Pós-Graduação em Ciência da Computação

Luís Felipe Prado D'Andrada

Um sistema de detecção de intrusão de tempo real e baseado em anomalias para redes CAN automotivas

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

Recife
2020

Luís Felipe Prado D'Andrada

Um sistema de detecção de intrusão de tempo real e baseado em anomalias para redes CAN automotivas

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Redes de Computadores
Orientador: Divanilson Rodrigo de Sousa Campelo

Recife
2020

Catalogação na fonte
Bibliotecária Arabelly Ascoli CRB4-2068

D178s D'Andrada, Luís Felipe Prado
Um sistema de detecção de intrusão de tempo real e baseado em anomalias para redes CAN automotivas / Luís Felipe Prado D'Andrada. – 2020.
65 f.: il. fig., tab.

Orientador: Divanilson Rodrigo de Sousa Campelo
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação. Recife, 2020.
Inclui referências e apêndices.

1. Controller Area Network. 2. Sistema de detecção de intrusão. 3. Redes intraveiculares. 4. Segurança de redes. I. Campelo, Divanilson Rodrigo de Sousa (orientador). II. Título.

004.6 CDD (22. ed.) UFPE-CCEN 2020-94

Luís Felipe Prado D'Andrada

“Um sistema de detecção de intrusão de tempo real e baseado em anomalias para redes CAN automotivas”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 13 de fevereiro de 2020.

BANCA EXAMINADORA

_____

Prof. Dr. Daniel Carvalho da Cunha
Centro de Informática / UFPE

_____

Prof. Dr. Victor Wanderley Costa de Medeiros
Departamento de Estatística e Informática/UFRPE

_____

Prof. Dr. Divanilson Rodrigo de Sousa Campelo
Centro de Informática / UFPE
(**Orientador**)

Dedico esta dissertação aos meus pais, a toda minha família e amigos.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, que, cada um do seu jeito, me apoiaram, confiaram em mim e me fizeram crescer como pessoa.

Aos meus colegas de grupo de pesquisa, em especial a Paulo, que foram essenciais no percurso do desenvolvimento deste trabalho.

Ao meu orientador Divanilson Campelo, que me guiou nessa desafiadora trajetória de pesquisa acadêmica, com seus conselhos de valor imensurável.

E a todos os professores do Centro de Informática, por todos os ensinamentos, sejam acadêmicos ou não.

RESUMO

A *Controller Area Network* (CAN) é a tecnologia de rede intraveicular mais presente em automóveis. Apesar disso, como a CAN não foi projetada para se defender de ataques cibernéticos, soluções para mitigar estes ataques têm sido propostas nos últimos anos. Trabalhos anteriores mostraram que detectar anomalias no tráfego da rede CAN é uma solução promissora para o aumento da segurança veicular. Um dos principais desafios na prevenção da transmissão de um quadro CAN malicioso é a capacidade de detectar anomalias até o fim da transmissão do quadro. Este trabalho apresenta um sistema de detecção de intrusão de tempo real e baseado em anomalias capaz de atender a esse *deadline* através da utilização do algoritmo de detecção *Isolation Forest* implementado em uma linguagem de descrição de hardware. Uma taxa de verdadeiro positivo maior que 99% foi alcançada nos cenários de teste. O sistema requer menos que 1μs para avaliar o *payload* de um quadro, portanto sendo possível detectar a anomalia antes do fim do quadro.

Palavras-chaves: Controller Area Network. Sistema de detecção de intrusão. Redes intraveiculares. Segurança de redes.

ABSTRACT

The Controller Area Network (CAN) is the most pervasive in-vehicle network technology in cars. However, since CAN was designed with no security concerns, solutions to mitigate cyber attacks on CAN networks have been proposed in the last years. Prior works have shown that detecting anomalies in the CAN network traffic is a promising solution for increasing vehicle security. One of the main challenges in preventing a malicious CAN frame transmission is to be able to detect the anomaly before the end of the frame. This work presents a real-time anomaly-based Intrusion Detection System (IDS) capable of meeting this deadline by using the Isolation Forest detection algorithm implemented in a hardware description language. A true positive rate higher than 99% is achieved in test scenarios. The system requires less than 1μs to evaluate a frame's payload, thus being able to detect the anomaly before the end of the frame.

Keywords: Controller Area Network. Intrusion Detection System. Intravehicular networks. Network Security.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AD	*Anomaly-based Detection*
AUROC	*Area Under Receiver Operating Characteristics*
CAN	*Controller Area Network*
CAN FD	*CAN with Flexible Data Rate*
CRC	*Cyclic Redundancy Check*
CUSUM	*Cumulative Sum*
DLC	*Data Length Code*
DNN	*Deep Neural Network*
DoS	*Denial of Service*
ECU	*Electronic Control Unit*
EOF	*End Of Frame*
FN	Falso Negativo
FP	Falso Positivo
FPGA	*Field Programmable Gate Array*
GAN	*Generative Adversarial Nets*
GIDS	*GAN-based IDS*
GPIO	*General Purpose Input Output*
HIDS	*Host-based Intrusion Detection System*
ID	*Identifier*
IDE	*Identifier Extension Bit*
IDS	*Intrusion Detection System*
IPS	*Intrusion Prevention System*
ISO	*International Organization for Standardization*
LODA	*Lightweight On-Line Detector of Anomalies*
NIDS	*Network-based Intrusion Detection System*
NRZ	*Non Return to Zero*
OBD	*On-board diagnostics*
OCSVM	*One-Class Support Vector Machine*

PDU	*Protocol Data Unit*
RAIC	*Real-time Anomaly-based IDS for CAN*
RNN	Recurrent Neural Network
ROC	*Receiver Operating Characteristics*
RTOS	*Real-Time Operating System*
RTR	*Remote Request*
SBRC	Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos
SD	*Signature-based Detection*
SoF	*Start of Frame*
SPA	*Stateful Protocol Analysis*
TFP	Taxa de Falso Positivo
TVP	Taxa de Verdadeiro Positivo
VN	Verdadeiro Negativo
VP	Verdadeiro Positivo

SUMÁRIO

1 INTRODUÇÃO

Este Capítulo apresenta o contexto para segurança em redes automotivas, em especial a rede *Controller Area Network* (CAN), e justifica a escolha de um sistema de detecção de intrusão (*Intrusion Detection System*, IDS), como uma linha de defesa para ataques nesta rede. Por fim, são apresentados os objetivos deste trabalho e a estrutura da dissertação.

1.1 MOTIVAÇÃO

Um automóvel moderno é composto por dezenas de unidades de controle eletrônico (*Electronic Control Units*, ECUs), que são responsáveis por controlar sistemas elétricos, como os faróis ou mesmo a direção. Essas ECUs são, normalmente, conectadas por redes intraveiculares, como a CAN.

CAN é um barramento serial multi-mestre desenvolvido pela Bosch nos anos 80 e posteriormente padronizado pela *International Organization for Standardization* (ISO). Inicialmente, a rede CAN veio para substituir as comunicações ponto a ponto que, devido ao aumento no número de dispositivos eletrônicos utilizados num automóvel, tornaram-se bastante ineficientes e custosas. Assim, era necessária uma rede intraveicular que utilizasse um meio de transmissão compartilhado, reduzindo bastante a utilização de cabos e a complexidade da rede.

Atualmente, a tecnologia de rede CAN ainda é a mais presente em automóveis, apesar de já existirem requisitos diferentes da época em que foi concebida. Naquela época, não existia uma grande preocupação com a segurança da rede intraveicular, pois o automóvel era completamente isolado, sem conexão nenhuma com o mundo externo. Porém, com a crescente conectividade presente em automóveis modernos (CHECKOWAY et al., 2011) (MILLER; VALASEK, 2014), como *Bluetooth*, *WiFi* e as tecnologias 4G e 5G, ataques remotos a carros se tornaram uma possibilidade (MILLER; VALASEK, 2015) (GREENBERG, 2015). Diante das vulnerabilidades já expostas e as possíveis consequências das mesmas, melhorar a segurança das redes intraveiculares tornou-se uma demanda de interesse da sociedade.

(LIU et al., 2017) descreve cinco metodologias de ataque a redes intraveiculares que são eficazes, tendo sido comprovadas pela literatura: (i) *sniffing* de quadros, (ii) falsificação de quadros, (iii) injeção de quadros, (iv) ataque de repetição e (v) *Denial of Service* (DoS). O *sniffing* de quadros, que é a base para outros ataques, consiste em observar e gravar o tráfego do barramento CAN a fim de aprender os detalhes dos quadros CAN e assim aprender sobre o funcionamento da rede. Além disso, um atacante pode enviar quadros com dados aleatórios a fim de observar e ganhar conhecimento sobre o comportamento do sistema. Tendo aprendido sobre funções do sistema, o atacante pode mandar quadros

projetados para realizar ataques específicos. Estes quadros são válidos para o sistema, porém contêm dados falsos que podem enganar o sistema, caracterizando a falsificação de quadros, também conhecido como *spoofing*. A injeção de quadros se dá através do envio de quadros para a rede por um nó malicioso, que pode ser, por exemplo, um laptop conectado a porta *On-board diagnostics* (OBD), uma ECU reprogramada ou mesmo um sistema de telemetria infectado por um malware. Um ataque de repetição é mais simples que os ataques mencionados anteriormente, consistindo do envio de quadros válidos que foram previamente capturados através de um *sniffer* no tempo apropriado. Apesar de ser um ataque simples de executar, ele pode causar ameaças significantes. Por fim, o ataque DoS se aproveita do esquema de prioridades de mensagens da CAN para impedir que nós genuínos da rede transmitam seus quadros em tempo hábil, causando indisponibilidade.

1.2 JUSTIFICATIVA

A fim de adicionar segurança à rede CAN, uma possível solução é o uso de um IDS, cujo funcionamento, em geral, consiste na análise dos dados trafegados e alguma indicação, caso seja detectado um quadro suspeito na rede. O uso de um IDS como um meio para aumentar a segurança da rede intraveicular já foi sugerido na literatura (MILLER; VALASEK, 2014) (LIU et al., 2017). Essa abordagem é adequada ao cenário da rede CAN automotiva, pois, como o módulo IDS pode ser agregado a uma rede em funcionamento, não requer que o sistema seja reprojetado, além de funcionar bem num cenário com recursos limitados e de tempo real. Essas características tornam o IDS uma alternativa atrativa para a indústria automotiva.

Além da abordagem de detecção de intrusão, (LIU et al., 2017) elenca mais duas contramedidas a ataques a rede CAN: o uso de encriptação e autenticação nas mensagens trafegadas na rede e a separação de potenciais interfaces de ataque da rede CAN. O uso de ferramentas criptográficas traria perdas na eficiência da transmissão, ou seja, agregaria um *overhead* ao protocolo. Além disso, seria necessário realizar o gerenciamento das chaves criptográficas, incluindo sua distribuição e atualização. Essas características, que podem exigir que o sistema seja reprojetado, tornam essa abordagem menos atrativa para a indústria, apesar de ser bastante eficaz na prevenção de ataques. A separação de outras interfaces da rede CAN é uma forma de adicionar mais uma camada de defesa a ataques na rede CAN. Esta abordagem visa impedir que um atacante sequer tenha acesso a rede CAN, portanto, apesar de importante, não tem como objetivo mitigar as vulnerabilidades intrínsecas desta rede.

Uma possível evolução em relação ao uso de um IDS, seria a utilização de um sistema de prevenção de intrusão (*Intrusion Prevention System*, IPS). Este tipo de sistema, além de detectar quadros suspeitos, também é capaz de agir sobre a rede, impedindo que um ataque alcance seu objetivo. (ABBOTT-MCCUNE; SHAY, 2016) e (GIANNOPOULOS; WYGLINSKI; CHAPMAN, 2017) sugerem a inserção de um erro de *bit stuffing* durante a

transmissão de um quadro malicioso a fim de impedir que ele seja transmitido com sucesso pela rede. Porém, a fim de conseguir impedir a transmissão de um quadro, é necessário que a detecção seja realizada rapidamente, antes da transmissão completa do quadro. A maioria dos IDSs propostos na literatura não conseguem atender a este requisito temporal, principalmente se utilizarem técnicas mais complexas, como aprendizagem de máquina.

(ARAUJO-FILHO, 2018) compara o desempenho de algoritmos de detecção de anomalias no cenário da rede CAN automotiva. A detecção de anomalias é uma técnica adequada para um cenário pouco explorado por atacantes, como a rede CAN automotiva, pois é capaz de detectar ataques desconhecidos. (ARAUJO-FILHO, 2018) concluiu que o algoritmo *Isolation Forest* apresenta bons resultados neste cenário, como uma taxa de precisão de mais de 99%. Como o algoritmo do *Isolation Forest* utiliza árvores de decisão para detectar anomalias, é possível exportar o modelo treinado para uma *Field Programmable Gate Array* (FPGA) usando uma metodologia similar a proposta em (STRUHARIK, 2011), que possui um limite superior para o tempo de execução.

Uma solução em FPGA pode ser utilizada a fim de atender a restrição temporal requerida, porém uma abordagem em CPU também pode ser possível. (BUSCHJÄGER; MORIK, 2017) compara ambas as abordagens para a execução de árvores de decisão e *Random Forests*, que possuem grandes semelhanças com o *Isolation Forest*. Eles concluíram, através dos seus experimentos, que a abordagem em FPGA é definitivamente melhor em termos de consumo de energia, além de prover um *throughput* aceitável, considerando uma única árvore. Para uma floresta, caso o modelo consiga atender as limitações da FPGA, o *throughput* tende a ser melhor, considerando o potencial de paralelização da FPGA.

Levando em consideração o seu comprovado desempenho no cenário tratado, baixa complexidade temporal de seu algoritmo de detecção, que será explanado em mais detalhes no Capítulo 2, e a possibilidade de realizar uma implementação eficiente em hardware, o algoritmo *Isolation Forest* foi selecionado.

1.3 OBJETIVOS

Diante do cenário apresentado nas Seções anteriores, este trabalho propõe o *Real-time Anomaly-based IDS for CAN* (RAIC), um sistema de detecção de intrusão de tempo real e baseado em anomalias que atende ao requisito temporal necessário a construção de um IPS, ou seja, é capaz de responder antes do fim do envio completo do quadro CAN.

Os objetivos principais deste trabalho são:

- Propor uma metodologia de construção de um IDS de tempo real baseado em anomalias, o RAIC;
- Propor um IPS para CAN, viabilizado pelo RAIC;

- Validar a metodologia proposta através da realização de experimentos em um cenário simulado.

Além disso, temos como objetivos secundários:

- Propor uma implementação em hardware para o algoritmo de avaliação de uma amostra do *Isolation Forest*, que inclui desde o cálculo do caminho percorrido pela amostra em uma árvore até a comparação do *score* obtido com o limiar definido;
- Disponibilizar implementações em linguagem de alto nível que auxiliaram no desenvolvimento do código em linguagem de descrição de hardware do algoritmo supracitado, automatizando a criação de parte do código;
- Disponibilizar as modificações realizadas na biblioteca *sklearn* a fim de obter acesso à estrutura interna do modelo *Isolation Forest* treinado.

Seguindo esta proposta, o resultado deste trabalho foi submetido ao Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC) 2020 sob a forma de um artigo intitulado "A Real-time Anomaly-based Intrusion Detection System for Automotive Controller Area Networks". O artigo foi aceito e será apresentado no SBRC, que será realizado de 7 a 10 de dezembro de 2020 na cidade do Rio de Janeiro.

1.4 ESTRUTURA DA DISSERTAÇÃO

Esta dissertação é estruturada da seguinte forma. O capitulo 2 aborda tópicos necessários para uma melhor compreensão do trabalho e da solução proposta. O Capítulo 3 descreve o estado da arte relacionado à detecção de intrusão em redes CAN automotivas. A metodologia para a construção do IDS proposto é descrita no Capítulo 4. O experimento realizado para validação da metodologia é descrito no Capítulo 5, enquanto os resultados obtidos são apresentados no Capítulo 6. Por fim, as conclusões acerca deste trabalho são discorridas no Capítulo 7, bem como os trabalhos futuros.

2 BACKGROUND

Este capítulo aborda tecnologias relacionadas à proposta deste trabalho, sendo essencial para o entendimento do mesmo. Na seção 2.1, a rede CAN é explanada, abordando desde características físicas a detalhes do protocolo. A seção 2.2 aborda as principais classificações de IDS e suas características. Na seção 2.3 é descrito o funcionamento do algoritmo de detecção de anomalias *Isolation Forest*. Por fim, na seção 2.4 são apresentadas as métricas de avaliação de classificadores binários utilizadas neste trabalho.

