OD1NF1ST: True Skip Intrusion Detection and Avionics Network Cyber-Attack Simulation

Journal:	*Transactions on Cyber-Physical Systems*
Manuscript ID	TCPS-2021-0064
Manuscript Type:	Original Manuscript
Date Submitted by the Author:	09-Sep-2021
Complete List of Authors:	Wrana, Michael; Queen's University, Elsayed, Marwa; Queen's University Lounis, Karim; Queen's University Mansour, Ziad; Queen's University Ding, Steven; Queen's University Zulkernine, Mohammad; Queen's University, School of Computing
Keywords:	Intrusion detection systems, Artificial intelligence, Embedded and cyber-physical systems

OD1NF1ST: True Skip Intrusion Detection and Avionics Network Cyber-Attack Simulation

MICHAEL WRANA, MARWA ELSAYED, KARIM LOUNIS, ZIAD MANSOUR, STEVEN H. H. DING, and MOHAMMAD ZULKERNINE, Queen's University, Canada

MIL-STD-1553 is a communication bus that has been used by many military avionics platforms such as the F-15 and F-35 fighter jet for almost 50 years. Recently, it has become clear that the lack of security on MIL-STD-1553 and the requirement for internet communication between planes has revealed numerous potential attack vectors for malicious parties. Prevention of these attacks by modernizing the MIL-STD-1553 is not practical due to the military applications and existing far-reaching installations of the bus. We present a software system that can simulate bus transmissions to create easy, replicable, and large datasets of MIL-STD-1553 communications. We also propose an intrusion detection system (IDS) which can identify anomalies and the precise type of attack using recurrent neural networks with a reinforcement learning true-skip data selection algorithm. Our IDS outperforms existing networks designed for MIL-STD-1553 in binary anomaly detection tasks while also performing attack classification and minimizing computational resource cost. Our simulator can generate more data with higher fidelity than existing methods and integrate attack scenarios with greater detail. Furthermore, the simulator and IDS can be combined to form a web-based attack-defense game.

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**; • **Security and privacy** → **Intrusion detection systems**; • **Computer systems organization** → **Embedded and cyber-physical systems**.

ACM Reference Format:
Michael Wrana, Marwa Elsayed, Karim Lounis, Ziad Mansour, Steven H. H. Ding, and Mohammad Zulkernine. 2021. OD1NF1ST: True Skip Intrusion Detection and Avionics Network Cyber-Attack Simulation. 1, 1 (September 2021), 20 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The MIL-STD-1553 protocol is a standard for communications that was developed by the United States Department of Defense mainly for use in avionics platforms [5]. MIL-STD-1553 is intended to allow for communication between different devices physically connected through a bus. Since its creation, the communication bus has been installed on many avionics platforms in service with the United States Air Force and Navy, NATO coalition aircraft in Europe, Russian fighter jets, and satellites operated by NASA.

The original specification focuses heavily on fault tolerance and reliability due to the intended military and aerospace applications. The MIL-STD-1553 is synchronous (using a master/slave topology), halfduplex, and deterministic. The bus uses a multi-point topology to link devices (terminals) through a shared backbone (bus). Despite the care taken to design MIL-STD-1553's reliability system, security protocols were merely an afterthought. In the 1970s when MIL-STD-1553

All the authors are with School of Computing, Queen's University, Canada. E-mails:{16mmw, marwa.afifi, karim.lounis, steven.ding, mz}@queensu.ca .
Authors' address: Michael Wrana; Marwa Elsayed; Karim Lounis; Ziad Mansour; Steven H. H. Ding; Mohammad Zulkernine, Queen's University, 557 Goodwin Hall, Kingston, Ontario, Canada, K7L 2N8.

XXXX-XXXX/2021/9-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

was first developed, our modern notion of cybersecurity did not exist. Since first being installed in the F-16 Fighting Falcon, MIL-STD-1553 has spread to billions of dollars worth of internet-connected military hardware across the globe. However, the system is not robust towards modern cyber-attacks like denial of service (DoS) or man-in-the-middle (MITM) [10]. Redesigning MIL-STD-1553 from scratch has been tried but is impractical due to the bus' wide-reaching installations [20]. An alternative solution is to augment the existing protocol with protection.

The best form for such an augmented protection device is an intrusion detection system (IDS) [17]. An IDS is not designed to stop attacks from being possible but rather identify when an intrusion has happened and notify the relevant parties and can provide the required security to avionic and satellite platforms while being practical to apply to existing installations of MIL-STD-1553. Recent studies have proposed several data-driven IDS solutions for MIL-STD-1553 [8, 11, 18, 21, 22]. Existing work has adopted a co-simulation method whereby communication traffic is directly observed from the physical bus while data being transmitted is algorithmically generated as synthetic messages. The results described are promising, however they suffer from several common issues:

- I1: Short simulation duration and synthetic communication messages. Much of the existing work simulates benign traffic and cyber attacks in a very limited timeframe of 1-20 minutes. The generated data cannot effectively evaluate how the proposed IDS adopts to changing environments in different flight scenarios. Additionally, the data is collected by algorithmically generated messages rather than actual flight communications.
- I2: Limited number of attack scenarios and their combinations. Most of the existing work is implemented and independently simulated with only 1-3 attacks.
- I3: Only consider one type of communication messages due to computational limits. Existing methods only analyze a subset of transmitted words (command words). However, based on our analysis of the protocol, some attacks can be successfully executed without using any command words.

I1 and I2 are caused by the lack of a comprehensive and flexible full-stack simulation system that is able to 1) repetitively simulate bus traffic in a long run, 2) simulate cyber attacks of various vectors for many runs, and 3) simulate normal behaviors with high-fidelity flight data. I3 was caused by the lack of a learning system that is able to scale with the number of messages being communicated. Furthermore, current IDS' only provide binary anomaly detection and cannot explain the type of attack occurring. The main contributions of this paper are as follows:

- We build a complete open-source simulation system of the MIL-STD-1553 communication protocol which enables implementation of custom and pre-defined advanced cyber attacks over an arbitrary simulation platform. We also collect high-fidelity data over long time periods.
- We propose a new neural network architecture that provides a data skipping mechanism specifically designed for IDS'. It can identify whether a message is considered anomalous and specify the precise type of attack with greater performance compared to other methods. Our IDS can be integrated with the simulator to allow for a real-time attack-defense game.
- We experimentally demonstrate the effectiveness of our IDS through comparison with existing state-of-the-art neural network models designed for avionics platforms.

2 ARCHITECTURE AND PROTOCOLS

First, we will provide an overview of the MIL-STD-1553 protocol in terms of its components, word types, and message protocols.

2.1 Bus Architecture

MIL-STD-1553 consists of a Bus Controller (BC) controlling multiple terminals (RTs) linked by a physical bus which provides a single data path between the controller and associated terminals.

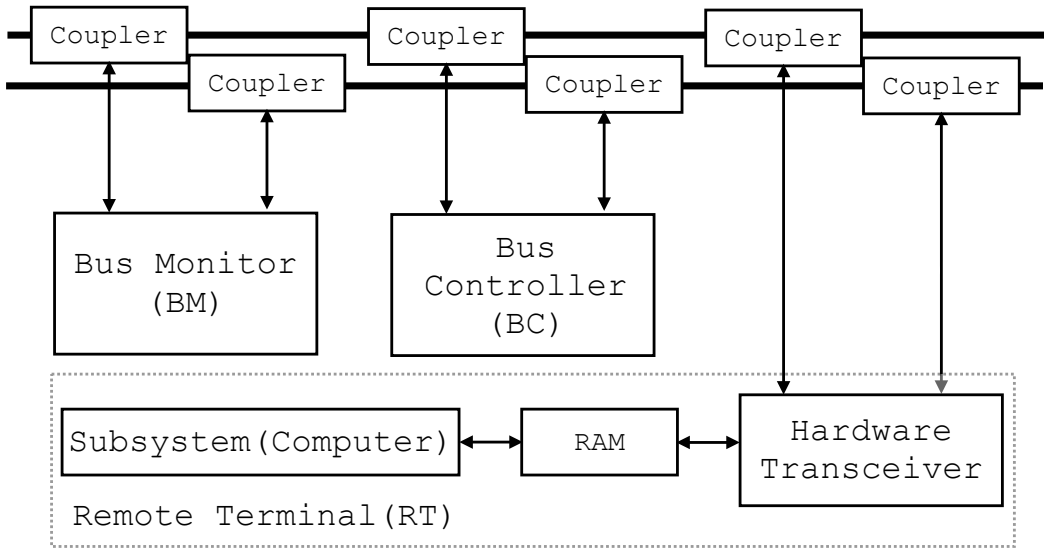

Fig. 1. MIL-STD-1553 Architecture and its Primary Components

Every device linked to the data line can read every word (sent in plain text) that is transmitted. The system also supports double and triple redundancy by coupling each device to multiple data lines. **Figure 1** shows a double-redundant system.

