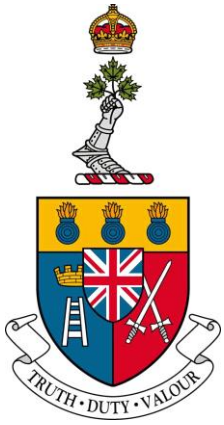
ANOMALY DETECTION FOR THE MIL-STD-1553B MULTIPLEX DATA BUS USING AN LSTM AUTOENCODER

UN DÉTECTEUR D'ANOMALIES PAR AUTO-ENCODEUR RÉCURRENT À MÉMOIRE COURT-TERME ET LONG TERME POUR MIL-STD-1553B



A Thesis Submitted to the Division of Graduate Studies
of the Royal Military College of Canada
by

Alec Harlow, BSc
Captain

In Partial Fulfillment of the Requirements for the Degree of
Master of Applied Science in Electrical and Computer Engineering

March, 2022

Acknowledgments

I would like to thank my supervisors, Dr. Vincent Roberge and Mr. Brian Lachine, for their continual guidance and mentorship throughout this research.

Abstract

Due to the modernization of aircraft systems and connectivity to ground based networks, including the Internet, in commercial and military aviation, real-time systems that were thought to be “air-gapped” are becoming more susceptible to cyber-attack. Most real-time systems that communicate using the Military Standard 1553B Multiplex Data Bus (MIL-STD-1553B) protocol do not have the ability to detect such attacks. Rightly so, these systems were originally developed with safety and redundancy in mind, not security. These two factors introduce attack vectors to MIL-STD-1553B communication buses and expose associated avionics systems to remote exploitation. Recent approaches to anomaly detection for the MIL-STD-1553B data bus have leveraged statistical analysis, Markov Chain modeling, remote terminal fingerprinting techniques and signature-based detection. However, their comparative effectiveness is unknown. In the case of the statistical analysis technique, the accuracy and precision in detecting the start and stop time of anomalous events are not ideal for conducting investigations. Deep learning techniques have shown to be an effective means of anomaly detection and applying these techniques to the MIL-STD-1553B Data Bus could provide more accurate and precise detection times when anomalies or attacks are present, when compared to known statistical analysis, leading to more efficient forensic investigations of anomalous events.

The aim of this research is to improve the time-related performance metrics when detecting attacks on the MIL-STD-1553B data bus traffic using a LSTM autoencoder. In order to accomplish this aim, an LSTM autoencoder detector was developed and tested on two separate datasets from different MIL-STD-1553B network architectures, totaling 27 threat instances over 9 scenarios. The detector was then compared to the MIL-STD-1553B Anomaly-Based Intrusion Detection System (MAIDENS) detector, a statistical-based intrusion detection system. The LSTM

autoencoder detected every threat instance with no false positive or false negative results and improved on the time-related performance metrics when compared to the MAIDENS detector. The results demonstrated this deep learning technique as an effective method for accurately identifying anomalies on a MIL-STD-1553B Data Bus.

Résumé

En raison de la modernisation des systèmes d'aéronefs et de la connectivité aux réseaux au sol, y compris Internet, dans l'aviation commerciale et militaire, les systèmes en temps réel que l'on croyait “isolés” deviennent de plus en plus vulnérables aux cyberattaques. La plupart des systèmes en temps réel qui communiquent à l'aide du protocole Military Standard 1553B Multiplex Data Bus (MIL-STD-1553B) n'ont pas la capacité de détecter de telles attaques. À juste titre, ces systèmes ont été initialement développés dans un souci de sécurité et de redondance, et non de sécurité. Ces deux facteurs introduisent des vecteurs d'attaque dans les bus de communication MIL-STD-1553B et exposent les systèmes avioniques associés à une exploitation à distance. Les approches récentes de détection d'anomalies pour le bus de données MIL-STD-1553B ont tiré parti de l'analyse statistique, de la modélisation de la chaîne de Markov, des techniques d'empreinte digitale des terminaux distants et de la détection basée sur les signatures. Cependant, leur efficacité comparative est inconnue. Dans le cas de la technique d'analyse statistique, l'exactitude et la précision de la détection de l'heure de début et d'arrêt des événements anormaux ne sont pas idéales pour mener des enquêtes. Les techniques d'apprentissage en profondeur se sont révélées être un moyen efficace de détection des anomalies et l'application de ces techniques au bus de données MIL-STD-1553B pourrait fournir des temps de détection plus précis et précis lorsque des anomalies ou des attaques sont présentes, par rapport à l'analyse statistique connue, conduisant à enquêtes médico-légales plus efficaces sur les événements anormaux.

L'objectif de cette recherche est d'améliorer les mesures de performances liées au temps lors de la détection d'attaques sur le trafic du bus de données MIL-STD-1553B à l'aide d'un auto-encodeur LSTM. Afin d'atteindre cet objectif, un détecteur d'auto-encodeur LSTM a été développé et testé sur deux ensembles de données distincts provenant de différentes architectures de réseau MIL-STD-1553B,

totalisant 27 instances de menace sur 9 scénarios. Le détecteur a ensuite été comparé au détecteur MAIDENS (Système de détection d'intrusion basé sur les anomalies) MIL-STD-1553B, un système de détection d'intrusion basé sur les statistiques. L'encodeur automatique LSTM a détecté chaque instance de menace sans résultats faux positifs ou faux négatifs et a amélioré les mesures de performances liées au temps par rapport au détecteur MAIDENS. Les résultats ont démontré que cette technique d'apprentissage en profondeur est une méthode efficace pour identifier avec précision les anomalies sur un bus de données MIL-STD-1553B.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The MIL-STD-1553B protocol is used in numerous military and civilian aircraft to communicate key information between flight instruments, sensors and other avionic systems. Due to the modernization and connectivity of platform and operational systems to the Internet in commercial and military aviation, real-time systems that were thought to be “air-gapped” are becoming more susceptible to cyber-attacks. Unfortunately, most real-time systems that communicate using the Military Standard 1553B Multiplex Data Bus (MIL-STD-1553B) protocol, do not have the ability to detect such attacks or anomalies. Additionally, the ability to convert the existing systems to a more secure protocol would be expensive and unrealistic. Therefore, the ideal approach would be to put a system in place that could monitor traffic on the bus and provide the ability to detect anomalies that occur.

Current MIL-STD-1553B anomaly detection techniques are able to detect a number of attack techniques that include, but are not limited to: Denial of Service (DoS) attacks, Remote Terminal (RT) spoofing attacks, Bus Controller (BC) spoofing attacks and attacks that manipulate RT messages. The MIL-STD-1553B Anomaly-Based Intrusion Detection System (MAIDENS), a statistics-based intrusion detection system purposed by Généreux et al. [1] is able to detect all of the aforementioned attacks. With other detectors such as: a Markov chain model proposed by Stan et al. [2], a fingerprinting method proposed Stan et al. [3] and a signature detection method purposed by Bernard [4], being able to detect a smaller subset of attack types. The issue faced with a detector like MAIDENS [1], even though it has an impressive zero false positive rate, is that the detection time accuracy and precision achieved when indicating the start and stop time of an attack

is not ideal for conducting forensic analysis on a detected attack. Having a detector that could reduce the detection window and thereby the number of messages associated with potential attack traffic, would significantly improve analysis efforts post attack detection.

To address the need to improve the time-related performance metrics when detecting attacks on the MIL-STD-1553B Data Bus, this thesis presents research into the detection of anomalies on the MIL-STD-1553B Data Bus using a Long Short-Term Memory (LSTM) autoencoder deep learning (DL) technique. It identifies the improvement in time-related performance metrics of an LSTM autoencoder detector in identifying attacks in MIL-STD-1553B bus traffic when compared to the MAIDENS detector purposed by Généreux et al. [1].

1.1 Motivation

The motivation for this research originates from the lack of security of the MIL-STD-1553B protocol and the vulnerabilities that stem from it. Since these networks are used for communication of core aircraft systems and are in some cases required for safe flight operations, the need to validate and verify system security and integrity should be a priority. Additionally, in the context of detecting attacks, there is a need to be more accurate and precise when detecting attacks from a timing-related performance standpoint, in order to reduce the amount of traffic that needs to be analyzed. Having a detector that can reduce the detection window and thereby the number of messages associated with potential attack traffic, would significantly improve analysis efforts post attack detection.

1.2 Statement of Deficiency

The statistics-based intrusion detection system, MAIDENS, can detect every attack event or threat occurrence that it has been tested against with perfect event

classification accuracy and precision. However, if you consider the time-related performance metrics MAIDENS can only detect an anomaly event within a range of 10 ±8 seconds [1]. This equates to about 10,000 ±8000 messages of traffic, based on the data transfer rate from the MAIDENS dataset, and would take an expert a significant amount of time to analyse each message to find the actual start and stop time of an attack. This deficiency stems from the inability to label each individual message through available MIL-STD-1553B recording methods. Leading to the need to rely on metrics for event classification instead of the more traditional confusion matrix-based approach, that would identify each message as a true/false positive or negative. Therefore, being able to increase the time-related performance metrics, specifically the detection time accuracy and detection time precision of the start and stop time of an attack, would drastically decrease the number of messages needed to be analysed during an investigation. In this context, accuracy is defined as the closeness of agreement between a test result and the accepted reference value and precision is defined as the agreement between independent test results obtained under stipulated conditions of detection [5].

1.3 Aim

The aim of this research is to improve the time-related performance metrics when detecting attacks on the MIL-STD-1553B data bus traffic using a LSTM autoencoder. The time related performance metrics refer to the detection time accuracy and the detection time precision when indicating the start and stop time of an attack event on the MIL-STD-1553B data bus. This approach will be validated through comparison with MAIDENS, a statistical-based anomaly detection method. The LSTM Autoencoder will aim to maintain the same detection capabilities when compared to the statistical-based anomaly detection method. Once an anomaly is detected it is important to be able to accurately and precisely identify the suspected

traffic so that it can be analysed and determination can be made if it is malicious or benign in nature.

In order to accomplish the aim, two datasets of MIL-STD-1553B Data Bus traffic will be used, dataset 1 will be from the work conducted in MAIDENS [1] and includes five threat scenarios. The second dataset will be captured on proprietary test equipment provided by the Royal Military College of Canada (RMC) and consists of four threat scenarios. An initial baseline of normal MIL-STD-1553B Data Bus traffic will be collected on an emulated MIL-STD-1553B environment that consists of MIL-STD-1553B hardware and a combination of emulated and simulated RTs. Once a baseline model is established, scenarios will be created using a tool built by Paquet in [6] to generate traffic recordings with anomalous events. The time related performance metrics of both MAIDENS and the LSTM autoencoder methods to detect these anomalies will be evaluated in terms of improved detection time accuracy and detection time precision. In the context of this research, improving these two time-related detection metrics is referred to as improving the effectiveness of anomaly detection. This research will also identify a supplementary list of features that describe MIL-STD-1553B message traffic that can be used as a baseline set of features in follow on or future works for MIL-STD-1553B detectors.

1.4 Research Activities

The following three phases were conducted in order to achieve the aim of this research:

1. Data Acquisition Pipeline:
    a. Generation and collection of MIL-STD-1553B baseline and anomalous traffic recordings.
    b. Extraction of features derived from MIL-STD-1553B message traffic.

2. LSTM Autoencoder Model Development:
    a. Creation of a feature extraction tool to ingest MIL-STD-1553B recordings and prepare a usable dataset for model creation.
    b. The creation and application of a LSTM Autoencoder detector to create a baseline model to evaluate subsequent traffic against for anomaly detection.
3. Anomaly Detection and Validation:
    a. Validation of the detector based upon its ability to accurately and precisely detect anomalies in comparison to MAIDENS using the same datasets.

1.5 Results

A detector was created using an LSTM autoencoder that was successful in creating a baseline model from the collected MIL-STD-1553B bus traffic, using the proposed feature set in Appendix A. The LSTM autoencoder was able to detect anomalies on two separate datasets from different MIL-STD-1553B network architectures, totaling 27 threat instances over 9 scenarios. The LSTM autoencoder detected every threat instance for dataset 1 and was compared to the MAIDENS detector. When compared to the MADIENS detector, the LSTM autoencoder detector significantly improved the detection time accuracy and detection time precision when indicating the start and stop time of an attack event. The degree of detection accuracy would be more beneficial, compared to MAIDENS, in a forensics investigation as the detected times are closer to the actual attack time, resulting in far fewer frames to needlessly analyse.

1.6 Organization

The remainder of this document will outline the problem space and describe the research in greater detail, by explaining the methodology and design, as well as by

presenting the results and analysis of the DL anomaly detector. Chapter 2 will provide further background of the MIL-STD-1553B protocol, the currently known attack techniques, as well as current detection techniques and methods. Chapter 3 describes the methodology followed that led to the design of the LSTM autoencoder model and overall deep learning pipeline. Chapter 4 presents the results of the developed anomaly detector, their analysis and validation of the stated aim. Chapter 5 will then provide a conclusion to the research.

Chapter 2

Background

In this chapter the MIL-STD-1553B protocol is described in detail along with its inherent vulnerabilities. Methods for intrusion detection are discussed to include an overview of machine learning (ML), deep learning (DL), Long Short-Term Memory (LSTM) artificial neural networks, autoencoders, and selected examples of how such techniques have been implemented for anomaly detection. Current methods for anomaly detection on MIL-STD-1553B are then presented, along with an explanation of what type of attacks they are able to detect. Finally, validation metrics for anomaly detection and time-related performance are provided.

2.1 Military Standard 1553B

MIL-STD-1553B is a military standard bus communications protocol published in 1973 by the United States Department of Defense. The standard uses a multipoint topology of remote terminals (RTs) connected by a dual redundancy data bus as depicted in Figure 1. One terminal is designated as the bus controller (BC) and initiates and directs all communication on the bus. A bus monitor (BM) can be attached to the bus, but traditionally fills the role of a data historian, analogous to a "black box" on an aircraft [7].

There are a total of 32 addresses (0 – 31) on a MIL-STD-1553B bus network, one address (31) is reserved as a broadcast address and the remaining 31 addresses are for potential RTs that can be connected to the data bus. Each address has up to 32 sub-addresses or addressable data buffers that are used to read and write data to and from, with sub-addresses 0 and 31 being reserved for mode codes (section 2.1.3). Information is transferred through three 20-bit messages: a command word, a status

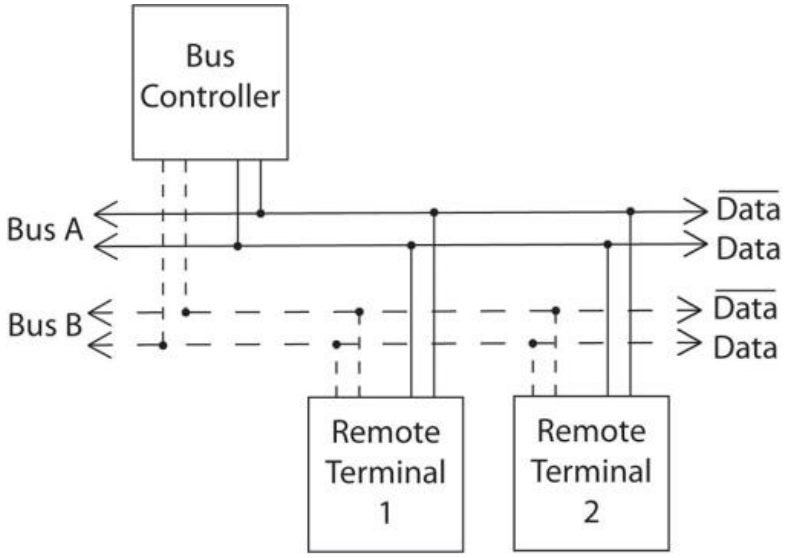


Figure 1 – Example of a MIL-STD-1553B Bus Topology Consisting of a BC and Two RTs Connected by a Dual Redundancy Data Bus [1]

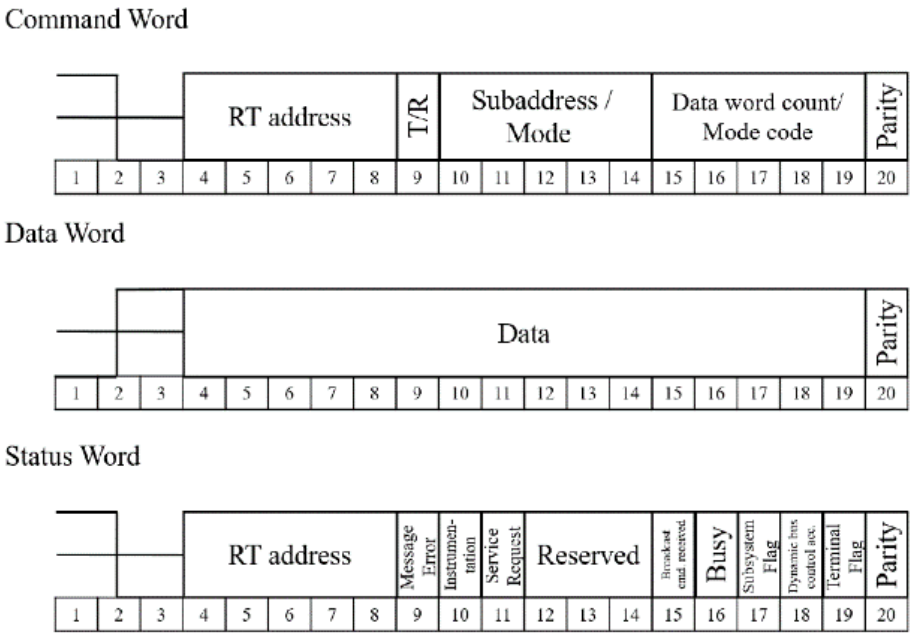


Figure 2 – Per Bit Breakdown of Command, Data and Status Words for MIL-STD-1553B [2]

word and a data word. The per-bit-breakdown of each word is showed in Figure 2. The protocol uses these three types of words for all communication on the bus and in order to initiate communication a command word must be sent by the BC [7]. This ensures that all traffic on the bus is scheduled and reliable.

2.1.1 Network Communication

As described in the previous section, communication on a MIL-STD-1553B bus is directed and controlled by the BC. A MIL-STD-1553B bus will contain at least one BC and may contain one or many redundant BCs that can take control in case of a failure. No RT can transmit on the bus without first receiving instructions by means of a command word from a BC. The BC controls bus traffic with a predefined cyclic messaging schedule, referend to as the master schedule, where messages can be either periodic or aperiodic. Periodic messages are always transmitted within a fixed time period. Aperiodic messages are conditional and are not sent at a fixed interval, however they will be transmitted within the same allotted time slot only if the condition to transmit that specific aperiodic message has been met. A collection of periodic and aperiodic messages forms a minor frame within the master schedule and a collection of minor frames form a major frame or the entirety of the master schedule as shown in Figure 3. The number of minor and major frames are unique to each network design and is based on the number of RTs and the types of messages required to be sent. Messages are combinations of the three available word types (section 2.1), that can be in one of two message formats: data messages and control messages [7]. These two message formats will be explained further is the following subsections.

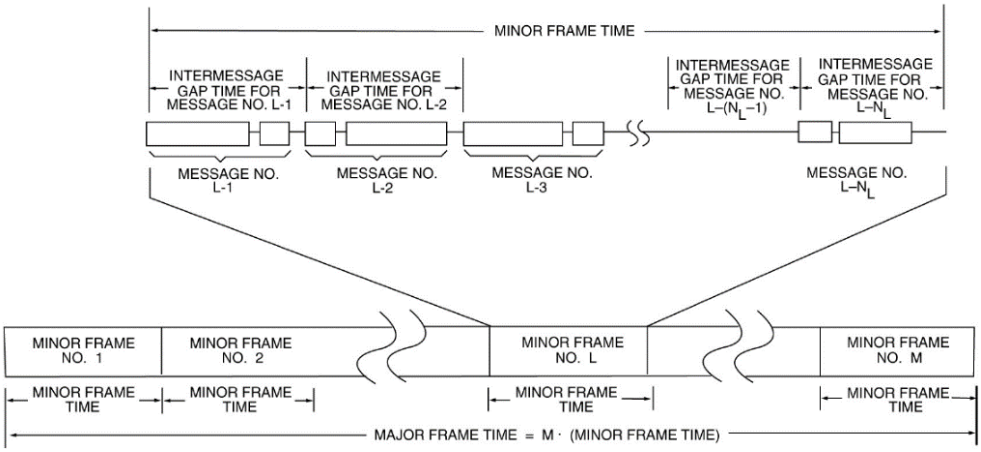


Figure 3 – MIL-STD-1553B Breakdown Major and Minor Frames [7]

2.1.2 Data Messages

To transfer data across the bus the BC will always initialize the communications by issuing a command word to the intended destination RT(s). The BC will direct RTs to read and/or write from specified sub-address or data buffers. Single data transfers can be BC to RT, RT to BC and RT to RT. Broadcast data transfers can be from the BC to RT(s) and from RT to RT(s). Broadcast messages differ as they do not require a status message to confirm receipt of the message. Figure 4 depicts these message formats in further detail [7].

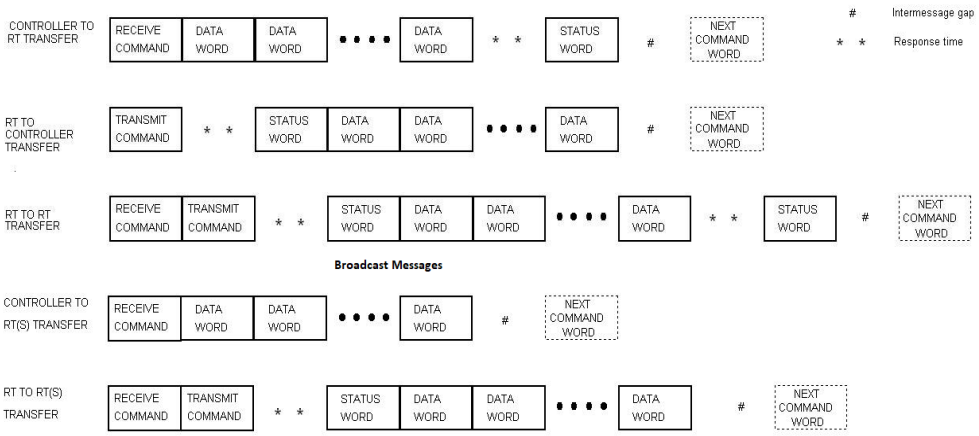


Figure 4 – MIL-STD-1553B Data Message Formats [7]

2.1.3 Control Messages

The BC monitors and controls the bus by issuing mode commands to RTs. These control messages contain a command word and associated data words or consist of a command word containing a single mode code. Mode codes are predetermined functions, described by the MIL-STD-1553B protocol [7], as shown in Table 1. A mode code may direct an RT to reply with only a status word or a status word and the corresponding data words. The BC can also send broadcast control messages to RT(s) without requiring a response. The types of control messages are depicted in Figure 5.