2.1 CONTROLLER AREA NETWORK

2.1.1 Visão geral

A rede CAN nasceu da necessidade de uso de um meio compartilhado para transmissão de dados entre ECUs. Entre os fatores que contribuíram para que a rede CAN fosse largamente adotada pela indústria estão a sua política de licença aberta e a alta disponibilidade de controladores e *transceivers* CAN para a indústria. Apesar da crescente demanda por maior largura de banda, a rede CAN continua sendo uma das mais relevantes tecnologias de rede intraveicular, principalmente devido ao seu custo-benefício e por ser uma solução consolidada, característica importante para sistemas onde uma falha pode resultar em danos a equipamentos ou mesmo à integridade física de uma pessoa.

A tecnologia de rede CAN engloba ambas as camadas física e de enlace do modelo ISO/OSI, portanto sua unidade de dados de protocolo (*Protocol Data Unit*, PDU) é o quadro. A camada física define como os sinais são transmitidos, descrevendo, por exemplo, o meio de transmissão, a codificação dos bits e o *Bit Timing*, ou seja, quando se deve amostrar o sinal do bit. Já a camada de enlace é responsável por funcionalidades com um nível mais alto, como a arbitração, formato e tipos dos quadros, além do *Bit Stuffing*.

2.1.2 Camada Física

O tipo de sinal utilizado é digital com codificação de bits *Non Return to Zero* (NRZ), o que garante um número mínimo de transições e alta resiliência a interferência externa. Os dois bits são definidos como dominante e recessivo, sendo o bit 0 tipicamente associado ao dominante. A fim de possibilitar um barramento serial multi-mestre, é utilizada uma resolução de colisão determinística no mais baixo nível, significando que, se vários nós tentarem mudar o estado do barramento, a configuração dominante vai prevalecer sobre a recessiva. (NATALE et al., 2012)

Todos os nós devem estar sincronizados para que concordem com o valor do bit que está sendo transmitido na rede. Para isso, cada nó implementa um protocolo de sincronização que utiliza as transições do sinal da rede. A fim de garantir que não existam longas

sequências de bits iguais, ou seja, sem transições, a técnica de *Bit Stuffing* é utilizada. Esta técnica consiste em inserir um bit complementar após uma sequência de 5 bits igualmente valorados, assim forçando as transições. Além da sincronização nas transições durante a transmissão de um quadro, que é chamada de *Resynchronization*, ao início de um quadro é realizada uma *Hard Synchronization*.

Para que os mecanismos de sinalização de erros, arbitração e confirmação de mensagens funcionem corretamente, o protocolo requer que os nós sejam capazes de mudar o estado de um bit sendo transmitido, de recessivo para dominante. Para que todos os nós da rede consigam perceber essa mudança, o tempo total de transmissão de um bit deve ser grande o suficiente para que o sinal saia de um nó transmissor, chegue no receptor mais distante e volte para o transmissor. A duração da transmissão de um bit é composta por 4 segmentos: segmento de sincronização, de propagação, de fase, dividido em 1 e 2, e o *sample point*.

O meio físico utilizado não é especificado pela rede CAN, existindo ISOs que descrevem meios diferentes para utilização com CAN, sendo a mais comum a ISO 11989-2. Esta ISO define o meio como um barramento diferencial com dois cabos, identificados como CAN_H e CAN_L, com impedância característica de 120 Ω. As taxas de transmissão de bits variam entre 10Kbit/s a 1Mbit/s. Para que os sinais não sejam refletidos numa terminação do barramento, é necessária a utilização de um resistor de terminação, tipicamente com valor de impedância igual a impedância característica do cabo utilizado.

Como a transmissão do bit requer tempo suficiente para que o mesmo se propague até o nó mais distante e volte para o emissor, a utilização de taxas de transmissão de bits maiores implica na utilização de cabos com comprimento máximo menor. Por exemplo, numa taxa de transmissão de 500Kbit/s, o comprimento máximo do barramento é de 100m, enquanto a 1Mbit/s, o comprimento máximo é de apenas 25m (CAN IN AUTOMATION (CIA), 2018).

2.1.3 Camada de Enlace

2.1.3.1 O quadro CAN

A especificação do protocolo CAN (BOSCH, 1991) define 4 tipos de quadros: *Data Frame*, *Remote Frame*, *Error Frame* e o *Overload Frame*.

O *Data Frame* é utilizado para a troca de dados entres os nós da rede. Este quadro possui dois formatos: o formato base e o estendido; sendo a principal diferença entre os dois tipos a quantidade de bits utilizada para o campo *Identifier* (ID). O *Remote Frame* compartilha o mesmo formato que o *Data Frame*, porém seu objetivo é requisitar uma mensagem com um identificador específico, enquanto o *Data Frame* envia informações específicas definidas através do identificador utilizado. Os campos presentes nestes quadros, em seu formato base, são descritos em ordem a seguir.

- *Start of Frame* (SoF) - 1 bit

O SoF determina o início de um quadro, sendo sempre um bit dominante, ou seja, de valor 0.

- ID - 11 bits

  O ID é um identificador único para um determinado tipo de mensagem a ser enviada. Seu valor é utilizado na arbitração e, por isso, é responsável pela prioridade de mensagens

- *Remote Request* (RTR) - 1 bit

  O campo RTR determina se o quadro é um *Data Frame* ou um *Remote Frame* através do seu valor dominante ou recessivo, respectivamente.

- *Identifier Extension Bit* (IDE) - 1 bit

  O campo IDE determina se o quadro está no formato base, possuindo um valor dominante, ou formato estendido, possuindo um valor recessivo.

- Bit reservado - 1 bit

  Este campo contém apenas um bit reservado para uso futuro. Seu valor esperado é dominante, porém, no caso de conter um bit recessivo, não implica em erro.

- *Data Length Code* (DLC) - 4 bits

  Este campo define a quantidade de dados que serão enviados no campo de dados, de forma que o valor enviado neste campo é igual a quantidade de bytes esperados no campo de dados. O valor esperado neste campo é um número de 0 a 8, porém, caso seja enviado um valor maior que 8, nenhum erro deve ser gerado e o valor a ser considerado deve ser apenas 8.

- Campo de dados - 0 a 64 bits

  Neste campo, são enviados os dados da mensagem, podendo ser de uma mensagem vazia, ou seja, com 0 bytes, a uma mensagem com o tamanho máximo de 8 bytes ou 64 bits. A quantidade de bits é sempre múltipla de 8.

- *Cyclic Redundancy Check* (CRC) - 15 bits

  Este campo contém o CRC, que é utilizado para identificar erros na transmissão dos dados. Caso o valor contido neste campo seja diferente do esperado, será levantado um erro de CRC, que deve ser enviado após o campo *ACK Delimiter*.

- CRC Delimiter - 1 bit

  Este campo apenas delimita o fim dos bits de CRC. Deve sempre ter o valor dominante, pois um valor recessivo neste campo deve gerar um erro de forma.

- *ACK Slot* - 1 bit

  Ao receber uma mensagem corretamente, um receptor deve enviar, através deste campo, sua confirmação de recebimento. O transmissor deve enviar um bit recessivo neste campo, enquanto todos os receptores que receberam a mensagem corretamente devem enviar um bit dominante. Caso o transmissor perceba que o valor do barramento mudou para dominante, irá entender que sua mensagem foi corretamente recebida. Quando isto não acontece, o transmissor irá emitir um erro de *ACK*.

- *ACK Delimiter* - 1 bit

  É o bit seguinte ao *ACK Slot*, fazendo com que o mesmo seja cercado de bits recessivos, pelo menos em situações livres de erros. Este bit deve sempre ser recessivo, levando a um erro de forma caso contrário.

- *End Of Frame* (EOF) - 7 bit

  É o último campo do quadro CAN, indicando o seu fim. Todos os seus bits devem ser recessivos, porém, para receptores, o último bit deste campo é tratado como *don't care*, ou seja, pode assumir tanto o valor dominante como o recessivo. Caso os valores recebidos neste campo não estejam de acordo com o descrito previamente, deve ser levantado um erro de forma.

O *Error Frame* é tipo de mensagem transmitida quando um nó detecta um erro, sendo composto por dois campos: *Error Flags* e *Error Delimiter*. Um nó, ao detectar um erro, deverá iniciar a transmissão do *Error Flags* a partir do bit posterior ao qual o erro foi detectado, com exceção do *CRC Error* que irá iniciar a transmissão do *Error Flags* após o *ACK Delimiter*.

Por conta de possíveis atrasos de um nó para outro na detecção de erros em um *Data Frame* ou um *Remote Frame*, é possível que um nó inicie a transmissão de *Error Flags* após o outro, fazendo com que exista uma superposição na transmissão deste campo. Por isso, no pior dos casos, é possível que o tamanho do campo *Error Flags* seja, no máximo, o dobro do seu tamanho mínimo.

O protocolo CAN define 3 situações para o envio de um *Overload Frame*:

- Quando um nó necessita de um tempo adicional para receber o próximo *Data Frame* ou *Remote Frame*.

- Detecção de um bit dominante no primeiro ou segundo bit do *Intermission*

- Se um nó receber um bit dominante no oitavo (último) bit do *Error Delimiter ou Overload Delimiter*. Neste caso, será enviado um *Overload Frame* e o contador de erros não será alterado.

A estrutura do *Overload Frame* e *Error Frame* é igual, sendo descrita a seguir.

- *Error Flags / Overload Flags* - 6 a 12 bits

  Todos os bits deste campo são dominantes. Por conta da superposição, seu tamanho pode variar entre 6 a 12 bits.

- *Error Delimiter / Overload Delimiter* - 8 bits

  Todos os bits deste campo são recessivos e indicam o fim do *frame*.

Ambos os *Data Frame* e *Remote Frame* sempre serão precedidos por um campo de bits chamado *Interframe Spacing* e este campo pode ser precedido de qualquer um dos quatro tipos de quadros descritos anteriormente. O *Interframe Spacing* possui duas partes: o *Intermission* e o *Bus Idle*.

- *Intermission* - 3 bits

  Durante a transmissão deste campo nenhum nó pode iniciar uma transmissão de *Data Frame* ou *Remote Frame*. Os 3 bits deste campo devem ser enviados como recessivos. Caso o primeiro ou o segundo bit seja lido como dominante, este será interpretado como o primeiro bit do campo *Overload Flags*. Além disso, por conta da tolerância da duração do bit, caso o terceiro bit deste campo seja lido como dominante, este será reconhecido como o SoF do próximo *Data Frame* ou *Remote Frame*.

- *Bus Idle* - 0 a ∞ bits

  A transmissão deste campo pode ser por uma quantidade indefinida de bits, pois representa que o barramento está livre para transmitir. Enquanto nenhum nó deseja transmitir, o estado do barramento deve ser sempre recessivo. Quando é identificado um bit dominante, este deve ser considerado como o SoF do próximo *Data Frame* ou *Remote Frame*.

2.1.3.2 *Bit Stuffing*

O uso da técnica de *Bit Stuffing* fica ativo em um *Data Frame* ou *Remote Frame* entre os campos SoF e CRC, não incluindo o CRC Delimiter. Após o envio de 5 bits de mesmo nível lógico é necessário enviar um bit de nível lógico oposto, causando uma transição, necessária para efeitos de sincronização. Este bit adicionado artificialmente deve ser descartado pelo receptor, já que não representa nenhuma parte de nenhum campo. Porém, caso seja detectada uma sequência com 6 bits de mesmo nível lógico, o receptor irá detectar um *Stuff Error* e um *Error Frame* deverá ser transmitido no bit posterior.

2.1.3.3 Arbitração

O protocolo de arbitração de CAN é tanto baseado em prioridades quanto não preemptivo, pois uma mensagem que está sendo transmitida não pode ser preemptada por mensagens de maior prioridade depois que a transmissão já foi iniciada.

A arbitração se dá assim que o inicio do quadro é transmitido, ou seja, desde o SoF. Todos os nós que desejam transmitir começam a enviar o seus respectivos identificadores, que representam a prioridade da mensagem. Quando um nó envia um bit recessivo do seu identificador e lê um bit dominante no barramento, deve parar de transmitir, pois um outro nó está enviando uma mensagem de maior prioridade. Este mecanismo pode ser explorado por um atacante que deseja ferir a disponibilidade do sistema, fazendo com que várias mensagens com alta prioridade sejam enviadas, impedindo que mensagens autênticas trafeguem na rede, assim caracterizando um ataque DoS.

2.1.3.4 Tipos de erros

O protocolo CAN descreve 5 tipos de erros descritos a seguir.

- *Bit Error*

  Um nó transmissor também monitora o estado do barramento. Um *Bit Error* ocorre quando o valor lido no barramento é diferente do valor que está sendo enviado, sendo a exceção os bits enviados no processo de arbitração ou do *ACK Slot*.

- *Stuff Error*

  Este erro é detectado quando seis bits consecutivos de mesmo valor são recebidos em campos que o mecanismo de *bit stuffing* está ativo.

- *CRC Error*

  Este erro é acontece quando o valor de CRC computado por um nó receptor difere do valor recebido no campo CRC.

- *Form Error*

  Este erro acontece quando um campo de forma fixa contém um ou mais bits ilegais.

- *ACK Error*

  Este erro é detectado por um transmissor se um bit recessivo é lido no ACK Slot

2.1.4 Considerações de segurança

Após entender o funcionamento da rede CAN, é possível identificar algumas de suas vulnerabilidades. O quadro de dados possui um código CRC, que tem como objetivo detectar erros na transmissão dos dados. Porém, esse código não é capaz de realizar

nenhum tipo de autenticação, ou seja, não é capaz de garantir que o emissor da mensagem é um emissor legítimo. Além disso, também não há encriptação de mensagens, assim permitindo o *eavesdropping*, ou seja, a interceptação da comunicação por parte de um atacante. Como toda a comunicação é realizada em modo *broadcast*, todos os nós da rede escutam todas as trocas de mensagens. Assim, caso um atacante consiga acesso à rede através de apenas um nó, ele terá acesso a todas as comunicações entre ECUs daquela rede. Por fim, em decorrência do esquema de prioridade utilizado, um atacante pode realizar um ataque DoS apenas enviando continuamente quadros com alta prioridade, ou seja, com valores de ID baixos, usualmente zero. Estas vulnerabilidades, intrínsecas à rede CAN, permitem a execução dos ataques descritos na seção 1.1.

2.1.5 CAN FD

A rede CAN com taxa de dados flexível (*CAN with Flexible Data Rate*, CAN FD) surgiu da necessidade de maiores taxas de transmissão, a fim de suportar novas funcionalidades requeridas por um veículo moderno. Assim, a rede CAN FD possui duas grandes vantagens: uma maior largura de banda e um reduzido custo de migração e atualização, pois utiliza a mesma camada física que a CAN. (HARTWICH et al., 2012)

O funcionamento da CAN FD é, em muitos pontos, similar a CAN. Porém, a principal diferença está na possibilidade de realizar a transmissão dos campos de dados e CRC a uma velocidade diferente em relação aos outros campos do quadro. Isto é possível porque não é necessário realizar arbitração nos campos de dados. Portanto, não existe a necessidade do nó transmissor esperar a possível transmissão de outros nós durante a transmissão dos dados e CRC.

2.2 IDS, IPS E SUAS CLASSIFICAÇÕES

A função de um IDS é automatizar as tarefas de monitoramento de eventos que acontecem em uma rede e analisar sinais de problemas de segurança (SHIREY, 2007). A sua principal diferença em relação a um IPS se dá na ação a ser tomada após a detecção da atividade suspeita, pois o último tem a capacidade de realizar alguma ação que impeça a ameaça de atingir o seu objetivo (STAVROULAKIS; STAMP, 2010).