2.2 Bus Components

There are three main devices connected to a MIL-STD-1553 system which are summarized below and in **Figure 1**.

Bus Controller (BC): The bus controller is the master device and manages all communications over the network. It can initiate data transmission between devices.

Remote Terminal (RT): The remote terminal can be used to connect the data bus to a computing subsystem or link one MIL-STD-1553 to another. RTs are slaves and can only send words through the bus when commanded to by the BC. There can be a maximum of 30 RTs on a single network.

Bus Monitor (BM): The bus monitor records words that are transmitted through MIL-STD-1553. Depending on available storage, every word or some specified subset may be recorded. This component is critical for an IDS, as the data collected by the BM can be used to train models and evaluate a system for intrusions.

2.3 Word Types

There are three unique types of messages (words) that can be sent by devices on MIL-STD-1553 which are detailed in **Figure 2**. Each word consists of 20 bits. The first 3 are for synchronization and contain a Manchester code [16] that allows for clock and data information to be sent on the same wire. The next 16 contain a payload that differs depending on the message type. The final bit is for error/parity verification.

Command: Command words are used by the BC to instruct a specific device on the bus to take an action.

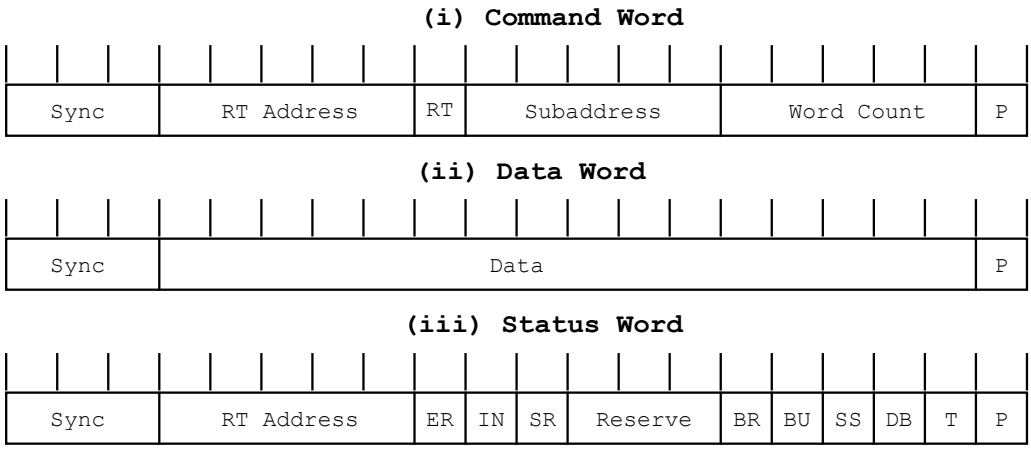

Fig. 2. Contents of (i) Command (ii) Data (iii) Status Words in MIL-STD-1553 Protocol

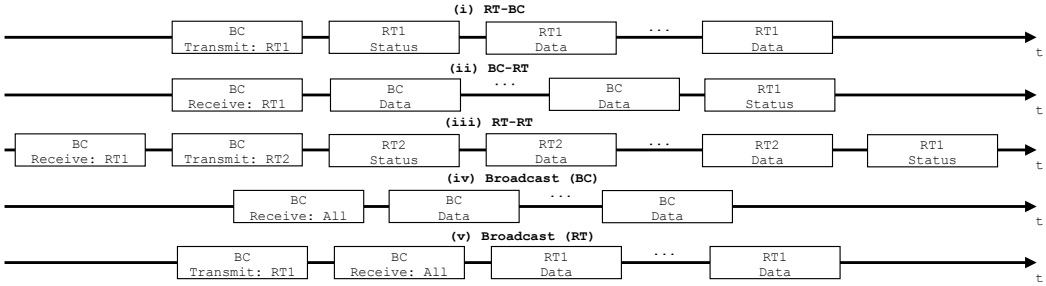

Fig. 3. Data Transfer Protocols on MIL-STD-1553 for (i) RT-BC (ii) BC-RT (iii) RT-RT (iv) Broadcast (BC) (v) Broadcast (RT)

Data: Data words are used to send information from one device to another. The 16 bits of payload contain the data being transmitted over the bus.

Status: Status words are used by devices to respond to the BC before data transmission or after data reception.

2.4 Message Protocols

There are four types of message protocols that can be initiated by the BC on MIL-STD-1553 which are summarized in **Figure 3**. The standard does not specify a particular order of command sequences, this is left to be programmed by each individual user. On most military aircraft the BC has a preset cyclic schedule of data transfers to be executed.

RT-BC: This protocol allows for one-way data transfer from an RT to the BC. First, the BC sends a `transmit` word specifying an RT which will be sending data. Next, the targeted RT will respond with a `status` word indicating any issues before transmission. Immediately following the status word, the RT will send a number of `data` words as specified in `transmit` to the bus.

BC-RT: This protocol facilitates one-way data transfer from the BC to an RT. First, the BC sends a `receive` indicating the RT that will be receiving data. Immediately following this, the BC will send a number of `data` words as specified in `receive` to the bus. Following reception of

the data words, the targeted RT will respond with a status word indicating any issues with the transmission.

RT-RT: This protocol provides one-way data transfer from an RT to another RT. First, the BC sends receive to the bus indicating RT1 which is receiving data. Next, the BC sends a transmit specifying RT2 which is sending data. RT2 responds with a status word indicating any problems immediately followed by the requested number of data words. RT1 finishes the protocol by sending a status word indicating any issues with the transmission.

Broadcast: There are two defined types of broadcast protocols which allow one device to send data to every other device. First, the BC broadcast which begins with the BC sending receive to all devices followed by the specified number of data words. No RTs will respond following after the data words have been transmitted. Alternatively, the BC can first send a transmit word indicating a specific RT. The targeted RT will send data to the bus and every device will interpret it.

3 BACKGROUND & RELATED WORKS

System Security. Here, we summarize the main issues with security on MIL-STD-1553 and describe existing solutions to these problems. There are five typical fundamental cybersecurity categories: Authentication, Confidentiality, Integrity, Availability, and Non-repudiation. The triad of confidentiality, integrity, and availability all rest on the service provided by authentication. Since no authentication system exists in MIL-STD-1553, these three paradigms are severely compromised within the network.

Maintaining confidentiality on MIL-STD-1553 is crucial due to the sensitive and easily exploitable nature of secure military communications. There is no consideration of this paradigm in MIL-STD-1553, as all messages are publicly available to every device. Appliances have been designed which can break system confidentiality without direct installation [4].

MIL-STD-1553 is somewhat robust towards integrity attacks as words cannot be intercepted or modified after transmission [23]. Attacks such as MITM that modify words are theoretically possible although have infinitesimal chances of success due to the small inter-message time gap [10]. Some IDS' have been designed which can detect specific types of integrity attacks on the platform [8, 11].

MIL-STD-1553 is vulnerable to many availability attacks as collisions between words are possible if a malicious party is sending messages at unexpected times [10]. Most existing IDS' can successfully detect some form of availability attacks [8, 11, 18, 22].

Data Generation. One of the major challenges associated with this research area is appropriate dataset generation. An overview of existing approaches can be found in **Table 1**. There are no publicly accessible databases containing real message sequences. As a result, different authors have created independent datasets to prove the effectiveness of their IDS.

Yahalom *et al.* have created three publicly available datasets to be used for IDS testing [26]. Networks struggled to train on this dataset due to the limited attack scenarios (two) and lack of malicious messages [21].

Most authors used a semi-physical setup involving a physical bus emulator and software IDS [6]. Stan *et al.* created their dataset and attack scenarios using a multi-PC physically simulated environment where one machine is benign and another malicious [22]. Losier *et al.* and He *et al.* created benign data with a physical simulation then digitally inserted attack scenarios to later be detected by the IDS [11, 18]. Genereux *et al.* used a purely software-based simulation where data and attacks are generated digitally [8].

