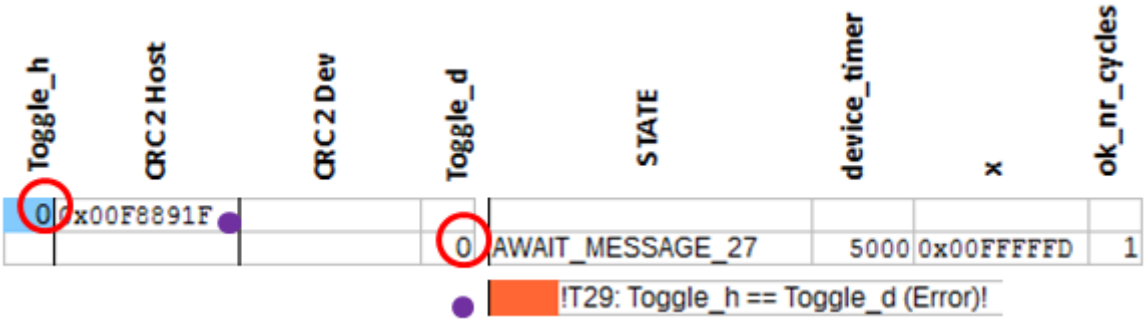
As próximas três mensagens, um *ack* do *F-Device* para o *F-Host*, a mensagem do *F-Host* para o *F-Device* e um novo *ack* do *F-Device* para o *F-Host* também são descartadas. O primeiro *ack* é descartado pela falta de sincronia entre os *bits Toggle_d* e *Toggle_h* que precisariam ser iguais, mas não são e também porque os sinais *R_cons_nr* e *cons_nr_R* são diferentes (figura 6.21). Já a mensagem enviada pelo *F-Host* não é aceita pela falta de sincronia entre os *Toggles Bits* que deveriam ser diferentes, mas são iguais, como é ilustrado na figura 6.22. O próximo *ack* enviado pelo *F-Device*, figura 6.23, não é aceito porque os *Toggles Bits* também estão fora de sincronia, eles deveriam ser iguais, mas são diferentes.

Figura 6.21 – *Ack* do *F-Device* descartado pelo *F-Host*



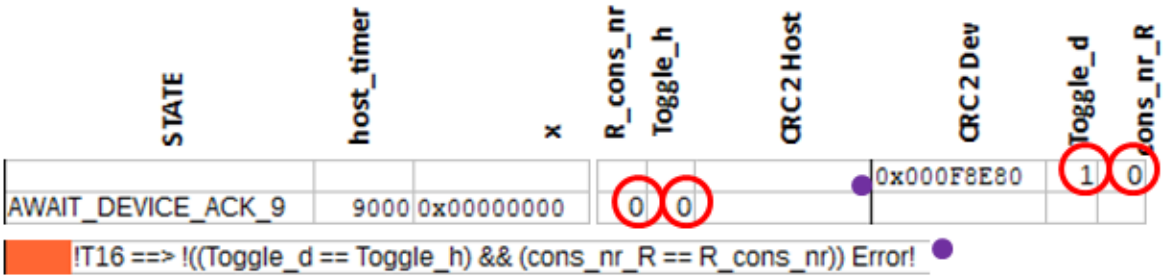
Fonte: Planilha Gerada com os Resultados do Teste de Atraso.

Figura 6.22 – Mensagem descartada do *F-Host* para o *F-Device*



Fonte: Planilha Gerada com os Resultados do Teste de Atraso.

Figura 6.23 – *Ack* do *F-Device* descartado pelo *F-Host*



Fonte: Planilha Gerada com os Resultados do Teste de Atraso.

Já as próximas mensagens do *F-Host* para o *F-Device* e os *acks* do *F-Device* para o *F-Host* são aceitas, pois os *bits* do *Toggle_d* e do *Toggle_h* voltaram à sincronia normal. Ou seja, *Toggle_h == Toggle_d* quando o *F-Host* recebe um *ack* do *F-Device* e *Toggle_h >< Toggle_d* quando o *F-Device* recebe uma mensagem do *F-Host*. Essa situação pode ser visualizada na figura 6.24.

Figura 6.24 – *Toggle Bits* sincronizados entre os dois dispositivos *PROFIsafe*

Toggle_h				Toggle_d
1	0x0005FA46			
1			0	
1				
	0x0005FA46	0x0005FA46	0	
1				
		0x007A8318	1	
1			1	
			1	
1	0x007A8318	0x007A8318		

Fonte: Planilha Gerada com os Resultados do Teste de Atraso.

Após duas mensagens recebidas com sucesso, como ilustrado na figura 6.25, o *WD_timeout* do *F-Device* recebe o valor “0”, assim o *F-Host* reconhece que não há mais falhas informadas pelo *F-Device*, alterando o valor da sua variável *not_faults* para “1”.

Figura 6.25 – Após o recebimento de duas mensagens sem falhas do *F-Host*, o *WD_timeout* do *F-Device* não reporta mais falhas ao *F-Host*

CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	WD_timeout	STATE	device_timer	x	ok_nr_cycles
0x00ECCCCA								
		1	1	1	AWAIT_MESSAGE_27	4000	0x00000000	1
0x00ECCCCA	0x00ECCCCA	1	1	1	CHECK_MESSAGE_28	5000	0x00000000	1
	0x0048F6A1	0	1	0	PREPARE_ACK_26	1000	0x00000000	2
		0	1	0				
		0	1	0	AWAIT_MESSAGE_27	2000	0x00000000	2

Fonte: Planilha Gerada com os Resultados do Teste de Atraso.

Após três recebimentos normais, ocasionando o incremento da variável *ok_nr_cycles* no *F-Device* para o valor “3”, este dispositivo volta ao ciclo normal (sem falha) dos estados 23, 24 e 25. No quarto recebimento normal das mensagens enviadas pelo *F-Host*, a variável *ok_nr_cycles* recebe o valor “4”, sendo a condição necessária para o *Status Bit FV_activated*

receber o valor “0”. Essa situação é apresentada pela figura 6.26, a qual pode ser vista a seguir. Contudo, o *F-Device* vai seguir trabalhando com valores seguros de saída (*FVo*), caso ele seja um dispositivo de saída, até que ele receba do *F-Host* o *Control Bit activate_FV* com o valor “0”.

Figura 6.26 – *F-Device* retorna ao estado normal de operação e depois desliga o *Status Bit FV_activated* que será enviado ao *F-Host*

CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	WD_timeout	STATE	device_timer	x	ok_nr_cycles
0x0035204D	0x0035204D	1	1	0	CHECK_MESSAGE_25	1000	0x00000002	3
	0x003EC37B	0	0	0	PREPARE_ACK_23	2000	0x00000002	4
		0	0	0				
		0	0	0	AWAIT_MESSAGE_24	3000	0x00000002	4
0x003EC37B	0x003EC37B							
		0	0	0	AWAIT_MESSAGE_24	4000	0x00000002	4
0x001E3CCE								
		0	0	0	AWAIT_MESSAGE_24	5000	0x00000002	4
0x001E3CCE	0x001E3CCE	0	0	0	CHECK_MESSAGE_25	1000	0x00000003	4

Fonte: Planilha Gerada com os Resultados do Teste de Atraso.

Figura 6.27 – *F-Host* desliga os *bits FV_activated_S* e *activated_FV* e retorna ao seu estado normal de operação

STATE	host_timer	x	[not_faults]	FV_activated_S	OA_C	activate_FV	Toggle_h	CRC 2 Host	CRC 2 Dev
AWAIT_DEVICE_ACK_9	3000	0x00000002	1	1	0	1	0		
								0x0035204D	0x0035204D
AWAIT_DEVICE_ACK_9	4000	0x00000002	1	1	0	1	0		
									0x003EC37B
AWAIT_DEVICE_ACK_9	5000	0x00000002	1	1	1	1	0		
CHECK_DEVICE_ACK_1	1000	0x00000002	1	1	1	1	0	0x003EC37B	0x003EC37B
PREPARE_MESSAGE_5	2000	0x00000003	1	0	0	0	1	0x001E3CCE	
						0	1		
AWAIT_DEVICE_ACK_6	3000	0x00000003	1	0	0	0	1		
								0x001E3CCE	0x001E3CCE
AWAIT_DEVICE_ACK_6	4000	0x00000003	1	0	0	0	1		
									0x00310903
AWAIT_DEVICE_ACK_6	5000	0x00000003	1	0	0	0	1		
CHECK_DEVICE_ACK_4	1000	0x00000003	1	0	0	0	1	0x00310903	0x00310903

Fonte: Planilha Gerada com os Resultados do Teste de Atraso.

Quando o *F-Host* receber o *ack* do *F-Device*, ele vai atribuir o valor "0" para o *Control Bit activate_FV* e para a variável *FV_activated_S* porque o *Status Bit FV_activated* tem valor "0". A variável *FV_activated_S* com o valor "0" vai indicar para a aplicação segura que o *F-Host driver* vai enviar dados de operação normal (*PVi*) do *F-Device* para ela se ele for um dispositivo de entrada ou que o *F-Device* vai receber dados normais de operação (*PVo*) do *F-Host driver* se ele for um dispositivo de saída.

Assim que o operador ativar o *OA_C* (*Operator Acknowledgement*) na aplicação do *F-Host*, ele retorna ao ciclo normal de operação dos estados 5, 6 e 4, como foi ilustrado na figura 6.27 acima. No instante em que o *F-Host* enviar uma nova mensagem, o *F-Device* vai receber o *Control Bit activate_FV* com o valor "0", e assim ele volta a entregar dados normais de operação (*PVo*) para a aplicação do *F-Device* se ele for um dispositivo de saída.

6.3.5 Teste de Sequência Incorreta

A mensagem escolhida do *F-Host* para o *F-Device* tem o valor do seu *Toggle_h* invertido. O *F-Device* detecta a corrupção da mensagem através da checagem do *CRC*, ativando os *Status Bits CE_CRC* e *FV_activated*, passando a trabalhar com valores seguros para os dados de saída (*FVo*) que são entregues à aplicação do *F-Device* caso ele seja um dispositivo de saída. Ao detectar o erro, o *F-Device* vai para o ciclo de estados 26, 27 e 28 para tratamento de falha, como é descrito na figura 6.28.

Figura 6.28 – Detecção de erro de *CRC* pelo *F-Device*

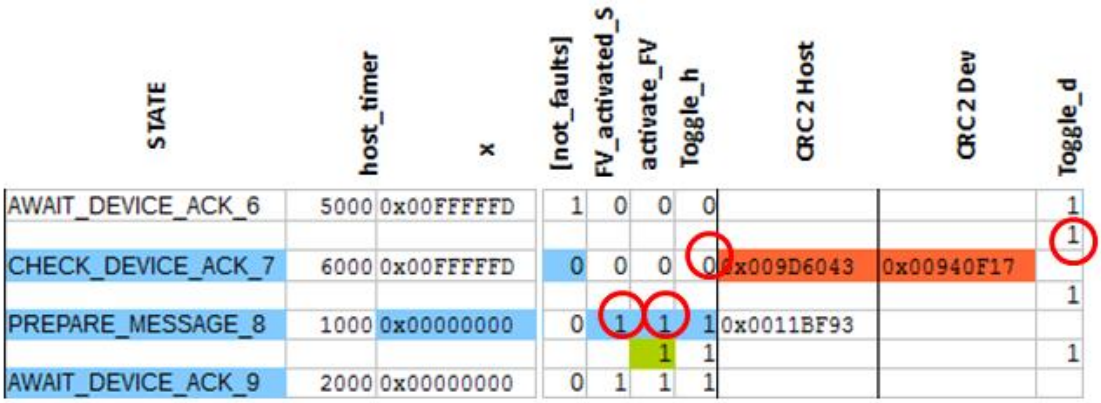
CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	CE_CRC	STATE	device_timer	x	ok_nr_cycles
0x001013D9								
		1	0	0	AWAIT_MESSAGE_24	5000	0x00FFFFFC	4
0x001013D9	0x0093C604	1	0	0	CHECK_MESSAGE_25	6000	0x00FFFFFC	4
	0x00940F17	1	1	1	PREPARE_ACK_26	1000	0x00FFFFFC	0
		1	1	1				
		1	1	1	AWAIT_MESSAGE_27	2000	0x00FFFFFC	0

Fonte: Planilha Gerada com os Resultados do Teste de Sequência Incorreta.

O *F-Host* recebe um *ack* do *F-Device* com o *Toggle_d* diferente do *Toggle_h*, gerando erro de *CRC* no dispositivo e com os *Status Bits CE_CRC="1"* indicando que o *F-Device*

detectou um erro de *CRC e FV_activated=”1”* sinalizando que o *F-Device* está em um estado seguro de operação, trabalhando com dados seguros se ele for um dispositivo de saída. A seguir o *F-Host* entra no ciclo de estados 8, 9 e 10 para tratamento de falha. Então o *F-Host* atribui o valor “1” para a variável *FV_activated_S* e para o *Control Bit activate_FV* como é mostrado na figura 6.29.