Os IDSs podem ser classificados quanto ao método de detecção utilizado: detecção baseada em assinaturas (*Signature-based Detection*, SD), detecção baseada em anomalias (*Anomaly-based Detection*, AD) e análise de protocolo (*Stateful Protocol Analysis*, SPA). A detecção baseada em assinaturas (SD) compara padrões de ataques conhecidos com eventos detectados a fim de reconhecer possíveis intrusões. É o método mais simples e efetivo para detectar ataques conhecidos, porém pode ser inefetivo na detecção de ataques desconhecidos ou até mesmo ataques variantes de ataques conhecidos. Além disso, requer que a base de dados de assinaturas esteja sempre atualizada. A detecção baseada em ano-

malias (AD) compara os eventos detectados com o comportamento normal ou esperado, que é obtido através da monitoração de atividades regulares. Qualquer evento que tenha um comportamento diferente do esperado será considerado uma anomalia. Esta abordagem é efetiva tanto para ataques conhecidos quanto para ataques desconhecidos, porém pode apresentar maiores taxas de falso positivo que a SD. Por fim, na SPA, o IDS conhece e rastreia os estados dos protocolos, procurando inconsistências dos eventos detectados com o comportamento esperado. A SPA pode parecer similar a AD, porém a principal diferença é que na SPA a comparação se dá com um modelo já especificado, enquanto na AD o modelo é construído através do monitoramento da rede ou *host*. (LIAO et al., 2013)

Os IDSs também podem ser classificados quanto ao tipo de tecnologia utilizada para a implantação. Por exemplo, o sistema de detecção de intrusão baseado em *host* (*Host-based Intrusion Detection System*, HIDS), coleta informações de um *host* e detecta intrusões que o tenham como alvo. Já um sistema de detecção de intrusão baseado em rede (*Network-based Intrusion Detection System*, NIDS), coleta o tráfego da rede em um segmento específico, analisando a atividade das aplicações e protocolos a fim de reconhecer atividades suspeitas, (LIAO et al., 2013).

DUPONT et al. também classifica os NIDSs quanto à profundidade de inspeção, em baseado em fluxo e baseado em *payload*. Um fluxo pode ser definido como uma coleção de pacotes que compartilham algumas propriedades e são enviados com uma certa frequência. No contexto da rede CAN, a maioria das ECUs se comunicam com um período especifico, assim possibilitando a criação de modelos com os padrões de comunicação para cada ID. Assim, a abordagem baseada em fluxo para redes CAN detecta ataques quando for observado qualquer desvio de comportamento nos fluxos de quadros, como, por exemplo, uma variação na frequência de envio das mensagens de um determinado ID. Já a abordagem baseada em *payload* foca no conteúdo dos quadros. Caso um atacante injete um quadro malicioso que não afete os padrões de comunicação da rede, este ataque seria indetectável através da abordagem baseada em fluxo. A fim de detectar este tipo de ataque, é necessário analisar o conteúdo dos quadros e comparar com um modelo considerado legítimo, seja este construído a partir da observação da rede ou gerado através de especificações.

2.3 ISOLATION FOREST

O *Isolation Forest*, também chamado de *iForest*, é um método de detecção de anomalias que utiliza árvores de decisão na sua estrutura. Para detectar anomalias, o algoritmo tira proveito do fato de que as anomalias são "poucas e diferentes". Como as anomalias tendem a ser isoladas mais próximas da raiz da árvore, as amostras com um menor comprimento de caminho médio pela floresta possuem maior probabilidade de serem anomalias. Entre as vantagens do *iForest*, estão a sua complexidade de tempo linear com uma constante pequena e um baixo requisito de memória. Além disso, tem a capacidade de tratar

problemas com bases de dados extremamente grandes e com alta dimensionalidade. (LIU; TING; ZHOU, 2008)

2.3.1 *iTree*

A *iTree* (*Isolation Tree*) é uma estrutura em árvore na qual cada nó consiste em um nó sem filho, definido como nó externo, ou um nó com um teste e dois nós filhos, definido como nó interno. Um teste consiste em um atributo e um valor de limiar, de modo que o teste divide os dados entre os nós filhos. Na Figura 1, que mostra uma *iTree* de exemplo, os nós 0, 1, 3 e 6 são nós internos, enquanto os nós 2, 4, 5, 7 e 8 são nós externos.

Dada uma amostra de dados $X = x_1, ..., x_n$, para construir uma *iTree*, X é recursivamente dividida, selecionando aleatoriamente um atributo q e um valor de limiar p, até que (i) a árvore alcance sua altura limite, (ii) $|X| = 1$ ou (iii) todas as amostras em X possuam os mesmos valores.

Por definição, o limite de altura da árvore, denotado por l, vale $\lceil log_2 n \rceil$, que é um valor próximo a altura média de uma árvore. O objetivo de construir árvores limitadas é reduzir o tamanho do modelo e limitar o tempo de busca na árvore. Como os pontos de interesse tendem a possuir caminhos médios menores que a média de altura da árvore, esta limitação não impede a detecção de anomalias. No exemplo ilustrado na Figura 1, como $n = 7$, a altura máxima da árvore é $\lceil log_2 7 \rceil = 3$.

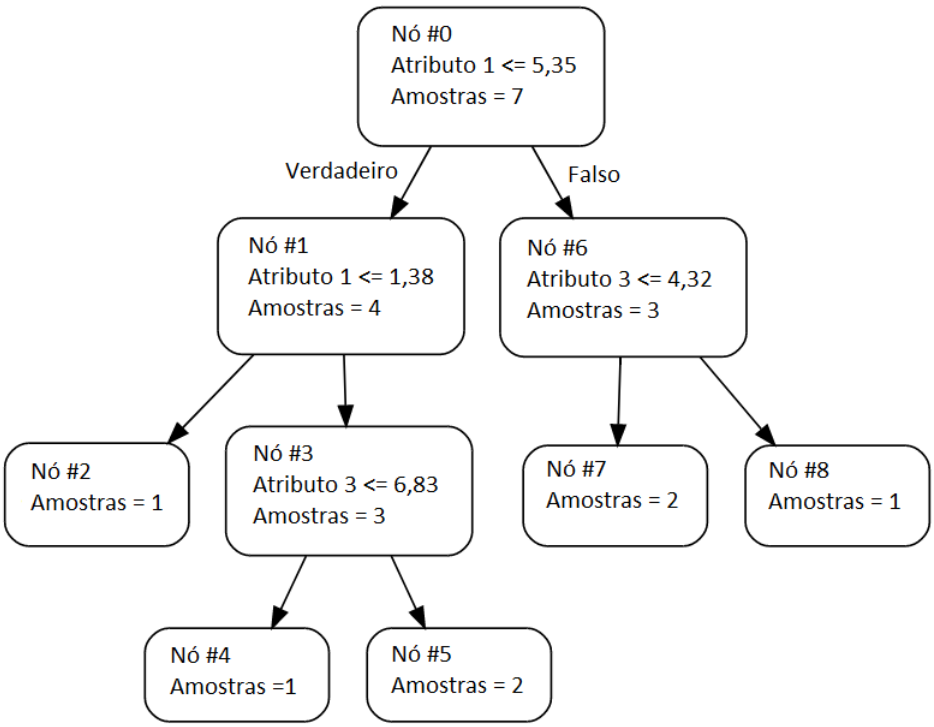

Figura 1 – Um exemplo de uma *iTree*.

2.3.2 Etapa de treinamento

O algoritmo *Isolation Forest* cria um *ensemble* de *iTrees* para um determinado conjunto de dados, considerando duas variáveis: t, o número de árvores para construir, e ψ, o tamanho da subamostra. Para cada uma das t árvores, é realizada uma amostragem de ψ itens do conjunto de dados, que será utilizada na construção da respectiva *iTree*, conforme descrito na subseção 2.3.1. Ambos os parâmetros influenciam diretamente no desempenho de detecção de anomalias e também no tamanho total do modelo gerado.

O tamanho da subamostra controla o tamanho das árvores por 2 motivos: i) o número máximo de nós é de $2\psi - 1$ e ii) o limite de altura da árvore, denotado por l, vale $\lceil log_2 \psi \rceil$. O primeiro motivo se justifica por meio de uma propriedade de árvores binárias, na qual o número de nós externos (folhas) é no máximo o número de nós internos acrescido de 1. Como ψ é a quantidade máxima de folhas que a árvore construída pode ter, o número total de nós não pode ultrapassar $2\psi - 1$. Por fim, o segundo motivo se dá através de uma definição no próprio algoritmo de treinamento da *iTree*, a fim de que a altura da árvore, que implica na performance de buscas, seja limitada.

A quantidade de *iTrees* na floresta é um fator multiplicador no tamanho do modelo gerado. Seja s o tamanho máximo de uma *iTree* com tamanho de subamostra ψ, o tamanho máximo do modelo é st.

Apesar de gerar um modelo maior, aumentar os valores de ambos os parâmetros possibilita um melhor desempenho na detecção de anomalias. Esta melhora, porém, é reduzida quanto maior forem os valores, sendo sugerida de forma empírica uma quantidade de $t = 100$ árvores e $\psi = 256$.

2.3.3 Etapa de avaliação

Nesta etapa, deve ser produzido um *score* que indicará, numericamente, o quanto o dado analisado difere de um dado considerado normal. Esse *score* deve ser gerado a partir do valor do caminho médio $E(h(x))$ da instância analisada x por toda a floresta.

Um único caminho $h(x)$ é calculado contando o número de arestas da raiz até um nó de terminação, considerando o caminho percorrido por x através da *iTree*. Contudo, já que a *iTree* possui um limite de altura, também é, posteriormente, adicionado a $h(x)$ um fator de ajuste. Como mostrado na Figura 1, cada nó possui a informação da quantidade de amostras de X que percorreram a árvore até aquele ponto e este valor é utilizado para determinar a altura esperada da árvore não construída a partir deste ponto, que é o valor do fator de ajuste utilizado.

O *score* é calculado através da Equação (2.1),

$$s(x, \psi) = 2^{-\frac{E(h(x))}{c(\psi)}} \tag{2.1}$$

em que $c(\psi)$ é o comprimento médio do caminho de buscas malsucedidas em árvores de busca binária, ou seja, a altura esperada de uma *iTree* construída a partir de ψ amostras.

$c(\cdot)$ é definida como na Equação (2.2),

$$c(x) = 2H(x-1) - (2(x-1)/x) \tag{2.2}$$

onde $H(\cdot)$ é o número harmônico.

A partir da Equação (2.1), percebe-se que, quando $E(h(x))$ se aproxima do valor de $c(\psi)$, o valor do *score* fica próximo de $0,5$. Já quando $E(h(x))$ se aproxima de 0, o valor de s tende a 1. E, por fim, quando $E(h(x))$ tende a $\psi - 1$, que é o maior valor possível para $E(h(x))$, o valor do *score* tende a 0.

A partir do *score* deve-se inferir se a amostra é uma anomalia ou não. Pode-se dizer que, quando o valor de s é muito próximo de 1, a instância avaliada é certamente uma anomalia, enquanto valores de s abaixo de $0,5$ podem ser consideradas instâncias normais. Apesar disso, para cada problema específico é, geralmente, definido um valor de limiar, considerando as especificidades do problema em questão.

2.4 MÉTRICAS DE AVALIAÇÃO DE UM CLASSIFICADOR BINÁRIO

A fim de facilitar a compreensão do leitor, as métricas de avaliação utilizadas neste trabalho serão brevemente explanadas.

A Tabela 1 mostra os quatro possíveis resultados obtidos para uma classificação binária. Um Verdadeiro Positivo (VP) acontece quando o classificador acerta a previsão de uma amostra positiva, no nosso caso uma anomalia. Em um Falso Positivo (FP), o classificador também considera a amostra como positiva, porém neste caso a amostra é realmente negativa, ou seja, um quadro CAN regular classificado, erroneamente, como anomalia. Um Verdadeiro Negativo (VN) acontece quando o classificador acerta a previsão de uma amostra negativa, que no nosso caso seria um quadro CAN regular. Por fim, no Falso Negativo (FN) o sistema também considera a amostra como negativa, porém na realidade ela é positiva, ou seja, uma anomalia erroneamente classificada como tráfego regular.

As métricas de desempenho são obtidas a partir destes quatro possíveis resultados da classificação, sendo descritas a seguir.

- Acurácia

  A acurácia é definida como $\frac{VP+VN}{VP+VN+FP+FN}$, consistindo basicamente do resultado geral da classificação, ou seja, a taxa total de acerto do classificador.

Tabela 1 – Matriz de confusão.

	Positivo (Real)	Negativo (Real)
Positivo (Previsto)	VP	FP
Negativo (Previsto)	FN	VN

- Precisão

  A precisão mede o quanto o classificador acertou nas suas previsões positivas. É definida como $\frac{VP}{VP+FP}$, ou seja, o total de verdadeiros positivos entre todos os considerados positivos pelo classificador.

- *Recall*

  O *Recall*, também chamado de Taxa de Verdadeiro Positivo (TVP) ou sensibilidade, é definido como $\frac{VP}{VP+FN}$, sendo a taxa de acerto do classificador entre as amostras positivas.

- Taxa de Falso Positivo (TFP)

  A TFP é definida como $\frac{FP}{FP+VN}$, sendo a taxa de erros do classificador entre as amostras negativas.

- F_β *score*

  O F_β *score* é uma medida que combina a precisão com o *recall*, dando mais importância para cada uma dependendo do valor de β. É definido pela equação a seguir:

$$F_\beta = (1 + \beta^2)\frac{pr}{\beta^2 p + r} \tag{2.3}$$

  em que p é a taxa de precisão e r é a taxa de *recall*.

- *Receiver Operating Characteristics* (ROC)

  A curva ROC traça um gráfico entre a TVP e a TFP, variando o limiar de classificação. Para cada limiar, um ponto no gráfico mostra a TVP e a TFP obtidas, assim sendo possivel observar o *tradeoff* entre essas métricas. Esta curva permite ter uma ideia de quão bom é o classificador utilizado, ou seja, se consegue realizar uma boa separação da base de dados, obtendo uma alta TVP e uma baixa TFP. Um indicador bastante utilizado é também a *Area Under Receiver Operating Characteristics* (AUROC), que, numericamente, mede a qualidade de um classificador.

2.5 CONSIDERAÇÕES FINAIS

Neste Capítulo foram abordadas características físicas e lógicas da rede CAN, que possui vulnerabilidades intrínsecas as quais este trabalho visa mitigar. Foram descritas algumas classificações de sistemas de detecção de intrusão e suas diferenças, a fim de situar o leitor acerca do tipo de solução este trabalho propõe. O algoritmo de detecção selecionado, o *Isolation Forest*, foi analisado a fundo, sendo explicados os parâmetros, a estrutura, o cálculo do *score* e todas as informações necessárias a compreensão do mesmo. Por fim, foram apresentadas as métricas de avaliação de desempenho de classificadores binários utilizadas neste trabalho, a fim de demonstrar a efetividade da solução apresentada.

3 TRABALHOS RELACIONADOS

Existem diversos trabalhos que propõem um sistema de detecção de intrusão para CAN, porém, no melhor do nosso conhecimento, nenhum apresenta um IDS de tempo real e baseado em anomalias que seja capaz de realizar a detecção antes do fim da transmissão do quadro CAN. Este requisito temporal é necessário para que o sistema seja capaz de impedir que quadros anômalos sejam enviados pela rede, como será discutido em mais detalhes no Capítulo 4.

(KOYAMA et al., 2019) propõe uma detecção de anomalias baseada nos intervalos entre as mensagens e os seus conteúdos, alcançando uma acurácia média de $99,99\%$, considerando os quatro veículos testados em seus experimentos. Como o método detecta anomalias em tuplas de três mensagens, a solução é incapaz de indicar qual mensagem é anômala. O tempo de resposta no cenário de teste foi em média $450\mu s$, portanto sendo também incapaz de atender o *deadline* requerido.

(OLUFOWOBI et al., 2019) detecta comportamentos anômalos analisando sequências de mensagens. O artigo considera uma abordagem baseada em técnicas de detecção de mudança, utilizando o algoritmo *Cumulative Sum* (CUSUM) adaptativo para detectar mudanças estatísticas e intrusões no fluxo de mensagens do barramento CAN. Nos cenários de teste do experimento para validar a abordagem proposta por (OLUFOWOBI et al., 2019), a solução apresentou tempos médios de detecção na ordem de dezenas de milissegundos.

(TAYLOR; JAPKOWICZ; LEBLANC, 2015) adapta uma técnica baseada em fluxos utilizada em sistemas de controle industriais para o cenário automotivo. É construído um modelo de *One-Class Support Vector Machine* (OCSVM), utilizando estatísticas extraídas dos fluxos, principalmente informações de frequência de mensagens. São obtidas AUROC que variam entre $0,96$ e $0,99$, dependendo do cenário de teste avaliado. Apesar de obter um bom resultado, não é disponibilizada nenhuma informação referente ao tempo de detecção.

(MARTINELLI et al., 2017) propõe um método de detecção de ataques utilizando algoritmos de classificação *fuzzy*. São utilizados os oito bytes de dados que formam o (*payload*) do quadro CAN como atributos para o treinamento dos classificadores *fuzzy* selecionados. Apesar de alcançar precisões que variam entre 0,85 e 1, dependendo do cenário de ataque e do algoritmo utilizado, o tempo de detecção no cenário de testes não foi informado.