Intrusion Detection Systems. Intrusion Detection Systems have been used to secure similar embedded systems [1, 9]. A summary of existing MIL-STD-1553 IDS' is in **Table 1**. Stan *et al.* generated a baseline using a Markov Chain statistical model and evaluated its effectiveness against

Table 1. Overview and Comparison of Existing Intrusion Detection Systems for MIL-STD-1553

Author	Dataset	IDS Type	IDS Method	Security
Stan et al.	semi-physical	Baseline	Markov Chain	A
Genereux et al.	software	Baseline	Histogram	I, A
Losier et al.	semi-physical	Baseline	Histogram	-
He et al.	semi-physical	Classification	Naïve Bayes	I, A
Onodueze et al.	software	Classification	Deep Learning	I, A

some pre-planned attack scenarios [22]. Genereux *et al.* used a similar system by generating a baseline histogram of message timings then compared the graph during pre-calculated attack scenarios [8]. Losier *et al.* suggested a similar histogram approach, however the detailed attack scenarios are classified and thus cannot be compared [18]. This approach and IDS design has proven to be effective, but the scope and generalizability to the real world were somewhat limited. Another IDS proposed by He *et al.* instead used a Naive Bayes classifier to identify if a message is anomalous or not [11].

Machine learning based anomaly detection has been shown to be effective in cyber-physical systems [7, 13, 15, 25]. Some machine learning based IDS' have also been tried on MIL-STD-1553 [21]. Onodueze *et al.* indicated that the imbalanced public dataset [26] and lack of attack scenarios resulted in poor performance. Furthermore, any practical IDS for MIL-STD-1553 also must function with the limited computing power that is available on older avionics platforms [24]. Some solutions to minimizing the computational efficiency for Recurrent Neural Network (RNN) based anomaly detectors include the skip-RNN and leap-LSTM [2, 12].

4 SIMULATION SYSTEM

The simulator component of OD1NF1ST presents significant advantages over existing semi-physical and software systems designed to replicate the communication bus.

First, our simulator can produce communication data over nearly unlimited time periods with greater detail compared to existing methodologies. The system provides high-fidelity data containing exact timestamps and word contents without the need for user interaction. For example, in this paper we use 42 hours of continuous MIL-STD-1553 communications which contains 250x more data than any previously published method.

Secondly, our software package allows for effortless implementation of advanced cyber attack scenarios and we provide exploits of 10 different MIL-STD-1553 vulnerabilities covering all the three system security categories. This is an improvement over other methods that only integrate 2-4 intrusion scenarios and focus on specific subsets of system security paradigms. Furthermore, as research continues in this area, new attacks can be easily integrated into the existing system. This provides researchers with a future-proof way to collect up-to-date data for training any MIL-STD-1553-based IDS.

Thirdly, our simulator can be simply linked with other software to create an efficient and simple pipeline. OD1NF1ST can begin with raw flight simulation data and end with real-time classification of word sequences.

Here, we will provide an overview of the simulation package in terms of its architecture, implementation, and pipeline integration.

4.1 Simulation Software

OD1NF1ST has two main modes of operation: terminal-based data collection, and web-based interactive flight & attack simulation. A repository containing the terminal-based mode, interactive display, sample datasets, and attack scenarios can be found on the project Github [1]. An overview of the system architecture is described in **Figure 4**. The simulator consists of three main components:

- The transmission layer which replicates the raw data transfer between devices connected to MIL-STD-1553.
- The event system which controls how devices coupled to the bus react based on the information being transmitted.
- The Interactive Display package which can be used to host a live interactive website and simulate real-time attack and defense scenarios.

4.2 Transmission Layer

Our software simulator uses two different systems to imitate the data transmission (physical and message layers) of MIL-STD-1553. The physical layer is replicated using the User Datagram Protocol (UDP) computer networking protocol. UDP allows devices to send datagrams to other hosts on an IP network without the need for prior communications to create a data line. The message layer is simulated using a software-based proxy manager. The manager controls the simulator's proxy servers along with writing and implementing their policies and filtering resource requests.

4.2.1 The Proxy Manager. The proxy manager is a process on the simulator used to administrate and control any communications sent through UDP. It contains objects which represent the physical bus itself as well as any devices which would be coupled to it. The default system configuration contains a maximum of 32 devices: 30 RTs, 1 BC and 1 BM. Backup BCs or additional BMs to replace a faulty BC can be added on demand. Connection between multiple MIL-STD-1553s through communication-configured RTs is also supported.

4.2.2 Device. Each device runs as an independent thread and contains a proxy to its respective device object. A single unified class allows for easy conversion of a backup BM to BC or BC to RT (as is possible on MIL-STD-1553) without reconnecting to the bus. Each object contains device information such as:

- Mode: `RT | BC | BM`
- State: `transmit | receive | idle | off`
- A reference to `bus` objects this device is coupled to
- A queue of messages transmitted over the network but not yet processed

Each device proxy also contains 2 UDP sockets: one for message reception and another for transmission. Every word transmitted to the `bus` is read by every device through its reception socket and added to its individual message queue. The devices process words one by one in their queue and react accordingly using the event system.

4.2.3 Bus. This object represents the main data line and every device is coupled to it through UDP sockets. Multiple buses can be created and coupled to each device to simulate doubly and triply redundant systems. The bus object contains system information such as:

- A proxy for communication over the UDP network
- A dictionary of references to all devices connected to the bus.
- Methods for sending data to all other devices and receiving transmissions from another device.

[1] OD1NF1ST Repository. https://github.com/L1NNA/OD1NF1ST

Table 2. Overview of the OD1NF1ST Event System and Related Communication Protocols

Event Name	Description	Protocols
RT_REC_CMD	RT reads a `transmit` command word and `address == this.address`	BC-RT RT-RT Broadcast
RT_TRS_CMD	RT reads a `receive` command word and `address == this.address`	RT-BC RT-RT Broadcast
RT_REC_DATA	RT reads a data word and `this.state==receive`	RT-RT BC-RT Broadcast
BC_REC_DATA	BC reads a data word after sending `command:transmit` to an RT	RT-BC Broadcast
BC_REC_STS	BC reads a status word	BC-RT

4.3 Event System

When a message is transmitted through the UDP network and received by a device, it must be processed and the device should react appropriately based on the MIL-STD-1553 protocol. Every word is always read by each device connected to the network, however the devices will behave differently depending on their current state and mode. To replicate this, OD1NF1ST uses an event system which doubles as a way to monitor and train the IDS. An example of some events in the system are described in **Table 2**.

4.4 Flight Simulator Pipeline

The OD1NF1ST simulator package can also be integrated with flight simulation software and an IDS to create a full data pipeline. Here, we will summarize our system for linking the simulator to its inputs and outputs. An overview of the pipeline can be seen in **Figure 4**.

- We first collect flight data so the content of each word being transmitted through the simulated bus is realistic. The main tool used to collect data is Microsoft Flight Simulator 2020[2] (MSFS) and the SimConnect[3] (SC) Python package. MSFS creates realistic aircraft flight scenarios and SC allows for real-time collection of data with matching video.
- The MSFS recording can feed directly into the simulator which has a pre-programmed cyclic communication scenario similar to ones found on military aircraft. The communication pattern can be altered by the user if required. The simulator produces either real-time recordings (for the interactive display) or packs them into a JSON[4] file for saving and processing. Each file contains objects representing words with a timestamp, payload details, and anomaly label. Each file contains an attack type label but is not guaranteed to have an intrusion. The length of recordings for each JSON are user-dependent with a default of 3-5 minutes.
- Next we process the JSONs and convert them to integer sequences to feed the IDS. The data can be split for cross-validation and performance analysis or used exclusively for training. Once trained, the network can be used to detect intrusions on real-time simulator data through the interactive display component (Section 4.5). Details on the conversion process and our IDS can be found in Sections 6 and 7.