Figura 6.29 – A variável *FV_activated_S* e o *Control Bit activate_FV* recebem o valor “1” após aviso de erro de *CRC* pelo *F-Device*

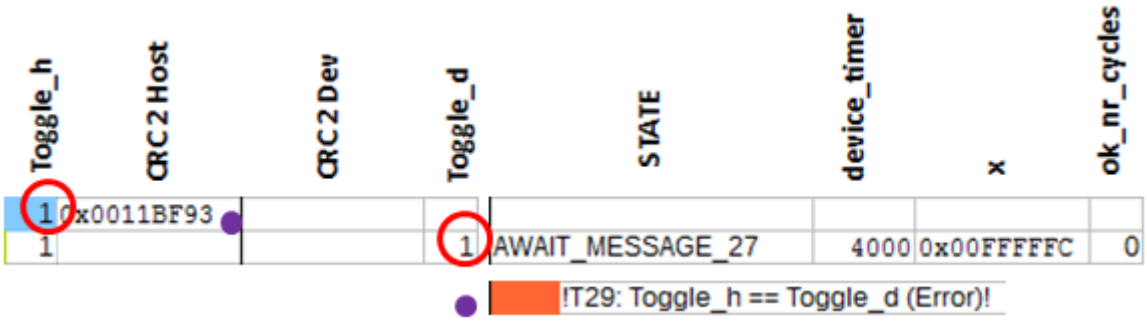


Fonte: Planilha Gerada com os Resultados do Teste de Sequência Incorreta.

A variável *FV_activated_S* com o valor “1” vai sinalizar para a aplicação segura no *F-Host* que o *F-Host driver* está enviando dados seguros (*FVo*) para o *F-Device* se ele for um atuador ou que ela está recebendo valores seguros de entrada do *F-Host driver* (*FVi*) se o *F-Device* for um sensor.

A próxima mensagem do *F-Host* para o *F-Device* será descartada pela falta de sincronia entre os *bits* do *Toggle_d* e do *Toggle_h*, os quais deveriam ter valores diferentes, mas possuem valores iguais. Isso é mostrado na figura 6.30.

Figura 6.30 – Mensagem descartada do *F-Host* para o *F-Device*

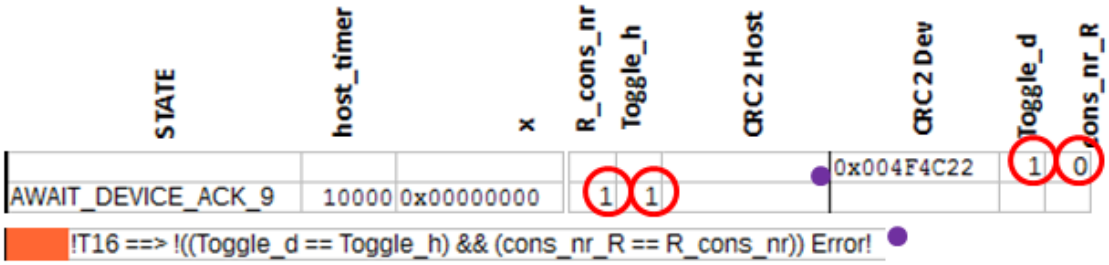


Fonte: Planilha Gerada com os Resultados do Teste de Sequência Incorreta.

Já o *ack* enviado pelo *F-Device* para o *F-Host* será descartado pela condição T16,

onde *R_cons_nr* é diferente do *cons_nr_R*, como pode ser visto na figura 6.31.

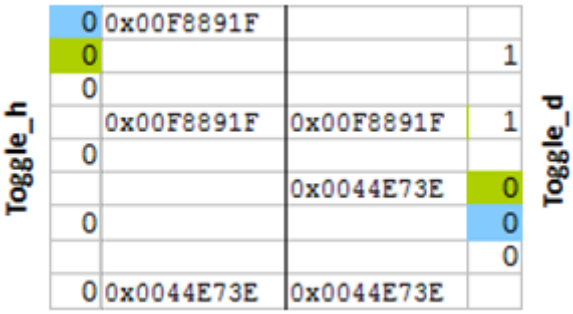
Figura 6.31 – *Ack* do *F-Device* descartado pelo *F-Host*



Fonte: Planilha Gerada com os Resultados do Teste de Sequência Incorreta.

Depois disso, os dois dispositivos seguem trocando mensagens porque o *Toggle_h* e o *Toggle_d* estão sincronizados, ou seja, *Toggle_h == Toggle_d* quando o *F-Host* recebe um *ack* do *F-Device* e *Toggle_h >< Toggle_d* quando o *F-Device* recebe uma mensagem do *F-Host*. Essa condição é apresentada na figura 6.32.

Figura 6.32 – *Toggle Bits* sincronizados entre os dois dispositivos *PROFIsafe*



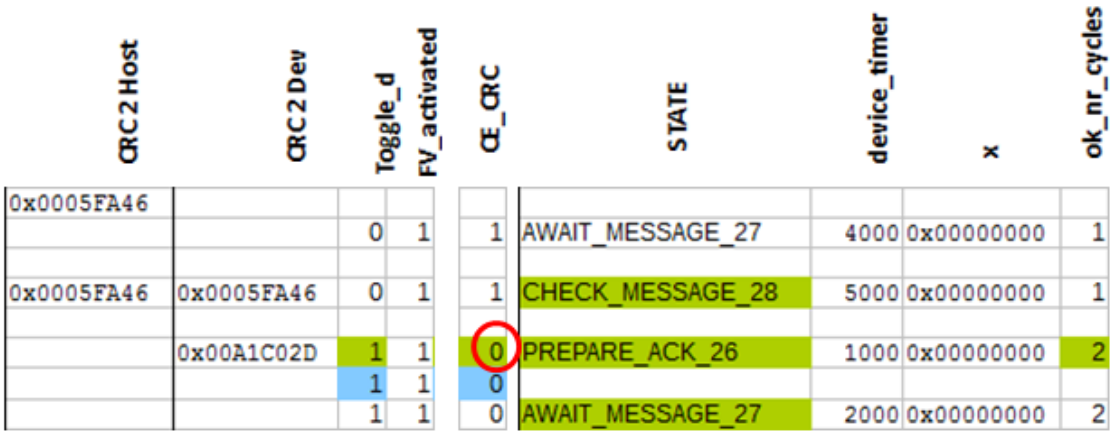
Fonte: Planilha Gerada com os Resultados do Teste de Sequência Incorreta.

Após duas mensagens recebidas com sucesso, o *CE_CRC* do *F-Device* recebe o valor “0”, assim o *F-Host* reconhece que não há mais falhas informadas pelo *F-Device*, alterando o valor da sua variável *not_faults* para “1”. Isso é mostrado na figura 6.33.

Após três recebimentos normais (ocasionando o incremento da variável *ok_nr_cycles* no *F-Device* para o valor “3”), o *F-Device* retorna ao ciclo normal (sem falhas) dos estados 23, 24 e 25.

No quarto recebimento normal das mensagens enviadas pelo *F-Host*, a variável *ok_nr_cycles* recebe o valor “4”, sendo a condição necessária para o *Status Bit FV_activated* receber o valor “0”, como pode ser visto na figura 6.34. Contudo, o *F-Device* vai seguir trabalhando com valores seguros de saída (*FVo*), caso ele seja um dispositivo de saída, até que ele receba do *F-Host* o *Control Bit activate_FV* com o valor “0”.

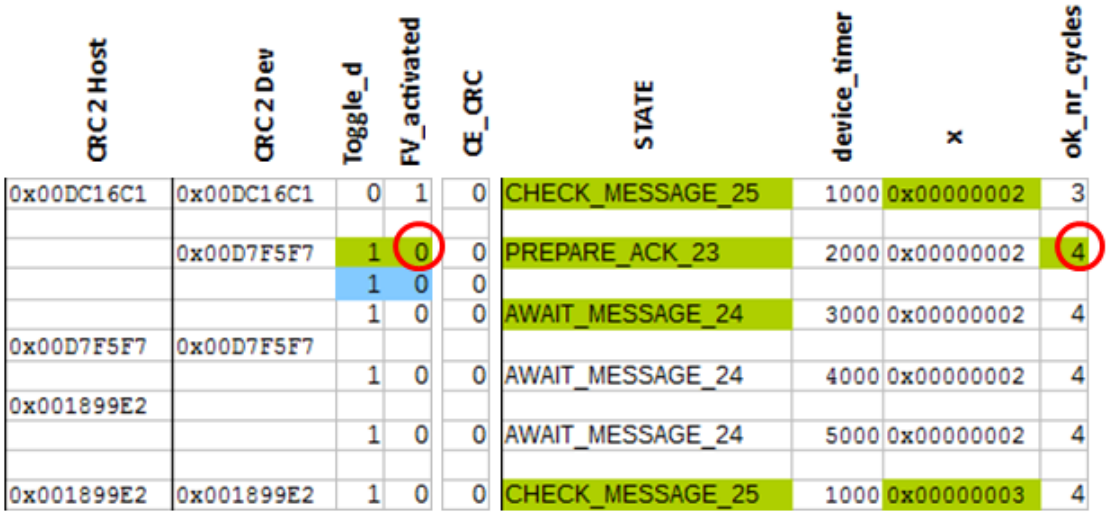
Figura 6.33 – Após o recebimento de duas mensagens sem falhas do *F-Host*, o *CE_CRC* do *F-Device* não reporta mais falhas ao *F-Host*



CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	CE_CRC	STATE	device_timer	x	ok_nr_cycles
0x0005FA46								
		0	1	1	AWAIT_MESSAGE_27	4000	0x00000000	1
0x0005FA46	0x0005FA46	0	1	1	CHECK_MESSAGE_28	5000	0x00000000	1
	0x00A1C02D	1	1	0	PREPARE_ACK_26	1000	0x00000000	2
		1	1	0				
		1	1	0	AWAIT_MESSAGE_27	2000	0x00000000	2

Fonte: Planilha Gerada com os Resultados do Teste de Sequência Incorreta.

Figura 6.34 – *F-Device* retorna ao estado normal de operação e depois desliga o *Status Bit FV_activated* que será enviado ao *F-Host*

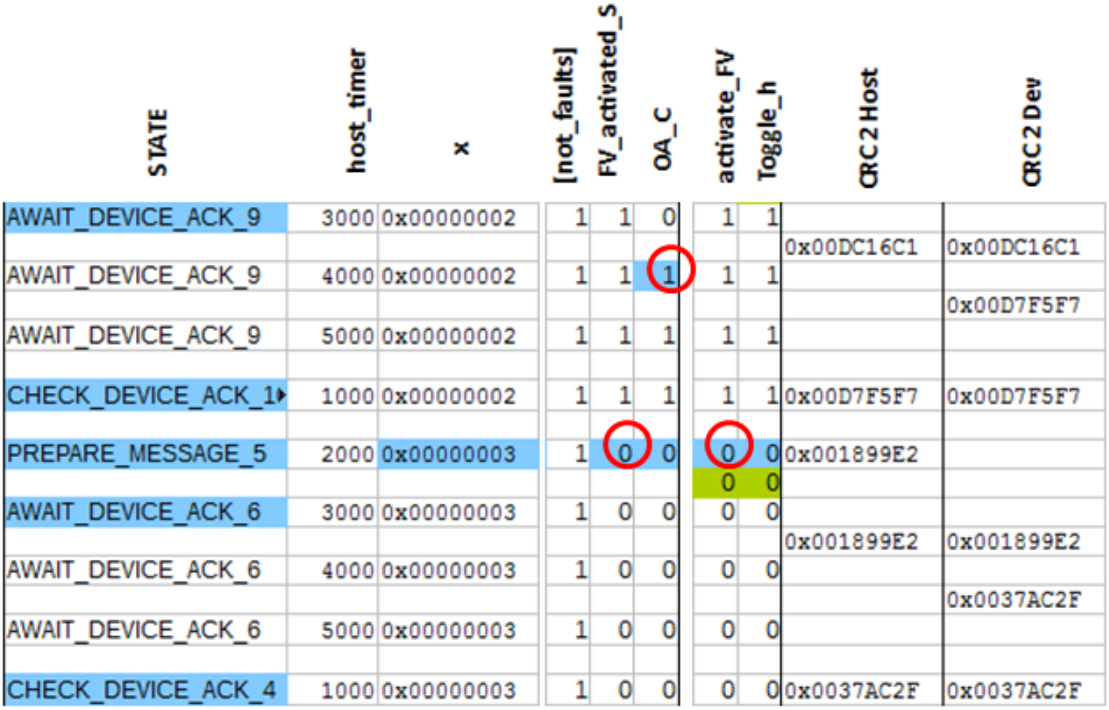


CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	CE_CRC	STATE	device_timer	x	ok_nr_cycles
0x00DC16C1	0x00DC16C1	0	1	0	CHECK_MESSAGE_25	1000	0x00000002	3
	0x00D7F5F7	1	0	0	PREPARE_ACK_23	2000	0x00000002	4
		1	0	0				
		1	0	0	AWAIT_MESSAGE_24	3000	0x00000002	4
0x00D7F5F7	0x00D7F5F7							
		1	0	0	AWAIT_MESSAGE_24	4000	0x00000002	4
0x001899E2								
		1	0	0	AWAIT_MESSAGE_24	5000	0x00000002	4
0x001899E2	0x001899E2	1	0	0	CHECK_MESSAGE_25	1000	0x00000003	4

Fonte: Planilha Gerada com os Resultados do Teste de Sequência Incorreta.