(KANG; KANG, 2016) propõe uma técnica de detecção de intrusão que utiliza uma *Deep Neural Network* (DNN). Está técnica alcança 99,9% de taxa de acurácia no cenário avaliado e pode responder em tempo real. Porém, como o requisito temporal utilizado é na ordem de milissegundos, a solução não é capaz de responder antes do fim da transmissão completa de um quadro CAN. Além disso, uma DNN demanda um alto poder computacional, requisito que pode ser pouco atrativo para a indústria automotiva.

(TAYLOR; LEBLANC; JAPKOWICZ, 2016) utiliza uma Recurrent Neural Network (RNN) na construção de um IDS. Para cada identificador CAN utilizado pelo sistema automotivo, a RNN é treinada para prever o conteúdo do próximo quadro. A solução detecta anomalias de forma independente para o fluxo de quadros de cada identificador. Para a maioria dos identificadores e tipos de ataque avaliados, a solução de (TAYLOR; LEBLANC; JAPKOWICZ, 2016) obtém uma AUROC maior que $0,99$.

(ARAUJO-FILHO, 2018) compara o desempenho de detecção de anomalias na rede CAN entre os algoritmos OCSVM e *Isolation Forest*, utilizando como atributos os bytes do campo de dados do quadro CAN. No cenário de testes avaliado, foi obtida uma acurácia média de $99,72\%$ com o *Isolation Forest* e $97,06\%$ com o OCSVM. Como os testes foram realizados num sistema operacional padrão, não adequado para sistemas de tempo real, não foi possível medir o tempo de detecção de um único quadro.

(SEO; SONG; KIM, 2018) propõem o *GAN-based IDS* (GIDS), que utiliza o modelo de aprendizagem profunda *Generative Adversarial Nets* (GAN). São construídos dois modelos, um que é treinado para ataques conhecidos e outro que é treinado para ataques desconhecidos. Para treinar os modelos, foram utilizados dados capturados de um Hyundai Sonata, utilizando uma Raspberry Pi 3 conectada ao barramento CAN através da porta OBD. O GIDS obteve acurácia média de 100% para o primeiro modelo e 98% para o segundo modelo. Uma de suas características citadas pelo autor é a capacidade de realizar detecção de intrusão em tempo real, porém nenhum requisito temporal é definido.

(WEBER et al., 2018) propõem uma abordagem híbrida, utilizando tanto a detecção baseada em anomalias quanto à detecção baseada em especificação. O IDS proposto por (WEBER et al., 2018) realiza uma checagem estática no seu estágio inicial, verificando, por exemplo, se um determinado valor em uma mensagem específica é válido. Após a checagem estática, é realizada a detecção de anomalias, através do algoritmo *Lightweight On-Line Detector of Anomalies* (LODA), que analisa uma sequência de mensagens.

(ABBOTT-MCCUNE; SHAY, 2016) propõem uma solução capaz de evoluir para um IPS. A abordagem de (ABBOTT-MCCUNE; SHAY, 2016) consiste em checar se o identificador de um quadro CAN corresponde a um dos identificadores pré-programados na ECU, ou seja, se a ECU realmente tem a permissão de enviar mensagens com este identificador, e se a ECU está transmitindo dados no instante da checagem, portanto detectando se um nó malicioso está tentando se passar por outro nó. Da mesma forma, também é verificado se a ECU está tentando enviar quadros com identificadores diferentes dos que estão pré-programados. A fim de verificar estas inconsistências, a solução de (ABBOTT-MCCUNE; SHAY, 2016) requer um módulo adicional por nó CAN na rede. Como consiste apenas em simples operações lógicas, a solução identifica as inconsistências assim que o campo de arbitração, que contém o identificador, é enviado. Apesar de responder rápido o suficiente para impedir a transmissão de quadros maliciosos na rede, a abordagem não permite identificar ataques baseados no conteúdo das mensagens. Portanto, não é capaz

Tabela 2 – Comparação entre trabalhos.

Trabalho	Dados utilizados	Modelo	Tempo de detecção de um quadro
(KANG; KANG, 2016)	CAN ID Payload	Aprendido	$>$ 1ms
(KOYAMA et al., 2019)	Intervalo entre quadros Payload	Aprendido	Não aplicável
(SEO; SONG; KIM, 2018)	CAN ID Payload	Aprendido	Não disponível
(ABBOTT-MCCUNE; SHAY, 2016)	CAN ID	Especificado	$\leq 10^{-6}$s
(TAYLOR; LEBLANC; JAPKOWICZ, 2016)	Payload	Aprendido	Não disponível
(OLUFOWOBI et al., 2019)	Frequência Payload	Aprendido	$>$1ms
(MARTINELLI et al., 2017)	Payload	Aprendido	Não disponível
(TAYLOR; JAPKOWICZ; LEBLANC, 2015)	Frequência Payload	Aprendido	Não disponível
(ARAUJO-FILHO, 2018)	Payload	Aprendido	Não disponível
(WEBER et al., 2018)	Payload	Híbrido	Não disponível

de detectar ataques onde a ECU comprometida é justamente a que tem a permissão de mandar mensagens com o identificar do quadro malicioso.

A Tabela 2 compara os trabalhos mencionados anteriormente quanto aos dados utilizados pela solução, se o modelo é aprendido através de uma base de dados ou especificado através de documentações do sistema, e quanto ao tempo de detecção de um quadro. Além disso, a tabela mostra que muitos autores não mencionam o tempo de detecção das soluções propostas por eles, que é uma informação importante a fim de realizar alguma ação quando um ataque é detectado.

4 METODOLOGIA

Neste Capítulo são descritas as técnicas e métodos necessários para a construção de um IPS para CAN, desde a construção de um modelo em linguagem de alto nível até o método de corrupção do quadro CAN.

4.1 VISÃO GERAL

A metodologia proposta consiste em 4 etapas: i) Aquisição de dados, ii) Treinamento do modelo, iii) Exportação do modelo, iv) Integração com módulo injetor de erros. Os dados trafegados na rede CAN são utilizados no treinamento de uma *Isolation Forest*, que terá sua estrutura exportada para código de descrição de hardware, a fim de que a detecção seja realizada numa FPGA. Por fim, é possível integrar o detector de anomalias em FPGA com um módulo gerador de erros na rede CAN, assim impedindo que um quadro anômalo seja corretamente transmitido na rede.

Dado que o treinamento do modelo não possui restrições de tempo ou memória, é sugerido que seja realizado utilizando uma linguagem de programação de alto nível que possua bibliotecas de software específicas de aprendizagem de máquina. Assim, será reduzido o tempo dispendido na implementação, sem prejuízo no desempenho do sistema final, que apenas utilizará o modelo criado em software.

É importante ressaltar que o formato dos dados trafegados na rede CAN pode variar de acordo com o fabricante e modelo do automóvel. Sendo assim, através desta metodologia é possível produzir um IPS específico para o modelo do automóvel cujos dados foram coletados, não sendo garantida eficácia do sistema criado para outros casos.

4.2 AQUISIÇÃO DE DADOS

A aquisição de dados se dá através do uso de um módulo *sniffer*, que escuta a rede e guarda os dados trafegados. É possível realizar este procedimento através da porta OBD, que está presente na maioria dos automóveis comerciais.

Para fins de treinamento do modelo do *Isolation Forest*, descrito na subseção 2.3.2, são utilizados dados do funcionamento normal do automóvel, ou seja, não são incluídos dados de ataque. Isto facilita o treinamento do modelo, de forma que não é necessário ter conhecimento prévio de possíveis ataques, característica de um IDS/IPS baseado em anomalias. Apesar disso, é possível se utilizar de dados de um ataque específico a fim de determinar um limiar de detecção ótimo para este tipo de ataque, assim melhorando a eficácia do IDS na detecção do mesmo. Os tipos de dados presentes em cada parte da base de dados estão descritos na Tabela 3.

Tabela 3 – Tipos de dados.

Base de dados	Dados legítimos	Dados de ataque
Treinamento	X	
Validação	X	X
Teste	X	X

4.3 TREINAMENTO DO MODELO

Utilizando a base de dados de treinamento, que contém apenas dados de funcionamento normal do automóvel, é treinado um modelo de uma Isolation Forest em linguagem de alto nível, como Python. O treinamento desse modelo requer como entrada dois parâmetros: a quantidade de árvores da floresta t e o tamanho da sub-amostra ψ a ser utilizada, ou seja, o tamanho máximo das árvores. Ambos os parâmetros apresentam um tradeoff: quanto maior a quantidade de árvores e maior o seu tamanho, melhor a detecção de anomalias. Para determinar os valores a serem utilizados, deve-se levar em conta a capacidade da FPGA que será utilizada, assim utiliza-se a melhor configuração possível que se adéque à placa alvo. A fim de facilitar o cálculo realizado na FPGA, é aconselhável utilizar um número de árvores que seja uma potência de 2, fazendo com que a operação de divisão, que existe no cálculo da média, possa ser traduzida para apenas operações de *shift*.

A Figura 2 mostra as etapas da criação do modelo. Com o modelo treinado, é possível obter um score de anomalia para qualquer quadro CAN, porém ainda é necessário definir o limiar que decidirá se um quadro é anômalo ou não. Este limiar é definido a partir do resultado obtido com os dados de validação; são testados vários limiares e é selecionado o que obtiver melhores métricas de detecção, priorizando as métricas que forem mais compatíveis com a necessidade do sistema final. Por fim, o modelo e o limiar definido são testados com a base de teste, que contém uma maior quantidade de dados, a fim de verificar a eficácia do modelo com dados que não foram treinados nem otimizados no sistema final, garantindo uma generalização para o modelo.

A separação das bases de dados em treinamento, validação e teste não é suficiente para uma boa avaliação de um modelo. Por exemplo, os dados selecionados para o conjunto de

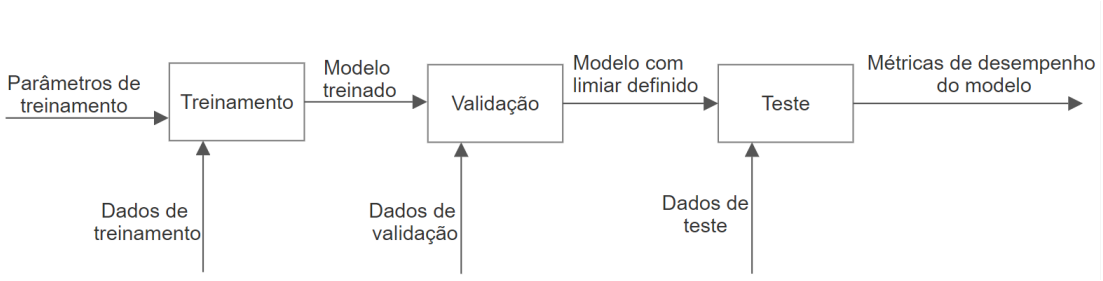

Figura 2 – Diagrama de blocos para a criação do modelo.

testes podem ter sidos “por acaso” os que resultariam em uma melhor taxa de acerto no conjunto de testes. É interessante a realização de uma validação cruzada, que possibilita uma avaliação do desempenho do sistema utilizando dados diferentes para cada um dos conjuntos definidos.

4.4 EXPORTAÇÃO DO MODELO

Com o modelo gerado em linguagem de alto nível, é necessário exportar os dados da floresta para utilização durante a execução do algoritmo em hardware. Esta exportação pode variar de acordo com as ferramentas utilizadas para a implementação do *Isolation Forest*, podendo ser necessário realizar modificações na biblioteca de software utilizada a fim de obter a estrutura interna completa das árvores. Esta estrutura deve ser codificada e será armazenada em memória para uso na implementação em linguagem de descrição de hardware, assim obtendo acesso facilmente a estrutura da árvore. Uma possível codificação está descrita nas Tabelas 4 e 5.

Para realizar a detecção de anomalias em tempo real, é necessário implementar um algoritmo para a obtenção do *score* de anomalia, que utiliza o modelo treinado da *Isolation Forest*. Como vimos na Seção 2.3, o score de anomalia é obtido a partir do comprimento do caminho que o dado analisado percorre na árvore e é definido pela Equação 2.1. A implementação em hardware deste algoritmo é baseada no *design* proposto em (BUSCHJÄGER; MORIK, 2017), sendo melhor detalhada na Seção 5.4. Assim como a implementação de (STRUHARIK, 2011), o *design* proposto neste trabalho possui um limite de tempo para a execução. Como cada nó da árvore leva um ciclo de relógio para ser avaliado, o tempo máximo que uma amostra x leva para percorrer uma árvore de altura h é hf^{-1}, em que f é a frequência do relógio da FPGA. A Tabela 6 mostra os tempos de execução estimados para diferentes configurações de f e h.

A fim de facilitar o cálculo a ser implementado em hardware, são feitas algumas considerações: (i) não é necessário utilizar o *score* definido pelo algoritmo do *iForest*, podendo-se utilizar um *score* derivado; (ii) o valor de $c(\psi)$ é uma constante; (iii) como evidenciado na Tabela 5, o fator de ajuste pode ser armazenado na estrutura da árvore, evitando o cálculo do mesmo em hardware; (iv) o número de árvores utilizadas t também será uma constante no nosso cálculo do *score* de anomalia.

Um *score* derivado s_d, obtido a partir do *score* padrão s, pode ser utilizado na implementação em hardware. Este score pode ser definido através da Equação (4.1), tal que

$$s_d = -c(\psi) log_2(s(x, \psi)) = E(h(x)) \tag{4.1}$$

sendo essencialmente a média do caminho médio percorrido pelo dado analisado entre as árvores da floresta, acrescido do seu fator de ajuste. Como as árvores utilizadas no

Tabela 4 – Formato dos dados para nós internos.

Nome do campo	Descrição	Tamanho (bits)
Feature	Atributo a ser avaliado	3
Is leaf node	Valor booleano	1
Threshold	Valor do limiar de detecção	8
Reserved	Bits reservados	4
Left	Endereço do nó filho esquerdo	12
Right	Endereço do nó filho direito	12

Tabela 5 – Formato dos dados para nós externos.

Nome do campo	Descrição	Tamanho (bits)
Reserved	Bits reservados	3
Is leaf Node	Valor booleano	1
Reserved	Bits reservados	12
Adjustment	Fator de ajuste	24

Tabela 6 – Exemplos de tempos de execução para alguns valores de h e f.

h	$10MHz$	$25MHz$	$50MHz$	$100MHz$
8	$0,8\mu s$	$0,32\mu s$	$0,16\mu s$	$0,08\mu s$
16	$1,6\mu s$	$0,64\mu s$	$0,32\mu s$	$0,16\mu s$
32	$3,2\mu s$	$1,28\mu s$	$0,64\mu s$	$0,32\mu s$

algoritmo são limitadas, o fator de ajuste é necessário para levar em conta a altura de uma subárvore não construída, sendo este termo já incluído em $h(x)$.

Utilizando-se da consideração (iv), é possível simplificar mais ainda o cálculo realizado em hardware, obtendo outro *score* derivado, o qual iremos chamar de *score* modificado, que será efetivamente utilizado na solução e é descrito pela Equação 4.2, tal que

$$s_m = s_d t = \sum_{y=1}^{t} h(y). \tag{4.2}$$

A fim de utilizar o *score* modificado s_m, o valor de limiar obtido na etapa de treinamento do modelo deve ser transformado, utilizando as Equações (4.1) e (4.2). O *score* modificado permite que a comparação realizada em hardware seja apenas da soma dos caminhos $\sum_{y=1}^{t} h(y)$ com o valor de limiar s_{th}, já modificado através da Equação (4.2), sendo desnecessário qualquer outro cálculo em hardware. O resultado desta comparação será o resultado do sistema, ou seja, se a amostra avaliada será considerada uma anomalia ou não.

4.5 INTEGRAÇÃO COM MÓDULO INJETOR DE ERROS

Conforme visto na seção 2.1, é possível introduzir erros na rede durante a transmissão de um quadro, efetivamente corrompendo-o. Um possível erro a ser utilizado para esse fim é o erro de *bit stuffing*, que ocorre quando seis bits iguais são recebidos consecutivamente. Essa abordagem de corrupção através do erro de *bit stuffing* já foi sugerida por (GIANNOPOULOS; WYGLINSKI; CHAPMAN, 2017) e (ARAUJO-FILHO, 2018).

