THESIS

TOWARDS CYBERSECURITY COUNTERMEASURES FOR SAE J1708/1587 NETWORK PROTOCOL IN HEAVY-DUTY VEHICLES

Submitted by

David C. Nnaji

Department of Systems Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2022

Master's Committee:

Advisor: Jeremy Daily

Steve Simske
Sudeep Pasricha
Indrajit Ray

ABSTRACT

TOWARDS CYBERSECURITY COUNTERMEASURES FOR SAE J1708/1587 NETWORK PROTOCOL IN HEAVY-DUTY VEHICLES

Heavy vehicles are crucial to a functioning economy and society since they are responsible for delivering people and goods across the country. These systems rely on various forms of in-vehicle communication between electronic control units (ECU) for reliable operation. In recent years, numerous vulnerabilities inherent to unauthenticated in-vehicle communication have been identified in academia, industry, sponsored events, and real-world attacks. Current defensive cybersecurity research is primarily aimed at securing the controller area network (CAN) and other conventional systems. However, little to no defensive research has been conducted on legacy systems, and only recently have state-of-the-art attacks been identified in public disclosures or discussed in published works. Despite the age of the technology, the associated vulnerabilities from legacy networks are likely to persist for many years due to long equipment service life, cost-reluctance from fleets, and powerline bridge standardization. If system-wide security is desired by the industry, proportional research in this field is warranted.

In this thesis, I examine the application of simple signature-based and anomaly-based intrusion detection on legacy serial data communication between ECUs in heavy-duty (HD) applications defined in SAE J1708 and J1587 building on previously published work. This is accomplished through the design and development of a prototype network gateway tailored to the requirements defined within the two protocols. Additionally, this thesis contributes the embedded software utility developed for the prototype gateway for open use and validates its functionality through robust unit testing. Ultimately, the intrusion detection system is deployed, tested, and evaluated on a retrofitted dual air brake system simulator (DABSS)

managed by Dr. Jeremy Daily at the Powerhouse Energy Institute. An assessment of the effectiveness of the mitigation against four attack scenarios followed by recommendations for improvements and future work are provided in the final chapters.

ACKNOWLEDGEMENTS

First, I would like to thank Dr. Jeremy Daily for welcoming me to his research team as an undergraduate in 2018, for granting me the opportunity to explore the field of automotive engineering and cybersecurity, and for his utmost patience and support during the writing of this thesis. Of course, thank you most of all for leading by example as both an academic and professional engineer. I'd also like to thank the members of my distinguished committee for their thoughtful considerations, time, and guidance.

Additionally, I'd like to thank the faculty and staff of the Department of Systems Engineering at Colorado State University for their instruction, guidance, facilities, resources, and academic support. I'd also like to thank the National Motor Freight Traffic Association (NMFTA) and Barber-Nichols for supporting my education through their scholarship programs.

Finally, to my friends Aaron and Sam - it was fun hanging in Fort Collins with you two!

DEDICATION

This thesis is dedicated to the aspiring and persevering engineering student.

Keep learning. Keep moving forward.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Motivation

What does the future of heavy vehicles look like? To answer this question, researchers and industry stakeholders assess existing problems and needs, explore the application of new technologies, generate conceptual solutions, and subsequently design, build and test their ideas in hopes that their solution is satisfactory in addressing the original need. Among these is the need to protect these systems from unintentional information disclosure, theft, disruption, or damage.

As heavy vehicles continually improve and gain complexity with each innovation, it becomes increasingly difficult to assess the system's security posture and defend against misuse. But why is this important?

Semi-trucks are a component of "critical infrastructure" – a term that the government uses to describe essential assets for a functioning society and economy. They transport goods across the country, equipment for businesses, and resources that help power cities keeping many aspects of day-to-day life convenient. While a single encompassing attack on heavy vehicles is not a present danger, the risk of incidents involving small fleets or individuals is a concern among industry professionals—the threat? A cyber-attack facilitated by vulnerabilities inherent to exposed onboard and wireless network communication.

For reasons discussed in the subsequent section, heavy vehicles are particularly at risk because of publicly available standards, the inherent trust of in-vehicle electronic control units (ECU), and an expanding attack surface from innovations in the field. Many vulnerabilities have already been exposed and are well understood. For example, one team of researchers used empirical experiments to demonstrate that ECUs on the controller area network (CAN) were susceptible to targeted denial-of-service (DOS) attacks if access to the network was

obtained [2]. Such an attack would have profound consequences on vehicle dynamics during operation, the worst of which could be fatal to the driver or the public.

Conversely, mitigations and defense concepts are also actively under investigation. For example, a team of researchers at the University of Michigan have developed a clock-based intrusion detection system (IDS) called CIDS to aid in fingerprinting compromised modules on CAN [3]. More recently, research aimed at practical and deployable mitigations that account for compatibility constraints imposed by legacy CAN and message protocol definitions (i.e., SAE J1939) has gained traction [4]. Yet, despite the wealth of publications available, suggested mitigation concepts for modules connected by legacy networks (i.e., SAE J1708, J1587) are rarely mentioned or thoroughly discussed in published defense research.

In a 2018 publication by the National Highway Traffic Safety Administration (NHTSA), the authors found no security discussions or information regarding J1708/1587 network security disclosed by representative industry partners that were suitable to include in their national report on heavy-duty (HD) cybersecurity [5]. This lack of research is the case even when the mitigation concepts explored in the mainstream concerning J1939 networks could be extended or adapted to legacy protocols.

Though the NHTSA did not conclude on a specific cause, stakeholders may view the vulnerabilities associated with J1708/1587 networks as less probable or impactful. Some may also point out the infrequent amount of time trailers are actually in service. By some estimates, trailers are detached from trucks up to 75% of the time [6]. Technical concerns also exist around the implementation details and busload overhead on a relatively slower network.

Despite these valid objections, the risks of a cyberattack intrinsic to existing legacy networks are likely to persist due to long equipment service life, cost-reluctance from fleets, and powerline bridge standardization. Thus, as long as system-wide security in HD vehicles is sought, J1708/1587 network security will remain a worthwhile research segment.

Given the lack of research, this thesis aims to contribute to the body of knowledge regarding J1708/1587 network security by exploring the application of simple intrusion detection. This is accomplished by applying concepts proposed by conventional in-vehicle defense research and implementing this on custom electronic hardware. The design and development of a J1708/1587 gateway are described in the subsequent chapters. Additionally, this thesis contributes a software utility developed for the gateway for other interested researchers to apply on low-cost embedded platforms in future investigations. Finally, various unit tests were performed to verify the gateway hardware and software against any logical or functional inconsistency, culminating in integrated tests on a retrofitted heavy vehicle simulator.

Ultimately, the intrusion detection system is deployed, tested, and evaluated on the simulator, and an assessment of the mitigation concept is conducted. Finally, in light of conclusions drawn in the last chapter, recommendations for improvements and future work are provided. Altogether, this defense concept builds upon previously published work by the author and researchers at Colorado State University (CSU) for applications in the J1708/1587 domain. The ultimate goal of this work is to generate the tools and the preliminary research for future investigations of deployable network defenses for serial communication defined by SAE J1708/1587.

1.2 Background

1.2.1 Vehicle Classification and Architecture

Before a discussion of relevant work can continue, it is essential to clarify what a heavy-duty vehicle is and how it fundamentally differs from other vehicle types. Although many organizations have generated classification systems, the Federal Highway Administration (FHWA) provides a set of 13 categories that serve as the basis for most other classification systems in the United States [7]. A visual representation of the first ten classes is provided in Figure 1.1 courtesy of the FHWA [8].

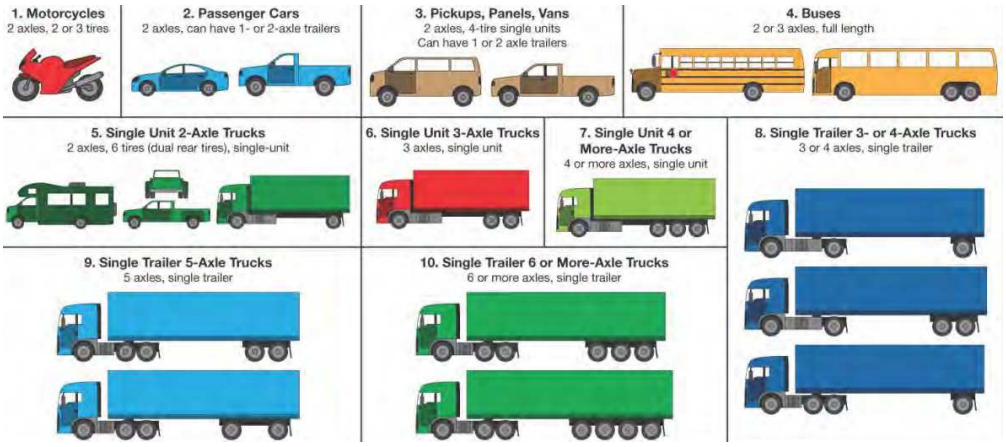


Figure 1.1: Common vehicle types for each FHWA class

For the sake of discussion, it is often easier to compare vehicles based on their gross vehicle weight rating (GVWR). As shown in Table 1.1, weight-based categorization consists of classes 1 through 8. These classes are further grouped into light-duty (LD), medium-duty (MD), and heavy-duty categories. Despite some overlap with large box trucks, the HD class is distinguished by the inclusion of triple-axle vehicles with single trailers (class 8).

Table 1.1: FHWA Vehicle Classifications, Gross Vehicle Weight Rating (GVWR) Ranges and Respective Categories

Class	GVWR Range	GVWR Category
Class 1 - 2	≤ 10,000 lbs.	Light-Duty Vehicles
Class 3 - 6	10,001 – 26,000 lbs.	Medium-Duty Vehicles
Class 7 - 8	26,000+ lbs.	Heavy-Duty Vehicles

Regardless of the classification, practically all modern vehicles contain numerous embedded computing systems known as ECUs that control various subsystems and report information on a shared network. The most critical ECUs control steering, braking, powertrain, and other subsystems impacting vehicle dynamics. ECUs are also crucial for indicating when maintenance is needed, reporting diagnostic information, and performing safety-critical functions in response to external events.

Generally speaking, sensors aboard the vehicle provide real-time signals to their respective ECUs. Once these signals are processed, the ECU decides to perform some action (i.e., actuating a motor, providing voltage to a signal light, or reporting information to another ECU). Collectively, these ECUs exchange information on a shared bus resembling the generic form shown in Figure 1.2.

Figure 1.2: A generic vehicle network

ECUs are also found on box trailers (and the like) to control brake lamps and any electronic air braking systems (ABS). Multiple cables are used to intermittently link the primary air supply and battery power from the truck to the trailer as shown in Figures 1.3b through d. The glad-hand connectors in blue and red are used to connect the supply and secondary air. The J560 connector in green provides system power. SAE J560 defines the physical connecting harness to extend power and signals across the system [9]. In most HD applications in the US, this power line also carries serial data between the tractor and any trailers equipped with bridging hardware. The pin definitions for this cable are provided in Figure 1.3a.

1.2.2 CAN and SAE J1939

Networking protocols, often published in larger documents called standards, establish the governing rules for communication between ECUs. Although many standards exist for MD/HD vehicles across market regions, the two most relevant standards for operation in North America are CAN (defined in ISO 11898) and SAE J1939.

SAE J1939 was created by the Society of Automotive Engineers (SAE) and is a collection of documents based on the ISO/OSI protocol stack that defines modern in-vehicle commu-

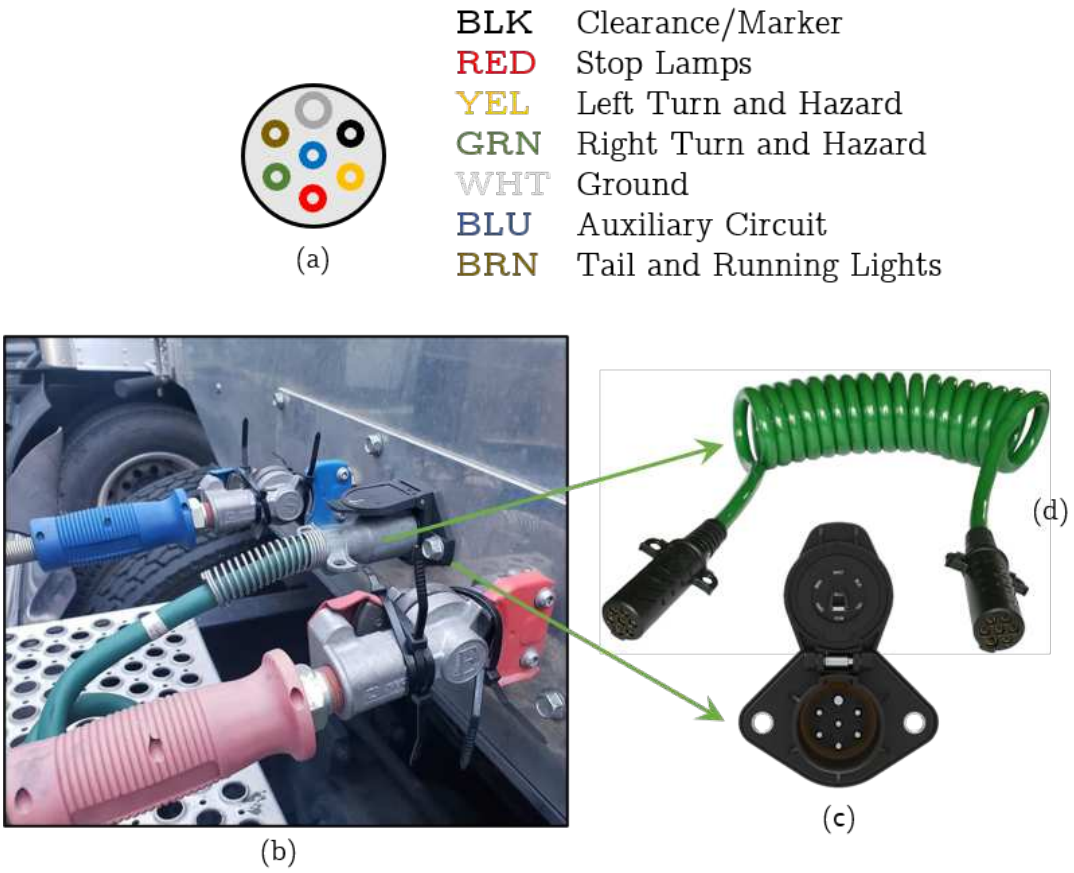


Figure 1.3: (a) Pin definitions for J560 connections, (b) Auxiliary linkages between truck and trailer (truck-side). The glad-hand connectors in blue and red are used to connect the primary air supply. The J560 connector in green provides system power. (c) The interior pins of a J560 socket. (d) A J560 cable plug.

nication for HD applications. Table 1.2 summarizes the relationship between the documents and the model. In sum, SAE J1939 specifies requirements ranging from how the modules should be physically connected (physical layer) to how an application should process and interpret messages. In theory, messages are "packaged" down the protocol stack from one module and are "unpacked" as they travel up the stack on another device to be interpreted and handled by its application.

Specifically, the SAE J1939 physical layer and message structure are adopted from ISO 11898 by the International Standards Organization (ISO). In the CAN 2.0b specification, messages consist of a 29-bit header called the CAN-ID and a maximum payload of 8-bytes. In J1939, the CAN-ID contains the source address, parameter group number, and priority.

Table 1.2: The SAE J1939 relationship to the ISO/OSI model.

OSI Reference Model	Vehicle Standards
Application	J1939/71 J1939/73
Presentation	-
Session	-
Transport	J1939/21
Network	J1939/31
Data Link	J1939/21 ISO 11898
Physical	J1939/11 J1939/13 J1939/14 J1939/15 ISO 11898

A summary of notable features between the two protocols is provided in Table 1.3 to avoid needless descriptions of the many documents and semantics that make up the standards.

1.2.3 SAE J1708 and J1587

Historically, SAE J1939 was not the first diagnostic protocol with widespread use in HD applications. SAE J1708 and SAE J1587 (hereafter collectively referred to as J1708/1587) have existed since 1986 and retain use today. However, today it is mainly used to facilitate communication between controllers on a trailer and to bridge communication between the tractor and trailer networks. Table 1.5 summarizes the relationship between the documents and the ISO/OSI model.

Network Parameters

Unlike its successor, the J1708/1587 protocol is slow by current standards and is physically composed of a twisted pair of 18 gauge wires up to 40 meters in length. Like CAN, messages are broadcast by the transmitter to every connected device on the bus. ECUs on J1708 are responsible for bus-synchronization, bus access verification, error handling, and

Table 1.3: SAE J1939 Feature Summary

Feature	Description
Network Speed	250 or 500 kbit/s
Network Topology	Multi-client serial bus connected via 18-guage twisted wire pair (1 twist/inch) with 120Ω terminating resistors. Based on CAN protocol.
Message Format	29-bit extended identifier and 8-byte data payload. Refer to Table 1.4 for the extended ID format.
Address Claiming	Most addresses are pre-assigned to an ECU depending on its function by SAE 1939. However, an address claiming procedure has also been defined for ECUs with multiple or proprietary functions.
Transmission	All messages sent by an ECU are visible to every node on the network. It is left to the receiving ECU to determine if a message should be handled or ignored.
Transport Protocol	A specific PGN is assigned for transmitting payloads larger than 8-bytes. The payload is segmented by the transmitter and reassembled by the receiving ECU.

Table 1.4: SAE J1939 29-bit CAN ID Format

Priority	Reserved	Data Page	PDU Format	PDU Specific	Source Address
3 bits	1 bit	1 bit	8 bits	8 bits	8 bits

other network stabilizing features since it is undefined in the protocol. Table 1.6 summarizes the notable features of SAE J1708/1587.

Protocol

As shown in Figure 1.4a, a J1708 message consists of a series of 10-bit units called characters which contain a start bit, a single byte of information transmitted least significant bit (LSB) first, and a stop bit. A pause in communication lasting at least the time of a single character denotes the end of a single message.

Messages that appear on the bus contain a header known as the message identifier (MID), data characters, and a checksum character (CHK) totaling no more than 21-characters al-

Table 1.5: The SAE J1708/1587 relationship to the ISO/OSI model.

OSI Reference Model	Vehicle Standards
Application Presentation	J1587
Session Transport Network	- - -
Data Link	J1587 J1708
Physical	J1708 J2497

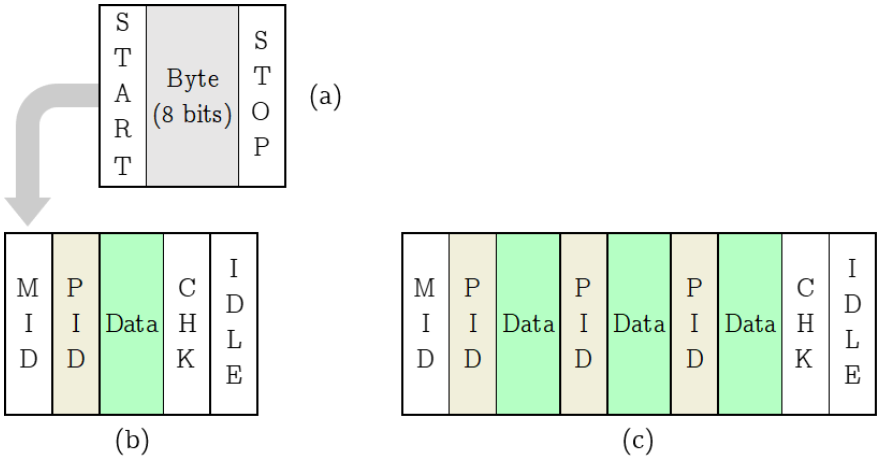


Figure 1.4: J1708/1587 Message Format: (a) Character format, (b) Single parameter J1708 message format, (c) Multi-parameter J1708 message format.

together, as shown in Table 1.7. The MID is always the first character of a message, and its value ranges from 0 to 255. It denotes which ECU transmitted the message. MIDs are allocated to different transmitter categories in SAE J1708 and J1587. For example, MIDs 0 to 7 are reserved for engine messages.

The data characters can contain one or more parameters related to the transmitting system. The current engine speed or oil temperature are examples of common parameters. A parameter is packaged in the structure shown in Figure 1.4b. The first character of every parameter is the parameter identifier (PID) which can range from a value of 0 to 255. The actual encoded data follows the PID and can span multiple characters. Although most PIDs

Table 1.6: SAE J1708/1587 Feature Summary

Feature	Description
Network Speed	9600 bit/s
Network Topology	Multi-client RS485 serial bus connected via 18-gauge twisted wire pair (1 twist/inch) up to 130 feet.
Message Format	1-byte Message Identifier (MID), 19-byte (max) data payload, and 1-byte Checksum (CHK).
Address Claiming	Nearly all MIDs are pre-assigned to an ECU depending on its function. A dynamic address claim exists only for trailer gateway devices.
Transmission	All messages sent by an ECU are visible to every node on the network. It is left to the receiving ECU to determine if a message should be handled or ignored.
Transport Protocol	Connection management PID 197 (0xC5) and its sub-commands are used for segmenting payloads exceeding 21 bytes. Segments are reassembled by the recipient.

Table 1.7: J1587 Message Content

MID	Data Characters	Checksum
1 byte	19 bytes (max)	1 byte

are used to transmit specific data, others facilitate network functionality such as parameter requests, transport protocol, data link escape, and page extension. All PIDs are assigned and defined at length in SAE J1587 [10]. As shown in Figure 1.4c, multiple parameters can be packed in a single message so long as it does not extend the message length beyond 21-characters.

As an example, suppose the message in Figure 1.5 is captured from a J1708 bus. Using the definitions provided in J1587, the first byte indicates the message came from a brake controller. The message also contains four parameters which are reported in various encoded formats. The actual values for these parameters are tabulated in the figure below the message. The final byte shows the checksum value.

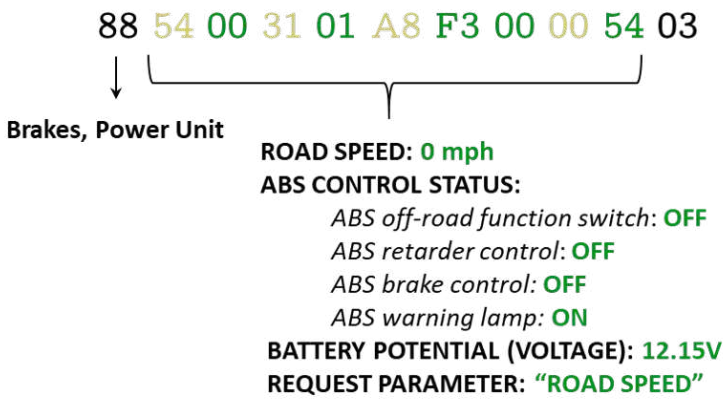


Figure 1.5: Example J1708/1587 Message Decode

1.2.4 SAE J2497

As noted previously, the physical harness connecting vehicle power between the tractor and trailers is also the de-facto means of bidirectional serial communications for systems in the United States. SAE J2497 defines the messaging protocol for Power Line Communications (PLC) for commercial vehicles [11]. In particular, it provides the hardware and software interface requirements, system protocol, messaging format and encoding, signal characteristics, and message definitions for PLC.

Although the underlying messaging protocol is based mainly on SAE J1708/1587, the messages are physically transmitted using radio frequency (RF) "chirps" instead of a differential voltage to represent each bit. The presence or absence of an encoded chirp determines the logical state of the line at any given time. Furthermore, a single message consists of a preamble followed by a data body. The compositions of these sections are summarized in Figure 1.6a, but further definitions are available in the standard. The frequency waveform for the first "INIT" bit is also provided in Figure 1.6b.

1.2.5 Issue 1: A Publicly Available Standard

The advent of onboard diagnostics (OBD) in the United States is attributable primarily to mandates from the California Air Resources Board (CARB) and the Environmental Protection Agency (EPA) regarding engine emission controls during the mid-90s. In the

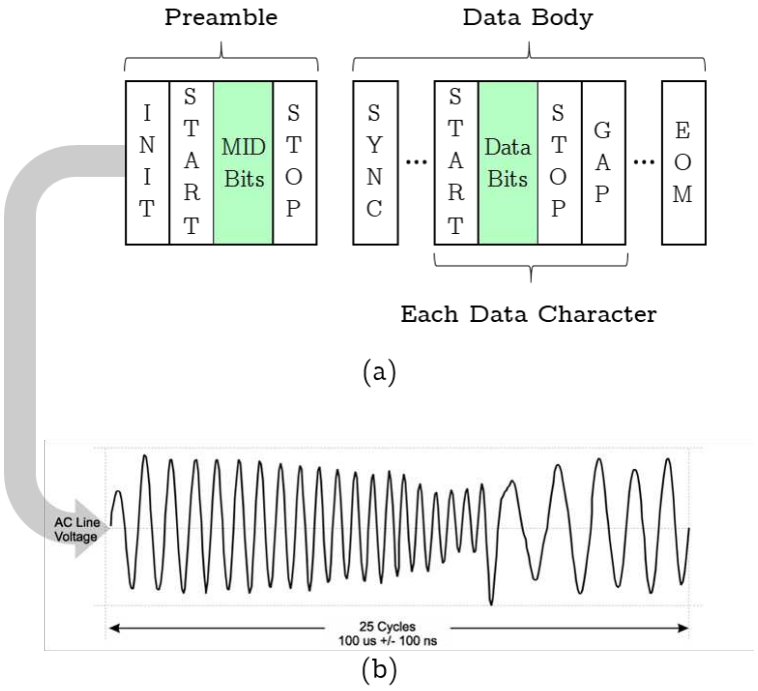


Figure 1.6: (a) J2497 PLC Message Format. (b) SUPERIORΘ2 Frequency Waveform (PLC "Chirp")

first place, CARB and the EPA were established to protect and preserve public safety and the environment through legislation. Legislated OBD was intended to support this goal by requiring specific technologies and features in all road vehicles and non-road mobile machinery so that emissions-related malfunctions could be reliably identified and reported by law enforcement and other regulatory stakeholders [12] [13].

Over time, these early diagnostic technologies matured into robust in-vehicle communication systems as they were applied in other use cases. Ultimately, this led to the development of modern standards such as CAN and SAE J1939. These standards are now used industry-wide and have helped businesses minimize expenses and maximize maintainability and quality across the vehicle life span. The adoption of public standards is particularly true for HD vehicles, where proprietary network solutions are less common relative to smaller vehicle classes.

Although standardization of network communication has resulted in numerous benefits overall, it has also unintentionally made HD vehicles comparatively more susceptible to

cyber-attacks. For example, in an observation from the NHTSA, vehicles that utilize the J1939 protocol, which is a publicly available standard, reduce the reverse engineering effort needed to design vehicle attacks [5]. In contrast, reverse engineering proprietary CAN messages on a passenger automobile is a labor-intensive process.

This availability has allowed researchers to perform a variety of exploits specific to HD vehicles. For example, one can spoof the exact message concerning engine RPM, adjust this parameter, and observe how it affects vehicle behavior [14]. Alternatively, the standards can be used to identify specific messages related to seed-key exchanges between a diagnostic tool and ECU [15]. Others have even reverse-engineered proprietary CAN messages without the use of such standards [16]. Finally, the widespread use of public standards also enables a single vulnerability to potentially scale across vehicles produced by different OEMs of various makes and models [5].

These issues extend to J1708/1587 networks because its governing documents are also public. As in the previous case, an attacker with access to the documents can use the MID and PID definitions to decode standardized messages and infer the information in proprietary ones. Worse, J1708/1587 messages have a narrower range of definitions, theoretically making attacks easier to craft and more deterministic.

Although regulations from CARB and the EPA have a profound influence on why these standards remain publicly available, customer requirements and horizontal integration within HD vehicle development have also encouraged this to a minor extent. Specifically, there is a need for flexibility and interoperability among disparate developers within the supply chain. Unlike passenger and LD vehicle markets, customers of HD vehicles often have the option to configure subsystems to their desired specifications. Both J1708/1587 and J1939 provide the needed flexibility for various supplier components to be integrated reliably by providing generalized message formats available to different stakeholders.

1.2.6 Issue 2: Inherent Trust

Security issues within the protocol also bear repercussions across the network architecture. As noted earlier, control units within the tractor are connected by multiple buses. As shown in Figure 1.7, a typical MD/HD network architecture can have multiple CAN channels, a J1708 bus, and battery power carrying J2497 traffic that connects to a primary diagnostic port. This diagnostic port is defined in SAE J1939 and is typically located under the steering column. A pin-out is provided in Figure 1.8.

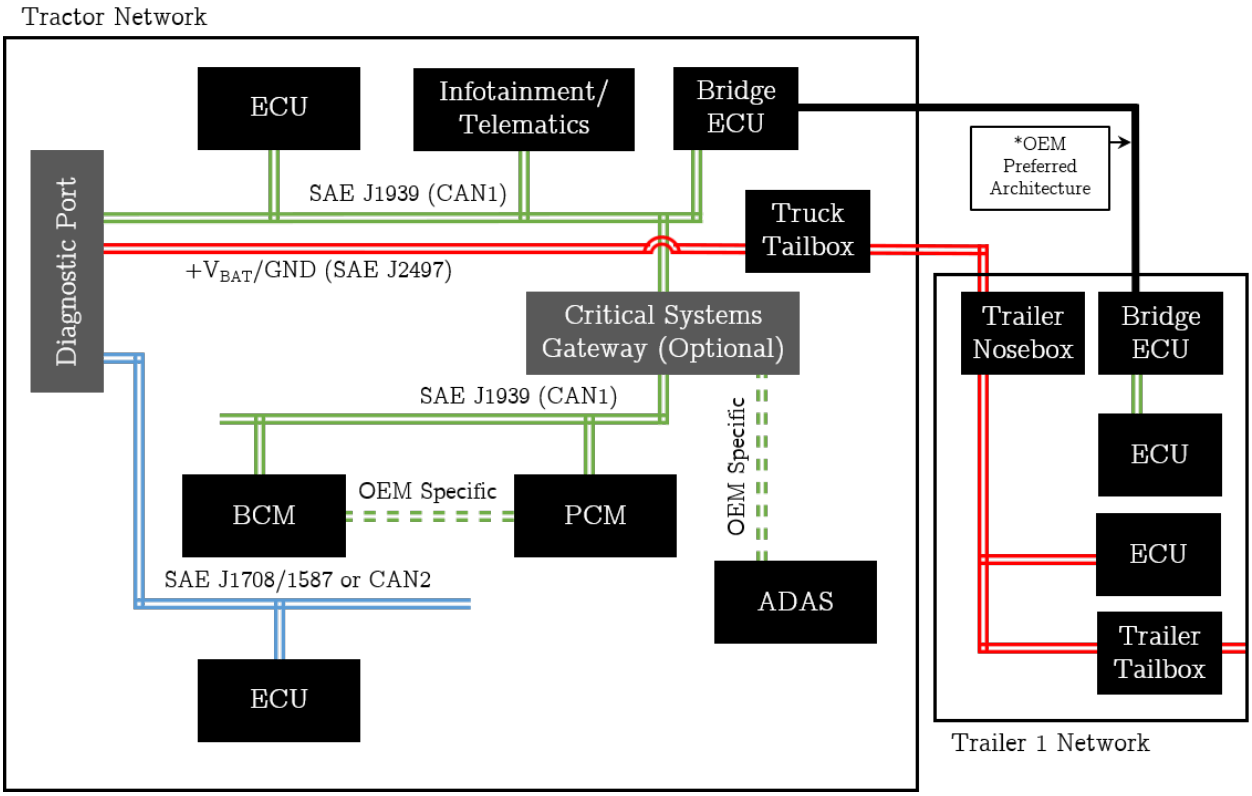


Figure 1.7: A typical MD/HD network architecture

According to research conducted by the NHTSA, basically all MD/HD trucks implement the architecture shown in Figure 1.7. In the network implementations discussed thus far, all messages are received by all the connected nodes. By definition, headers in these messaging schemes are functionally linked to an ECU handling a specific vehicle subsystem. For example, a message with a global source address 11 (0x0B) in J1939 conveys a message that is

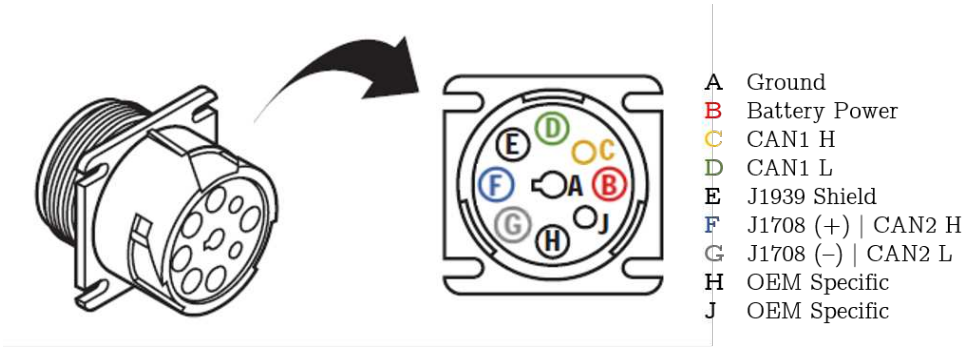


Figure 1.8: 9-pin diagnostic connector diagram

assumed to come from the primary brake system controller [17]. Similarly, MID 137 (0x89) identifies the trailer brake controller on a J1708/1587 or J2497 network [10].

The protocols assume that ECUs will only transmit messages using their pre-defined identifiers, which means all the connected devices trust all messages. Historically, this has been accepted because of the air-gapped nature of vehicle systems, and this design has enabled the reliable messaging needed to improve system safety and performance. However, since individual modules do not contain strong authentication mechanisms, this increases the impact and success rate of any malicious messages sent to the vehicle through a rogue device or compromised ECU.

1.2.7 Issue 3: Increasing Connectivity and Complexity

Although J1708/1587 and the following standards have provided the basis for MD/HD in-vehicle communication, other forms of connectivity are used pervasively. Telematics devices, for example, provide onboard services via GPS and cellular and are used extensively for fleet support and management. It is also not uncommon to uncover short-range wireless communication such as Bluetooth and Wi-fi used by third-party devices for similar services. As fleet management technology matures and longer-term advancements in transportation technology continue, the associated vulnerabilities are likely to grow more complex as well.

Emerging trends in semi-trailer development also suggest a landscape with more connectivity. In a recent report, a leading trailer OEM remarked that the number of ECUs

on trailers would increase in response to forward-looking fleets seeking advantages in the market [6]. In addition, fleets are increasingly interested in trailers with centralized CAN-based architectures, like those in Europe, instead of the current PLC architecture coupled with third-party wireless technologies. Altogether, technical advancements to the tractor and trailer will expand the attack surface and complicate future in-vehicle defense strategies.

However, the emergence of new technology does not necessarily mean older technologies will be immediately replaced. Instead, existing behavior within the industry suggests that new technology is likely to be used *in addition to* rather than *in place of* older technologies—increasing network fragmentation.

As a past report notes, "a fragmented ownership structure in the truck and trailer landscape discourages early adoption of new connectivity technology" [6]. With estimates of 1.2 million active trucking companies (fleets) in the market, each with its preferred OEMs, trailer technology, and connectivity systems, the prospect of rapid change within the market is doubtful.

On a cost basis alone, fleets are understandably reluctant to make cumbersome hardware changes since this requires pausing revenue-generating operations. Moreover, coordinating such changes would also be daunting, given the nomadic nature of trailers during their extensive lifespan.

Within information-sharing communities, fleet managers and carriers have consistently expressed reluctance to revisions within the existing J560 standard, given that there are millions of trucks and trailers currently in operation with this hardware [6]. For these stakeholders, maximizing equipment utilization is a high priority. Given the impediments to changes in hardware, developers in this space will be expected to design with legacy compatibility in mind. Consequently, the application of J1708/1587 could persist further into the future. Therefore, security research on J1708/1587 networks will continue to be warranted.

1.3 Objectives

Given the issues regarding network security discussed in the preceding section, the primary objective is to contribute to HD vehicle systems research in hopes that it will progress the state-of-the-art and improve the overall safety of the public and its critical infrastructure. Additionally, this thesis is written in hopes that it will consolidate information regarding trailer network cybersecurity for outside reference. With this in mind, the following objectives have been defined:

- The first technical objective is to design and develop a single multi-purpose embedded system that can interface with a trailer network via the diagnostic gateway or direct wire tap.
- The second technical objective is designing and developing an open-source software library for a low-cost embedded platform. This software should provide essential scripting utilities such as capturing, interpreting, and transmitting messages on a J1708/1587 bus.
- The third objective is functional hardware and software validation in a practical experiment.
- As previously noted, published mitigation strategies already exist in both academic and industry discourse for SAE J1939. Consequently, another objective is to identify and design a mitigation scheme that enhances J1708/1587 network integrity.
- The last objective is to evaluate the proposed mitigation concept in a practical experiment.