Table 1 – Assigned Mode Codes [7]

Transmit-Receive Bit	Mode Code	Function	Associated Data Word	Broadcast Command
1	00000	Dynamic bus control	No	No
1	00001	Synchronize	No	Yes
1	00010	Transmit status word	No	No
1	00011	Initiate self-test	No	Yes
1	00100	Transmitter shutdown	No	Yes
1	00101	Override transmitter shutdown	No	Yes
1	00110	Inhibit terminal flag bit	No	Yes
1	00111	Override inhibit terminal flag bit	No	Yes
1	01000	Reset remote terminal	No	Yes
1	01001	Reserved	No	-
1	01111	Reserved	No	-
1	10000	Transmit vector word	Yes	No
0	10001	Synchronize	Yes	Yes
1	10010	Transmit last command	Yes	No
1	10011	Transmit bit word	Yes	No
0	10100	Selected transmitter shutdown	Yes	Yes
0	10101	Override selected transmitter shutdown	Yes	Yes
1 or 0	10110	Reserved	Yes	-
1 or 0	11111	Reserved	Yes	-

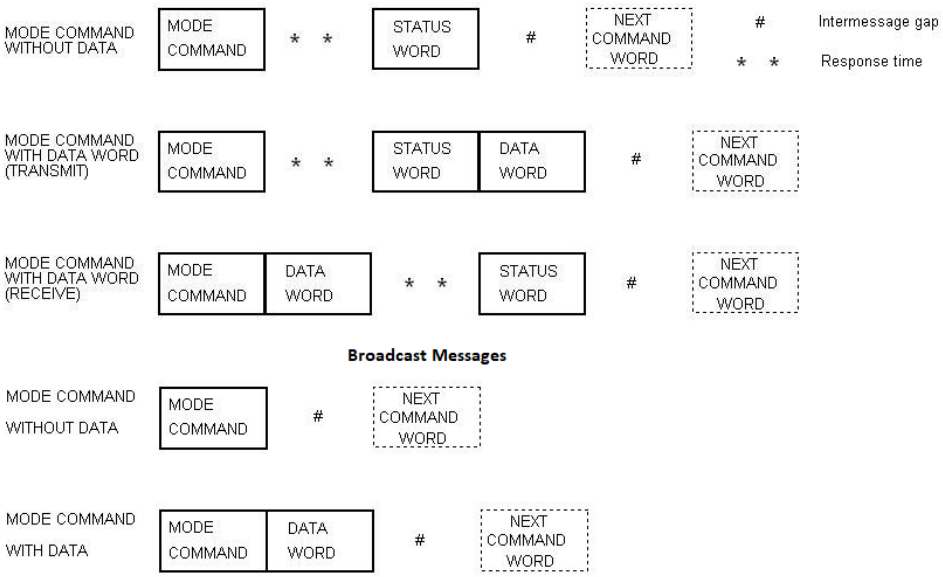


Figure 5 – MIL-STD-1553B Control Message Formats [7]

2.2 MIL-STD-1553B Vulnerabilities

The MIL-STD-1553B Protocol was designed with safety and reliability, not security in mind. Every communication exchange is preprogrammed and follows a cyclic schedule controlled by the BC. Manufacturers of RTs are expected to follow the guidelines defined by the MIL-STD-1553B protocol [7]. However, attackers are not limited by these guidelines set by the design documentation and can use its flaws to achieve their desired effect.

Stan et al. [2] showcase two main attack categories: message manipulation and behaviour manipulation. Message manipulation referring to modification of legitimate words (command, data, or status) that are transmitted over the bus. Behaviour modification refers to altering how a component would normally operate. Using these two types of attacks methods Stan et al. in [2] identifies 3 main types of threats to the MIL-STD-1553B communications protocol: denial of service (DoS),

data leakage, and data integrity violation. These attacks can be carried out by two means. The first is by a rogue RT: a device that is not intended to be connected to the data bus and was not part of the original design. The second is by a compromised RT: a RT that is part of the original design, but has been maliciously modified by some means. In addition to the types of attacks described in Stan et al. [2], a study by Lounis et al. in [8] reviews and analyses the attack vectors on the MIL-STD-1553B data bus as well. They identify 4 types of attacks that can occur: fabrication attacks, interception attacks, interruption attacks, and modification attacks. These attacks have some overlap with the types of attacks in [2] and contain attack vectors relating to RT components not just the MIL-STD-1553B protocol. For the purpose of this research, the types of attacks described by Stan et al. in [2] will be used and are explained in more detail in the following sections.

2.2.1 Denial of Service

A Denial of Service (DoS) disrupts or blocks communication from occurring [2]. On the MIL-STD-1553B Bus the simplest example of a DoS would be in the form of behaviour manipulation. The attack would consist of a compromised RT or rogue RT that is attached to the data bus, in a BC mode, allowing it to flooding the bus and potentially the redundant bus with messages. This would prevent all other communications from occurring on the bus, which depending the systems relying on the MIL-STD-1553B data bus communications, could have disastrous consequences. Message manipulation for this type of attack would refer to a compromised RT, in a BC mode, changing the fields of a command word to control data routing and causing collisions on the bus. An example would be flipping the transmit/receive bit from receive to transmit. The more complex form of a DoS is a targeted RT DoS explained in the next section.

2.2.1.1 Targeted Remote Terminal Denial of Service

Targeted Remote Terminal Denial of Service (RT DoS) is achieved when a compromised RT or a rogue RT attached to the data bus targets specific RTs or specific RT sub-addresses. This type of behaviour manipulation attack can be achieved by transmitting when the targeted RT is receiving a command or when it is transmitting a response to denying the target RT or RT sub-address. Message manipulation can be achieved the same as in the previous section, but specifically targeting only specified RTs or RT sub-addresses.

2.2.2 Data Leakage

Data leakage is defined by [2] as the unauthorized transmission of data between system components. For example, if an attacker wanted to leak data on the bus using a message manipulation technique, a compromised RT could make changes to the word count (WC) field or the terminal address (TA) in a command word. This could cause an RT to transmit exceeding data words, or to transmit data words to another RT. A behavioural manipulation technique would include an RT that could have been compromised to communicate over a covert channel. A covert channel as described in [9] is defined as “a process that alters a particular data item, and the receiving process detects and interprets the value of the altered data to receive information”. A covert channel could be set up between a BC and compromised RT using an acyclic transfer storage channel attack described in [10] and shown in Figure 6. This attack is accomplished by altering the reserve bit field in the status word to be received by an intended RT. A covert channel attack would allow an attacker to gather information about the system and could potentially leak it outside the aircraft using legitimate communication systems like a radio or satellite communications connected RT.

2.2.3 Data Integrity Violation

RT Data Manipulation described in [2] involves the rogue device or compromised RT that injects or modifies any portion of the data on the bus, also known as spoofing. This could occur by transmitting on the data bus masquerading as RT or the BC. A rouge or compromised device could also transmit data words at the same time as targeted RT, corrupting the data for the intended RT. Another way to perform a data integrity violation attack is known as RT hijacking.

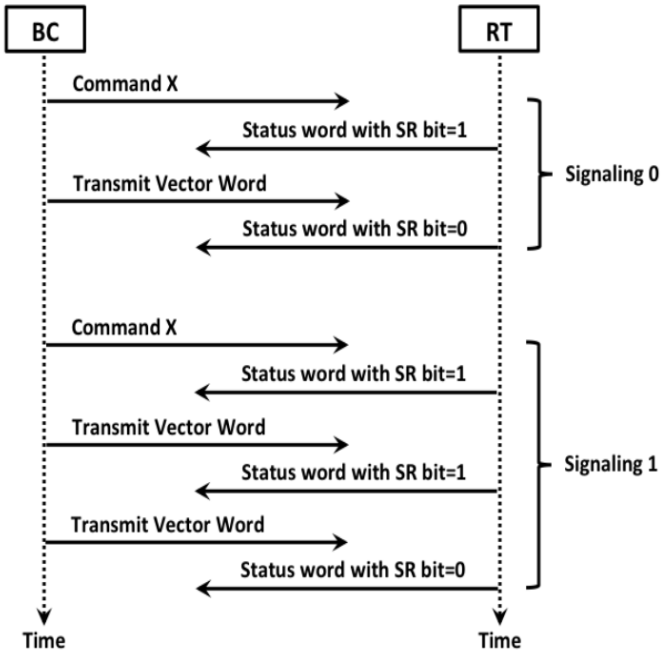


Figure 6 – Acyclic Transfer Storage Channel Attack [10]

2.2.3.1 RT Hijacking

RT hijacking is a form of data integrity violation and uses a compromised or rogue RT to take over or control a legitimate RT. A means of RT hijacking is presented in the works of Paquet [6], the exact implementation could not be disclosed due to proprietary restrictions. However, an alternate means of RT hijacking is presented by Lounis et al. in [8] described as a Man in the Middle attack. In this example, the

attacker is impersonating the BC and causes an illegal data transfer between two legitimate RTs. This attack is illustrated in the Message Sequence Chart in Figure 7 (where δt is the RT's admissible delay), the attacker observes a data transfer ($Data_i$) from RT_i to RT_j and then generates another data transfer ($Data_j$) from RT_j to RT_i once the former is completed [8]. An attack of this nature is only achievable if there is enough of a timing gap in the buses master schedule to allow for a message inject, otherwise message collisions will occur. Alternatively, the main BC and master schedule would need to be altered to achieve this type of attack. The benefit of conducting the attack as described is unclear and assumed to be a theoretical scenario.

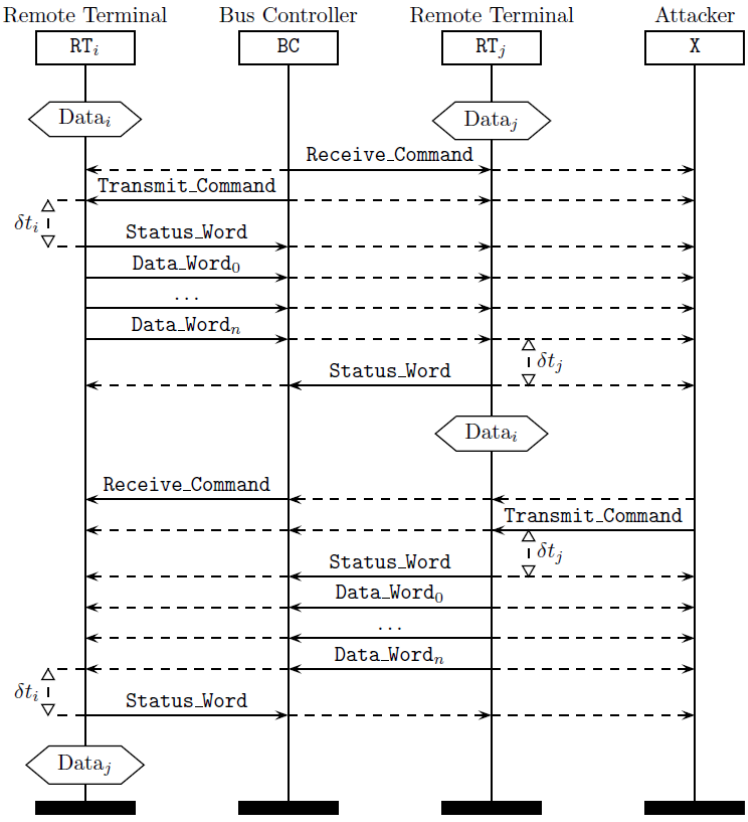


Figure 7 – Man in the Middle Attack [8]

2.3 Intrusion Detection Systems

An Intrusion Detection System (IDS) is defined as a system that can discover, determine, and identify unauthorized use, duplication, alteration, and destruction of information systems [11]. An IDS is generally characterized one of two ways: either it is misuse-based (also known as signature-based) or anomaly-based. Misuse-based detection uses known signatures in order to detect attacks or intrusions [12]. They are effective for detecting known type of attacks without generating an overwhelming number of false alarms. The limitation of misuse-based detection is that by itself it cannot detect novel attacks. It is also limited to the user's ability to update the database of known signatures. Anomaly-based techniques model the normal network and system behaviour, and identify anomalies as deviations from normal behaviour [12]. Anomaly-based detection methods are appealing for a few reasons: they have the ability to detect zero-day attacks, the profiles of normal activity are customizable based on system in which they are implemented on, and the output from anomaly-based detection engines can be used to define alert signatures for misuse-based detectors [12]. Anomaly-based techniques can be broken down into two main categories for detection: statistical-based detection and machine learning-based detection, which includes the subcategory deep learning.

2.3.1 Statistical Anomaly Detection

Statistical-based anomaly detection uses statistical theories as a basis to detect anomalous activity. Statistical methods can use the occurrence of normal events in a system in order to create a baseline profile. This baseline profile is then used to compare subsequent events the occur on the system and events that are far enough away from the baseline are considered to be anomalous [13]. For example, a frequency-based approach would create a baseline of the average number of packets sent over a given time period for each component of the system. A threshold for each component would then be determined based on the allowable number of packets

above and below the average. Once the system is in operation, if a component sends too many or not enough packets over specified period of time, based on its given threshold, it would then be classified as anomalous. If the activity on the system fluctuates in unpredictable ways it will be very difficult to use a statistical method to model the baseline of the system and set meaningful threshold values that would be able to identify anomalous behaviour. This is where a model that would be able to adapt or learn the system would be more applicable.

2.3.2 Anomaly Detection using Machine Learning

ML is generally described as a set of mathematical techniques, implemented on computer systems, that enables a process of information mining, pattern discovery and drawing inferences from data. A potential difference between statistical-based and ML is that a statistical-based methods require a defined feature set to model a system, whereas ML methods are able to perform pattern discovery, when a feature set is unavailable. ML can then be broken down into two main classes: supervised and unsupervised. Supervised learning can adopt a Bayesian approach to knowledge discovery, using probabilities of previous observed events to infer probabilities of new events. Unsupervised learning usually draws abstractions from unlabeled datasets and applies these to new data [13]. The following sections will describe some ML algorithms, both supervised and unsupervised.

2.3.2.1 Bayesian Network

A Bayesian network models the probabilistic relationships between variables with the ability to represent causal relationships depicted in [14]. It is a type of supervised ML method that requires, labeled data to train and create a proper learnt model to apply data to, in order to determine if it is anomalous or benign. Each state in the model is based on the combined probabilities of related variables. As in Figure 8 the probability of an engine powered system such as a car being in the “Moving” state is based on the combined probabilities of the “Engine On” and “Gas Usage”

variables. Based on Figure 8, if both variables that transition to “Moving” are true, then there is a 99 percent probability that the “Moving” is also true. If the state happens to be false, probabilistically it could be determined to be anomalous. The draw back with such a model is that the probabilistic relationships between states requires expert knowledge to be built effectively or efficient algorithms that can perform inference [14]. For smaller systems this can be done with ease, but as the system grows with complexity so does the time required to model the system.

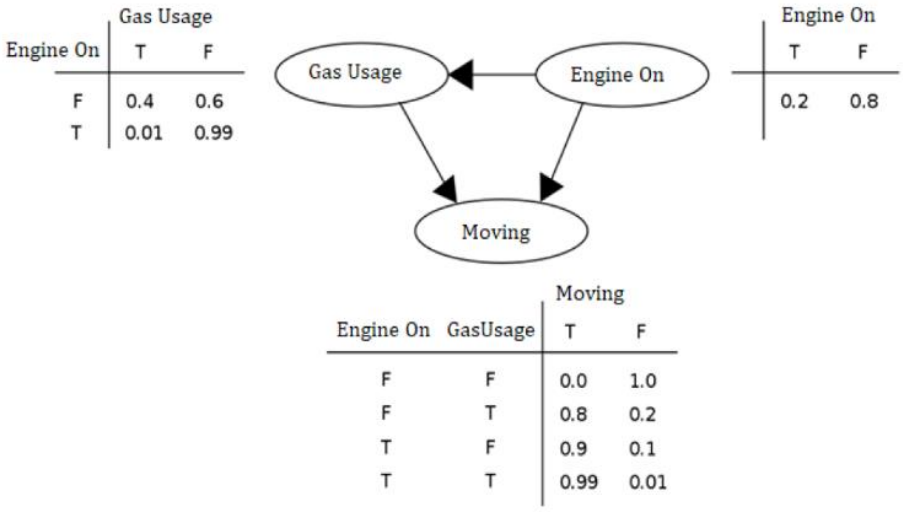


Figure 8 – Simple Bayesian Probability Network

2.3.2.2 Markov Chain

A Markov chain model, is primarily used to model the probabilities of transitioning between states in a system. This makes it ideal for representing valid transitions between messages from a training set of legitimate messages [2]. Where a Markov Chain model differs from a Bayesian network is that the probability of the next transition is based solely on the current state of the system and not the combined probabilities of previous variables within the system. This means that if a system has three possible states: S1, S2, and S3, where S1 is the starting state and it can transition between all states including itself, the combined probability of

transitioning between states must equal 1. An example of how this concept could be applied is if the probabilities of moving between states were as follows: S1 to S1 is 10%, S1 to S2 is 90%, and S1 to S3 is 0%. If over a certain timeframe S1 transitions back to S1 more than 10% of the time a flag could be raised to indicate anomalous behaviour. If at any point the system makes a transition to S3 a flag could be raised instantaneously. This sort of modeling requires that all states and the transitions between them be well defined in a system, making it less ideal for non-deterministic systems, but suitable for deterministic systems.

2.3.2.3 Support Vector Machine

A Support Vector Machine (SVM) is a supervised learning method used for binary or multi-class classification, regression and outlier detection, that aims to separate data into separate data classes [15]. It achieves this by placing a hyperplane with the largest separation or margin between the type classes. One such example would be using an SVM to classify valid and invalid data. The data must be well defined and there must be a large enough margin of difference between the two types of data so that invalid data is not classified as valid or vice versa. If the two types of data are too similar there is a greater chance that the SVM would classify them incorrectly.

An extension of the SVM algorithm that can be used with unlabeled data is a one class support vector machine (OCSVM). The OCSVM algorithm takes input data and maps it into a high dimensional feature space, it then iteratively finds the maximal margin hyperplane, which best separates the training data from the origin as seen in Figure 9. The origin represents all data points that have low similarity to the training set and is where the outliers of the dataset are supposed to lie. This is ideal for intrusion detection as the training data can be the baseline for how the system is supposed to behave. Anything that does not fit within the one class can be considered anomalous, meaning that system does not need to know what invalid data looks like, just valid data.

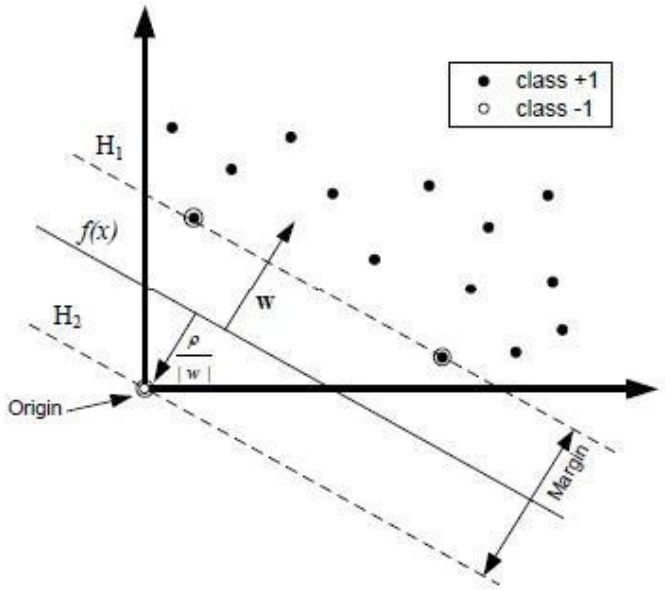


Figure 9 – OCSVM Mapping Input Data Into a High Dimensional Feature Space [15]

2.3.2.4 Clustering

Clustering refers to unsupervised learning algorithms which do not require pre-labeled data to extract rules for grouping similar data instances [16]. There is a breadth of clustering algorithms that can be applied to datasets depending on the desired outcome. They are ideally used in situations where there is little prior information available about the data and can be used to make a (preliminary) assessment of the data's structure based on the interrelationships of the data points [16]. The benefit of clustering algorithms is that nothing needs to be known about the data, however anomalous data that is designed to look like normal data is difficult to detect with these methods. More information on the various types of clustering algorithms and their applications can be found in [16].

2.3.2.5 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a way of representing how a brain would process information. An ANN is composed of artificial neurons that are able to perform certain computations on their inputs [17]. The simplest form of an ANN, depicted in Figure 10, is a feedforward neural network, where input data passes through hidden layers to an output layer. In an ANN these hidden layers take weighted inputs and the corresponding number of neurons in the next hidden layer is a reduction of the previous later starting from input to output. The downfall with ANN based learning is that as the number of distinct features in the data increase, the more time that is required to generate an effective model of the data.

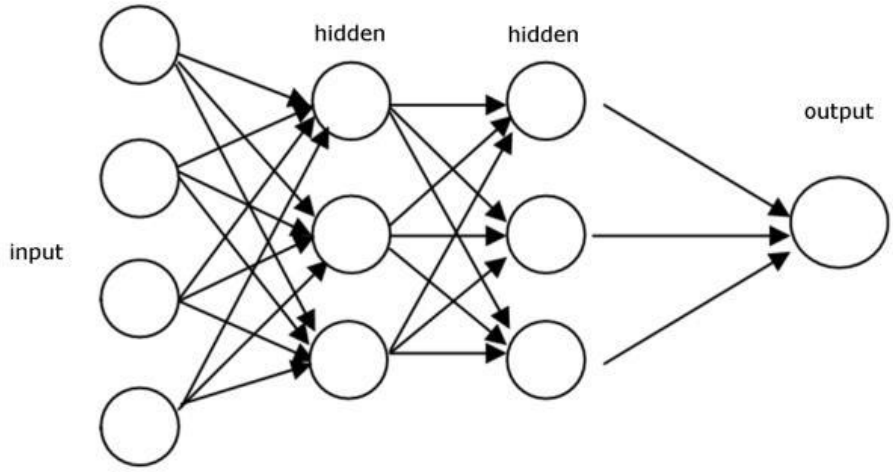


Figure 10 – Example of a Feedforward ANN

2.3.3 Anomaly Detection using Deep Learning

While ML techniques can be adapted to most problem sets, as the problem space increases so does the complexity and the harder it becomes to apply a ML method. This section will explore the potential of DL methods that can be applied for anomaly detection. DL is a subset of ML that refers to a specific class of multilayered models

that use many layers of simpler statistical components to learn a representation of the data [13]. DL networks can utilize supervised and unsupervised learning methods or a combination of both. These statistical models are usually layers of Artificial Neural Networks (ANNs) and can be adapted to fit the required application. "The essence of deep learning is to compute hierarchical features or representations of the observational data, where the higher-level features or factors are defined from lower-level ones." [18] The ability of DL to derive higher-level features makes them ideal for pattern analysis and classification.