Para utilizar esta técnica, é necessário detectar o ataque antes da transmissão completa do quadro CAN. Através do formato do quadro CAN descrito na subseção 2.1.3.1, pode-se perceber que existem 5 campos de dados após a transmissão dos dados, somando 25 bits. Segundo a própria especificação do protocolo CAN (BOSCH, 1991), o último bit do campo EOF é tratado como *don't care*, assim restando efetivamente 24 bits após a transmissão do *payload*. Como o erro de *bit stuffing* requer 6 bits para ocorrer, é necessário detectar o ataque num tempo menor que a transmissão de 18 bits pela rede CAN. Assim, o prazo para detecção de ataques é de apenas $36\mu s$, considerando uma rede CAN a 512Kbps, velocidade mais comumente utilizada. Este cálculo considera o pior caso de transmissão, no qual não existe nenhuma sequência de 6 ou mais bits repetidos nos dados a serem enviados pela rede CAN nos campos de dados restantes, o que adicionaria mais bits trafegados pela rede, justamente devido ao *bit stuffing*.

Considerando os tempos de detecção apresentados na Tabela 6, este requisito temporal será completamente atendido. Em uma rede CAN a 1Mbps, a maior velocidade permitida pelo protocolo, o *deadline* seria de $18\mu s$, assim sendo atendida até mesmo pela configuração mais lenta apresentada na Tabela 6, cujo tempo de detecção é de $3,2\mu s$.

Introduzir outros tipos de erros na rede CAN, como um *Form Error* no campo EOF, a fim de corromper um quadro malicioso também é, teoricamente, possível. Apesar disso, a proposição do uso de outros erros, além do erro de *bit stuffing*, como meio de impedir a transmissão de um quadro não foi encontrado na literatura pelo autor deste trabalho.

A Figura 3 apresenta um diagrama de blocos de um sistema IPS para CAN, integrando de forma simples o IDS construído através da metodologia apresentada neste Capítulo e um módulo injetor de erros na rede CAN. O módulo IDS lê o quadro trafegado na rede e, caso detecte alguma anomalia, manda uma notificação de intrusão ao módulo de injeção de erros, que corrompe o quadro através da transmissão dos bits adequados.

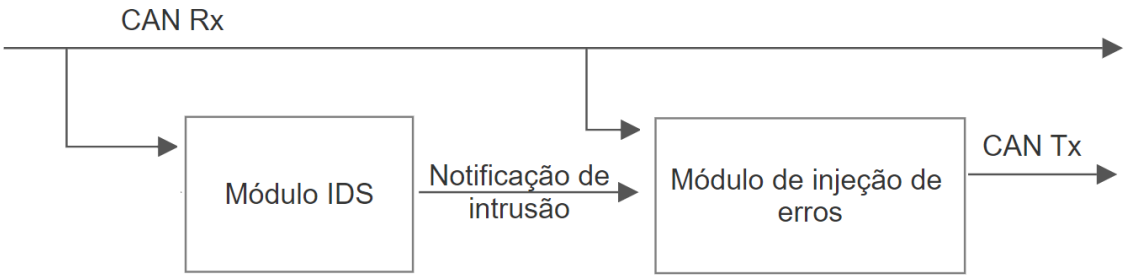

Figura 3 – Diagrama de blocos para um possível IPS.

4.6 CONSIDERAÇÕES FINAIS

Neste capítulo foi descrito o procedimento necessário a criação de um sistema de detecção de intrusão de tempo real e baseado em anomalias para a rede CAN automotiva. Inicialmente, são adquiridos os dados necessários ao treinamento, validação e teste do modelo. Utilizando apenas dados de tráfego legítimo da rede, é treinada uma *Isolation Forest*. A fim de utilizar um limiar de detecção ótimo e verificar o desempenho do modelo, são utilizados dados legítimos e também dados de ataque nas etapas de validação e treinamento. O modelo treinado é utilizado em uma implementação do algoritmo de obtenção do *score* de anomalia do *Isolation Forest* em FPGA. Essa implementação em FPGA garante um tempo máximo para a obtenção da classificação de uma amostra, assim evidenciando o aspecto *real-time* da solução apresentada. Por fim, é sugerida a integração do detector de anomalias criado com um módulo injetor de erros na rede CAN, a fim de impedir a transmissão de um quadro considerado anômalo.

5 EXPERIMENTOS

Neste Capítulo é descrito o experimento realizado a fim de validar a metodologia proposta, incluindo os dados e ferramentas utilizadas, o treinamento da *Isolation Forest* e sua exportação para linguagem de hardware.

5.1 AMBIENTE E CONFIGURAÇÃO DO EXPERIMENTO

Os dispositivos utilizados no experimento foram: (i) uma Raspberry Pi 3 Model B (UPTON; HALFACREE, 2014), (ii) uma placa Terasic DE2-115 (TERASIC, 2012), (iii) um laptop com um processador Intel i5 2,5GHz e com 8GB de RAM, e (iv) um analisador lógico Saleae Logic Pro 8.

No laptop, a *iForest* foi treinada e testada usando a linguagem de programação Python e a biblioteca de aprendizagem de máquina *scikit-learn* (SCIKIT-LEARN, 2018). O modelo treinado foi extraído por meio da adição de rotinas no código da biblioteca, possibilitando o acesso a estrutura interna da mesma. O modelo consiste apenas nas *iTrees* treinadas, tendo sido extraídas sob o formato definido nas Tabelas 4 e 5.

O módulo para detecção de anomalias que utiliza os dados da árvores já treinadas foi desenvolvido em linguagem Verilog, no software Quartus. Através deste software, também foi possível testar o módulo desenvolvido quanto à sua corretude, através do ambiente de simulação.

A fim de testar o módulo desenvolvido em um ambiente físico, dados de teste foram carregados na Raspberry Pi, que os transmite através de portas *General Purpose Input Output* (GPIO) para a DE2-115, onde está carregado o módulo de detecção de anomalias. O resultado, ou seja, se a amostra foi considerada uma anomalia ou não, é enviado, assim que disponível, para a Raspberry, que armazena esta informação em arquivo. Além disso, a fim de medir o tempo gasto pela FPGA para avaliar os dados de um quadro CAN, foi utilizado o analisador lógico. Esta configuração do experimento pode ser verificada na Figura 4.

Obter um IDS que detecta os ataques em tempo hábil é suficiente para demonstrar a possibilidade de construção de um IPS, devido a simplicidade de uma possível integração com o módulo injetor de erros. Sendo assim, um experimento com os dois módulos integrados não se faz necessário.

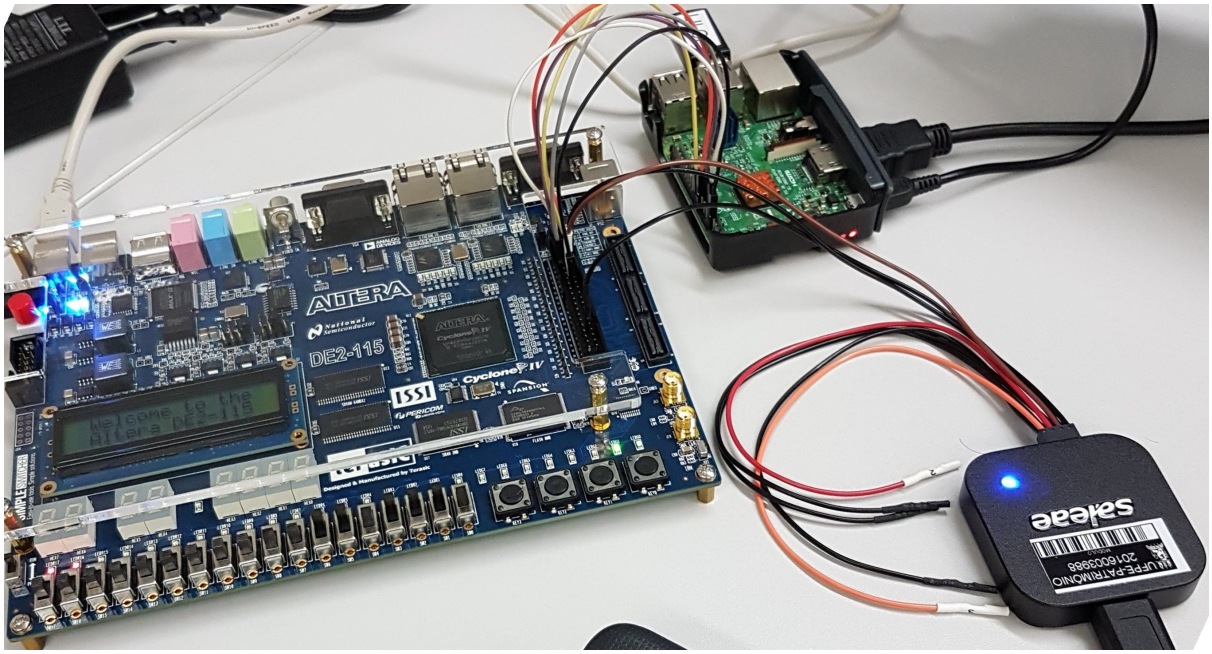

Figura 4 – Configuração do experimento.

5.2 A BASE DE DADOS

A obtenção de uma base de dados de ataques à rede CAN de um automóvel é uma tarefa difícil, considerando que a indústria automotiva não costuma compartilhar este tipo de material. Além disso, a criação de uma base de dados deste tipo requer, no mínimo, um automóvel de testes no qual fosse possível, além de registrar os dados trafegados na rede, injetar quadros de ataque, assim tornando esta alternativa inviável economicamente.

A base de dados utilizada foi obtida do *Hacking and Countermeasure Research Lab* (KIM, 2018), que permite o uso dos dados gratuitamente para fins acadêmicos. Este *dataset* também foi utilizado em (SEO; SONG; KIM, 2018), consistindo em dados coletados de um Hyundai Sonata. São disponibilizadas informações de *timestamp*, identificador, valor do campo DLC e oito bytes do *payload* para alguns milhões de quadros trafegados na rede CAN, além do rótulo de quadro intruso ou quadro normal. O *dataset* contém dados para 4 tipos de ataque: *DoS*, *Fuzzy*, *Spoofing Gear* e *Spoofing RPM*.

Um ataque DoS consiste na injeção de quadros de alta prioridade na rede, ou seja, com valores baixos no campo ID do quadro CAN, normalmente sendo utilizados todos os bits dominantes. O propósito dessa injeção de quadros é impedir que o tráfego legítimo da rede seja transmitido, assim podendo causar indisponibilidade em funcionalidades críticas do sistema. Como o método proposto por este trabalho consiste na corrupção do quadro CAN para impedir que um ataque seja bem sucedido, o ataque DoS não será considerado, pois, mesmo corrompendo o quadro de ataque, ainda será possível que o ataque ocupe o meio de transmissão, causando a indisponibilidade.

O ataque *Fuzzy* é, normalmente, uma forma para um atacante explorar a rede e buscar

vulnerabilidades. Este ataque consiste no envio de quadros com dados aleatórios a fim de encontrar comportamentos inesperados no sistema. Através das informações acerca do sistema obtida com o ataque *Fuzzy*, o atacante pode elaborar ataques mais específicos, normalmente mais danosos ao sistema.

Em um contexto de redes IP, um ataque de *spoofing* consiste em se passar por um nó específico da rede, enviando pacotes com endereço de remetente do nó personificado. No contexto de redes CAN, na qual não existe endereço do remetente, o atacante falsifica o campo ID, que tem mais relação com a informação que está sendo enviada do que com o emissor. Este tipo de ataque permite ao atacante enganar o sistema com informações falsas, assim podendo gerar graves consequências, principalmente em um sistema crítico.

Para cada um desses 3 ataques que serão analisados, será utilizado um conjunto de dados diferente, gerando também 3 modelos de *iForest*, um para cada ataque. Dentre o conjunto de dados utilizado em cada cenário de ataque, também é realizada outra divisão dos dados, em treinamento, validação e teste, assim como descrito na Seção 4.2.

5.3 TREINAMENTO DO MODELO

Apesar da base de dados conter várias outras informações acerca dos quadros trafegados na rede, os únicos atributos que serão utilizados na criação do modelo são os oito bytes presentes no *payload*. Essa escolha nos permite alcançar taxas de detecção comparáveis com o estado da arte, utilizando os atributos com maior importância, além de gerar um modelo pequeno, que possa ser exportado para a FPGA. Além disso, não é realizado nenhum pré-processamento, sendo os atributos utilizados no treinamento da *iForest* exatamente os bytes de dados do quadro CAN, assim contribuindo para um menor tempo de resposta do sistema final.

Inicialmente, foram utilizados altos valores para os parâmetros do *iForest*, que são o número de árvores t e o tamanho da subamostragem ψ. Porém, logo foi percebido que tais valores não permitiriam a exportação do modelo para a FPGA, por limitação do total de elementos lógicos disponíveis. Foram testadas diversas configurações de valores, sendo $t = 16$ e $\psi = 10.000$ a melhor configuração, considerando as métricas de desempenho analisadas, que caberia na FPGA, portanto a configuração a ser utilizada. As métricas utilizadas para avaliar o desempenho do modelo foram: AUROC, precisão e *recall*, obtidas utilizando os dados de teste. A definição do limiar de detecção, que decide o *score* limite para considerar uma amostra como anomalia, também se baseia nestas métricas, porém obtidas utilizando os dados de validação.

A Tabela 7 mostra a precisão média e o desvio-padrão obtidos na validação cruzada para o *dataset* do ataque *Fuzzy*, utilizando algumas configurações de parâmetros que poderiam ser exportadas para a FPGA. Estas métricas foram obtidas utilizando um limiar que otimiza o F_1 *score* do modelo na base de validação. Utilizando $\psi = 256$ e $t = 100$, que são valores sugeridos em (LIU; TING; ZHOU, 2008), obtém-se uma precisão de $0,79$, que

Tabela 7 – Precisão obtida com diversos parâmetros.

ψ	t	Precisão	Desvio-padrão
256	32	0,7805	0,0549
256	64	0,8083	0,0449
256	100	0,7981	0,0372
4000	16	0,8818	0,0159
4000	32	0,8981	0,0145
10000	16	0,9136	0,0090

é um valor inferior aos obtidos por outras soluções. Neste mesmo valor de ψ, percebe-se que a quantidade de árvores t não consegue influenciar tanto na precisão obtida, assim sendo necessário aumentar o valor de subamostragem. Neste cenário, a configuração com $t = 16$ e $\psi = 10.000$ obteve a melhor precisão. Não é interessante utilizar configurações com valores de t muito baixos, pois o valor do *score* de anomalia correria maior risco de não convergir para seu valor médio, considerando o aspecto aleatório na própria criação da *iTree*.

Por fim, como a biblioteca não disponibiliza a estrutura interna do modelo do *iForest* treinado, foi necessário implementar rotinas que permitissem o acesso a essa estrutura, assim permitindo a criação de arquivos de memória com o formato descrito nas Tabelas 4 e 5. Esse código, adicionado à implementação da *Isolation Forest* pela biblioteca de software *scikit-learn*, está disponível no Apêndice B.

5.4 DETECTANDO ANOMALIAS NA FPGA

A placa utilizada no experimento foi a DE2-115 (TERASIC, 2012), que possui um oscilador de 50MHz, 2 conectores com pinos de GPIO disponíveis para uso e uma Altera Cyclone® IV 4CE115 FPGA, onde será carregado o código de descrição de hardware implementado. A implementação do IDS necessitou de 91% dos 114.480 elementos lógicos disponíveis na FPGA, conforme o *report* obtido do software Quartus exibido na Figura 5.

A implementação em hardware do algoritmo para obtenção do valor do caminho percorrido por uma amostra em uma *iTree* é mostrada na Figura 6, obtida através do software Quartus. Os multiplexadores à esquerda selecionam o atributo que será avaliado, dependendo do valor do campo *Feature*, presente na descrição de cada nó interno da *iTree*, conforme Tabela 4. Os registradores à direita armazenam os valores do endereço do nó atual e a sua profundidade na árvore, ou seja, a quantidade de arestas da raiz até o nó atual. As entradas do módulo são apenas as informações do nó atual, os atributos da amostra sendo avaliada, um sinal de *clock* e um sinal de *enable*. Por fim, as saídas são o endereço do próximo nó a ser avaliado, que será utilizado para selecionar na memória os dados referentes ao próximo nó a ser percorrido, e a profundidade do nó, já acrescida do

Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	103,923 / 114,480 (91 %)
Total registers	361
Total pins	70 / 529 (13 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Figura 5 – Utilização de recursos da FPGA.

fator de ajuste, gerando o comprimento do caminho $h(x)$.