1.4 Approach

As emphasized within the systems engineering program, the soundness of a development approach has an immense effect on the success of a project. Accordingly, a suitable system

concept and development framework was sought with aspects resembling a traditional V-model. Since cybersecurity is a crucial aspect of the thesis objectives, variations of the traditional V-model were researched. SAE J3061 contains guidance on vehicle cybersecurity principles and development processes based on existing practices observed in commercial, government, and engineering settings [18].

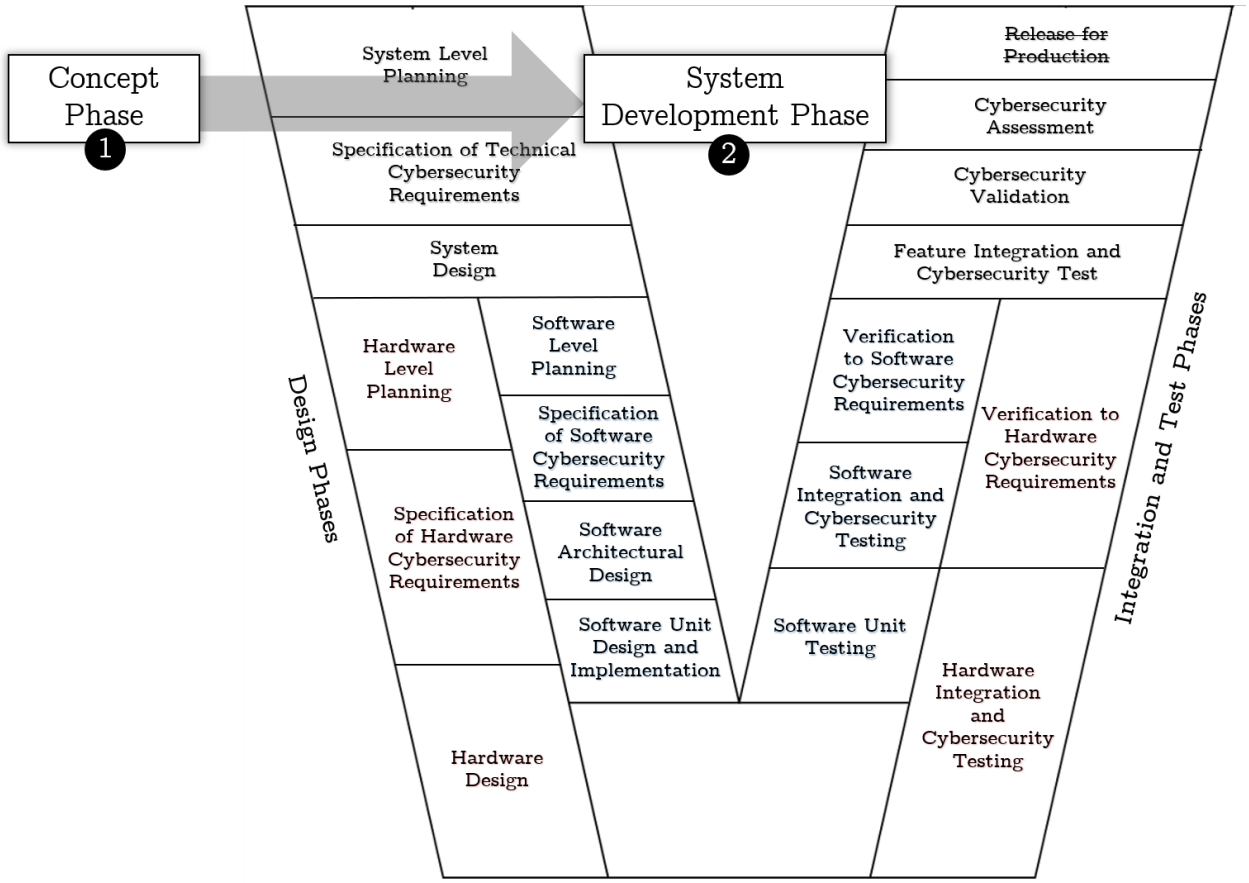


Figure 1.9: Overall cybersecurity framework adapted from SAE J3061 informing the project approach

SAE J3061 also describes a cybersecurity framework that consists of a "conceptual phase", a "systems-level product development" phase, and a "production and operation" phase. Portions of the framework, such as the production and operation phases, management activities,

and others, were deemed beyond this project's scope. However, even after tailoring, much of the framework was still adopted in the approach.

1.5 Contributions

In light of the objectives and having followed a structured approach, the principal contributions of this thesis are:

- The circuit design and documentation of a multi-purpose J1708/1587 network gateway.
- An open-source software library for the Teensy 4.0 development board that provides essential utilities for reading, processing, and transmitting J1708/1587-based messages. This library also contains utilities for configuring the J1708/1587 network gateway and provides example scripts used in testing and experiments.
- Hardware upgrades to the dual air brake system simulator (DABSS) maintained by the SystemsCyber research group at CSU.
- A distributed intrusion detection system concept based on the J1708/1587 network gateway.

1.6 Overview

This thesis is divided into 7 chapters:

- Chapter 1 establishes the motivation for this project and provides the necessary background on vehicle classification and network protocols SAE J1939 and SAE J1708/1587. HD vehicle cybersecurity issues stem from open standardization, inherent trust between connected devices, and increasing complexity. Chapter 1 also presented the objectives of the work, the rationale behind the project approach, and a list of technical contributions.

- Chapter 2 will define the project scope and examine the underlying threats to heavy vehicle systems discussed in the literature, with particular consideration paid to vulnerabilities stemming from the trailer. Subsequently, an underlying research need is identified based on the most elevated threats. Next, a set of cybersecurity goals are defined, and an experimental concept is devised to achieve the goal. Finally, the necessary tasks and features for the concept to work are translated into high-level requirements.

- Chapter 3 will discuss the preliminary stages of system development, including selecting a vehicle test platform and retrofitting efforts.

- Chapter 4 discusses the hardware of the J1708/1587 prototype gateway. This includes the specification of hardware requirements, as well as integration, testing, and verification activities.

- Chapter 5 discusses the software design of the J1708 library. This includes the specification of software requirements and descriptions of the software architecture, functional programming, integrated unit testing, integration, and verification activities.

- Chapter 6 discusses integrating the distributed intrusion detection system on DABSS, verification testing, the threat model, validation test procedures, and the system test results. A cybersecurity assessment of the efficacy of the system concept is presented in the final section.

- Chapter 7 includes thoughtful conclusions in light of the original objectives, identifies project limitations, and provides a parting statement regarding future work.

Chapter 2

Concept Phase

2.1 Feature Definition

This activity aims to limit the scope to specific portions of the vehicle for subsequent analysis. As noted in Chapter 1, a trailer network consists of a 12- or 24-volt battery and ground lines that run the length of the body. A 7-pin cable connects this network segment for other ECUs on the truck to communicate on J2497. In addition, a J1708/1587 network may also exist within the truck. Thus, the J2497 PLC harness, any J1708/1587 harnesses, and any connected ECUs to either network are included in the feature definition.

Network bridges (i.e., J560 cables) and their associated physical interfaces at the truck tailbox, trailer nosebox, and trailer tailboxes are also included. We will also apply the cybersecurity analysis to other conventional interfaces, such as the primary diagnostic port and other non-conventional ports that expose the PLC harness, such as 12-volt accessory outlets within the cabin. Given the threats yet to be discussed, we only assume trust around a single gateway connected between a node (i.e., ECU, port, or outlet) and the primary network backbone.

2.2 Threat Analysis and Risk Assessment (TARA)

The goal of a Threat Analysis and Risk Assessment (TARA) is to identify threats and assess any risk and residual risk of the identified threats in a procedural manner [18]. When completed, the critical threats are identified so that we can apply the most valuable controls.

Although there are many conventional analysis approaches, such as EVITA and HEAVENS, a cybersecurity literature review is conducted in place of a formal TARA [19] [20]. Since incremental differences exist between attack vectors explored on passenger automobiles and those on other vehicle classes, only a summary of work outside the HD space will

be presented. This review will focus on threats and mitigations relevant to commercial vehicles and their associated network protocols (i.e., J1939, J1708, J2497). Following the study, a summary of threats and mitigations will be discussed.

2.2.1 Literature Review

The following examinations are loosely organized by topic for the reader. The initial review covers notable work in the passenger vehicle domain. Subsequent reviews cover recent threats specific to heavy vehicle trailers, then transition to network defense technologies and experimental mitigations deployed on heavy vehicles.

Notable work in the Passenger Vehicle Domain

Preceding investigations on HD vehicle network cybersecurity, several prominent demonstrations of vehicle vulnerabilities had existed for passenger cars. Most notably, practical experiments have been conducted on an unnamed sedan [21], a Toyota Prius and Ford Escape in 2014 [16], and a Jeep Grand Cherokee in 2015 [22]. In addition, more sophisticated attacks were developed in 2011 and 2015 by some of the same research teams. In both instances, the authors focused on remote attack vectors, and findings showed a consistent lack of security controls and safeguards in wireless vehicle features ranging from RF to Wifi.

After these demonstrations, more attention and investment have been given to mitigations, training, enhanced security development processes, information sharing and analysis centers (ISAC), and other security-related activities. These changes have informed the HD industry, spawning organizations and events such as the Automotive Information Sharing and Analysis Center (AutoISAC), the National Motor Freight Carrier Association (NMFTA), and the CyberTruck Challenge.

Burakova et al. – Truck Hacking: An Experimental Analysis of the SAE J1939 Standard [14]

In a 2016 paper by researchers at the University of Michigan, the authors presented the first experimental demonstration of implementable attacks on heavy vehicles. Unlike their contemporaries in the consumer space, their experiments focused on exploiting messages present in J1939 through direct access instead of backdoor exploits or wireless software vulnerabilities. However, as was concluded for OBD-II, the paper showed physical access to an HD diagnostic port by an attacker significantly threatened safety-critical systems.

The authors conducted their experiments on a 2006 class-8 semi-tractor, and limited experiments were conducted on a 2001 school bus. The researchers used replay attacks and logged communication between the vehicle and diagnostic tools to identify safety-critical messages that caused physical reactions. These messages were isolated and studied until specific SPNs were identified that correlated to the responses.

Seven PGNs and three safety-critical PGNs were identified. The latter three could control RPM and disable engine braking during operation in specific cases. Further research is proposed in the penultimate section, including a desire to investigate truck trailers.

Mukherjee et al. – Practical DoS Attacks on Embedded Networks in Commercial Vehicles [23]

In a joint effort by researchers from Colorado State University and the University of Tulsa, the first scholarly work aimed explicitly at exploiting the semantics of the J1939 protocol specification was published in 2016. After introducing the underlying technology and protocols, the authors present their threat model. Their work assumes an attacker with direct access to the CAN bus and an ability to inject arbitrary messages. Finally, the authors performed testing on a network of three ECUs and two Beagle Bone Blacks equipped with CAN hardware and Linux CAN utilities.

Three attacks were presented. The first is a request overload attack in which a node overwhelms a specific ECU on the network with a high volume of messages such that it can

no longer perform regular activities. In the second, an attacker sends spoofed RTS messages with a modified payload size count to an ECU requesting information. These messages can induce improper memory allocation when correctly timed, leading to a buffer overflow. Subsequently, this causes the ECU to crash. In the final attack, a malicious node attempts to block communication between two ECUs by preemptively connecting to one ECU with a spoofed source address in an RTS message. Without a proper end-of-message (EOM) acknowledgment, the logical connection remains consumed so long as the attacker maintains it with periodic clear-to-send (CTS) transmissions.

A key implication of this work is the ability to deny access to an arbitrary number of specific ECUs. Additionally, J1708/1587 provides similar mechanisms which have yet to be formally examined in academic publications.

NMFTA – Power Line Truck Hacking: 2TOOLS4PLC4TRUCKS [24] [25]

In a presentation at DEF CON '28 in 2020, researchers Ben Gardiner of the NMFTA and Chris Poor of Assured Information Security (AIS) discussed their team's work on hacking J2497 (aka PLC4TRUCKS). After summarizing key features of the protocol, they highlighted traditional tools used to interface with PLC and their technical limitations.

In place of a conventional tool, the "Truck Duck Beagle Bone Cape" developed in previous work was retrofitted with new hardware, tested, and validated on a live PLC harness. The presenters wrote new code to provide new PLC networking utilities from the command line. Adapters for wired communication between the Truck Duck and trailer were created by retrofitting commercially available J560 cables. Together, these efforts resulted in "gr-2497" [26], a tool for reading and decoding PLC frames, and "plc4trucksduck" [27] which enables PLC writing.

During their testing, they identified significant radio electrical noise emanating from the trailer harness, and after further testing and investigation, they achieved the first reported wireless read of J2497 traffic. Given the impact of remote reading, a formal disclosure was delivered to the US Cybersecurity and Infrastructure Security Agency (CISA).

NMFTA – Disclosure of Confirmed Remote Write to J2497 aka PLC4TRUCKS [28]

In a public letter from Ben Gardiner, the senior cybersecurity researcher at the NMFTA, a new attack on J2497 was formally disclosed. In partnership with AIS, their research indicates that J2497 receivers are susceptible to remote writing of J2497 messages up to 12 feet. Thus, practically all ABS equipment from the top three suppliers is now affected for tractors and trailers.

The impact of this short-range vulnerability is severe. As expected of the underlying protocol, trailer ABS equipment listens and responds to any perceived commands without any authentication. Thus, air brake actuation is feasible. The NMFTA has already forced solenoid valve tests, "roll-call", and LAMP ON controls to activate on actual equipment in numerous lab tests at various distances.

Gardiner provided suggested mitigations in a subsequent report discussed later in this review. Additionally, major US authorities were notified, including the US FBI Cyber Division and the CISA under the DHS. Technical descriptions regarding how he and his partners crafted these attacks were not fully disclosed; though, the equipment needed to exploit this vulnerability (SDR, signal conditioning, linear power amplification, and balun wire antenna) is estimated at around $300.

USDC and NIST – Guidelines on Firewalls and Firewall Policy [29]

The National Institute of Standards and Technology (MIST) is a physical sciences laboratory and a non-regulatory agency of the United States Department of Commerce (USDC). Its mission is to advance American innovation in various fields, and cybersecurity is among them. In cybersecurity, NIST develops standards, guidelines, best practices, and other resources that support the needs of US industries.

In a 2009 report, the authors Karen Scarfone and Paul Hoffman provided an organized overview of firewall technologies and how they work and evaluated standard firewall and network architectures. The authors also present firewall policies (i.e., IP address-based,

application-based, user id-based), planning, and implementation approaches. At the end of each chapter, a summary of recommendations is provided.

Although firewalls and firewall policy have existed for a long time, their application on HD networks is a topic of interest within the field. The report has several notable recommendations pertinent to the ongoing research task. For example, the authors recommend defaulting firewall policies to block all inbound and outbound traffic, with exceptions made for desired traffic. They also promote more nuance packet inspection when possible. Robust firewalls can better contextualize abnormal behaviors by inspecting both the traffic's source and destination and the content. Thus, there is evidence that we could apply the same procedures to legacy HD networks. For example, although not all messages contain destination addresses in J1708/1587, discussion in Section 1.2.3 showed that a J1708/1587 messages MIDs function similarly to CAN-ID source addresses.

The authors also recommend maintaining a consistent methodology when planning and implementing a firewall. One such method, the "five steps", includes: plan, configure, test, deploy, and manage steps.

NXP – NXP Stinger Module [30]

The "Secure TJA115x CAN Transceiver Family" by NXP Semiconductors was one of the first chip-level security solutions for securing CAN communication. It is a drop-in replacement for a standard CAN transceiver; the IC typically integrated into devices that interface with CAN. The company markets this product as part of a cryptography-free distributed intrusion detection system.

In addition to its primary interfacing functions, the TJA115x contains firewall features on both the transmit and receive lines, message tampering protections, and a leaky bucket procedure to enforce message rates on the transmit line. Security is achieved by blocking any host ECU messages not contained on a configurable allow-list and transmitting error frames on behalf of the ECU when any black-listed message appears on the receiving line. In

addition, a leaky bucket procedure caps the host ECU message rate to prevent a bus flood attack from a compromised node.

NXP argues that this is a viable solution in a defense-in-depth (DiD) strategy and avoids many technical drawbacks of a cryptography-based solution compliant with AUTOSAR SHE specifications.

USDC, NIST – Guide to Intrusion Detection and Prevention Systems (IDPS) [31]

In a 2007 NIST publication, authors Karen Scarfone and Peter Mell provide practical guidance on intrusion detection systems (IDS) and intrusion prevention systems (IPS) technologies. The authors define *intrusion detection* as "the process of monitoring the events occurring on a computer system or network and analyzing them for signs of possible incidents." An *incident* is the occurrence of a violation of computer security policies, acceptable use, or standard security practices. Throughout several sections, the authors provide an overview of the different types of these technologies and provide recommendations intended to assist with "designing, implementing, configuring, securing, monitoring, and maintaining intrusion detection and prevention systems (IDPS)."

Section 2, in particular, covers the fundamentals of IDPS. IDS is "software that automates the intrusion detection process." IPS is IDS software that also attempts to stop possible incidents from proceeding. The primary use of IDPS is to identify and or protect against potential incidents. However, IDPS can also help identify security policy problems, document existing threats, and deter actors from violating the security policy.

Functionally, IDPS often record information related to observed events, notify security administrators of important observed events, and can produce event summary reports. Preventative capabilities can often include stopping the attack itself (i.e., by terminating ongoing connections with an identified attacker or by blocking all access to a particular resource.) However, some systems can also change the security environment or change an attack's content to reduce its impact. The authors also identify three common methodologies IDPSs use

to provide accurate detection. These include *signature-based*, *anomaly-based*, and *stateful* detection methods.

The authors categorize these systems into four technological groups: network-based, wireless, network behavior analysis (NBA), and host-based. Network-based IDPS monitors the network traffic for particular devices and analyzes messaging activity to identify abnormal activity. This is particularly relevant to HD networks where a widely adhered messaging protocol such as J1708 and J1587 could provide the framework for messaging enforcement policy.

However, the authors also point out the limitations of each approach. For example, network-based IDPS require regular tuning and customization to accommodate changes in network activity. Network-based IDPS may also be unable to perform deep analysis under high traffic conditions. A crucial takeaway from the publication notes that *a single methodology is likely insufficient if thorough network security is desired.* As the authors put it, the combination of multiple types of IDPS and technologies are likely "to achieve more comprehensive and accurate detection and prevention of malicious activity."

Daily et. al. – Securing Heavy Vehicle Diagnostics [32]

In a technical demonstration paper titled "Demo: Securing Heavy Vehicle Diagnostics," researchers Dr. Jeremy Daily, Benjamin Ettlinger, and David Nnaji of CSU showed how a secure channel could thwart a compromised vehicle diagnostic adapter (VDA), VDA PC driver, or middle-person attack established between a legitimate VDA and the diagnostic port.

The authors facilitated the mitigation concept using a secure diagnostic application on the PC and a secure gateway on the vehicle network. First, an initial routine involving Diffie-Hellman (ECDH) protocol over CAN takes place to encipher session keys. Then, with the session keys created and an additional initialization vector, an AES-128 cipher in CBC mode is set up to secure communication between the application and the gateway.

The authors discuss the design and functionality of the secure gateway in a separate published paper presented at AutoSec 2021.

P. Murvay and B. Groza – J1939 Protocol Shortcomings and PKI-based Countermeasure [33]

In "Security Shortcomings and Countermeasures for the SAE J1939 Commercial Vehicle Bus Protocol," the authors Bogdan Groza and Pal-Stefan Murvay examine the protocol-level security shortcoming of J1939, discuss countermeasures based on enhancements to the protocol based on public-key infrastructure (PKI), and evaluate the impact of implementing their countermeasure to network communication and reliability.

The authors note that the CAN bus is used extensively across vehicle classes and that many practical attacks have already been demonstrated primarily on passenger automobiles. Given the architectural similarities between vehicle classes, the authors expand upon previous research by using CANoe simulations to explore how they could adjust these attacks for J1939. They also discuss attacks that are unique to the protocol.

Given a rogue or compromised node on the network with the ability to craft an arbitrary J1939 CAN message, the authors demonstrated how they could disrupt a genuine J1939 address claim procedure. Modified messages could also be used to perform a Distributed Denial of Service (DDoS) attack or node-to-node connection blocking. They also discussed how the transport protocol could be interrupted and how to exclude a node from a working set.

Since a lack of authentication is at the root of these issues, the authors present a security mechanism based on public-key infrastructure (PKI) comprised of an initialization and run-time routine. Data exchanges occur between a security designated ECU and a standard ECU bearing an OEM-certified public key. The authors also presented a modified but marginally less secure version of the two routines for non-OEM-certified ECUs.

A messaging scheme based on CAN and CAN-FD was designed to facilitate the security concept. Then the defense concept was baseline tested and validated against different attacks

in CANoe. In the concluding statements, the authors note that the concept may require some design trade-offs to implement the cryptographic functionality on less performant controllers, but that high-end and future controllers should be able to handle the overhead.

Daily et. al. – Secure Gateway and CAN Conditioner [34]

In "Securing CAN Traffic on J1939 Networks," the authors present a distributed intrusion detection system that leverages network firewalls and PKI-based authentication to identify malicious behavior on a J1939 network. This concept is implemented using a device called the "CAN-conditioner," which is placed between each ECU and the shared network. A similar device called the "secure gateway" is placed between the diagnostic port and the shared CAN bus and has the added function of initializing the authentication routine. The security protocol consists of two phases: an initial secure key exchange process and a run-time CMAC calculation process.

A hardware security module (HSM) was integrated into each device to perform rapid cryptographic operations required in the secure messaging scheme. The HSM helped reduce the computation cost associated with PKI-based messaging and provided a root of trust for security services. The authors also introduced custom J1939 PGNs to update data security statuses and exchange security information.

Because each device was equipped with two isolated CAN transceivers, SA-based access controls and message transmission rules could be configured and applied to control outbound and inbound traffic rates. Therefore, the devices also acted as network firewalls.

These devices were integrated and tested on three modules and the diagnostic port of a Class 6 Kenworth box truck. Although some underlying assumptions concerning the key-distribution system exist, the authors observed promising preliminary results such as network resilience to middle-person attacks, rogue nodes, and DDoS from a compromised node in their practical experimentation. Overall, the defense attempts to identify all falsely injected CAN packets.

Kim et. al. – ShadowAuth [4]

In the introduction to "`ShadowAuth`: Backward Compatible Automatic CAN Authentication for Legacy ECUs," the authors address the ongoing challenge of designing practical and deployable authentication schemes for communication between ECUs on CAN. Although many mitigation strategies exist in past research, the authors note that prior proposals have not satisfied the necessary compatibility and real-time constraints at one time.

In response to this problem, the authors propose `ShadowAuth`, a new authentication system called which offers backward-compatible authentication without redefining CAN packet definitions or acquiring ECU source code. `ShadowAuth` was also designed to minimize any latency effects on real-time communication during vehicle operation that is often attributable to message authentication. Furthermore, the authors contribute an automatic and architecture-agnostic method to implant authentication functions on existing ECU firmware via binary code injection. Their evaluations also indicate that the system mitigates well-known vulnerabilities of CAN and bus-off attacks within 60ms with 100% accuracy.

After providing an introduction of their work and sufficient background on in-vehicle systems and past research, the authors describe their research objectives in discussions of their threat model and system design. The implementation details of `ShadowAuth` are as follows.

Given the ECU firmware binary, `ShadowAuth` identifies CAN transmission functions using static and binary analyses. Using a combination of binary code injection techniques discussed in prior work, the existing ECU firmware and HMAC generation code are implanted in each identified CAN transmission function. An authenticator node is added to the CAN bus (implementation details are flexible here) by adding the relevant code to an existing ECU or gateway or by adding a new node to the bus. Once all the nodes are retrofitted with the proper authentication code, each "operational packet" sent on the bus is authenticated by an "authentication packet." These authentication packets are designed in such a way so that it does not conflict with existing CAN packet or protocol definitions but also communicate

relevant per-packet security information with high integrity. Authentication on the bus succeeds when the authenticator finds the correct match between an op-packet and an auth-packet. Otherwise, authentication fails before the worst-case period of 60ms. `ShadowAuth` follows an "accept-first-authenticate-later" policy which is intended to minimize the effects on real-time vehicle operation.

The authors performed their binary injection experiments using a set of three ARM-based open-source ECU firmware on three different hardware platforms. The `ShadowAuth` performance tests were conducted using J1939 CAN traffic replays from a 2015 Kenworth and the CyberTruck research truck [35]. Incidentally, this data was collected during my trip from CSU to Detroit, MI. Their findings showed that `ShadowAuth` detects packet injection and bus-off attacks while maintaining compatibility without conflict with the J1939 protocol.

Their results also showed that CAN transmission functions are detected with 100% accuracy without ground truth. Given that a single auth-message corresponds to each op-packet, the authors also evaluated the impacts on network busload. In their simulations, 43% of auth-packets were sent without delay; meaning the bus was available at the time of transmission. In the worst case, the patched controller sent packets in 14.0ms, acceptably below the 60ms theoretical value. Furthermore, the addition of HMAC generation code incurred a negligible 4μs execution time increase and a 212KB code-size increase. In cases where flash memory is scarce, the authors proposed modifications to the hashing algorithm to reduce the code size.

NMFTA – Mitigations Options to J2497 Attacks

In a report released by the NMFTA in March 2022, the authors presented protection methods and sets of combined methods as practical solutions to a recently exposed vulnerability in J2497 (PLC4TRUCKS) (referred to as "Radio-frequency Induced Remote Write"). Specifically, nine protections and seven solutions are discussed in detail to provide various levels of protection against the risk.

Many of the outlined protections propose physically shielding the hardware to suppress most of the noisy RF given off by the power lines. However, given the novel nature of the attack, IDS and IPS mitigations were not explored.

2.2.2 Review Findings

Wired Threats

The simplest threat model is an attacker with physical access to the network. A typical argument against this model is that an adversary could already impair the vehicle by removing bolts, slashing tires, or cutting brake lines. However, as Burkaova et al. point out, these arguments tend to overlook more nuanced aspects of a wired attack, such as sustained access and minimal evidence [14].

The literature review revealed several wired attack vectors relevant to HD systems.

Diagnostic Tools. Under normal circumstances, the diagnostic port is an important interface for servicing and testing the vehicle. First issued in 1999, SAE J1939-13 has specified the requirements for diagnostic connectors used for all off-board J1939 communication for HD vehicles [36]. In addition, the legislated use of the diagnostic port by the EPA has been around since 2009 for regulating emissions from new and in-use on-highway vehicles and engines.

Manufacturers and regulators have used diagnostic tools to help identify faults in trucks for decades. These tools are linked to a PC running trusted test software and a standardized API on one end and the diagnostic port on the other. However, a compromised device could mount an attack on various networks depending on the built-in hardware. These compromises were explored in [32], though not specifically regarding J1708/1587.

Rogue Node. The diagnostic port has also emerged as the conventional entry point for general research or compromise. Any device could be left plugged in to extend remote access, flash malicious firmware, extract firmware, or slyly log continuous information. Even non-malicious research activity could produce unexpectedly damaging results.

According to the NHTSA, many passenger vehicle OEMs are investigating (or have already implemented) firewalls, gateways, and segmented architectures to isolate access from a node at the diagnostic port [5]. In addition, heavy-duty vehicle OEMs are adopting these strategies as well.

The exposed bridge in a tractor-trailer configuration is also a viable attack vector in light of new investigations from the NMFTA [24]. Although this idea is not new [14], these hacks have been mostly overlooked in practice.

Third-Party "Dongles". Insurance agencies have used third-party dongles in passenger vehicles for prorating premiums based on good driving patterns. Analogous devices exist in the HD space. Others satisfy niche needs for fleets and vehicle operators (i.e., telematics, logging, driver assistance). However, since these devices are often beyond the control of the OEMs, they pose a higher risk compared to typical VDAs because of dubious vendors on the market.

Standards. Attacks on the CAN network range from the obvious to the precise. Closer to the precise side, protocol-based (data-link layer) DoS were discussed in the literature review by Mukherjee et al. [23]. These types of attacks can be used to target specific ECUs.

Unlike the proprietary messages in passenger vehicles, open standards allow adversaries to log data and evaluate it later. They can craft an attack based on the information gathered and deploy it after completion. Other functionality enabled by the standard could also be exploited, including reading and writing from ECM memory and reverse engineering UDS seed/key exchanges. There is also a need for publications on attack procedures that exploit SAE J1708/1587 message semantics.

Infotainment. Multimedia interfaces like USB ports, CD drives, SD card slots, and auxiliary audio jacks are integrated into infotainment cluster modules in all classes of vehicles. Portions of these modules connect to a low-speed CAN bus and have been exploited in past research [22].

Power Line and 12V Outlets. 12V accessory outlets are often available in various locations within the cabin. This outlet is supplied by battery power and could serve as a potential entry point for any device with custom PLC hardware. The battery line could also be accessed via the diagnostic port. Direct J2497 attacks were recently demonstrated on trucks [28] but could more likely occur on large shared transit vehicles where exposed power outlets are available.

Body Builder Interfaces. A unique feature of MD/HD vehicles is vehicle bodybuilding, in which custom equipment is fabricated and installed onto a base chassis. Cement mixers, electric service trucks, transit vans, tow trucks, and harvesters are examples of heavy vehicles with custom bodies. These trucks can have unique interfaces to various networks that could increase the attack surface in unique ways.

Wireless Threats

Though more difficult to achieve in practice, remote access to the network offers an attacker more surreptitious and scalable control. As noted in the literature review, there have been many successful remote attacks on passenger vehicles. The NHTSA has compiled a list of similar short- and long-range wireless attack vectors in the HD space as well [5]. Only wireless threats relevant to the feature definition have been examined for brevity.

Fleet Management Systems. Many MD/HD fleets deploy management systems (FMS) to improve cost, operations, and asset tracking and management. Many FMS services offer hardware and software solutions that integrate with the semi truck-trailer that may rely on telematics, GPS, CAN, and other forms of networked communication. The use of FMS is also expected to grow, given the requirements for hours of service in the FMCSA ELD mandate of 2017 [37]. In addition, weak security at various points in the FMS software architecture could be exploited to gain remote access to the vehicle networks.

Trailer RF Noise. RF-induced read and write to PLC is a newly disclosed zero-day vulnerability that affects communication on J2497 [24] [25] [28]. This happens because the long contiguous wires used for PLC communication act as radiating elements. This

effect is amplified as higher voltages are applied to the lines. PLC transmitters are also allowed to be "noisy" to ensure transmissions propagate across the length of the vehicles when multiple trailers are combined. Complicating the issue is the fact that the underlying messaging protocol on J2497 is essentially J1708/1587 with a different physical layer. The implication of this threat is severe. An attacker can circumvent the challenges associated with compromising an ECU or connected device and simply broadcast arbitrary messages to the vehicle bus from a confirmed range of up to 12 feet.

2.2.3 Impact

A heavy vehicle is often considered a cyber-physical system, meaning it is a computer system where mechanisms that act within the physical world are controlled or monitored by computer algorithms. Thus, a mounted attack on a vehicle network in operation would pose an immense risk to the proper functioning of safety-critical systems related to steering, braking, and power generation.

Although automatic and semi-automatic steering systems are available on advanced passenger cars, the same technology has only been implemented in limited prototype developments on heavy vehicles. Based on stakeholder interviews, the NHTSA found that cyber-physical control of heavy-vehicle steering is not prevalent [5].

Automatic braking, however, is commonplace across all vehicle classes. Heavy trucks use pneumatic air-braking systems with wheel-end modulators, unlike passenger vehicles with motor-based hydraulics with wheel-end valves. Heavy trucks have three braking systems: service, parking, and emergency. Thus, an attack on the brakes through the vehicle network would necessarily differ from that on a passenger automobile.

The aim of a network-based brake attack would likely be to engage the emergency service brakes fully during operation or partially disengage the brakes at any given time. Although the latter case has not been studied, Burakova et al. encumbered engine-braking through

spoofing on the vehicle network during active driving [14]. In addition, the NMFTA has also shown that trailer brakes can be engaged via spoofed PLC when the vehicle is idle [24].

Potential attack scenarios also exist for powertrain systems. These include unintended acceleration, disallowance of engine downshifting, and unintended increase in engine speed (RPM). Although minimal demonstrations of powertrain vulnerabilities exist across vehicle classes, Burakova et al. achieved limited control of engine RPM causing direct acceleration in experimental work on a class 8 truck [14].

2.2.4 Mitigation

Many research papers presenting new vulnerabilities and attacks in commercial and consumer spaces have also suggested mitigation methods. The NHTSA has provided six categories to organize these methods shown in Table 2.1.

Table 2.1: Mitigation Categories

Category
Secure Architectures
Secure Applications
– Gateways and Firewalls
– Intrusion Detection and Protection Systems
– Secure Diagnostics Authentication Schemes
Secure Development Processes
Secure Hardware
– Embedded Hardware Security Modules
– Secure Boot and Trusted Platform Modules
Safety and Plausibility Checks

Although each method plays a critical role in an overall cybersecurity strategy, secure applications and security hardware warrant additional exploration in the context of this thesis.

Gateways and Firewalls

A gateway is networking hardware or software that allows data to flow from one network to another. For example, gateways could link two different CAN networks in a vehicle. A gateway can also translate a message from one protocol to another if intended to connect two networks using different protocols. It can also function as a firewall. A firewall is security hardware or software that blocks messages on either network path based on predetermined rules. A gateway or bridge ECU typically functions as a firewall since they are rarely separate devices.

As noted by [29], gateways and firewalls have extensive use in IP security and have a critical role in future mitigation strategies for vehicle networks. Nearly all of the explored mitigations in the literature review used them.

Intrusion Detection and Protection Systems (IDS/IPS)

An intrusion detection system is security hardware or software that monitors network traffic and flags activity that violates protocol or appears abnormal. An intrusion protection system takes additional measures to eliminate the source of the violation, minimize network disruptions, and protect any information in response to abnormal behavior. The application of both security systems for future vehicle networking is the topic of ongoing discussion within academia and industry, as examined in the literature review.

Secure Diagnostics Authentication Schemes

Although official standards do not exist for secure diagnostic authentication, there are recommended practices for OEMs working on their implementations. Most notably, AUTOSAR SecOC (SecOC) [38] supplies resource-efficient and practicable requirements for payload authentication, but tailored techniques are also likely to exist. In most cases, these approaches rely on cryptographic functions and key management. When applied to vehicle communication such as CAN, OEMs tailor these techniques to comply with payload size, bandwidth, and latency requirements. Additionally, these schemes require thoughtful con-

sideration concerning key-management and distribution strategy, potentially over the use-life of the device. This last effort can be prohibitively costly and complex.

Still, the application of secure diagnostics authentication schemes for future vehicle networking is the topic of ongoing discussion within academia and industry, as shown in the literature review. These schemes mostly derive from challenges during the early internet and have been proven effective for hardening network security.

Embedded Hardware Security Modules

An HSM is essential in any security scheme involving public-key infrastructure. Among other things, an embedded HSM provides secure cryptographic key storage and can perform cryptographic functions such as encryption, decryption, random number generation, signing, and verification. Even during cryptographic operations, a certified HSM does not reveal any information about the secrets it contains. Security is made possible by segmented memory regions and carefully designed hardware operations. Furthermore, these designs significantly accelerate cryptographic operations compared to software implementations.

Discussions of HSM use in ECUs as a part of securing diagnostics is a topic of consensus within academia, as shown in the literature review. However, the heavy-duty ECU development lags behind passenger automobile development in this regard [5].

2.2.5 Risk Assessment

With an understanding of J1708/1587 and PLC network entry points, we now explore the associated risks. We begin by listing probable attack objectives in the order of increasing impact in Table 2.2.

Table 2.2 also lists possible methods for an attacker to reach these goals assuming the attacker has network access. These methods were observed in the research discussed in the literature review. Although all the methodologies exist on J1939, the semantic differences for the same J1708/1587 are assumed to be negligible. An attack tree is shown in Figure 2.1 to understand the relationship between the attack goals and methods. Given the various

Table 2.2: Attack Goals and Methods

Attack Goals	Attack Method(s)
Network surveillance	Exploit vehicle network entry point
Data exfiltration	Message spoofing Message logging
Unintended vehicle actuation	Message spoofing Message fuzzing Message replay
Denial of service (DoS) (Specific ECU)	Malformed RTS False address claim Connection exhaustion Request overloading
Distributed denial of service (DDoS)	Request message flooding Random message flooding Repeated DoS

impacts of the items in Table 2.2, only the last three items were designated as attack goals in the attack graph. The attack tree consists of six objects:

Assets (AA): These are persistent capabilities leveraged for more sophisticated attacks.