[2]MSFS Website. https://www.flightsimulator.com/
[3]SimConnect API. https://pypi.org/project/SimConnect/
[4]JavaScript Object Notation. https://www.json.org/json-en.html

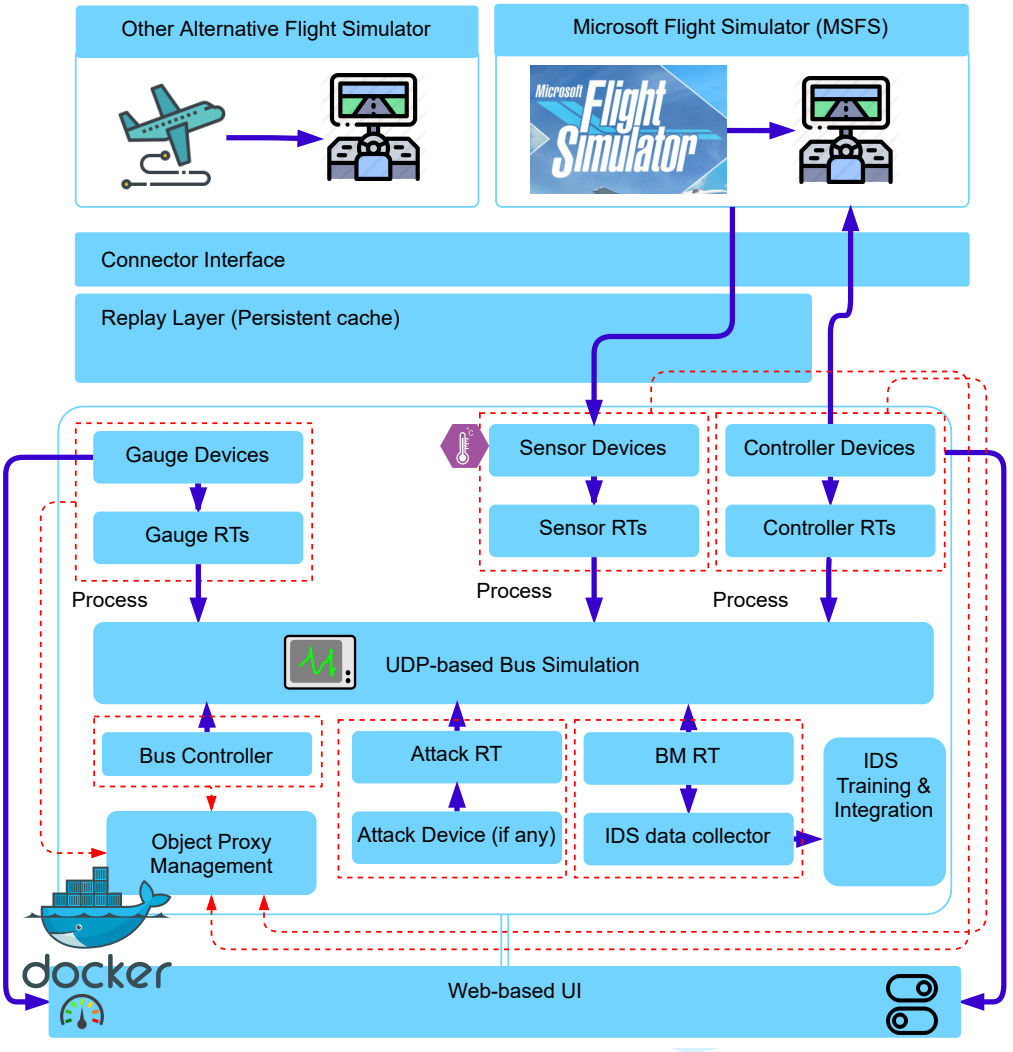

Fig. 4. Overview of Integrated Pipeline to Process OD1NF1ST Data

4.5 Interactive Display

The interactive display is an appealing feature of OD1NF1ST. In this system, a live simulation of a pre-planned flight route is hosted on a website, as seen in **Figure 5**. The transmissions of MIL-STD-1553 along with raw flight data are displayed to the user in **Figure 5 (i) and (ii)**. Here, the user acts as the 'hacker' and can launch any of the pre-defined attack scenarios against the bus or any connected device at any time in **Figure 5 (iii)**. The impact of such an attack can be visualized in the device communications and data display. An IDS is actively running and will attempt to detect the intrusions launched by the user. The identification of malicious messages is also shown to the user to see if their attacks were effective in beating the IDS in **Figure 5 (iv)**. Such a system can be hosted and used to crowd-source attack scenarios that defeat the IDS. This

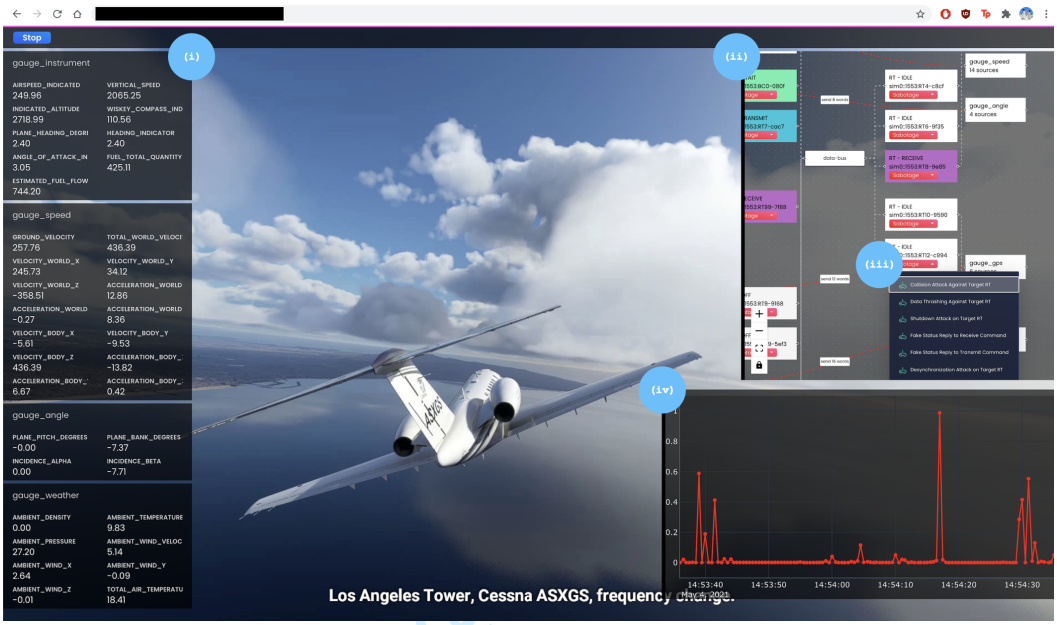

Fig. 5. OD1NF1ST Interactive Web Display Containing (i) Real-Time Flight Data (ii) Bus Simulation Overview (iii) Attack Launching Panel (iv) Intrusion Detection System. One can launch attacks on arbitrary system components and observe the IDS' response in real-time.

Table 3. Summary of MIL-STD-1553 Vulnerabilities Exploited by OD1NF1ST

Attack Type	Category	Protocols Exploited
Random-Word Generation	A	All
Man-in-the-Middle	C, I, A	All
Status Word Manipulation	I, A	RT-BC, BC-RT, RT-RT
Command Invalidation	A	RT-BC, RT-RT
TX Shutdown	I, A	All
Desynchronization	I, A	All
Data Trashing	A	BC-RT, RT-RT
Data Word Corruption	I	RT-BC, RT-RT

can be used to further improve both the complexity and capability of the attacks as well as the defence mechanisms.

5 ATTACK SCENARIOS

Our software package implements ten different MIL-STD-1553 exploits in eight categories. Here, we will describe each attack in detail and how it takes advantage of the communication protocol. The full implementation of each of these attack scenarios can be found on our Github[5]. A summary of the scenarios can be found in **Table 3**.