Como o parâmetro do número de árvores utilizadas é $t = 16$, são utilizados 16 módulos para computar o caminho percorrido por uma amostra, descritos no parágrafo anterior, sendo um para cada árvore. Cada um desses módulos está ligado a uma memória diferente, que armazena a estrutura de uma árvore pertencente à floresta. Quando todas as árvores chegam em um nó folha, identificado através do campo *Is leaf node*, presente no formato sugerido para as árvores descrito nas Tabelas 4 e 5, os caminhos $h(x)$ são somados e o resultado é comparado ao limiar modificado. Caso o limiar seja maior que o somatório dos caminhos $h(x)$, a amostra avaliada é considerada uma anomalia.

O código, em linguagem Verilog, da implementação supracitada é disponibilizado na íntegra no Apêndice A, sendo também disponibilizados scripts em Python que automatizaram a criação de parte do código Verilog no Apêndice B. Cada *iTree* presente no modelo criado em Python gerou um arquivo ".list"contendo os dados de todos os nós da árvore, conforme formato disponibilizado nas Tabelas 4 e 5. Através de um script Python, são gerados arquivos Verilog que implementam um módulo de memória que carrega os valores presentes nos arquivos ".list". Cada um dos módulos de memória é ligado a um módulo que implementa o cálculo do *score* de anomalia (Figura 6), assim possibilitando a obtenção do *score* para cada árvore do modelo importado. Por fim, também com auxilio de um script em Python, foi gerado o arquivo Verilog do sistema final, que é capaz de agregar os valores de *score* de cada árvore e comparar com um limiar previamente definido.

Esta implementação foi validada quanto à sua corretude através do teste descrito na Seção 5.1, no qual uma Raspberry Pi envia para a FPGA, onde o código foi carregado, todo o conjunto de teste utilizado, que contém, aproximadamente, três milhões de amostras. As amostras de teste são enviadas uma por vez, esperando a resposta do FPGA antes de enviar a próxima. Todas as respostas da FPGA recebidas pela Raspberry Pi são armazenadas em um arquivo, a fim de que possa ser comparado ao arquivo que contém as respostas do algoritmo de detecção implementado em Python. Como os arquivos pos-

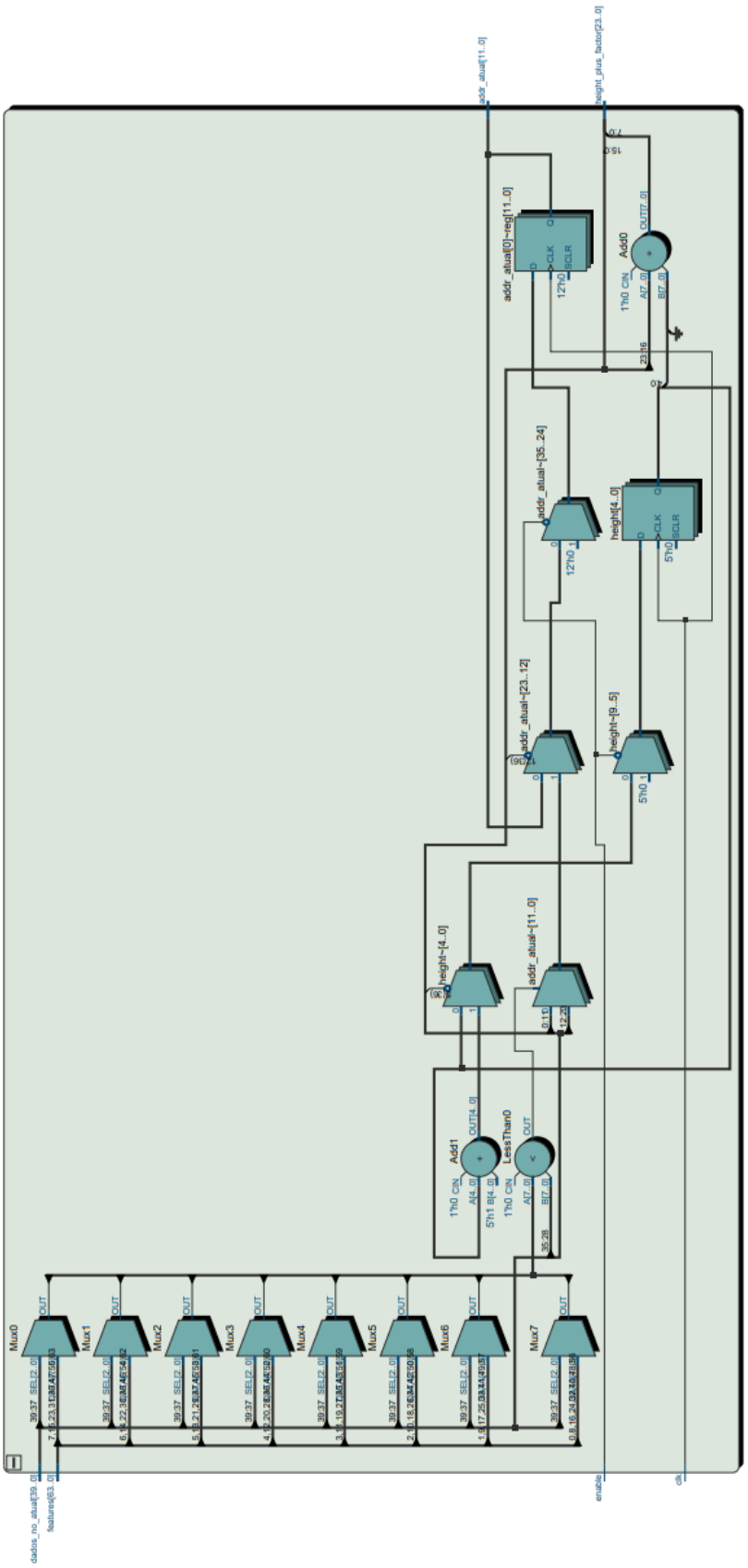

Figura 6 – Implementação de uma *Isolation Tree* em hardware.

suem todas as respostas iguais, ou seja, são idênticos, a implementação do algoritmo de detecção em hardware foi considerada correta.

Em relação ao tempo de execução, conforme descrito na Seção 4.4, o algoritmo possui um tempo máximo para sua execução dependendo da frequência f e altura máxima das árvores h_{max}. Como a amostra percorre todas as árvores paralelamente, a quantidade de árvores não importa, apenas a altura da maior árvore. Para árvores com $\psi = 10.000$, a altura máxima é $h_{max} = \lceil log_2 10.000 \rceil = 14$, portanto o tempo máximo de execução é de $14/50.000.000$ segundos, ou seja, $0,28 \mu s$.

5.5 CONSIDERAÇÕES FINAIS

Neste Capítulo o experimento realizado para validação da metodologia proposta foi apresentado. O *setup* do experimento inclui uma placa com uma FPGA, uma Raspberry Pi, um laptop e um analisador lógico. Um modelo de *Isolation Forest* é treinado, validado e testado utilizando uma base de dados disponibilizada por outros pesquisadores, dado que uma base como esta não é disponibilizada pela indústria. Através de modificações na biblioteca de software utilizada, é possível obter a estrutura interna do modelo treinado, que será utilizada na implementação em hardware. A fim de testar a implementação em hardware, todo o conjunto de testes foi analisado através desta solução, tendo sido obtido o mesmo resultado que a implementação em Python para todas as amostras.

6 RESULTADOS

Neste Capítulo são expostos os resultados obtidos através dos experimentos detalhados no Capítulo 5. Na Seção 6.1, estão os resultados referentes ao desempenho do algoritmo de detecção, utilizando-se de métricas como a AUROC e taxas de verdadeiro positivo. Na seção 6.2, é apresentado o resultado obtido no experimento descrito na seção 5.4, que provará a capacidade do modelo proposto em efetivamente bloquear quadros CAN indesejados.

6.1 DESEMPENHO DO ALGORITMO DE DETECÇÃO

Para cada cenário de ataque descrito na Seção 5.2 serão exibidas métricas de desempenho de detecção: a curva ROC, a área sob a curva ROC (AUROC), precisão e *recall*; que são utilizadas para a verificação da capacidade do algoritmo em realizar a separação entre um tráfego autêntico e um tráfego de ataque. Essas métricas, exceto a curva ROC, foram obtidas a partir de uma validação cruzada *10-fold*, assim serão expostas as médias e os desvios-padrão das métricas selecionadas. As curvas ROC, expostas nas Figuras 7, 8 e 9, foram obtidas a partir de um dos modelos gerados na validação cruzada, porém as curvas para os 10 *folds* são similares em cada cenário de ataque. Dessa forma, essas imagens representam bem o desempenho do modelo para cada cenário.

O limiar de detecção de detecção utilizado para os cenários de ataque de *spoofing* foi o que obteve o melhor F_1 *score* na etapa de validação, assim dando igual importância para a precisão e o *recall*. Porém, analisando as curvas ROC obtidas para ambos os cenários de *spoofing*, nas Figuras 8 e 9, fica claro que existe apenas um limiar ótimo, sendo qualquer outro limiar pior ou igual nas taxas de falso positivo e verdadeiro positivo.

No cenário de ataque *fuzzy*, visando minimizar a quantidade de falsos positivos, que podem ser problemáticos no contexto automotivo, é proposta a utilização de dois limiares diferentes: (i) um que é capaz de detectar mais anomalias, porém com uma maior quantidade de falsos positivos; e (ii) outro que é capaz de detectar menos anomalias que o primeiro, porém minimizando os falsos positivos. O primeiro limiar foi obtido através do melhor F_2 *score*, enquanto o segundo foi obtido através do melhor $F_{0,05}$ *score*. Essa abordagem permite prevenir ativamente apenas os quadros CAN que o sistema considerar "mais anômalo" usando o segundo limiar, que minimiza os falsos positivos, e tomar outras ações menos imediatas se o quadro CAN apenas superar o primeiro limiar.

Nas Tabelas 8, 9 e 10 estão as métricas obtidas para cada cenário de ataque e os respectivos desvios-padrão. As AUROCs obtidas foram superiores a $0,98$, indicando que o método consegue realizar uma boa separação dos dados. Nos cenários de ataque de *spoofing*, apesar de obter um *recall* de 100%, existiram alguns falsos positivos que não

Tabela 8 – Métricas obtidas para o *dataset Fuzzy*.

	AUROC	Precisão	*Recall*	Acurácia
Média	0,998	Limiar 1: 0,872 Limiar 2: 0,996	Limiar 1: 0,997 Limiar 2: 0,704	Limiar 1: 0,979 Limiar 2: 0,959
Desvio-padrão	<0,001	Limiar 1: 0,026 Limiar 2: 0,002	Limiar 1: 0,001 Limiar 2: 0,079	Limiar 1: 0,004 Limiar 2: 0,010

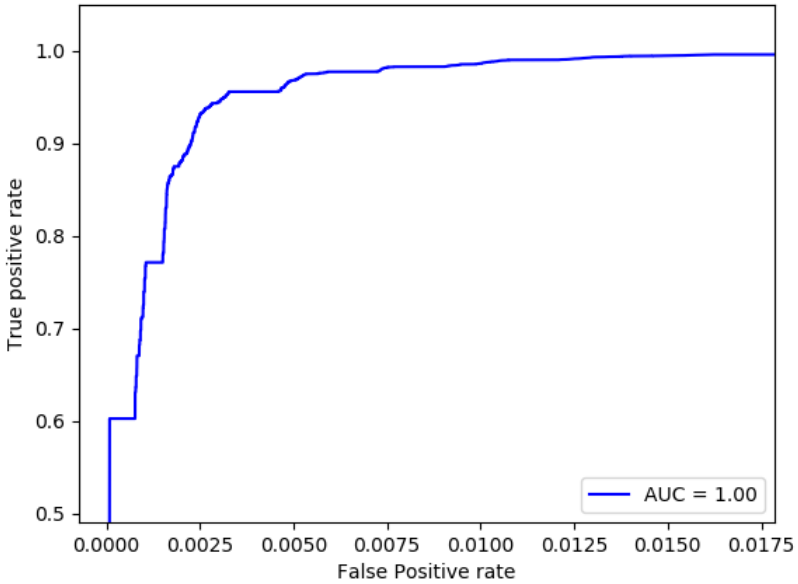

Figura 7 – Curva ROC para o *dataset Fuzzy*.

Tabela 9 – Métricas obtidas para o *dataset Spoofing (Gear)*

	AUROC	Precisão	*Recall*	Acurácia
Média	0,996	0,980	1	0,997
Desvio-padrão	0,001	0,009	0	0,001

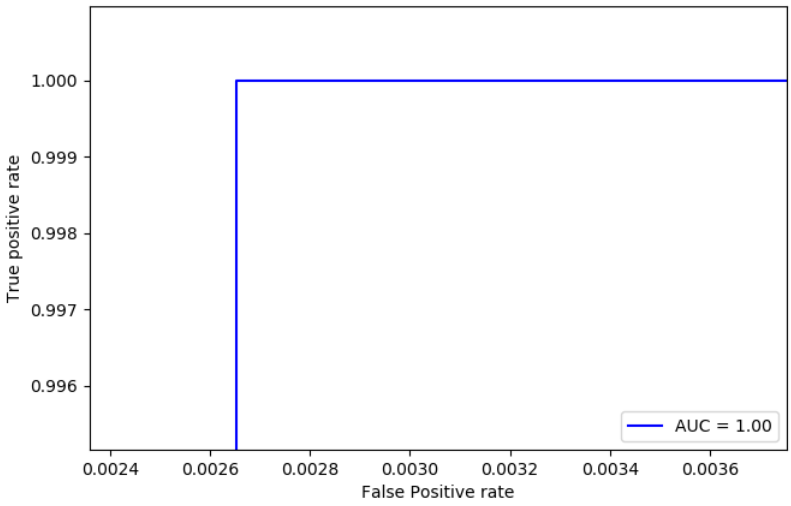

Figura 8 – Curva ROC para o *dataset Spoofing (Gear)*.

Tabela 10 – Métricas obtidas para o *dataset Spoofing (RPM)*.

	AUROC	Precisão	*Recall*	Acurácia
Média	0.984	0,918	1	0,986
Desvio-padrão	0,014	0,070	0	0,012

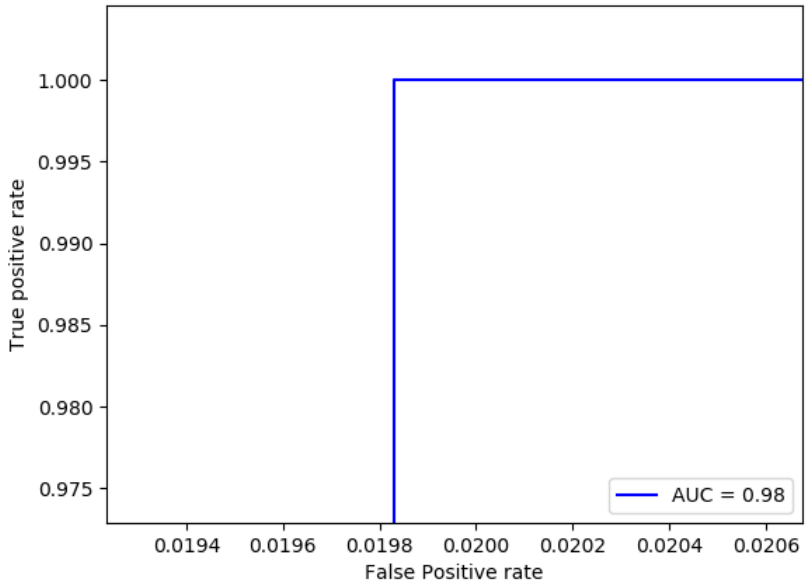

Figura 9 – Curva ROC para o *dataset Spoofing (RPM)*.

foram possíveis de classificar corretamente, mesmo mudando o limiar, assim implicando em uma menor taxa de precisão, principalmente no cenário de *spoofing* de dados de RPM. Para o cenário de ataque *fuzzy*, podemos perceber a diferença entre os limiares utilizados; o primeiro possui uma menor taxa de precisão, porém maior *recall*, enquanto o segundo apresenta justamente o contrário. Porém, no segundo limiar, o *recall* foi sacrificado a um valor de $0,704$ a fim de reduzir o número de falsos positivos, significando que menos ataques serão identificados.

Comparar os resultados obtidos neste trabalho com outras soluções já apresentadas não faz sentido caso o problema atacado seja diferente, ou seja, caso os cenários de ataque e a base de dados considerada não sejam os mesmos. Por isso, a fim de demonstrar que, apesar de ser capaz de responder em menos de $1\mu s$, a solução proposta apresenta também métricas de detecção comparáveis com o estado da arte, o resultado obtido neste trabalho será comparado com a solução apresentada em (SEO; SONG; KIM, 2018), o GIDS, a qual utiliza a mesma base de dados em seus experimentos.