Methods (AM): Particular procedure for accomplishing part of an attack objective or goal.

Objectives (AO): A necessary milestone within the attack goal.

Goals (AG): Typically, a fully realized compromise from the viewpoint of the system designer and an ultimate desired result of the attacker.

AND Gates: Indicate that all the connected items are needed to achieve the parent item. Designated by a diamond.

OR Gates: Indicate that any connected item can achieve the parent item. Designated by a vertical bar.

There are various unexplored items within the attack tree, including the preceding activities required for AA1-AA4 and cases where only read or write capabilities exist but not

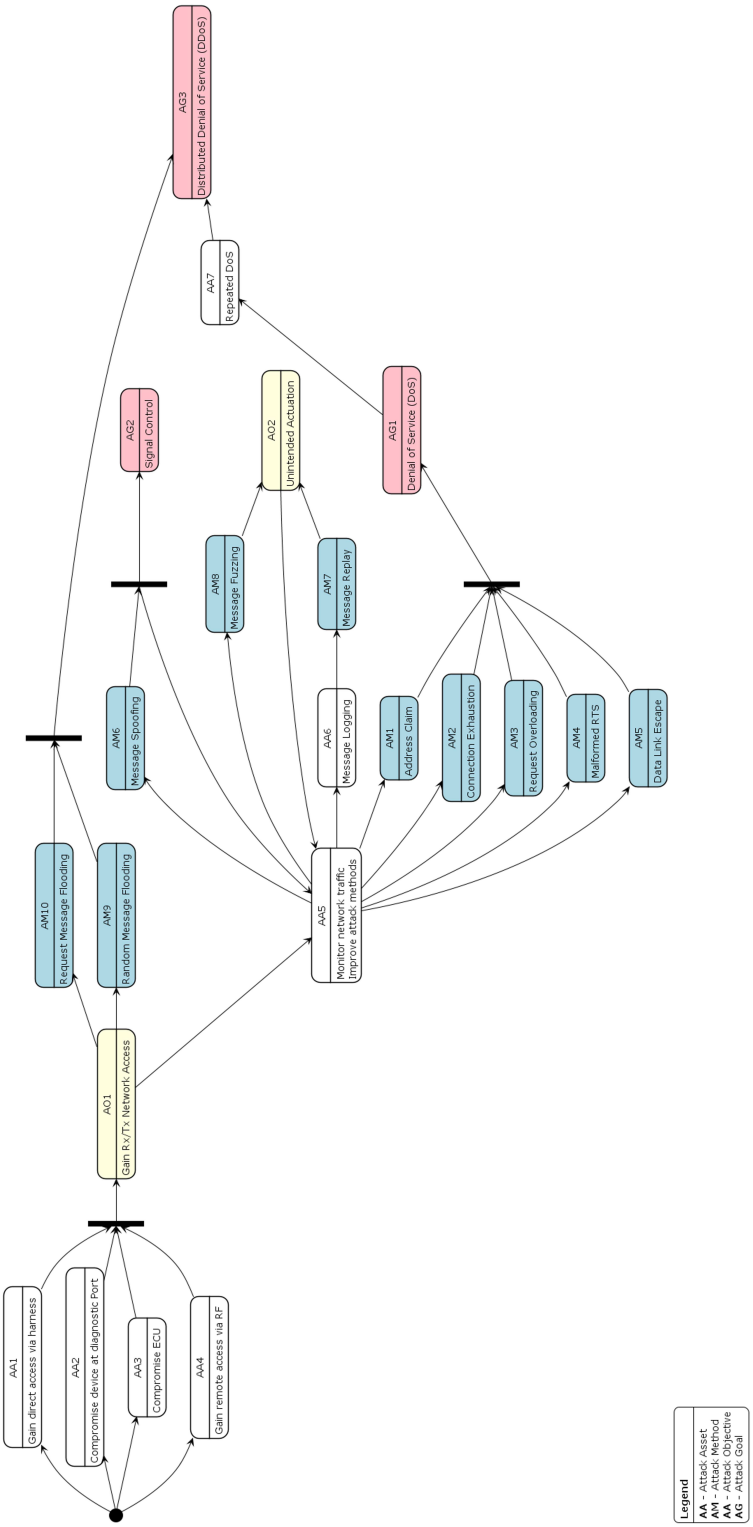


Figure 2.1: Threat Analysis Attack Graph

together. However, the attack tree presented is assumed to reflect the worst-case scenario in attacks of this nature.

2.3 Cybersecurity Goals and Concept

2.3.1 Goals

Although many attack methods exist on the attack tree, many rely on spoofing messages with a header that matches the controller of interest. In the case of J1939, this implies modifying the SA of the message. For J1708/1587 protocols, this will be the MID. AG1 and AG2 both rely on these methods in some form. Thus, any proposed mitigation should reliably differentiate between a legitimate message from an ECU and a spoofed one.

Identification of abnormal transmission rates, as in the case of message flooding, would also help improve the mitigation strategy. In this paper, flooding refers to sending a large number of messages directly to an ECU or network at a high rate. This rate can often approach the theoretical bus maximums. AG3 relies on this type of method.

An important nuance that is not apparent in the attack graph is where the attack originates. AA1 and AA4 are rogue nodes that *join* the network. AA3 and AA2 are compromised nodes that are *integrated* and assumed to be trustworthy. In the case of a flooding attack mounted by a rogue node, the identification procedure is essentially the same as identifying a spoofing attack since a legitimate node will observe the intrusion. However, a flooding attack mounted by a compromised node could appear to the rest of the network as legitimate high traffic. Thus, it is also necessary to differentiate between normal and abnormal transmission rates on a compromised node or limit it entirely.

Given these needs, the following eight cybersecurity goals were identified:

[**CG1**] Prevent DoS attacks via message spoofing from a rogue node

[**CG2**] Identify DoS attacks via message spoofing from a rogue node

[**CG3**] Prevent DDoS attacks via flooding from a rogue node

[**CG4**] Identify DDoS attacks via flooding from a rogue node

[**CG5**] Prevent DoS attacks via message spoofing from a compromised node

[**CG6**] Identify DoS attacks via message spoofing from a compromised node

[**CG7**] Prevent DDoS attacks via flooding from a compromised node

[**CG8**] Identify DDoS attacks via flooding from a compromised node

2.3.2 Concept

Although securing in-vehicle communication is an ongoing challenge, past research presented in the literature review provides suitable options for accomplishing our cybersecurity goals.

One viable approach is to authenticate each message using public-key cryptography. If every ECU transmits a message with a unique signature for verification by its intended recipient, then message spoofing becomes immensely more challenging and identifiable. Furthermore, flooding attacks become more identifiable since numerous faulty messages would occur over a small period of time from an unauthenticated attacker.

Although standard practices exist for implementing strong PKI on embedded systems, this security architecture incurs some notable drawbacks. A report by NXP summarizes the challenges of implementing the state of the art cryptographic solutions concisely as follows [30]:

- Requires competent key management
- Requires a freshness value procedure at startup for replay protection
- May delay the readiness of authentication if a grace period is implemented
- Introduces transmission latency message authentication code (MAC) is calculated

- Increases the network busload to transport the additional parameters (i.e., MAC and freshness value)
- Consumes more processing cycles in the sender and receiver applications
- May increase the sender application complexity. (i.e., Queued messages may be invalidated or deprioritized when new freshness values are calculated.)

Attacks to J1708/1587 and PLC are still novel, and the impacts of implementing PKI on these protocols in practical demonstrations do not yet exist. Still, it can be assumed that the drawbacks would be similar. However, some exceptions may exist since the aforementioned drawbacks were only discussed for CAN-based communication with payload maximums of 8-bytes. Murvay and Groza showed that incorporating alternative busses with larger payload maximums could offset some disadvantages [33]. This is an important finding, considering J1708 allows up to 18-bytes in the payload after the MID, the obligatory PID, and checksum characters are taken into account. Therefore, the design of such a mitigation strategy will be an exciting focus in future work.

The addition of strategically placed gateways and firewalls can also identify and mitigate flooding or spoofing attacks from compromised nodes. These concepts have been explored practically by [30] and [34] for CAN networks and could effectively extend to J1708/1587. Given the promising application of this strategy, the concept is summarized as follows:

1. A trusted gateway node is incorporated between each ECU or standard access point on the network. The device associated with each gateway is called the "host." (12-volt outlets in the cab, tailboxes, and noseboxes could also theoretically incorporate this hardware, though this is not explored seriously).
2. Like the NXP "Stinger" concept [30], each gateway functions as a firewall for its host by applying an access control list on its transmit and receive lines. If an outbound message from the host contains a MID that is not on the allowlist, it is blocked. Similarly, any

inbound message on the gateway blocklist is stopped before reaching the host. The allow list and blocklist can be adjusted as needed so that normal communication is maintained.

3. If messages are received that impersonate a gateway host, it will notify all other gateways that a spoofing attempt was made. If a compromised node attempts to spoof messages, the gateway will block it and note the incident. If enough incidents occur in either case, all gateways will be notified to potentially block messages using the suspicious MID.

4. To mitigate the risk of a flooding attack, message transmission rates should be monitored. If a flooding attack on the transmit path or receive path is identified, the network is notified, and all messages are blocked until the next system reset.

This concept has many advantages when compared to PKI-based methodologies. In sum, the concept avoids nearly all the drawbacks associated with cryptography, is less complex, easier to prototype on J1708/1587, and potentially easier to manage and configure. Additionally, the concept is simple enough to be integrated into a larger cyber-defense strategy.

However, one drawback is that the approach relies on established trust in all the gateways. If a single gateway is compromised, an adversary could masquerade as the gateway host and progress towards their attack goal. One way to combat this is to implement a trust anchor and additional cryptographic defense. Though compelling, subsequent analysis will assume that the gateway is trusted. Therefore, a gateway compromise is beyond the scope of the paper.

Furthermore, this system does not fulfill all of the desired cyber security goals. Although it may be possible to detect network abnormalities from a rogue or compromised node at any time, preventing them from happening is only possible if we assume the attack is only mounted from a known access point or established controller. However, remote attack vectors such as the RF-induce write to J2497 show that this is not always the case. In the case of

an RF-induced flooding attack, malicious messages could be identified and blocked by each gateway, but messages would persist on the public network. Worse, the attacker could force all gateways to block the MID of a legitimate node.

However, one could argue that confirming an attack has occurred is sufficient to warrant additional human interventions. Furthermore, identification of an attack could be the first step in subsequent mitigation efforts. Regardless, a deep analysis of prevention efforts concerning this specific case is resigned to future work.

There is also the complication of differentiating legitimate high busloads and message flooding. For example, legitimate high network activity often occurs during firmware updates or the generation of crash reports when large files are divided into numerous segments to transmit on the vehicle network. In SAE J1708, this activity may be triggered using the data-link escape PID 254 (0xFE) followed by a proprietary PID and the connection management PID 197 (0xC5) defined in the J1587 transport protocol [10]. It's also possible that numerous valid multi-parameter requests happen to take place in a small time window.

Many DoS attacks are detected by counting events during specified periods of time and alerting when threshold values are exceeded. However, indiscriminate flagging of flooding activity based solely on high busload activity would lead to false positives in the cases outlined above. Thus, more nuanced approaches are warranted that take into account more of the networking context. One approach from NIST suggests using "stateful inspection," which improves upon message filtering by tracking the state of ongoing connections. This means that time, destination addresses, and connection state information, when applicable, are used with the MID to determine if a violation has been detected.

For example, a data transfer between two ECUs using PID 197 has three states: connection establishment (Control Command 1, RTS), connection usage (Control Command 2, CTS, and PID 198), and termination (Control Command 3 and 4, EOM/Abort) [10]. Thus, a legitimate connection should theoretically only occur between the two devices and end within a reasonable time. If an attacker attempts to flood a network with repeated requests for long

connections with multiple devices, a stateful rule set could detect an abnormal amount of ongoing connections and abort them. A similar strategy could use a timeout value to abort lengthy connections.

For the sake of simplicity, the application of stateful anomaly detection is deferred to future work. In the current work, a predetermined occurrence count of continuous high busload activity combined with message rate limiting mechanisms should be sufficient for demonstrating the feasibility of the mitigation concept.

2.4 Functional Requirements

Before defining the functional cybersecurity requirements, the definition of flooding shall be defined as exceeding a pre-configured message transmission rate for a continuous fixed period. The rate and period definitions are defined at the discretion of the system designer. Outbound messages are those sent by the host. Inbound messages are those that are received by the gateway from the shared network. Given the cyber goals, concept, and definition, the following functional cybersecurity requirements in regards to detection are defined as follows:

[**CR1**] The combined system shall identify message spoofing of a specific MID on the shared J1708/1587 network.

[**CR2**] Each system shall identify message spoofing attempts from its host.

[**CR3**] The combined system shall identify message flooding attempts from a specific MID on the shared J1708/1587 network.

[**CR4**] Each system shall identify outbound message flooding attempts from its host.

The following functional cybersecurity requirements for the concept in regards to mitigation are as defined follows:

[**CR5**] Each system shall block inbound messages from an identified rogue MID on the shared J1708/1587 network.

[**CR6**] Each system shall block all outbound message spoofing attempts from its host.

[**CR7**] The system shall block inbound messages from any MID that has attempted to flood the shared J1708/1587 network.

[**CR8**] Each system shall block outbound message flooding attempts from its host.

These requirements are revisited in Chapter 7.

Chapter 3

System Development

This chapter describes selecting an HD vehicle platform for deploying and testing the cybersecurity concept. First, the system goals are identified, and a system-level concept is defined. Then, with the desired qualities identified, the system requirements are drafted, followed by the early stages of system design.

3.1 System Concept

Ideally, we deploy the prototype mitigation concept on a real heavy-duty vehicle network for testing and evaluation. However, this is not practical since the proposed system would require significant retrofitting. Since this level of effort is beyond the project's scope, there is a need for a suitable analog with only the necessary elements for satisfying the cybersecurity requirements. A harness, power, a few ECUs, sensors, and actuation systems encompass most of these elements. Furthermore, given the low complexity of the analog, any faults originating from the integration procedures are also more identifiable. Thus, the first system goal is defined as follows:

SG1: Select a vehicle simulation system for security concept integration and testing.

The system concept involves four unique components: an ECU, a gateway, a rogue node, and a compromised node. The SystemsCyber research group possesses many suitable ECUs for the first component.

Although many devices could serve as a programmable gateway (i.e., the BeagleBone Black with HD/TC, TruckDuck), the design of a custom embedded device is achievable with the author's previous project experience. With this flexibility, the device could also be programmed to execute all the necessary functions required by the gateway, rogue node, and compromised node. An alternative way to develop a compromised node is to reflash an ECU

with modified firmware. However, this is prohibitively complex. Thus, the second system goal is defined as follows:

SG2: Develop a general-purpose device with programmable hardware, J1708/1587 network interfaces, that can perform the gateway, attacker, and observer functions.

With SG1 and SG2 defined, the remaining goals follow naturally:

SG3: Retrofit the vehicle simulation system for testing the mitigation concept

SG4: Integrate the mitigation concept into the vehicle simulation system

SG5: Test the integrated mitigation concept and evaluate it against the cybersecurity requirements

3.2 System Requirements

The following requirements were defined using the system goals and concept:

[**SR1**] The simulation system shall contain representative sensors, actuators, harnesses, and ECUs of a semi-truck and trailer with a J1708/1587 network.

[**SR2**] The system shall maintain functionality for extended periods of operation.

[**SR3**] The system shall provide interfaces for the integration of the network security concept.

[**SR4**] The network security concept shall be tested and validated on the simulation system.

3.3 System Design

The SystemsCyber research group possesses a tractor and trailer dual air-brake system simulator (DABSS) built by the Service Training Academy of Daimler Trucks NA [39].

DABSS includes a fully functional air supply system (except for the compressor), primary and secondary air brake systems, and a parking brake system. The system also contains air pressure gauges, connection fittings, glad-hands, functioning slack adjusters, and a Wabco trailer ECU. The panoramic photo in Figure 3.1 shows the simulator before any retrofitting and the location of the major mechanical components. The air system schematic for a 4S/4M tractor system is also provided in Figure A.1 of Appendix A.

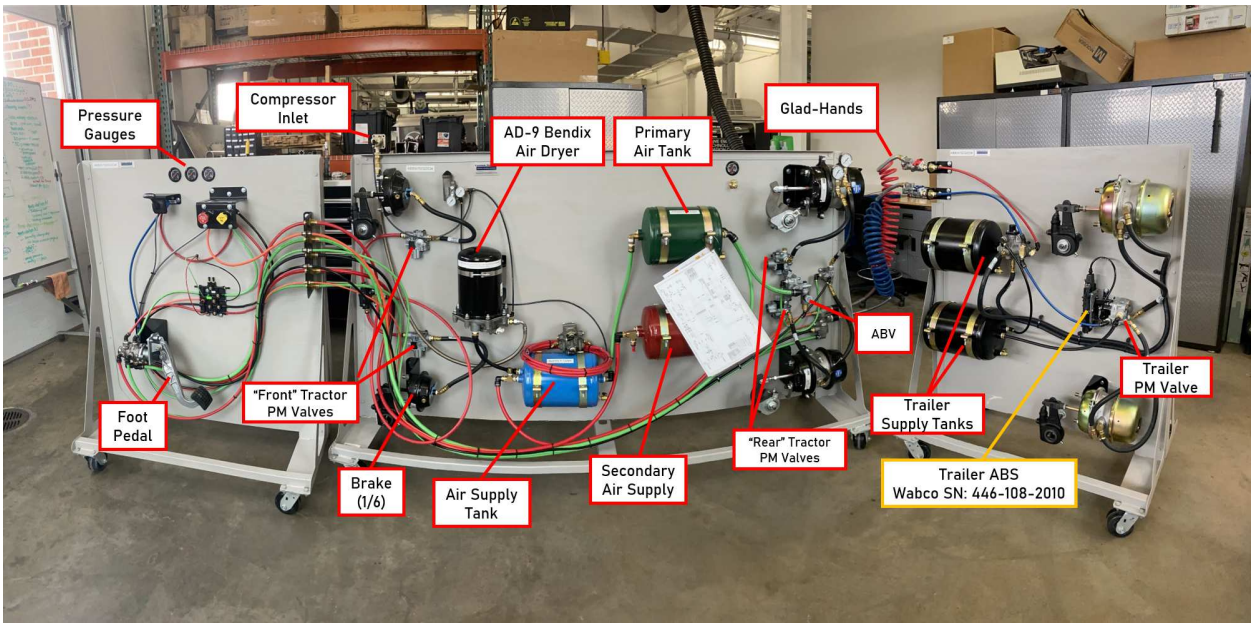


Figure 3.1: Dual Air Brake System Simulator (DABSS) with Component Labels

Despite all the stock functionality, DABSS does not satisfy SR1. Figure 3.2 shows the system configuration as received, and Figure A.2 shows the target baseline configuration. Given these differences, the system was retrofitted with additional components using available resources and documentation described in the following paragraphs.

Tractor ABS Controller and Integration. Although DABSS contains the air systems for a tractor-trailer combination, it only came with a trailer ABS controller. Two suitable tractor ABS controllers were located: a Bendix Premium Cab Model EC60 controller and a Meritor Wabco controller. In terms of functionality, both the Bendix and Wabco offer 4S/4M ABS, ATC, ABV, and retarder connections. However, only a license to the Wabco

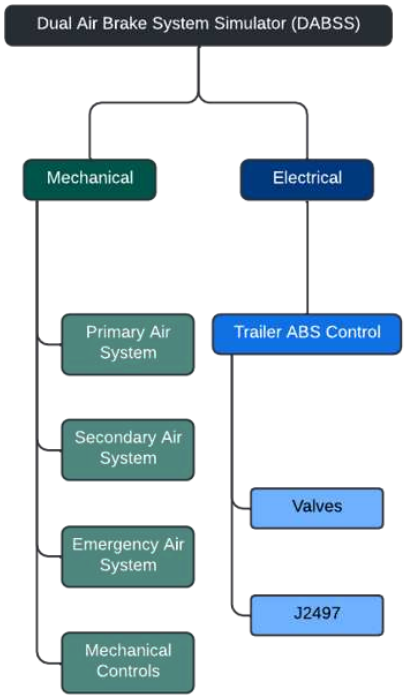


Figure 3.2: "As Received" DABSS Configuration

Toolbox software could be obtained. Aside from the air dryer, all the remaining components are Wabco branded as well. Ultimately, the Wabco module was selected because of the ease of integration and compatibility with the existing DABSS hardware. A photo of the module is provided in Figure 3.3.

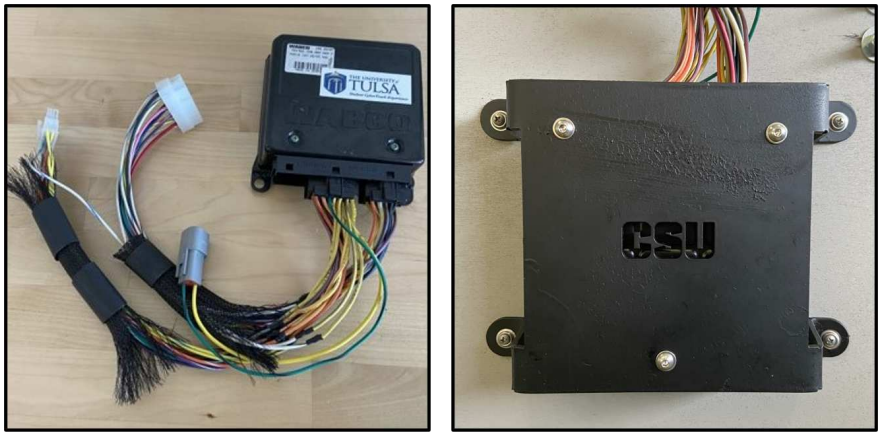


Figure 3.3: (a) WABCO Tractor ABS Module (b) ABS Module Fixed to Mounting Bracket

After locating the maintenance manual for the controller, the power, pressure modulator valve (PMV), and ABV connections were identified [40]. A copy of the wiring schematic is provided in Appendix A.3. Using an official Wabco cable catalog, the bayonet cables for connecting the valves and ABV were identified online. However, available wiring materials were used to create custom connections to these components instead. Data and power were provided using a custom harness.

Power. A 12V battery, fuse box, and switch were used to provide the appropriate energy to the electrical harnesses. A custom battery box was also built and mounted at the bottom of the board.

Harness and Mounts. Custom component mounts (i.e., Figure 3.3b and 3.4b) were designed in CAD software and fabricated using a water-jet cutter and forming equipment available at the Powerhouse Energy Campus. The mechanical drawings for these parts are provided in Appendices A.6 to A.9. A wiring harness was also fabricated to connect the mounted components. The supporting harness diagram is provided in Appendix A.5.

Trailer ABS Controller Integration. A Wabco power cable was acquired from a local supplier to break out the input power and PLC connections from the trailer ABS. A custom cable was made to bridge the mounted J560 socket and power cable using the pin-out provided by the maintenance manual shown in Figure 3.5 [1]. A standard J560 cable and a commercial J560 PLC adapter were used to bridge the trailer ECU and forward PLC data to the J1708 bus. With this connection, all data between the two controllers could be observed.

The trailer ABS controller already came equipped with a connector to the trailer modulator valve. Two ports for wheel speed sensors were also available on the controller. Although the lab possesses an experimental tone-ring generator, these were not populated during experimentation.

Air Fitting. A single hose coupling was added to the inlet of the DABSS air supply as shown in Figure 3.4a. The facility compressor helped supply air at a controlled operating pressure of 80 psi for testing.

Figure 3.4: (a) DABSS Air Inlet Coupling, (b) J560 Socket and Mounting Bracket

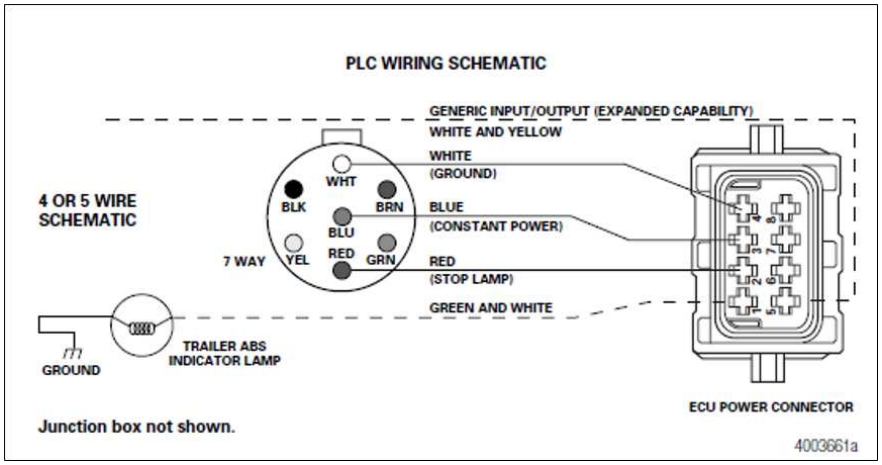


Figure 3.5: Wabco Power Cable Wiring Diagram [1]

Diagnostic Software, Logging, and Troubleshooting. When the system was powered on, live network traffic was initially observed using a DG DPA5 and the DG Diagnostic software application. A Wabco Toolbox license was later acquired through Dr. Daily. By using proprietary messages, this software displayed more specific static and dynamic information from the tractor and trailer controllers. The software was also used to identify faults, write static information to the controllers, and test the PMVs when the system was pressurized.

Table 3.1 shows the baseline faults observed after the system was retrofitted, pressurized, and connected only to the PMVs and ABV. The faults occurred because the tractor and trailer ECUs were not connected to wheel speed sensors or brake lamps. Despite the faults,

a baseline log of J1708 and J1939 traffic from each controller was acquired using the logging utility in DG Diagnostics. Because of the cumbersome formatting of the log files produced by the software, this data was recaptured with a completed gateway in a more suitable format later in the project.

Table 3.1: DABSS Baseline Faults

Fault #	Description	Status	SID	FMI	Count
1	Right Rear Sensor - Open	ACTIVE	4	5	1
2	Left Rear Sensor - Open	ACTIVE	3	5	1
3	Left Rear Sensor - Open	ACTIVE	1	5	1
4	Right Front Sensor - Open	ACTIVE	2	5	1
1	Sensor YE1 open circuit	ACTIVE	4	5	1

Analysis of each unit in isolation revealed the message identifiers claimed by the two controllers (before any address claiming procedures), the DG diagnostic application, and the Wabco Toolbox application (via the DPA5). These are provided in Table 3.2.

Table 3.2: DABSS Baseline Message Identifier Table

Component	MID (DEC)	MID (HEX)
Tractor ECU	136	0x88
Trailer ECU	137	0x89
Trailer ECU (PLC)	10	0x0A
DG Diagnostics Application	136	0xAC
Wabco Toolbox Application	136	0xAC

Support Documentation. Table 3.3 shows an ordered list and description of all the supporting documentation that was located during DABSS retrofitting. These documents are provided in the thesis repository since they are difficult to acquire online. Table 3.3 also provides the list of components that were acquired or identified during the development effort. A schematic for connecting the two controllers to DABSS was created in Altium and is provided in Appendix A.4.

Table 3.3: DABSS Retrofitting Components and Documentation

Part	Part Number	Acquired
Nexiq Universal J560 PLC Adapter	MPS604020	✓
Phillips Trailer Connector Cord Socket	16-725	✓
J560 Cable	-	✓
DG DPA5 Dual CAN Protocol Adapter Kit	DPA5-KIT	✓
DG Diagnostics Software	-	✓
Meritor Wabco Toolbox Software	820023-12M	✓
Wabco Trailer ABS Module	446-108-201-0	✓
Wabco Tractor ABS Module	446-004-603-0	✓
Bendix EC-60 Module	486-107-104	✓
Air Dryer Harness Plug	109-869-N	-
8m PMV Bayonet 3-Wire Cable	449-514-080-0	-
8m PMV Bayonet 2-Wire Cable	449-415-080-0	-
Wabco Trailer ECU Power Cable	449-326-010-0	✓
1/4" Air Hose Coupling	1077T18	✓
Bendix Air Dryer Service Manual	-	✓
Bendix EC60 Service Manual	-	✓
Wabco Tractor ABS Service Manual	-	✓
Wabco Trailer ABS Service Manual	-	✓
Wabco Cable Catalog	-	✓
DABSS ABS Retrofit Wiring Schematic	-	✓
Wabco Tractor ABS Mounting Bracket	-	✓
Diagnostic Connector Mounting Brackets	-	✓
J560 Adapter Mounting Bracket	-	✓
J560 Socket Plug Mounting Bracket	-	✓
12V Battery and Box	-	✓
Custom Wiring Harnesses	-	✓
Fuse box, Fuses, Switch, and Voltmeter	-	✓

3.4 Requirements Verification

Figure 3.7 shows the DABSS configuration after retrofitting. Although this does not meet the target configuration in Appendix A.2, the addition of the tractor controller, valves, power, air, and realistic J1939, J1708, and J2497 data is considered satisfactory for SR1. Furthermore, logging activities were conducted that indicate the system is operational for extended periods of time. Thus, SR2 is also satisfied. A photograph of the completed electrical system on the back of DABSS is shown in Figure 3.6. Completion of SR4 is covered in Chapter 6.

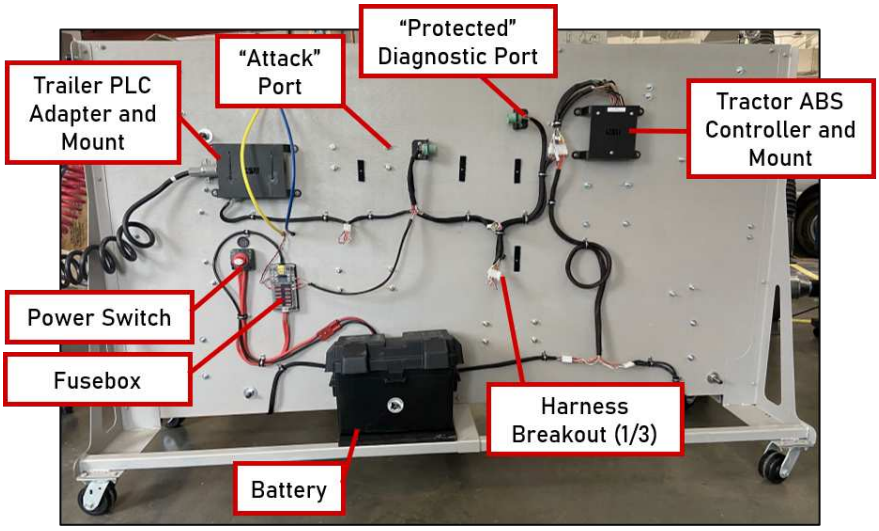


Figure 3.6: DABSS Completed Electrical System

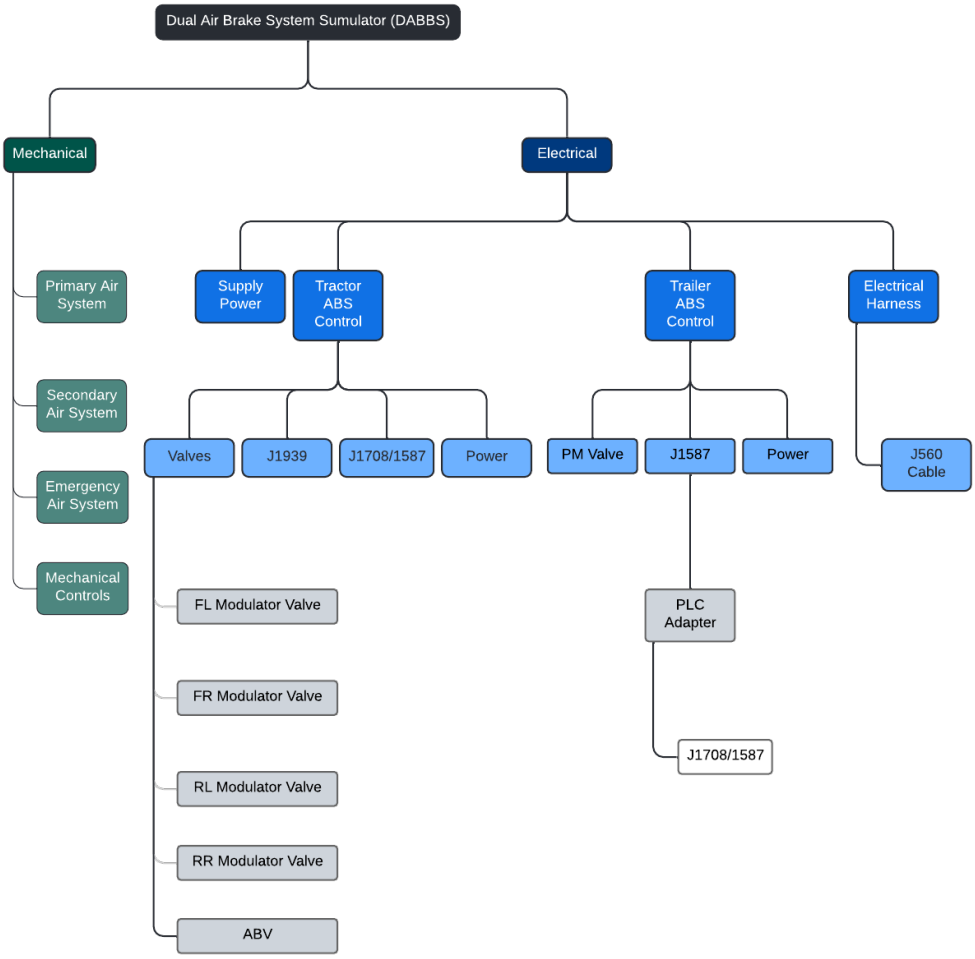


Figure 3.7: "Retrofitted" DABSS Configuration

Chapter 4

Hardware Design, Integration, and Testing

SAE J3061 describes various activities during hardware-level development that do not necessarily apply to the current work. For example, the authors recommend assessing the existing hardware-level vulnerabilities on the vehicle that could expose controller firmware, re-flash utilities, memory, communications networks, and wireless access points. Although some of this analysis was conducted in the previous threat analysis section, it is important to restate that much of these risks fall outside the scope of the current work and cybersecurity goals.

The authors highlight useful mitigation options as well. For example, HSMs and other forms of trusted hardware are usually necessary for performing any cryptographic processing on a real-time system. Although the use of this technology is also observed in novel network defense strategies proposed by [41], [34], and [32], it falls outside the scope of the current hardware design because the system concept does not require cryptography in the first place. To reiterate, the proposed security concept is modeled after the secure CAN transceiver by NXP, which theoretically extends core ECU functionality and could conceivably be integrated as a chip on the ECU itself.

Vulnerability and penetration testing also help determine if the hardware has been secured against creative exploitation. However, the proposed concept serves only as mitigation at the network protocol level and is not intended as a complete security implementation with vigorous hardware-specific detail.

Given these exclusions, a discussion of the hardware goals, requirements, design, implementation, and verification testing remains. These details are provided in the following sections.

4.1 Hardware Concept and Goals

The proposed hardware concept emulates the Ethernet router-gateway found in a typical home. The public network is connected to one port of the device, and the local host is connected to another separate port. The gateway program inspects incoming messages on both lines and forwards them depending on specific rules. Since a programmable gateway node is needed to apply the cybersecurity concept on DABSS, the hardware should resemble the illustration in Figure 4.1. From the illustration, it is clear that at least two J1708 hardware connections are needed to link the host ECU and the shared bus. Since the necessary hardware to enable these operations significantly overlap, the first hardware goal is merely a derivative of SG2 discussed in chapter three.

Figure 4.1: Simple Gateway Model

HWG1: Design a physical hardware prototype with the necessary J1708 data bus circuits to perform gateway, rogue node, compromised node, and observatory communication.

Of course, a simple network connection is insufficient without signal processing and output.

HWG2: Select a computing platform that enables the necessary computation, signal processing, and output to perform gateway, rogue node, compromised node, and observatory operations.

SG4 also informs the hardware goals because the device will ultimately be physically connected to controllers on DABSS. Since the ECUs and valves on DABSS operate on 12-volts, the prototype should be 12V compatible as well.

HWG3: The prototype must be compatible with 12V electrical systems.

SG4 also implies that the hardware design should enable integration to the DABBS network harness.

HWG4: The prototype should provide physical points-of-access to network peripherals for DABSS integration.

4.2 Hardware Requirements

The following requirements were defined using the hardware goals and concept:

[**HR1**] The device shall apply best practices for serial data bus electrical circuits defined in SAE J1708 Section 4.