2.3.3.1 Recurrent Neural Networks

A Recurrent Neural Network (RNN) is an adaptation of feed-forward neural networks (section 2.3.2.5) in that they use cyclic connections making them more powerful in modeling sequences of data [19]. Using backpropagation, the input weights between the hidden layers are fine-tuned and calibrated. They can be used to predict and model sequence data using previous data samples, with no additional class information. The basic RNN cell is shown in Figure 11 and described by equation (1) [20].

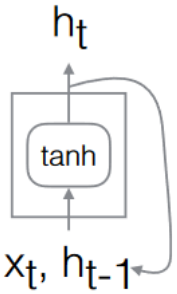


Figure 11 – Basic RNN Cell [21]

The output h_t is a function of both the input x_t and the previous output h_{t-1}:

$$h_t = f(Wx_t + Uh_{t-1} + b) \quad (1)$$

where W and U are weight matrices, b is a bias term, and f is a non-linear transformation (e.g. the sigmoid or tanh function). RNNs make use of present x_t information and the past h_{t-1} information, however they are difficult to train since the weights W and U are adjusted according to the derivative of the output. This feedback loop that causes errors to shrink or grow exponentially and are known as the vanishing and exploding gradient problems. Where the vanishing gradient problem is the issue that occurs as the RNN updates the weights through backpropagation, the update becomes vanishingly small preventing the network from further training. Shown in Figure 12 "The shading of the nodes in the unfolded network indicates their sensitivity to the inputs at time one (the darker the shade, the greater the sensitivity). The sensitivity decays over time as new inputs overwrite the activations of the hidden layer, and the network 'forgets' the first inputs." [22] Opposingly, the exploding gradient deals with the update becoming too big, making the model completely ineffective at learning.

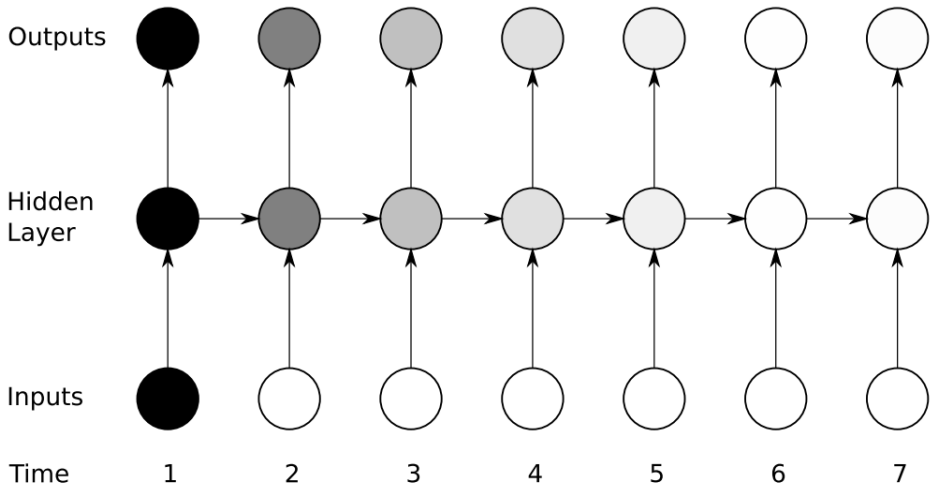


Figure 12 – The Vanishing Gradient Problem for RNNs [22]

2.3.3.2 Long Short-Term Memory Network

Long Short-Term Memory (LSTM) networks were developed by Hochreiter et al. in [20] to deal with the exploding and vanishing gradient problems associated with RNNs. LSTMs are composed of cells with, an input gate i_t, an output gate o_t and a forget gate f_t shown in Figure 13 from [21].

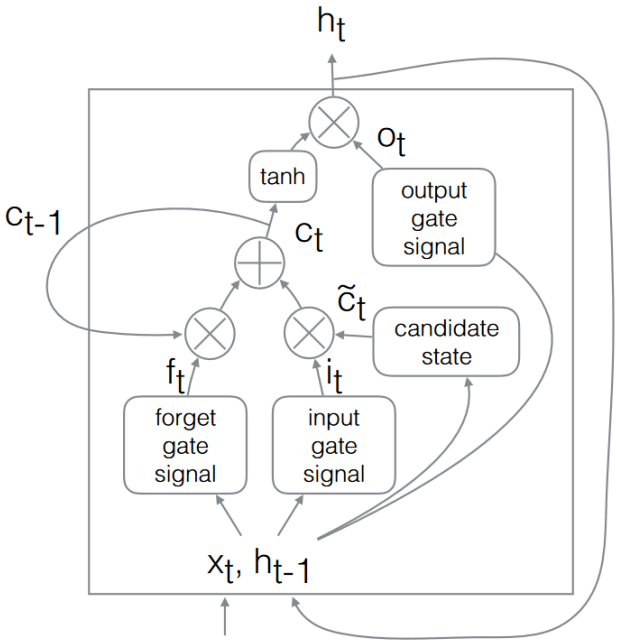


Figure 13 – LSTM Basic Cell[21]

The gates depend on both the input x_t and the previous output h_{t-1}, and controls how much the internal state depends on the new input and previous state using learned weights:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad (2)$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \quad (3)$$

where W_i, U_i, W_f, U_f, are the learned weights, and b_i and b_f are learned bias terms. The sigmoid function σ is typically used for the gate terms to scale their output

between 0 and 1. Equations for i_t (2) and f_t (3) determine the new cell state c_t as a linear combination of the previous c_{t-1} and the new candidate internal state $\tilde{c}_t$:

$$\tilde{c}_t = tanh(W_c x_t + U_c h_{t-1} + b_c) \quad (4)$$

$$c_t = f_t c_{t-1} + i\tilde{c}_t \quad (5)$$

where W_c and U_c are weight matrices and $b_c$is the bias. The output h_t, is controlled by a gate function o_t that also has it own weights W_o, U_o and b_o as the bias:

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad (6)$$

$$h_t = o_t \tanh(c_t) \quad (7)$$

Each cell is meant to remember values over an arbitrary amount of time, with the three gates regulating the flow of information in and out of the cell. Overall, the LSTM architecture enables a more effective means of modeling a dataset than an RNN, as LSTMs do not suffer as significantly from errors in back propagation when updating the weights of each gate.

As with RNNs, LSTMs can be trained to predict the next set of data values in a sequence. However, due to the optimizations of LSTMs they are now more widely used than traditional RNNs. LSTMs are used in technologies such as speech recognition software, handwriting recognitions, and notably, IDSs [21], [23]. Because an LSTM can be used to predict the values in a series, the difference between the predicted value and the actual value can be used to detect anomalies if the difference is above a set threshold. An LSTM model can also be optimized by selecting the appropriate hyper-parameter setting explained in the next section.

2.3.3.3 LSTM Network Hyper-parameters

Hyper-parameters in ML, refer to attributes or values that are used to control and fine-tune the learning process of a ML model. The values are set prior to training a

model and affect the speed and quality of the leaning process [24]. Hyper-parameter search is commonly performed manually, via rules-of-thumb [25] or by testing sets of hyper-parameters on a predefined grid [26]. Bergstra et al. in [27] states that the task of determining hyper-parameter settings for deep architectures is an extremely difficult and resource intensive optimization task. Some common hyper-parameters used in training LSTMs are defined by Gaillard et al. in [28] and in TensorFlow's application programming interface (API) documentation in [29]:

1. Epochs: Indicates the number of passes of the entire training dataset the ML algorithm has completed.
2. Time-steps or input length: Refers to the length of input sequence used to predict the output sequence.
3. Batch size: Refers to the number of training examples that are utilized in one iteration before updating the model's internal parameters.
4. Layer count: The number of trainable layers (LSTMs, RNN, etc.) a model contains.
5. Neuron count: The number of nodes that make up each layer of a model.
6. Early Stopping: Stops training when a monitored metric has stopped improving.

2.3.3.4 Autoencoders

Autoencoders are a type of ANN used to learn efficient data encodings in an unsupervised manner [30]. They aim to learn the representation or encoding of a set of data by training the network to ignore signal "noise". An autoencoder consists of an encoder and a decoder shown in Figure 14. The encoder takes high-dimensional input data and translates it into latent low-dimensional data. The decoder then receives the encoder's output data with the objective to reconstruct the original high-dimensional input data. There would then be some reconstruction error, but if a model is trained on known "good" data a threshold would be set to an acceptable

level. This technique can then be used for anomaly detection by comparing the reconstruction error of data with and without anomalies. Usually, a high reconstruction error, over a set threshold would be indicative of an anomaly.

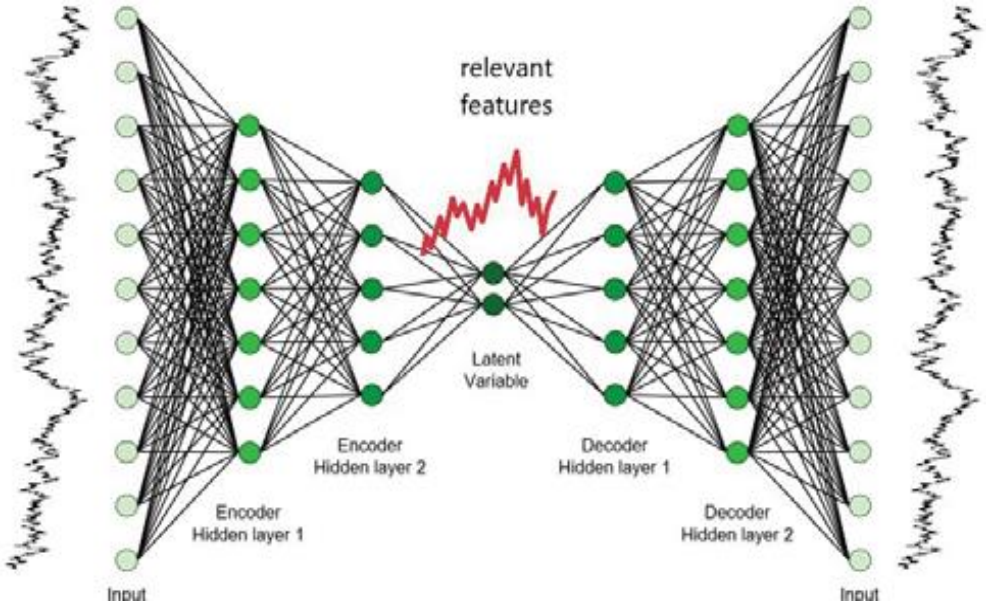


Figure 14 – Conceptual Example of an Autoencoder [30]

2.3.3.5 LSTM Autoencoders

An LSTM autoencoder uses LSTM layers to learn the compressed representations of a dataset [30]. It combines the best of an autoencoders ability for signal reconstruction and an LSTM's ability to effectively learn and model a dataset. The main differences of a LSTM autoencoder and a regular autoencoder is that the main blocks of the network architecture are LSTM cells (section 2.3.3.2) as shown in Figure 15.

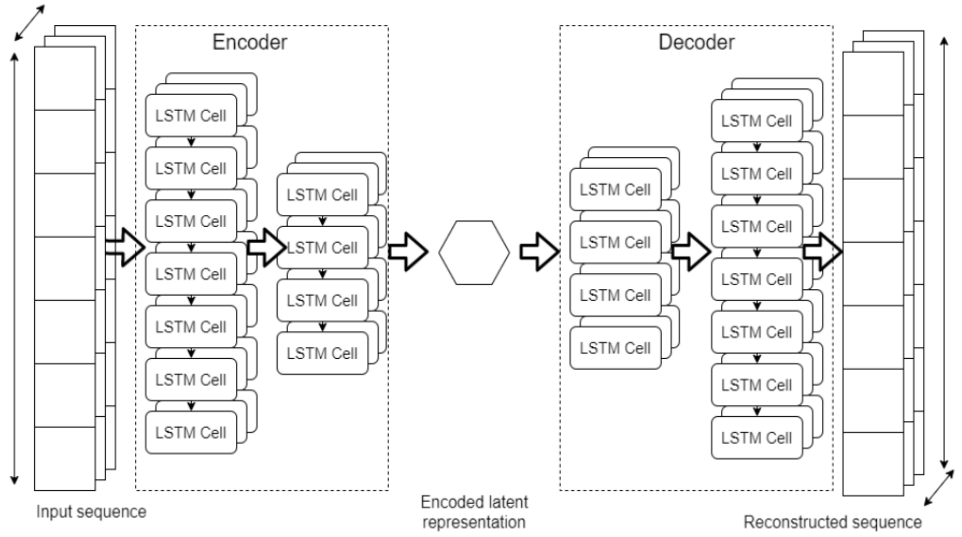


Figure 15 – Architecture of a LSTM Autoencoder [31]

An LSTM autoencoder's, effectiveness in reconstructing the input signal or input values is indicated by its reconstruction error. One method for calculating the reconstruction error is by using the Mean Absolute Error (MAE). The MAE is the absolute difference between the actual and predicted values [32] defined as:

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|x_i - y_i| \tag{8}$$

where m represents the total number of data points in the time series, x_i is the real measured time series in the original scale of the dataset, and y_i is the predicted output of the time series [32]. The higher the MAE value the more inaccurate the predicted values were from the input values, the opposite being true for a lower value. This is useful when determining if data contains anomalies, as the more anomalies contained in the data, the more difficult it will be to recreate the input data resulting in a higher MAE value for each recreated value. The MAE can then be used as a method for setting a threshold value, where any recreated value that is higher than the threshold

can be considered anomalous. One method for setting the appropriate threshold value is using the standard deviation (σ) calculated from the MAE. Standard deviation is defined as:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}} \tag{9}$$

where x_i is the value of the i^{th} point in the dataset, $\bar{x}$ is the mean value and n is the number of data points in the dataset. The specified number of standard deviations away from the mean can then be set as the threshold value. The default value being 3 standard deviations away from the mean [33], however depending on the variance of the dataset an appropriate threshold can be set.

2.3.4 Practical Applications of LSTM Networks

There are examples where LSTM networks either alone or in combination with other methods are used for anomaly detection. One such method was purposed by Taylor et al. in [21] using a LSTM RNN to detect anomalies on the automotive Controller Area Network (CAN) bus. The proposed detector accepts raw CAN bus data words as an input and works by learning to predict the next data word originating from each sender on the bus. If there are bits in the next word that are not expected they are flagged as anomalies. This predictive type of modeling would be ideal for behaviour-based anomaly detection, as it is able "to detect data packet anomalies that are unusual only in the context of the rest of the sequence." [21]. With the added benefit that, "The LSTM based anomaly detector can be applied to many different vehicles without substantial modification." [21].

As described in Provotar et al. [30] autoencoders that use deep encoders and deep decoders offer many advantages over the standard single layer encoders and single layer decoders. The depth can exponentially reduce the computational cost of representing some functions. It can exponentially decrease the amount of training

needed to learn some functions. As well, experimentally, deep autoencoders yield a better compression compared to linear autoencoders. Reference [30] shows the application of LSTM autoencoders for anomaly detection with a high degree of successful anomaly detection.

In recent years, there has been additional research in the field of using LSTM Autoencoders for anomaly detection, shown in works like: [34], [35], and [36]. Maleki et al. in [34] apply a LSTM autoencoder to detect anomalies in CPU utilization, in a cloud computing environment. They concluded that an advantage of their applied method was the detection of both abrupt and gradually developing anomalies. Said Elsayed et al. in [35] use a hybrid method of a LSTM Autoencoder combined with a OCSVM to detect anomalies in web traffic. Their method showed promising results, however due to the nature of web traffic, further real-world testing was required to confirm the detectors viability. Kabore et al. in [36] review multiple deep learning techniques for anomaly detection on ICS systems. The use of LSTM and stacked Autoencoder based anomaly detectors were demonstrated to achieve positive anomaly detection results.

The success of the LSTM RNN model in [21] for anomaly detection on the CAN bus is a good indication that LSTMs will work well for anomaly detection on other bus like networks. Combined with the promising anomaly detection rates of the LSTM autoencoder presented in [30], [34] and [36], the MIL-STD-1553B data bus would be a good test case for anomaly detection using a LSTM autoencoder DL method. Leading the focus of this research to explore the effectiveness of the LSTM autoencoder DL method for anomaly detection on MIL-STD-1553B networks.

2.4 Existing MIL-STD-1553B Detection Methods

Four anomaly detection methods that have been developed specifically for MIL-STD-1553B networks have been explored in this research. First is a Markov chain

model described by Stan et al. in [2], second is the RT authentication module or RT Fingerprinting described in Stan et al. [3], third is a MIL-STD-1553B Anomaly-Based Intrusion Detection System (MAIDENS) using time-based histogram comparison [1], and fourth is a signature based method proposed by Charles Bernard in [4].

2.4.1 Markov Chain Model

Stan et al. in [2] proposed a Markov chain model anomaly detector for the MIL-STD-1553B bus based on transmitted periodic and aperiodic messages. By extracting command and timing features from MIL-STD-1553B messages such as source terminal Address (TA), destination TA, time cycle, they could be used to define Markov model states (section 2.3.2.2). The aim of this detector was to address three types of attack scenarios: A DoS attack, a spoofing attack on a bus with a significant amount of traffic, and a spoofing attack on a less busy bus. In the test environment described in [2] they were able to use the model against the three attack methods and detect all anomalies with a zero false positive rate. Further to this, to validate their false positive rate, they applied their learning model to real MIL-STD-1553B log data. They found that they needed to train their model for around 5 seconds to achieve a near zero false positive rate. They noted the inability of their model to trigger anomalies when a corrupt RT is spoofing the BC or another RT and only manipulating the data words.

2.4.2 RT Fingerprinting

Stan et al. in [3] propose a RT authentication module that builds on their previous work in [2] that utilizes supervised and unsupervised ML algorithms to characterize legitimate signals and generates fingerprints for the system components. “The proposed module is based on analog signal analysis and utilizes the components’ hardware and manufacturing inconsistencies for this purpose. The signal analysis produces a unique profile for each monitored RT, which is later used for

authentication. This method is able to detect illegitimate connected devices to the data bus, as well as spoofing attacks." [3] Their RT authentication process is detailed in Figure 16. In combination, with their sequenced-based module, the detector has a high detection and low false positive rate for the tested attacks. However, their detector did not cover attack methods that utilize only data and status words.

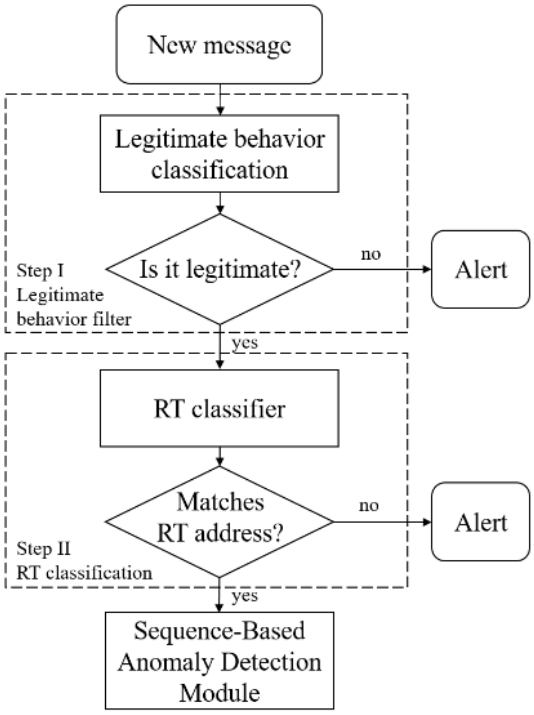


Figure 16 – RT Authentication Process [3]

2.4.3 MAIDENS

MAIDENS [1] is a statistical anomaly-based intrusion detection system that uses a time-based histogram comparison method based on research conducted by Losier et al. in [37]. The time-based features of the MIL-STD-1553B bus are used to create a baseline histogram representation of known, intrusion-free data. By plotting the frequency of the values of a given feature, they are able to compare run-time data to

the baseline data and if it is a certain percentage away from the baseline it would be considered anomalous [1]. Based on the results in [1] the presented detector was able to identify the start and stop times in which a threat occurrence took place across five scenarios with no false positive or negative results. The five scenarios included four DoS type attacks: DoS using Command words, DoS using Status words, RT specific DoS and RT specific sub-address DOS. It also included one data manipulation attack: RT Hijack, all described in section 2.2. The detection times ranged from within 0.38 seconds to 47.11 seconds of an attacks start and stop time. This equates to detecting an attack within 380 to 47,110 messages, assuming a bus averages one message every 0.001 seconds, which is not ideal when investigating an attack or threat instance given the potential for a significant number of bus frames to analyse that may not be related to the attack. Overall, the MADIENS detector is effective at detecting attack threat occurrences, however the accuracy of the detection times could be improved upon to assist in the investigation of detected threat occurrences.

2.4.4 Signature Based Detection

The method proposed by Bernard in [4] is a signature-based intrusion detection method for MIL-STD-1533. It also utilized two anomaly-based detection techniques: word repetition analysis and RT frequency analysis. Word repetition analysis attempts to detect attacks that would use repeat words where they otherwise would not be. The example given is where an attacker impersonates a RT by sending status words close in time to legitimate ones [4]. RT frequency analysis detects attacks by analysing the frequency RT addresses appear in traffic samples. Any deviation from the expected frequency of transmission could be indicative of a defective device or an attack. The research demonstrated that signature-based detection was effective at detecting attacks that have clear attack indications, like a command word-initiated DoS (BC spoofing) that uses the transmitter shut down and start up mode codes or status word data leakage attack that hides data in reserved

bits in the status word. When combined with RT frequency analysis, signature-based detection was effective at detecting unauthorized RT address if an attack uses an RT address that is not supposed to be used on the bus.

Signature-based detection however, was shown not to be effective at detecting a compromised RT scenario where a compromised RT changes the number of command or status words it receives or responds to. The word repetition analysis implemented was also demonstrated not to be an effective means of detection. In both of these scenarios, a usable signature could not be generated that would reliably trigger a detection event without an excessive amount of false positive detections. Overall, the application of signature-based network security monitoring is a viable tactic for detecting undesirable activity on a MIL-STD-1553B bus. However, signature-based detection requires the collection of known attacks or illegal messages on the MIL-STD-1553B data bus in order to flag undesirable actions. It was demonstrated that this method was ineffective of detecting data integrity attacks as no signatures could be set for the data.