A tabela 11 mostra as métricas obtidas pelo GIDS, considerando o desempenho do modelo treinado para ataques desconhecidos, e pelo RAIC, a solução apresentada neste trabalho. O GIDS desenvolve dois modelos, sendo um treinado para ataques conhecidos e outro treinado para ataques desconhecidos. A comparação com o modelo treinado para ataques desconhecidos do GIDS é a mais adequada, pois o RAIC não utiliza dados de

Tabela 11 – Comparação com o GIDS.

Tipo do ataque	Solução	*Recall*	Precisão	Acurácia
Fuzzy	GIDS	0.996	0.973	0.98
	RAIC (Limiar 1)	0.997	0.872	0.979
	RAIC (Limiar 2)	0.704	0.996	0.959
Spoof (RPM)	GIDS	0.99	0.983	0.98
	RAIC	1	0.918	0.986
Spoof (Gear)	GIDS	0.965	0.981	0.962
	RAIC	1	0.98	0.997

ataque em seu treinamento, sendo, portanto, projetado para a defesa de ataques desconhecidos. Para o cenário de ataque *fuzzy*, o GIDS, em geral, tem um desempenho superior ao RAIC, porém, com a utilização dos dois limiares, essa vantagem pode ser mitigada. Além disso, o RAIC possui a vantagem de poder corromper os quadros que ultrapassarem o limiar 2. Nos cenários de ataque de *spoofing*, os desempenhos foram similares, tendo o RAIC obtido melhores taxas no *spoofing gear*, enquanto o GIDS mostrou uma pequena vantagem no *spoofing RPM*.

6.2 TEMPO DE EXECUÇÃO EM FPGA

Conforme descrito no Capítulo 5, foi realizada uma medição do tempo que a FPGA leva para executar o algoritmo para uma amostra do conjunto de testes. Para isso, foi utilizado o analisador lógico Saleae Logic Pro 8, que é capaz de realizar 500 milhões de medições por segundo, sendo suficiente para verificarmos o tempo com uma precisão de $2ns$, que é dez vezes menor que o período do relógio da FPGA utilizada.

A Figura 10 mostra os sinais *Enable*, que indica quando o algoritmo pode começar a rodar, e *Done*, que indica quando o sistema concluiu a execução e já possui a resposta pronta. Portanto, a diferença entre os pontos A1 e A2 representa o tempo de execução da solução, que é de aproximadamente $0,28\mu s$, que é o valor esperado, conforme mencionado na Seção 5.4.

O tempo de $0,28\mu s$, obtido na avaliação de uma amostra do conjunto de testes, é o

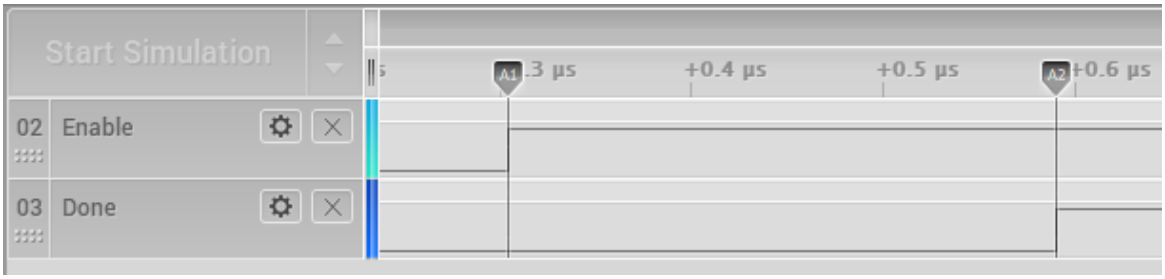

Figura 10 – Tempo de detecção.

maior tempo possível para a configuração de parâmetros utilizadas no experimento. É possível que amostras, principalmente anômalas, sejam avaliadas em menos tempo, pois geralmente não percorrem as árvores até uma folha de profundidade máxima.

Este resultado permite ao sistema atender o requisito temporal mencionado na Seção 4.5, assim possibilitando a construção de um IPS, conforme ilustrado na Figura 3.

6.3 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentadas as métricas de desempenho obtidas pela solução proposta, o RAIC, bem como uma comparação com outra solução proposta na literatura que utiliza a mesma base de dados, o GIDS. O RAIC obteve uma AUROC superior a $0,98$ nos três cenários de ataque, indicando que é capaz de realizar uma boa separação entre dados normais e anômalos. Utilizando dois limiares é possível mitigar a taxa de falso positivo no cenário de ataque *Fuzzy*, porém esta técnica não foi efetiva para os cenários de ataque de *Spoofing*. O RAIC apresenta desempenho similar ao GIDS nos cenários de ataque de *Spoofing*, tendo obtido melhores taxas de *Recall* e acurácia. No cenário de ataque *Fuzzy*, o desempenho do GIDS foi, em geral, superior. Apesar disso, o uso do RAIC tem como vantagem a possibilidade da corrupção de um quadro, sendo capaz de impedir a transmissão de quadros anômalos com uma baixa taxa de falsos positivos, através da utilização dos dois limiares. Por fim, o tempo de resposta da solução implementada em FPGA foi auferido através do analisador lógico. Como esperado, a solução apresentou um tempo de execução na ordem de décimos de microssegundos.

7 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Neste trabalho, é proposto um sistema de detecção de intrusão de tempo real e baseado em anomalias que é capaz de detectar anomalias em um quadro CAN antes da transmissão completa do mesmo, possibilitando que o quadro seja corrompido se considerado anômalo. Apesar de muitos outros IDS baseados em anomalias terem sido propostos para CAN, nenhum atende o requisito temporal para evoluir para um sistema de prevenção de intrusão. O RAIC mostrou um desempenho de detecção semelhante a outros IDSs propostos na literatura, além de possuir uma implantação viável e com bom custo-benefício, pois necessita de apenas uma FPGA.

Para uma rede CAN de 500kbps, responder em menos de 36μs permite que o IDS efetivamente impeça que quadros maliciosos sejam enviados pela rede se combinado com uma solução injetora de erros. Apesar da corrupção de um quadro suspeito já ter sido sugerida na literatura como um meio de prevenir um ataque, ainda são necessários estudos sobre o impacto dessa abordagem nos sistemas automotivos, especialmente considerando que um sistema de detecção de intrusão baseado em anomalias é suscetível a falso positivos.

É possível que uma abordagem beaseada em CPU também consiga atender ao requisito temporal mencionado. Porém, a fim de atender ao requisito temporal, é necessário ter controle total sobre o processador utilizado ou utilizar um sistema operacional de tempo real (*Real-Time Operating System*, RTOS), não sendo adequada a utilização de um sistema operacional de uso geral. Entre as vantagens da abordagem em CPU, estariam o custo e uma menor limitação de memória. Apesar disso, possivelmente não seria possível a execução em paralelo de todas as árvores, o que faria o tempo de execução crescer também com o número de árvores utilizadas, tornando um desafio a implementação de uma solução utilizando esta abordagem.

Isolation Forests podem alcançar taxas de precisão maiores. Os resultados apresentados foram limitar com base nos recursos da FPGA utilizada e na metodologia de exportação da *iTree*. Como mencionado na Seção 5.3, a FPGA empregada limitou a utilização de maiores valores nos parâmetros ψ e t, o tamanho da subamostragem e o número de árvores, respectivamente. Por exemplo, o uso de um atributo adicional que seja o valor diferencial entre *payloads* de mesmo ID poderia permitir que o sistema detectasse ataques que antes não seriam identificáveis. Porém, adicionar um atributo deste tipo traria uma complexidade maior ao sistema, com a necessidade de armazenar os últimos *payloads* de cada ID.

A abordagem apresentada poderia também ser estendida para o protocolo CAN FD. Porém, o número de atributos iria aumentar devido ao maior tamanho do *payload* em CAN FD. Com mais atributos, também poderia ser necessário utilizar mais árvores ou árvores maiores para alcançar um bom desempenho de detecção de anomalias. Em relação

a restrição de tempo, como CAN FD pode transmitir o campo de dados e o campo CRC utilizando maiores velocidades de transmissão, o tempo para a realização da detecção seria ainda menor, porém ainda alcançável, pois o IDS proposto é capaz de responder em menos de 1μs.

REFERÊNCIAS

ABBOTT-MCCUNE, S.; SHAY, L. A. Intrusion prevention system of automotive network can bus. In: *2016 IEEE International Carnahan Conference on Security Technology (ICCST)*. [S.l.: s.n.], 2016. p. 1–8. ISSN 2153-0742.

ARAUJO-FILHO, P. Freitas de. *Contributions to in-vehicle networks: error injection and intrusion detection system for CAN, and audio video bridging synchronization*. Dissertação (Mestrado) — Universidade Federal de Pernambuco, 2018.

BOSCH, R. Can specification version 2.0. *Published by Robert Bosch GmbH (September 1991)*, 1991.

BUSCHJÄGER, S.; MORIK, K. Decision tree and random forest implementations for fast filtering of sensor data. *IEEE Transactions on Circuits and Systems I: Regular Papers*, IEEE, v. 65, n. 1, p. 209–222, 2017.

CAN IN AUTOMATION (CIA). *CAN Knowledge*. 2018. <https://www.can-cia.org/can-knowledge/>. Acessado em: Dezembro-2018.

CHECKOWAY, S.; MCCOY, D.; KANTOR, B.; ANDERSON, D.; SHACHAM, H.; SAVAGE, S.; KOSCHER, K.; CZESKIS, A.; ROESNER, F.; KOHNO, T. et al. Comprehensive experimental analyses of automotive attack surfaces. In: SAN FRANCISCO. *USENIX Security Symposium*. [S.l.], 2011. v. 4, p. 447–462.

DUPONT, G.; HARTOG, J. d.; ETALLE, S.; LEKIDIS, A. Network intrusion detection systems for in-vehicle network-technical report. *arXiv preprint arXiv:1905.11587*, 2019.

GIANNOPOULOS, H.; WYGLINSKI, A. M.; CHAPMAN, J. Securing vehicular controller area networks: An approach to active bus-level countermeasures. *IEEE Vehicular Technology Magazine*, v. 12, n. 4, p. 60–68, Dec 2017.

GREENBERG, A. Hackers remotely kill a jeep on the highway—with me in it. *Wired*, Jul 2015. Acessado em: Novembro-2017. Disponível em: <https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/>.

HARTWICH, F. et al. Can with flexible data-rate. In: *Proc. iCC*. [S.l.: s.n.], 2012. p. 1–9.

KANG, M.; KANG, J. A novel intrusion detection method using deep neural network for in-vehicle network security. In: *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*. [S.l.: s.n.], 2016. p. 1–5. ISSN null.

KIM, H. K. *Car-Hacking Dataset for the intrusion detection*. 2018. <https://sites.google.com/a/hksecurity.net/ocslab/Datasets/CAN-intrusion-dataset>. Acessado em: Dezembro-2018.

KOYAMA, T.; SHIBAHARA, T.; HASEGAWA, K.; OKANO, Y.; TANAKA, M.; OSHIMA, Y. Anomaly detection for mixed transmission can messages using quantized intervals and absolute difference of payloads. In: ACM. *Proceedings of the ACM Workshop on Automotive Cybersecurity*. [S.l.], 2019. p. 19–24.

LIAO, H.-J.; LIN, C.-H. R.; LIN, Y.-C.; TUNG, K.-Y. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, Elsevier, v. 36, n. 1, p. 16–24, 2013.

LIU, F. T.; TING, K. M.; ZHOU, Z.-H. Isolation forest. In: IEEE. *2008 Eighth IEEE International Conference on Data Mining*. [S.l.], 2008. p. 413–422.

LIU, J.; ZHANG, S.; SUN, W.; SHI, Y. In-vehicle network attacks and countermeasures: Challenges and future directions. *IEEE Network*, IEEE, v. 31, n. 5, p. 50–58, 2017.

MARTINELLI, F.; MERCALDO, F.; NARDONE, V.; SANTONE, A. Car hacking identification through fuzzy logic algorithms. In: IEEE. *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. [S.l.], 2017. p. 1–7.

MILLER, C.; VALASEK, C. A survey of remote automotive attack surfaces. *black hat USA*, v. 2014, p. 94, 2014.

MILLER, C.; VALASEK, C. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, v. 2015, p. 91, 2015.

NATALE, M. D.; ZENG, H.; GIUSTO, P.; GHOSAL, A. *Understanding and using the controller area network communication protocol: theory and practice*. [S.l.]: Springer Science & Business Media, 2012.

OLUFOWOBI, H.; EZEOBI, U.; MUHATI, E.; ROBINSON, G.; YOUNG, C.; ZAMBRENO, J.; BLOOM, G. Anomaly detection approach using adaptive cumulative sum algorithm for controller area network. In: ACM. *Proceedings of the ACM Workshop on Automotive Cybersecurity*. [S.l.], 2019. p. 25–30.

SCIKIT-LEARN. *scikit-learn : Machine Learning in Python*. 2018. <https://scikit-learn.org>. Acessado em: Dezembro-2018.

SEO, E.; SONG, H. M.; KIM, H. K. Gids: Gan based intrusion detection system for in-vehicle network. In: *2018 16th Annual Conference on Privacy, Security and Trust (PST)*. [S.l.: s.n.], 2018. p. 1–6.

SHIREY, R. *RFC 4949-Internet Security Glossary*. [S.l.]: Version, 2007.

STAVROULAKIS, P.; STAMP, M. *Handbook of information and communication security*. [S.l.]: Springer Science & Business Media, 2010.

STRUHARIK, J. Implementing decision trees in hardware. In: IEEE. *2011 IEEE 9th International Symposium on Intelligent Systems and Informatics*. [S.l.], 2011. p. 41–46.

TAYLOR, A.; JAPKOWICZ, N.; LEBLANC, S. Frequency-based anomaly detection for the automotive can bus. In: IEEE. *2015 World Congress on Industrial Control Systems Security (WCICSS)*. [S.l.], 2015. p. 45–49.

TAYLOR, A.; LEBLANC, S.; JAPKOWICZ, N. Anomaly detection in automobile control network data with long short-term memory networks. In: IEEE. *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. [S.l.], 2016. p. 130–139.

TERASIC. *Altera DE2-115 Development and Education Board*. 2012. <https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=502>. Acessado em: Dezembro-2018.

UPTON, E.; HALFACREE, G. *Raspberry Pi user guide*. [S.l.]: John Wiley & Sons, 2014.

WEBER, M.; KLUG, S.; SAX, E.; ZIMMER, B. Embedded hybrid anomaly detection for automotive can communication. In: . [S.l.: s.n.], 2018.