[**HR2**] The device shall physically isolate multiple J1708/1576 data bus connections.

[**HR3**] The device shall provide physical connectors to access J1708/1587 network peripherals.

[**HR4**] The device shall include configurable GPIO and UART capabilities.

[**HR5**] The device shall be programmable with a standard programming language and IDE.

[**HR6**] The prototype shall be compatible with 12V electrical systems.

[**HR7**] The device shall maintain continuous functionality during extended operation.

[**HR8**] The device shall enable the reading of data available on a standard J1708 vehicle network.

[**HR9**] The device shall enable data transmission on a standard J1708 vehicle network.

4.3 Design and Implementation

The first goal within the hardware design phase was the selection of a micro-computing platform. Since GPIO and UART pins were the only rigid hardware requirements, the selection criteria were mostly unconstrained. On the microcontroller side, many feature-rich MCUs were available that could have satisfied these requirements. These include the ESP32 family, the STM32 family, the RP2040, and some open-source Arduino and Teensy families of microcontrollers, among other less popular platforms.

Larger micro-computing systems with configurable GPIO pins were also available such as the Raspberry Pi 4 and the BeagleBone Black. In the latter case, two PCB shields have already been made for hacking truck networks by research teams at the University of Tulsa and Colorado State University, respectively [42] [43]. Additional hardware modifications to the TruckDuck to enable J1708 serial-data processing are also available [26]. However, these devices have many unneeded components and features to accomplish the goals of the system. Furthermore, the cape and the microcomputer itself are relatively expensive when compared to more minimal designs. Since they are not available for purchase, they also require additional hand assembly and software configuration.

Given the many remaining options, additional criteria were applied to narrow the selection process. This is provided in Table B.1 and Table B.2. Ultimately, the Teensy 4.0 board, the latest iteration of the platform, was selected because it is more capable than many of the latest Arduino boards. It also provided a relatively easy programming environment through the Arduino IDE. Plenty of hardware libraries, examples, and vehicle-related projects exist on the Teensy platform to draw inspiration from as well. The Teensy 4.0 also has CAN peripherals that could be used in future work.

Although the Teensy 4.0 provides up to seven hardware serial connections, none of them alone allows the device to communicate on a J1708-based serial bus. Instead, SAE J1708 provides a circuit schematic for a serial data bus node with passive termination.

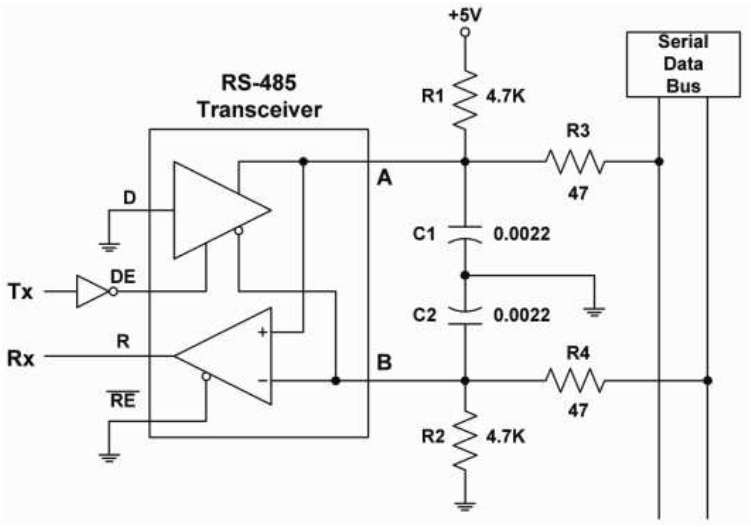


Figure 4.2: Standard J1708 Circuit

This circuit, shown in Fig 4.2, uses standard RS-485 transceivers, a pull-up resistor, a pull-down resistor, and some additional passive components for EMI suppression; though, these latter parts were removed after testing. The transmit and receive lines were connected to the matching pins on the Teensy. Instead of inverting the transmit signal in software, a simple NOT gate IC was added to the circuit. Since the device is expected to service both the host and the network, a duplicate J1708 circuit was added.

Conveniently, the Teensy 4.0 can take a 5.5V maximum input, and the A-lines of the J1708 circuits need to be pulled to 5V. Since a 12V DC supply voltage is expected, a step-down circuit was also added to the hardware design. The voltage regulation circuit was adapted from previous work [34]. It consists of a switching DC-DC regulator and some additional capacitors for smoothing the input power when the device is powered on and off.

Five programmable LED indicators were also added to assist with debugging during software design, troubleshooting during integration, and external monitoring during testing. External connections to the J1708 ports and input power were provided with male and female 2-position screw-jack connector terminals.

A complete circuit diagram is provided in Figure B.1. Components were selected that could be easily integrated on a single breadboard. A bill of materials is provided in Table B.2.

4.4 Hardware Verification and Validation

The components from the BOM were acquired and integrated on a single breadboard. Using the circuit schematic as a reference, continuity was exhaustively verified by probing each connecting wire with a digital multimeter. To begin validation, the hardware requirements were translated into the series of test procedures listed in Table B.3.

Each test item was conducted in the same environment, which consisted of the device under test (DUT), an external 12V power supply, a custom J1708/1587 data cable, and two ECUs (Bendix EC60 and Caterpillar C15 ECM). Only one controller was connected at any given time. Additionally, a digital logic analyzer was fastened to the RX, TX, J1708 A, and J1708 B lines to observe all messages in transit.

In most cases, a simple script was written and flashed to the device via USB before the test. Then the device was placed in the test environment and powered. The outputs were observed from the serial monitor and the logic analyzer software. Subsequently, each test result was analyzed and given a pass (complete) or fail. Robust software development began once all the test items were passed. All test scripts and testing artifacts are available in full in the thesis repository. However, a summary of a few important functional unit tests is provided in the following paragraphs.

4.4.1 J1708 Circuit Functional Tests

The Teensy 4.0 has up to seven UART ports that can be configured for J1708/1587 monitoring. Throughout the project, Serial lines 3 and 4 were defined as the shared and host networks, respectively. The program provided in Listing 4.1 was written in the C/C++ Arduino programming language to read data from serial port 3.

```
#include "J1708.h"
J1708 j1708_3;
void setup() {
  Serial.begin(115200);
  j1708_3.ShowRxData=true;
  j1708_3.ShowTime=true;
  j1708_3.ShowPort=true;
  j1708_3.ShowChecksum=true;
  j1708_3.ShowLength=true;
  j1708_3.ShowErrors=true;
  j1708_3.RxLEDOn=true;
  j1708_3.begin(3);
}
void loop() {
  j1708_3.J1708Update();
}
```

Listing 4.1: J1708 Read Test Script

In sum, a port object called `j1708_3` is instantiated and configured to print in the format desired. Then the object is bound to the Teensy serial hardware at the end of the `setup()` function. In the main `loop()` function, `J1708Update()` is called repeatedly to processes any incoming bytes and split them into messages after each idle time. Then, each message is printed to the serial monitor in the format specified in `setup()`.

When the device was connected to the unit test environment with the Bendix EC60, the output in Figure 4.3a was observed. This message log was saved and compared against logs recorded in a previous session by a DG DPA5 vehicle diagnostic adapter. As shown in Figure 4.3b, the same messages that appeared in the test log also appeared in the VDA log and in the live application. These messages were further verified using the "Asynchronous Data" analyzer function in Logic 2. As shown in Figure 4.3c, the signal confirmed that messages were received and promptly processed in the correct order. It was also discovered in the

analog capture that the bare minimal read procedure was able to ignore transients on the bus that could result in false reads.

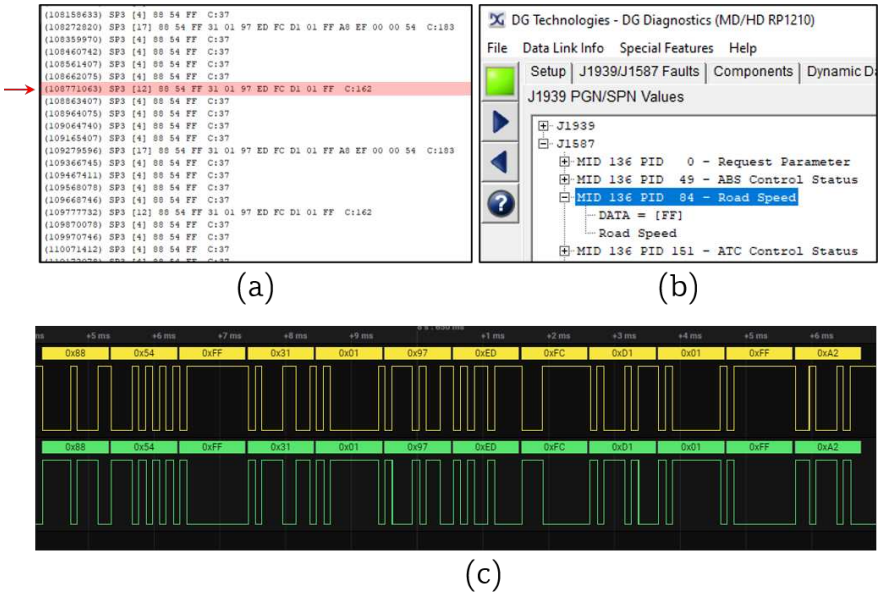


Figure 4.3: Hardware Test Result: Simple J1708 Read from Bendix EC60

A similar script was written to test if messages could be sent on the same bus. The program snippet provided in Listing 4.2 shows the looping portion of the script that transmits random messages once per second on serial port 3 using a constant MID, `senderMID`.

```
1  void loop() {
2    j1708_3.J1708Update();
3    if (msgTimer >= interval) {
4      int messageLength = random(1,20);
5      uint8_t message[messageLength];
6      message[0]=senderMID;
7      for (int i=1; i<messageLength; i++) {
8          message[i] = random(0,255);
9      }
10   j1708_3.J1708Send(message,messageLength,8);
11   msgTimer = 0;
12   }
```

```
13 }
```

Listing 4.2: J1708 Transmit Test Script

As before, a port object called `j1708_3` was instantiated, configured, and bound in the setup function. In the loop, a pre-defined timer triggered a message transmission once per `interval` using the `J1708Send()` function.

When the device was connected to the unit test environment with only the DPA5, the output in Figure 4.4a was observed. This message log was saved and compared against logs recorded in the same session by the diagnostic adapter. As shown in Figure 4.4b, all the same messages that appeared in the test log also appeared in the VDA log. These messages were further verified using the "Asynchronous Data" analyzer function in Logic 2. As shown in Figure 4.4c, the signal confirmed that messages were transmitted once per second with the constant MID (0x77) and random data.

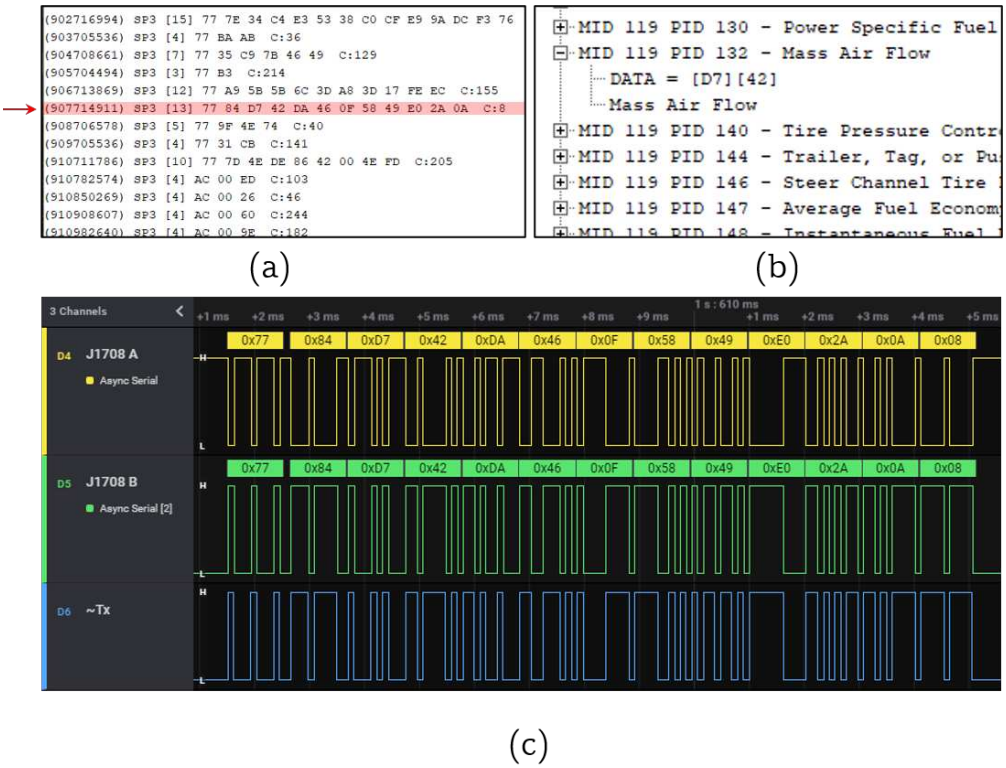


Figure 4.4: Hardware Test Result: Simple J1708 Message Transmit

The same two tests were carried out on host port four by modifying the `begin()` method in both instances. An additional test where the component ID was requested from the Bendix module also elicited a valid response.

4.4.2 Prolonged Operation

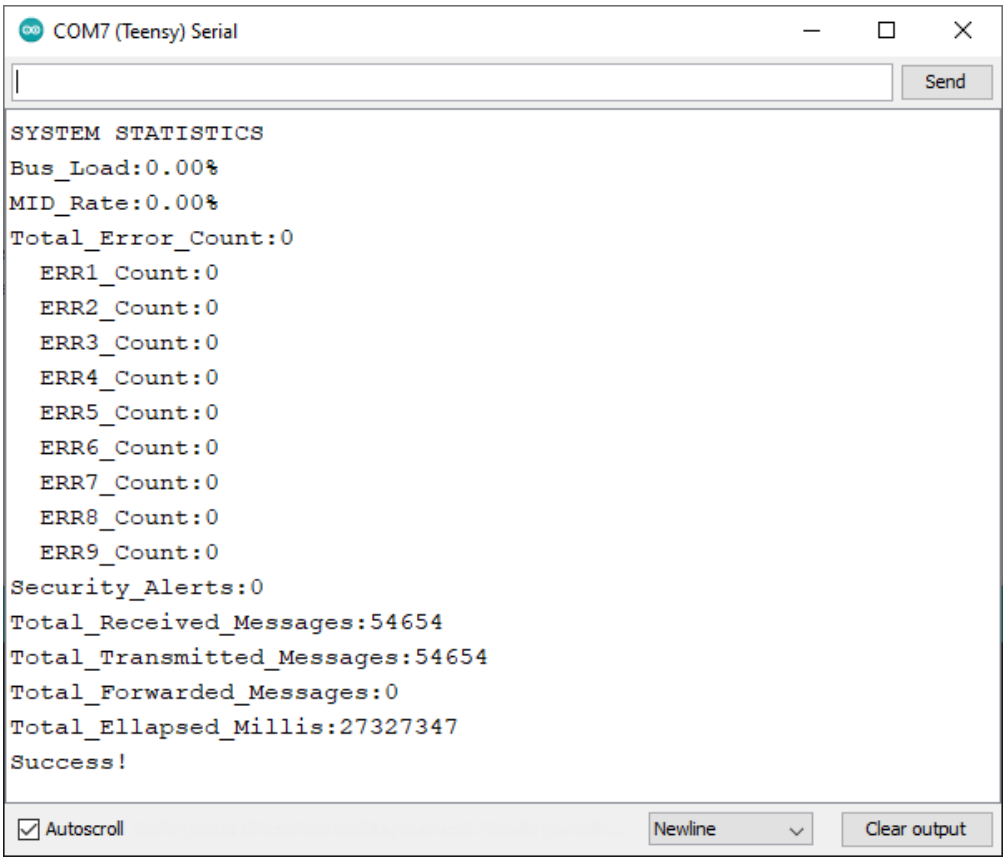


Figure 4.5: Hardware Test Result: Extended Operation Test

To estimate hardware reliability, an extended operation test was written for the prototype. A simple message was sent once per 500ms and added to a total transmit count. This message was echoed back on the receiving line and tallied with a separate counter. This script was left to run for several hours. Figure 4.5 shows reported statistics from such a test after 7 hours. If we calculate the number of expected messages, n as,

$$\frac{t}{\alpha} = n$$

where t is the elapsed time, α is the time between message transmissions, observe that the prototype sent and received 54,654 messages as expected without error. Thus, it is reasonable to accept that the prototype satisfies HR7 as designed.

4.4.3 Voltage Regulation Tests

The voltage regulator circuit was designed to ensure smooth power downs when the gateway is disconnected from 12-volts and minimize voltage spikes when it is powered. Additionally, 0.1 μF capacitors are placed near each IC as simple surge protection. To verify this had the desired smoothing effect, the device was placed in isolation, and an analog trace was recorded at the 5V output of the voltage regulator, 3.3V output of the Teensy, and the A and B lines of the J1708 bus.

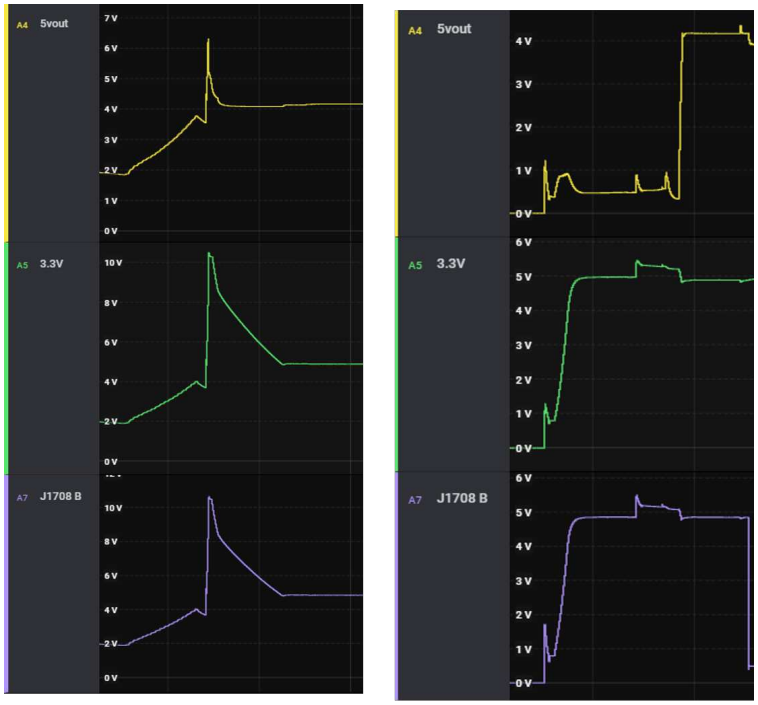


Figure 4.6: Hardware Test Result: Voltage Surge Protection

The capture, shown in Figure 4.6a, shows an analog trace of the device without any protection when it received the 5V output from a bare voltage regulator. An alarming 6V+, 10V+, and 10V+ spike occurred across the device's 5V output, 3.3V, and J1708 B lines, respectively, when it was initially connected. A rapid 350 ms voltage drop occurred when it was disconnected. Conversely, Figure 4.6b showed that the time taken for the line to drop below 1V was extended by over 5s when additional capacitors were added. These measures also cut the maximum voltage spike to 5.6V. This test was critical for avoiding a design that could damage the Teensy hardware. Furthermore, this testing helped ensure that HR6 was satisfied.

4.5 Summary

An electrical design that satisfied hardware requirements HR1 - HR6 was translated into a circuit schematic and implemented on a single breadboard prototype. This prototype was verified against the schematic for any inconsistency. The prototype was also validated through repeated tests using a bench-top test environment and customized run-time programs flashed to the device. Post-analysis of the hardware test results indicated that the prototype satisfied HR7 - HR9, thus completing all the hardware requirements.

Chapter 5

Software Design, Integration, and Testing

This chapter describes the software development process as it was shown in the "V" model in Figure 1.9. Ultimately, the bulk functionality of the gateway was implemented on its embedded software. Thus, many core design and test activities are described in detail. It is worth noting that some cybersecurity-related activities described in SAE J3061 were skipped. For example, the document provides software architecture analysis guidelines for minimizing the exposure of sensitive data in transit within the device. Although these considerations are important, the resulting requirements would have been overly stringent for a simple gateway prototype. In similar instances where recommendations exceed the project's scope, we apply discretion and dismiss these tasks accordingly.

5.1 Software Level Planning

5.1.1 Software Concept

Although the gateway can be modeled as having any number of potential ports, in the simplest case, there are only two: a port for the host and a port for the public-facing network.

Returning to Figure 4.1, we consider the abstract model for a single host. In this model, data input on either side is inspected and processed by the gateway as it is received. If a response is warranted, the gateway outputs any necessary information to the appropriate channel. The receiving, processing, and transmitting tasks can be carefully divided into separate functions and collected in a single library.

Using the library utilities, more nuanced behavior can be designed and written into a run-time script. For example, when the hardware is intended to function as a gateway, a gateway script and the library can be compiled and flashed to the device. Alternatively, if rogue behavior is desired, a different script where the same hardware attempts to spoof

messages can be flashed to the device. Thus, combining the library and script should allow us to achieve the system and cyber goals.

Additionally, suppose the gateways were programmed with a simple command and configuration API. In that case, the user could also write external scripts to automate specific tasks, log information, monitor network statistics, or control behaviors. These scripts would be especially useful during verification and validation testing. For example, a simple test conducted through serial commands using a Python script would eliminate the need to reflash the hardware with a custom run-time script each time new behavior was desired. This should also reduce the test time and some of the human oversight.

5.1.2 Development Tools

Out of the many software development options for the Teensy 4.0, the Arduino IDE and Teensyduino were chosen to quickly compile and flash sketches to the device. The Arduino C/C++ language also supports functional modularity, abstraction, and determinism, which are good characteristics in any embedded programming language. Python and Jupyter Notebooks were used to interact with the gateway serial API and perform many of the unit and integration tests.

5.2 Software Requirements

Before listing the requirements, there are a few definitions that need more context and clarification. A blocklist is defined as a set of discrete MIDs that have been previously determined to be associated with malicious activity. An allowlist is defined as the inverse set of discrete MIDs (i.e., those that have not been previously determined to be associated with malicious activity). The union of these sets is the set of all 256 possible MIDs. In software, these are both implemented together as a boolean-array ACL. The subsequent requirements have been topically grouped on behalf of the reader.

Network Related Requirements

In light of the relevant standards, the following software requirements are defined regarding network activity:

[**SWR1**] The software shall apply best practices for serial data bus network access defined in SAE J1708 Section 5. This includes methods for message delimiting, bus synchronization, access, and contention.

[**SWR2**] The software shall count the following network error(s) at each port: transmit collision and general transmit error (i.e., disconnected).

[**SWR3**] The software shall apply best practices for serial data bus messaging protocol defined in SAE J1708 Section 6. This includes restrictions on message format and length.

[**SWR4**] The software shall count the following messaging error(s) at each port: invalid checksum, invalid message length.

[**SWR5**] The software shall provide transport protocol functionality defined in SAE J1587 Appendix B. This protocol shall be used to segment and reassemble any data payload that causes a message to exceed the maximum length when transferred over the J1708/1587 network.

General Functional Requirements

Given the software concept, the following software requirements are defined regarding general functionality:

[**SWR6**] The software shall provide an API for accessing device information, network and messaging statistics, sending messages, and applying gateway configurations.

[**SWR7**] The software shall provide a configurable access control list for each network port.

[**SWR8**] The software shall provide the following configurable security parameters: maximum busload, maximum busload occurrence count, and host MID.

Security Related Requirements

In light of the security goals and concept, the following software requirements are defined regarding J1708 network security:

[**SWR9**] The software shall provide an external indication that a security anomaly has been detected (i.e., LED indication).

[**SWR10**] The software shall not forward any message with a MID on a network port blocklist.

[**SWR11**] The software shall count each instance a message MID matches the current configured MID of its host on a network port.

[**SWR12**] The software shall notify the network with the designated alert message each instance a message MID matches the current configured MID of its host on a network port.

[**SWR13**] The software shall notify the network with the designated alert message each instance a rogue node is identified.

[**SWR14**] The software shall count each instance a "Message Spoof" alert message is received for a specific MID.

[**SWR15**] The software shall add the MID of a rogue node to the network port blocklist if it receives an "Imposter Node" security alert message.

Additional requirements concerning message flooding threats are defined as follows:

[**SWR16**] The software shall continuously monitor the message busload on each network port.

[**SWR17**] The software shall count each instance the network busload exceeds the maximum configured busload.

[**SWR18**] The software shall notify the network with the designated alert message when a continuous high busload is detected and attributable to a specific MID.

[**SWR19**] The software shall notify the network with the designated alert message when a continuous high busload is detected and attributable to the current configured MID of its host on a network port.

[**SWR20**] The software shall add the MID of a compromised node to the network port blocklist if it receives an "Abnormal Message Rate" security alert message.

[**SWR21**] The software shall add the MID of a compromised host to the network port blocklist if it receives a "Compromised Host" security alert message.

[**SWR22**] The software shall count each unique instance an imposter or compromised node is identified.

5.3 Software Architecture

The software architecture centers around a `J1708` class which represents a single J1708 data port. The class exposes attributes and methods that allow the user to configure routing behavior, message processing, security policy, and other controls before and after compilation. The class UML diagram in Figure 5.1 is condensed for brevity but shows many of the important attributes and methods. Visibility is denoted with a "+" for public and "-" for private. Relationships are shown with specialized arrows. The diagram also provides the attribute and method return types after each name.

In most sketches, a host and network object is instantiated to represent each connection. This is illustrated by the two instances `J1708_Host` and `J1708_Network`. The `begin()` method binds the object to a specified serial port and two digital output pins. Two objects

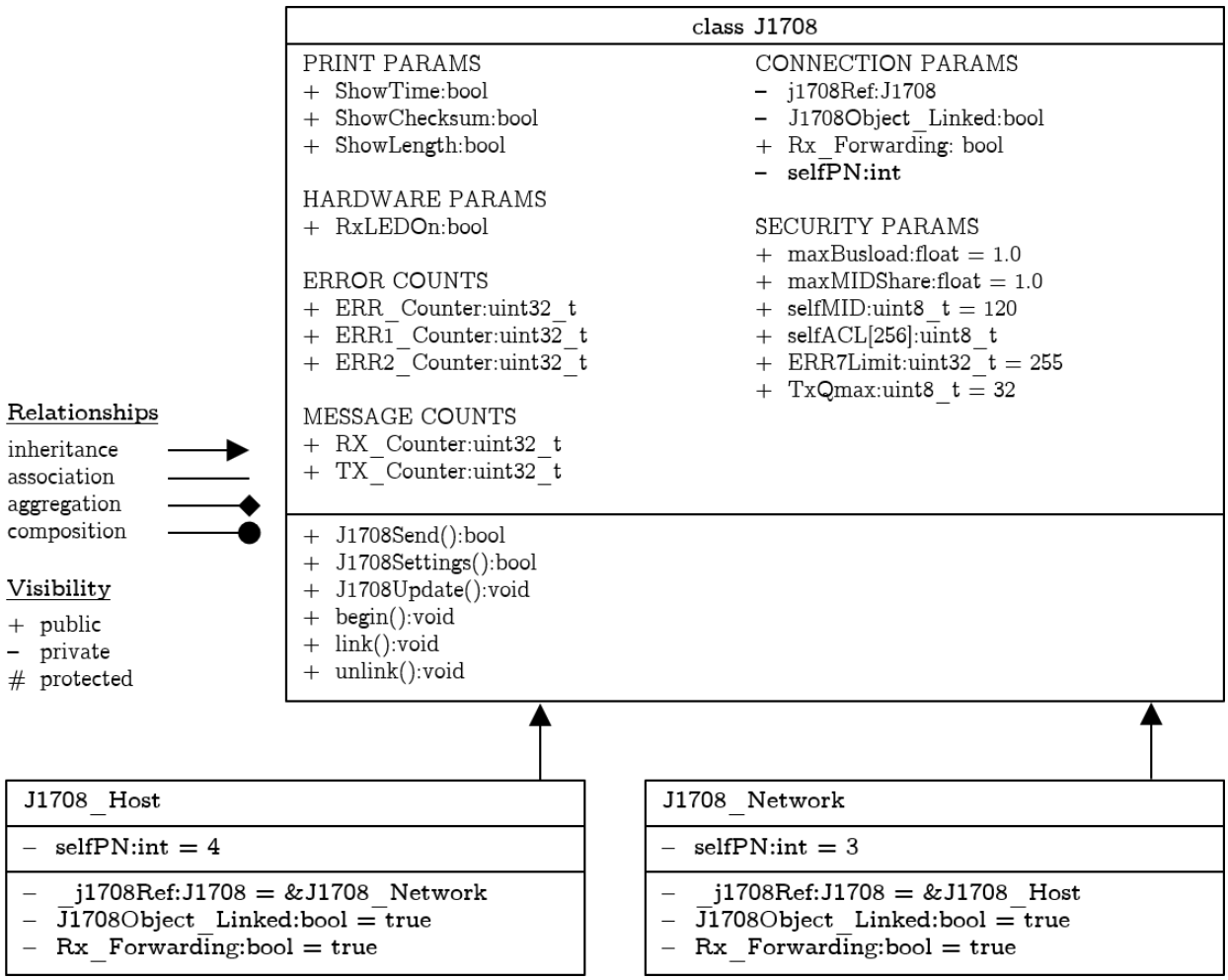


Figure 5.1: J1708 Class Diagram.

can be linked together by passing a reference from one to the other with the `link()` method. When a `J1708` object is linked, it forwards messages received on its connected serial port to the processing routine of the linked object. Vice-versa, performing the same procedure with the other object creates a logical two-way linkage.

Only a handful of attributes are available for manipulation outside the class. Some notable examples include error counts, print settings, and the access control list. However, commands can be sent to `Serial1` during run-time to make changes on the fly. This class, along with some string parsing utilities, is packaged in a single header file that serves as the library.

With an understanding of the classes themselves, we can now broadly define the software architecture. We begin using the specific case previously illustrated in Figure 4.1. The component diagram in Figure 5.2 shows how two `J1708` objects could be configured to create a gateway between an ECU and the shared J1708/1587 network. First, two objects must be instantiated, bound to their respective hardware serial ports, and logically linked together. Once completed, each object should have three primary input signals and three output signals. In the diagram, signal types are denoted with different arrowheads. A signal with two arrowheads indicates that it flows abstractly in both directions. However, no specific message structure or format is implied by these signal types alone. Instead, specific colors are used to illustrate where distinct formats occur.

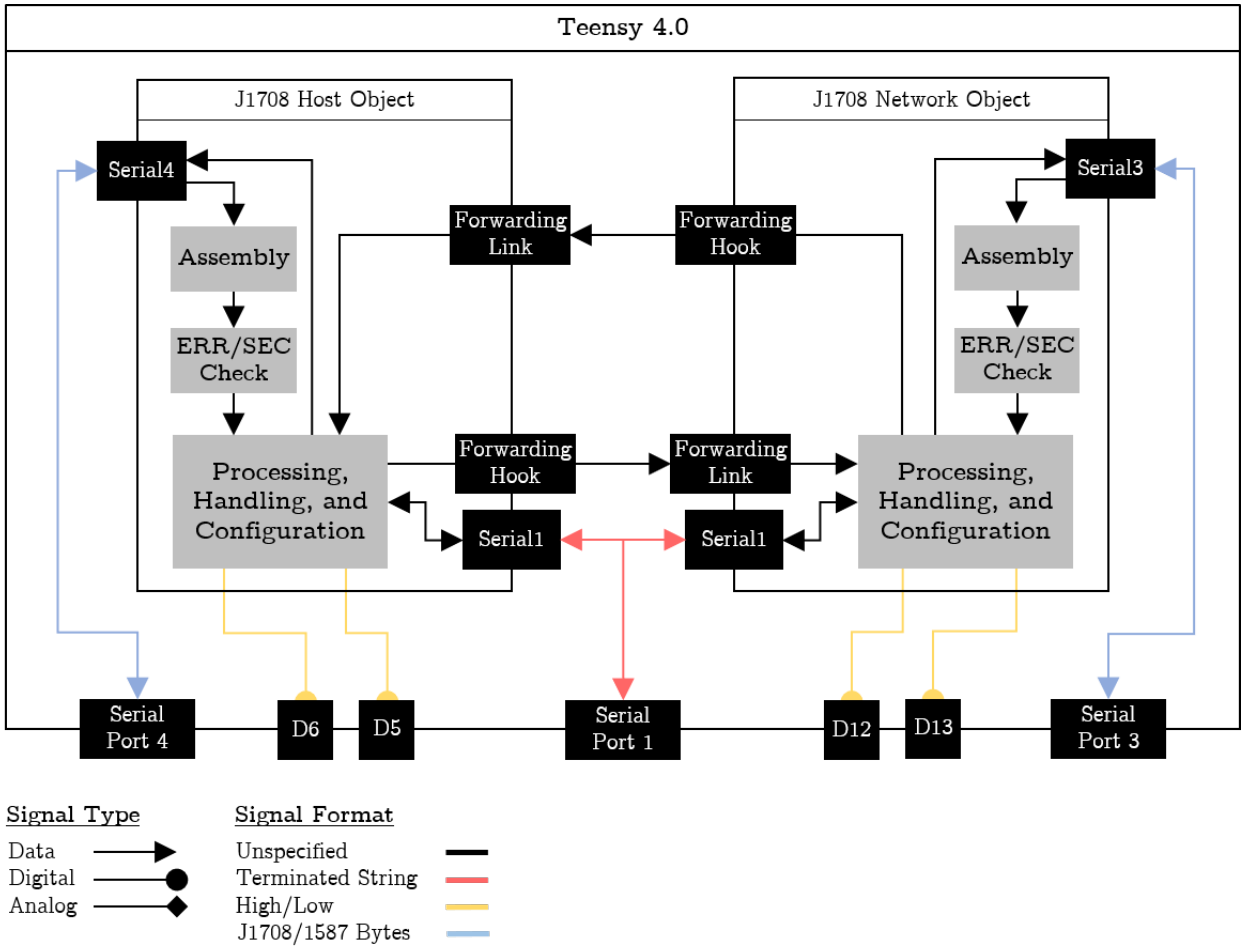


Figure 5.2: Gateway Software Architecture Component Diagram

To better understand how J1708/1587 messages flow through the program, suppose message bytes are received at `Serial3` from the network in Figure 5.2. These bytes get collected and assembled by the network `J1708` object in the order they appear until an idle time (T_i) is observed. Once a complete message is assembled, it is checked for errors and compared to the security policy.

Ideally, J1708/1587 messages are passed throughout the program using a common data structure like that shown in Figure 5.3. The `struct` contains all the necessary metadata associated with the message as it is passed through different processes.

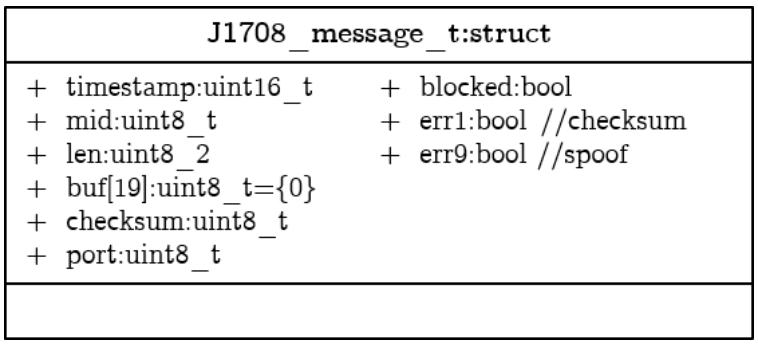


Figure 5.3: j1708_message_t Class Diagram

In cases where errors or security violations are identified, the message can be flagged for subsequent processes to either handle (i.e., increase the error counts, notify the network) or ignore. This helps prevent a section of software from failing due to malformed or corrupted input. Theoretically, this should also reduce the time spent in the main loop if a faulty message is caught early during processing. In the current architecture, ten total errors are logged. Table 5.1 provides the type, description, and numbering convention of each.

SAE J3061 recommends preserving and protecting these error counts and messages to aid in forensic analysis of intrusion attempts. The Teensy 4.0 has 1080 bytes of emulated EEPROM for saving the error counts periodically. However, this is largely insufficient for saving more than a few dozen anomalous messages. Therefore, only error counts can be stored. Currently, protecting the EEPROM from tampering is considered well outside the

Table 5.1: Table of Software Error Types

No.	Type	Description
ERR1	Message	Checksum Error
ERR2	Software	Receive Buffer Overflow
ERR3	Software	Transmit Buffer Overflow
ERR4	Network	Transmit Collision Error
ERR5	Network	General Transmit Error
ERR6	Network	High Busload Warning
ERR7	Security	Spoofed Message Warning
ERR8	Security	Rogue Node Detected
ERR9	Security	Abnormal Message Rate Detected
ERR10	Security	Compromised Node Detected

scope of the project. In future work, messages could be stored with an SD card or other external storage.