Quando o *F-Host* receber o *ack* do *F-Device*, ele vai atribuir o valor “0” para o *Control Bit activate_FV* e para a variável *FV_activated_S* porque o *Status Bit FV_activated* tem valor “0”. A variável *FV_activated_S* com o valor “0” avisa para a aplicação segura que o *F-Host driver* vai enviar dados de operação normal (*PVi*) do *F-Device* para ela se ele for um dispositivo de entrada ou que o *F-Device* vai receber dados normais de operação (*PVo*) do *F-Host driver* se ele for um dispositivo de saída. Assim que o operador ativar o *OA_C* (*Operator Acknowledgement*) na aplicação do *F-Host*, ele retorna ao ciclo normal de operação dos estados 5, 6 e 4, como mostra a figura 6.35.

Figura 6.35 – *F-Host* desliga os *bits FV_activated_S* e *activated_FV* e retorna ao seu estado normal de operação



STATE	host_timer	x	[not_faults]	FV_activated_S	OA_C	activate_FV	Toggle_h	CRC 2 Host	CRC 2 Dev
AWAIT_DEVICE_ACK_9	3000	0x00000002	1	1	0	1	1		
								0x00DC16C1	0x00DC16C1
AWAIT_DEVICE_ACK_9	4000	0x00000002	1	1	1	1	1		
									0x00D7F5F7
AWAIT_DEVICE_ACK_9	5000	0x00000002	1	1	1	1	1		
CHECK_DEVICE_ACK_1	1000	0x00000002	1	1	1	1	1	0x00D7F5F7	0x00D7F5F7
PREPARE_MESSAGE_5	2000	0x00000003	1	0	0	0	0	0x001899E2	
						0	0		
AWAIT_DEVICE_ACK_6	3000	0x00000003	1	0	0	0	0		
								0x001899E2	0x001899E2
AWAIT_DEVICE_ACK_6	4000	0x00000003	1	0	0	0	0		
									0x0037AC2F
AWAIT_DEVICE_ACK_6	5000	0x00000003	1	0	0	0	0		
CHECK_DEVICE_ACK_4	1000	0x00000003	1	0	0	0	0	0x0037AC2F	0x0037AC2F

Fonte: Planilha Gerada com os Resultados do Teste de Sequência Incorreta.

No instante em que o *F-Host* enviar uma nova mensagem, o *F-Device* vai receber o *Control Bit activate_FV* com o valor “0”, e assim ele volta a entregar dados normais de operação (*PVo*) para a aplicação do *F-Device* se ele for um dispositivo de saída.

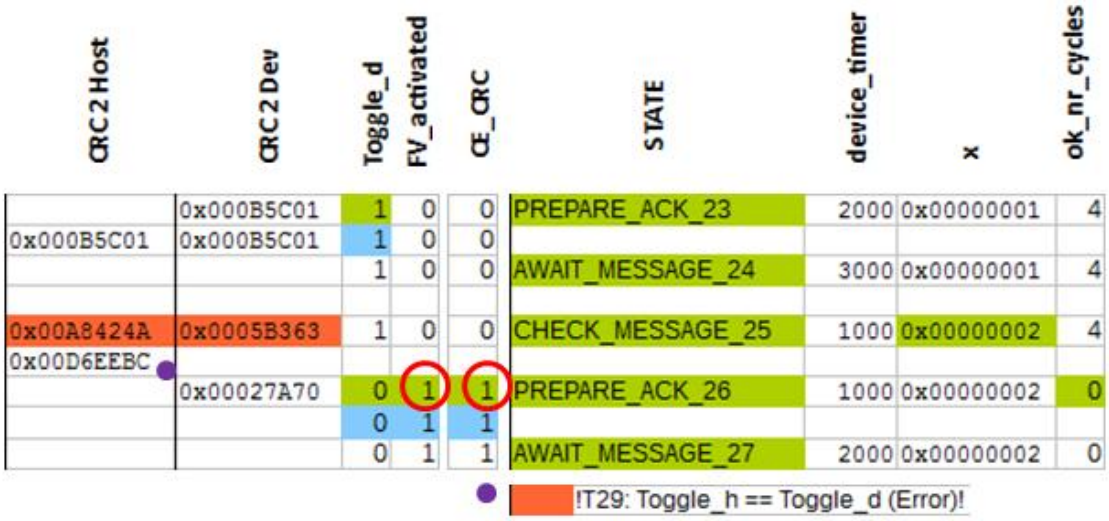
6.3.6 Teste de Inserção de Mensagens

Nesse teste, a mensagem escolhida e que foi anteriormente copiada e modificada para simular um pacote de fora da comunicação entre o *F-Host* e o *F-Device*, é inserida na comunicação no sentido do *F-Host* para o *F-Device*. Ela gera um erro de *CRC* no *F-Device* porque ela não faz parte da comunicação entre os dispositivos, possuindo um valor de *CRC2* com base em parâmetros seguros não utilizados pelo par *F-Host* e *F-Device*.

Ao detectar o erro, o *F-Device* ativa os *Status Bits CE_CRC* e *FV_activated*, passa a trabalhar com valores seguros para os dados de saída (*FVo*) que são entregues à aplicação do *F-Device* se ele for um dispositivo de saída e vai para o ciclo de estados 26, 27 e 28 para tratamento de falha. Nesse estado o *F-Device* descarta a primeira mensagem enviada pelo *F-Host* logo após a chegada da mensagem inserida porque o *Toggle_h* e o *Toggle_d* estão fora

de sintonia (*Toggle_h == Toggle_d*). A figura 6.36 ilustra essa situação.

Figura 6.36 – O *F-Device* detecta a mensagem inserida através de erro de *CRC*



CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	CE_CRC	STATE	device_timer	x	ok_nr_cycles
	0x000B5C01	1	0	0	PREPARE_ACK_23	2000	0x00000001	4
0x000B5C01	0x000B5C01	1	0	0				
		1	0	0	AWAIT_MESSAGE_24	3000	0x00000001	4
0x00A8424A	0x0005B363	1	0	0	CHECK_MESSAGE_25	1000	0x00000002	4
0x00D6EEBC								
	0x00027A70	0	1	1	PREPARE_ACK_26	1000	0x00000002	0
		0	1	1				
		0	1	1	AWAIT_MESSAGE_27	2000	0x00000002	0

Fonte: Planilha Gerada com os Resultados do Teste de Inserção de Mensagem.

O *F-Host* vai receber um *ack* do *F-Device* com os *Status Bit CE_CRC="1"* indicando que o *F-Device* detectou um erro de *CRC e FV_activated="1"* sinalizando que o *F-Device* está em um estado seguro de operação, trabalhando com dados seguros se ele for um dispositivo de saída. Então o *F-Host* atribui o valor "1" para a variável *FV_activated_S* e para o *Control Bit activate_FV* e ele entra no ciclo de estados 8, 9 e 10 para tratamento de falha, como é mostrado na figura 6.37.

Figura 6.37 – A variável *FV_activated_S* e o *Control Bit activate_FV* recebem o valor "1" após aviso de erro de *CRC* pelo *F-Device*

STATE	host_timer	x	[not_faults]	FV_activated_S	activate_FV	Toggle_h	CRC 2 Host	CRC 2 Dev
AWAIT_DEVICE_ACK_6	3000	0x00000002	1	0	0	0		
					0	0		
CHECK_DEVICE_ACK_4	1000	0x00000002	0	0	0	0	0x00027A70	0x00027A70
PREPARE_MESSAGE_8	2000	0x00000000	0	1	1	1	0x00C2E24C	
					1	1		
AWAIT_DEVICE_ACK_9	3000	0x00000000	0	1	1	1		

Fonte: Planilha Gerada com os Resultados do Teste de Inserção de Mensagem.

A variável *FV_activated_S* com o valor "1" vai sinalizar para a aplicação segura que o *F-Host driver* está enviando dados seguros (*FVo*) para o *F-Device* se ele for um atuador ou

que ela está recebendo valores seguros do *F-Host driver* (*FVi*) se o *F-Device* for um sensor.

A seguir, como mostra a figura 6.38, os dois dispositivos seguem trocando mensagens porque o *Toggle_h* e o *Toggle_d* estão sincronizados, ou seja, *Toggle_h == Toggle_d* quando o *F-Host* recebe um *ack* do *F-Device* e *Toggle_h >< Toggle_d* quando o *F-Device* recebe uma mensagem do *F-Host*.

Figura 6.38 – *Toggle Bits* sincronizados entre os dois dispositivos *PROFIsafe*

Toggle_h				Toggle_d
		0x00027A70	0	
0			0	
0			0	
0	0x00027A70	0x00027A70		
			0	
1	0x00C2E24C			
1			0	
1				
	0x00C2E24C	0x00C2E24C	0	

Fonte: Planilha Gerada com os Resultados do Teste de Inserção de Mensagem.

Após duas mensagens recebidas com sucesso, como mostra a figura 6.39, o *CE_CRC* do *F-Device* recebe o valor "0", assim o *F-Host* reconhece que não há mais falhas informadas pelo *F-Device*, alterando o valor da sua variável *not_faults* para "1".

Figura 6.39 – Após o recebimento de duas mensagens sem falhas do *F-Host*, o *CE_CRC* do *F-Device* não reporta mais falhas ao *F-Host*

CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	CE_CRC	STATE	device_timer	x	ok_nr_cycles
0x002BD4C0								
		1	1	1	AWAIT_MESSAGE_27	4000	0x00000000	1
0x002BD4C0	0x002BD4C0	1	1	1	CHECK_MESSAGE_28	5000	0x00000000	1
	0x008FEEAB	0	1	0	PREPARE_ACK_26	1000	0x00000000	2
		0	1	0				
		0	1	0	AWAIT_MESSAGE_27	2000	0x00000000	2

Fonte: Planilha Gerada com os Resultados do Teste de Inserção de Mensagem.

Após três recebimentos normais (a variável *ok_nr_cycles* no *F-Device* fica com o valor "3"), o *F-Device* retorna ao seu ciclo normal de funcionamento (sem falhas), representado pelos estados 23, 24 e 25. Na quarta mensagem enviada sem falha pelo *F-Host*, a variável *ok_nr_cycles* recebe o valor "4", sendo a condição necessária para o *Status Bit*

FV_activated receber o valor “0”. Esse comportamento é apresentado na figura 6.40, a qual pode ser visualizada a seguir. Contudo, o *F-Device* vai seguir trabalhando com valores seguros de saída (*FVo*), caso ele seja um dispositivo de saída, até que ele receba do *F-Host* o *Control Bit activate_FV* com o valor “0”.

Figura 6.40 – *F-Device* retorna ao estado normal de operação e depois desliga o *Status Bit FV_activated* que será enviado ao *F-Host*

CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	CE_CRC	STATE	device_timer	x	ok_nr_cycles
0x00DAB3ED	0x00DAB3ED	1	1	0	CHECK_MESSAGE_25	1000	0x00000002	3
	0x00D150DB	0	0	0	PREPARE_ACK_23	2000	0x00000002	4
		0	0	0				
		0	0	0	AWAIT_MESSAGE_24	3000	0x00000002	4
0x00D150DB	0x00D150DB							
		0	0	0	AWAIT_MESSAGE_24	4000	0x00000002	4
0x00D924C4								
		0	0	0	AWAIT_MESSAGE_24	5000	0x00000002	4
0x00D924C4	0x00D924C4	0	0	0	CHECK_MESSAGE_25	1000	0x00000003	4

Fonte: Planilha Gerada com os Resultados do Teste de Inserção de Mensagem.