APÊNDICE A – CÓDIGOS EM VERILOG

Código A.1 – Código em linguagem de programação Verilog de uma iTree

```
1
// Implementacao de uma iTree
3 // Entradas: Clock, Enable, Features, Dados do no atual
// Saidas: Comprimento do caminho percorrido adicionado do fator de correcao,
    endereco do no
5 module UN (clk, enable, features, height_plus_factor, addr_atual, dados_no_atual);

7 input wire clk;
input wire enable;
9
input wire [63:0] features;
11
reg [4:0] height;
13 output wire [23:0] height_plus_factor;
output reg [11:0] addr_atual;
15
input wire [39:0] dados_no_atual;
17
reg [7:0] selected_feature;
19 wire result;
//dados_no_atual[36] = leaf or not leaf
21 assign result = (selected_feature < dados_no_atual[35:28]);
assign height_plus_factor = dados_no_atual[23:0] + (height << 16);
23
always @ (dados_no_atual or features)
25 begin

27         //Verifica qual feature sera analisada
          case (dados_no_atual[39:37])
29
          3'b000 : selected_feature = features[63:56];
31        3'b001 : selected_feature = features[55:48];
          3'b010 : selected_feature = features[47:40];
33        3'b011 : selected_feature = features[39:32];
          3'b100 : selected_feature = features[31:24];
35        3'b101 : selected_feature = features[23:16];
          3'b110 : selected_feature = features[15:8];
37        3'b111 : selected_feature = features[7:0];
          endcase
39
end
41
always @(posedge clk)
43 begin
    // Se enable nao estiver ativado, retoma as variaveis ao estado inicial
45     if(~enable)
       begin
47        addr_atual = 0;
          height = 0;
49     end
```

```
        // Se enable estiver ativado, calcula o proximo endereco com base no resultado
            da comparacao
51      else
        begin
53          // Enquanto nao chegar num no folha, realiza estas operacoes
            if(~dados_no_atual[36])
55          begin
                if(result)
57                  addr_atual[11:0] = dados_no_atual[23:12];
                else
59                  addr_atual[11:0] = dados_no_atual[11:0];
                height = height + 1;
61          end
        end
63  end

65
    endmodule
```

Código A.2 – Código em linguagem de programação Verilog de uma iForest

```
1
module forest16_testegpio(
3
    ////////////// CLOCK //////////
5   input                       CLOCK_50,
    input                       CLOCK2_50,
7   input                       CLOCK3_50,

9   ////////////// LED //////////
    output           [8:0]      LEDG,
11  output          [17:0]      LEDR,

13  ////////////// KEY //////////
    input            [3:0]      KEY,
15
    ////////////// SW //////////
17  input           [17:0]      SW,

19  ////////////// RS232 //////////
    input                       UART_CTS,
21  output                      UART_RTS,
    input                       UART_RXD,
23  output                      UART_TXD,

25  ////////////// GPIO, GPIO connect to GPIO Default //////////
    inout           [35:0]      GPIO
27 );

29
//=======================================================
31 //  REG/WIRE declarations
//=======================================================
33
wire clk;
35 /*input wire*/ reg [63:0] features;
wire [2:0] position;
37 wire [7:0] data;
reg  [7:0] stored_data;
39 wire [31:0] height;
wire done;
41 wire isAnomaly;
wire /*reg*/ enable;
43 wire uart_data_rdy;
wire [7:0]uart_data_in;
45 wire data_sent;
reg uart_send_data_rdy;
47 reg [7:0] uart_send_data;
wire reset;
49 wire [23:0] arvore0_height;
wire [11:0] arvore0_addr;
51 wire [39:0] arvore0_data;
wire [23:0] arvore1_height;
53 wire [11:0] arvore1_addr;
wire [39:0] arvore1_data;
55 wire [23:0] arvore2_height;
```

```
    wire [11:0] arvore2_addr;
57  wire [39:0] arvore2_data;
    wire [23:0] arvore3_height;
59  wire [11:0] arvore3_addr;
    wire [39:0] arvore3_data;
61  wire [23:0] arvore4_height;
    wire [11:0] arvore4_addr;
63  wire [39:0] arvore4_data;
    wire [23:0] arvore5_height;
65  wire [11:0] arvore5_addr;
    wire [39:0] arvore5_data;
67  wire [23:0] arvore6_height;
    wire [11:0] arvore6_addr;
69  wire [39:0] arvore6_data;
    wire [23:0] arvore7_height;
71  wire [11:0] arvore7_addr;
    wire [39:0] arvore7_data;
73  wire [23:0] arvore8_height;
    wire [11:0] arvore8_addr;
75  wire [39:0] arvore8_data;
    wire [23:0] arvore9_height;
77  wire [11:0] arvore9_addr;
    wire [39:0] arvore9_data;
79  wire [23:0] arvore10_height;
    wire [11:0] arvore10_addr;
81  wire [39:0] arvore10_data;
    wire [23:0] arvore11_height;
83  wire [11:0] arvore11_addr;
    wire [39:0] arvore11_data;
85  wire [23:0] arvore12_height;
    wire [11:0] arvore12_addr;
87  wire [39:0] arvore12_data;
    wire [23:0] arvore13_height;
89  wire [11:0] arvore13_addr;
    wire [39:0] arvore13_data;
91  wire [23:0] arvore14_height;
    wire [11:0] arvore14_addr;
93  wire [39:0] arvore14_data;
    wire [23:0] arvore15_height;
95  wire [11:0] arvore15_addr;
    wire [39:0] arvore15_data;
97
    //=============================================================
99  //  Structural coding
    //=============================================================
101
    always @ (posedge clk)
103 begin
          if (~KEY[1])
105         begin
          case (position)
107
            3'b000 : features[63:56] = SW[7:0];
109         3'b001 : features[55:48] = SW[7:0];
            3'b010 : features[47:40] = SW[7:0];
111         3'b011 : features[39:32] = SW[7:0];
            3'b100 : features[31:24] = SW[7:0];
```

```
113         3'b101 : features[23:16] = SW[7:0];
            3'b110 : features[15:8] = SW[7:0];
115         3'b111 : features[7:0] = SW[7:0];
            endcase
117        end
    end
119

121 always @ (posedge clk)
    begin
123       case (position)

125         3'b000 : stored_data=features[63:56];
            3'b001 : stored_data=features[55:48];
127         3'b010 : stored_data=features[47:40];
            3'b011 : stored_data=features[39:32];
129         3'b100 : stored_data=features[31:24];
            3'b101 : stored_data=features[23:16];
131         3'b110 : stored_data=features[15:8];
            3'b111 : stored_data=features[7:0];
133         endcase
    end
135
    always@ (posedge clk)
137 begin
        case (SW[14:13])
139     2'b00 : stored_score = height[7:0];
        2'b01 : stored_score = height[15:8];
141     2'b10 : stored_score = height[23:16];
        2'b11 : stored_score = height[31:24];
143     endcase
    end
145
    reg [7:0] stored_score;
147 assign LEDR[15:8] = stored_score;
    assign LEDR[7:0] = stored_data;
149 assign position = SW[17:15];
    assign enable = ~KEY[0];
151 assign reset = ~KEY[2];
    assign clk = CLOCK_50;
153 assign LEDG[0] = done;
    assign LEDG[1] = isAnomaly;
155
    assign GPIO[0] = enable;
157 assign GPIO[1] = done;
    assign GPIO[2] = isAnomaly;
159
    //assign features = SW;
161 assign height = arvore0_height + arvore1_height + arvore2_height + arvore3_height +
        arvore4_height + arvore5_height + arvore6_height + arvore7_height +
        arvore8_height + arvore9_height + arvore10_height + arvore11_height +
        arvore12_height + arvore13_height + arvore14_height + arvore15_height;
    assign done = arvore0_data[36] & arvore1_data[36] & arvore2_data[36] & arvore3_data
        [36] & arvore4_data[36] & arvore5_data[36] & arvore6_data[36] & arvore7_data[36]
         & arvore8_data[36] & arvore9_data[36] & arvore10_data[36] & arvore11_data[36] &
         arvore12_data[36] & arvore13_data[36] & arvore14_data[36] & arvore15_data[36];
163 assign isAnomaly = (height < 32'd13670490);
```

```
    UN   arvore0 ( .clk(clk), .enable(enable), .features(features), .height_plus_factor(
        arvore0_height), .addr_atual(arvore0_addr), .dados_no_atual(arvore0_data)  );
165 m3_0  arvore0_m3( .addr(arvore0_addr),  .data(arvore0_data)  );
    UN   arvore1 ( .clk(clk), .enable(enable), .features(features), .height_plus_factor(
        arvore1_height), .addr_atual(arvore1_addr), .dados_no_atual(arvore1_data)  );
167 m3_1  arvore1_m3( .addr(arvore1_addr),  .data(arvore1_data)  );
    UN   arvore2 ( .clk(clk), .enable(enable), .features(features), .height_plus_factor(
        arvore2_height), .addr_atual(arvore2_addr), .dados_no_atual(arvore2_data)  );
169 m3_2  arvore2_m3( .addr(arvore2_addr),  .data(arvore2_data)  );
    UN   arvore3 ( .clk(clk), .enable(enable), .features(features), .height_plus_factor(
        arvore3_height), .addr_atual(arvore3_addr), .dados_no_atual(arvore3_data)  );
171 m3_3  arvore3_m3( .addr(arvore3_addr),  .data(arvore3_data)  );
    UN   arvore4 ( .clk(clk), .enable(enable), .features(features), .height_plus_factor(
        arvore4_height), .addr_atual(arvore4_addr), .dados_no_atual(arvore4_data)  );
173 m3_4  arvore4_m3( .addr(arvore4_addr),  .data(arvore4_data)  );
    UN   arvore5 ( .clk(clk), .enable(enable), .features(features), .height_plus_factor(
        arvore5_height), .addr_atual(arvore5_addr), .dados_no_atual(arvore5_data)  );
175 m3_5  arvore5_m3( .addr(arvore5_addr),  .data(arvore5_data)  );
    UN   arvore6 ( .clk(clk), .enable(enable), .features(features), .height_plus_factor(
        arvore6_height), .addr_atual(arvore6_addr), .dados_no_atual(arvore6_data)  );
177 m3_6  arvore6_m3( .addr(arvore6_addr),  .data(arvore6_data)  );
    UN   arvore7 ( .clk(clk), .enable(enable), .features(features), .height_plus_factor(
        arvore7_height), .addr_atual(arvore7_addr), .dados_no_atual(arvore7_data)  );
179 m3_7  arvore7_m3( .addr(arvore7_addr),  .data(arvore7_data)  );
    UN   arvore8 ( .clk(clk), .enable(enable), .features(features), .height_plus_factor(
        arvore8_height), .addr_atual(arvore8_addr), .dados_no_atual(arvore8_data)  );
181 m3_8  arvore8_m3( .addr(arvore8_addr),  .data(arvore8_data)  );
    UN   arvore9 ( .clk(clk), .enable(enable), .features(features), .height_plus_factor(
        arvore9_height), .addr_atual(arvore9_addr), .dados_no_atual(arvore9_data)  );
183 m3_9  arvore9_m3( .addr(arvore9_addr),  .data(arvore9_data)  );
    UN   arvore10 ( .clk(clk), .enable(enable), .features(features), .height_plus_factor
        (arvore10_height), .addr_atual(arvore10_addr), .dados_no_atual(arvore10_data)  )
        ;
185 m3_10  arvore10_m3( .addr(arvore10_addr),  .data(arvore10_data)  );
    UN   arvore11 ( .clk(clk), .enable(enable), .features(features), .height_plus_factor
        (arvore11_height), .addr_atual(arvore11_addr), .dados_no_atual(arvore11_data)  )
        ;
187 m3_11  arvore11_m3( .addr(arvore11_addr),  .data(arvore11_data)  );
    UN   arvore12 ( .clk(clk), .enable(enable), .features(features), .height_plus_factor
        (arvore12_height), .addr_atual(arvore12_addr), .dados_no_atual(arvore12_data)  )
        ;
189 m3_12  arvore12_m3( .addr(arvore12_addr),  .data(arvore12_data)  );
    UN   arvore13 ( .clk(clk), .enable(enable), .features(features), .height_plus_factor
        (arvore13_height), .addr_atual(arvore13_addr), .dados_no_atual(arvore13_data)  )
        ;
191 m3_13  arvore13_m3( .addr(arvore13_addr),  .data(arvore13_data)  );
    UN   arvore14 ( .clk(clk), .enable(enable), .features(features), .height_plus_factor
        (arvore14_height), .addr_atual(arvore14_addr), .dados_no_atual(arvore14_data)  )
        ;
193 m3_14  arvore14_m3( .addr(arvore14_addr),  .data(arvore14_data)  );
    UN   arvore15 ( .clk(clk), .enable(enable), .features(features), .height_plus_factor
        (arvore15_height), .addr_atual(arvore15_addr), .dados_no_atual(arvore15_data)  )
        ;
195 m3_15  arvore15_m3( .addr(arvore15_addr),  .data(arvore15_data)  );

197
    endmodule
```

Código A.3 – Exemplo de código Verilog gerado por script em Python para implementar uma memória que importa a estrutura de uma iTree de um arquivo .list

```
1

3 module m3_6( addr, data);
  input [11:0] addr;
5 output wire[39:0] data;
  reg [39:0] memoria[3072:0];
7 assign data [39:0] = memoria[addr];

9 initial begin
      $readmemb("mem_arvore6.list",memoria);
11 end
   endmodule
```

APÊNDICE B – CÓDIGOS EM PYTHON

Código B.1 – Código em linguagem de programação Python para a geração do código em Verilog de uma iForest

```
def criarFloresta(qtdArvores, threshold):
    filename = "forest"+str(qtdArvores)+".v"
    file = open(filename, 'w')
    file.write("module forest"+str(qtdArvores)+" (clk, enable, features, height,
        done, isAnomaly);\n")
    file.write("input wire clk;\n")
    file.write("input wire enable;\n")
    file.write("input wire [63:0] features;\n")
    file.write("output wire [31:0] height;\n")
    file.write("output wire done;\n")
    file.write("output wire isAnomaly;\n")
    file.write("\n")
    for i in range(0,qtdArvores):
        file.write ("wire [23:0] arvore"+str(i)+"_height;\n")
        file.write ("wire [11:0] arvore"+str(i)+"_addr;\n")
        file.write ("wire [39:0] arvore"+str(i)+"_data;\n")
    file.write("\n")
    file.write("assign height = arvore0_height")
    for i in range(1,qtdArvores):
        file.write (" + arvore"+str(i)+"_height")
    file.write(";\n")
    file.write("assign done = arvore0_data[36]")
    for i in range(1,qtdArvores):
        file.write (" & arvore"+str(i)+"_data[36]")
    file.write(";\n")
    file.write("assign isAnomaly = (height < 32\'d"+str(threshold)+");\n")
    for i in range(0,qtdArvores):
        file.write ("UN   arvore"+str(i)+" ( .clk(clk), .enable(enable), .features(
            features), .height_plus_factor(arvore"+str(i)+"_height), .addr_atual(
            arvore"+str(i)+"_addr), .dados_no_atual(arvore"+str(i)+"_data)  );\n")
        file.write ("m3_"+str(i)+"  arvore"+str(i)+"_m3( .addr(arvore"+str(i)+"_addr
            ),  .data(arvore"+str(i)+"_data)  );\n")
    file.write("endmodule")
```

Código B.2 – Código em linguagem de programação Python para a geração dos arquivos de código em Verilog referentes às memórias

```
def criarArquivosDeMemoria(qtd):
    for i in range(0,qtd):
        criarArquivoDeMemoria(i)

// A estrutura da itree esta contida no arquivo .list que e importado pelo .v gerado aqui
def criarArquivoDeMemoria(index):
    filename = "mem_arvore"+str(index)+".list"
    lines = file_len(filename)
    file = open("m3_"+str(index)+".v",'w')
    file.write("module m3_"+str(index)+"( addr, data);\n")
    file.write("input [11:0] addr;\noutput wire[39:0] data;\n")
    file.write("reg [39:0] memoria[" + str(lines-1) + ":0];\n")
    file.write("assign data [39:0] = memoria[addr];\n\n")
    file.write("initial begin\n\t$readmemb(\""+filename+"\",memoria);\n")
    file.write("end\n endmodule")
    file.close()

def file_len(fname):
    with open(fname) as f:
        for i, l in enumerate(f):
            pass
    return i + 1
```

Código B.3 – Código em linguagem de programação Python adicionados ao arquivo iforest.py da biblioteca sklearn, responsável por gerar os arquivos .list contendo as estruturas das iTrees

```
1
3
    \\Este metodo foi adicionado a classe IsolationForest
5   def print_memory_file(self):
        i = 0
7       for tree in self.estimators_:
            save_tree_structure(tree,"mem_arvore"+str(i)+".list")
9           i = i+1

11
def save_tree_structure(tree, filename, feature_names = None):
13      left      = tree.tree_.children_left
        right     = tree.tree_.children_right
15      threshold = tree.tree_.threshold
        features  = tree.tree_.feature
17      file = open(filename,'w')
        for i in range(0, tree.tree_.node_count):
19          print_mem_line( file, i,tree.tree_, threshold[i] , features[i] , 0  , left[i
                ] , right[i]  )
        file.close()
21

23  def print_mem_line(file,node_id, tree, threshold , feature, height, left, right):
        #Se o no nao for uma folha
25      if(left!= -1):
            file.write(np.binary_repr(feature, width=3))
27          file.write("0")
            file.write(np.binary_repr(int(np.ceil(threshold)), width=8))
29          file.write(np.binary_repr(height, width=4))
            file.write(np.binary_repr(left, width=12))
31          file.write(np.binary_repr(right, width=12))
            file.write("\n")
33      else: #Se for uma folha
            file.write(np.binary_repr(feature, width=3))
35          file.write("1")
            file.write(np.binary_repr(int(np.ceil(threshold)), width=8))
37          file.write(np.binary_repr(height, width=4))
            adding_factor = _average_path_length(tree.n_node_samples[node_id])
39          #print(adding_factor)
            adding_factor = int(np.floor( adding_factor * (2**16)))
41          #print(adding_factor)
            str = np.binary_repr(adding_factor, width=24)
43          #print(str)
            file.write(str)
45          #file.write(np.binary_repr(right, width=12))
            file.write("\n")
```