If the message is free of any errors, it can be passed to various processing units and handled based on the current user settings. This includes printing to the serial monitor, turning on LED indicators, and forwarding to any connected J1708 links. Forwarded J1708 messages do not go through additional checks and are sent when the line at the reference becomes available. In the current example, this is `Serial4`.

If we abstract the internal components further, we obtain a general object that can be easily applied to other use cases in future work. Furthermore, this should allow every hardware serial port on the Teesny to be connected to a J1708 bus and logically combined or monitored in any way, as illustrated in Figure 5.4

Similarly, data sent by a USB communications port or the Arduino IDE serial monitor is collected by an interrupt service routine (ISR) and assembled into an Arduino `String` upon reception of a termination character. Relevant commands are passed to the appropriate `J1708` object, where it is handled by the command processing method `J1708Settings()`.

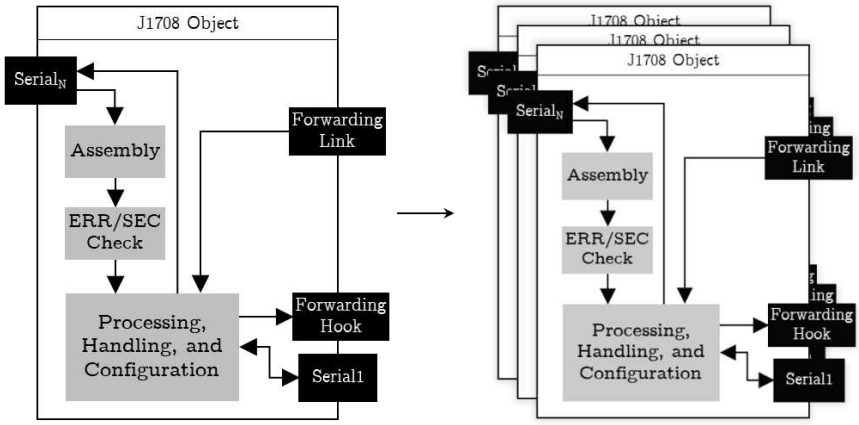


Figure 5.4: J1708 Object Multiple Instances

5.4 Unit Design, Test, and Verification

5.4.1 J1708 Object Setup

Three methods were created to initialize a `J1708` object and create logical connections between ports: `begin()`, `link()`, and `unlink()`. When passed a port number, `begin()` initializes the default LED indicators and binds to the serial port specified at the default baud rate of 9600 bits per second. The port number is saved as a private attribute.

`link()` takes a `J1708` object memory address as a parameter and saves it as a reference. It also updates the link status of the object to `true`. By default, it will forward all unblocked messages. The `unlink()` method does the opposite and will delete the previously saved memory address.

These functions were tested in isolation using a dummy class object and a hardware loop-back. During testing, a serial connection was successfully started within the object. Storage of an object reference was also successfully executed and subsequently freed using `unlink()` to prevent memory leaks.

5.4.2 Configuration and Test Controls

A serial command interface was designed to help facilitate the configuration of the gateway. Additionally, this API helped support testing efforts throughout all development and testing phases.

In sum, commands are entered via the USB serial connection from the Arduino serial monitor or by another program connected to the port. The gateway delimits messages by noting the reception of a newline character. In the run-time script, this message is preprocessed and passed to a specific `J1708` object using the `J1708Settings()` public method where the API is partially implemented. When a J1708 object parses the command, it will perform an action using one of its built-in utilities or simply update a specific attribute (`RxLEDOn`, `selfMID`, `showLength` etc.).

Since serial parsing had already been tested to ignore malformed or extreme input during hardware testing, the remaining unit tests revolved around ensuring the commands were properly segmented, indexed, interpreted, and compared. Thus, when a new command was implemented, all standard options and values were individually tested before continued use. Figure 5.5 shows the properly handled output from the `help` option of the `j1708config` command and a list of available subcommands and options.

5.4.3 Receive J1708

Reading and writing messages to the network are two fundamental functions of the gateway. Repeated calls to `J1708Rx()` check the bus for available characters and collect them in a buffer in the order they are received. Appendix B.3 shows a logical flowchart of the function. The routine keeps track of the total byte count for bus load estimation and raises a flag as a message is received to prevent preemptive message transmissions. A message is delimited after an idle line on the receiving wire is observed for 12-bit times. The checksum counter (ERR1) is incremented when any message with a bad checksum is identified. Under normal network conditions, when the gateway is not transmitting, this can occur if two con-

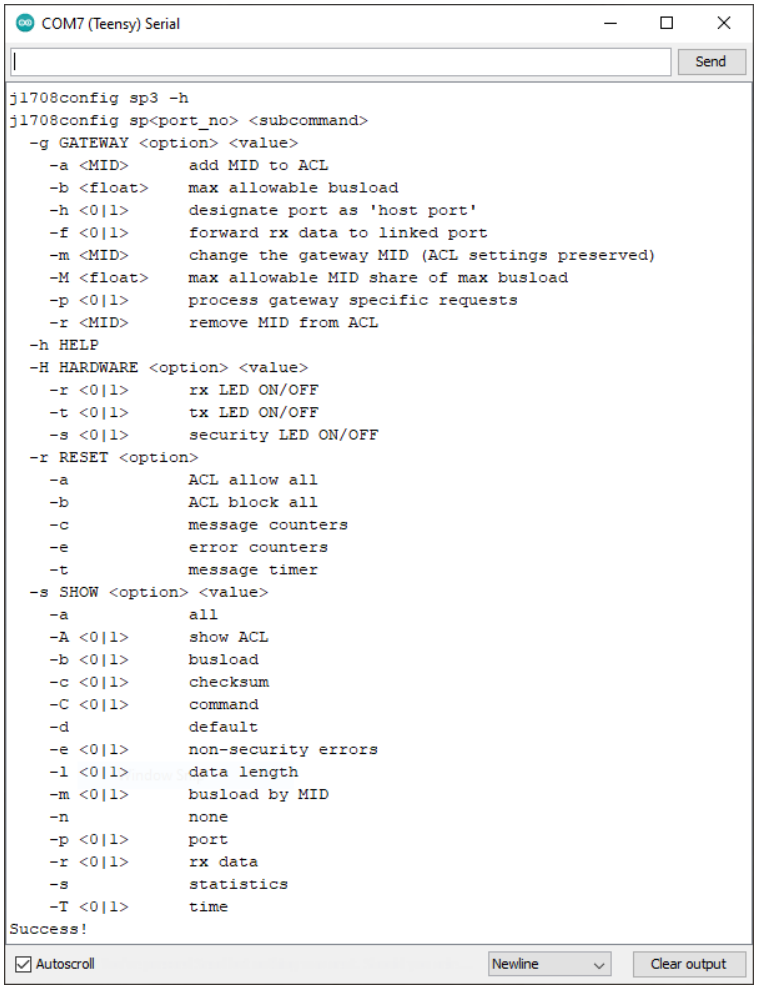
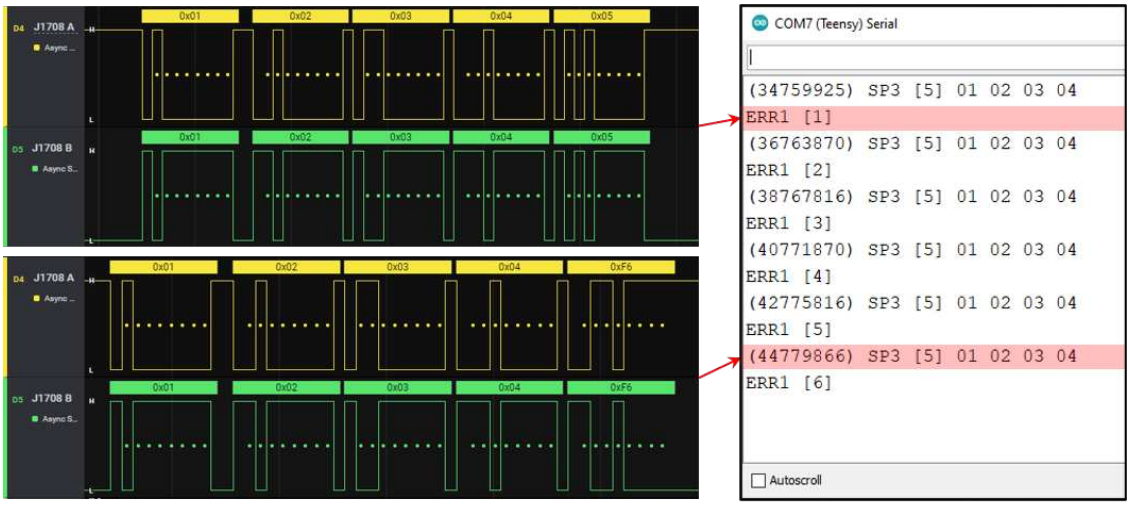


Figure 5.5: J1708 Prototype Gateway Serial API

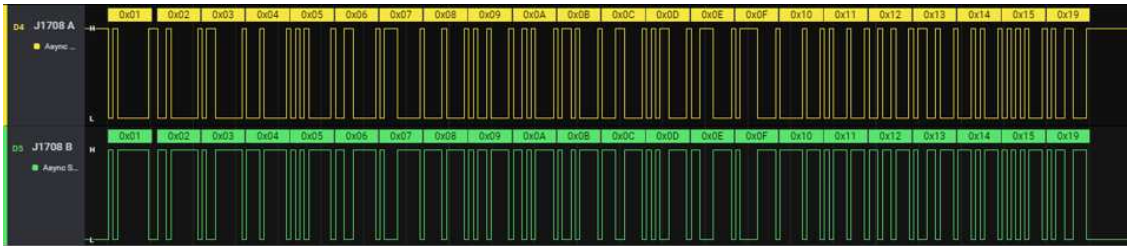
trollers attempt to access the bus at the same time. The message length is returned after each successfully received message. During run-time, this value is used to read the bytes from the receive buffer.

The previous testing during hardware validation confirmed the program's ability to read and delimit messages. To confirm ERR1-detection, two identical devices were connected to the same J1708 bus. One device was programmed to alternately send a message with a faulty checksum and a message with a correct checksum each second. The other device listened and reported to the serial monitor. The signal capture in Figure 5.6a confirmed the presence of valid and invalid messages on the bus. The result in Figure 5.6b showed that ERR1 and non-ERR1 messages were reported as expected.

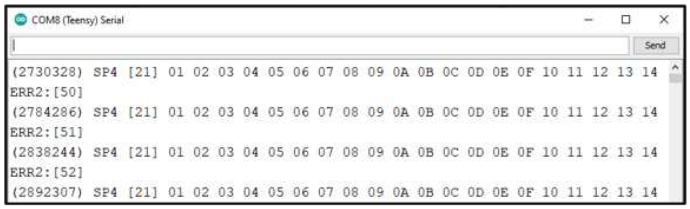


(a) (b)

Figure 5.6: ERR1 Unit Test Result: (a) Digital signal capture of a J1708 message with bad checksum (top) and the same message with a proper checksum (bottom). (b) Reported messages and errors from the prototype gateway.



(a)



(b)

Figure 5.7: ERR2 Unit Test Result: (a) Digital signal capture of a 22-byte J1708 message. (b) Reported messages and errors from the prototype gateway.

Under normal circumstances, the gateway can receive a message longer than 21-bytes and report it on the bus assuming its length is below the received message buffer size, and it has a valid checksum. By default, this buffer size is set to 21. An ERR2 counter is incremented if the receive buffer ever fills up. To test ERR2 handling, the receive buffer size

was reduced to 21-bytes. Using the same hardware setup as before, a 22-byte message with a valid checksum was rapidly transmitted to the bus defying network protocol. Meanwhile, a valid 21-byte message was also transmitted rapidly to the bus. As shown in Figure 5.7, unit testing confirmed that ERR2 instances were properly reported and handled and did not affect the reporting of other network activity.

5.4.4 Schedule and Transmit J1708

Messages are scheduled with `J1708Send()` and transmitted with the `J1708Tx()` methods. `J1708Send()` takes a data array (excluding the checksum) under 21-bytes, the message length, and priority as parameters. When called, it adds the input to a circular buffer and updates the transmit queue counter. When the receiving line is not busy, `J1708Update()` will check the queue for a pending message and wait for the required bus access period (T_a) before calling `J1708Tx()` to transmit the message. This transmission delay period is implemented in another part of the source code and is calculated with the message priority (P), the idle time (T_i), and a single bit time (T_b) using Equations 5.1 and 5.2. If at any point the transmit queue buffer is full, an ERR3 flag is raised and reported on `Serial1`. Early experiments showed that a default queue size of 32 messages was more than sufficient for delivering messages at the maximum transmission rates.

$$T_a = T_i + (2T_b)P \tag{5.1}$$

$$T_i = 10 * T_b \tag{5.2}$$

Equation 5.3 defines a single bit time where β represents the network baud rate.

$$T_b = \frac{1}{\beta} = \frac{1}{9600} = 104.1\bar{6} \text{ ms} \tag{5.3}$$

Appendix B.4 shows the `J1708Tx()` logical flowchart. Before transmitting a message, the function sends only the MID and checks if another controller attempted to access the bus.

If the device receives the same MID within an inter-character time (defined in the standard as $2T_b$), it will calculate and append the payload checksum and then deliver the rest of the message characters [44]. If the device encounters a different MID, a collision has occurred. The program will drop the message, increment the ERR4 counter and report it on `Serial1`. If no MID is received, then an ERR5 counter is incremented to indicate a potential bad connection. A busy flag is raised during transmission to prevent other procedures from interrupting. The function raises another flag to indicate that the next message it will see is an echo.

Past testing during hardware validation confirmed the program's ability to write bytes to the bus. To verify ERR3 detection, two tests were conducted in which the device attempted to queue a message with `J1708Send()` once per second. In the first test, a sufficiently sized message queue was defined using the standard transmit delay time. The result in Figure 5.8a showed a message transmission occurred at the expected rate. In the second test, the maximum message queue was temporarily reduced to two messages, and the transmit delay time was increased to 2 seconds. The result in Figure 5.8b indicated a message count reduction compared to the previous test over the same period of time. ERR3 occurrences were reported as expected.

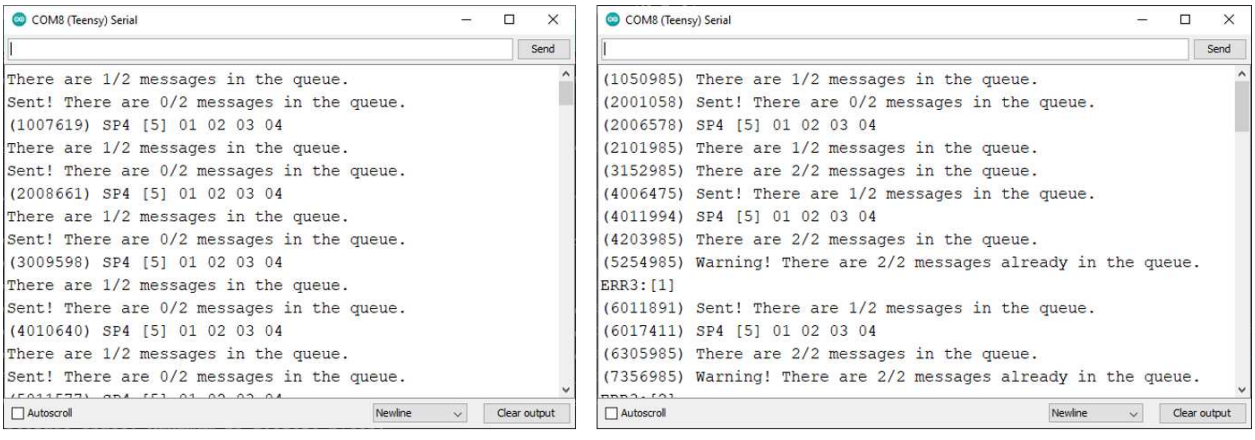


Figure 5.8: ERR3 Unit Test Results: (a) Normal message queuing, transmission, and de-queuing. (b) Message queuing, queue overflow, and ERR3.

To confirm ERR4 detection, two identical devices were connected to the same J1708 bus. One device was programmed to send messages at a high message rate, while the other attempted to send messages at a nominal rate to induce artificial collisions. Bus traffic signals were recorded using the Saleae logic analyzer. As shown in Figure 5.9, a small $150\mu s$ inter-character delay occurred between the MID and the rest of the payload when a message was sent by the gateway. This likely occurred because of the small calculation that is performed to detect collisions after every MID is sent. However, this delay is considered within the $208\mu s$ delay allowed within the standard. Consequently, when a collision eventually occurred, it was successfully caught and reported on `Serial1` by the offending node as both an ERR4 and ERR1. Both devices also reported the same expected number of ERR4s when strict message timings were applied in subsequent tests. Independent observations made with analog captures confirmed the occurrence of each reported collision in the most successful tests.

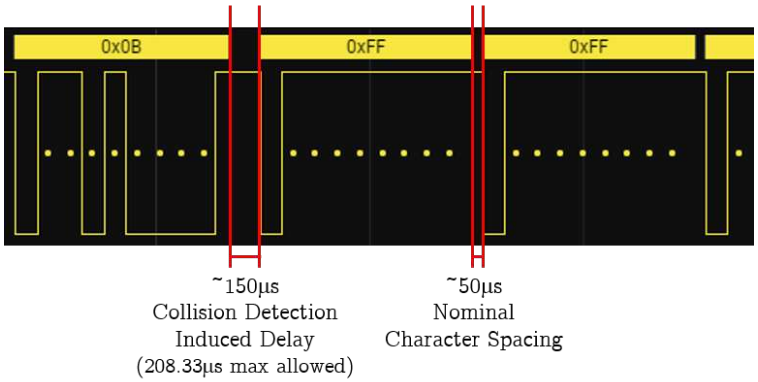


Figure 5.9: Induced Inter-Character Bit Time Delay

To verify ERR5 detection, the transmit line was disconnected from the bus, and a message was queued with `J1708Send()` via the `Serial1` API. Every attempt resulted in an ERR5 as expected.

5.4.5 Security Alerts and Reporting

The software requirements state that the gateway shall send a security alert message if it detects anomalous bus activity. This could be achieved with the existing semantics for fault notification defined in SAE J1587 shown in Table 5.2. For instance, PID 194 (0xC2), known as the "Transmitter System Diagnostic Code and Occurrence Count Table," is used by any transmitting device to notify other components on the bus of the diagnostic condition on the transmitting ECU. This message contains the MID, a list of diagnostic codes and occurrence counts, and the PID or subsystem identification number (SID). A single diagnostic code byte contains a fault-mode identifier (FMI) which describes the type of failure detected among a list of mostly pre-defined methods. Together, this information can be used by service personnel to determine which subsystem is failing and how it is failing.

Table 5.2: SAE J1587 PID 194 Definition

{x, C2, n, a, b, c}

x, message identifier (MID)
C2, DTC and OC table (PID)
n, payload bytes
a, SID or PID of the diagnostic code
b, diagnostic code character
c , occurrence count

In the event message spoofing or flooding is detected, MID 163 (0xC2), "Vehicle Security", could be used in conjunction with PID 194 to deliver a periodic security notification. However, there are a few challenges. First, the existing definition for PID 194 provides a maximum occurrence count of 255. If a threshold value is used to confirm the presence of a rogue node and that value is set to a larger value, then the recipient will need to use PID 195 (0xC3), "Diagnostic Data Request," to obtain the latest counts. A subsequent message will also have to be sent using PID (0xC4), "Diagnostic Data/Count Clear Response." However, pre-existing manufacturer definitions could exist for the response that does not

provide sufficient information for detection decisions. There is also the complication that all the gateways would have to transmit the notification using the same identifier, MID 163. Finally, the overhead of transmitting multiple diagnostic code messages is undesirable since there is no means of identifying the offending MID in a single message under the existing scheme.

Accordingly, a unique PID is proposed to capture relevant security information for these specific cases. Within the SAE J1587 specification, PID 255 (0xFF) is reserved for any "Page Extension" purposes. This parameter allows additional PIDs to be defined outside the initial 255. Certain ranges of PIDs follow a given pattern. For example, the first 128 PIDS on all pages are reserved for parameters requiring only 1-byte of data. PIDs 128-191 (64 total) are reserved for 2-byte parameters, and PIDs 192-255 (62 total) are set aside for multi-byte parameters.

Given the minimal payload requirements for a security notification (1 message) and the established patterns for parameter definitions within the standard, PID 762 (0xFFFFFA) has been selected as the parameter identifier to notify gateways of security occurrences. A formal definition is provided in Appendix B.5. With this definition in place, individual gateways can share updated information about the security status of their respective hosts and support distributed intrusion detection on the shared bus.

5.4.6 Message Firewall

A public access control list is created within each `J1708` class when it is instantiated. As indicated in Figure 5.1, this is implemented as a boolean array. By default, the gateway claims MID 120 (0x78) from the unassigned J1708 identifier range when it is powered. It is the only blocked identifier on its ACL.

After a complete message is received, a run-time procedure first checks if the message MID is blocked using the ACL. If the MID is not blocked and another `J1708` object is linked, the message will be forwarded and passed along to the rest of the program. Conversely, if a MID

is blocked and a `J1708` object is linked, the message will not be forwarded. If the message MID matches `selfMID` (which should be configured to that of its host), an ERR7 counter, indicating a spoofed message, is incremented, and a "Message Spoof Alert" is transmitted on the originating bus.

Gateways that receive the message spoof alert forward the metadata to their handling procedure which synchronizes the ERR7 occurrence count for that MID. If successive ERR7s exceed a pre-configured threshold within the legitimate host gateway, an "Imposter Node Alert" message with the MID of the offending node is also broadcast to the bus. Consequently, the ECU or node associated with the MID is considered rogue, and all recipients of the alert will add the MID to their blocklist.

After an imposter node alert is sent, subsequent message spoofing is no longer reported with the message spoof alert. Rather, an imposter node alert is sent every 10 seconds.

The ERR7 counter was tested using two prototypes that functioned as a gateway and rogue node. The rogue node was programmed to send a message periodically with the MID of the gateway and was connected to the "shared" bus of the gateway node. The next test log shown in Figure 5.10b was translated into the sequence diagram for clarity. The results showed that the trusted gateway caught the first instance of message spoofing and reported the incident with the appropriate network security notification type. The gateway reported subsequent incidents as expected. The ERR7 counter also reflected these incidents in the tallied network statistics on both devices.

The data capture and translated sequence diagram in Figure 5.11 showed the trusted gateway elevated to the imposter node alert once the ERR7 count rose above the configured maximum of 256 occurrences. The trusted gateway continued to send periodic imposter alerts as expected at a fixed rate.

An analog capture and the recorded message logs showed the absence of message traffic on the "host-side" bus of the trusted gateway. This verified that none of the spoofed messages were forwarded to the host at any point during the test.

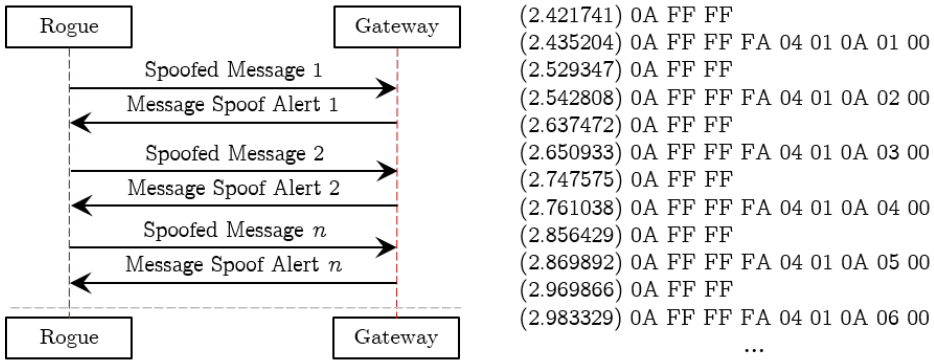


Figure 5.10: ERR7 Unit Test Results: (a) Message spoof reporting sequence diagram. (b) A portion of the test log from the gateway.

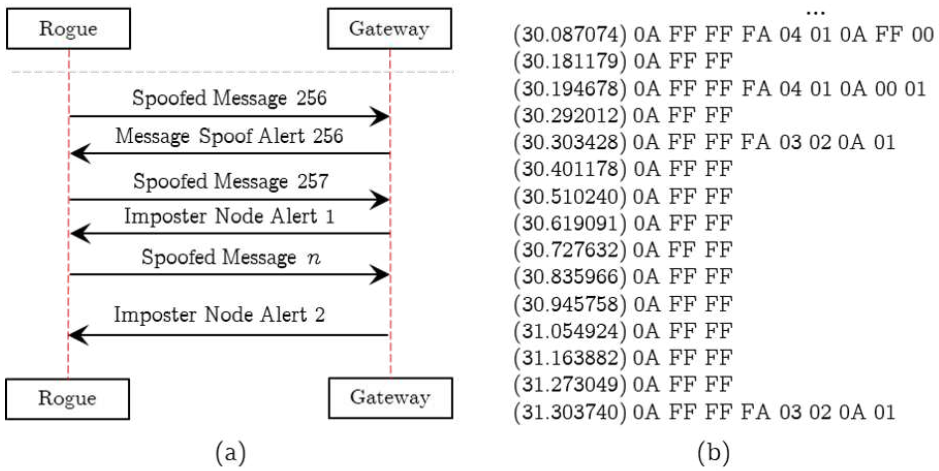


Figure 5.11: ERR8 Unit Test 1 Results: (a) Rogue node identification sequence diagram. (b) Imposter alert portion of the test log from the gateway.

The test was repeated with the addition of a second gateway. This passive node was configured with a MID 0x0C and message forwarding to its host network. The sequence diagram and bus capture in Figure 5.12 showed that rogue messages were received at the "shared" port (SP3) and forwarded to the "host" port (SP4) until the passive gateway received an imposter alert message containing the address of the spoofed address 0x0A. The device and network statistics puled from the device after the test also confirmed that the ACL, ERR7, and ERR8 counts were updated correctly across all three devices.

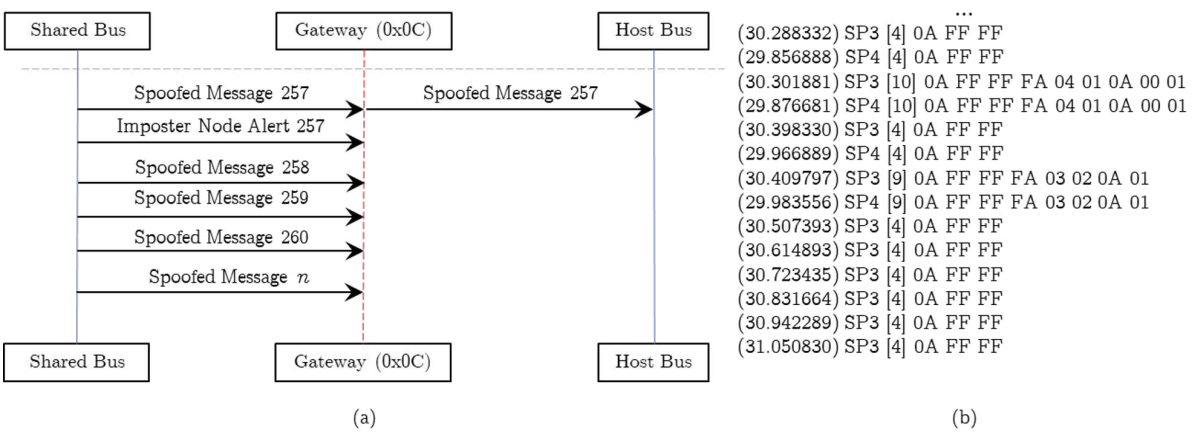


Figure 5.12: ERR8 Unit Test 2 Results: (a) Sequence diagram of message forwarding and subsequent blocking by a gateway observer upon alert reception. (b) A portion of the unit test log.

5.4.7 Bus Load Monitoring and Message Rate Enforcement

The amount of network traffic at any given time is measured to help identify message flooding attempts. This continuous tracking is accomplished by counting the total received bytes within a single second and comparing that to the theoretical maximum, B_{max}. To obtain this maximum, we can use Equation 5.4 where β is the network baud rate, N_{max} is the maximum message size permitted by the protocol, N_c is the number of bits in a single character, and Ω is the message overhead.

$$B_{max} = \left\lfloor \frac{\beta}{N_{max} * N_c + \Omega} \right\rfloor * N_{max} \tag{5.4}$$

Section 6 of the J1708 protocol states that no message shall exceed 21 characters. A single character in J1708 is 10 bits long (1 start, 8 data, and 1 stop bit). By definition, the highest priority messages incur the lowest overhead. Therefore, inferring from 5.4 and 5.1, we obtain the value for B_{max} on a J1708 bus as follows:

$$B_{max} = \left\lfloor \frac{9600}{(21 * 10) + (10 + 2)} \right\rfloor * 21 = 903 \text{ bytes}$$

Although more bytes could be sent when violations to the established protocol are considered, we assume that anomalies can be detected for any activity within the 903-byte maximum.

When the run-time routine is called during `loop()`, the total number of bytes received within the calculation period, denoted as B_{total}, are used to derive the latest busload measurement using Equation 5.5.

$$\gamma_i = \frac{B_{total}}{B_{max}} \tag{5.5}$$

A dependable measure of γ_i is crucial for detecting abnormal network activity such as message flooding. Consequently, the first unit tests were designed to tune the traffic sampling rates to maximize responsiveness under various traffic patterns. Early testing indicated a 1-second sampling rate provided an acceptable balance of responsiveness to changing bus activity and accuracy to actual bus loads measured with an analog probe. Lower sample rates (<0.75s) were often too responsive and caused saw-toothing measurements about the average busload value. Low sampling rates were also more prone to aliasing when various periodic messages were tested.

A second test using two nodes was designed to gauge the γ_i-sampling accuracy of the prototype software. One device was programmed to send 100 periodic 21-byte messages for increasing periods approaching 1.25ms ($12T_b$). The digital signal capture shown in Figure 5.13 confirmed the actual message periods during steady state was approximately 1.45ms at its highest. The receiver device reported these messages and its calculated γ_i each period. The plotted test results in Figure 5.14 showed that γ_i was measured slightly higher on average compared to conservative theoretical loads calculated for each message period. It also showed that the error increased as message rates increased. However, the trade-off between responsiveness and accuracy was considered sufficient for these errors to be acceptable. The valleys observed between period increases were caused by programmed 0.5s pauses between message rate changes during testing.

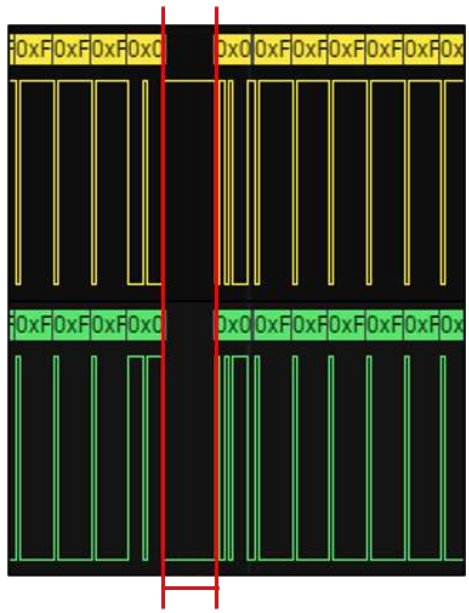


Figure 5.13: Message Time Delay During High Busload Test

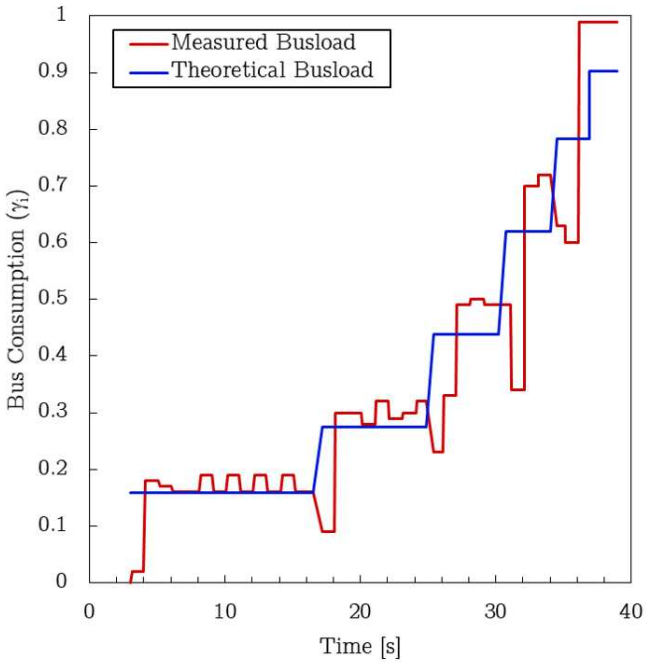


Figure 5.14: Busload Functional Test Result: Theoretical and Measured Busload vs Time

If the measured busload exceeds a configured maximum value at a network port, then an ERR6 is raised and the gateway will begin tracking the relative share of busload attributable

to each MID. This share of busload is denoted as μ_i and is periodically calculated using Equation 5.6. Specifically, μ_i is obtained by summing the message bytes attributable to each MID within the calculation period, B_{MID}, and dividing the fractional total by the current overall total, B_{total}.

$$\mu_i = \frac{\sum B_{MID}}{B_{total}} \tag{5.6}$$

Two unit tests were written in Python and coordinated through the serial API to verify the accuracy of the software implementation. These tests used an identical procedure to those used to verify the busload measurements. In the first test, a single node sent 21-byte messages using MID 0x0A. The recipient calculated γ_i and the μ_i and reported these values through the API during each calculation period. When the γ_i and the scaled MID share, $\gamma_i * \mu_i$, were plotted against time as shown in Figure 5.15, these values overlapped as expected.

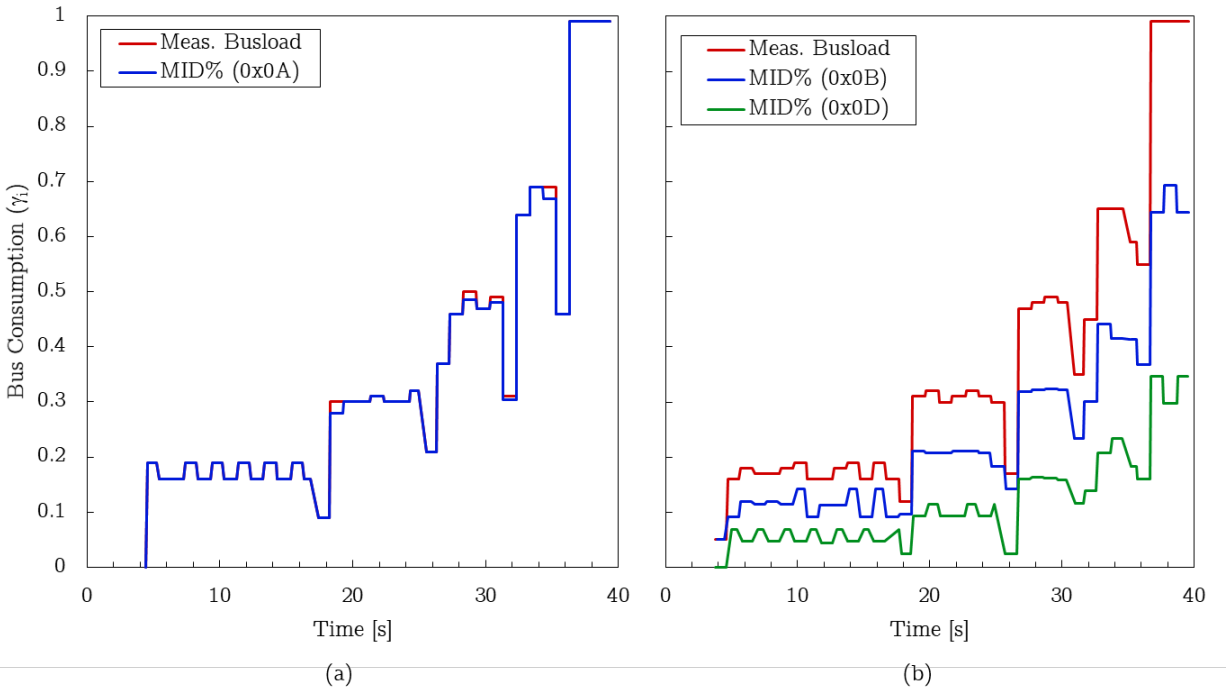


Figure 5.15: Real-Time Busload Share Calculation Tests: (a) One Node - MID 0x0A (100%) (b) Two Nodes - MID 0x0B (66%) and MID 0x0D (33%)

In the second test, the sender sent a combination of two 21-byte messages with a constant proportion over time. When γ_i and the two scaled μ_i values were plotted against time in Figure 5.15, the results showed a steady 2/1 proportion of total busload contributed by MID 0x0B messages and 0x0D messages, respectively.