[5]OD1NF1ST Attack Scenarios. hidden for anonymity

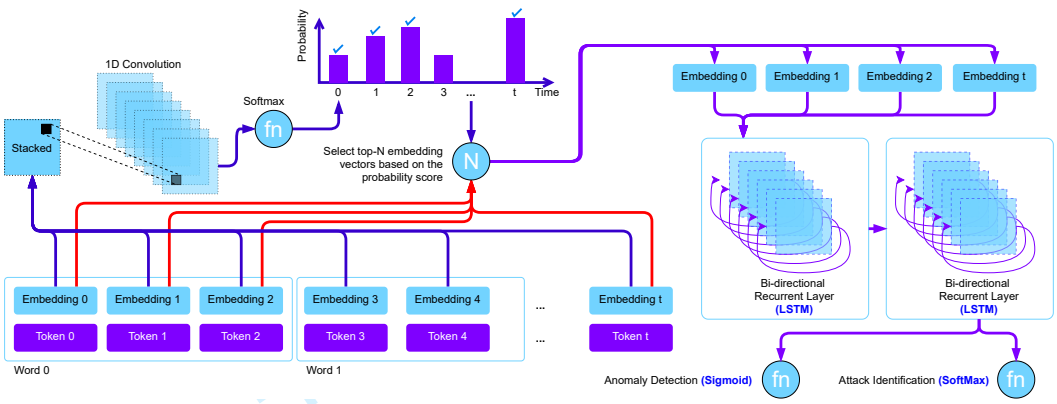

Fig. 6. OD1NF1ST Intrusion Detection System Architecture. Given a list of words being observed in a large sliding window, the IDS first converts each word into sub-tokens and maps each token to its respective embedding vector. Next, a 1D Convolution operation coupled with the softmax activation function estimates a probability $\in [0, 1]$ for each token. Then, a top-N operation selects the embedding vectors based on their respective probability score (in this example $N = 4$). Finally a two-layer Bi-Directional LSTM model is used for final predictions. It is noted that the top-N operation is non-differentiable and thus uses a REINFORCE Algorithm for training.

Random-Word Generation. This is a relatively simple Denial of Service attack [19]. The attacker waits for any word to be sent on the bus and attempts to also send a word containing random bits at the same time. The messages will collide on the bus and the benign messages are rendered unreadable. Alternatively, the attacker can simply constantly spam words onto the bus causing collisions. This attack has two different styles where either the currently communicating RT or any coupled RT can be targeted by the attacker.

Man-in-the-Middle (MITM). In MITM, the attacker seeks to insert a data transfer between two targeted RTs: RT0 and RT1. The attacker poses as a BC and sends fake `command:receive (address0)` and `command:transmit (address1)` words to the bus. The RTs have no way to identify the message source and immediately comply. If successful, the attacker can either gain access to classified data or send manipulated data to a device on the network. Although MITM is extremely dangerous, it requires the real BC to send no words during the entire operation, as any command words transmitted would cause a message collision rendering the attacker's data unreadable. Due to the cyclic nature of MIL-STD-1553, MITM attacks will most likely become ineffective DoS attacks.

Status Word Manipulation. To perform status word manipulation, the attacker impersonates an RT and attempts to intercept communication between two devices by sending a fake status word to the BC before the real one. This can be used to force a device into idle mode waiting for a word which never arrives. The timing is extremely precise and the attacker only has 14 μs to generate the fake word. Typically, it takes 20 μs to send messages to the bus, however this does not render the attack impossible. Instead, the RT will process the fake status word as a command word (since command and status words have the same three Manchester code sync bits). This type of attack can target either `receive` or `transmit` commands.

As an example, consider a typical benign RT-BC protocol. First, the BC sends a `command:transmit (address)` word targeting the RT. Upon detection of this word, the attacker will transmit a malicious status word to the bus. This fake word has two purposes: the BC will read it as `status:OK` from the RT and await data, while the RT will read it as `command` and return to idle. The final result

of status word manipulation is a frozen bus while the BC awaits data. Although this individual attack is not particularly threatening, it opens the bus to previously improbable attacks such as MITM.

Command Invalidation. In command invalidation, the attacker sends fake command words to the bus immediately following a benign one. This can be used to prevent regular data transmission and force an RT into idle mode. To implement this attack, the attacker must insert their word during the extended delay between command and status words. This means command invalidation is only possible during specific protocols. The legitimate BC may try and re-establish connection with the targeted RT due to the missing status word, so the attacker must use a command word which does not require any response such as `command:receive`.

As an example, consider a typical benign RT-RT protocol. First, the BC sends a `command:transmit (address0)` and `command: receive (address1)` for RT0 to send data to RT1. Following the second command word, the attacker inserts a false command word resetting RT1 to idle mode. Next, RT0 will send a `status` word followed by the `data` words. RT1 will not correctly process this data because it has been reset by the attacker.

TX Shutdown. In a TX Shutdown attack the attacker sends a `command:shutdown (address)` word to the main bus, disabling a particular RT. In a system that is not double or triple redundant this attack simply disables the target. When multiple bus lines are active, the attack is more difficult since the `status` word response on the backup bus may reactivate the target. In this case, the attacker must use `command:shutdown (all)` on both buses. Although that disables RTs on the network, it is impossible to go undetected since communications will cease. The currently active BC will simply begin to reactivate devices as is necessary during normal operation. Alternatively, the attacker can target a particular RT on a backup bus and disable its connection. A backup bus TX shutdown can easily go undetected and completely disable an RT in the case of partial bus or system failure.

Desynchronization. During a desynchronization attack the attacker sends a `command: resync (address)` word to the targeted RT, resetting its clock. This results in devices sending words at incorrect times causing message collisions and limiting the bus' transmission capabilities. The attack can be extended if the attacker follows the malicious `command:resync (address)` word with a `data` word containing new synchronization information. The targeted RT would no longer need to respond with a `status` word. By providing precise synchronization information, the attacker can control the flow of data on the compromised network. Furthermore, the attacker can send a `command:resync (all)` word, thus resetting the clock of every RT in the network.

Data Trashing. In a data trashing attack, the attacker inserts a command word during routine data transfer between two RTs. This results in the receiving RT trashing all the transmitted data words without processing them due to a state change upon reading the malicious command word. To implement this attack, the attacker waits for an RT-RT communication to enter the `data` transmission phase. A fake `command` is then inserted by the attacker before the final `data` can be sent. If the malicious command word does not require the reception RT to respond with a status word, the BC will detect the error and re-attempt the data transfer.

As an example, consider a benign data transfer protocol between two devices, RT0 and RT1. The BC initiates the transfer by sending `command:receive (address1)` and `command:transmit (address0)`. Before the final data word from RT0 is sent, the attacker inserts a fake `command:transmit (address1)`. RT1 will switch to transmission mode, and the BC will assume the transfer was successful upon reception of a `status:OK` from RT1. The BC will think the transfer was successful, however the `status:OK` was actually a response to the malicious command word.

Data Word Corruption. In a data word corruption attack, the attacker attempts to insert fake data words during a benign transmission and have them processed by the receiving device. If

the attacker has an intimate understanding of the purpose for each RT, data word corruption can be used to manipulate sensors and their display readings. The attack exploits the time gap between a `transmit` command and the `status` word response. The attacker attempts to insert a fake `status:OK` followed by manipulated `data` words. If successful, the targeted RT will receive and process whatever data the attacker wants, and the system will be unaware of the intrusion.

As an example, consider a benign data transfer protocol between two devices, RT0 and RT1. The BC initiates the transfer by sending `command:receive (address1)` and `command:transmit (address0)`. The attacker inserts a false `status:OK` before RT0 can respond normally. RT0 will enter `idle` mode, and the attacker is free to send the manipulated data words. RT1 will process them normally and respond with its own `status:OK` word. The BC will assume the transfer was successful and continue normal operations.

6 INTRUSION DETECTION SYSTEM

Aside from raw performance, one major factor that defines the effectiveness of an IDS for MIL-STD-1553 is runtime. Many avionics platforms operate with limited computing power and every acceptable model must abide by strict resource constraints. Although RNNs are well-suited to the task of sequential anomaly detection [14], they use significantly more resources per data point compared to other network architectures. This problem is well-known and methods exist to reduce the computational cost of RNNs such as the skipRNN and leapLSTM [2, 12].

In the case of a skipRNN, instead of simply computing the LSTM or gated recurrent unit (GRU) output an update is applied at each timestamp t using the following formula:

$$s_t = u_t \cdot S(s_{t-1}, x_t) + (1 - u_t) \cdot s_{t-1}$$

Where s_t is the state, S is the LSTM/GRU output function, x_t is the data, and u_t is the binary update gate. Although the state updates are skipped, the runtime of this model actually increases due to the additional calculations needed to evaluate this equation. The original LSTM/GRU output must be calculated (even if not used) to make the network differentiable for training.