Figura 6.41 – *F-Host* desliga os *bits FV_activated_S* e *activated_FV* e retorna ao seu estado normal de operação

STATE	host_timer	x	[not_faults]	FV_activated_S	OA_C	activate_FV	Toggle_h	CRC 2 Host	CRC 2 Dev
AWAIT_DEVICE_ACK_9	3000	0x00000002	1	1	0	1	0		
								0x00DAB3ED	0x00DAB3ED
AWAIT_DEVICE_ACK_9	4000	0x00000002	1	1	0	1	0		
									0x00D150DB
AWAIT_DEVICE_ACK_9	5000	0x00000002	1	1	1	1	0		
CHECK_DEVICE_ACK_1	1000	0x00000002	1	1	1	1	0	0x00D150DB	0x00D150DB
PREPARE_MESSAGE_5	2000	0x00000003	1	0	0	0	1	0x00D924C4	
						0	1		
AWAIT_DEVICE_ACK_6	3000	0x00000003	1	0	0	0	1		
								0x00D924C4	0x00D924C4
AWAIT_DEVICE_ACK_6	4000	0x00000003	1	0	0	0	1		
									0x00F61109
AWAIT_DEVICE_ACK_6	5000	0x00000003	1	0	0	0	1		
CHECK_DEVICE_ACK_4	1000	0x00000003	1	0	0	0	1	0x00F61109	0x00F61109

Fonte: Planilha Gerada com os Resultados do Teste de Inserção de Mensagem.

Quando o *F-Host* receber o *ack* do *F-Device*, ele vai atribuir o valor “0” para o *Control Bit activate_FV* e para a variável *FV_activated_S* porque o *Status Bit FV_activated* tem valor “0”. A variável *FV_activated_S* com o valor “0” avisa a aplicação segura no *F-Host* que o *F-Host driver* vai enviar dados de operação normal (*PVi*) do *F-Device* para ela se ele for um dispositivo de entrada ou que o *F-Device* vai receber dados normais de operação (*PVo*) do *F-Host driver* se ele for um dispositivo de saída.

Assim que o operador ativar o *OA_C* (*Operator Acknowledgement*) na aplicação do *F-Host*, ele retorna ao ciclo normal de operação dos estados 5, 6 e 4, como é ilustrado na figura 6.41. No momento em que o *F-Host* enviar uma nova mensagem, o *F-Device* vai receber o *Control Bit activate_FV* com o valor “0”, e assim ele volta a entregar dados normais de operação (*PVo*) para a aplicação do *F-Device* se ele for um dispositivo de saída.

6.3.7 Teste de Descarte de Pacotes

Nesse experimento, uma mensagem escolhida na direção *F-Host* → *F-Device* é descartada. O *F-Device* detecta a perda da mensagem através da contagem de *timeout*, como pode ser visto na figura 6.42.

Figura 6.42 – Detecção de erro de *Timeout* pelo *F-Device*

CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	WD_timeout	STATE	device_timer	x	ok_nr_cycles
0x001013D9								
		1	0	0	AWAIT_MESSAGE_24	5000	0x00FFFFFC	4
		1	0	0	AWAIT_MESSAGE_24	6000	0x00FFFFFC	4
		1	0	0	AWAIT_MESSAGE_24	7000	0x00FFFFFC	4
		1	0	0	AWAIT_MESSAGE_24	8000	0x00FFFFFC	4
		1	0	0	AWAIT_MESSAGE_24	9000	0x00FFFFFC	4
		1	0	0	AWAIT_MESSAGE_24	10000	0x00FFFFFC	4
		1	0	0	AWAIT_MESSAGE_24	11000	0x00FFFFFC	4
	0x00EFD758	1	1	1	PREPARE_ACK_26	1000	0x00FFFFFC	0
		1	1	1				
		1	1	1	AWAIT_MESSAGE_27	2000	0x00FFFFFC	0

Fonte: Planilha Gerada com os Resultados do Teste de Perda de Pacotes.

Ele então ativa os *Status Bits WD_timeout* e *FV_activated* e também passa a trabalhar com valores seguros para os dados de saída (*FVo*) que são entregues à aplicação do *F-Device*, caso ele seja um dispositivo de saída. Ao detectar o não recebimento da mensagem, o *F-Device* entra no ciclo de estados 26, 27 e 28 para tratamento de falhas.

Devido à perda da mensagem, o *Toggle_d* do *F-Device* não é incrementado. Dessa forma, o *ack* enviado pelo *F-Device* será recusado pelo *F-Host*, por erro de *CRC*. O *F-Host* então atribui o valor "1" para a variável *FV_activated_S* e para o *Control Bit activate_FV* e ele entra no ciclo de estados 8, 9 e 10 para tratamento de falha, como apresentado na figura 6.43. A variável *FV_activated_S* com o valor "1" vai indicar para a aplicação segura que o *F-Host driver* está enviando dados seguros (*FVo*) para o *F-Device* se ele for um atuador ou que ela vai receber valores seguros do *F-Host driver* (*FVi*) se o *F-Device* for um sensor.

Figura 6.43 – O *F-Host* vai para o estado 8 para tratar a falha de erro de *CRC* detectada

STATE	host_timer	x	[not_faults]	FV_activated_S	activate_FV	Toggle_h	CRC 2 Host	CRC 2 Dev
AWAIT_DEVICE_ACK_6	9000	0x00FFFFFD	1	0	0	0		
								0x00EFD758
AWAIT_DEVICE_ACK_6	10000	0x00FFFFFD	1	0	0	0		
CHECK_DEVICE_ACK_7	11000	0x00FFFFFD	0	0	0	0	0x00E6B80C	0x00EFD758
PREPARE_MESSAGE_8	1000	0x00000000	0	1	1	1	0x0011BF93	
					1	1		
AWAIT_DEVICE_ACK_9	2000	0x00000000	0	1	1	1		

Fonte: Planilha Gerada com os Resultados do Teste de Perda de Pacotes.

A próxima mensagem enviada do *F-Host* para o *F-Device* e correspondente *ack* enviado do *F-Device* para o *F-Host* também são descartados.

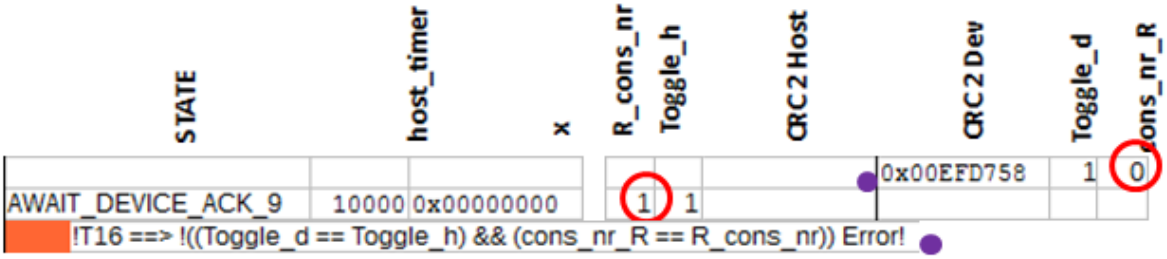
Figura 6.44 – *F-Device* não aceita mensagem do *F-Host* por falta de sincronia no *Toggle_h*

Toggle_h	CRC 2 Host	CRC 2 Dev	Toggle_d	STATE	device_timer	x	ok_nr_cycles
1	0x0011BF93						
1			1	AWAIT_MESSAGE_27	4000	0x00FFFFFC	0
				!T29: Toggle_h == Toggle_d (Error)!			

Fonte: Planilha Gerada com os Resultados do Teste de Perda de Pacotes.

A mensagem enviada pelo *F-Host* é descartada pela falta de sincronia entre os *bits* do *Toggle_d* e do *Toggle_h* que deveriam ser diferentes, mas são iguais como ilustra a figura 6.44. Já o *ack* enviado pelo *F-Device* não é aceito pelo *F-Host* por que os sinais *R_cons_nr* e *cons_nr_R* são diferentes, como pode ser visualizado na figura 6.45.

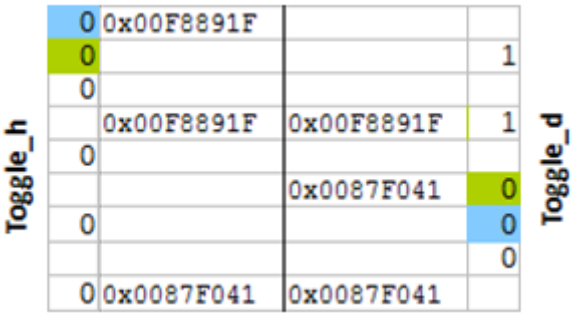
Figura 6.45 – *F-Host* não aceita mensagem do *F-Device* por que os sinais *R_cons_nr* e *cons_nr_R* são diferentes



Fonte: Planilha Gerada com os Resultados do Teste de Perda de Pacotes.

Já as próximas mensagens do *F-Host* para o *F-Device* e os *acks* do *F-Device* para o *F-Host* são aceitas, pois os *bits* do *Toggle_d* e do *Toggle_h* voltaram à sincronia normal. Ou seja, *Toggle_h == Toggle_d* quando o *F-Host* recebe um *ack* do *F-Device* e *Toggle_h >< Toggle_d* quando o *F-Device* recebe uma mensagem do *F-Host*. Essa situação pode ser visualizada na figura 6.46.

Figura 6.46 – *Toggle Bits* sincronizados entre os dois dispositivos *PROFIsafe*



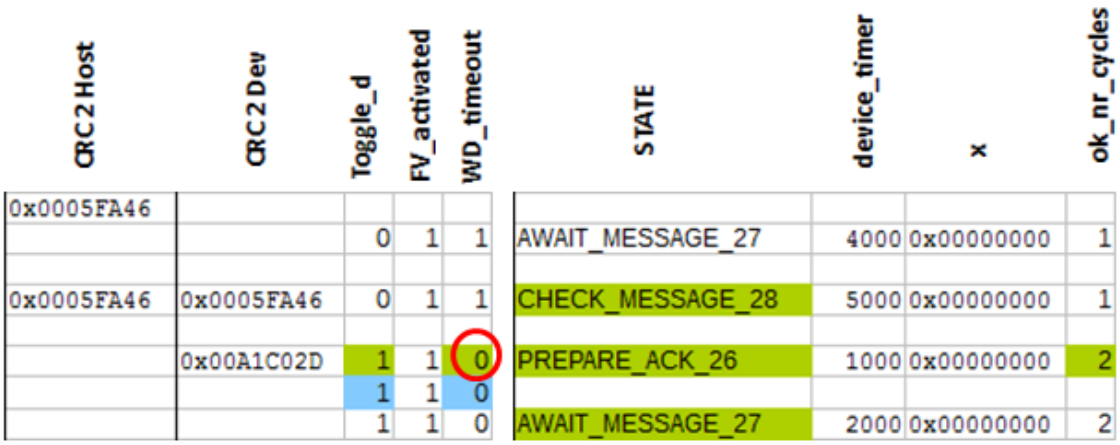
Fonte: Planilha Gerada com os Resultados do Teste de Perda de Pacotes.

Após duas mensagens recebidas com sucesso, como ilustrado na figura 6.47, o *WD_timeout* do *F-Device* recebe o valor "0", assim o *F-Host* reconhece que não há mais falhas informadas pelo *F-Device*, alterando o valor da sua variável *not_faults* para "1".

Após três recebimentos normais, ocasionando o incremento da variável *ok_nr_cycles* no *F-Device* para o valor "3", este dispositivo volta ao seu ciclo normal de funcionamento (sem falha) dos estados 23, 24 e 25.

No quarto recebimento normal das mensagens enviadas pelo *F-Host*, a variável *ok_nr_cycles* recebe o valor "4", sendo a condição necessária para o *Status Bit FV_activated* receber o valor "0". Essa situação é apresentada pela figura 6.48. Contudo, o *F-Device* vai seguir trabalhando com valores seguros de saída (*FVo*), caso ele seja um dispositivo de saída, até que ele receba do *F-Host* o *Control Bit activate_FV* com o valor "0".

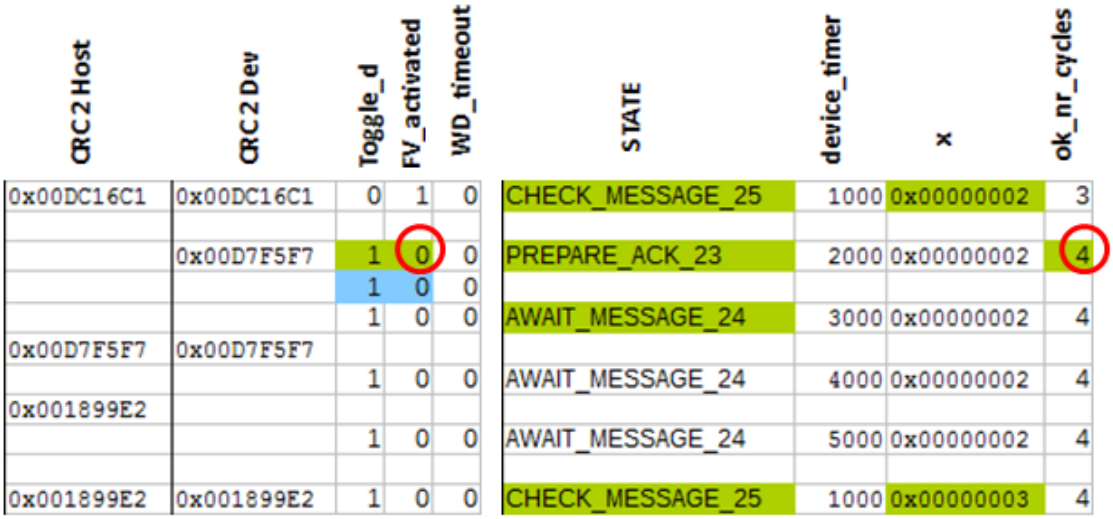
Figura 6.47 – Após o recebimento de duas mensagens sem falhas do *F-Host*, o *WD_timeout* do *F-Device* não reporta mais falhas ao *F-Host*



CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	WD_timeout	STATE	device_timer	x	ok_nr_cycles
0x0005FA46								
		0	1	1	AWAIT_MESSAGE_27	4000	0x00000000	1
0x0005FA46	0x0005FA46	0	1	1	CHECK_MESSAGE_28	5000	0x00000000	1
	0x00A1C02D	1	1	0	PREPARE_ACK_26	1000	0x00000000	2
		1	1	0				
		1	1	0	AWAIT_MESSAGE_27	2000	0x00000000	2

Fonte: Planilha Gerada com os Resultados do Teste de Perda de Pacotes.