If a configured number of consecutive ERR6s occur and the share of network activity for any MID exceeds a threshold, an ERR9 is triggered, and the MID is blocked. N_{CE6} and μ_{max} represent these values respectively and are both configurable through the gateway API. By design, N_{CE6} is sampled every 0.5s. The first occurrence of an ERR9 initiates an "Abnormal Message Rate Alert," which eventually causes all gateways to permanently block the MID. Alternatively, if a port has been configured as a "host port," an ERR 10 is triggered instead. The occurrence of an ERR10 also causes the "Compromised Host Alert" to be sent on a non-host port.

To test this functionality, a closed loop experiment was conducted using three devices on the same bus controlled through the serial API. One device, programmed with MID 0x0B, sent messages at a high rate. Another device, configured with MID 0x0C, sent messages at a low transmission rate. A third device, configured with MID 0x0A, was given the following configuration set: $\{\gamma_{max} : 0.2,\ \mu_{max} : 0.8,\ N_{CE6}{:}4\}$. The first two thresholds appear as horizontal lines on Figure 5.16 when plotted.

After 21s of network activity, an "Abnormal Message Rate Alert" was successfully transmitted by the receiver when the network conditions were met. The vertical line on the plot indicates the time the alert was received by the controller. Each node on the bus interpreted this ERR9 and incremented its security error count appropriately. Subsequent incoming messages were blocked by the software ports that received the notification.

The "shared" and "host" port busload measurements were also captured and plotted. Observation of the results showed a drop in resource consumption on the host bus after messages from 0x0B were blocked at the shared port. Thus, the only remaining busload came from MID 0x0C during the last 10s of the test.

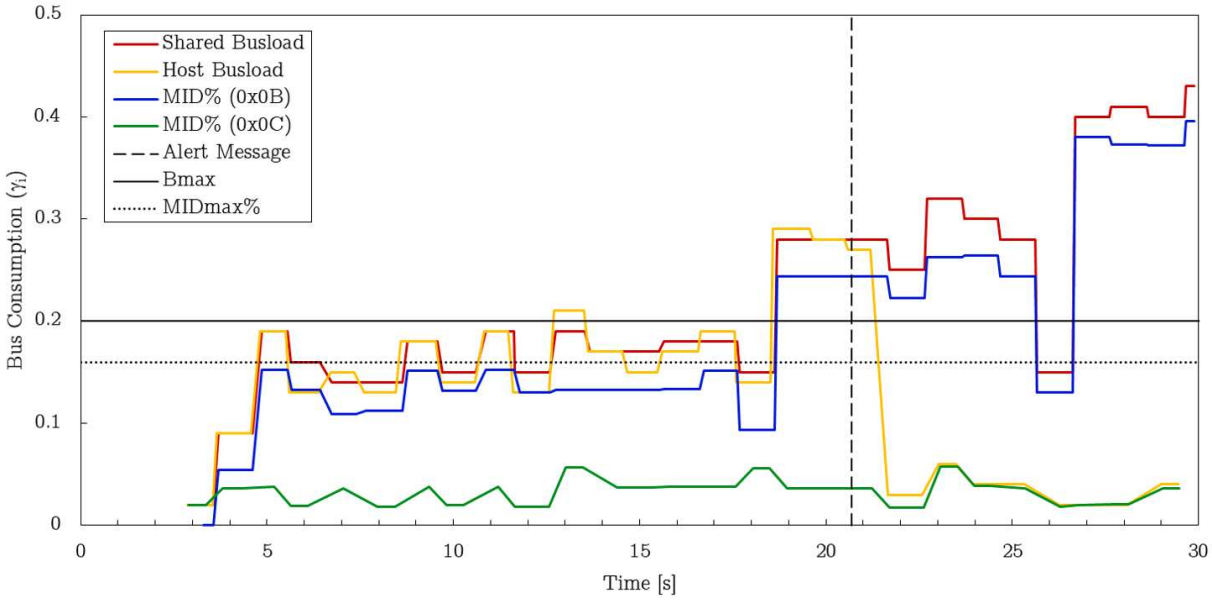


Figure 5.16: ERR9 Unit Test Results: Shared and host port busload versus time as seen by the receiver with MID 0x0A. Threshold values are plotted as horizontal lines. Alert message reception is denoted with a vertical line. Proportional bus consumption for MID 0x0B and 0x0C plotted against time. The prototype gateway had the following configuration: $\{\gamma_{max}: 0.2,\ \mu_{max}: 0.8,\ N_{CE6}{:}4\}$

A similar test was conducted for ERR10 verification and was successful. Observations showed the appropriate message was broadcast to the shared port only when a forwarding enabled, host-designated port was linked to another port.

5.4.8 Message Processing and Handling

`J1708Listen()` combines all the previously described functional units into a single routine. It is intended to be called regularly by `loop()` or by any of the `SerialEvent` interrupt functions of the corresponding Teensy 4 serial ports [45]. The flowchart in Appendix B.6 shows the general procedure that occurs each time it is called. In sum, the routine checks if a message is actively being received on the bus. If so, it will prioritize this activity until the message is complete. Subsequently, the message is inspected for standard errors and security violations. All messages, errors, and violations are displayed on `Serial1` according to the active display configuration.

When the routine is not actively receiving a message, the latest received message is parsed and marked for further handling if it is relevant using `J1708Parse()`. For example, if an RTS message is received, `J1708Parse()` will identify, PID 197 (0xC5) and the RTS command byte (0x01) and return a reference to the CTS handler. Immediately after, `J1708Listen()` uses this reference to call the appropriate handler. In the current example, this would be `CTSHandler()`. A single message is also transmitted from the global dispatch queue if any are available. If not, `J1708Listen()` completes. Any subsequent messages are processed and sent on the next call.

`J1708Update()` is a wrapper function that enhances programming flexibility when writing scripts for other use cases. The programmer can use this function to perform specific parts of `J1708Listen()` or to run a custom routine altogether. This helps reduce unnecessary computation at the programmer's discretion.

The primary concern among these functions is how the additional processing will affect message forwarding rates. It is assumed that given enough messages to process, messages may be dropped in transit to the host. Thus, a metric test was conducted to understand these limitations under high network activity. These tests are described in Section 5.5.

5.4.9 Leaky Bucket Mechanism

The queuing functionality of `J1708Send()` was identified as a means of limiting the bandwidth and burstiness of traffic flow from an arbitrary port in a manner resembling a leaky bucket. Given a suitable configuration by the policy designer, this control feature is intended as a first defense against flooding from a compromised host.

To implement these controls, a penalty parameter was defined and combined with the existing message processing. During normal operation, the transmit queue is filled and emptied asynchronously. If at any time the transmit queue is filled, and another transmission is attempted with `J1708Send()`, the incoming message is dropped, and a transmit penalty count, n_P, is incremented. Subsequent occurrences will continue to increase the penalty

count until the default maximum is reached. At transmission time, the penalty count and a time constant, T_P, are multiplied together and added to the bus access time as shown in Equation 5.7. Each successful transmission decrements the penalty count by one.

$$T'_a = T_i + (2T_b)P + n_P T_P \tag{5.7}$$

A simple experiment involving a series of controlled tests was conducted to quantify the effects of the leaky bucket controls. The test setup involved one device that served as a host and one that served as a standard gateway connected by a data bus. Messages from the host device were forwarded by the host port of the gateway to the shared port. Prior to each test, a leaky bucket size was applied to the shared port of the gateway. The maximum penalty count and penalty time were held constant across all tests at $n_P = 3$ and $T_P = 1s$, respectively. During each test, messages were transmitted by the host at a fixed rate using a constant MID and random data for a total duration of 30 seconds.

A log of all traffic on the shared bus and host bus was recorded. Each device port also collected post-test statistics, including all error counts. Sixteen tests spanning four message rates and four leaky bucket sizes were conducted across four experiments.

The grouped bar plot in Figure 5.17a compares the average shared busload for four leaky bucket sizes recovered from the logs at the end of each test. One bar cluster is plotted for each message rate. Colors and positions are consistent within each cluster. For example, we observe that when the host transmits random messages once every 0.05s, the average shared busload is roughly 20% across all leaky bucket sizes.

The grouped bar plot in Figure 5.17b compares the total ERR3 occurrences on the shared port for four leaky bucket sizes across all experimentation. Again, one bar cluster is plotted for each message rate, and the colors and positions are consistent within each cluster. For example, at a message rate of 0.05s, we observe no ERR3s indicating that the gateway forwarded all host messages. We can conclude that this message rate (and lower message rates) correlate to the average busloads observed in the first bar plot.

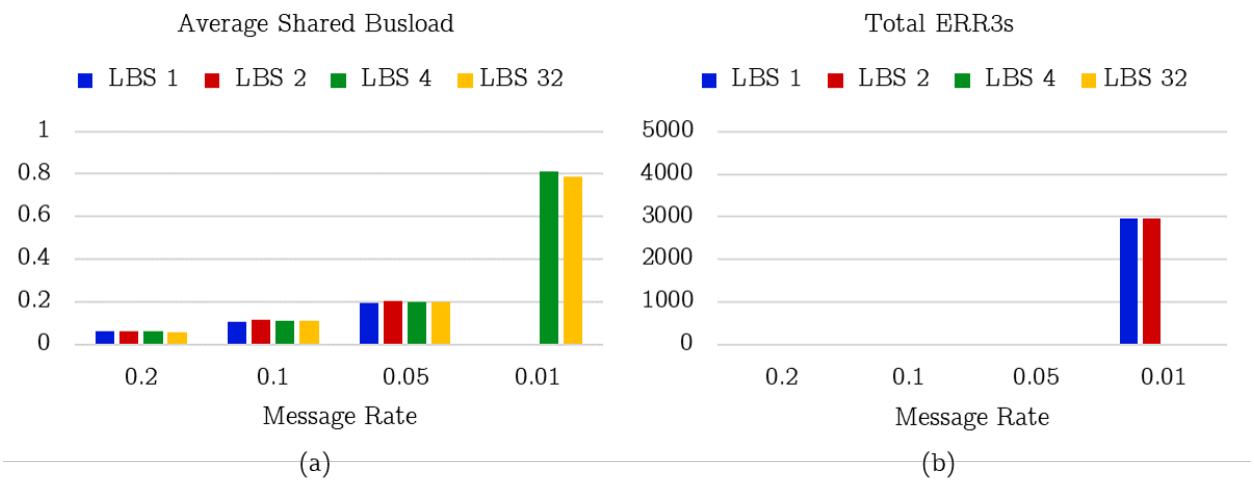


Figure 5.17: Leaky Bucket Unit Test Results: The average shared busload (a) and the total ERR3 count (b) for four leaky bucket sizes across four host message transmission periods.

At the highest message rate tested, the results show the ERR3 count increased for bucket sizes two and below. Given the relatively high penalty value, this blocked nearly all messages from being forwarded and effectively reduced the average busload to zero. Larger bucket sizes allowed traffic to pass through normally, resulting in approximately 80% average busload on the shared bus.

Later studies revealed that reduced penalty times also reduced the number of dropped messages at smaller bucket sizes. This also increased the average busload on the shared bus. Together, these results show that the application of the leaky-bucket procedure to the appropriate port increased the availability of the bus and penalized transmit violations proportional to the level of network abuse in cases where message transmission rates were below 0.03 seconds. These results also provided metrics that were used as the starting rationale for tuning the transmit queue size based on host busload profiles.

5.4.10 J1587 Transport Protocol Utility

A J1587 transport protocol utility is also available for sending data payloads greater than 19-bytes (excluding the MID and checksum) in the run-time script or through the serial API. The `J1708Transport()` method takes the message bytes, byte count, source,

and destination MIDs as input and will send the appropriate RTS message to a specific ECU. Subsequent transport messaging procedures are automatically handled using PID 198 (0xC6), "connection mode data transfer" (CDP). These messages are observable from the serial port display. A `J1708` object can only support one ongoing connection at a time, and connections lasting over 10 seconds are automatically aborted with the "Abort" connection management control command. Although the maximum allowable payload can be sent with the utility, payloads are capped at 256-bytes to preserve dynamic memory. Each gateway is equipped with an RTS handler, a CDP handler, and a CTS routine that perform the necessary processes for receiving transported data on behalf of the configured host MID. However, this functionality is only available when enabled to prevent the gateway from intercepting normal communication to its host.

Three tests were conducted to verify the functionality of the J1587 transport utility. In the first test, two gateway devices were connected to the same bus. One device had gateway processing enabled. The other device was given a 30-byte data payload to transport through the serial API. Figure 5.18a shows the recorded communication between the two devices using a default segment size of 15-bytes. This test was repeated successfully with various segmentation sizes and payloads.

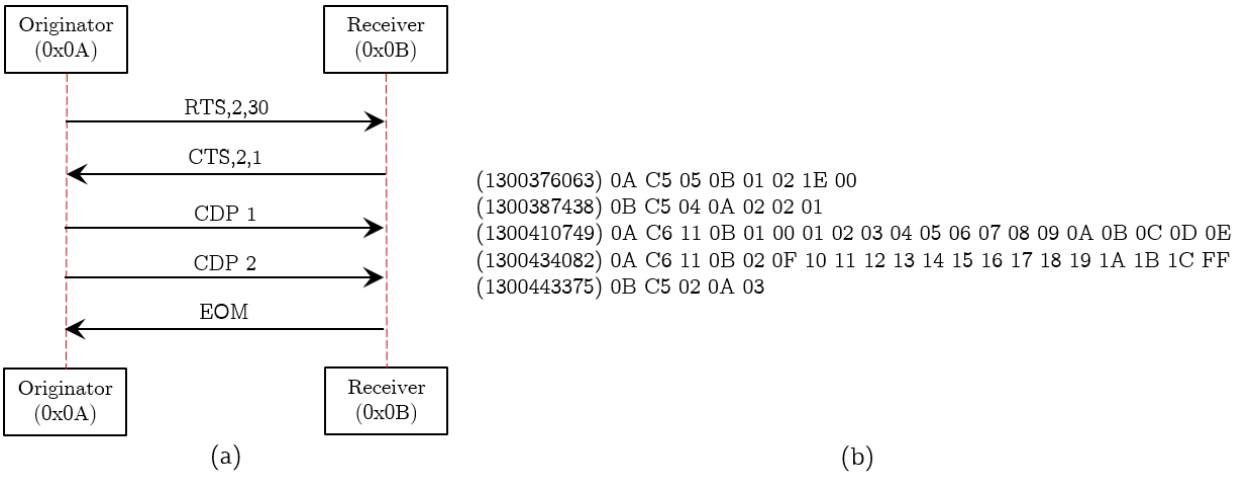


Figure 5.18: J1587 Transport Protocol Utility: Unit Test 1 Results

Another test was conducted to determine if prolonged connections were reliably aborted by the originating station after a given time. Again two devices were connected to the same bus. However, no response was given after the first CTS message from the receiver. The recorded communication in Figure 5.19 shows the connection was aborted by the originating node after 10 seconds of idle connection use.

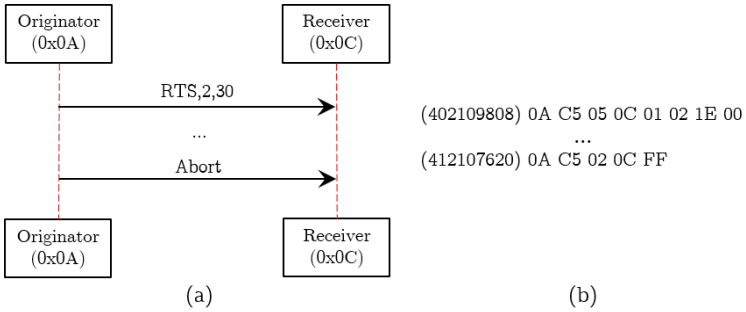


Figure 5.19: J1587 Transport Protocol Utility: Unit Test 2 Results

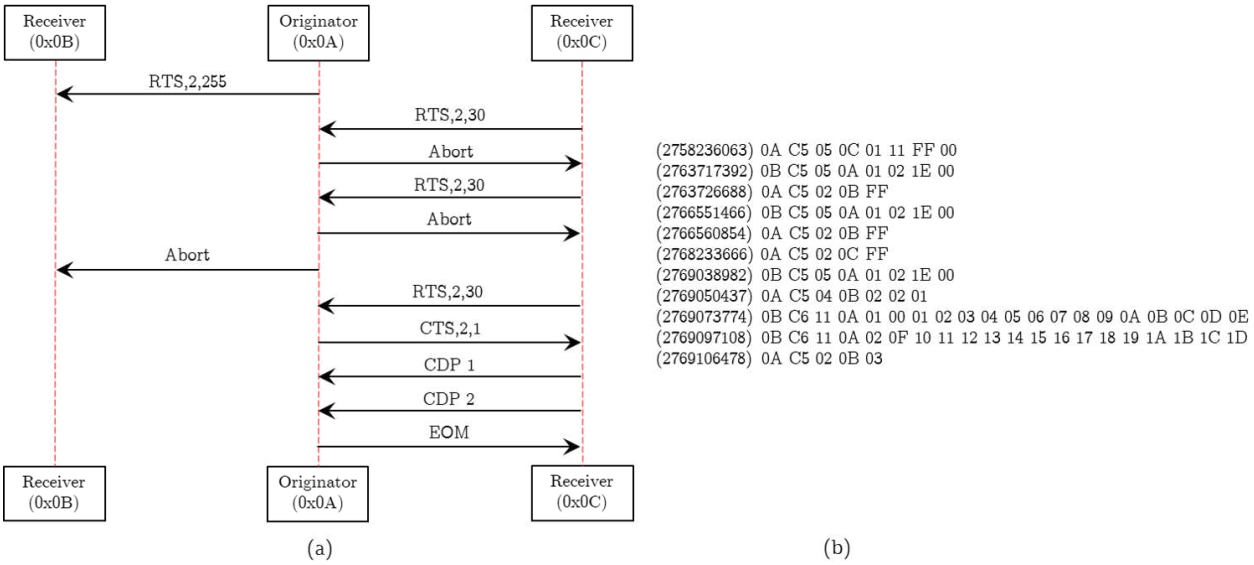


Figure 5.20: J1587 Transport Protocol Utility: Unit Test 3 Results

A final test was conducted to ensure an ongoing transport session could not be interrupted pre-maturely unless one node disconnected with the "Abort" message. A clean disconnection

should immediately allow a new connection to be made without waiting for the old connection to timeout. In the test setup, three nodes were connected to the same bus, and the session timeout for all nodes was increased to 20s. The recorded session in Figure 5.20 showed a RTS message was delivered to receiver 0x0B from originator 0x0A. After 10s of idle connection use, the session was aborted by the originating station. Despite previous attempts to send an RTS message, a new RTS was immediately successful from a second originating station with MID 0x0C after the abort message was delivered by station 0x0A.

With the success of all three tests, other unique edge cases and details within each transport handler were explored with smaller-scale testing.

5.5 Integrated Software Testing and Validation

After all the implemented features had been unit tested, the combined software was commissioned on the prototype hardware and subjected to network experiments. One such experiment aimed to identify specific combinations of gateway settings that could reduce message capture rates in one direction. Serial printing, message forwarding, and gateway specific processing were three settings of particular concern. These three tasks were known to consume more processing cycles on the Teensy 4.0 than any other process in the source code. Subsequently, all eight possible combinations of these settings were organized into a formal 2^3 factorial design that was coordinated using an automated Python script and the gateway serial API.

The experimental setup consisted of two prototype gateways connected to a single J1708 bus. The receiver gateway was subjected to sustained bursts of 500 21-byte messages from the other device at message rates approaching the theoretical maximums permitted by the protocol. Each setting combination was deployed as a single test. The receiver system statistics and message logs were gathered from each device after each test for later study.

The three factors are designated A, F, and P (i.e., print all, forward messages, gateway processing) based on the command options shown in Figure 5.5. The total dropped messages

in each test, η, was used as the response variable. The experiment was replicated twice with a randomized run-order for a total of 16 observations. The raw data is provided in Appendix B.4. Plus and minus signs are used to indicate the different levels.

Although the results were positive, an underwhelming average of 0 dropped messages were recorded across all the observations. Thus, subsequent analysis of the response variable was suspended. Observations captured with the Saleae also showed a worse case message forwarding delay of 1.4ms. Given the results from both these tests, the gateway was deemed fit for system integration.

5.6 Summary

This section covered the software design, integration, and testing activities during the development of the prototype J1708/J1587 gateway. Having followed a structured approach with iterative unit testing, the software requirements outlined in section 5.2 were satisfied. A tabulated summary of relevant unit tests and their related requirements are provided in Appendix B.5. All test artifacts, including analog captures, test logs, modified gateway firmware, and Python scripts, are available in the "software" directory of the thesis repository. The next chapter will discuss the integration of the completed prototype into DABSS and system testing.

Chapter 6

System Test and Evaluation

This chapter discusses the integration of the prototype gateway and the DABSS electronic subsystem. This chapter also presents the working threat model and the application of the network defense against four attack scenarios in fulfillment of SR4. The results are compiled after each test and evaluated in the final section.

6.1 Feature Integration and Testing

To begin the integration process, a baseline J1708 data log of the system without any security enhancements was captured using the logging features of a gateway prototype. These logs were consistent with those captured by the diagnostic tool during the retrofitting activities discussed in Chapter 3. Given the careful preparation of the DABSS electrical harness, the integration of the prototype was relatively straightforward.

First, custom Molex connectors were prepared to replace the loop-back connections between the shared J1708 bus and each network host. Next, a screw terminal adapter corresponding to the gateway channel and 12V power was added to the other end of the cable. Finally, each device was temporarily mounted to the board and linked to its corresponding connection points as shown in Figure 6.1.

Having mounted and integrated the gateways, a simple pass-through script was flashed to the prototypes with a non-restrictive security configuration. This configuration also allowed the system to be power-cycled without reconfiguring the devices each time. A second traffic capture from the shared bus indicated no conflicts or changes in the traffic patterns compared to the baseline. Additionally, no differences were observed in the controls and reports generated by Wabco Toolbox.

Finally, an analysis of the network traffic was conducted to determine suitable security settings for each gateway. The plot in Figure 6.2 was generated as part of this work. The

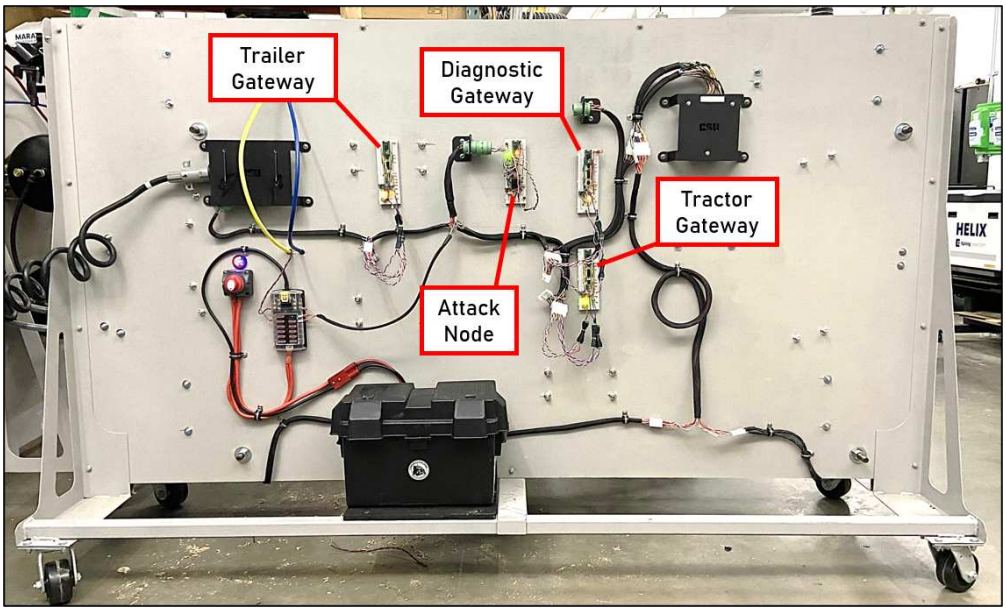


Figure 6.1: Back-side photo of the DABSS with the prototype J1708 gateways at each ABS controller and the diagnostic port.

plot in blue shows the total busload over time during different phases of communication between the three applications. Notably, the maximum busload observed at any time was roughly 30% on the shared bus. The other three plots show the share of the total busload contributed by each application over time. The maximum observed contribution by any application was roughly 18.6%, which accounted for roughly 64% of network traffic at that instance (γ_i=0.29).

Using these observations, the maximum busload threshold, γ_{max}, was set to 35%. The maximum contribution value, μ_{max}, was set to 60%. The traffic pattern was used to determine the leaky bucket parameters. Ultimately, the baseline security configurations shown in Tables 6.1 and 6.2 were selected after additional tuning of the network parameters.

These configurations were respectively flashed to each gateway and subjected to prolonged operation. Aside from stabilizing the MID claim procedure between the two ABS controllers, no meaningful conflicts or faults were observed. Therefore, having achieved desirable network stability with the added security hardware, the system was deemed fit for validation testing.

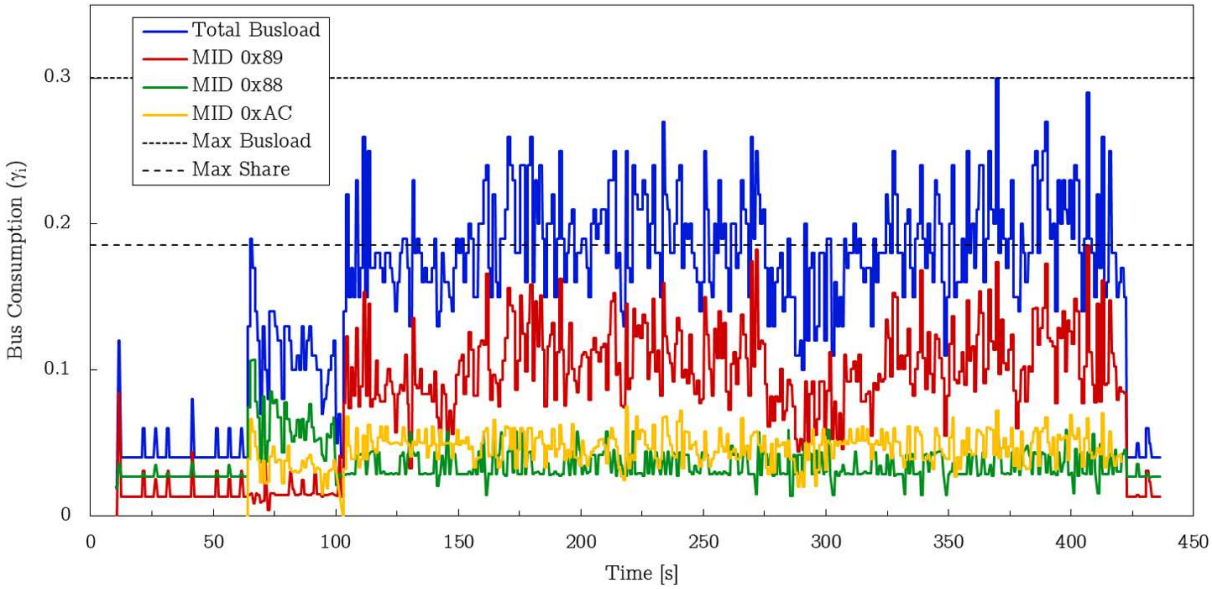


Figure 6.2: DABSS Composite Traffic Profile

Table 6.1: Gateway Host Port Security Configuration

Component	Diagnostic Port	Tractor ABS	Trailer ABS
Host Port	Serial 4	Serial 4	Serial 4
Host Port Allow List	0xAC	0x88	0x89
Host Port Self MID	0xAC	0x88	0x89
Host Gateway Processing	FALSE	FALSE	FALSE
Host Max Busload Threshold	0.35	0.35	0.35
Host Max MID Share	0.60	0.60	0.60
Host ERR7 Max Occurrence	65000	65000	65000
Host Rx Forwarding	TRUE	TRUE	TRUE
Host Tx Queue Size	32	32	32

Table 6.2: Gateway Shared Port Security Configuration

Component	Diagnostic Port	Tractor ABS	Trailer ABS
Shared Port	Serial 3	Serial 3	Serial 3
Shared Port Allow List	{0x89,0x88}	{0x89,0xAC}	{0x88,0xAC}
Shared Port Self MID	0xAC	0x88	0x89
Shared Gateway Processing	FALSE	FALSE	FALSE
Shared Max Busload Threshold	0.35	0.35	0.35
Shared Max MID Share	0.60	0.60	0.60
Shared ERR7 Max Occurrence	5	5	5
Shared Rx Forwarding	TRUE	TRUE	TRUE
Shared Tx Queue Size	1	2	2

6.2 Threat Model

Drawing from the analysis and discussion in Chapter 2, two specific adversaries are defined: a rogue node and a compromised node. A rogue node is defined as any device capable of transmitting arbitrary J1587 messages onto the J1708 bus that is not an inherent component of the vehicle system. This definition is intentionally vague to include both wired and wireless attack vectors. This can also be considered an attack mounted on the receive path of the gateway.

A compromised node is defined as any inherently trusted device that has been exploited or manipulated for physical access to the J1708 bus, including ECUs and standard devices connected to the diagnostic port. In contrast to a rogue node, messages from a compromised node are sent using the hardware equipped on the device. They can be generalized as an attack mounted on the transmit path of the diagnostic gateway.

Based on these adversaries and the discussion in Chapter 2, we select two types of attacks carried out by each of the adversaries for a total of 4 unique attack scenarios: message spoofing by a rogue node, message flooding by a rogue node, message spoofing by a compromised node, and message flooding by a compromised node. These scenarios are tabulated in Table 6.3 and described in the following subsections.

Table 6.3: Attack Scenarios for Validation Testing

Designation	Adversary	Attack Type
A	Rogue Node	Message Spoofing
B	Rogue Node	Message Flooding
C	Compromised Node	Message Spoofing
D	Compromised Node	Message Flooding

6.2.1 Rogue Node: Message Spoofing

In this attack scenario, a rogue device controlled by the adversary injects fabricated messages on a J1708 bus to impersonate an ECU. The objective of this attack can vary, but

some form of vehicle manipulation or denial of service is typically expected, as explored in Section 2.2.5. This attack is illustrated in Figure 6.3.

A less targeted form of message spoofing is message fuzzing. This technique is often used in software development tests and involves injecting random data on the bus to uncover any unexpected behavior.

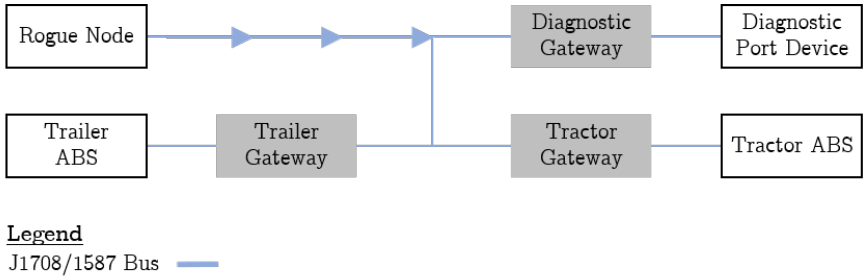


Figure 6.3: Rogue Node Attack Configuration

Two custom Python scripts, `Rogue-Fuzz.ipynb` and `Rogue-Spoof.ipynb`, were written to control a rogue node during validation testing using commands in the serial API. The rogue node sends messages with random data at a fixed rate in the first program. Attack parameters given at the start of the program control the duration of the attack, the inter-message period, and whether or not to use a constant MID or a random MID for each message.

In the second script, the rogue node attempts to initialize a diagnostic session with the tractor ECU by impersonating the Wabco diagnostic software with MID 172 (0xAC). Subsequent messages are intended to cause pressure buildups on the DABSS's relay valves. Specific control of these valves was discovered by analyzing diagnostic messages sent by the Wabco Toolbox software during verification testing. This test was selected partially because these pressure buildups are easy to hear when an attack is successful.

6.2.2 Rogue Node: Message Flooding

In this attack scenario, a rogue device controlled by the adversary fabricates and injects messages on a J1708 bus at a high rate to overwhelm a controller so that it cannot examine permitted network traffic. The objective of this attack can vary, but denial of service is typically expected. This attack is illustrated in Figure 6.3.

A custom Python script, `Rogue-Flood.ipynb`, was written to carry out different forms of this attack using a rogue node controlled through the serial API. The attack parameters are identical to those in `Rogue-Fuzz.ipynb`.

6.2.3 Compromised Node: Message Spoofing

In this attack scenario, a rogue device controlled by the adversary fabricates and injects messages with a forged MID and data on a J1708 bus. As before, the objective of this attack can vary, but some form of vehicle manipulation or denial of service is typically expected. This attack is illustrated in Figure 6.4. `Compromised-Fuzz.ipynb` was written to control a compromised node during validation testing through the serial API. The attack parameters are identical to those in `Rogue-Fuzz.ipynb`.

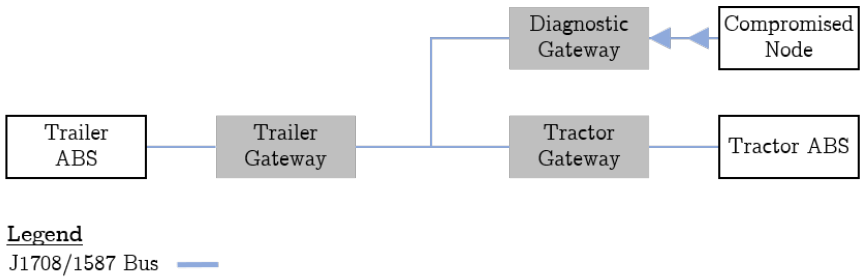


Figure 6.4: Compromised Node Attack Configuration

6.2.4 Compromised Node: Message Flooding

In this attack scenario, a compromised device with a full range of transmitting capabilities fabricates and injects messages on a J1708 bus at a high rate in an attempt to overwhelm

other controllers on the bus. Figure 6.4 is used to illustrate this configuration. `Compromised-Flood.ipynb`, was written to control a compromised node during validation testing through the serial API. All attack scripts used during validation testing are available in the thesis repository.

6.3 Security Concept Validation Testing

6.3.1 Rogue Node: Setup, Procedure, and Results

A gateway device designated as the attack node was connected to the DABSS J1708 bus in a manner consistent with Figure 6.3. In addition, it was given the basic configuration shown in Table 6.4 so that it did not interact with the bus outside of transmitting spoofed messages. All traffic logs generated during testing are available in the thesis repository.

Table 6.4: Attacker Node Port Configuration

Parameter	Host Port	Shared Port
Hardware Serial Port	Serial 4	Serial 3
Port Allow List	ALL	ALL
Port Self MID	0x78	0x78
Gateway Processing	FALSE	FALSE
Max Busload Threshold	2.00	2.00
Max MID Share	2.00	2.00
ERR7 Max Occurrence	65000	65000
Rx Forwarding	FALSE	FALSE
Tx Queue Size	32	32

Message Fuzzing

`Rogue-Fuzz.ipynb` was used to send random messages with a random MID every 0.2s for 30 seconds on the bus. This attack consistently caused the diagnostic software to crash after a few seconds in the unprotected state. To mitigate the effects of network abuse, the DABSS gateways were configured with the baseline protection provided in Tables 6.1 and 6.2.

After a system reset, fuzzing was resumed. All of the shared network traffic was captured by the test script. Host traffic logs from each of the gateways were also recovered after testing was completed. The results showed that none of the fuzzed messages reached any of the host controllers. In the singular instance that a falsified message used a MID protected by one of the host gateways on the bus, the violation was immediately reported with a message spoof alert.

Message Spoofing

`Rogue-Spoof.ipynb` was used to send spoofed diagnostic messages. This attack caused artificial pressure buildups at each of the six DABSS valves in the unprotected state. To mitigate the effects of network abuse, the DABSS gateways were configured with the baseline protection given in Tables 6.1 and 6.2.

After a system reset, the spoofing attack was conducted once more. All of the shared and host network traffic was captured and preserved. Post-analysis of these records showed that the diagnostic port gateway issued an imposter node alert after five spoofed message violations. This alert caused all other gateways to block subsequent messages with the diagnostic port MID. Analysis of the host logs confirmed the last message received by all other hosts was the imposter alert. No clicks occurred from any of the valves while the attack was active.