2.4.5 Demonstrated Anomaly Detection on MIL-STD-1553B

The state of anomaly detection on MIL-STD-1553B has certainly improved since its inception. The aforementioned detectors have demonstrated their ability to detect anomalies over different threat scenarios. These scenarios have been generalized and presented in Table 2. As interest in this fields grows more detectors and methods will be created, as well as the ways in which threat actors can carry out attacks. There is always a need to continue to mature and develop means of detection, as well improve the accuracy and precision of these detection techniques so that they stay relevant and useful in real-word application.

Table 2–- Capabilities of Current Anomaly Detectors for the MIL-STD-1553B Data Bus

Attack Type and/or Level of Detail Detected		MAIDENS [1]	Markov Chain Model[2]	RT Fingerprinting [3]	Signature Detection[4]
1	Detection of Denial of Service (DoS)	Able	Able	Able	Able
2	Detection of Targeted RT Denial of Service (RT DoS)	Able	Able	Able	Able
3	Detection of BC spoofing (Data Integrity)	Able	Able	Able	Able
4	Detection of Covert Channel Attack (Data leakage)	Not Tested	Not Tested	Not Tested	Able
5	Detection of RT message manipulation (Data Integrity)	Able	Unable	Unable	Unable

2.5 Detection Methods

There are many validation techniques available for determining the effectiveness of detection methods. Precision, Recall and Accuracy in terms of classification are good measures when the data contains labeled anomalous and benign data to determine false and true negative rates. As well as, the standard measure of accuracy and precision used in this work when determining detection times of anomalous event. The following section will describe these means of validation.

2.5.1 Precision, Recall and Accuracy (Classification)

The performance metrics that are normally collected for evaluating the effectiveness of ML methods or for classification are: precision, recall and accuracy [38]. In order to calculate these metrics, the following will be collected for each method:

- True positive (t_p): detector identifies an anomaly correctly
- False positive (f_p): detector incorrectly labels benign traffic as anomalous
- True negative (t_n): detector identifies benign traffic correctly
- False negative (f_n): detector incorrectly labels anomalous traffic as benign

Precision identifies the proportion of positive identifications that were actually correct, defined as:

$$\text{Precision} = \frac{t_p}{t_p + f_p} \tag{10}$$

Where a precision of 1 means that the predictor had no false positives or found all of the anomalies and a 0 means that the predictor only had false positives and no anomalies were found. However, this does not take into account the false negatives that were predicted. This is where recall is used to identify the proportion of true positives that were identified correctly. Defined as:

$$\text{Recall} = \frac{t_p}{t_p + f_n} \tag{11}$$

Where a recall of 1 means that all anomalies were identified correctly and a 0 meaning that they were all labelled incorrectly. Accuracy is then the fraction of predictions that the model has identified correctly defined as:

$$\text{Accuracy} = \frac{t_p + t_n}{t_p + f_p + t_n + f_n} \tag{12}$$

Using all three metrics gives a better representation of the data as the accuracy alone does not show if there is a significant disparity between the number of positive and negative labels. If a method does not produce false positive or false negative results these metrics will not be necessary as the method perfectly detected an event.

2.5.2 Accuracy and Precision (Detection Time)

Accuracy is defined as the closeness of agreement between a test result and the accepted reference value [5]. Precision is defined as the agreement between independent test results obtained under stipulated conditions. Precision depends on the distribution of random errors and does not relate to the true value or the specified value [5]. For this work detection time accuracy ($Accuracy_{DT}$) will be in terms of average detection time and is defined as:

$$Accuracy_{DT} = \frac{\sum x}{n} \tag{13}$$

where x is the sum of the event detection time and n is the number of detection events. Detection time precision ($Precision_{DT}$) will be in terms of the mean absolute deviation (MAD) of the event detection times defined as:

$$Precision_{DT} = \frac{1}{n}\sum_{i=1}^{n} |x_i - \bar{x}| \tag{14}$$

where n is the number of detection events, x_i is the i^{th}number in the events and $\bar{x}$ is the average event detection time. Combined, these validation metrics will be used to determine how accurate and precise a detector is at detecting an event occurrence. In this work we will use $Accuracy_{DT}$ (detection time accuracy) and $Precision_{DT}$ (detection time precision) as measure to indicate the start and stop of an attack.

2.6 Summary

With the currently available detection methods for MIL-STD-1553B each method has its strength when it comes to anomaly detection. Table 2 showcases a number of detection methods and what they are able and unable to detect. The signature-based detection method can detect anomalies with known anomalous signatures, the Markov chain method can detect DoS and spoofing attacks and MAIDENS can detect attacks that involve data manipulation. While, these methods are able to detect anomalies, the degree of $Accuracy_{DT}$ and $Precision_{DT}$ are just as important when it comes to analysing any detected anomaly. The advantages then of using ML and DL methods, would then be a method that could accurately detect all known attacks and possibly unknown attack methods, while increasing in the $Accuracy_{DT}$ and $Precision_{DT}$. Looking at the applicability of the LSTM autoencoder DL method, the next chapter will go on to describe the methodology and design of applying this method for detection of anomalies on a MIL-STD-1553B data bus, with the intent of increasing the detection time accuracy and precision.

Chapter 3

Methodology and Design

This chapter outlines the research methodology and design used to develop an anomaly detector that utilized the LSTM autoencoder DL technique described in Chapter 2 for the MIL-STD-1553B data bus. The first phase is the development of a data acquisition pipeline that will be used to collect and prepare MIL-STD-1553B bus traffic. This data acquisition pipeline will also extract defining features of the data. The second phase is the creation of a LSTM-based autoencoder method that will utilize the data from the previous phase for baseline creation and anomaly detection. Finally, the third phase is the validation of the DL model's ability to accurately and precisely detect report on anomalies.

3.1 Phase 1: Data Acquisition Pipeline

In order to perform detection on a set of data with a ML or DL technique it has to be in a viable format to be ingested by the detector. This set of data points can be in the form of a stream of data, like a sequence of messages from a MIL-STD-1553B data bus. In order to collect and create a baseline dataset of MIL-STD 1553B network traffic, a bus recorder was used to record the communications of simulated and/or emulated RTs over a physical MIL-STD 1553B data bus shown in Figure 17. The MIL-STD 1553B data bus was setup similarly to the generic test bench and collection architecture also shown in Figure 17. The data collected was that of a "benign" flight, consisting of an aircraft in flight, where no anomalies were enacted. Once the baseline dataset was collected it was used to create a baseline model in the next phase.

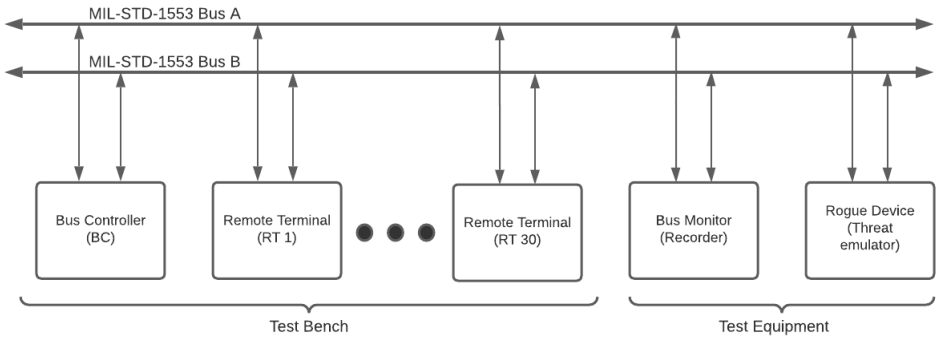


Figure 17 – Generic MIL-STD-1553B Test Bench and Collection Architecture

The collection of anomalous traffic is similar to the collection of the baseline data except for the presence of a rogue device connected to the physical MIL-STD-1553B bus, to simulate a maliciously attached device that produces anomalies on the bus in the form of different scenarios. The scenarios were pulled from the following types of attacks:

1) DoS
    a) Network Disruption (Command word DoS)
    b) Network Disruption (Status word DoS)
2) Targeted RT DoS
    a) RT deny
    b) RT subaddress deny
3) Data Integrity Violation
    a) RT hijack

Using the tool presented by Paquet in [6] each of the above five attack scenarios can be carried out on a MIL-STD-1553B bus under test. The attack methodology is laid out in Chapter 2.2 and each attack is described in more detailed in Table 3.

Table 3 – Scenario Attack Descriptions

Scenario	Brief Description
1a)	A complete data bus DoS attack where a rogue RT in BC mode uses commands words transmitted on the data bus in order to deny all bus communication.
1b)	A complete data bus DoS attack where a rogue RT in BC mode transmits status words after each normally transmitted command word transmitted on the data bus to deny all bus communication.
2a)	A targeted DoS attack where a rogue RT in BC mode transmits status words after each normally transmitted command word intended for a specified RT to deny communication to or from that RT.
2b)	A targeted DoS attack where a rogue RT in BC mode transmits status words after each normally transmitted command word intended for a specified RT sub-address to deny communication to or from that RT sub-address.
3a)	The implementation of this attack cannot be disclosed due to it being proprietary in nature, however it essentially causes the data for an intended recipient RT to be modified and accepted as if it were the intended values.

Attack scenario 1a) is a DoS attack that uses command words transmitted on the data bus by a rogue RT in order to deny all bus communication. Attack scenario 1b) is also a DoS attack that transmits status words after each normally transmitted command word to achieve the same outcome. Attack scenarios 2a) and 2b) transmit status words after a command word is transmitted for the targeted RT or RT sub-address. The exact implementation of Attack scenario 3a) cannot be disclosed due to it being proprietary in nature. It is however, essentially causing the data for an intended recipient RT to be modified and accepted as if it were the intended values and achieves the same desired outcome as the man in the middle attack described in section 2.2.3.1.

3.1.1 Feature Extraction

The raw data from the data collection phase cannot be directly fed into a DL model, therefore the data must be prepared in order to do so. This process is known as feature extraction. The data is transformed by deriving each feature manually or by other means of automatic feature generation. For the purpose of this research the chosen features were determined from previous works of MAIDENS [1] and Stan et al. [2]. As well, additional features were created by extracting features from the Abaco BusTools BMDX format [39].

The initial feature set available for MIL-STD-1553B are described in the works of MAIDENS [1] and Stan et al. [2] shown in Table 4. This feature set defines the fundamental features of a message on the data bus, but can be expanded upon to better describe the entirety of the messages.

Table 4 - Features of MIL-STD-1553B Network Traffic [1], [2]

	Feature name	**Description**
Command Features	Source Terminal Address	The address of the RT sending the data.
	Source Sub-Address	The sub-address of the sending RT.
	Destination Terminal Address	The address of the receiving RT.
	Destination Sub-Address	The sub-address of the receiving RT.
	Channel	The channel on which the message was sent.
	Word Count	The number of data words sent in the message.
	Is Mode Code	Whether the command is a mode code or not.
Time-Based Features	Time Cycle	Time cycle of the message.
	Response Time	Time for an RT to respond to a command.
	Inter-message Gap	Time from end of one message to the start of the next.
	Periodicity	Mean time difference in start times of messages over a given period.
	Bus Utilization	Percent use of the bus over a given time period.
	Data Throughput	Number of data words sent over the bus over a given time period.

Table A-1 in Appendix A was created based on Table 4, less the time-based features, to describe the command and status word features of BC-RT and RT-RT messages. For BC-RT messages the RT-RT specific features (e.g. CMD2-addr, CMD2-TR) are zeroized to standardize input. The BMDX file format, shown in Figure 18, contains more information on a message than what is recorded on the bus. The additional information is contingent on the type of bus recorder; however, the supplementary Message Status information from [39] can be used to extract additional features from a message. The feature list extracted from the Abaco BusTools Message Status word is contained in Appendix A, Table A-2.

```
#define BT1553_MBUFCOUNT 32 /* Data words in api_bm_mbuf struct */

typedef struct api_bm_mbuf
  {
  BT_U32BIT         messno;     // Message number (generated by API, 1-based)
  BT_U32BIT         int_status; // Interrupt status from board
  BT1553_TIME       time;       // Time of message (64-bits, 1 us  or 1ns LSB)
  BT1553_COMMAND    command1;   // 1553 command word #1 (Receive for RT-RT)
  BT_U16BIT         status_c1;  // 1553 command word #1 error status
  BT1553_COMMAND    command2;   // 1553 command word #2 (Transmit for RT-RT)
  BT_U16BIT         status_c2;  // 1553 command word #2 error status

  BT1553_BMRESPONSE response1;  // 1553 response time #1 (byte)
  BT1553_BMRESPONSE response2;  // 1553 response time #2 (byte)
  BT1553_STATUS     status1;    // 1553 status word #1 (Transmit for RT-RT or Broadcast RT-RT)
  BT_U16BIT         status_s1;  // 1553 status word #1 error status

  BT1553_STATUS     status2;    // 1553 status word #2 (Receive for RT-RT, NULL for BCST RT-RT)
  BT_U16BIT         status_s2;  // 1553 status word #2 error status

  BT_U16BIT         value[BT1553_MBUFCOUNT];   // 1553 data words
  BT_U16BIT         status[BT1553_MBUFCOUNT];  // 1553 status for data words (16-bit words)
 }
API_BM_MBUF;
```

Figure 18 – Abaco BusTools BMDX Structure [39]

Table A-3 in Appendix A is a combination of Table A-1 and Table A-2 to compile the 61 features be extracted from the MIL-STD-1553B protocol and used in the LSTM autoencoder model development in the next section.

3.2 Phase 2: DL Model Development

The model development phase utilizes the baseline data recording from the previous phase with the TensorFlow and Pandas Python libraries [29] to create a LSTM autoencoder model of the data. TensorFlow is an open-source software library for ML and can be used to implement DL techniques [40] and Pandas is an open-source data analysis and manipulation tool that can handle large datasets required for DL. [41] The TensorFlow software is a core tool for DL model creation and requires the user to program and configure their desired model, which is described further in the following section.

3.2.1 LSTM Autoencoder Model Setup

The input shape of the initial LSTM autoencoder layer is recommended to be about half of the number of input features [30]. Since the feature set from Phase 1 contains 61 features the initial LSTM layer was chose to have 30 nodes. To make the encoder portion of the autoencoder, the next LSTM layer should be a reduction in the number of nodes. It is recommended the reduction in nodes is at least 50% [30], making the second layer have 15 nodes. The last LSTM layer is then expanded back to the size of the original layer, back up to 30 nodes, to make the second and third layers the decoder portion of the autoencoder. The last layer of the model is the output layer, which is a dense layer that is used to change the dimensions of the last LSTM layer to map the output to the number of input features [42]. The final proposed LSTM autoencoder model is shown in Figure 19 and is a numerical representation of the design described in Chapter 2.

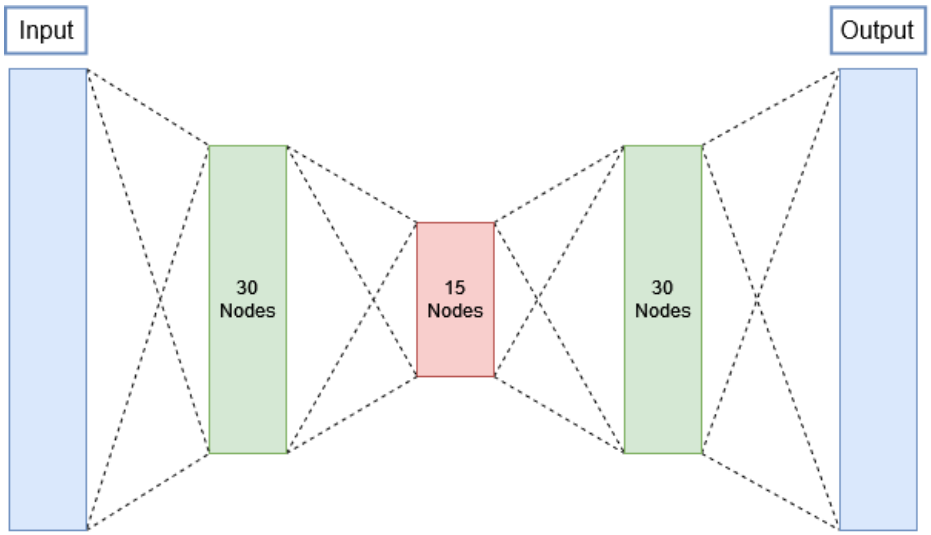


Figure 19 – Proposed LSTM Autoencoder Layers

3.2.2 LSTM Autoencoder Training

As discussed in Chapter 2, there is a need to test different hyper-parameter settings to find a balance between the optimization of model creation and the time to produce a viable model. Hyper-parameter tuning is problem dependant and will vary greatly depending on the data set. The works of Ranjan [43] recommends an initial list of hyper-parameters when initiating testing of model design. The main hyper-parameter settings that are recommended are: 300 epochs, a time-step of 1, a batch size of 5 and no callback for early stopping. These initial values will be adapted through experimentation to achieve the desired detection accuracy and optimum training time.

3.2.3 LSTM Autoencoder Threshold

Once a LSTM autoencoder is trained the threshold value for anomaly detection will need to be determined. Using the metrics described in Chapter 2 the threshold can be set. The threshold can be set anywhere from 2-3 standard deviations from the average MAE. The threshold value should change based on the variability of the

data, but once set should remain the same throughout all scenarios for comparison. Setting the threshold too high may lead to missing event detections or false negatives and setting the threshold too low may lead to too many event detections or false positives. The initial threshold value will need to be set through a series of trial and error until acceptable results are achieved. Once the threshold is set, the model is now ready to accept recordings for anomaly detection. The next phase will describe how the results of the DL model will be validated.

3.3 Phase 3: Anomaly Detection and Validation

This phase will determine the LSTM autoencoder's ability to detect anomalies from the different recorded scenarios. In order to trigger an anomaly detection event, the feature extracted data from phase 1 is run though the LSTM autoencoder model that was trained on the baseline data in phase 2. The detector will then generate a MAE for each message indicating how well it was able to recreate each message feature. If the MAE is higher than the determined threshold value then that message will be identified as anomalous.

Validation activities include comparing performance and time metrics between MAIDENS and the LSTM Autoencoder. To enable this, the $Accuracy_{DT}$ and $Precision_{DT}$ of the MAIDENS and LSTM Autoencoder detectors are calculated. The $Accuracy_{DT}$ will then be taken from the three attack events in each scenario as the average detected start and stop time. The closer the average is to zero the more accurate the detection times are for that scenario. $Precision_{DT}$ will also be calculated using the mean absolute deviation described in Chapter 2 for each scenario.

3.4 Summary

The three phases above describe the process for anomaly detection using a LSTM autoencoder. The first phase consists of the collection of a data pipeline for baseline recordings as well as recordings that will contain anomalous traffic. The second phase is the creation of a baseline LSTM autoencoder model and subsequent anomaly detection from the data collected in phase 1. Finally, the third phase is the validation of the LSTM autoencoder detector's ability to detect anomalies. This flow is necessary to properly collect and validate a dataset appropriately, the results from which are described in the following Chapter.

Chapter 4

Results

In this chapter, a LSTM autoencoder anomaly detector is built and evaluated using the design developed in Chapter 3. The implementation of the LSTM autoencoder model is detailed, followed by a description of the two datasets from separate MIL-STD-1553B network architectures. The results from the detector are then presented and compared to a statistical anomaly detector for the first of the two datasets. The detector is then evaluated on its ability to accurately and precisely detect the start and stop time of anomalies in both datasets.

4.1 Experimental Design

The LSTM autoencoder anomaly detection pipeline was implemented using the following hardware and software components:

- Processor: AMD Ryzen 7 3800X 8-Core
- RAM: 64 GB DDR4 Memory
- GPU: AMD Radeon RX 5700 XT 8 GB GDDR6
- SSD: 1 TB NVMe Gen 3
- Operating System: Windows 10
- Abaco BusTools Software
- Abaco BusTools Bus Recorder
- TensorFlow library
- Scikit-learn toolkit
- Pandas library
- NumPy library

4.1.1 LSTM Autoencoder Model

The LSTM autoencoder model that was used for this experiment was composed of three LSTM layers that follows the design outlined in Chapter 3. Table A-3 in Appendix A details the 61 features that have been extracted from the MIL-STD-

1553B protocol and will be used as the input for the LSTM autoencoder. As outlined in Chapter 3, the initial layer had 30 nodes. The next layer was then composed of half as many nodes, down 15 nodes, making the first two layers, the encoder portion of the LSTM network. The last LSTM layer was then expanded back to the size of the first layer, up to 30 nodes, to make the second and third layers the decoder portion of the LSTM network. The output of the last layer feeds the results into a 61-node dense layer, one node for each feature. The final LSTM autoencoder network is shown in Figure 20 and resembles the design described in Chapter 3.

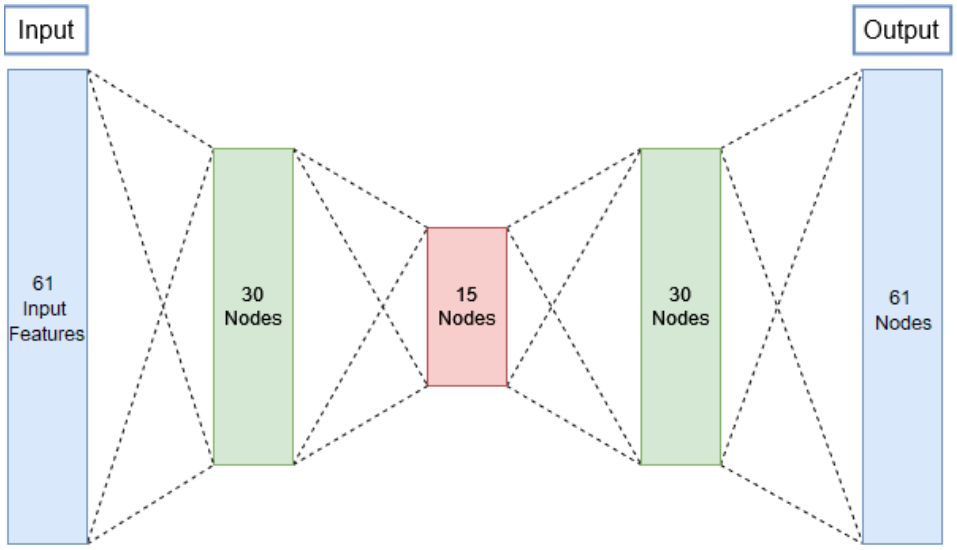


Figure 20 –LSTM Autoencoder Layers

As discussed in Chapter 3, there was a need to test different hyper-parameter settings to find a balance between the optimization of model creation and the time to produce a viable model. The works of Ranjan [43] were used as a starting point for the initial hyper-parameter tuning. Initially, the model was trained with 300 epochs, a time-step of 1 and a batch size of 5 with no callback for early stopping. After some fine tuning, the hyper-parameters that were selected for model creation were: 100 epochs, a time-step of 1, a batch size of 2000, and an early stopping callback if there

was no significant change in validation (MAE) loss for three consecutive epochs. These final hyper-parameter settings decreased the time to train the LSTM autoencoder, from a few hours to around half an hour to train. The hyper-parameter tuning was not the focus of this research, but is normally part of the model development process and was conducted in order to allow the model to train in a faster period of time without negatively effecting the results.