In the case of leapLSTM, data points are skipped rather than state updates. The output of the LSTMs at each timestamp t are calculated using the following formula:

$$h_t = \begin{cases} S(h_{t-1}, x_t) & \text{if } d_t = 0, \\ h_{t-1} & \text{if } d_t = 1 \end{cases}$$

Where h_t is the hidden state, S is the LSTM output function, x_t is the data, and d_t is the binary update gate. Although a leapLSTM allows for true skipping of the computational cost of processing data points, the method for calculating d_t is very expensive. For each sequence the output of a reversed RNN, LSTM_r and three-layer CNN are passed through a two layer multi-layer perceptron (MLP) using the following formulae:

$$s_t = \text{ReLU}(W_1[x_t; \mathbf{f}_{follow}(t); \mathbf{f}_{precede}(t)] + b_1)$$
$$\pi_t = \text{softmax}(W_2 \cdot s_t + b_2)$$

Where s_t is the MLP node output, W_i are the node weights, b_i is the bias, and π_t is the probability of setting d_t to 0 or 1. $\mathbf{f}(t)$ is computed according to the following equation applied to the data preceding and following t.

$$\mathbf{f}(t) = \begin{cases} [\text{LSTM}_r(t+1); \text{CNN}(t+1)] & \text{if } t < T, \\ \mathbf{h}_{end} & \text{if } t = T \end{cases}$$

Where T are the total number of timestamps and h_{end} is the features when the sequence reaches the end.

6.1 True-Skip Mechanism

To address the critical resource issues we propose a network that consists of three components: skip mechanism, recurrent network, and word classification. A diagram of the system architecture can be found in **Figure 6**.

The network takes as input a sequence of configurable but fixed length containing processed words from the OD1NF1ST simulator. The sequence is embedded using a layer with 8 nodes which is sufficient to include the 256 words in the MIL-STD-1553 dictionary since each each integer is between 0-255 ($2^8 = 256$).

The sequences are then passed through a convolutional layer which attempts to predict the words that are most likely to be malicious. This layer provides a rough guess as to whether messages could be part of an attack scenario. Rapidly identifying possible intrusions is important for MIL-STD-1553 data, since words are transmitted very quickly (typically 140-160 μs between messages). A large amount of data must be analyzed efficiently by the IDS and the runtime of such a simple network is very low. The output at each layer is computed as:

$$C_k^{(l)} = \text{softmax}(\sum_c W_k^{(l),c} \cdot X^{(l-1),c} + B_k^l)$$

Where k is the filter size, l is the layer, c denotes channel number, X is the data, W is the weight, and B is the bias. In our network, this can be simplified, since we use a single layer and filter size of 1 to produce a single prediction for each word:

$$C = \text{softmax}(\sum_c W^c \cdot X^c + B)$$

Next, we select the top n words with the highest probability of being intrusions and pass them to the next layer. Since only n words from the whole sequence are passed to the recurrent network, we save major computational resources in this step. Word selection is non-differentiable and thus cannot be trained. To solve this problem, we apply a reinforcement learning algorithm (described in section 6.2) to support backpropogation. REINFORCE trains the network on if its prediction about whether a message is anomalous or not is correct. We only train top-n using binary anomaly labels.

Next, we use two Bi-directional LSTM layers, one with 32 nodes followed by another with 128. This part of the IDS is used as a more exhaustive predictor to produce the most accurate guess possible. The recurrent network is trained using both binary anomaly detection and attack classification. The output of each layer for the recurrent portion of the network is computed as:

$$h_t = o_t \circ \sigma(f_t \circ c_{t-1} + i_t \circ \tilde{c}_t)$$

Where h_t is the hidden state, o_t, f_t, i_t, $\tilde{c}_t$ are the output gate, forget gate, update gate, and cell input activation vector respectively. $\circ$ is the Hadamard product. Each gate's output is calculated as:

$$g_t = \sigma(W_g x_t + U_g h_{t-1} + b_g)$$

Where W_g, U_g, and b_g are the weight matricies and bias vector paramters learned during training. x_t is the data and h_{t-1} is the hidden state.

The output of the recurrent layers are passed to a single fully-connected layer with ReLU activation followed by two output layers. One layer produces binary anomaly detection using Sigmoid activation and the other guesses the precise attack class with Softmax activation.

6.2 REINFORCE for Policy Optimization

To make the model parameters trainable we apply a REINFORCE algorithm to the top-n selection based on the convolutional layer output. The objective function of our network is based on the sum of loss in three different categories:

$$\mathcal{L}_{\text{OD1N}} = \mathcal{L}_{\text{ANOM}} + \mathcal{L}_{\text{ATTK}} + \mathcal{L}_{\text{TOPN}}$$

Where $\mathcal{L}_{\text{ANOM}}$ is from binary anomaly detection, $\mathcal{L}_{\text{ATTK}}$ is from attack classification, and $\mathcal{L}_{\text{TOPN}}$ is from the most relevant word selection. To compute the loss for binary anomaly detection and categorical attack classification we use cross-entropy:

$$J(\theta) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log\, \hat{y}_i + (1 - y_i) \cdot \log\, (1 - \hat{y}_i)$$

Where θ are the model parameters, N is the output size, $\hat{y}_i$ is the i-th scalar value in the model output, and y_i is the corresponding target value. To find $\mathcal{L}_{\text{TOPN}}$ we use reinforcement learning and provide a binary reward to the network if the selected words are actually anomalies. The objective function for policy gradients is defined as:

$$J(\theta) = \mathbb{E}[\sum_{t_l=0}^{T_l-1} r_{t_l+1}]$$

Where θ is the model parameters, t_l and T_l are the reinforcement learning current and terminal time respectively. r_{t+1} is the reward for performing an action: a_t in a state: s_t based on the reward function: R. So, $r_{t+1} = R(a_t, s_t)$. We optimize the policy by taking the gradient ascent using the partial derivative with respect to θ. Thus, the policy gradient function is:

$$J(\theta) = \nabla_\theta \sum_{t_l=0}^{T_l-1} \nabla_\theta \log \pi_\theta(a_{t_l}|s_{t_l}) G_t$$

Where θ are the model parameters to our policy algorithm, π_θ. ∇_θ represents the gradient of loss, a_{t_l} and s_{t_l} represent the action and state, and G_t is the discounted cumulative reward. Although each of the words is associated with a time, the probability of each one being malicious is calculated in parallel. Thus, there is only a single timestamp in the reinforcement algorithm. Additionally, by assuming that a correct anomaly prediction will yield a reward of 1 and 0 vice versa, the objective function can be simplified to:

$$J(\theta) = \begin{cases} \nabla_\theta \log \pi_\theta(a|s) \times 1, & \text{if anomaly prediction is correct,} \\ \nabla_\theta \log \pi_\theta(a|s) \times 0, & \text{otherwise} \end{cases}$$

The combined loss equation allows the network to train on its predictions about which words are relevant in each sequence and which sequences contain malicious activity.

7 EXPERIMENT

To evaluate OD1NF1ST, we generated a dataset using the simulation package and compare our IDS to other state-of-the-art methods. We also perform cross-validation on the attack types to verify our IDS' ability to independently detect each different type of intrusion.

Table 4. Summary of the Attacks in our Datasets

Attack Type	Instances (Cruise)	Instances (Takeoff)
Random Word Generation (RT)	7900	1398
Random Word Generation (Bus)	40	47
Status Word Manipulation (Transmit)	5784	298
Status Word Manipulation (Receive)	2765	948
Man in the Middle	416	122
Command Invalidation	17	26
TX Shutdown	57	46
Desynchronization	168	59
Data Trashing	616	43
Data Word Corruption	170	53
Total Malicious	17933	3040
Total Benign	1662752	256544
Presence	1.08%	1.18%

7.1 Data Processing and Environment

To generate data for the network we followed the pipeline as described above in Section 4. First, using MSFS, two flight paths were established. The plane used for these trips was a Cessna Citation CJ4. The first was from (Anonymous) to (Anonymous) and takes approximately 8:30h to complete. The second was from (Anonymous) to (Anonymous) and takes approximately 1:30h to complete. The simulator was run twice to obtain two unique CSV files containing detailed flight information for the whole journey.

Next, the CSV files were processed through the OD1NF1ST simulator to generate two sets of JSON files. The first, labelled cruise, is larger and contains attacks while the planes were in-flight. The second, labelled takeoff, is shorter and attacks were attempted during takeoff and landing. The communication patterns in both datasets were created using the same predetermined cyclic pattern based on similar protocols in use by military aircraft.