Message Flooding

In this experiment, three variations of message flooding were studied. The DABSS gateways were configured with the same baseline protection provided in Tables 6.1 and 6.2 across all tests. Unlike in previous experiments, the maximum busload, γ_{max}, and the maximum allowable resource consumption by any host, μ_{max}, were expected to define the test outcome. All of the shared and host network traffic was captured and preserved during the experiment.

`Rogue-Flood.ipynb` was used in the first test to inject random payloads from an engine controller using MID 1 (0x01) every 0.05s for 30 seconds on the bus. These parameters

were known from previous experiments to raise network traffic relatively close to the maximum security thresholds. Thus, it was helpful to assess the gateway's susceptibility to false positives.

The results showed two separate periods lasting roughly 1s where γ_i measured above the threshold. The ERR6 counters from the shared ports on each network host also denote these occurrences. However, the default maximum sustained duration for a flooding attack to be identified with a security notification is 2s. Since the shared traffic records did not show a security notification and all the devices reported zero ERR9s, the gateways successfully maintained a true-negative status. Inspection of the host logs also showed that none of the injected messages on the shared bus made it to the host, which is consistent with the configured access control policy.

In a second test, `Rogue-Flood.ipynb` was used to inject random payloads from an engine controller using MID 1 (0x01) every 0.01s for 30 seconds on the bus. Messages from the script caused the busload to increase above 82% approximately 1s after the attack. Over 90% of this activity was attributable to MID 1. Post analysis of these records showed that an abnormal message rate was reported using PID 762 by the tractor gateway 3s into the attack. Although the access policy prevented this data from reaching the hosts anyway, analysis of the attacker access control list showed that MID 1 was blocked by the end of the test. Thus, the gateways successfully detected a true positive.

In the final test, `Rogue-Flood.ipynb` was used to inject random payloads from a random MID every 0.01s for 30 seconds on the bus. Messages from this script caused the busload to increase above 82% approximately 1s after the start of the attack. No more than 11% of this activity was attributable to a specific MID. Since this did not violate the security policy, no abnormal message rate notifications were delivered by any host gateways. The traffic records also showed multiple instances where this attack triggered a spoofed message notification. Ultimately, enough spoofed messages were delivered such that three different imposter alert messages were delivered by each of the hosts. This caused all the hosts to be blocked on

the shared bus. Again, the records showed that none of the injected messages with blocked MIDs on the shared bus were forwarded to any hosts.

6.3.2 Compromised Node: Setup, Procedure, and Results

A gateway device designated as the attack node was connected to the transmit path of the DABSS diagnostic port in a manner consistent with Figure 6.4. In addition, it was given the basic configuration shown in Table 6.4 so that it did not interact with the bus outside of injecting attack messages. All traffic logs generated during testing are available in the thesis repository.

Message Spoofing

`Compromised-Fuzz.ipynb` was used to send random messages with a random MID every 0.25s for 30 seconds on the bus. To mitigate the effects of network abuse, the DABSS gateways were configured with the baseline protection provided in Tables 6.1 and 6.2. All of the shared network traffic was captured by the test script. Host traffic logs from each of the gateways were also recovered after testing was completed. The results showed that the diagnostic port gateway forwarded only messages with MID 172 (0xAC) to the shared bus. The diagnostic port gateway blocked all other messages since they had MIDs that appeared on the transmit path ACL. Given the nominal message rates and the short duration of the attack, the gateway reported no security errors.

Message Flooding

`Compromised-Flood.ipynb` was used to send random messages with MID 172 (0xAC) every 0.01s for 30 seconds on the bus. These attack parameters were selected to violate the host busload usage policy, causing an ERR10. Consistent with the previous testing, the DABSS gateways were configured with the baseline protections provided in Tables 6.1 and 6.2, and all traffic logs were preserved for later study.

The results showed a complete true-positive response to the attack. As expected, the shared bus traffic log revealed that the diagnostic gateway successfully reported the incident with a compromised host alert. Subsequent notifications were delivered only periodically as designed. The post-test results also showed that the two other gateways blocked MID 172 (0xAC) after receiving this notification and incremented their respective ERR8 counts. Out of 3000 total message attempts, 212 ERR3s were counted by the diagnostic gateway, indicating the leaky bucket was active and functioning before the notification occurred.

6.4 Final Cybersecurity Assessment

6.4.1 Determining Gateway Parameters

The gateway security parameters can be grouped into three categories of various critical functions: access control, leaky bucket, and busload consumption. Parameters concerning access control were relatively straightforward to define, assuming one-to-one matching of a finite set of host MIDs.

However, defining parameters in the latter two categories such that false positives across the system were reduced was a cumbersome and time-consuming process. Even after obtaining traffic profiles from each host and the shared bus, it was often unclear what threshold would maximize detection accuracy without iterative experimentation. Moreover, after settling on a suitable configuration, additional post-testing is warranted to determine false-positive sensitivity over long periods and account for various edge cases where high busloads are expected. As a result, only a fraction of suitable configurations were explored.

Furthermore, past research shows that a static configuration is susceptible to errors if traffic patterns change. Although this may not be the case with unchanged ECU firmware, it *could* be the case for traffic at the diagnostic port. Despite these challenges, the composite security configurations provided in Tables 6.1 and 6.2 performed as expected in the four attack scenarios and improved attack detection overall.

6.4.2 Defense Effectiveness

Under the given assumptions, the current security concepts are primarily effective against attacks on the receive side of the gateway. Across all the rogue node tests, the system detected all instances of message spoofing and reported it. In cases where a message was received that was not permitted by the access policy, the message was dropped before reaching the host controller.

However, some gateways passed spoofed messages to their hosts until the maximum ERR7 threshold was reached. Under the current defense, an attacker can send messages up to this threshold before being stopped. Thus, the designer must determine a balance concerning the sensitivity of the system to policy violations and the number of messages afforded to the attacker. This threshold is difficult to ascertain because of the unknown nature of sophisticated J1708-based network attacks.

High message rates on the receive path caused by flooding attacks from a single MID are also detected, assuming suitable busload thresholds have been defined. Once identified, messages from the MID mounting the attack are blocked. However, this defense was less effective when multiple MIDs were used. In this case, any undefended MIDs will be passed to the host. This issue does not exist if all MIDs are defended or blocked across the system. However, the current defense cannot stop the offender from continuing the attack even after detection.

Assuming a non-host MID is used, the current security concept is highly effective against spoofing attacks on the transmit path of the gateway. In all cases where a compromised host attempted to send a message with a MID that was not initially assigned to it, the gateway blocked the transmission. However, the current defense could not stop injected messages when the assigned MID was used. The combination of the leaky-bucket mechanism and a suitable host busload policy effectively detected and prevented flooding from a compromised node on the DABSS.

6.4.3 Real World Practicability

In its current form, the defense has some notable strengths that lend themselves to real-world use. Namely, it can prevent message spoofing and identify flooding without the technical challenges associated with other cryptographic-based strategies. These challenges include key management, busload increases, and message processing complexity. These benefits were identified and discussed in Chapter 2. Additionally, the configuration of the gateways may be comparatively easier to deal with compared to managing keys across hosts in a PKI-based proposed by state-of-the-art defense research.

The current strategy does not account for the possibility of multiple MIDs claimed by a single host. Presently, the gateway can protect only one MID in the deployed software design. Although multi-MID defense was beyond the test assumptions, this feature can be implemented in future work without complications to the existing strategy. This feature would also allow the security policy designer to defend (instead of block) MIDs known to never exist on the bus.

Leaky bucket procedures may also conflict with real-time transmission requirements. However, most modern trucks are assumed to utilize very little of the available bandwidth on a J1708 bus while in service. In such cases, the limitations associated with leaky buckets applied to the host transmitter might be mitigated with some additional tuning of the defense parameters.

There are also challenges associated with the defense effectiveness and "configurability," as discussed in the preceding subsections. Overall, the defense may lack the determinism needed under stringent security requirements. Additionally, the current strategy assumes a gateway supports all transmitting nodes, and all MIDs are either defended or blocked across the system. If external challenges exist such that these requirements cannot be satisfied, the lack of protection may undermine the entire defense. In other words, the current implementation is highly dependent on the broad deployment of the gateways across multiple hosts.

Chapter 7

Conclusion

The following sections summarize the results from system testing, highlight the limitations that had the most significant potential impact on the quality of the research findings, and provide recommendations for future work.

7.1 Thesis Review

Throughout six chapters, this thesis provided background discussion on SAE J1708/1587, conceptualized a security concept, discussed the preparation of a vehicle network simulator, and described the design and development of a prototype network gateway. The defense concept was tested against four attack scenarios in the penultimate chapter. In light of the original objectives, the following conclusions are drawn from the test results:

1. Message spoofing on the gateway receive path was successfully identified by the defense system.

2. Only spoofing attempts with non-host MIDs were identified by the defense system.

3. The system successfully identified busload consumption from a specific application that exceeded pre-configured thresholds on the receive and transmit paths of the defense system.

4. Spoofed messages that appeared on the block list of both the transmit and receive paths were successfully blocked by each host protected by the defense system.

5. The system successfully blocked messages from a specific application that exceeded pre-configured busload consumption thresholds on the receive and transmit paths of the defense system.

However, there were some shortcomings,

1. Spoofed messages are permitted by the system on the receive path until a pre-defined threshold is reached.

2. Falsely injected messages on the transmit path using a MID of the host were permitted.

3. System testing in Chapter 6 revealed that parameter tuning is a cumbersome, time-consuming, and at times imprecise process. Furthermore, any defense configuration will be tailored specifically to anticipated attack scenarios and may not be adequate for defending against novel or dynamic attacks.

4. The application of the security concept in its current form does not account for deployment challenges in the real world.

To my knowledge, this is the first exploration of J1708/1587 intrusion detection in academic research. The following list recaps the other thesis contributions:

1. A J1708 library for the Teensy 4.0 embedded platform.

2. Circuit design of a J1708 gateway.

3. Retrofits to vehicle simulator at CSU.

4. A working J1708 network defense concept.

Furthermore, the gateway software was bundled into a separate repository and published for open use on Github under the name "J1708_T4" [46].

7.2 Limitations

As noted in Chapter 1, prior research on J1708 network security is limited. This thesis only explored a defense concept attuned to the simplest cases of message spoofing and continuous message flooding at fixed rates. Related work on automotive CAN has revealed that more intelligent abuse of modern protocol semantics can achieve equivalent negative

outcomes with fewer messages. Given the similarities between J1939 and J1708, it is appropriate to assume these exist in J1708 as well.

Despite the importance of exploring sophisticated attacks, an analysis of the basic attack scenarios provides general observations of network behavior in real-time. Given the unexplored nature of J1708 networks, the current findings should assist with more nuanced studies in the future. Furthermore, the present work provided the basis for developing essential J1708 network inspection tools. These tools should minimize the groundwork effort in future studies.

In-vehicle testing of the gateways could have provided observations that better resemble those in a real-world HD environment. However, this thesis conducted tests on an HD vehicle network simulator in place of an actual truck. This compromise was beneficial for several reasons: (1) Given the intrusive and costly nature of retrofitting ECUs on a real truck, a vehicle simulator was chosen to save resources. (2) Changes to the system could be performed quickly. (3) The defense is still in the early stages of development. However, these activities may be incorporated in later work should the project mature through technical improvements to the gateway and future development of the defense.

7.3 Future Work

Working on this project uncovered additional research questions that did not fall within the original scope of the work. Furthermore, technical limitations were exposed after development that could be resolved with future improvements to the gateway hardware and software.

General Research. An investigation of J1708/1587 protocol-specific attacks is warranted. Namely, this includes exploring protocol-based DoS attacks, the effects of J1708/1587 attacks on other network behavior, and parameter spoofing. Work conducted by Mukherjee et al. is recommended as a starting reference for the scope of work [23].

Discussion in Chapter 2 identified the following areas of future research tasks: (1) Apply and investigate the impacts of implementing cryptography-based authentication on J1708/1587. (2) Apply and investigate stateful anomaly detection (i.e., state-based packet inspection) to a J1708 IDS.

DABSS Improvements. Ideally, future research on the DABSS J1708 bus will be conducted in a fault-free environment. At present, wheel speed generators and sensors are all that is needed to accomplish this. Additionally, the system needs lamps and an electrical connection to the air dryer.

Technical Gateway Improvements. The addition of an HSM to the hardware design is desired to support future studies using cryptographic defenses. The following features and changes are recommended for future iterations of the gateway software: (1) Multiple enforced MIDs from a single port; (2) Improve architecture and data processing using metadata in `j1708_message_t` data structure; (3) Save gateway configuration to EEPROM. (4) Refactor the leaky-bucket design such that arbitrary message rates can be regulated.

Bibliography

[1] WABCO, *Enhanced Easy-Stop Trailer ABS with PLC: Maintenance Manual MM-0180*, 4th ed., WABCO, February 2010.

[2] S. Mukherjee, H. Shirazi, I. Ray, J. Daily, and R. Gamble, "Practical DoS attacks on embedded networks in commercial vehicles," in *Information Systems Security*, I. Ray, M.-S. Gaur, M. Conti, D. Sanghi, and V. Kamakoti, Eds. Cham: Springer International Publishing, 2016, pp. 23–42.

[3] K.-T. Cho and K. G. Shin, "Fingerprinting electronic control units for vehicle intrusion detection," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, August 2016, pp. 911–927. [Online]. Available: usenix.org/conference/usenixsecurity16/technical-sessions/presentation/cho

[4] S. Kim, G. Ye, T. Kim, J. Rhee, Y. Jeon, A. Bianchi, D. Xu, and D. Tian, "ShadowAuth: Backward-compatible automatic CAN authentication for legacy ECUs," in *ACM Asia Conference on Computer and Communications Security (ASIA CCS '22)*, Nagasaki, Japan, May 2022.

[5] S. Stachowski., R. Bielawski, and A. Weimerskirch, "Cybersecurity research considerations for heavy vehicles (report no. dot hs 812 636)," National Highway Traffic Safety Administration, Washington, DC, Tech. Rep., December 2018.

[6] Fleet Pulse, "6 trends that will define the future of truck and trailer connectivity," Great Dane, Tech. Rep., August 2021. [Online]. Available: greatdane.com/wp-content/uploads/2021/08/6-Trends-That-Will-Define-the-Future-of-Truck-Trailer-Connectivity.pdf

[7] M. Hallenbeck, O. Selezneva, and R. Quinley, "Verification, refinement, and applicability of long-term pavement performance vehicle classification rules," Federal Highway Administration, McLean, VA, Tech. Rep., November 2014.

[8] F. H. Administration, "FHWA vehicle classification chart," Web. [Online]. Available: avcog.org/DocumentCenter/View/3244FHWA-Vehicle-Classification-Chart-PDF

[9] SAE, *J560_202002: Primary and Auxiliary Seven Conductor Electrical Connector for Truck-Trailer Jumper Cable*, Society of Automotive Engineers, Warrendale, PA, February 2020.

[10] ——, *J1587_201301: Electronic Data Interchange Between Microcomputer Systems in Heavy-Duty Vehicle Applications*, Society of Automotive Engineers, Warrendale, PA, January 2013.

[11] ——, *J2497_201207: Power Line Carrier Communications for Commercial Vehicles*, Society of Automotive Engineers, Warrendale, PA, July 2012.

[12] CARB, "Title 13 - California Code of Regulation," *California Code of Regulations*, Jul 2008. [Online]. Available: ww3.arb.ca.gov/regact/regs-13.htm

[13] USEPA, "Part 86 - control of emissions from new and in-use highway vehicles and engines," *Code of Federal Regulations*, 2010. [Online]. Available: www.ecfr.gov/current/title-40/chapter-I/subchapter-C/part-86

[14] Y. Burakova, B. Hass, L. Millar, and A. Weimerskirch, "Truck hacking: An experimental analysis of the sae j1939 standard," in *WOOT*, 2016.

[15] Elizabeth Bate, "Some ELDs May Present Cybersecurity Risk," *Today's Trucking*, June 2018. [Online]. Available: truckinginfo.com/304411/some-elds-may-present-cybersecurity-risk

[16] C. Miller and C. Valasek, "Adventures in automotive networks and control units," 2014. [Online]. Available: ioactive.com/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf

[17] SAE, *J1939DA_202201: J1939 Digital Annex*, Society of Automotive Engineers, Warrendale, PA, January 2022.

[18] ——, *J3061_202112: Cybersecurity Guidebook for Cyber-Physical Vehicle Systems*, Society of Automotive Engineers, Warrendale, PA, August 2021.

[19] A. Ruddle, B. Weyl, S. Idrees, Y. Roudier, M. Friedewald, T. Leimbach, A. Fuchs, S. Gürgens, O. Henninger, R. Rieke, M. Ritscher, H. Broberg, L. Apvrille, R. Pacalet, and G. Pedroza, "Security requirements for automotive on-board networks based on dark-side scenarios. deliverable d2.3: Evita. e-safety vehicle intrusion protected applications," *Fraunhofer ISI*, 01 2009.

[20] M. Islam, C. Sandberg, A. Bokesand, H. B. Tomas Olovsson, P. Kleberger, A. Lautenbach, A. Hansson, A. Soderberg-Rivkin, and S. P. Kadhirvelan, "Deliverable d2 - security models," HEAVENS Consortium, Sweden, Tech. Rep., 2016. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3488904.3493378

[21] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," in *2010 IEEE Symposium on Security and Privacy*, 2010, pp. 447–462.

[22] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle." 2015, DEF CON '23. [Online]. Available: ioactive.com/pdfs/IOActive_Remote_Car_Hacking.pdf

[23] S. Mukherjee, H. Shirazi, I. Ray, J. S. Daily, and R. F. Gamble, "Practical DoS attacks on embedded networks in commercial vehicles," in *ICISS*, 2016.

[24] B. Gardiner and C. Poore, "Power line truck hacking:2TOOLS4PLC4TRUCKS," 2020. [Online]. Available: nmfta.org/documents/ctsrp/Power_Line_Truck_Hacking_2TOOLS4PLC4TRUCKS.pdf?v=1

[25] ——, "DEF CON Safe Mode ICS Village - Ben Gardiner - PowerLine Truck Hacking 2TOOLS4PLC4TRUCKS," 2020, NMFTA. [Online]. Available: youtube.com/watch?v=GJOoYAxCFgI&t=4s

[26] "gr-j2497," Github, 2020. [Online]. Available: github.com/ainfosec/gr-j2497

[27] "plc4trucksduck," Github, 2020. [Online]. Available: github.com/TruckHacking/plc4trucksduck

[28] B. Gardiner, "Disclosure of confirmed remote write to J2497 aka PLC4TRUCKS," NMFTA, Alexandria, VA, Letter, March 2022. [Online]. Available: nmfta.org/documents/ctsrp/Disclosure_of_Confirmed_Remote_Write_v4_DIST.pdf?v=1

[29] K. Scarfone and P. Hoffman, "Guidelines on firewalls and firewall policy," September 2009. [Online]. Available: tsapps.nist.gov/publication/get_pdf.cfm?pub_id=901083

[30] NXP, "Securing CAN communication efficiently with minimal system impact," NXP Semiconductor, Tempe, Arizona, Commercial Report, 2020. [Online]. Available: nxp.com/docs/en/white-paper/SECURECARTRANA4FS.pdf

[31] K. Scarfone and P. Mell, "Guide to intrusion detection and prevention systems (idps)," February 2007. [Online]. Available: nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-94.pdf

[32] J. Daily, D. Nnaji, and B. Ettlinger, "Demo: Securing heavy vehicle diagnostics," in *The Network and Distributed System Security Symposium (NDSS)*, February 2021.

[33] P.-S. Murvay and B. Groza, "Security shortcomings and countermeasures for the sae j1939 commercial vehicle bus protocol," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 5, pp. 4325–4339, 2018.

[34] J. Daily, D. Nnaji, and B. Ettlinger, "Securing CAN traffic on J1939 networks," in *The Network and Distributed System Security Symposium (NDSS)*, February 2021.

[35] J. Daily, "Heavy vehicle CAN data," 2021. [Online]. Available: https://www.engr.colostate.edu/~jdaily/J1939/candata.html

[36] SAE, *J1939/13_201610: Off-Board Diagnostic Connector*, Society of Automotive Engineers, Warrendale, PA, October 2016.

[37] D. of Transportation, *Electronic Logging Devices and Hours of Service Supporting Documents*, Federal Motor Carrier Safety Administration, Washington D.C., December 2015, 49 CFR Parts 385,386,390, & 395.

[38] AUTOSAR, *Specification of Secure Onboard Communication*, 4th ed., AUTOSAR, December 2017.

[39] "Service training aids," Brochure, Daimler Trucks North America: Service Training Academy. [Online]. Available: dtnaarc.com/daimler/content/documents/campus_8/pc_html_images/_public_/Training_Aid_Brochure-2015_Vehicle.pdf

[40] WABCO, *Anti-Lock Braking System (ABS) and Electronic Stability Controls (ESC) Maintenance Manual*, 1st ed., WABCO, 2019.

[41] J. Daily and P. Kulkarni, "Secure heavy vehicle diagnostics," in *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium (GVSETS)*. Novi, MI: NDIA, August 2020.

[42] "Truckduckhardware," Github, 2016. [Online]. Available: github.com/TruckHacking/TruckDuckHardware

[43] "Truckcapeprojects," Github, 2016. [Online]. Available: github.com/SystemsCyber/TruckCapeProjects/tree/Repo-restructure/hardware

[44] SAE, *J1708_201609: Serial Data Communications Between Microcomputer Systems in Heavy-Duty Vehicle Applications*, Society of Automotive Engineers, Warrendale, PA, September 2016.

[45] "SerialEvent | Arduino Documentation." [Online]. Available: docs.arduino.cc/built-in-examples/communication/SerialEvent/

[46] "J1708_T4," Github, 2022. [Online]. Available: github.com/davidnnaji/J1708_T4