4.1.2 Datasets

There are two datasets used for this research. Dataset 1 is the dataset that was used in the MAIDENS [1] research and was collected from the CP140 Aurora test bench in CFB Greenwood, Nova Scotia and was composed of 15 RTs. Dataset 2 was collected on a proprietary MIL-STD-1553B emulated network provided by RMC in Kingston, Ontario and was composed 14 RTs. Each dataset contains one baseline data recording of normal data bus activity and separate recordings that contain scenarios with different attack types being carried out on the data bus. Each recording is that of an aircraft in straight and level flight. A limitation that was noted was the inability to label each individual message as anomalous or benign for each attack scenario. Instead, each attack is recorded and labeled by the initial start and stop time of an attack, marking entire sections of recordings as anomalous. Both datasets will be described in more detail in section 4.2 and section 4.3.

4.1.3 Results Overview

The LSTM autoencoder results for both datasets detected every anomaly occurrence with no false positive or false negative results, much like the MAIDENS detector for Dataset 1. While reporting an event classification accuracy, precision and recall of a 100% looks ideal, this comes from each attack in each dataset being labelled by the start and stop time of an attack occurrence and not having each individual message labelled (a limitation of the attack toolset being used). So, while each detector detects every attack occurrence, there is no way of confirming if every message that falls

above the set threshold measurement is a message associated with an attack (a false positive) and the same goes for every message that falls below the threshold, it cannot be confirmed for sure that it is not associated with the attack (a false negative). Therefore, the better metric to use in the case of these datasets, is the time-related performance metrics for $Accuracy_{DT}$ and $Precision_{DT}$, specifically of the start and stop time of each attack occurrence. When using these metrics, for every scenario in the dataset 1, the LSTM autoencoder greatly increased these time-related performance metrics when compared to the MAIDENS detector and had favorable results of its own for dataset 2. To note, each detector included results that indicated an attack occurrence occurred before the actual start time of an attack. It is hypothesized that this is due to the detector triggering on an individual word of a message, however due to the limitations of recording tools, timings are by message and not by individual words of a message. The results for each dataset will be discussed in more detail in the remainder of this Chapter.

4.2 Dataset 1 – MAIDENS Dataset

The MAIDENS [1] research used data from a CP140 Aurora test bench, consisting of real and simulated RTs. The BC master schedule and exact network design is proprietary in nature and was not disclosed. However, Figure 21 outlines the general architecture and data collection setup.

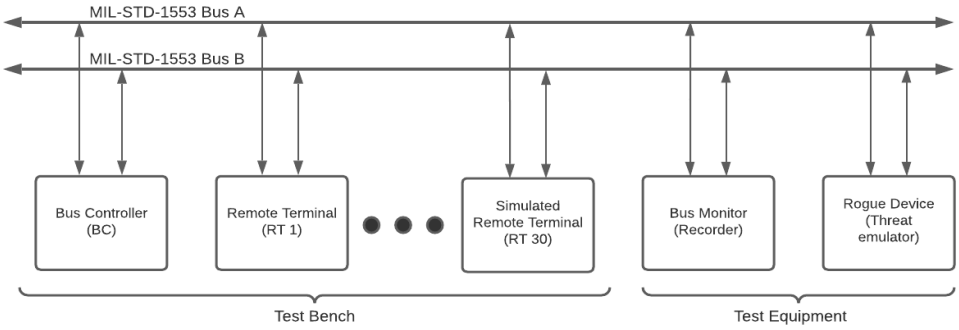


Figure 21 - High-Level CP140 MIL-STD-1553B Test Bench and Collection Architecture

The MAIDENS data consists of six BMDX data files, described in Chapter 3, that have been converted into a comma-separated values (CSV) format. The first file is the baseline data file that was recorded with no anomalous behaviour enacted on the bus and representative of an aircraft in a straight and level phase of flight. The other five BMDX files contain the recording for each of the five threat scenarios described in the work of Paquet [6] and are composed of DoS, targeted RT DoS and Data Integrity Violation attacks as outlined in Chapter 3:

1) DoS
    a) Test1a: Network disruption (Command word DoS)
    b) Test1b: Network disruption (Status word DoS)
2) Targeted RT DoS
    a) Test2a: RT deny
    b) Test2b: RT subaddress deny
3) Data Integrity Violation
    a) Test3a: RT hijack

The five recordings each contain about 20 minutes of bus data, contain three occurrences of a single type of threat scenario. Each threat scenario was carried out

approximately every 5 minutes, with each occurrence lasting for approximately one minute. The exact times when the scenarios started and ended were recorded.

4.2.1 MAIDENS Detector

The Maidens detector [1] was provided by RMC, and is a statistical anomaly-based detection system that uses time-based histogram comparison, described previously in Chapter 2. The detector uses the baseline BMDX file that has been converted into CSV format to create a baseline profile. The baseline profile is then used to detect anomalies in subsequent bus traffic that is analysed by the MAIDENS detector. For this research all of the MADIENS data was re-run through the detector and the same results were obtained as in [1]. The performance statistics are described in the results section below and will be used for comparison with the LSTM autoencoder anomaly detector.

4.2.1.1 MAIDENS Detector Results

Table 5 shows the detection results for each scenario and each threat occurrence within. When determining only whether an attack has occurred, the event classification accuracy, precision and recall were all calculated to be 100% indicating the MAIDENS detector had no issues detecting every attack occurrence (section 4.1.3). Table 6 however, shows the overall $Accuracy_{DT}$ and $Precision_{DT}$ results for each scenario. The MAIDENS detector, detected the start and stop time of the first scenario within 0.38 ±0.15 and 47.11 ±37.53 seconds, respectively. This meaning that the detector was within hundreds to thousands of messages of determining the start and stop time of the threat scenarios. The average start and stop time for all combined scenarios is an average of 0.64 ±0.40 seconds for the start time and 10.27 ±7.98 seconds for the stop time.

Table 5 - Dataset 1 MAIDENS Detection Results (reproduced from [1])

Scenario	Threat Occurrence	Actual Time Of Anomaly (minutes:seconds)		Identified Time Of Anomaly (minutes:seconds)	
		Start	End	Start	End
1a)	1	13:28:57.65	13:30:04.56	13:28:57.47	13:30:19.06
	2	13:35:03.92	13:36:56.54	13:35:03.32	13:37:19.96
	3	13:39:45.94	13:43:16.54	13:39:45.59	13:44:59.94
1b)	1	13:48:49.98	13:49:46.45	13:48:49.15	13:49:47.16
	2	13:53:47.29	13:55:46.75	13:53:46.93	13:55:47.80
	3	13:59:02.40	14:01:47.37	13:59:02.25	14:01:48.31
2a)	1	14:08.47:58	14:09:50.18	14:08:47.50	14:09:51.05
	2	14:13:45.90	14:15:52.45	14:13:45.84	14:15:53.61
	3	14:18:46.66	14:21:46.45	14:18:45.54	14:21:46.74
2b)	1	14:25:54.45	14:26:55.65	14:25:54.15	14:26:55.74
	2	14:30:57.73	14:33:23.53	14:30:56.06	14:33:25.29
	3	14:35:55.50	14:38:56.50	14:35:55.04	14:39:01.41
3a)	1	14:44:51.96	14:45:55.61	14:44:50.91	14:45:56.15
	2	14:49:55.57	14:51:49.18	14:49:55.13	14:51:49.35
	3	14:54:51.33	14:57:27.72	14:54:49.38	14:57:27.91

Table 6 - Dataset 1 MAIDENS Detection Time Accuracy and Precision Results

Scenario	Average Identified Start Time Difference (seconds)	Average Identified End Time Difference (seconds)
1a)	0.38 ±0.15	47.11 ±37.53
1b)	0.45 ±0.26	0.89 ±0.13
2a)	0.41 ±0.46	0.78 ±0.32
2b)	0.81 ±0.57	2.26 ±1.77
3a)	1.14 ±0.54	0.3 ±0.16
Average	**0.64 ±0.40**	**10.27 ±7.98**

4.2.2 LSTM Autoencoder Detector

The LSTM autoencoder detector was created using the methodology and design described in Chapter 3. The detector was created using the Python programming language with the TensorFlow library as the backend for the LSTM model creation. The detector performs two core functions. The first created a baseline model of the MIL-STD-1553B data bus undergoing analysis. The second function would compare subsequent data from the data bus under analysis for anomaly detection. The decoder accepts a converted BMDX file that has undergone feature extraction for baseline creation and anomaly detection in CSV format. The program outputs either a trained LSTM model or a labeled CSV file that contains all of the messages, the MAE of each message and whether each message was determined to be anomalous. The data collection pipeline, model creation, and anomaly detection processes are described in more detail in the following sections.

4.2.2.1 LSTM Autoencoder Data Collection Pipeline

Since the data is provided from the MAIDENS work, the data collection portion of this pipeline was already completed. The set up for the MAIDENS data collection saved the traffic capture in a usable format, in this case the BMDX format. In order for this data to be used by the LSTM autoencoder, feature extraction needed to be performed on the BMDX files. In order to complete this process, described previously in Chapter 3, a program was developed using the Python scripting language to convert the BMDX file type in CSV format into a CSV file with all MIL-STD-1553B features, described in Appendix A, extracted. The program is presented in Appendix D. The product from this pipeline was six CSV files, one for each of the 5 scenario recordings and one for the baseline recording.

4.2.2.2 LSTM Autoencoder Model Training

The CSV files from previous section have been converted from their original data file recordings into a format that is usable by the Pandas and TensorFlow libraries

so that a DL model could be created to represent the dataset. Using the model design described in section 4.1.1, the developed model was trained on the baseline dataset recording from the MAIDENS data. The baseline data file contains 753,896 messages and was split into a training and test set. The training set contains 80% of the data while the test set contains the remainder. Using the training set, a LSTM autoencoder model was created, while the test set was used to confirm the output of the model and used to create the threshold value. As discussed in Chapter 3 the threshold can be set using a value of 2 – 3 standard deviations from the average MAE value from the test set. Using a standard deviation of 2 the threshold for the remainder of the scenarios was 0.4941. Running the complete baseline dataset through the created model, none of the baseline model messages fell above the threshold, shown in Figure 22. This indicated that the model is a good representation of "normal" traffic as discussed in Chapter 3 and any future traffic that is run against this model that falls above the threshold value may be considered anomalous. Figure 23 shows a smaller sub-section of the baseline graph to demonstrate the variations that exist in the MAE values. Due to the RTs being simulated, opposed to being emulated, it is expected that the normal behaviour of the RTs will display very little variation, however it is not as perfect as Figure 22 would indicate.

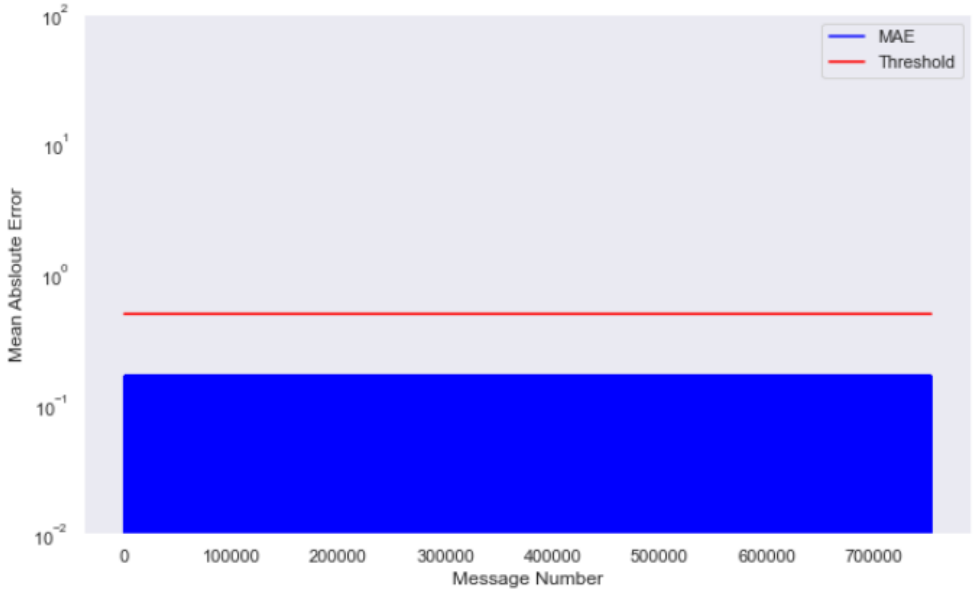


Figure 22 - Dataset 1 Baseline Anomaly Detection Results

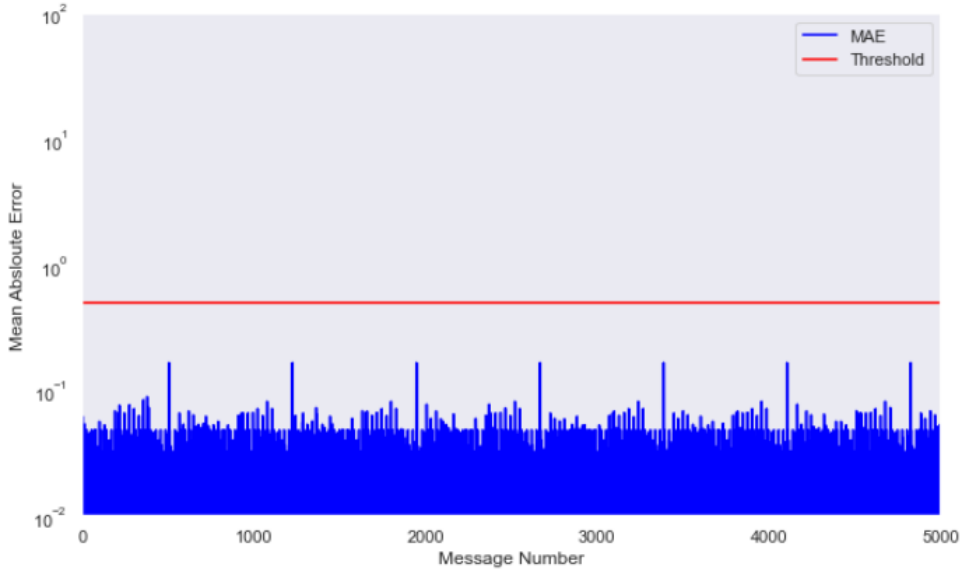


Figure 23 - Dataset 1 Baseline Anomaly Detection Results (Scaled 0 to 5000 Messages)

4.2.2.3 LSTM Autoencoder Anomaly Detection

In the previous section the baseline LSTM autoencoder model was created and a threshold value determined. The five scenario recordings that underwent feature extraction in section 4.2.3.1 can now be ingested by the model. Any of the produced MAE values for each message that fall above the threshold value of 0.4941 will be considered an anomalous message. As discussed in Chapter 3, a group of anomalous messages above the threshold will be considered a single event if no other messages go above the threshold in a 30 second window. This will allow for the determination of the start and stop times of the events. Appendix B showcases output of the LSTM autoencoder anomaly detector for each of scenarios, the results being discussed in the next section.

4.2.2.4 LSTM Autoencoder Results

The threshold value was set in section 4.2.2.2, to a MAE value of 0.4941. This threshold value led all attacks being detected for all five scenarios, showcased in Appendix B. Table 7 shows the detection results for each scenario and each threat occurrence within. When determining only whether an attack has occurred, the event classification accuracy, precision and recall were all calculated to be 100% indicating the LSTM Autoencoder detector had no issues detecting every attack occurrence (section 4.1.3). Table 8 however, shows the overall $Accuracy_{DT}$ and $Precision_{DT}$ results for each scenario. The LSTM autoencoder detector, detected the start and stop time of the first scenario within 0.0060 ±0.0058 and 0.0060 ±0.0031 seconds, respectively. This means that the detector was within 6 ±6 and 6 ±3 messages of determining the start and stop time of the threat scenarios. The overall average start and stop time for all combined scenarios is 0.0512 ±0.0354 and 0.2476 ±0.3194 seconds for determining an anomalous event or 50 ±35 and 250 ±320 messages.

Table 7 - Dataset 1 LSTM Autoencoder Detection Results

Scenario	Threat Occurrence	Actual Time of Anomaly (minutes:seconds)		Identified Time of Anomaly (minutes:seconds)	
		Start	End	Start	End
1a)	1	13:28:57.649195	13:30:04.556062	13:28:57.634574	13:30:04.553993
	2	13:35:03.919435	13:36:56.539377	13:35:03.917005	13:36:56.549938
	3	13:39:45.940323	13:43:16.540626	13:39:45.939523	13:43:16.545886
1b)	1	13:48:49.983588	13:49:46.453073	13:48:49.978587	13:49:46.453779
	2	13:53:47.289629	13:55:46.753689	13:53:47.287051	13:55:46.753755
	3	13:59:02.395148	14:01:47.36972	13:59:02.392748	14:01:47.352389
2a)	1	14:08.47:578865	14:09:50.178645	14:08:47.573866	14:09:50.173645
	2	14:13:45.897814	14:15:52.450973	14:13:45.893615	14:15:52.447294
	3	14:18:46.656755	14:21:46.448220	14:18:46.652555	14:21:46.445174
2b)	1	14:25:54.446686	14:26:55.645693	14:25:54.342154	14:26:55.643948
	2	14:30:57.733593	14:33:23.532993	14:30:57.768592	14:33:23.527994
	3	14:35:55.503482	14:38:56.502798	14:35:55.499274	14:39:00.098576
3a)	1	14:44:51.962728	14:45:55.607036	14:44:51.772394	14:45:55.512194
	2	14:49:55.566734	14:51:49.181563	14:49:55.372060	14:51:49.111575
	3	14:54:51.329332	14:57:27.720389	14:54:51.131489	14:57:27.709099

Table 8 - Dataset 1 LSTM Detection Time Accuracy and Precision Results

Scenario	Average Start Time Difference (seconds)	Average End Time Difference (seconds)
1a)	0.0060 ±0.0058	0.0060 ±0.0031
1b)	0.0033 ±0.0011	0.0060 ±0.0075
2a)	0.0045 ±0.0004	0.0039 ±0.0007
2b)	0.04791 ±0.0378	1.1964 ±1.5465
3a)	0.1943 ±0.1322	0.0587 ±0.0391
Average	**0.0512 ±0.0354**	**0.2476 ±0.3194**

4.2.3 Comparison between Detectors

The LSTM autoencoder anomaly detector outperformed the MAIDENS detector in every scenario. As shown in Table 9 there was an 1054.69% increase in average detection start time and a 4147.82% increase in average detection end time when comparing detection time in seconds. In terms of messages the MAIDENS detector can detect the start time within 640 ±400 messages and the stop time within 10270 ±7980 messages, whereas the LSTM autoencoder detector can detect the start time within 51 ±35 messages and the stop time within 250 ± 320 messages. Based on the median overall average, the LSTM autoencoder anomaly detector is 1058.82% more effective at detecting the start time and 4108% more effective at detecting the end time of an anomalous event in terms of messages than the MAIDENS detector.

Table 9 – Dataset 1 Average Detection Time Start and End in Seconds

	Average Start and End Detection Time (Seconds)					
	Start Low	Middle	Start High	End Low	Middle	End High
LSTM	0.0158	0.0512	0.0866	-0.0718	0.2476	0.567
MAIDENS	0.14	0.54	0.94	2.29	10.27	18.25
% Increase	886.076	1054.69	1085.45	3189.42	4147.82	3218.70

Table 10 – Dataset 1 Average Detection Time Start and End in Terms of Messages

	Average Start and End Detection Time (#Messages)					
	Start Low	Middle	Start High	End Low	Middle	End High
LSTM	16	51	86	-70	250	570
MAIDENS	140	540	940	2290	10270	18250
% Increase	875	1058.82	1093.02	3271.43	4108	3201.75

4.3 Dataset 2

Dataset 2 was collected from a proprietary MIL-STD-1553B data bus test bench provided by RMC. The test bench is composed of a combination of simulated and emulated RTs. However, since the BC master schedule and exact makeup of the MIL-STD-1553B network is proprietary in nature, it will not be fully disclosed and instead the general architecture and setup for data collection is shown in Figure 24.

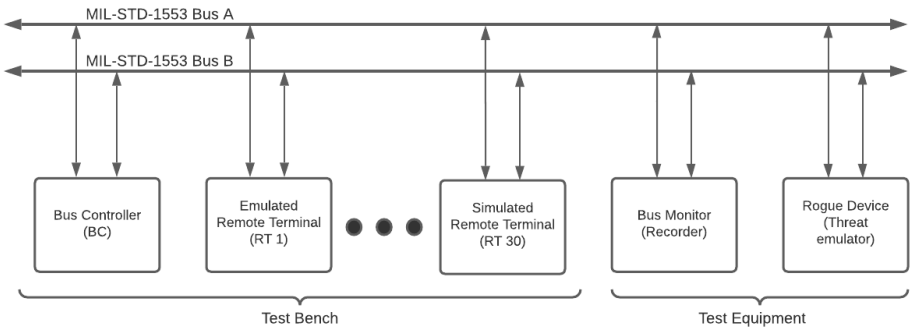


Figure 24 - Emulated MIL-STD-1553B Test Bench and Collection Architecture

For this dataset a total of five BMDX files were collected. The first file is a baseline recording that was collected from the bus during a normal operating state, representing the aircraft in straight and level flight. The remaining four BMDX files were the recordings of four separate threat scenarios. The four scenarios are executed over the physical data bus and recorded using an Abaco BusTools recorder, named

according to attack type and date recorded, and are composed of the attack categories outlined in Chapter 3:

1) DoS
    a) NetDisrupt statusword 250820 (disrupt): Network disruption (Status word DoS)
2) Targeted RT DoS
    a) RT-SA deny statusword rt18 sa32 250820 (deny): RT deny
    b) RT-SA deny statusword rt18 sa1 250820 (SA deny): RT subaddress deny
3) Data Integrity Violation
    a) Hijack rt18 sa6 w56 250820 (hijack): RT hijack

Each of the four recordings contain three occurrences of a single attack type, with each occurrence starting every 5 minutes and lasting for approximately one minute. The anomalies were created using a tool created by Paquet [6]. The start and end timing of each anomaly occurrence were recorded and examined post recording to properly tag the message timings. These scenarios were conducted throughout a re-enactment of the baseline scenario representing straight and level flight. Scenario 1a) in this dataset is equivalent to the scenario 1b) described in section 3.1. The DoS using command words could not be achieved due to this specific attack type crashing the bus monitor consistently during testing. The recordings will be referred to in the rest of this document according to their corresponding scenario number e.g. “NetDisrupt statusword 250820 (disrupt)” is referred to as Scenario 1a).