In the cruise dataset there are a total of 1662752 words with a attack presence of 1.08%. In the takeoff dataset there are a total of 256544 words with an attack presence of 1.18%. A detailed summary describing the presence of each type of attack can be found in **Table 4**. We included a similar number of each intrusion in the dataset, however the number of words required to execute each type varies significantly. Thus, there is significant deviation in the number of anomalies associated with each attack type.

7.2 Evaluation Methods and Results

We implemented a cross-validation approach for the out-of-sample evaluation of our solution and used a 10% training 90% validation split. Using a small proportion of training data avoids overfitting the network. There was no guarantee of a specific number of intrusions in either set to better approximate real world scenarios, where they would not be guaranteed or at regular intervals.

Our proposed model is compared with the following networks. (i),(ii),(iii) standard RNN, LSTM, and GRU (iv),(v) skipLSTM and skipGRU [2] (vi) leapLSTM [12] (vii) Naive Bayes [11] (viii) Markov Chain [22] (ix) Histogram Comparison [8] For network (i) we use two Recurrent layers each with 128 nodes. Networks (ii) and (iii) contain a single layer with 256 nodes. Networks (iv), (v), and (vi) all contain one layer with 128 nodes.

Table 5. Performance and Runtime Comparison of OD1NF1ST with other IDS' on Takeoff and Cruise Datasets

		Evaluated Deep Learning Solutions						Existing IDS Solutions for MIL-1553			
Solution		RNN	LSTM	GRU	skip-LSTM	skip-GRU	leap-LSTM	He [11]	Genereux [8]	Stan [22]	OD1NF1ST
Dataset 1 (Cruise)	Precision	0.9824	0.9822	0.9182	0.9676	0.9429	0.8087	0.7089	0.9837	**0.9896**	0.9530
	Recall	0.6500	0.6564	0.7530	0.6228	0.6692	0.3786	**0.9958**	0.5132	0.3779	0.7579
	F1	0.7824	0.7869	0.8274	0.7578	0.7828	0.5158	0.7925	0.6707	0.5469	**0.8443**
	AUC	0.8944	0.9568	0.9622	0.8988	0.9613	0.8574	0.9549	0.5817	0.6303	**0.9648**
	SCA	0.9349	0.9347	0.9381	0.9330	0.9366	0.9126	-	-	-	**0.9476**
	Runtime	500	399	385	1382	1388	1666	405	555	504	379
	Proportion	53.78%	67.40%	69.85%	19.46%	19.37%	16.14%	66.40%	48.45%	59.10%	**71.11%**
Dataset 2 (Takeoff)	Precision	0.6977	0.9821	0.9437	0.8108	0.9592	0.9844	**0.9982**	0.9741	0.9809	0.9322
	Recall	0.3626	0.7185	0.7788	0.6286	0.7166	0.7013	0.7830	0.4830	0.3919	**0.8018**
	F1	0.4772	0.8299	0.8533	0.7412	0.8204	0.8191	0.8605	0.6388	0.5587	**0.8621**
	AUC	0.7489	0.9557	0.9526	0.9361	0.9508	0.9518	0.7907	0.5304	0.5393	**0.9590**
	SCA	0.8696	0.9305	0.9307	0.9081	0.9296	0.9305	-	-	-	**0.9398**
	Runtime	67.8	57.8	56.6	178.1	172.4	263.7	70.3	84.3	65.7	56.0
	Proportion	57.43%	67.37%	68.79%	21.86%	22.59%	14.77%	55.39%	49.27%	59.27%	**69.53%**

To evaluate each models effectiveness we use seven metrics: (i) Precision (ii) Recall (iii) F1 (iv) Area Under ROC Curve (AUC) (v) Sparse Categorical Accuracy (SCA) (vi) Runtime. (vii) Proportion of messages analyzed. These types of intrusion detection systems provide no overhead to the operation of the communication bus, so we evaluate the efficiency of each method based on what proportion of messages could be classified during real-time operation of MIL-STD-1553. This percentage is based on the estimated inter-message time gap [3]. The three existing MIL-STD-1553 IDS' [11, 18, 22] evaluated for comparison do not perform attack classification, so no SCA metric is reported.

When evaluating networks on MIL-STD-1553 data, both F1 score and AUC must be considered for binary anomaly detection performance. Due to the large number of benign messages compared to malicious, a network that simply guesses benign every time can achieve F1 scores >0.98 [21]. In the case of random guessing, AUC will reveal the actual performance of the network. It is also important to consider the model's performance relative to runtime. A network that achieves a slightly higher accuracy over a longer time is not as applicable to MIL-STD-1553.

We compare the performance metrics as applied to the validation dataset for each network on both the long and short flight datasets in **Table 5**. It showed that for the cruise dataset, OD1NF1ST displayed excellent precision (0.95) that was comparable to other IDS' (0.91-0.98), with the exception of Leap-LSTM that was moderate (0.8087), and naïve Bayes, which was poor (0.7089). Recall was overall lower and much more variable, however GRU and OD1NF1ST outperformed the other IDS' (0.753 and 0.7579, respectively), while OD1NF1ST displayed the highest F1, AUC and SCA scores (0.84, 0.96 and 0.95, respectively). OD1NF1ST also had the lowest per-message classification time and would be able to analyze 71% of real-time data. On the Takeoff dataset, OD1NF1ST displayed similar performance with top F1, AUC and SCA scores (0.8621, 0.959, and 0.94, respectively).

Most of the networks achieved high precision but struggled with recall in both scenarios, which may reflect the sparse intrusions present in the dataset. Furthermore, existing recurrent time-saving mechanisms did not translate well to a dataset with very few anomalous messages which resulted in poor performance and runtime on the skip and leap networks.

Of the comparison models, traditional RNN performed adequately in the Cruise dataset, albeit with the poorest F1 score. However, RNN performance markedly declined on the smaller Takeoff dataset, suggesting that pre-trained weights should be passed to all networks in a real world scenario. If the model is trained on almost purely benign data it will become too sensitive and detect

Table 6. Attack Cross-Validation using OD1NF1ST IDS

Metric	Precision	Recall	F1	AUC
Random Word Generation (RT)	0.9737	0.9038	0.9374	0.9786
Random Word Generation (Bus)	0.9752	0.7979	0.8777	0.9507
Status Word Manipulation (TR)	0.9027	0.6896	0.7818	0.9639
Status Word Manipulation (REC)	0.8698	0.7873	0.8265	0.9789
Man in the Middle	0.9853	0.7417	0.8463	0.9185
Command Invalidation	0.9833	0.4103	0.5790	0.8639
TX Shutdown	0.9508	0.4142	0.5771	0.9041
Desynchronization	0.6666	0.7059	0.6587	0.8565
Data Trashing	0.9683	0.6025	0.7428	0.9481
Data Word Corruption	0.9441	0.8787	0.9102	0.9676

attacks when none are present. This is also confirmed by Onodueze et al. [21] who found that a network only trained using benign messages was incapable of correctly identifying anomalies.

Finally, while OD1NF1ST outperformed the other neural networks, they generally achieved, with the exception of the RNN, acceptable performance on both datasets. This demonstrates that even when faced with a limited number of malicious messages, machine learning is an effective strategy for MIL-STD-1553-based intrusion detection.

7.3 Attack Cross-Validation

As another method to evaluate the effectiveness of our IDS we partitioned the dataset into individual attack types and compared the performance. This allowed us to test if OD1NF1ST can identify the anomalous behaviors associated with each individual attack type. **Table 6** shows the results of attack cross-validation on the cruise dataset. For training and evaluating the model, we only consider binary anomaly detection metrics since each dataset only contains one type of attack.

The cross-validation table shows that our network can independently detect each of the different proposed intrusions. The most difficult attacks to detect overall were command invalidation, TX shutdown, desynchronization, and data trashing. TX Shutdown and command invalidation are both attacks with very few examples in the generated dataset (<0.01%), which may result in poor training and validation capabilities.

TX shutdown and desynchronization are challenging to identify because a malicious party sending command words to the targeted RT is no different from the genuine BC transmitting the same message. Data trashing and command invalidation both share a similar attack pattern whereby the attacker inserts a malicious command word to alter the state of a device on the network. These are difficult to detect because no collisions occur and messages are sent in the same way during a regular transmission. The main difference between these attacks and benign protocols are the way words are interpreted by devices, not the messages they send. The minimal differences between benign and malicious words creates a challenge for our IDS to successfully detect them.