Figura 6.48 – *F-Device* retorna ao estado normal de operação e depois desliga o *Status Bit FV_activated* que será enviado ao *F-Host*



CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	WD_timeout	STATE	device_timer	x	ok_nr_cycles
0x00DC16C1	0x00DC16C1	0	1	0	CHECK_MESSAGE_25	1000	0x00000002	3
	0x00D7F5F7	1	0	0	PREPARE_ACK_23	2000	0x00000002	4
		1	0	0				
		1	0	0	AWAIT_MESSAGE_24	3000	0x00000002	4
0x00D7F5F7	0x00D7F5F7							
		1	0	0	AWAIT_MESSAGE_24	4000	0x00000002	4
0x001899E2								
		1	0	0	AWAIT_MESSAGE_24	5000	0x00000002	4
0x001899E2	0x001899E2	1	0	0	CHECK_MESSAGE_25	1000	0x00000003	4

Fonte: Planilha Gerada com os Resultados do Teste de Perda de Pacotes.

Quando o *F-Host* receber o *ack* do *F-Device*, ele vai atribuir o valor "0" para o *Control Bit activate_FV* e para a variável *FV_activated_S* porque o *Status Bit FV_activated*

tem valor “0”. A variável *FV_activated_S* com o valor “0” indica para a aplicação segura que o *F-Host driver* vai enviar dados de operação normal (*PVi*) do *F-Device* para ela se ele for um dispositivo de entrada ou que o *F-Device* vai receber dados normais de operação (*PVo*) do *F-Host driver* se ele for um dispositivo de saída.

Assim que o operador ativar o *OA_C* (*Operator Acknowledgement*) na aplicação do *F-Host*, ele retorna ao ciclo normal de operação dos estados 5, 6 e 4, como mostra a figura 6.49.

Figura 6.49 – *F-Host* desliga os *bits FV_activated_S* e *activated_FV* e retorna ao seu estado normal de operação

STATE	host_timer	x	[not_faults]	FV_activated_S	OA_C	activate_FV	Toggle_h	CRC 2 Host	CRC 2 Dev
AWAIT_DEVICE_ACK_9	3000	0x00000002	1	1	0	1	1		
								0x00DC16C1	0x00DC16C1
AWAIT_DEVICE_ACK_9	4000	0x00000002	1	1	1	1	1		
									0x00D7F5F7
AWAIT_DEVICE_ACK_9	5000	0x00000002	1	1	1	1	1		
CHECK_DEVICE_ACK_1	1000	0x00000002	1	1	1	1	1	0x00D7F5F7	0x00D7F5F7
PREPARE_MESSAGE_5	2000	0x00000003	1	0	0	0	0	0x001899E2	
						0	0		
AWAIT_DEVICE_ACK_6	3000	0x00000003	1	0	0	0	0		
								0x001899E2	0x001899E2
AWAIT_DEVICE_ACK_6	4000	0x00000003	1	0	0	0	0		
									0x0037AC2F
AWAIT_DEVICE_ACK_6	5000	0x00000003	1	0	0	0	0		
CHECK_DEVICE_ACK_4	1000	0x00000003	1	0	0	0	0	0x0037AC2F	0x0037AC2F

Fonte: Planilha Gerada com os Resultados do Teste de Perda de Pacotes.

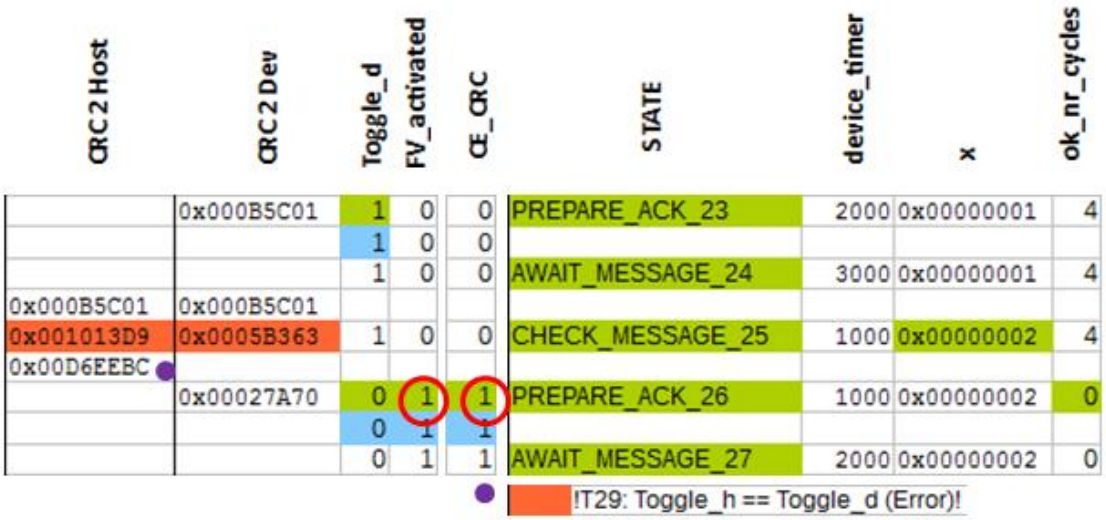
No momento em que o *F-Host* enviar uma nova mensagem, o *F-Device* vai receber o *Control Bit activate_FV* com o valor “0”, e assim ele volta a entregar dados normais de operação (*PVo*) para a aplicação do *F-Device* se ele for um dispositivo de saída.

6.3.8 Teste de Repetição de Mensagens

Nesse teste, a mensagem que foi escolhida e duplicada de acordo com a especificação do cenário de falhas, é inserida na comunicação no sentido do *F-Host* para o *F-Device*. Essa mensagem causa um erro de *CRC* no *F-Device* porque ela está fora do contexto da comunicação atual, possuindo um valor antigo do *CRC2*, correspondente a dados e parâmetros seguros obsoletos. Ao detectar o erro, o *F-Device* ativa os *Status Bits CE_CRC* e

FV_activated e também passa a trabalhar com valores seguros para os dados de saída (*FVo*) que são entregues à aplicação do *F-Device*, caso ele seja um dispositivo de saída. Em seguida ele vai para o ciclo de estados 26, 27 e 28 para tratamento de falha. Nesse estado ele não aceita a primeira mensagem enviada pelo *F-Host* logo após a chegada da mensagem duplicada porque o *Toggle_h* e o *Toggle_d* estão fora de sintonia (*Toggle_h == Toggle_d*). Essa situação é mostrada na figura 6.50.

Figura 6.50 – O *F-Device* detecta a mensagem duplicada através de erro de *CRC*

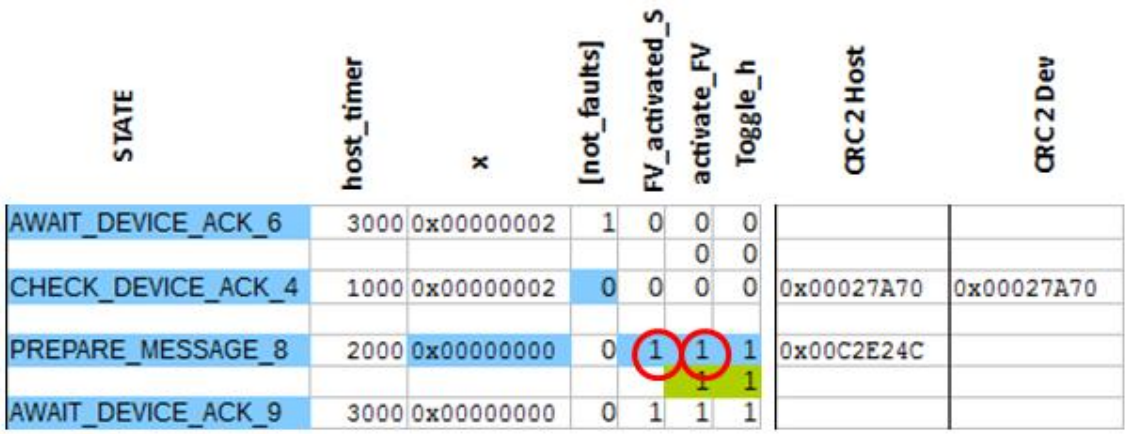


CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	CE_CRC	STATE	device_timer	x	ok_nr_cycles
	0x000B5C01	1	0	0	PREPARE_ACK_23	2000	0x00000001	4
		1	0	0				
		1	0	0	AWAIT_MESSAGE_24	3000	0x00000001	4
0x000B5C01	0x000B5C01							
0x001013D9	0x0005B363	1	0	0	CHECK_MESSAGE_25	1000	0x00000002	4
0x00D6EEBC								
	0x00027A70	0	1	1	PREPARE_ACK_26	1000	0x00000002	0
		0	1	1				
		0	1	1	AWAIT_MESSAGE_27	2000	0x00000002	0
					!T29: Toggle_h == Toggle_d (Error)!			

Fonte: Planilha Gerada com os Resultados do Teste de Repetição de Pacotes.

O *F-Host* vai receber um *ack* do *F-Device* com os *Status Bit CE_CRC="1"* sinalizando que o *F-Device* detectou um erro de *CRC e FV_activated="1"* indicando que o *F-Device* está em um estado seguro de operação, trabalhando com dados seguros se ele for um dispositivo de saída.

Figura 6.51 – A variável *FV_activated_S* e *o Control Bit activate_FV* recebem o valor "1" após aviso de erro de *CRC* pelo *F-Device*



STATE	host_timer	x	[not_faults]	FV_activated_S	activate_FV	Toggle_h	CRC 2 Host	CRC 2 Dev
AWAIT_DEVICE_ACK_6	3000	0x00000002	1	0	0	0		
					0	0		
CHECK_DEVICE_ACK_4	1000	0x00000002	0	0	0	0	0x00027A70	0x00027A70
PREPARE_MESSAGE_8	2000	0x00000000	0	1	1	1	0x00C2E24C	
					1	1		
AWAIT_DEVICE_ACK_9	3000	0x00000000	0	1	1	1		

Fonte: Planilha Gerada com os Resultados do Teste de Repetição de Pacotes.

Então o *F-Host* atribui o valor “1” para a variável *FV_activated_S* e para o *Control Bit activate_FV* e ele entra no ciclo de estados 8, 9 e 10 para tratamento de falha (figura 6.51).

A variável *FV_activated_S* com o valor “1” sinaliza para a aplicação segura que o *F-Host driver* está enviando dados seguros (*FVo*) para o *F-Device* se ele for um atuador ou que ela está recebendo valores seguros do *F-Host driver* (*FVi*) se o *F-Device* for um sensor, por exemplo.

A seguir, conforme ilustra a figura 6.52, os dois dispositivos seguem trocando mensagens porque o *Toggle_h* e o *Toggle_d* estão sincronizados, ou seja, *Toggle_h == Toggle_d* quando o *F-Host* recebe um *ack* do *F-Device* e *Toggle_h >< Toggle_d* quando o *F-Device* recebe uma mensagem do *F-Host*.

Figura 6.52 – *Toggle Bits* sincronizados entre os dois dispositivos *PROFIsafe*

Toggle_h				Toggle_d
		0x00027A70	0	
0			0	
0			0	
0	0x00027A70	0x00027A70		
			0	
1	0x00C2E24C			
1			0	
1				
	0x00C2E24C	0x00C2E24C	0	

Fonte: Planilha Gerada com os Resultados do Teste de Repetição de Pacotes.

Após duas mensagens recebidas com sucesso, como apresentado na figura 6.53, o *CE_CRC* do *F-Device* recebe o valor “0”, assim o *F-Host* reconhece que não há mais falhas informadas pelo *F-Device*, alterando o valor da sua variável *not_faults* para “1”.