4.3.1 MAIDENS Detector

Dataset 2 was run through the MAIDENS detector and no results were able to be obtained. It was determined that because of how the MIL-STD-1553B protocol was implemented on this test bench it could not be handled by the MAIDENS detector without significant changes to the source material. The inability of the MAIDENS

detector to handle this dataset is presumed to be because of how this particular bus architecture implements bus load optimization. The bus controller balances the bus utilization evenly between the main bus (bus A) and the redundant bus (bus B). This differs from dataset 1, where the main bus (bus A) was utilized for 99% of bus communications. The modification of the MAIDENS detector is out of scope for this work, but with proper modification it would most likely be able to ingest this data. This did however, showcase the importance of considering how the MIL-STD-1553B protocol could be implemented when designing an anomaly detection method.

4.3.2 LSTM Autoencoder Detector

The LSTM autoencoder detector was created using the methodology and design described in Chapter 3 and uses the same process described in section 4.2.3 with the addition of data acquisition from the test bench.

4.3.2.1 LSTM Autoencoder Data Collection Pipeline

For the collection of the recording of each scenario including the baseline an Abaco BusTools recorder was connected as well as the tool created by Paquet [6]. Once collected, the Abaco BusTools software was used to output the BMDX files in CSV format in order to use the feature extraction tool presented in Appendix D. The product from this pipeline is five CSV files, one for each of the four attack scenario recordings and one for the baseline recording.

4.3.2.2 LSTM Autoencoder Model Creation

Using the same model creation process in section 4.2, the feature extracted baseline CSV file from the data collection pipeline was then usable by the Pandas and TensorFlow libraries so that a DL model can be created to represent the dataset. Using the model design described in section 4.1, the model was trained on the baseline dataset that was collected. The baseline data file contains 995,633 messages

and is split into a training and test set as outlined in Chapter 3. The training set contains 80% of the data while the test set contains the remainder. Using the training set, a LSTM autoencoder model was created, while the test set was used to confirm the output of the model and used to create the threshold value. Initially, a standard deviation of 2 was used, the same as in dataset 1, but produced an unacceptable false positive rate. Due to the increase in variation of data compared to dataset 1, increasing the threshold for dataset 2 to 2.5 standard deviations better suited the dataset and produced acceptable results for all four scenarios. Using a standard deviation of 2.5 the threshold was determined to be 1.0351. After having run the complete baseline dataset through the created model, none of the baseline model messages fell above the threshold, shown in Figure 25. This indicated that the model is a good representation of "normal" traffic as discussed in Chapter 3 and any future traffic that is run against this model that rises above the threshold value may be considered anomalous.

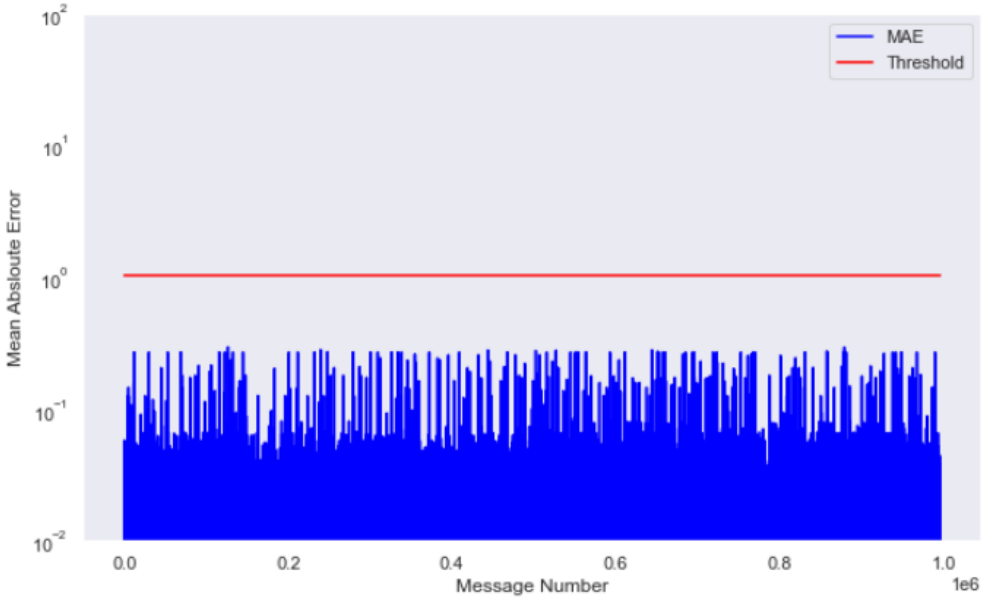


Figure 25 - Dataset 2 Baseline Anomaly Detection Results.

4.3.2.3 LSTM Autoencoder Anomaly Detection

In the previous section the baseline LSTM autoencoder model was created and a threshold value determined. The 4 scenario recordings that underwent feature extraction as described in section 4.3.2.1 were then ingested by the baseline model created in the previous section. Any of the MAE values that fell above the threshold value of 1.0351 will be considered an anomalous message. As with this discussed in section 4.2.2.2, any group of anomalous messages above the threshold will be considered a single event if no other messages go above the threshold in a 30 second window. Appendix C showcases the output of the LSTM autoencoder anomaly detector for each of the 4 scenarios, the results being discussed in the next section.

4.3.2.4 LSTM Autoencoder Results

The threshold value was set in section 4.3.2.2, to a MAE value of 1.0351. This threshold value led to all attacks being detected and no false positives being produced for all 4 scenarios, showcased in Appendix C. Table 11 showcases the detection results for each scenario and each threat occurrence within. When determining only whether an attack has occurred, the event classification accuracy, precision and recall were all calculated to be 100% indicating the LSTM Autoencoder detector had no issues detecting every attack occurrence (section 4.1.3). Table 12 however, shows the overall $Accuracy_{DT}$ and $Precision_{DT}$ results for each scenario. The LSTM autoencoder detector, detected the start and stop time of the first scenario within 0.0097 ±0.0120 and 0.0482 ±0.0582 seconds respectively. This means that the detector was within 20 and 120 messages of determining the start and stop time of the threat scenarios. The overall average start and stop time for all combined scenarios is 0.0071 ±0.0087 and 0.0698 ±0.0409 seconds for determining an anomalous event or within 15 and 100 messages.

Table 11 - Dataset 2 LSTM Autoencoder Detection Results

Scenario	Threat Occurrence	Actual Time of Anomaly (minutes:seconds)		Identified Time of Anomaly (minutes:seconds)	
		Start	End	Start	End
1a)	1	13:31:21.410110	13:32:20.276438	13:31:21.382417	13:32:20.274504
	2	13:36:21.585307	13:37:19.811250	13:36:21.585307	13:37:19.675735
	3	13:41:21.666338	13:42:20.350172	13:41:21.667805	13:42:20.342905
2a)	1	13:09:25.728274	13:10:25.680717	13:09:25.727660	13:10:25.680379
	2	13:14:25.767550	13:15:25.531311	13:14:25.767814	13:15:25.530763
	3	13:19:25.541643	13:20:25.878662	13:19:25.543358	13:20:25.877716
2b)	1	12:23:29.388951	12:25:28.701594	12:23:29.388951	12:25:28.701269
	2	12:28:29.137439	12:29:29.089815	12:28:29.188035	12:29:29.088796
	3	12:33:29.540529	12:34:29.741777	12:33:29.542288	12:34:29.740567
3a)	1	14:37:32.380410	14:38:27.244591	14:37:32.380119	14:38:27.177577
	2	14:42:32.412870	14:43:27.898795	14:42:32.412957	14:43:27.525835
	3	14:47:32.210372	14:48:27.393951	14:47:32.210796	14:48:27.161069

Table 12 - Dataset 2 LSTM Autoencoder Detection Time Accuracy and Precision Results

Scenario	Average Start Time Difference (seconds)	Average End Time Difference (seconds)
1a)	0.0097 ±0.0120	0.0482 ±0.0582
2a)	0.0009 ±0.0006	0.0006 ±0.0002
2b)	0.0175 ±0.0221	0.0009 ±0.0004
3a)	0.0003 ±0.0001	0.2242 ±0.1048
Average	**0.0071 ±0.0087**	**0.0685 ±0.0409**

4.4 Discussion

The LSTM Autoencoder proved a successful means for detecting anomalies on two different MIL-STD-1553B Bus architectures. For dataset 1, the LSTM Autoencoder outperformed the MAIDENS detector in every scenario and for dataset 2 the detector indicated the start and stop time of each event with a hundredth or thousandth of a second, detecting each the start and stop time of each scenario within a few messages of the threat occurrence.

For dataset 1, the LSTM autoencoder detector outperformed the MAIDENS detector in each of the 5 scenarios. Table 9 showcases the improved start and end time detections in seconds, while Table 10 showcases the improved start and end time detections in terms of number of messages. The LSTM autoencoder detector had an average detection start time of 0.0512 ±0.0354 seconds and an average detection stop time of 0.2476 ±0.3194 seconds over the 5 scenarios. This equates to the LSTM autoencoder being able to detect the start time of an anomalous event within approximately 51 ±35 messages and the stop time within 250 ±320 messages. Compared to the statistical-based method, MAIDENS [1], which had an average detection start time of 0.54 ±0.40 seconds or within 640 ±400 messages and an

average detection stop time of 10.27 ±7.98 seconds or within 10270 ±7980 messages.

For dataset 2, the MAIDENS detector was not able to process the recordings. This was likely due to how the bus architecture was implemented for dataset 2 and would require a significant rework of the MADIENS detector, which was out of scope for this work. However, the LSTM autoencoder detector was able to process the recording the same as in dataset 1 and the results are presented in Table 13 and Table 14. The LSTM autoencoder detector had an average start detection time of 0.0071 ±0.0087 seconds or within approximately 7 ±8 messages and an average stop time detection of 0.0685 ±0.0409 seconds or within approximately 68 ±41 messages. The results demonstrating this DL technique as an effective method for identifying attacks on a MIL-STD-1553B Data Bus.

Table 13 – Dataset 2 Average Detection Time Start and End in Seconds

	Average Start and End Detection Time (Seconds)					
	Start Low	Middle	Start High	End Low	Middle	End High
LSTM	-0.0016	0.0071	0.0158	0.0276	0.0685	0.1094

Table 14 – Dataset 2 Average Detection Time Start and End in Terms of Messages

	Average Start and End Detection Time (#Messages)					
	Start Low	Middle	Start High	End Low	Middle	End High
LSTM	-1	7	15	27	68	109

The significance of increasing the $Accuracy_{DT}$ of detection means reducing the number of messages needed to be sorted through to find the actual start and stop time of an event, if they are not already known. As an example, if a forensic investigator were trying to determine the start time of Scenario 1a) from dataset 1. Using the detection time from MAIDENS the operator would need to sort through approximately 540 to 1000 messages, before finding the actual start time. Using the

detection time of the LSTM autoencoder anomaly detector the operator would need to sort through approximately 50 to 90 messages. This is a significant decrease and would drastically reduce the amount of time needed to narrow down an anomalous event.

There were a couple of drawbacks that were noted during experimentation. The first is the manual setting of a threshold value. For both datasets an acceptable threshold value was able to be set for all anomaly detection. However, depending on how the MIL-STD-1553B protocol was implemented or what state of flight an aircraft is in during recording, this value will need to be adjusted accordingly. The general observation was that the busier and more complex the communication schedule is, the higher the threshold values needed to be set. The setting of the threshold value may require expert knowledge on the system and is a significant factor to ensure the detector is detecting only anomalous events. The second drawback noted related to data collection and how the threat emulation tool from Paquet [6] did not have the ability to tag individual messages as anomalous. The LSTM autoencoder by design is able to tag each message as anomalous or not. Overall, the increase in performance when compared to MADIENS is worth the few drawbacks, with each drawback that could be studied and expanded on further in the future to potentially minimize their effect.

The above results showcase the viability of DL methods as a means of detecting of anomaly on a MIL-STD-1553B Bus. An LSTM autoencoder is just one example of many available DL methods that can be explored for anomaly detection on network architectures like the MIL-STD-1553B protocol. Advances in means of detection of anomalies can not only inform users of intrusions on their networks, but can also be a means of fault detection for maintenance actions.

4.5 Summary

Overall, for dataset 1 when comparing the LSTM Autoencoder detector, it outperformed the MAIDENS detector for all five scenarios in both the $Accuracy_{DT}$ and $Precision_{DT}$ when indicating start and stop time of threat occurrences. For dataset 2 the LSTM Autoencoder was also able to detect the start and stop times of each anomalous scenario within a tenth to a hundredth of a second. Unfortunately, the MAIDENS detector was unable to process the data from dataset 2 resulting in the fact that no comparisons could be made between the two detectors. The results showcase the applicability of using the LSTM autoencoder DL method for anomaly detection on MIL-STD-1553B data bus.

Chapter 5

Conclusion

This research investigated the improvement of detection time accuracy and precision of threat scenarios on the MIL-STD-1553B data bus. This was accomplished by the implementation of an LSTM autoencoder DL detector to improve the overall effectiveness of anomaly detection times when compared to a statistical-based detection method. This chapter presents an overview of the motivations for this research, contributions of this research in the field of MIL-STD-153B anomaly detection, recommendations for future work based on this research, and finally recommendations based on the finding of this research.

5.1 Overview

While the detection of an actual cyber attack on a physical aircraft has yet to be realized, the consequences of such an event cannot be ignored. The difficulty of MIL-STD-1553B networks is that not every data bus network is implemented the same and it is too costly to replace and implement a different protocol. There are methods that have been developed for the MIL-STD-1553B protocol and include signature-based, and anomaly-based detection techniques [1], [2], [3], [4].

The aim of this research was to improve the time-related performance metrics when detecting attacks on the MIL-STD-1553B data bus traffic using a LSTM autoencoder deep learning technique. Not only was it successful in detecting different attacks on two different bus architectures, but it also outperformed the MAIDENS detector presented in [1] in both detection time accuracy ($Accuracy_{DT}$) and detection time precision ($Precision_{DT}$). The detector can handle different types of anomalous scenarios, detecting the start and stop timings of each anomalous

scenario to a hundredth or thousandth of a second with no false positive or negative results. The degree of detection accuracy would be more beneficial, compared to MAIDENS, in a forensics investigation as the detected time are closer to the actual attack time, resulting in far fewer frames to needlessly analyse.

There are some drawbacks to the LSTM autoencoder and the specific implementation throughout this work. The setting of a manual threshold value is variable based on the data bus under examination. This can require the use of expert knowledge in order to set an appropriate value for proper anomaly detection. Another drawback, is the setting of hyper-parameters and selection of features. The feature set proposed was used in its entirety, however some features may not be relevant or hinder the models training. This goes for the selection of hyper-parameters as well, there could very well be better parameters that optimize the model's performance. Lastly, the datasets used both represent an aircraft during straight and level flight and use simulated and emulated RTs. Using a real MIL-STD-1553B data bus would be beneficial to confirm the results of this study as well as traffic during different phases of flight.

This work provides substantial evidence of the applicability of the LSTM autoencoder detector for anomaly detection on MIL-STD-1553B networks. There are some deficiencies that could be improved upon to confirm viability on real MIL-STD-1553B data busses and optimize the detector. However, this work satisfies the overall aim of improving the detection capabilities when compared to a statistical-based detection method.

5.2 Contributions

The following contributions were made by this research:

1. An LSTM autoencoder detector that has an improved detection time accuracy and detection time precision than the MAIDENS detector.

2. The collection of benign and anomalous MIL-STD-1553B traffic datasets.
3. A tool for conducting feature extraction on MIL-STD-1553B traffic collected in BMDX format for input into ML or DL models.
4. A list of MIL-STD-1553B features for ML or DL modeling.

5.3 Future Work

This research investigated the application of a LSTM autoencoder detector in order to increase in efficiency in detecting anomalies on the MIL-STD-1553B data bus. This work also concludes that there are areas that could be improved upon and recommends the following to be investigated to further advance this work:

1. Development of a feature selection and optimization process.
2. Development of a model training and hyper-parameter selection and optimization process.
3. Development of a MIL-STD-1553B data bus dataset that would allow for detection of anomalies in different phases of operation (flight).

5.4 Recommendation

This research was conducted in a test environment with simulated and emulated devices. It is recommended that this anomaly detection method be evaluated on real MIL-STD-1553B traffic. Once tested and evaluated it could be used as a means of developing an effective IDS for implementation on the tested MIL-STD-1553B data bus network. This implementation could provide the ability to monitor traffic on MIL-STD-1553B traffic where nothing currently exists.

References

[1] S. J. J. Généreux, A. K. H. Lai, C. O. Fowles, V. R. Roberge, G. P. M. Vigeant, and J. R. Paquet, "MAIDENS: MIL-STD-1553 Anomaly-Based Intrusion Detection System Using Time-Based Histogram Comparison," 2019.

[2] O. Stan, Y. Elovici, A. Shabtai, G. Shugol, R. Tikochinski, and S. Kur, "Protecting Military Avionics Platforms from Attacks on MIL-STD-1553 Communication Bus," *ArXiv170705032 Cs*, Jul. 2017, Accessed: Jan. 28, 2019. [Online]. Available: http://arxiv.org/abs/1707.05032

[3] O. Stan, A. Cohen, Y. Elovici, and A. Shabtai, "Intrusion Detection System for the MIL-STD-1553 Communication Bus," *IEEE Trans. Aerosp. Electron. Syst.*, pp. 1–1, 2019, doi: 10.1109/TAES.2019.2961824.

[4] C. Bernard, "An application of network security monitoring to the MIL-STD-1553B data bus," Masters thesis, Royal Military College of Canada, Kingston, ON, 2019.

[5] "ISO 5725-1:1994(en), Accuracy (trueness and precision) of measurement methods and results — Part 1: General principles and definitions." https://www.iso.org/obp/ui/#iso:std:iso:5725:-1:ed-1:v1:en (accessed Sep. 20, 2022).

[6] J. Paquet, "Uncovering MIL-STD-1553 vulnerabilities: exploitability of military aircraft networks," Masters thesis, Royal Military College of Canada, 2014.

[7] "MIL-STD-1553 Designer's Guide." ILC Data Device Corporation, 1998.

[8] K. Lounis, Z. Mansour, M. Wrana, M. A. Elsayed, S. H. H. Ding, and M. Zulkernine, "A Review and Analysis of Attack Vectors on MIL-STD-1553 Communication Bus," *IEEE Trans. Aerosp. Electron. Syst.*, pp. 1–1, 2022, doi: 10.1109/TAES.2022.3177583.

[9] R. A. Kemmerer, "Shared Resource Matrix Methodology: An Approach to Identifying Storage and Timing Channels," *ACM Trans. Comput. Syst.*, vol. 1, no. 3, p. 22.

[10] T. D. Nguyen, "Towards MIL-STD-1553B Covert Channel Analysis:," Defense Technical Information Center, Fort Belvoir, VA, Jan. 2015. doi: 10.21236/ADA613900.

[11] S. Mukkamala, A. Sung, and A. Abraham, "Cyber Security Challenges: Designing Efficient Intrusion Detection Systems and Antivirus Tools," in *Enhancing Computer Security with Smart Technology*, illustrated ed., V. Rao Vemuri, Ed. CRC Press, 2005.

[12] S. Agrawal and J. Agrawal, "Survey on Anomaly Detection using Data Mining Techniques," *Procedia Comput. Sci.*, vol. 60, pp. 708–713, 2015, doi: 10.1016/j.procs.2015.08.220.

[13] C. Chio and D. Freeman, *Machine Learning & Security: Protecting Systems with Data and Algorithms*. O'Reilly, 2018.

[14] D. Heckerman, "A Tutorial on Learning with Bayesian Networks," in *Innovations in Bayesian Networks. Studies in Computational Intelligence*, vol. 156, D. E. Holmes and L. C. Jain, Eds. Springer Berlin Heidelberg. [Online]. Available: https://doi.org/10.1007/978-3-540-85066-3_3

[15] L. A. Maglaras and J. Jiang, "Intrusion detection in SCADA systems using machine learning techniques," in *2014 Science and Information Conference*, Aug. 2014, pp. 626–631. doi: 10.1109/SAI.2014.6918252.

[16] A. K. Jain, M. N. Murty, and P. J. Flynn, “Data clustering: a review,” *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, Sep. 1999, doi: 10.1145/331499.331504.
[17] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Netw.*, vol. 2, no. 5, pp. 359–366, Jan. 1989, doi: 10.1016/0893-6080(89)90020-8.
[18] L. Deng and D. Yu, “Deep Learning: Methods and Applications,” *Found. Trends® Signal Process.*, vol. 7, no. 3–4, pp. 197–387, 2014.
[19] J. Kim, J. Kim, H. L. T. Thu, and H. Kim, “Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection,” in *2016 International Conference on Platform Technology and Service (PlatCon)*, Feb. 2016, pp. 1–5. doi: 10.1109/PlatCon.2016.7456805.
[20] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[21] A. Taylor, S. Leblanc, and N. Japkowicz, “Anomaly Detection in Automobile Control Network Data with Long Short-Term Memory Networks,” in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, Oct. 2016, pp. 130–139. doi: 10.1109/DSAA.2016.20.
[22] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, vol. 385. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-24797-2.
[23] M. J. Potvin, “Detecting malicious anomalies in heavy duty vehicular networks,” Masters thesis, Royal Military College of Canada, Kingston, ON, 2020.
[24] M. Claesen and B. De Moor, “Hyperparameter Search in Machine Learning.” arXiv, Apr. 06, 2015. Accessed: Sep. 25, 2022. [Online]. Available: http://arxiv.org/abs/1502.02127
[25] G. E. Hinton, “A Practical Guide to Training Restricted Boltzmann Machines,” in *Neural Networks: Tricks of the Trade*, vol. 7700, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 599–619. doi: 10.1007/978-3-642-35289-8_32.
[26] F. Pedregosa *et al.*, “Scikit-learn: Machine learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
[27] J. Bergstra and Y. Bengio, “Random Search for Hyper-Parameter Optimization,” *J. Mach. Learn. Res.*, vol. 13, no. 10, pp. 281–305, 2012.
[28] F. Gaillard, “Epoch (machine learning) | Radiology Reference Article | Radiopaedia.org,” *Radiopaedia*. https://radiopaedia.org/articles/epoch-machine-learning?lang=us (accessed Sep. 25, 2022).
[29] “tf.keras.callbacks.EarlyStopping | TensorFlow v2.10.0,” *TensorFlow*. https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping (accessed Sep. 25, 2022).
[30] O. I. Provotar, Y. M. Linder, and M. M. Veres, “Unsupervised Anomaly Detection in Time Series Using LSTM-Based Autoencoders,” in *2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT)*, Dec. 2019, pp. 513–517. doi: 10.1109/ATIT49449.2019.9030505.
[31] J. Venskus, P. Treigys, and J. Markevičiūtė, “Unsupervised marine vessel trajectory prediction using LSTM network and wild bootstrapping techniques,” *Nonlinear Anal. Model. Control*, vol. 26, no. 4, pp. 718–737, Jul. 2021, doi: 10.15388/namc.2021.26.23056.