8 CONCLUSION

In conclusion, we present a software system for simulation of the MIL-STD-1553 platform which can be integrated in real-time with an IDS to create a web-based attack-defence game. The OD1NF1ST simulator can collect high resolution data over significantly longer recording periods compared to any existing method. Furthermore, it allows for future-proof integration of advanced cyber-attack scenarios on our MIL-STD-1553 simulator. Our IDS can detect anomalies and classify attacks with greater performance compared to other state-of-the-art methods with limited computational resources while achieving comparable performance to more expensive approaches.

The feasibility and capability of the attack scenarios are proven by their implementation in this simulation system as well as the theoretical basis for their existence as described by He *et al.* [10]. The data from the simulation environment can be pre-processed in different ways than described in Section 7 to provide even greater detail to the IDS. The generalizability of the system is provided by MSFS, which can simulate a custom flight scenario anytime, anywhere, and in a multitude of weather conditions. Unseen attacks can also be implemented on-demand using the framework to evaluate future discoveries of MIL-STD-1553 vulnerabilities.

There are two possible directions for future work in this research area: improvement in the simulation system and refinement of the intrusion detection system. The simulation system can be further enhanced to generate new datasets for testing and training different models. The IDS could be further improved and modified to be more generalizable to new attack scenarios as they are developed or new datasets generated from different flight scenarios.

REFERENCES

[1] M. Al-Subaie and M. Zulkernine. 2007. The Power of Temporal Pattern Processing in Anomaly Intrusion Detection. In *2007 IEEE International Conference on Communications*. 1391–1398. https://doi.org/10.1109/ICC.2007.234

[2] Victor Campos, Brendan Jou, Xavier Giró-i-Nieto, Jordi Torres, and Shih-Fu Chang. 2017. Skip RNN: Learning to Skip State Updates in Recurrent Neural Networks. *CoRR* abs/1708.06834 (2017). arXiv:1708.06834 http://arxiv.org/abs/1708.06834

[3] Condor Engineering 2000. *MIL-STD-1553 Tutorial.* Condor Engineering, Santa Barbara, CA, 93101.

[4] D De Santo, CS Malavenda, SP Romano, and C Vecchio. 2021. Exploiting the MIL-STD-1553 avionic data bus with an active cyber device. *Computers & Security* 100 (2021), 102097.

[5] Department of Defense 1978. *Aircraft Internal Time Division Multiplex Data Bus.* Department of Defense.

[6] Leroy Earhart. 1982. MIL-STD-1553: Testing and test equipment. In *Proc. Papers of the 2nd AFSC Avionics Standardization Conference*, Vol. 1. 349–365.

[7] Marwa A. Elsayed and Mohammad Zulkernine. 2020. PredictDeep: Security Analytics as a Service for Anomaly Detection and Prediction. *IEEE Access* 8 (2020), 45184–45197. https://doi.org/10.1109/ACCESS.2020.2977325

[8] Sebastien JJ Genereux, Alvin KH Lai, Craig O Fowles, Vincent R Roberge, Guillaume PM Vigeant, and Jeremy R Paquet. 2019. Maidens: Mil-std-1553 anomaly-based intrusion detection system using time-based histogram comparison. *IEEE Trans. Aerospace Electron. Systems* 56, 1 (2019), 276–284.

[9] Pierre-Francois Gimenez, Jonathan Roux, Eric Alata, Guillaume Auriol, Mohamed Kaaniche, and Vincent Nicomette. 2021. RIDS: Radio Intrusion Detection and Diagnosis System for Wireless Communications in Smart Environment. *ACM Trans. Cyber-Phys. Syst.* 5, 3, Article 24 (April 2021), 1 pages. https://doi.org/10.1145/3441458

[10] Daojing He, Xuru Li, Sammy Chan, Jiahao Gao, and Mohsen Guizani. 2019. Security analysis of a space-based wireless network. *IEEE Network* 33, 1 (2019), 36–43.

[11] Daojing He, Xiaoxia Liu, Jiajia Zheng, Sammy Chan, Sencun Zhu, Weidong Min, and Nadra Guizani. 2020. A lightweight and intelligent intrusion detection system for integrated electronic systems. *IEEE Network* 34, 4 (2020), 173–179.

[12] Ting Huang, Gehui Shen, and Zhi-Hong Deng. 2019. Leap-LSTM: Enhancing Long Short-Term Memory for Text Categorization. *CoRR* abs/1905.11558 (2019). arXiv:1905.11558 http://arxiv.org/abs/1905.11558

[13] Gonçalo Jesus, António Casimiro, and Anabela Oliveira. 2021. Using Machine Learning for Dependable Outlier Detection in Environmental Monitoring Systems. *ACM Trans. Cyber-Phys. Syst.* 5, 3, Article 29 (July 2021), 30 pages. https://doi.org/10.1145/3445812

[14] Tae-Young Kim and Sung-Bae Cho. 2018. Web traffic anomaly detection using C-LSTM neural networks. *Expert Systems with Applications* 106 (2018), 66–76. https://doi.org/10.1016/j.eswa.2018.04.004

[15] Liwei Kuang and Mohammad Zulkernine. 2008. An Anomaly Intrusion Detection Method Using the CSI-KNN Algorithm. In *Proceedings of the 2008 ACM Symposium on Applied Computing* (Fortaleza, Ceara, Brazil) *(SAC '08)*. Association for Computing Machinery, New York, NY, USA, 921–926. https://doi.org/10.1145/1363686.1363897

[16] V Lalitha and S Kathiravan. 2014. A review of manchester, miller, and fm0 encoding techniques. *SmartCR* 4, 6 (2014), 481–490.

[17] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. 2013. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications* 36, 1 (2013), 16–24.

[18] Capt Blaine Losier, Ron Smith, and Vincent Roberge. [n.d.]. DESIGN OF A TIME-BASED INTRUSION DETECTION ALGORITHM FOR THE MIL-STD-1553. ([n. d.]).

[19] Karim Lounis and Mohammad Zulkernine. 2020. Exploiting Race Condition for Wi-Fi Denial of Service Attacks. In *13th International Conference on Security of Information and Networks* (Merkez, Turkey) *(SIN 2020)*. Association for Computing Machinery, New York, NY, USA, Article 2, 8 pages. https://doi.org/10.1145/3433174.3433584

[20] JK Murdock and James R Koenig. 2000. Open systems avionics network to replace MIL-STD-1553. In *19th DASC. 19th Digital Avionics Systems Conference. Proceedings (Cat. No. 00CH37126)*, Vol. 1. IEEE, 4E5–1.

[21] Francis Onodueze and Darsana Josyula. 2020. Anomaly Detection on MIL-STD-1553 Dataset using Machine Learning Algorithms. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 592–598. https://doi.org/10.1109/TrustCom50675.2020.00084

[22] Orly Stan, Yuval Elovici, Asaf Shabtai, Gaby Shugol, Raz Tikochinski, and Shachar Kur. 2017. Protecting military avionics platforms from attacks on mil-std-1553 communication bus. *arXiv preprint arXiv:1707.05032* (2017).

[23] R Ben Truitt, Edward Sanchez, and Michael Garis. 2004. Using open networking standards over MIL-STD-1553 networks. In *Proceedings AUTOTESTCON 2004.* IEEE, 117–123.

[24] Chris Wiegand. 2018. F-35 air vehicle technology overview. In *2018 Aviation Technology, Integration, and Operations Conference*. 3368.

[25] Di Wu, Hanlin Zhu, Yongxin Zhu, Victor Chang, Cong He, Ching-Hsien Hsu, Hui Wang, Songlin Feng, Li Tian, and Zunkai Huang. 2020. Anomaly Detection Based on RBM-LSTM Neural Network for CPS in Advanced Driver Assistance System. *ACM Trans. Cyber-Phys. Syst.* 4, 3, Article 27 (May 2020), 17 pages. https://doi.org/10.1145/3377408

[26] Ran Yahalom, David Barishev, Alon Steren, Yonatan Nameri, Maxim Roytman, Angel Porgador, and Yuval Elovici. 2019. Datasets of RT spoofing attacks on MIL-STD-1553 communication traffic. *Data in brief* 23 (2019), 103863.