Figura 6.53 – Após o recebimento de duas mensagens sem falhas do *F-Host*, o *CE_CRC* do *F-Device* não reporta mais falhas ao *F-Host*

CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	CE_CRC	STATE	device_timer	x	ok_nr_cycles
0x002BD4C0								
		1	1	1	AWAIT_MESSAGE_27	4000	0x00000000	1
0x002BD4C0	0x002BD4C0	1	1	1	CHECK_MESSAGE_28	5000	0x00000000	1
	0x008FEEAB	0	1	0	PREPARE_ACK_26	1000	0x00000000	2
		0	1	0				
		0	1	0	AWAIT_MESSAGE_27	2000	0x00000000	2

Fonte: Planilha Gerada com os Resultados do Teste de Repetição de Pacotes.

Após três recebimentos normais (ocasionando o incremento da variável *ok_nr_cycles* no *F-Device* para o valor "3"), o *F-Device* retorna ao seu ciclo normal de funcionamento (sem falhas) dos estados 23, 24, e 25.

No quarto recebimento normal das mensagens enviadas pelo *F-Host*, a variável *ok_nr_cycles* recebe o valor "4", sendo a condição necessária para o *Status Bit FV_activated* receber o valor "0". Essa situação é mostrada na figura 6.54.

Contudo, o *F-Device* vai seguir trabalhando com valores seguros de saída (*FVo*), se ele for um dispositivo de saída, até que ele receba do *F-Host* o *Control Bit activate_FV* com o valor "0".

Figura 6.54 – *F-Device* retorna ao estado normal de operação e depois desliga o *Status Bit FV_activated* que será enviado ao *F-Host*

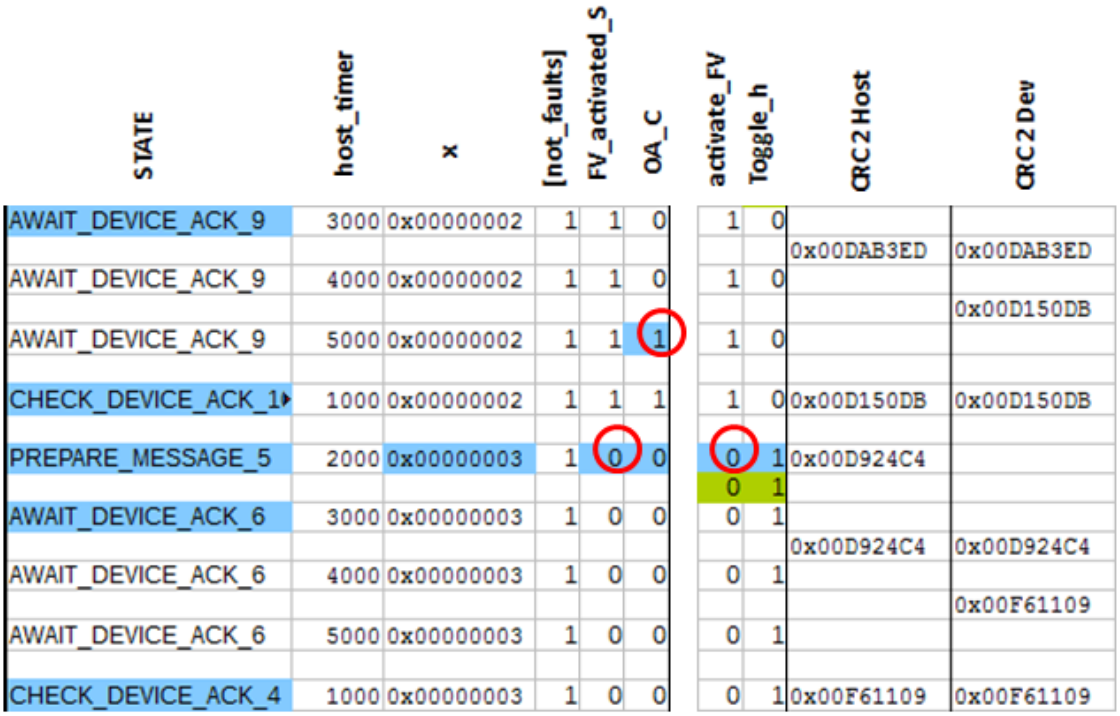
CRC 2 Host	CRC 2 Dev	Toggle_d	FV_activated	CE_CRC	STATE	device_timer	x	ok_nr_cycles
0x00DAB3ED	0x00DAB3ED	1	1	0	CHECK_MESSAGE_25	1000	0x00000002	3
	0x00D150DB	0	0	0	PREPARE_ACK_23	2000	0x00000002	4
		0	0	0				
		0	0	0	AWAIT_MESSAGE_24	3000	0x00000002	4
0x00D150DB	0x00D150DB							
		0	0	0	AWAIT_MESSAGE_24	4000	0x00000002	4
0x00D924C4								
		0	0	0	AWAIT_MESSAGE_24	5000	0x00000002	4
0x00D924C4	0x00D924C4	0	0	0	CHECK_MESSAGE_25	1000	0x00000003	4

Fonte: Planilha Gerada com os Resultados do Teste de Repetição de Pacotes.

Quando o *F-Host* receber o *ack* do *F-Device*, ele vai atribuir o valor "0" para o *Control Bit activate_FV* e para a variável *FV_activated_S* porque o *Status Bit FV_activated* tem valor "0". A variável *FV_activated_S* com o valor "0" vai indicar para a aplicação segura que o *F-Host driver* vai enviar dados de operação normal (*PVi*) do *F-Device* para ela se ele for um dispositivo de entrada ou que o *F-Device* vai receber dados normais de operação (*PVo*) do *F-Host driver* se ele for um dispositivo de saída.

Assim que o operador ativar o *OA_C* (*Operator Acknowledgement*) na aplicação do *F-Host*, ele retorna ao ciclo normal de operação dos estados 5, 6 e 4, como é ilustrada na figura 6.55. No instante em que o *F-Host* enviar uma nova mensagem, o *F-Device* vai receber o *Control Bit activate_FV* com o valor "0", e assim ele volta a entregar dados normais de operação (*PVo*) para a aplicação do *F-Device* se ele for um dispositivo de saída.

Figura 6.55 – *F-Host* desliga os *bits FV_activated_S* e *activated_FV* e retorna ao seu estado normal de operação



STATE	host_timer	x	[not_faults]	FV_activated_S	OA_C	activate_FV	Toggle_h	CRC 2 Host	CRC 2 Dev
AWAIT_DEVICE_ACK_9	3000	0x00000002	1	1	0	1	0		
								0x00DAB3ED	0x00DAB3ED
AWAIT_DEVICE_ACK_9	4000	0x00000002	1	1	0	1	0		
									0x00D150DB
AWAIT_DEVICE_ACK_9	5000	0x00000002	1	1	1	1	0		
CHECK_DEVICE_ACK_1	1000	0x00000002	1	1	1	1	0	0x00D150DB	0x00D150DB
PREPARE_MESSAGE_5	2000	0x00000003	1	0	0	0	1	0x00D924C4	
						0	1		
AWAIT_DEVICE_ACK_6	3000	0x00000003	1	0	0	0	1		
								0x00D924C4	0x00D924C4
AWAIT_DEVICE_ACK_6	4000	0x00000003	1	0	0	0	1		
									0x00F61109
AWAIT_DEVICE_ACK_6	5000	0x00000003	1	0	0	0	1		
CHECK_DEVICE_ACK_4	1000	0x00000003	1	0	0	0	1	0x00F61109	0x00F61109

Fonte: Planilha Gerada com os Resultados do Teste de Repetição de Pacotes.

6.4 Análise da Intrusividade Causada pelo Injetor

A intrusividade causada pelo injetor de falhas *FITT* corresponde ao tempo em que o pacote escolhido fica retido no injetor até que o processo de injeção de falhas seja concluído. Os tempos de captura e envio dos pacotes entre as interfaces de redes e o injetor de falhas são desconsiderados devido ao uso do *PF_RING*, que realiza essas operações instantaneamente.

A seguir, na tabela 6.1, é comparada a intrusividade estimada causada pelo injetor em cada um dos testes de injeção de falhas realizados com os dois computadores de *CPUs* diferentes utilizados nos experimentos. A intenção foi verificar se seria possível reduzir a intrusividade causada pelo injetor, ou seja, o tempo de execução das funções de injeção de falhas, com o uso de uma *CPU* com mais recursos computacionais como o *Core i7 3770* da *Intel* em relação ao *Phenom II 975* da *AMD*.

A análise dos valores da tabela mostrou que os tempos de intrusividade obtidos foram bem próximos para cada tipo de teste, com o computador com a *CPU Core i7* apresentando os tempos mais baixos na maioria dos experimentos. No entanto a diferença de tempo obtida para cada um dos testes não foi significativa, podendo-se considerar que os tempos estão na

mesma faixa de valores para cada classe de falhas.

Tabela 6.1 – Comparação dos tempos de intrusividade causados pelo injetor em duas configurações diferentes de computadores

	AMD Phenom II 975	Intel Core I7 3770
Endereçamento	313145 ns	274273 ns
Corrupção (bit)	200743 ns	194810 ns
Corrupção (byte)	243535 ns	181015 ns
Atraso	229727 ns	246795 ns
Sequência Incorreta	168208 ns	160532 ns
Inserção	146182 ns	186439 ns
Perda	120278 ns	114506 ns
Repetição	154611 ns	144584 ns

Fonte: Valores Retirados do Relatório de Injeção de Falhas.

O protocolo *PROFIsafe* especifica um tempo de *timeout* máximo de 65535 milissegundos entre o envio e o recebimento da próxima mensagem, sendo que na prática são utilizados geralmente valores na faixa de dezenas de milissegundos. Dessa forma, a intrusividade causada pelo injetor não pode ser dessa mesma ordem de magnitude, pois uma intrusividade temporal alta poderia ativar a detecção de *timeout* pela execução do injetor e não por uma falha em uma mensagem. Idealmente, a intrusividade deve ser na ordem de variações naturais no tempo de transmissão de pacotes na rede, ou seja, na faixa de microssegundos.

Nos experimentos realizados, os tempos de intrusividade registrados nos relatórios de injeção de falhas variaram de 114 microssegundos no teste de perda de pacotes (o menor) até 313 microssegundos no teste de erro de endereçamento (o maior), com uma pequena diferença entre as medidas de tempo do mesmo tipo de teste nos diferentes computadores.

Considerando o *timeout* de dispositivos reais na faixa de dezenas de milissegundos e a intrusividade temporal do injetor na faixa de centenas de microssegundos, a interferência temporal causada pelo *FITT* é muito pequena e dessa maneira, o injetor atende o importante requisito de não interferir com os resultados dos testes de injeção de falhas. Assim, o *FITT* pode ser usado para validar com segurança o funcionamento de implementações do protocolo *PROFIsafe*.

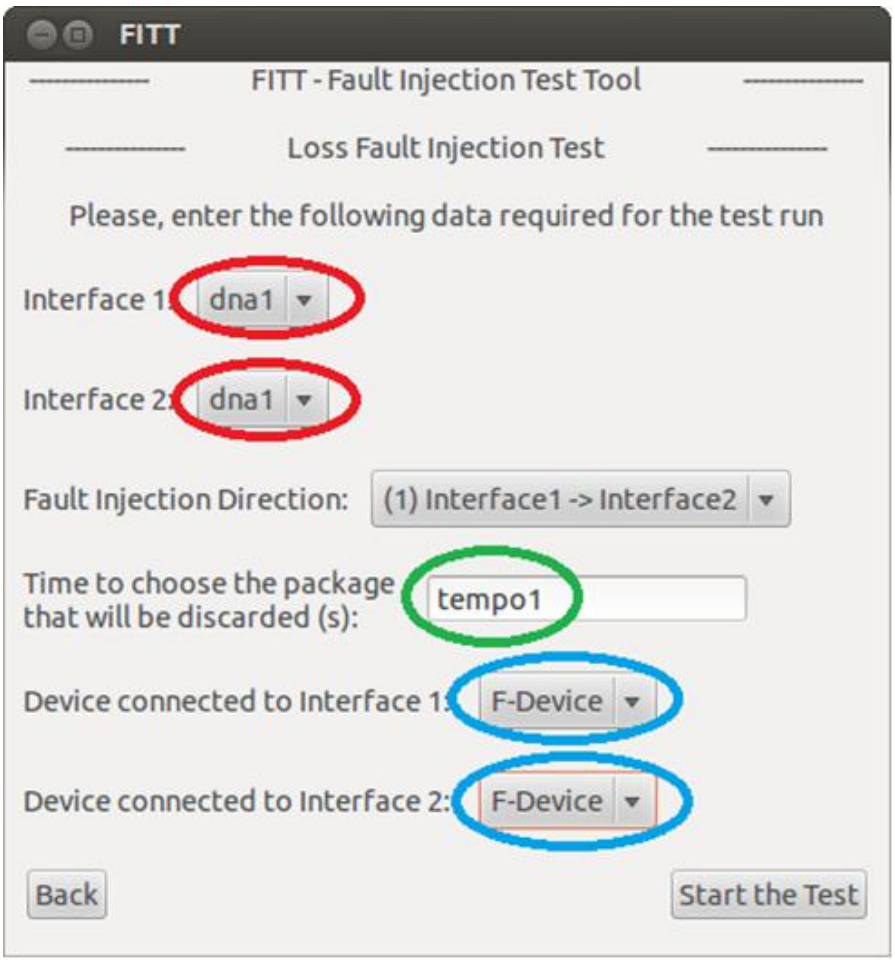
7 LIMITAÇÕES DO INJETOR DE FALHAS

Nesta seção é feita uma análise das limitações do injetor de falhas identificadas após o seu desenvolvimento e durante a realização dos testes. A análise dessas limitações é importante para que o *FITT* possa ser modificado futuramente, tornando-se uma ferramenta mais completa e com menos restrições relacionadas ao seu funcionamento.