[32] A. Almalaq and J. J. Zhang, "Evolutionary Deep Learning-Based Energy Consumption Prediction for Buildings," *IEEE Access*, vol. 7, pp. 1520–1531, 2019, doi: 10.1109/ACCESS.2018.2887023.
[33] T. Hasith Ram Varma, D. Dhwani, and A. M. Méan, "A Review of various statestical methods for Outlier Detection," *Int. J. Comput. Sci. Eng. Technol. IJCSET*, 2014.
[34] S. Maleki, S. Maleki, and N. R. Jennings, "Unsupervised anomaly detection with LSTM autoencoders using statistical data-filtering," *Appl. Soft Comput.*, vol. 108, p. 107443, Sep. 2021, doi: 10.1016/j.asoc.2021.107443.
[35] M. Said Elsayed, N.-A. Le-Khac, S. Dev, and A. D. Jurcut, "Network Anomaly Detection Using LSTM Based Autoencoder," in *Proceedings of the 16th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, Alicante Spain, Nov. 2020, pp. 37–45. doi: 10.1145/3416013.3426457.
[36] R. Kabore, A. Kouassi, R. N'goran, O. Asseu, Y. Kermarrec, and P. Lenca, "Review of Anomaly Detection Systems in Industrial Control Systems Using Deep Feature Learning Approach," *Engineering*, vol. 13, no. 01, pp. 30–44, 2021, doi: 10.4236/eng.2021.131003.
[37] B. F. Losier, R. Smith, and V. R. Roberge, "Design of a Time-Based Intrusion Detection Algorithm for the MIL-STD-1553." http://roberge.segfaults.net/joomla/index.php/publications (accessed Mar. 06, 2019).
[38] K. P. Shung, "Accuracy, Precision, Recall or F1?," *Medium*, Apr. 10, 2020. https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9 (accessed Sep. 25, 2022).
[39] L. S. (Abaco S. Anderson and N.-G. Us, "AN 009 The BMD and BMDX File Formats," p. 11.
[40] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," p. 21.
[41] "pandas - Python Data Analysis Library." https://pandas.pydata.org/ (accessed Sep. 27, 2022).
[42] Y. Verma, "A Complete Understanding of Dense Layers in Neural Networks," *Analytics India Magazine*, Sep. 19, 2021. https://analyticsindiamag.com/a-complete-understanding-of-dense-layers-in-neural-networks/ (accessed Sep. 27, 2022).
[43] C. Ranjan, "Step-by-step Understanding LSTM Autoencoder Layers," *Data Sci.*, Apr. 2019, [Online]. Available: https://towardsdatascience.com/step-by-step-understanding-lstm-autoencoder-layers-ffab055b6352

Appendix A - MIL-STD-1553B Feature Set

This Appendix contains 3 tables: Table A-1, that describes the command and status word features derived from the MIL-STD-1553 protocol. Table A-2, that describes the message word features derived from the MIL-STD-1553 protocol. Lastly, Table A-3 that is a compilation of the features in Tables A-1 and A-2.

Table A-1 – MIL-STD-1553B Command and Status Word Features

	Feature Name	Abbreviation	Description
BC-RT or RT-RT Message Features	Command 1 Address	CMD1-addr	The address of the RT transmitting or receiving data for BC-RT message. For RT-RT messages it is the address of the transmitting RT.
	Command 1 Transmission or Receive mode	CMD1-TR	Indicates whether CMD 1 RT is sending or receiving.
	Command 1 sub address	CMD1-subaddr	The sub-address of CMD 1 RT.
	Command 1 number of data words	CMD1-numword	Number of words to transmit or receive.
	Status word response time to command 1	RSP1	The amount of time in micro seconds (us) for CMD 1 RTs status word to respond to BC.
	Status word 1 Address	STS1-addr	Address of RT status word response to CMD 1.
	Status word 1 Error Bit	STS1-Error	Indicates if the Error bit in STS1 is set.

	Feature Name	Abbreviation	Description
	Status word 1 Instrumentation Bit	STS1-Inst	Indicates if the Instrumentation bit in STS1 is set.
	Status word 1 Service Request Bit	STS1-SerReq	Indicates if the Service Request bit in STS1 is set.
	Status word 1 Reserve Bit	STS1-Reserved	Indicates if the Reserved bit in STS1 is set.
	Status word 1 Broadcast Cmd Received Bit	STS1-BCRecv	Indicates if the Broadcast Cmd Received bit in STS1 is set.
	Status word 1 Busy Bit	STS1-Busy	Indicates if the Busy bit in STS1 is set.
	Status word 1 Subsystem Flag Bit	STS1-SubFlag	Indicates if the Subsystem Flag bit in is STS1 set.
	Status word 1 Dynamic Bus Acceptance Bit	STS1-DBAcc	Indicates if the Dynamic Bus Acceptance bit in STS1 is set.
	Status word 1 Terminal Flag Bit	STS1-TerFlag	Indicates if the Terminal Flag bit in STS1 is set.
RT-to-RT Message Features (Zeroized if not present)	Command 2 Address	CMD2-addr	The address of the RT transmitting or receiving data for BC-RT message. For RT-RT messages it is the address of the transmitting RT.
	Command 2 Transmission or Receive mode	CMD2-TR	Indicates whether CMD 2 RT is sending or receiving.
	Command 2 sub address	CMD2-subaddr	The sub-address of CMD 2 RT.
	Command 2 number of data words	CMD2-numword	Number of words to transmit or receive.
	Status word response time to command 2	RSP2	The amount of time in micro seconds (us) for CMD 2 RTs status word to respond to BC.

	Feature Name	Abbreviation	Description
	Status word 2 Address	STS2-addr	Address of RT status word response to CMD 2.
	Status word 2 Error Bit	STS2-Error	Indicates if the Error bit in STS2 is set.
	Status word 2 Instrumentation Bit	STS2-Inst	Indicates if the Instrumentation bit in STS2 is set.
	Status word 2 Service Request Bit	STS2-SerReq	Indicates if the Service Request bit in STS2 is set.
	Status word 2 Reserve Bit	STS2-Reserved	Indicates if the Reserved bit in STS2 is set.
	Status word 2 Broadcast Cmd Received Bit	STS2-BCRecv	Indicates if the Broadcast Cmd Received bit in STS2 is set.
	Status word 2 Busy Bit	STS2-Busy	Indicates if the Busy bit in STS2 is set.
	Status word 2 Subsystem Flag Bit	STS2-SubFlag	Indicates if the Subsystem Flag bit in is STS2 set.
	Status word 2 Dynamic Bus Acceptance Bit	STS2-DBAcc	Indicates if the Dynamic Bus Acceptance bit in STS2 is set.
	Status word 2 Terminal Flag Bit	STS2-TerFlag	Indicates if the Terminal Flag bit in STS2 is set.

Table A-2 – MIL-STD-1553B Message Status Features

Feature Name	Abbreviation	Description
Bad RT Address	BT1553_INT_BAD_RTADDR	This bit indicates that the RT address in the status word response is not identical to the RT addressed in the command word.
Bit Count	BT1553_INT_BIT_COUNT	This bit indicates that the bit count of one or more words in the message was not the expected value (16). This condition also sets the BT1553_INT_INVALID_WORD bit.
BM Overflow	BT1553_INT_BM_OVERFLOW	This bit indicates that the BM overflowed at this point and some messages were lost. The number lost can be determined by examining the message numbers.
Broadcast	BT1553_INT_BROADCAST	This bit indicates that the message was a broadcast message. This is set in BC, BM and RT messages.
Channel	BT1553_INT_CHANNEL	This bit indicates the channel on which the message was detected. If this bit is set, the message was detected on Bus B; otherwise, the message was detected on Bus A.
Early Response	BT1553_INT_EARLY_RESP	This bit indicates that the 1553 decoder detected a command sync with less than 2μs of bus dead time when a status word is expected.
End of Message	BT1553_INT_END_OF_MESS	This bit indicates that the parity bit of the last word of the message has been transmitted or received on the 1553 bus. This bit is set on every legal message with a valid command word, regardless of any other detected error conditions.
High Word	BT1553_INT_HIGH_WORD	This bit indicates that the message contained more data words than was defined in the word count field of the command word.
Invalid Word	BT1553_INT_INVALID_WORD	This bit indicates that any of the following was detected on one or more words of the message: inverted sync, invalid Manchester II encoding (including zero and crossing errors), bit count error, or parity error.
Inverted Sync	BT1553_INT_INVERTED_SYNC	This bit indicates that the sync field was inverted from what was expected by the message transfer format on one or more of the words in the message. This condition also sets the BT1553_INT_INVALID_WORD bit.
Late Response	BT1553_INT_LATE_RESP	This bit indicates that the 1553 decoder did not detect a command sync within the specified Late Response Timeout when a status word is expected. This time is set with the “wTimeout2” parameter of the BusTools_BC_Init function.
Low Word Flag	BT1553_INT_LOW_WORD	This bit indicates that the message contained fewer data words than were indicated in the word count field of the command word.
Message Error	BT1553_INT_ME_BIT	This bit indicates that the 1553 Status Word response had the Message Error bit set.

Feature Name	Abbreviation	Description
Mid Bit	BT1553_INT_MID_BIT	This bit indicates that successive mid-zero crossings were not within 150ns of the expected time for any successive bits in any word of the message (except the sync bit – see below). This condition also sets the BT1553_INT_INVALID_WORD bit.
Mode Code	BT1553_INT_MODE_CODE	This bit is set to indicate a mode code message.
No Command	BT1553_INT_NOCMD	This bit is set when the decoder does not see any message on the bus after a BC command is sent. This is probably the result of an improperly terminated bus. The BT1553_INT_NO_RESP bit is set as well.
NO Inter-Message Gap	BT1553_INT_NO_IMSG_GAP	This bit indicates that the mid-sync zero crossing of the next command sync was detected prior to 4μs preceding the mid-zero crossing of the parity bit of the last word of the current message. The next command sync may or may not produce a valid command word.
No Response	BT1553_INT_NO_RESP	This bit indicates that the 1553 decoder did not detect a command sync within the specified No Response Timeout time when a status word is expected. This time is set with the wTimeout1 parameter to the BusTools_BC_Init function.
Non-Continuous Data	BT1553_INT_NON_CONT_DATA	This bit indicates that a gap was detected between successive data words in the message. The hardware allows a 4-μs gap before declaring a Low Word error and beginning a search for the next command word.
Parity Error	BT1553_INT_PARITY	This bit indicates that a parity error was detected in one or more words of the message. Odd parity is used (per the MIL-STD-1553B Specification). This condition also sets the BT1553_INT_INVALID_WORD bit.
Reset RT	BT1553_INT_RESET_RT	This bit indicates that a valid Reset Terminal mode code command was received. The application must reset this RT to an initialized state.
Retry	BT1553_INT_RETRY	This bit indicates that an automatic retry was executed by the BC. This bit doesn't indicate whether the retry was successful. Failure of the retry results in the No Response error bit set or the Message Error bit in the RTs status word being set.
RT to RT Format	BT1553_INT_RT_RT_FORMAT	This bit indicates that the message is an RT to RT message. It is detected in hardware by two consecutive words with a command sync.
RT to RT not Responding	BT1553_RTRT_RCV_NRSP	This bit indicates which message on a RT to RT message is not responding. If BT1553_INT_NO_RESP is set for a RT to RT message check this bit. If set the receive command did not respond. If reset the transmit command did not respond.
Self Test	BT1553_INT_SELF_TEST	This bit indicates the reception of a Built-In-Test mode code command.

Feature Name	Abbreviation	Description
Trigger Begin	BT1553_INT_TRIG_BEGIN	This bit indicates that the BM trigger enable condition was met in this message. BM message gathering begins at this time. This bit is generated by the API interrupt service function.
Trigger End	BT1553_INT_TRIG_END	This bit indicates that the BM trigger disable condition was met in this message. BM message gathering terminates at this time. This bit is generated by the API interrupt service function.
Two Bus	BT1553_INT_TWO_BUS	This bit indicates that both buses (Bus A and Bus B) were active sometime during the message. This is a 1553 protocol error.
Wrong Bus	BT1553_INT_WRONG_BUS	This bit indicates that the RT responded on a different bus than the one on which the command word was transmitted. This is a 1553 protocol error.
Alternate Bus	BT1553_INT_ALT_BUS	This bit indicates that an automatic retry was executed by the BC on alternate bus. This bit doesn't indicate whether the retry was successful. Failure of the retry results in the No Response error bit set or the Message Error bit in the RTs status word being set.
Invalid RT to RT Transmission	BT1553_INV_RTRT_TX	Invalid RT to RT Transmission CMD2.

Table A-3 – MIL-STD-1553B Feature Set

	Feature Number	Feature Name	Abbreviation
BC-RT or RT-RT Message Features	1	Command 1 Address	CMD1-addr
	2	Command 1 Transmission or Receive mode	CMD1-TR
	3	Command 1 sub address	CMD1-subaddr
	4	Command 1 number of data words	CMD1-numword
	5	Status word response time to command 1	RSP1
	6	Status word 1 Address	STS1-addr
	7	Status word 1 Error Bit	STS1-Error

	Feature Number	Feature Name	Abbreviation
	8	Status word 1 Instrumentation Bit	STS1-Inst
	9	Status word 1 Service Request Bit	STS1-SerReq
	10	Status word 1 Reserve Bit	STS1-Reserved
	11	Status word 1 Broadcast Cmd Received Bit	STS1-BCRecv
	12	Status word 1 Busy Bit	STS1-Busy
	13	Status word 1 Subsystem Flag Bit	STS1-SubFlag
	14	Status word 1 Dynamic Bus Acceptance Bit	STS1-DBAcc
	15	Status word 1 Terminal Flag Bit	STS1-TerFlag
	16	Bad RT Address	BT1553_INT_BAD_RTADDR
	17	Bit Count	BT1553_INT_BIT_COUNT
	18	BM Overflow	BT1553_INT_BM_OVERFLOW
	19	Broadcast	BT1553_INT_BROADCAST
	20	Channel	BT1553_INT_CHANNEL
	21	Early Response	BT1553_INT_EARLY_RESP
	22	End of Message	BT1553_INT_END_OF_MESS
	23	High Word	BT1553_INT_HIGH_WORD
	24	Invalid Word	BT1553_INT_INVALID_WORD
	25	Inverted Sync	BT1553_INT_INVERTED_SYNC
	26	Late Response	BT1553_INT_LATE_RESP
	27	Low Word Flag	BT1553_INT_LOW_WORD
	28	Message Error	BT1553_INT_ME_BIT
	29	Mid Bit	BT1553_INT_MID_BIT
	30	Mode Code	BT1553_INT_MODE_CODE
	31	No Command	BT1553_INT_NOCMD
	32	No Inter-Message Gap	BT1553_INT_NO_IMSG_GAP

	Feature Number	Feature Name	Abbreviation
	33	No Response	BT1553_INT_NO_RESP
	34	Non-Continuous Data	BT1553_INT_NON_CONT_DATA
	35	Parity Error	BT1553_INT_PARITY
	36	Reset RT	BT1553_INT_RESET_RT
	37	Retry	BT1553_INT_RETRY
	38	RT to RT Format	BT1553_INT_RT_RT_FORMAT
	39	RT to RT not Responding	BT1553_RTRT_RCV_NRSP
	40	Self Test	BT1553_INT_SELF_TEST
	41	Trigger Begin	BT1553_INT_TRIG_BEGIN
	42	Trigger End	BT1553_INT_TRIG_END
	43	Two Bus	BT1553_INT_TWO_BUS
	44	Wrong Bus	BT1553_INT_WRONG_BUS
	45	Alternate Bus	BT1553_INT_ALT_BUS
	46	Invalid RT to RT Transmission	BT1553_INV_RTRT_TX
RT-RT Message Features (Zeroized if not present)	47	Command 2 Address	CMD2-addr
	48	Command 2 Transmission or Receive mode	CMD2-TR
	49	Command 2 sub address	CMD2-subaddr
	50	Command 2 number of data words	CMD2-numword
	51	Status word response time to command 2	RSP2
	52	Status word 2 Address	STS2-addr
	53	Status word 2 Error Bit	STS2-Error
	54	Status word 2 Instrumentation Bit	STS2-Inst
	55	Status word 2 Service Request Bit	STS2-SerReq
	56	Status word 2 Reserve Bit	STS2-Reserved

	Feature Number	Feature Name	Abbreviation
	57	Status word 2 Broadcast Cmd Received Bit	STS2-BCRecv
	58	Status word 2 Busy Bit	STS2-Busy
	59	Status word 2 Subsystem Flag Bit	STS2-SubFlag
	60	Status word 2 Dynamic Bus Acceptance Bit	STS2-DBAcc
	61	Status word 2 Terminal Flag Bit	STS2-TerFlag

Appendix B - Dataset 1 Model MAE vs MSG# Graphs

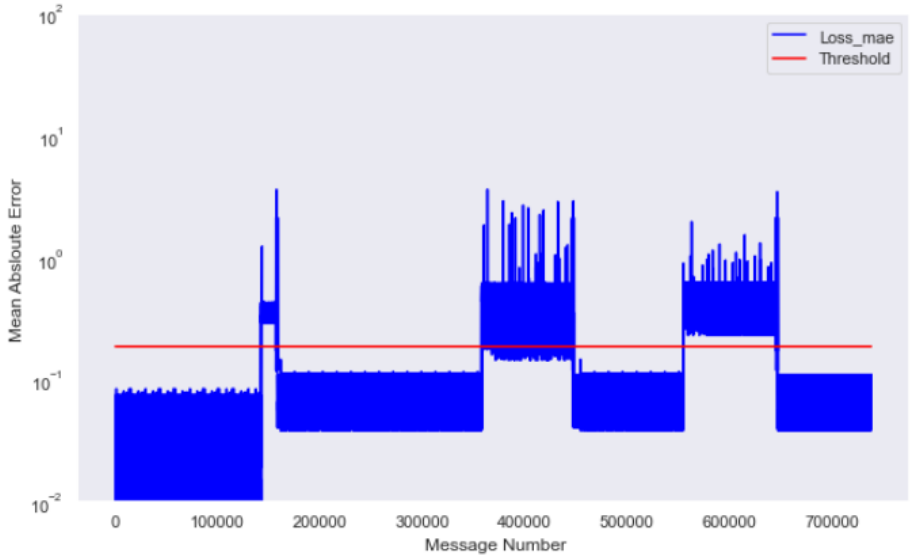


Figure B-1: Dataset 1 - MAE vs MSG # for Scenario 1a

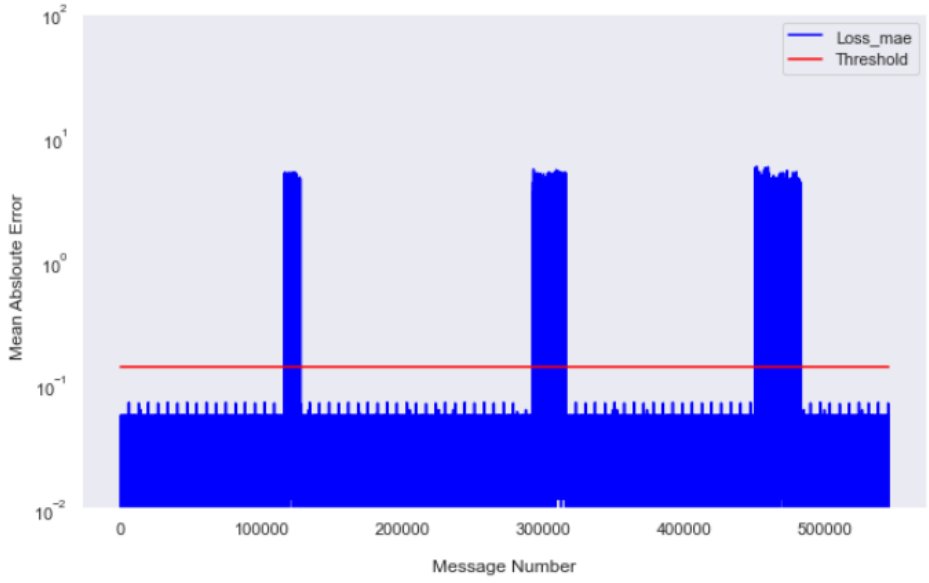


Figure B-2: Dataset 1 - MAE vs MSG # for Scenario 1b

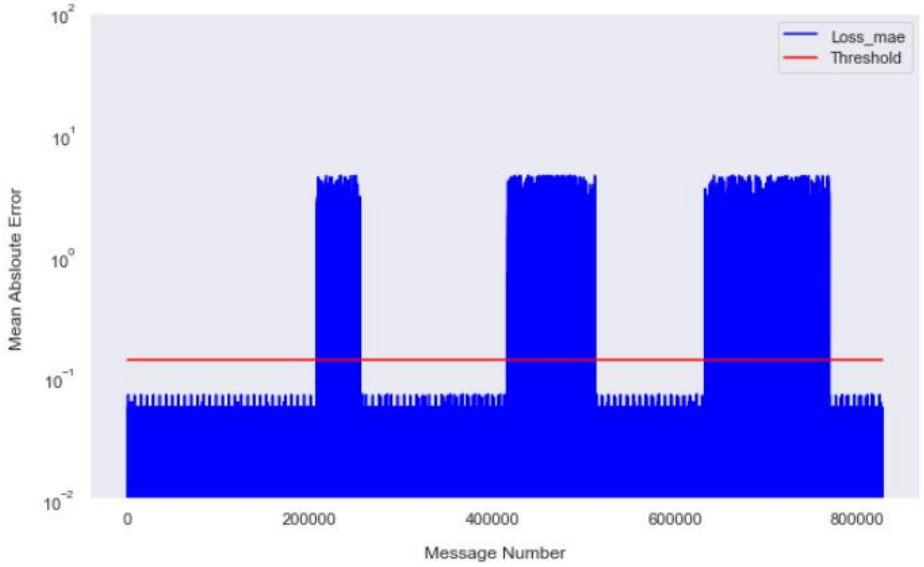


Figure B-3: Dataset 1 - MAE vs MSG # for Scenario 2a

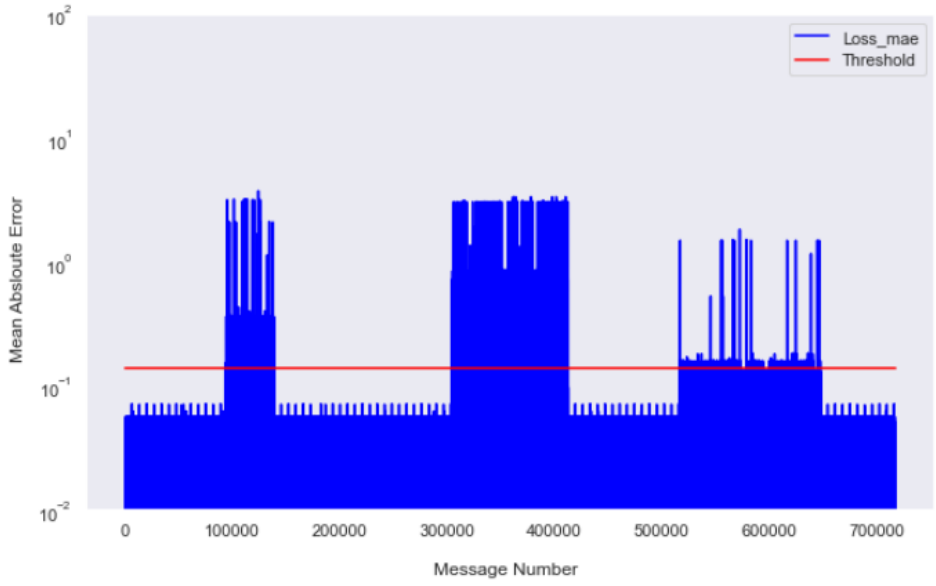


Figure B-4: Dataset 1 - MAE vs MSG # for Scenario 2b

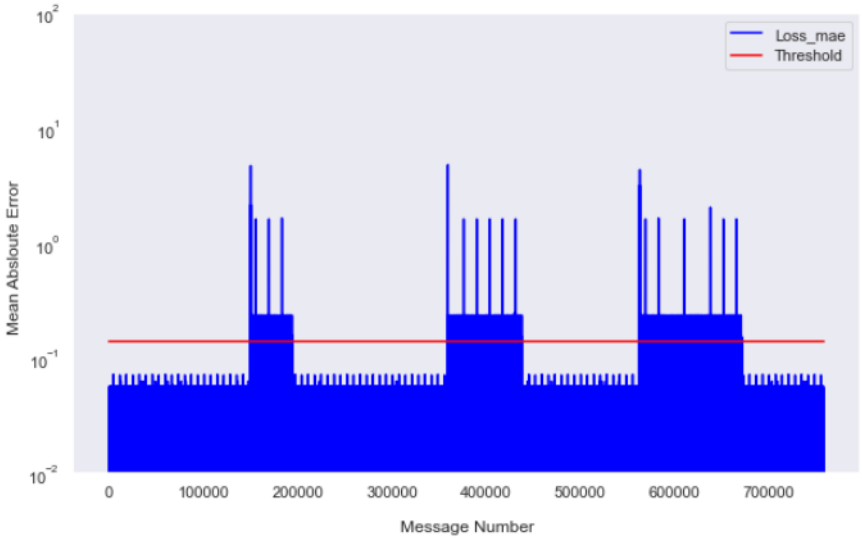


Figure B-5: Dataset 1 - MAE vs MSG # for Scenario 3a

Appendix C - Dataset 2 Model MAE vs MSG# Graphs

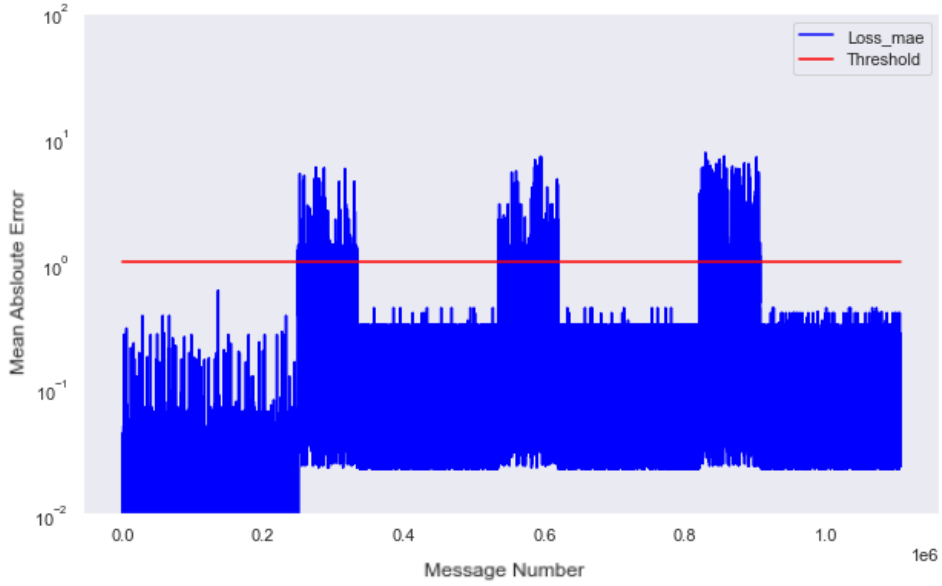


Figure C-1: Dataset 2 - MAE vs MSG # for Scenario 1a

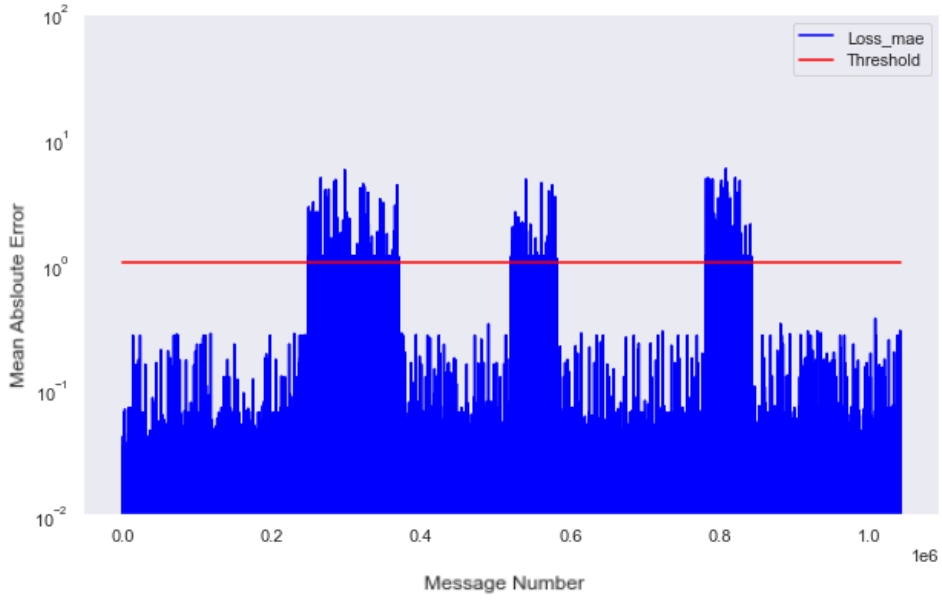


Figure C-2: Dataset 2 - MAE vs MSG # for Scenario 2a

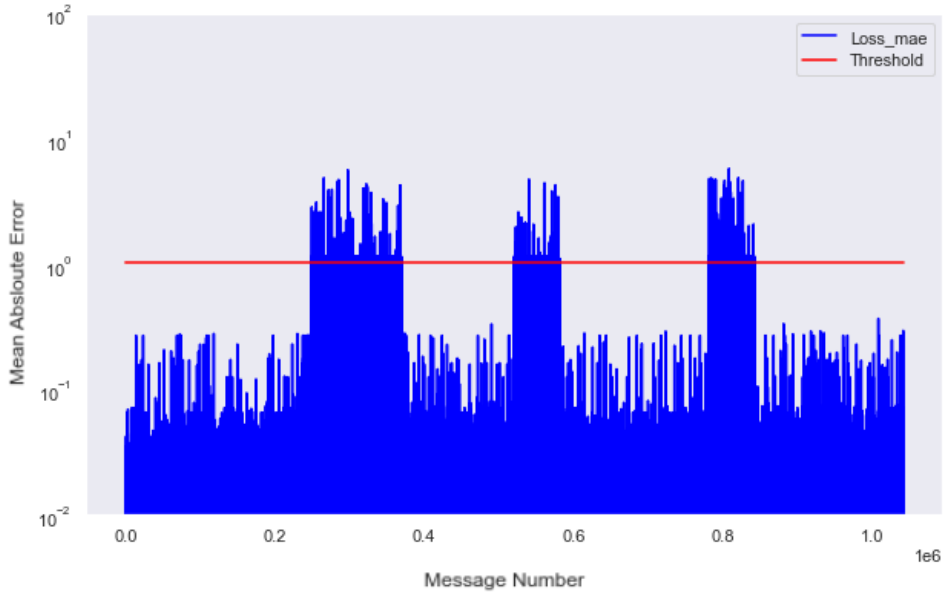


Figure C-3: Dataset 2 - MAE vs MSG # for Scenario 2b

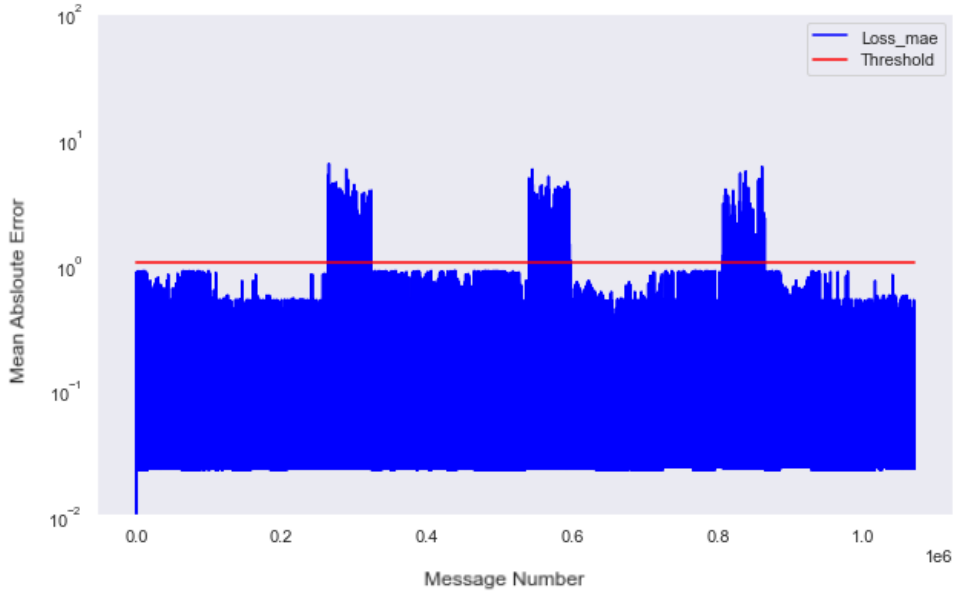


Figure C-4: Dataset 2 - MAE vs MSG # for Scenario 3a

Appendix D - Feature Extraction Program (Python)

```
1.  from __future__ import absolute_import, division, print_function,
    unicode_literals
2.  import tensorflow as tf
3.  import matplotlib as mpl
4.  import matplotlib.pyplot as plt
5.  import numpy as np
6.  import os
7.  import pandas as pd
8.
9.  # Reads in BMDX file converted in txt or csv format for feature
10. # extraction
11. # Outputs .csv file with extracted features
12.
13.
14. # Set path of converted BMDX file (.txt or .csv)
15. path =r'C:\Users\User\Desktop\DL\Final Dataset'
16. csv = r'baseline_inflight_5_240920.txt'
17. final_csv = csv.replace('.txt','')+"_Features.csv"
18. # use .csv instead of .txt if required
19.
20. csv_path=(path+"\\"+csv)
21. # use "," as the delimiter if required
22.
23. df = pd.read_csv(csv_path,delimiter="\t",converters={"CMD1  ": lambda
    x: int(x, 16),"CMD2  ": lambda x: int(x, 16),"Err/Sts": lambda x:
    int(x, 16),'Err/Sts.1': lambda x: int(x, 16), 'STS1  ': lambda x:
    int(x, 16), 'Err/Sts.2': lambda x: int(x, 16), 'STS2  ': lambda x:
    int(x, 16), 'Err/Sts.3': lambda x: int(x, 16),'Int/Status': lambda x:
    int(x, 16)})
24.
25. df = df.fillna(0)
26. bincolumns = ['CMD1  ', 'Err/Sts', 'CMD2  ','Err/Sts.1', 'STS1  ',
    'Err/Sts.2','STS2  ', 'Err/Sts.3']
27.
28. for col in bincolumns:
29.     df[col] = df[col].apply(lambda x: format(int(x), '04b'))
30.     df[col] = df[col].astype(str).str.pad(16,fillchar='0')
31.
32.
33. df['Int/Status'] = df['Int/Status'].apply(lambda x: format(int(x),
    '04b'))
34. df['Int/Status'] =
    df['Int/Status'].astype(str).str.pad(32,fillchar='0')
35.
36. allcolumns = ['MSG #  ', 'Time Stamp          ','CMD1  ',
    'Err/Sts','RSP1','Err/Sts.1', 'STS1  ','CMD2  ','RSP2',
    'Err/Sts.2','STS2  ', 'Err/Sts.3', 'Int/Status']
37. sdata = df[allcolumns]
38.
39.
```

```
40.
41. splitted = sdata['CMD1  '].apply(lambda x: pd.Series(list(x)))
42. splitted.columns = ['CMD1 B'+str(x) for x in splitted.columns]
43. sdata = sdata.join(splitted)
44.
45. splitted = sdata['STS1  '].apply(lambda x: pd.Series(list(x)))
46. splitted.columns = ['STS1 B'+str(x) for x in splitted.columns]
47. sdata = sdata.join(splitted)
48.
49. splitted = sdata['CMD2  '].apply(lambda x: pd.Series(list(x)))
50. splitted.columns = ['CMD2 B'+str(x) for x in splitted.columns]
51. sdata = sdata.join(splitted)
52.
53. splitted = sdata['STS2  '].apply(lambda x: pd.Series(list(x)))
54. splitted.columns = ['STS2 B'+str(x) for x in splitted.columns]
55. sdata = sdata.join(splitted)
56.
57. splitted = sdata['Int/Status'].apply(lambda x: pd.Series(list(x)))
58. splitted.columns = ['Int/Status B'+str(x) for x in splitted.columns]
59. sdata = sdata.join(splitted)
60.
61. sdata["CMD1-addr"] = (sdata["CMD1 B0"] + sdata["CMD1 B1"]+sdata["CMD1
    B2"]+sdata["CMD1 B3"]+sdata["CMD1 B4"]).apply(int, args=(2,))
62. sdata["CMD1-TR"] = sdata["CMD1 B5"]
63. sdata["CMD1-subaddr"] = (sdata["CMD1 B6"] + sdata["CMD1
    B7"]+sdata["CMD1 B8"]+sdata["CMD1 B9"]+sdata["CMD1 B10"]).apply(int,
    args=(2,))
64. sdata["CMD1-numword"] = (sdata["CMD1 B11"] + sdata["CMD1
    B12"]+sdata["CMD1 B13"]+sdata["CMD1 B14"]+sdata["CMD1 B15"]).apply(int,
    args=(2,))
65.
66. sdata["STS1-addr"] = (sdata["STS1 B0"] + sdata["STS1 B1"]+sdata["STS1
    B2"]+sdata["STS1 B3"]+sdata["STS1 B4"]).apply(int, args=(2,))
67. sdata["STS1-Error"] = sdata["STS1 B5"]
68. sdata["STS1-Inst"] = sdata["STS1 B6"]
69. sdata["STS1-SerReq"] = sdata["STS1 B7"]
70. sdata["STS1-Reserved"] = (sdata["STS1 B8"] + sdata["STS1
    B9"]+sdata["STS1 B10"]).apply(int, args=(2,))
71. sdata["STS1-BCRecv"] = sdata["STS1 B11"]
72. sdata["STS1-Busy"] = sdata["STS1 B12"]
73. sdata["STS1-SubFlag"] = sdata["STS1 B13"]
74. sdata["STS1-DBAcc"] = sdata["STS1 B14"]
75. sdata["STS1-TerFlag"] = sdata["STS1 B15"]
76. sdata["CMD2-addr"] = (sdata["CMD2 B0"] + sdata["CMD2 B1"]+sdata["CMD2
    B2"]+sdata["CMD2 B3"]+sdata["CMD2 B4"]).apply(int, args=(2,))
77. sdata["CMD2-TR"] = sdata["CMD2 B5"]
78. sdata["CMD2-subaddr"] = (sdata["CMD2 B6"] + sdata["CMD2
    B7"]+sdata["CMD2 B8"]+sdata["CMD2 B9"]+sdata["CMD2 B10"]).apply(int,
    args=(2,))
79. sdata["CMD2-numword"] = (sdata["CMD2 B11"] + sdata["CMD2
    B12"]+sdata["CMD2 B13"]+sdata["CMD2 B14"]+sdata["CMD2 B15"]).apply(int,
    args=(2,))
80.
81. sdata["STS2-addr"] = (sdata["STS2 B0"] + sdata["STS2 B1"]+sdata["STS2
    B2"]+sdata["STS2 B3"]+sdata["STS2 B4"]).apply(int, args=(2,))
```

```
sdata["STS2-Error"] = sdata["STS2 B5"]
sdata["STS2-Inst"] = sdata["STS2 B6"]
sdata["STS2-SerReq"] = sdata["STS2 B7"]
sdata["STS2-Reserved"] = (sdata["STS2 B8"] + sdata["STS2
    B9"]+sdata["STS2 B10"]).apply(int, args=(2,))
sdata["STS2-BCRecv"] = sdata["STS2 B11"]
sdata["STS2-Busy"] = sdata["STS2 B12"]
sdata["STS2-SubFlag"] = sdata["STS2 B13"]
sdata["STS2-DBAcc"] = sdata["STS2 B14"]
sdata["STS2-TerFlag"] = sdata["STS2 B15"]

sdata["BT1553_INT_HIGH_WORD"] = sdata["Int/Status B31"]
sdata["BT1553_INT_INVALID_WORD"] = sdata["Int/Status B30"]
sdata["BT1553_INT_LOW_WORD"] = sdata["Int/Status B29"]
sdata["BT1553_INT_INVERTED_SYNC"] = sdata["Int/Status B28"]

sdata["BT1553_INT_MID_BIT"] = sdata["Int/Status B27"]
sdata["BT1553_INT_TWO_BUS"] = sdata["Int/Status B26"]
sdata["BT1553_INT_PARITY"] = sdata["Int/Status B25"]
 sdata["BT1553_INT_NON_CONT_DATA"] = sdata["Int/Status B24"]

 sdata["BT1553_INT_EARLY_RESP"] = sdata["Int/Status B23"]
 sdata["BT1553_INT_LATE_RESP"] = sdata["Int/Status B22"]
 sdata["BT1553_INT_BAD_RTADDR"] = sdata["Int/Status B21"]
 sdata["BT1553_INT_CHANNEL"] = sdata["Int/Status B20"]
 # B19 (corresponds to bit 12 0x00001000L) is not used
 sdata["BT1553_INT_WRONG_BUS"] = sdata["Int/Status B18"]
 sdata["BT1553_INT_BIT_COUNT"] = sdata["Int/Status B17"]
 sdata["BT1553_INT_NO_IMSG_GAP"] = sdata["Int/Status B16"]

 sdata["BT1553_INT_END_OF_MESS"] = sdata["Int/Status B15"]
 sdata["BT1553_INT_BROADCAST"] = sdata["Int/Status B14"]
 sdata["BT1553_INT_RT_RT_FORMAT"] = sdata["Int/Status B13"]
 sdata["BT1553_INT_RESET_RT"] = sdata["Int/Status B12"]

 sdata["BT1553_INT_SELF_TEST"] = sdata["Int/Status B11"]
 sdata["BT1553_INT_MODE_CODE"] = sdata["Int/Status B10"]
 sdata["BT1553_INT_NOCMD"] = sdata["Int/Status B9"]
 sdata["BT1553_INV_RTRT_TX"] = sdata["Int/Status B8"]

 sdata["BT1553_RTRT_RCV_NRSP"] = sdata["Int/Status B7"]
 sdata["BT1553_INT_RETRY"] = sdata["Int/Status B6"]
 sdata["BT1553_INT_NO_RESP"] = sdata["Int/Status B5"]
 sdata["BT1553_INT_ME_BIT"] = sdata["Int/Status B4"]


 sdata["BT1553_INT_TRIG_BEGIN"] = sdata["Int/Status B3"]
 sdata["BT1553_INT_TRIG_END"] = sdata["Int/Status B2"]
 sdata["BT1553_INT_BM_OVERFLOW"] = sdata["Int/Status B1"]
 sdata["BT1553_INT_ALT_BUS"] = sdata["Int/Status B0"]
 col = ['Time Stamp          ','CMD1-addr', 'CMD1-TR', 'CMD1-
subaddr', 'CMD1-numword','RSP1','STS1-addr','STS1-Error','STS1-
Inst','STS1-SerReq','STS1-Reserved','STS1-BCRecv','STS1-Busy','STS1-
SubFlag','STS1-DBAcc','STS1-TerFlag', 'CMD2-addr', 'CMD2-TR', 'CMD2-
subaddr', 'CMD2-numword','RSP2','STS2-addr','STS2-Error', 'STS2-
```

```
    Inst','STS2-SerReq','STS2-Reserved','STS2-BCRecv','STS2-Busy','STS2-
    SubFlag','STS2-DBAcc','STS2-
    TerFlag','BT1553_INT_HIGH_WORD','BT1553_INT_INVALID_WORD','BT1553_INT_L
    OW_WORD','BT1553_INT_INVERTED_SYNC','BT1553_INT_MID_BIT','BT1553_INT_TW
    O_BUS','BT1553_INT_PARITY','BT1553_INT_NON_CONT_DATA','BT1553_INT_EARLY
    _RESP','BT1553_INT_LATE_RESP','BT1553_INT_BAD_RTADDR','BT1553_INT_CHANN
    EL','BT1553_INT_WRONG_BUS','BT1553_INT_BIT_COUNT','BT1553_INT_NO_IMSG_G
    AP','BT1553_INT_END_OF_MESS','BT1553_INT_BROADCAST','BT1553_INT_RT_RT_F
    ORMAT','BT1553_INT_RESET_RT','BT1553_INT_SELF_TEST',
    'BT1553_INT_MODE_CODE','BT1553_INT_NOCMD','BT1553_INV_RTRT_TX','BT1553_
    RTRT_RCV_NRSP','BT1553_INT_RETRY','BT1553_INT_NO_RESP','BT1553_INT_ME_B
    IT','BT1553_INT_TRIG_BEGIN','BT1553_INT_TRIG_END','BT1553_INT_BM_OVERFL
    OW','BT1553_INT_ALT_BUS']
132.
133.  finDF = sdata[col]
134.  finDF.index=sdata['MSG #  ']
135.  save_csv_path=(path+"\\"+final_csv)
136.  finDF.to_csv(save_csv_path)
```