7.1 Validação dos Dados de Entrada do Usuário

Um dos problemas está na entrada dos dados para configuração dos cenários de falhas na interface gráfica. Nas janelas responsáveis por receber essa configuração, não foi feita a validação dos dados de entrada digitados pelo usuário. Assim, se um campo receber um valor errado, como mostra a figura 7.1, isso pode gerar um erro na aplicação do injetor.

Figura 7.1 – Entrada de dados inválida do usuário



Fonte: Interface Gráfica do Injetor de Falhas *FITT*.

Por exemplo, o injetor não irá funcionar se nos campos para selecionar as interfaces 1

e 2 for escolhida a mesma interface *DNA*. Também irá ocorrer um erro de execução se no(s) campo(s) em que se deve definir o tempo de injeção de falhas for colocado qualquer palavra ou símbolo. Esse fato ocorre porque a(s) variável(s) que esperam receber um dado do tipo inteiro representando o tempo em segundos estarão recebendo um valor diferente do esperado.

Por outro lado, nos campos em que se deve definir qual dispositivo está conectado a qual das interfaces 1 e 2, se for escolhido o mesmo dispositivo (*F-Host* ou *F-Device*) para as duas interfaces, não irá ocorrer o erro. No entanto será difícil saber qual dispositivo (*F-Host* ou *F-Device*) que realmente está conectado em qual das interfaces e em qual deles as falhas estarão sendo injetadas.

Assim, no presente momento o adequado funcionamento do *FITT* depende da correta inserção de dados pelo seu utilizador.

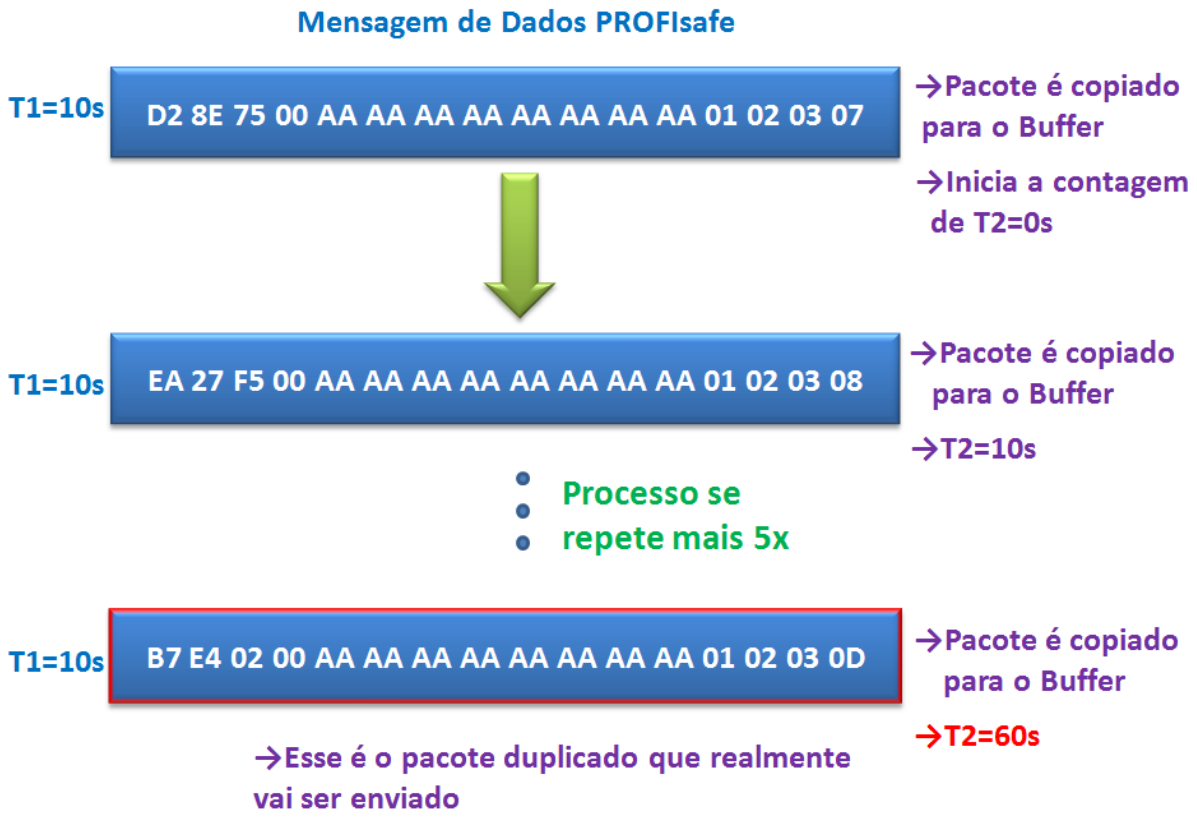
7.2 Limitação do Tamanho do Buffer e da escolha dos Tempos de Injeção de Falhas

Outro ponto limitante do injetor é o tamanho do *buffer* utilizado para armazenar o(s) pacote(s) escolhido(s) nos testes de repetição e inserção de pacotes. Como a área de memória alocada só tem espaço para receber um pacote, se o tempo definido para o envio do pacote a ser atrasado, duplicado ou inserido (tempo 2) for maior do que o tempo estabelecido para escolher o pacote que vai sofrer o processo de injeção de falhas (tempo 1), o pacote contido no *buffer* será sobre-escrito por um novo pacote. Isso acontece porque a cada vez que o tempo 1 é decorrido, o próximo pacote recebido será copiado para o *buffer* para ser duplicado ou alterado e depois inserido na comunicação. Dessa forma, nesses dois tipos de teste, o usuário pode pensar em escolher um determinado pacote para injetar falhas, mas na realidade um pacote diferente é que vai ser utilizado no processo de injeção de falhas.

Por exemplo, em um teste de duplicação de pacotes, se o tempo para escolha do pacote for definido em 10s (tempo1) e o tempo para envio do pacote duplicado for especificado como de 60s (tempo 2), o pacote escolhido inicialmente não será o pacote enviado quando o tempo 2 tiver decorrido. Isso acontece porque o tempo 1 terá sido contado sete vezes em ciclo antes que o tempo 2 tenha sido alcançado, pois o tempo 2 só começa a ser contado depois que o tempo 1 decorreu a primeira vez. Assim, no teste realizado, o pacote do *Safety PDU* inicialmente selecionado pelo primeiro ciclo de contagem do tempo 1 foi o ***D2 8E 75 00 AA AA AA AA AA AA AA AA 01 02 03 07***. Após o sétimo ciclo de contagem do tempo 1, o pacote escolhido e copiado para o *buffer* foi o com conteúdo ***B7 E4 02 00 AA AA AA AA AA***

AA AA AA 01 02 03 0D. Em seguida esse pacote foi enviado para o seu destino, sendo o pacote efetivamente duplicado no processo. Essa situação é ilustrada na figura 7.2.

Figura 7.2 – Limitação relacionada ao *buffer* de uma posição e tempo 1 < tempo 2



Fonte: Rodrigo Dobler.

7.3 Teste de Transições Específicas na Máquina de Estados do F-Device

Devido ao fato de a ferramenta injetar somente falhas síncronas, ou seja, no tempo especificado pelo operador, percebeu-se durante os testes específicos de algumas transições entre alguns estados, que a máquina do *F-Device* não conseguia se recuperar do erro inserido. Isso aconteceu porque o tempo de injeção de falhas de 10s é muito pequeno para os tempos de operação da implementação utilizada do *PROFIsafe*, não deixando tempo suficiente para a máquina de estados do *F-Device* se recuperar do erro inserido. Assim, ela não consegue sair do ciclo de tratamento de erro e voltar para o ciclo normal de operação. Por esse motivo, observou-se que a máquina de estados do *F-Device* ficou "presa" entre os estados 26 e 27 no teste específico da transição T32 e ficou alternando entre os estados 26, 27 e 28 no teste da transição T31 na máquina de estados do dispositivo *F-Device*.

7.3.1 Transição T31

No teste da transição T31 da máquina de estados do *F-Device*, foi utilizada a função de erro de endereçamento para gerar um erro de *CRC* na direção *F-Host → F-Device*. Essa função foi escolhida porque a partir do estado 28 de "Verificação de Mensagem", a condição para executar a transição T31 é a ocorrência de um erro de *CRC* ou que a variável *ok_nr_cicles* seja menor do que dois (*ok_nr_cicles*<2).

O erro de *CRC* da mensagem com o endereço de destino trocado leva a máquina de estados do *F-Device* para o estado 27. Ao enviar uma mensagem para o *F-Host*, o *F-Device* informa o erro de *CRC*, o que leva o *F-Host* a ir para o estado 8 para tratar a falha. Como o tempo de injeção de falhas de 10s é pequeno (em relação a implementação do *PROFIsafe* utilizada no trabalho), não há tempo suficiente para a máquina de estado do *F-Device* conseguir sair do ciclo de estados 26, 27 e 28. Em consequência disso, o *F-Host* também fica limitado ao seu ciclo de recuperação de falha 8, 9 e 10. Os dois dispositivos ficam trocando mensagens nos seus respectivos ciclos de recuperação de falhas até serem desligados.

7.3.2 Transição T32

Já para testar a transição T32 da máquina de estados do *F-Device*, foi utilizada a função de perda de pacotes na direção *F-Host → F-Device*. Essa função foi utilizada porque a partir do estado 27 de "Espera de Mensagem no *F-Device*", a condição para executar a transição T32 é a ocorrência de um *timeout* (*F-WD_time*) pelo não recebimento da mensagem enviada pelo *F-Host*. Nesse teste foi utilizado um tempo de injeção de falhas (tempo 1) igual a 10s, que é menor do que o tempo de *timeout* de 11s configurado nos dispositivos. A intenção com esse tempo baixo para descarte de pacotes foi forçar a ocorrência de duas ou mais perdas simultâneas de pacotes vindos do *F-Host* para gerar dois ou mais *timeouts* seguidos no *F-Device* e, assim, realizar a transição T32.

No entanto, como o tempo 1 escolhido para o descarte de pacotes, é menor do que o tempo de *timeout*, todos os pacotes vindos do *F-Host* serão descartados. Dessa forma, a máquina de estados do *F-Host* fica indefinidamente fazendo as transições entre os estados 8 de "Preparo de Mensagem" e 9 de "Espera de *Ack*". Já o *F-Device* vai ficar igualmente limitado aos estados 26 de "Preparação de *Ack*" e 27 de "Espera de Mensagem".

8 CONSIDERAÇÕES FINAIS

8.1 Conclusões

Este trabalho apresentou o projeto e o desenvolvimento do injetor de falhas *FITT*, implementado com a técnica de injeção de falhas por *software* para ser utilizado com o sistema operacional *Linux.* No início do projeto foram definidos vários objetivos para o projeto da ferramenta *FITT*, como foi mostrado na Seção 1.2. Após a conclusão do trabalho, foi possível verificar que todos os objetivos esperados foram alcançados.

Os testes de injeção de falhas realizados mostraram a eficiência do *FITT* para validar os mecanismos de tolerância a falhas do protocolo *PROFIsafe*. Isso se deve ao fato das funções de injeção de falhas terem sido desenvolvidas de acordo com a especificação de falhas de comunicação da norma *IEC 61508*. Nesses testes, foi possível verificar as diferentes transições entre os estados das máquinas dos dispositivos *PROFIsafe* quando eles detectaram os diferentes tipos de erros inseridos na comunicação. Assim que esses erros foram detectados, ambos os dispositivos foram pra os seus respectivos estados de recuperação de falhas, trocando mensagens até eliminar as falhas na comunicação. Assim que as condições necessárias foram satisfeitas, ambos os dispositivos voltaram aos seus estados normais de operação até que o ciclo de injeção de falhas recomeçasse.

Por utilizar uma abordagem externa de injeção de falhas em conjunto com o módulo do *PF_RING*, a ferramenta apresenta uma intrusividade muito pequena na realização dos testes, como pode ser visto na Seção 6.4. Isso é essencial para ferramentas de injeção de falhas, pois, elas não devem interferir nos testes, de forma a proporcionar resultados mais precisos. Nesse sentido, a utilização do *PF_RING* foi muito importante, pois ele é o responsável pelo envio e recebimento de pacotes diretamente entre as interfaces de rede e a ferramenta. Dessa forma, ele evita que os pacotes precisem passar pelas estruturas do *kernel* do *Linux*, evitando que atrasos adicionais sejam inseridos no processo de injeção de falhas.

Por cuidar da comunicação entre o injetor e as placas de rede, o *PF_RING* também facilitou o desenvolvimento da ferramenta, poupando a necessidade de desenvolver um módulo para o *kernel* do *Linux* para essa finalidade. Essas características podem ser utilizadas em outros injetores de falhas que possam e/ou precisem fazer uso dessas vantagens.

Outro ponto de destaque do *FITT* é a sua interface gráfica simples e amigável, a qual torna mais fácil e eficiente a configuração do cenário de falhas quando comparado a outros

injetores de falhas, como por exemplo o *Firmament*, no qual o cenário de falhas é especificado utilizando-se a linguagem *Assembly*. A interface gráfica também permite a visualização do relatório gerado ao final dos testes, possibilitando ao usuário analisar e avaliar os resultados obtidos de forma rápida.

A utilização da técnica de injeção de falhas por *software* foi muito importante no desenvolvimento do *FITT*, pois as funções de injeção de falhas puderam ser desenvolvidas de modo a simular os tipos de falhas descritos nas normas. A injeção de falhas por *software* também tem como vantagem permitir que o injetor de falhas evolua com o tempo, com a adição de novas funções de falhas para testar outros protocolos seguros. Nesse sentido, o *FITT* já possui algumas funções de injeção de falhas que podem ser utilizadas para testar outros protocolos seguros, como por exemplo, as funções de atraso, repetição e perda de pacotes. Outras funções como inserção, corrupção, erro de endereçamento e sequência incorreta precisam ser adaptadas ou reconstruídas para atender os requisitos das especificações desses protocolos seguros.

Na atual configuração, o injetor de falhas trabalha com pacotes *UDP/IP*, tipo *Ethernet* 0x0800, para a transmissão dos pacotes *PROFIsafe*. Para injetar falhas em outros protocolos de comunicação, o injetor apenas precisa ser configurado para selecionar outros tipos de pacotes sobre *Ethernet*. Por exemplo, para atuar sobre os pacotes do protocolo *PROFINET*, o *FITT* deve atuar nos pacotes com tipo *Ethernet* 0x8892. Já para testar o protocolo *EtherCAT*, o tipo *Ethernet* a ser escolhido deve ser 0x88A4. Essa modificação deve ser feita nas funções que controlam o recebimento e o envio dos pacotes nas duas interfaces *DNA* do injetor de falhas. Essas possibilidades de adaptações e modificações tornam o *FITT* uma ferramenta muito flexível para o teste de protocolos de comunicação seguros.

8.2 Trabalhos Futuros

O injetor de falhas *FITT*, na sua versão atual, é capaz de simular quase todos os tipos de falhas de comunicação descritos na norma *IEC 61508* e na especificação do protocolo *PROFIsafe*. No entanto, o injetor pode ser melhorado para se tornar uma ferramenta mais completa e eficiente ao realizar o processo de injeção de falhas.

Uma dessas melhorias diz respeito às funções de injeção de falhas para simular erros de mascaramento e de retransmissão contínua de mensagens armazenadas em *switches*, mesmo quando o emissor já tiver sido desligado. Elas não foram implementadas neste

trabalho devido a questões de tempo e de aumento de complexidade. Para simular a falha de mascaramento seria preciso alterar os dispositivos *PROFIsafe* para que eles também trabalhassem com mensagens não seguras, para causar uma "confusão" no envio e recebimento entre mensagens seguras e não seguras. Já para a função de retransmissão contínua de mensagens seria preciso criar um "*switch*" em *software*. Futuramente, com a adição dessas duas funções de injeção de falhas, o injetor estará simulando todos os erros possíveis descritos na especificação do *PROFIsafe*.

Uma mudança interessante na realização dos testes seria a possibilidade de permitir a especificação de mais de um tipo de falha para acontecerem em tempos específicos. Isso permitiria verificar o comportamento das máquinas de estados dos dispositivos *F-Host* e *F-Device* quando tipos diferentes de falhas acontecem ao longo do tempo.

A injeção de falhas assíncrona, em momentos diferentes ao longo do tempo, também é relevante, pois seria uma forma de contornar o problema relatado nos testes das transições T31 e T32 da máquina de estados do *F-Device*. Assim a máquina de estados conseguiria se recuperar dos erros inseridos, voltando ao estado normal de execução e não ficando restrita a essas transições devido à injeção de falhas síncrona num curto espaço de tempo.

Outra alteração diz respeito à implementação do *F-Host*, onde todos os seus dados poderiam ser transformados em uma *struct* para poder instanciar vários dispositivos em um único *main.c*, que controlaria todos esses dispositivos, simulando uma planta de processos industriais. Adicionalmente, o arquivo *ftimer.c*, que controla o tempo de *timeout*, deveria ser transformado em uma biblioteca para poder ser implementado em qualquer tipo de *hardware*.

Além disso, as implementações dos dispositivos *F-Host* e *F-Device* poderiam receber variáveis para conter os valores para os *F-Parameters*. Dessa forma, o valor para o *CRC1* passaria a ser calculado com os *F-Parameters* definidos e não especificado aleatoriamente.

Os testes apresentados para verificar o funcionamento dos mecanismos de tolerância a falhas foram realizados no sentido *F-Host* → *F-Device*. Assim, pretende-se apresentar futuramente em outros trabalhos o teste do protocolo *PROFIsafe* no sentido *F-Device* → *F-Host*, de forma a analisar a transição das máquinas de estados dos dois dispositivos.

Por fim, outro ponto a ser considerado é a adição de novas funções de injeção de falhas específicas para validar outros protocolos de comunicação seguros. Essas funções seriam criadas para injetar falhas de acordo com as particularidades da implementação de cada um desses protocolos de comunicação seguros.

REFERÊNCIAS

ARLAT,J., AGUERA, M., AMAT, L., CROUZET, Y., FABRE, J.C., LAPRIE, J.C., MARTINS, E., andPOWELL, D. "Fault Injection for Dependability Validation—A Methodology and Some Applications," **IEEE Transactions on Software Engineering.**, vol. 16, no. 2, pp. 166-182, Feb. 1990.

ARLAT, J., BOUÉ,J., and CROUZET,Y. "Validation-Based Development of Dependable Systems," **IEEE Micro**, vol. 19, no. 4, pp. 66-79, July/Aug. 1999.

ARLAT, J., CROUZET, Y., KARLSSON, J., FOLKESSON, P., FUCHS, E., LEBER, G.H. Comparison of physical and software implemented fault injection techniques, **IEEE Transactions on Computers**. Vol.52, No.8 Sept. 2003, pp.115-1133.

ARLAT, J., CROUZET, Y. and LARIE, J.C.. Fault Injection for Dependability Validation of Fault-Tolerant Computing Systems. In Proc. 19th International Symposium on Fault-Tolerant Computing (FTCS-19), (Chicago, IL, USA), pages 384-355. **IEEE CS Press**, 1989.

AVIZIENIS, A. and LAPRIE, J. "Dependable computing: from concepts to design diversity", **Proceedings IEEE**, vol. 74, no. 5, May 1986 pp.629 -638.

Barcellos, M., Woszezenki, C., and Munaretti, R. (2005). "Framework de Injeção de Falhas Simulada para Avaliação de Sistemas Distribuídos". In **Proc. Anais do 23o. Simpósio Brasileiro de Redes de Computadores** - SBRC, Fortaleza, CE, Brasil.

BENSO, A. and PRINETTO, P. (Editors), Fault InjectionTechniques and Tools for Embedded Systems Reliability Evaluation, **Kluwer Academic Publishers**, 2003.

BOURICIUS, W.G., CARTER, W.C., SCHNEIDER, P.R.," Reliability Modeling Techniques for Self-repairing Computer Systems", **Proceedings 24th ACM National Conference**, 1969 ,pp. 295-309.

Carreira, J., Madeira, H. and Silva, J.G. (1998). "Xception: A Technique for the Experimental Evaluation of Dependability". In **Modern Computers, IEEE Transactions on Software Engineering**., vol. 24, no. 2, pp. 125-136.

Carvalho, J.A., Carvalho, A.S. and Portugal, P.J. (2005). "Assessment of PROFIBUS networks using a fault injection framework". In 10th **IEEE Conference on Emerging Technologies and Factory Automation**.

CLARK, J. , PRADHAN, D. "Fault Injection. A method for validating computer-system dependability", **IEEE Computer**, June 1995.

DREBES, R. J. (2005). **FIRMAMENT: Um Módulo de Injeção de Falhas de Comunicação para Linux**. Dissertação (Mestrado em Ciência da Computação) – UFRGS.

Drebes, R., Silva, G., Trindade, J. and Weber, T. (2006). "A Kernel-based Communication Fault Injector for Dependability Testing of Distributed Systems". In **Hardware and**

Software, Verification and Testing, Haifa, pg 177–190.

EtherCAT FSoE 2015 – Safety over EtherCAT. Disponível em: <https://www.ethercat.org/en/safety.html>. Acesso em: Outubro 2015.

Glade 2015. Disponível em: < https://glade.gnome.org/>. Acesso em Março 2015.

GESSNER, D., BARRANCO, M., BALLESTEROS, A., PROENZA, J. Designing sfiCAN: A star-based physical fault injector for CAN, 16th **IEEE Conference on Emerging Technologies & Factory Automation (ETFA)**, 2011.

HSUEH, M.C., TSAI,T.K. and IYER, R.K. "Fault Injection Techniques and Tools," **IEEE Computer**, vol. 30, no. 4, pp. 75-82, Apr. 1997.

International Electrotechnical Commission *"IEC 61508* - Functional Safety Of Eletrical/Electronic/Programmable Electronic Safety-Related Systems", 2010.

International Electrotechnical Commission "IEC 61784-3 - Functional Safety Fieldbuses - General rules and profile definitions", 2010.

KARLSSON,J., LIDÉN, P., DAHLGREN, P., JOHANSSON, R. and GUNNEFLO, U."Using Heavy-Ion Radiation to Validate Fault-Handling Mechanisms," **IEEE Micro**, vol. 14, no. 1, pp. 8-23, Feb. 1994.

KIM, H., WHITE, A., SHIN, K. "Effects of Electromagnetic Interference on Controller-Computer Upsets and System Stability", **IEEE Transactions on Control Systems Technology**, Vol. 8, pp. 351-357, 2000.

LAPRIE, J.C. "Dependability - its attributes, impairments and means," in ***Predictable Dependable Cornputing Systems,*** Eds. Springer Verlag, 1995, pp. 3-18.

Maia, R., Enriques, L., Barbosa, R., Costa, D., and Madeira, H. (2005). "Xception Fault Injection and Robusteness testing Framework: a case-study of testing RTEMS". **In Workshop de Testes e Tolerância a Falhas**, 23o. Simpósio Brasileiro de Redes de Computadores - SBRC, Fortaleza, CE, Brasil.

Mühlhause, M., Diedrich, C., Riedl, M. and Schmidt, D. (2007). "Formalised specification of a test tool for safety related communication". In 12th IEEE **Conference on Emerging Technologies and Factory Automation**, Patras, Greece.

Neumann, P. (2007). "Communication in Industrial automation—What Is Going On?" **Control Engineering Practice** 15 (11): 1332–1347.

openSAFETY 2015. Disponível em: < http://www.open-safety.org/ >. Acesso em: Outubro 2015.

PI. 2015 – Profibus e Profinet International. Disponível em: <http://www.profibus.com/home/>. Acesso em: Novembro 2015.

PF_RING 2013. Disponível em: < http://www.ntop.org/products/pf_ring/>. Acesso em: Julho 2013.

PROFIBUS 2015. Disponível em: <http://www.profibus.com/technology/profibus/>. Acesso em: Outubro 2015.

PROFINET 2015. Disponível em: <http://www.profibus.com/technology/profinet/>. Acesso em: Outubro 2015.

PROFIsafe System Description – Profile for Safety Technology on PROFIBUS DP and PROFINET IO, 2010.

Siqueira, T., Fiss, B., Menegotto, C. Cechin, S. and Weber, T. (2009). "Avaliação Experimental de Estratégias de Tolerância a Falhas Do Protocolo SCTP Por Injeção de Falhas". In XXVII **Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos,** 1: pg. 915–929.

Thomesse, J. P. (2005). "Fieldbus technology in industrial automation," **Proceedings of the IEEE**, vol. 93, no. 6, pp. 1073–1101.

YU,Y., BASTIEN, B. and JOHNSON, B. (2005) "A state of research review on fault injection techniques and a case study". In **Reliability and Maintainability Symposium**. Proceedings. Annual, 24-27, 2005, pp. 386-392.