Appendix A

Dual Air Brake Simulation System

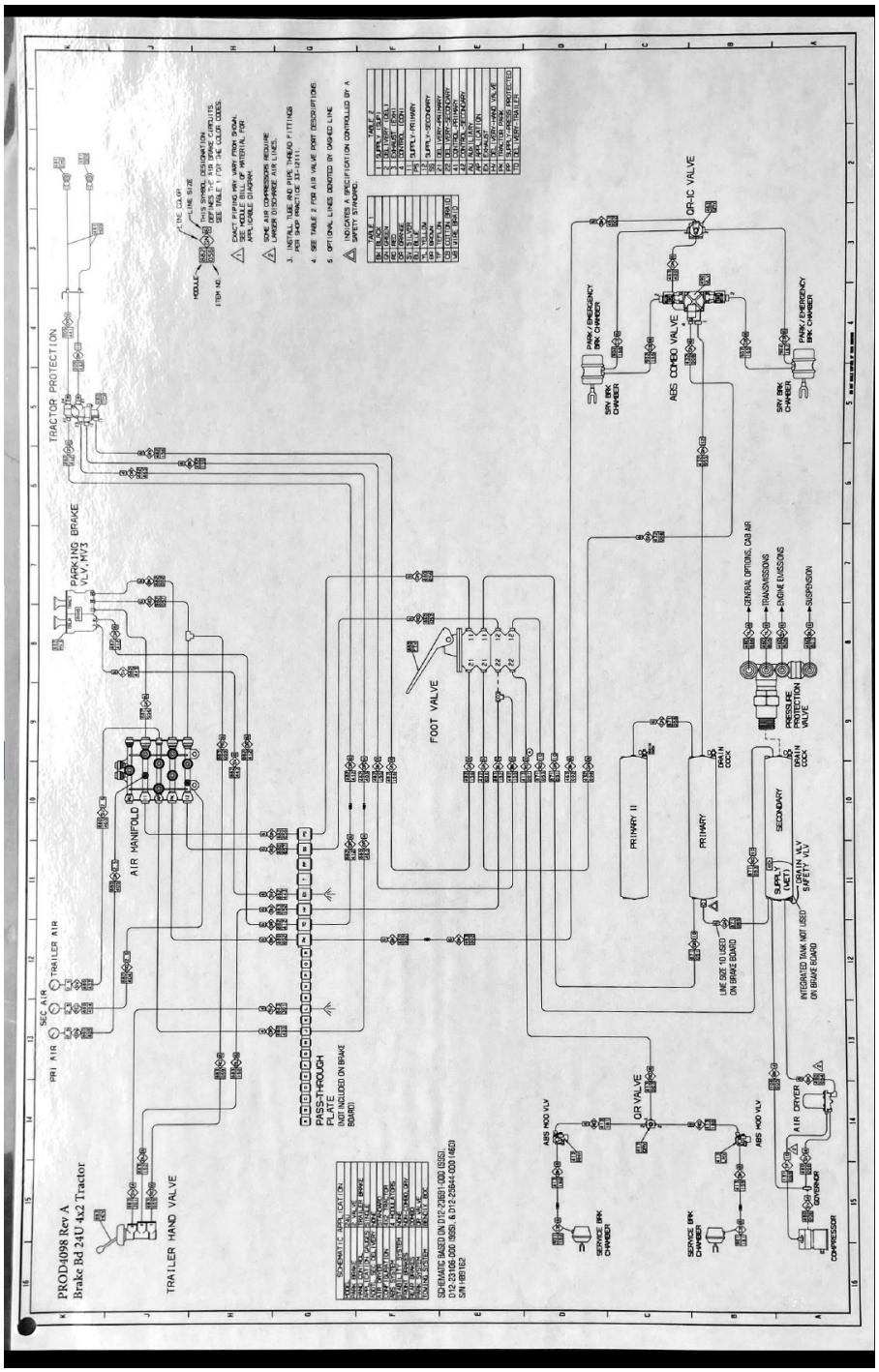


Figure A.1: DABSS Schematic

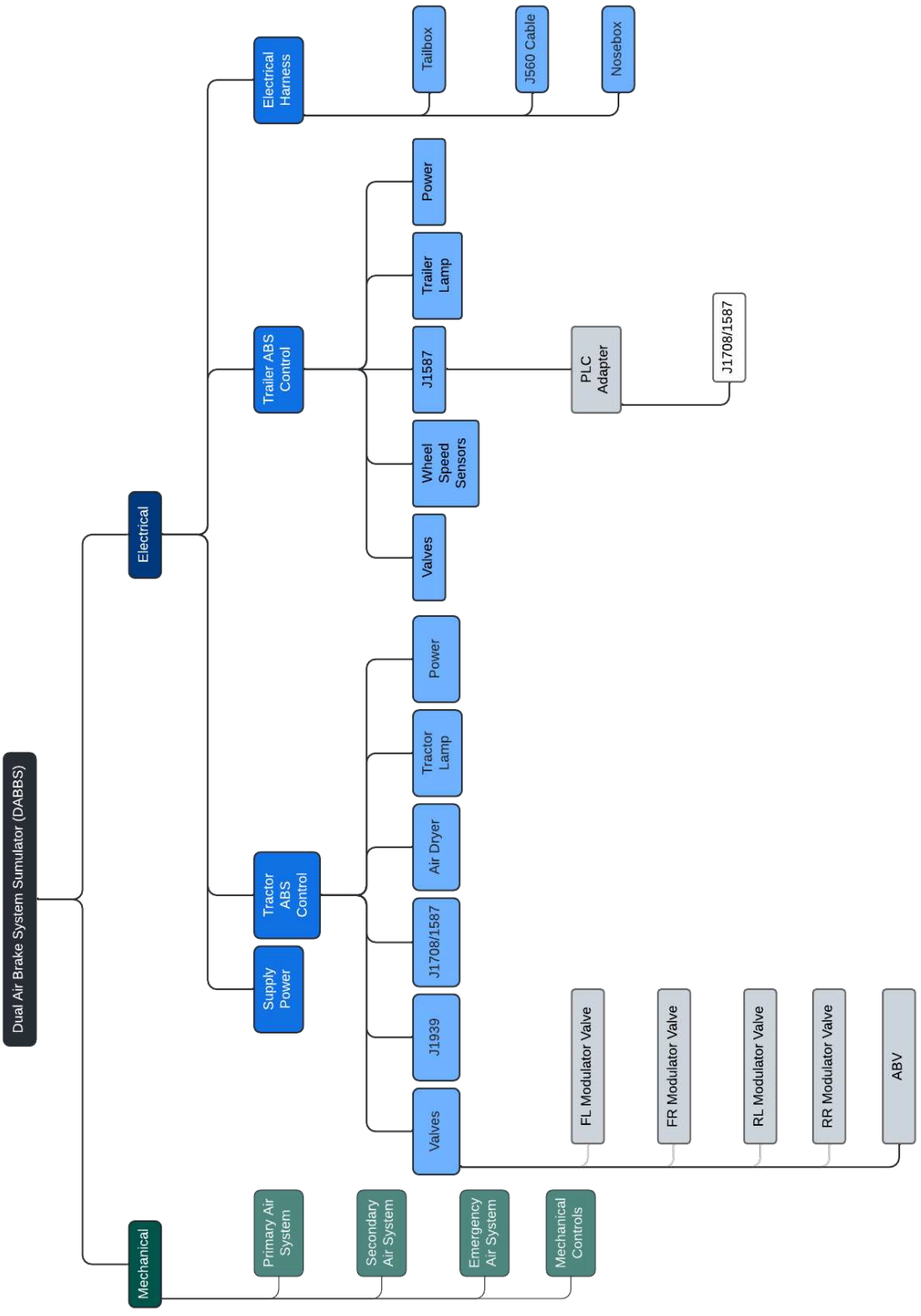


Figure A.2: "Target" Configuration for DABSS Retrofit

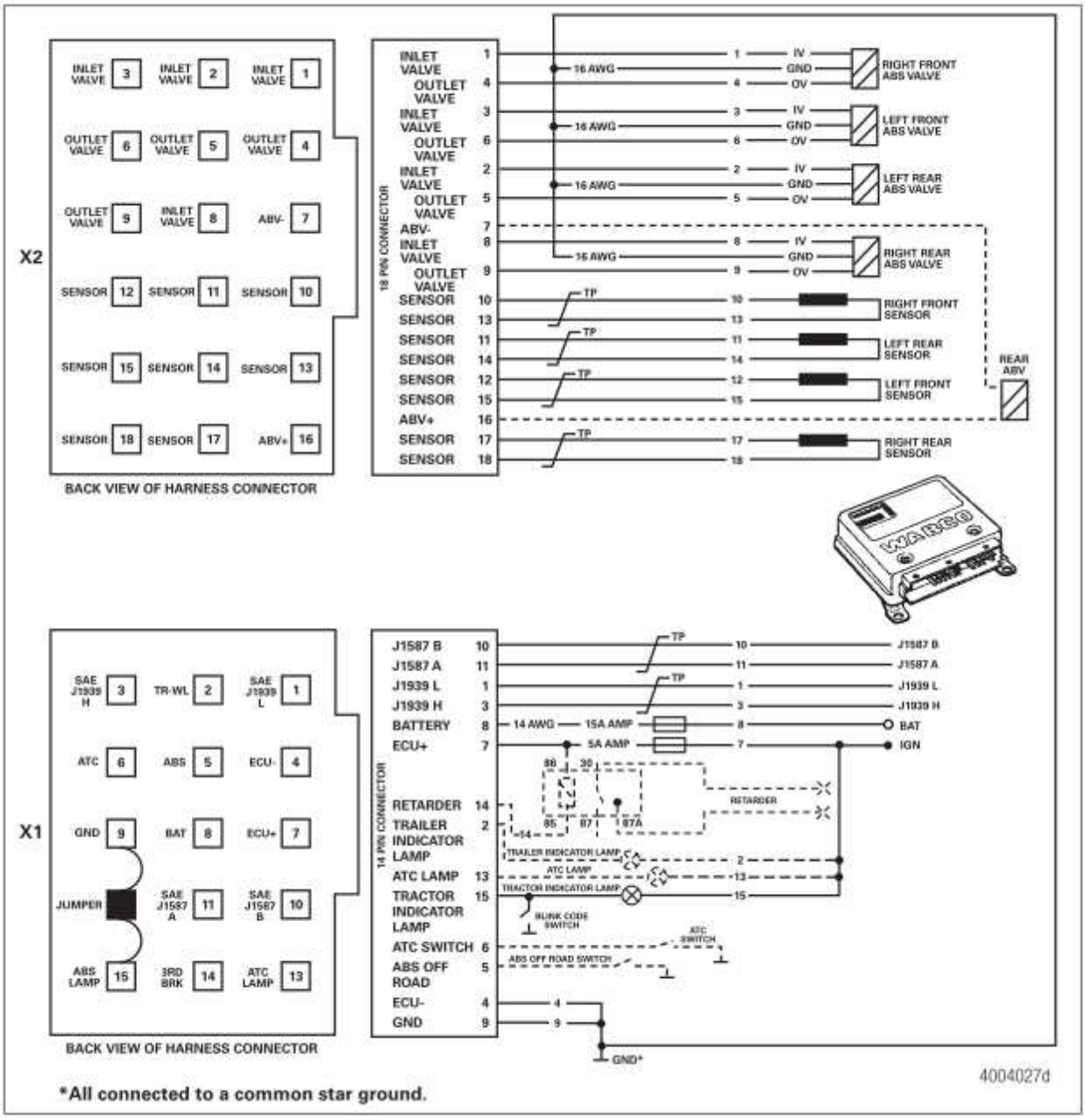


Figure A.3: WABCO Wiring Schematic

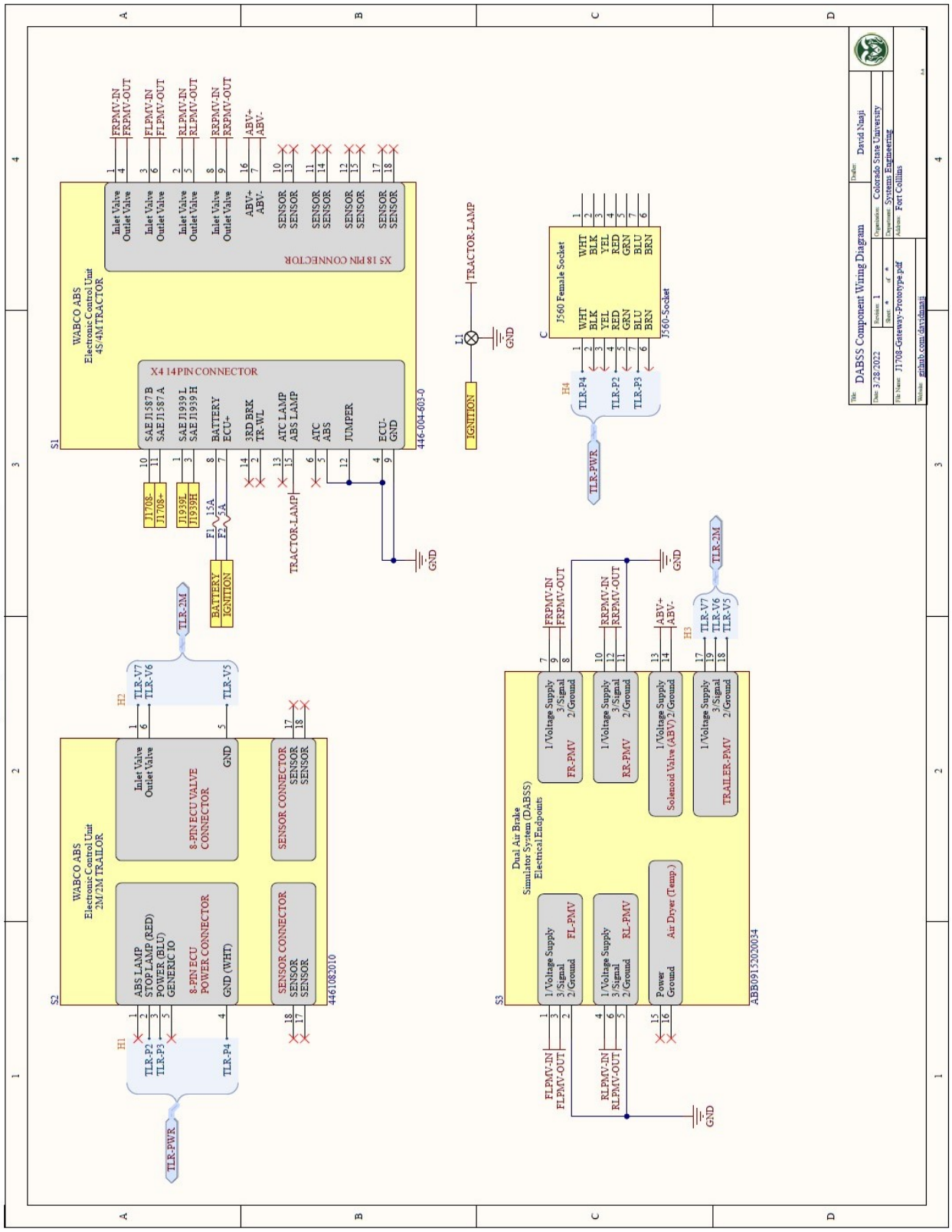


Figure A.4: DABSS ABS Retrofit Cabling Schematic

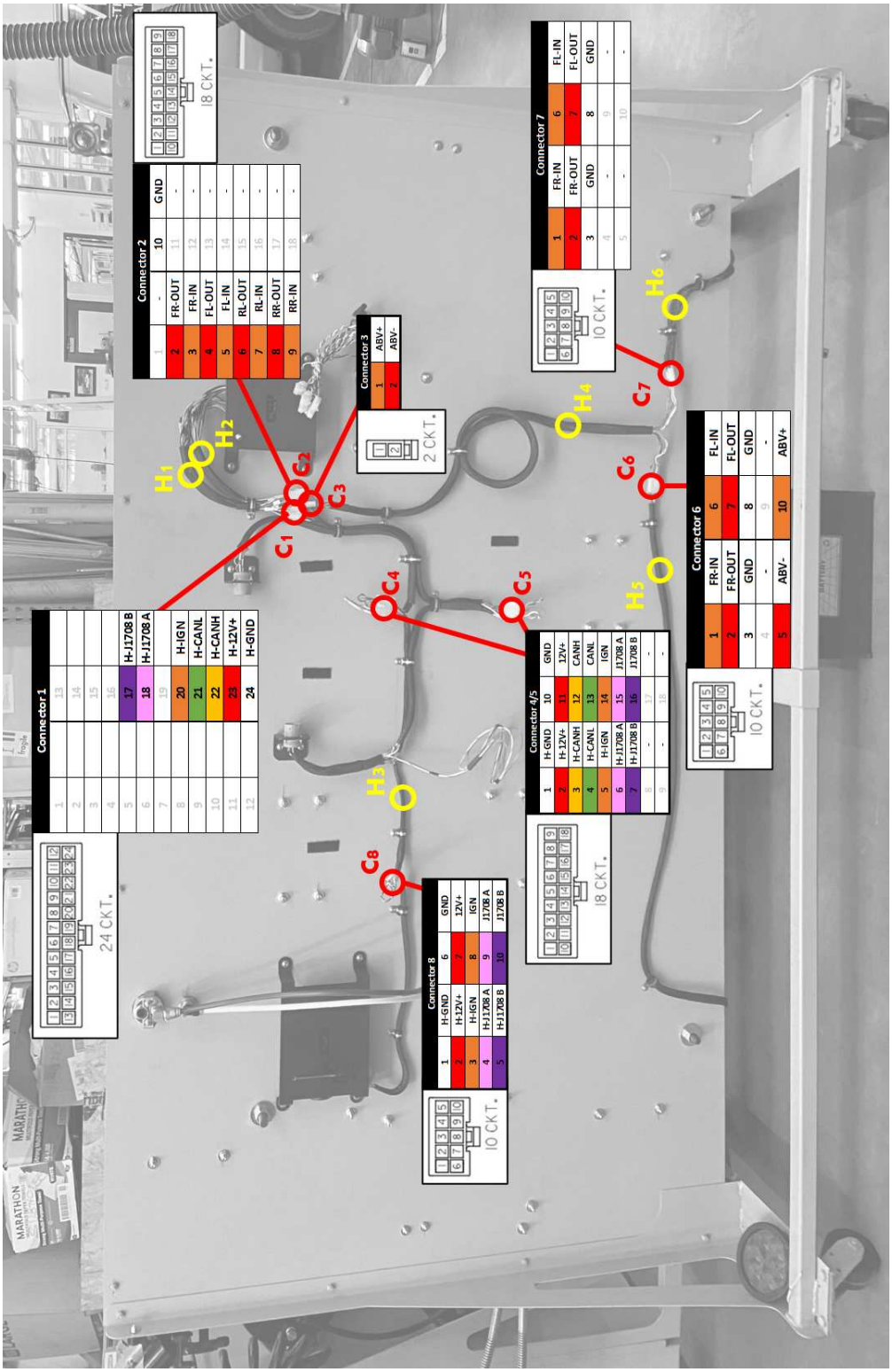


Figure A.5: DABSS Electrical Harness Diagram

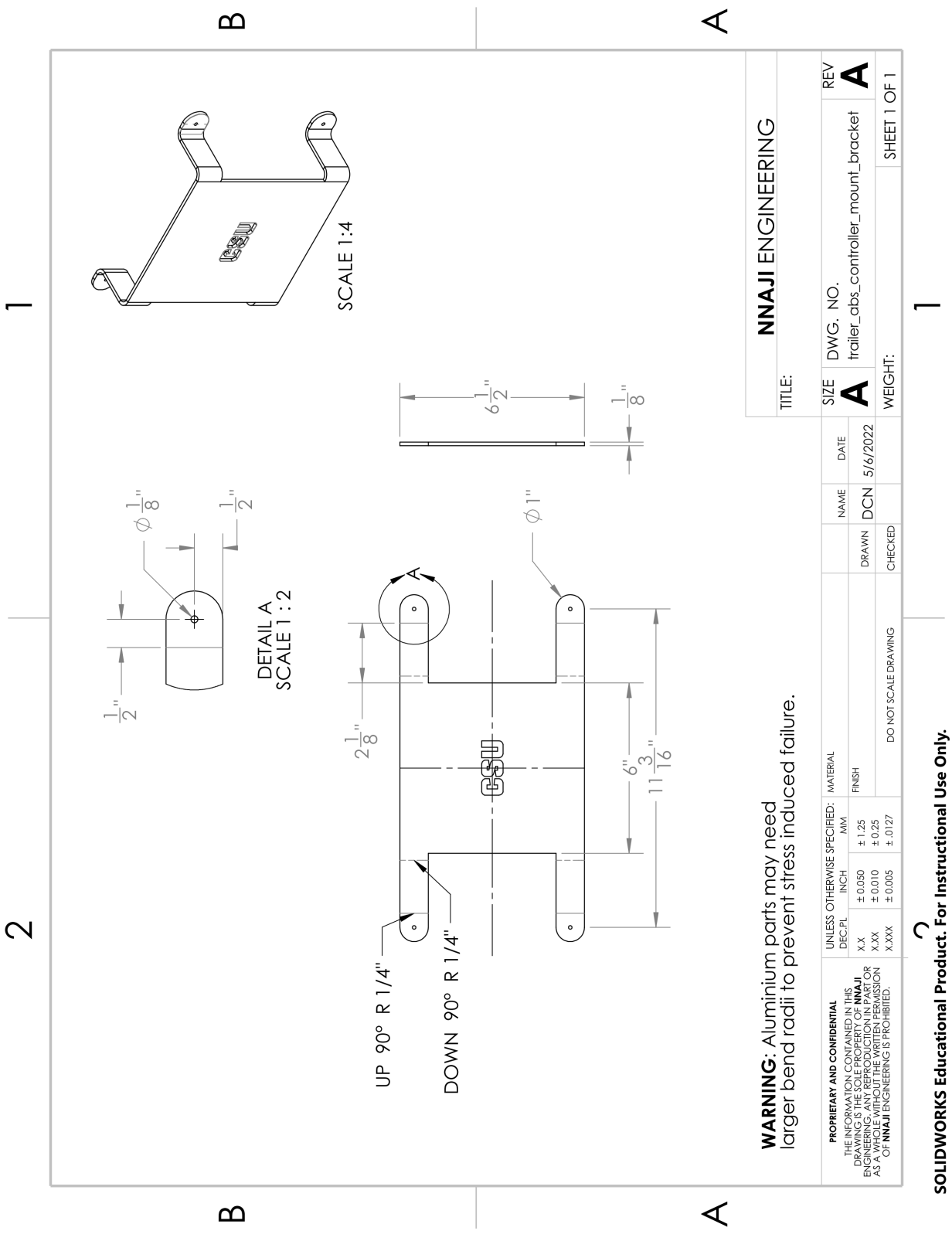


Figure A.6: Mechanical Drawing: Tractor ABS Controller Mounting Bracket

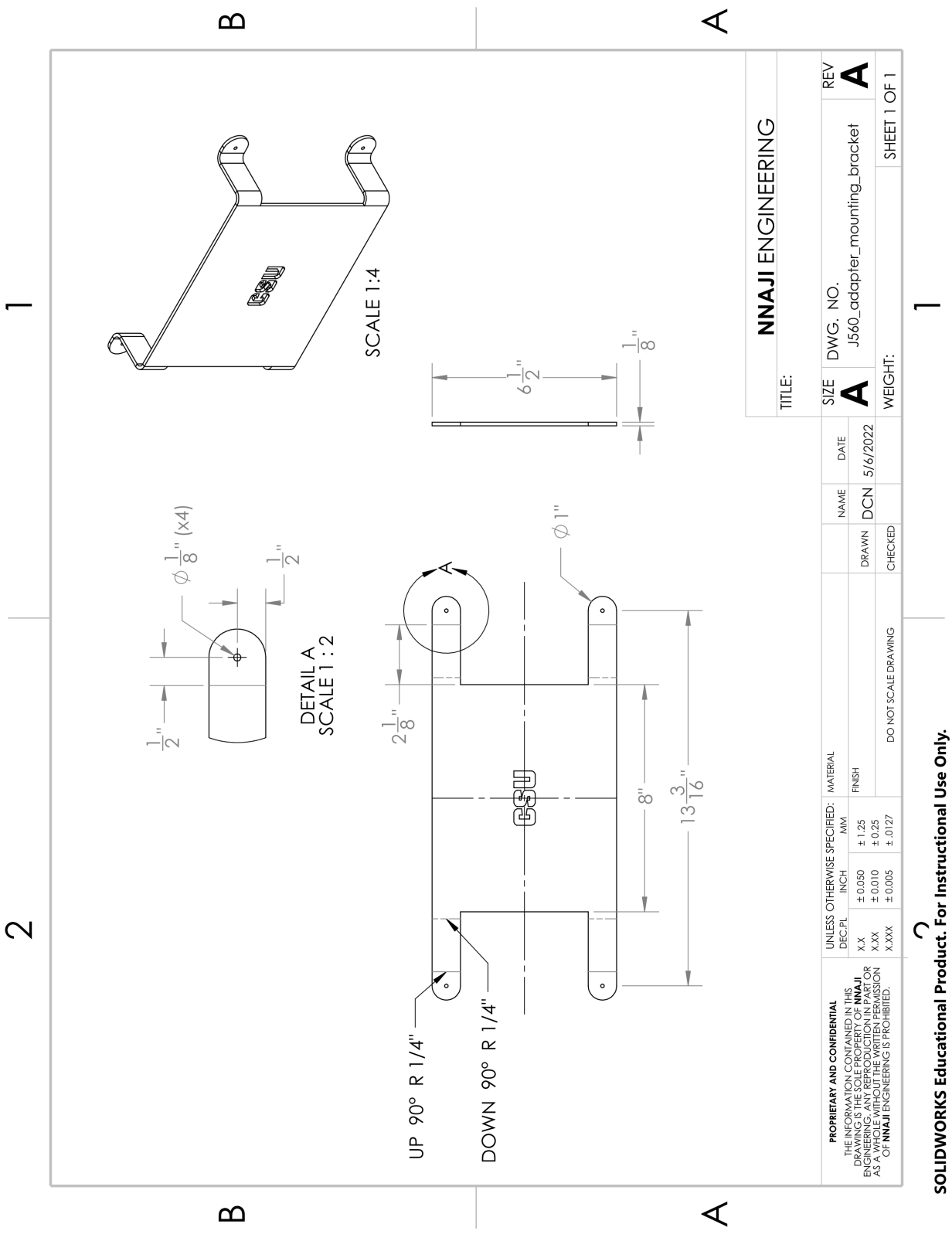


Figure A.7: Mechanical Drawing: J560 Adapter Mounting Bracket

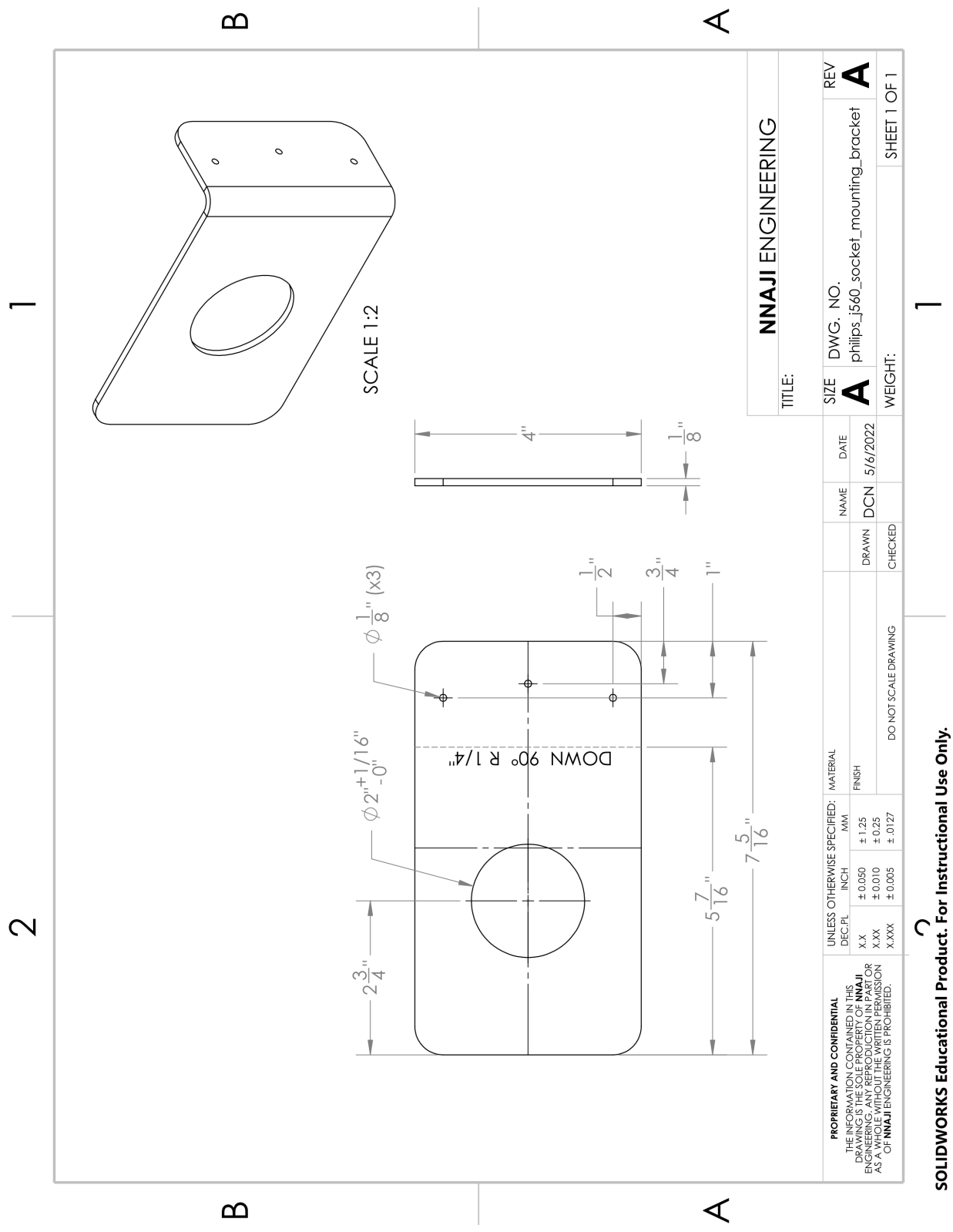


Figure A.8: Mechanical Drawing: J560 Socket Mounting Bracket

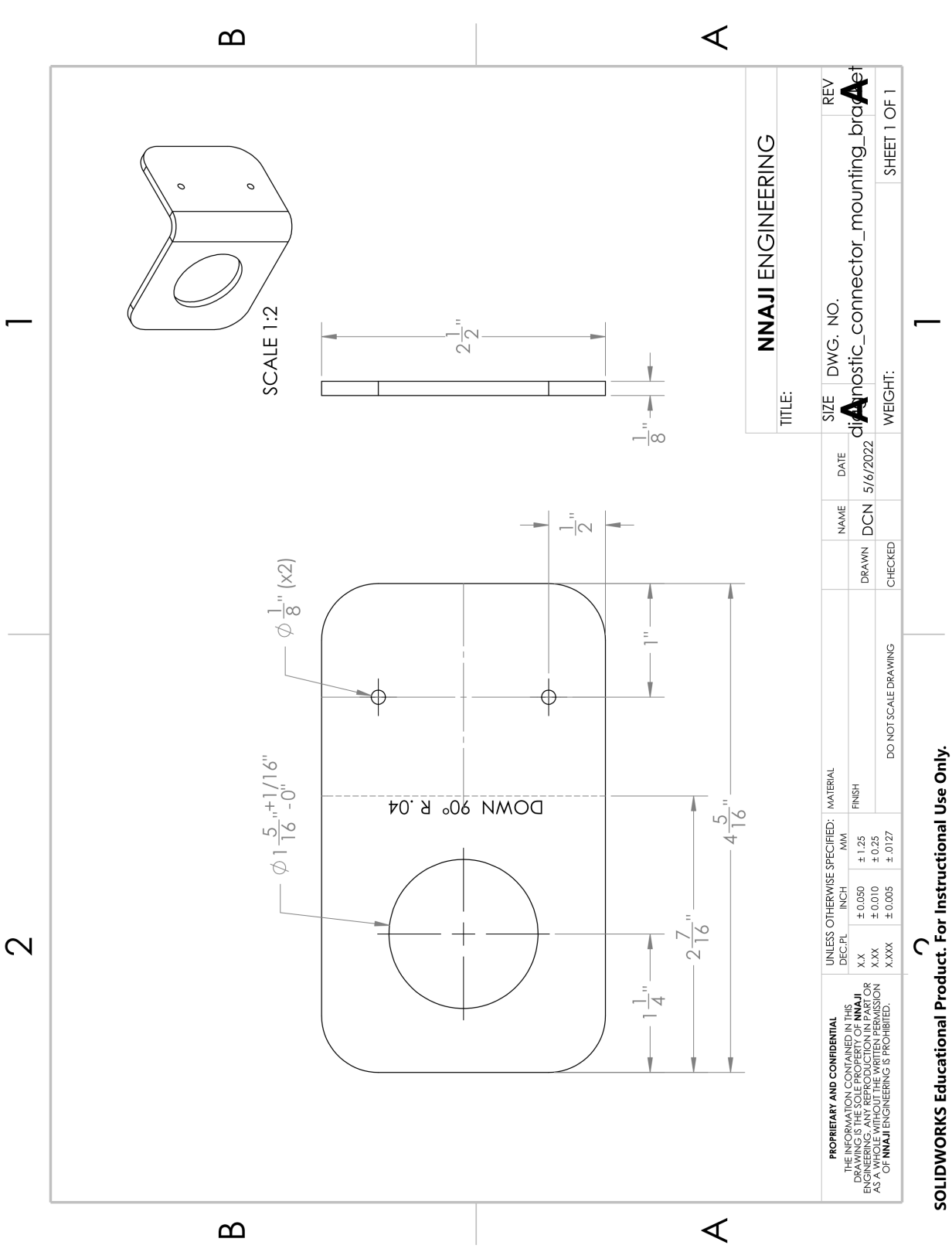


Figure A.9: Mechanical Drawing: Diagnostic Connector Mounting Bracket

Appendix B

Prototype Gateway

Table B.1: Hardware Processing Options

Family	Cost	Processor	Existing Truck Projects	Development Environment	External UART Count
Arduino	$23.00	ATmega328P	FALSE	Arduino IDE	1
Arduino	$20.70	ATmega328P	FALSE	Arduino IDE	1
Arduino	$20.70	ATmega32U4	FALSE	Arduino IDE	1
Teensy	$19.95	ARM Cortex-M7 (NXP i.MX)	TRUE	Arduino IDE	7
Teensy	$26.85	ARM Cortex-M7 (NXP i.MX)	TRUE	Arduino IDE	8
Espressif	$ 9.00	ESP32-S2-SOLO	FALSE	ESP-IDF SDK	1
Raspberry Pi	$ 4.00	ARM Cortex-M0+	FALSE	RP-Pico C/C++ SDK, RP-Pico Python SDK	2
STMicroelectronics	$10.99	ARM Cortex-M0+	FALSE	STM32 CubeIDE	2
BeagleBoard by TI	$65.14	ARM Cortex-A8	TRUE	Debian Linux, bone101	4
Raspberry Pi	$30.00	Quad core Cortex-A72	FALSE	Raspberry Pi OS	1

Table B.2: Hardware Selection Criteria and Scores

Board	Cost	Existing Truck Projects	Development Environment	External UART Count	Selection Score
Uno	2.6	0	4	0	6.6
Nano	2.7	0	4	0	6.7
Leonardo	2.7	0	4	0	6.7
Teensy 4.0	2.8	4	4	4	14.8
Teensy 4.1	2.4	4	4	4	14.4
ESP32-S2-DevKitC-1	3.4	0	1	0	4.4
Pico	3.8	0	2	0	5.8
Nucleo-32	3.3	0	0	0	3.3
Beagle Bone Black	0.0	4	2	4	10.0
Raspberry Pi 4	2.2	0	3	0	5.2

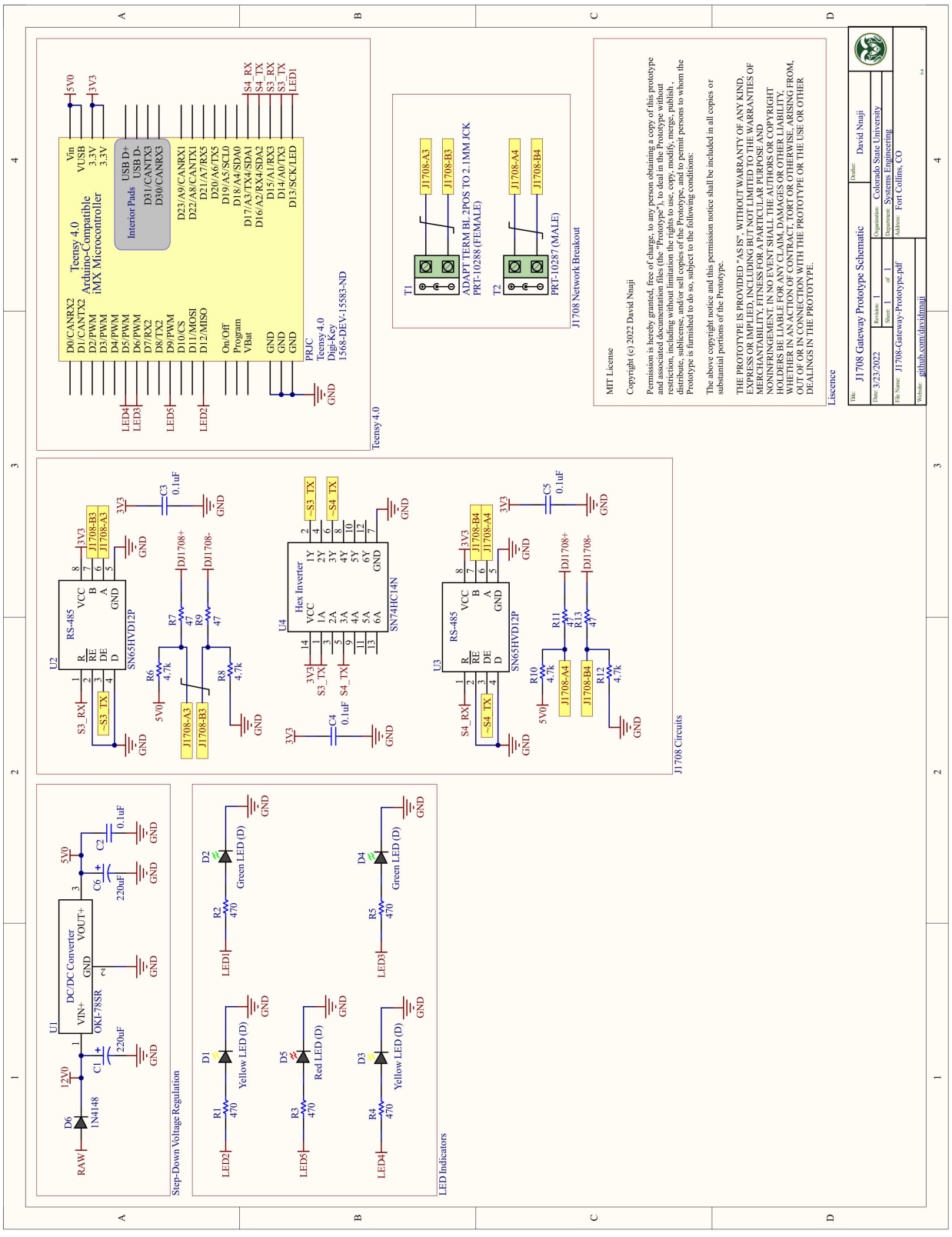


Figure B.1: Gateway Prototype Wiring Schematic

Figure B.2: Gateway Prototype Bill of Materials

Comment	Description	Schematic Designator	MFG Product Number	Qty	Manufacturer
220uF Capacitor	CAP ALUM 220UF 20% 50V RADIAL	C1	B41858C6227M000	1	EPCOS - TDK Electronics
0.1 uFCapacitor	CAP CER 0.1UF 100V X7R RADIAL	C2, C3, C4, C5	K104K20X7RH53H5	4	Vishay Beyschlag
Yellow LED (D)	LED YELLOW DIFFUSED T-1 3/4 T/H	D1, D3	LTL-4253	2	Lite-On Inc.
Green LED (D)	LED GREEN DIFFUSED T-1 3/4 T/H	D2, D4	LTL-4233	2	Lite-On Inc.
Red LED (D)	LED RED DIFFUSED T-1 3/4 T/H	D5	LTL-4223	1	Lite-On Inc.
Fast Switching Diode	DIODE GEN PURP 100V 200MA DO35	D6	1N4148FS-ND	1	onsemi
47 Ohm Resistor	RES 47 OHM 5% 1/4W AXIAL	R7, R9, R11, R13	CFR-25JB-52-47R	4	YAGEO
470 Ohm Resistor	RES 470 OHM 5% 1/4W AXIAL	R1, R2, R3, R4, R5	CFR-25JB-52-470R	5	YAGEO
4.7k Ohm Resistor	RES 4.7K OHM 5% 1/4W AXIAL	R6, R8, R10, R12	CFR-25JB-52-4K7	4	YAGEO
Teensy 4.0	32 Bit Arduino-Compatible iMX Microcontroller	Teensy 1	Teensy_4.0	1	PJRC
OKI-78SR	DC DC CONVERTER 5V 8W	U1	OKI-78SR-5/1.5-W36H-C	1	Murata Power Solutions Inc.
RS485 Transceiver	Single Transmitter/Receiver RS-485 8-Pin PDIP Tube	U2, U3	SN65HVD12P	2	Texas Instruments
Hex Inverter	Inverter IC 6 Channel Schmitt Trigger 14-PDIP	U4	SN74HC14N	1	Texas Instruments
Adapter Terminal Blocks	ADAPT TERM BL 2POS TO 2.1MM JCK	T1, T2	PRT-10288	2	SparkFun Electronics
Breadboard	Solderless 830pt breadboard	-	TW-E40-1020	1	TWIN INDUSTRIES
Jumper Wire	Assorted Jumper Wire Kit (Solid Copper)	-	MC001810	1	Brand: Global Specialties

Table B.3: Hardware Validation Tests

Component	Test Name	Status	Notes
J1708 Circuit 1	Rx Idle Line	✓	Power on the device via 12V and read a steady 0V on the B line with a probe.
J1708 Circuit 1	Tx Idle Line	✓	Power on the device via 12V and read a steady 5-4.5V on the A line with a probe
J1708 Circuit 1	Receive Data	✓	Received data bytes from a Bendix EC60 and CAT C15.
J1708 Circuit 1	Transmit Data	✓	Send a valid J1708 message on the Tx Line. Observe message in-transit on A and B lines with digital logic analyzer.
J1708 Circuit 1	Transmit Request Data	✓	Confirm a valid transmission by requesting PID 243 from a module. Response should show up.
J1708 Circuit 2	Rx Idle Line	✓	Refer to ”J1708 Circuit 1” Test
J1708 Circuit 2	Tx Idle Line	✓	Refer to ”J1708 Circuit 1” Test
J1708 Circuit 2	Receive Data	✓	Refer to ”J1708 Circuit 1” Test
J1708 Circuit 2	Transmit Data	✓	Refer to ”J1708 Circuit 1” Test
J1708 Circuit 2	Transmit Request Data	✓	Refer to ”J1708 Circuit 1” Test
Teensy 4.0	Serial Port Output	✓	Print a message to the serial port
Teensy 4.0	Serial Command Reading	✓	Send a message on the serial port. Observe command is processed and reported back.
Teensy 4.0, LED Indicators	LED Power Cycle	✓	Turn on and off LEDs 1-5 using GPIO.
Circuit Protection	Abrupt Discharge	✓	Observe rapid discharge
Circuit Protection	Abrupt High Power	✓	Observe voltage spike
Voltage Reg./Protection	Smooth Power Off	✓	Read steady 4-5V output with multimeter from the power circuit when device is connected to 12V input.
Voltage Reg./Protection	5V Output	✓	Read steady 5V output with multimeter from the power circuit when device is connected to 12V input.
All circuits	Prolonged operation	✓	Send a message every minute. Should observe 1440+ sent TX and RX message counts after 24 hour period and no errors.

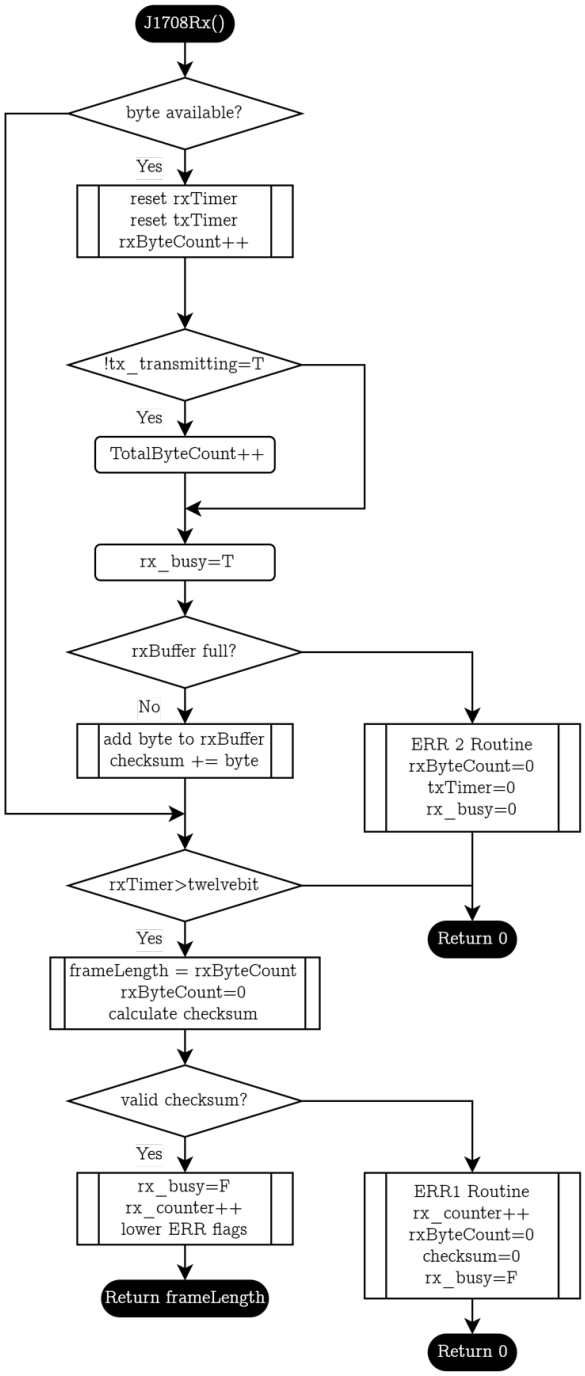


Figure B.3: J1708 Object Receive Message Flowchart

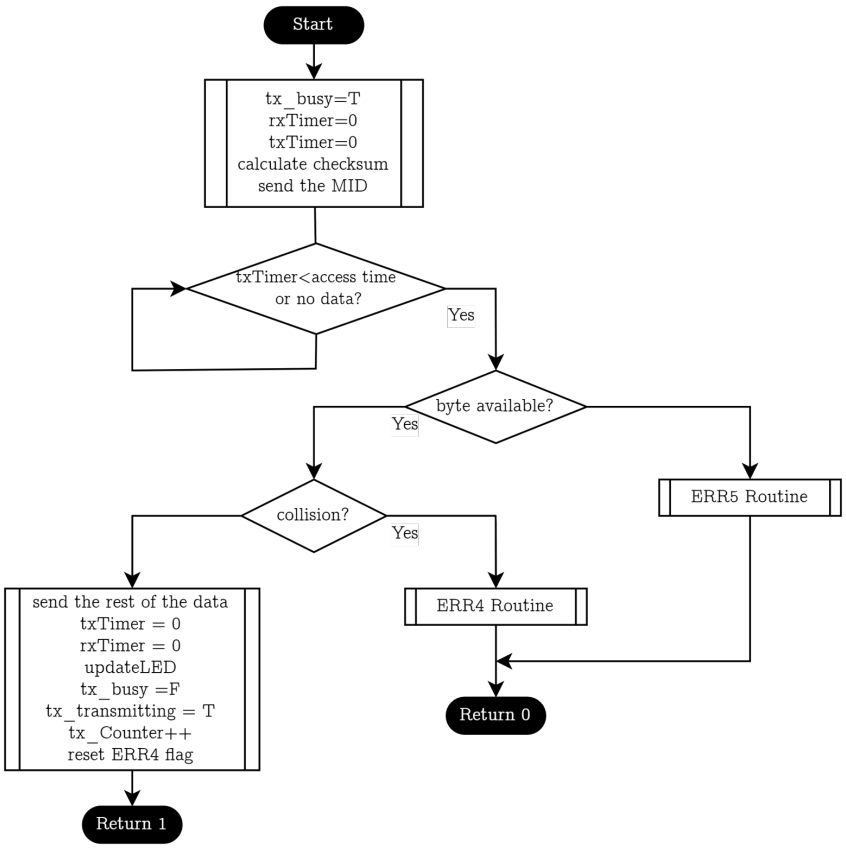


Figure B.4: J1708 Object Transmit Message Flowchart

Table B.4: Integrated Software 2^3 Factorial Test Results

Run No.	A	F	P	η
1	+	-	-	0
2	+	+	-	0
3	-	+	-	0
4	+	+	+	0
5	-	-	-	0
6	-	-	+	0
7	-	+	+	0
8	+	-	+	0
9	+	-	-	0
10	+	+	+	0
11	+	+	-	0
12	-	+	-	0
13	-	-	+	0
14	-	-	-	0
15	-	+	+	0
16	+	-	+	0

NETWORK SECURITY NOTIFICATION AND OCCURENCE COUNT
Used to indicate an abnormal message or node on the J1708/J1587 bus.

Parameter Data Length: 4 Characters
Data Type: Character 1 = Unsigned Short Integer
Character 2 = Unsigned Short Integer
Character 3 = Unsigned Short Integer
Character 4 = Unsigned Short Integer
Resolution: Binary
Maximum Range: 0 to 255
Transmission Update Period: On change or as needed
Message Priority: 8
Format:

PID	Data
762	n a b c1 c2

n— Number of parameter data characters = 4
a— Network Security Notification type
b— MID of the offending node
c— Data dependent on the Network Security Notification type

Network Security Notification Type 1: Message Spoof Alert (MSA)

Used by a legitimate host to notify nodes on the network that a message has been sent which did not originate from a known or trusted source.

DATA Two elements: The total number of observed occurrences message, c1 and c2 (transmitted least significant byte first; i.e., c2 is the least significant byte of the occurrence counter).

Network Security Notification Type 2: Imposter Node Alert (INA)

Used by a legitimate host to notify nodes on the network that an imposter node has been identified.

DATA One element: The total number of identified imposters on the bus, c1.

Network Security Notification Type 3: Abnormal Message Rate Alert (MRA)

Used by a legitimate node to notify hosts on the network that an abnormal message rate has been identified from a specific message identifier.

DATA None: -

Network Security Notification Type 4: Compromised Host Alert (CHA)

Used by a legitimate host to notify nodes it has been compromised.

DATA None: -

Figure B.5: PID 762 - Network Security Notification Definition

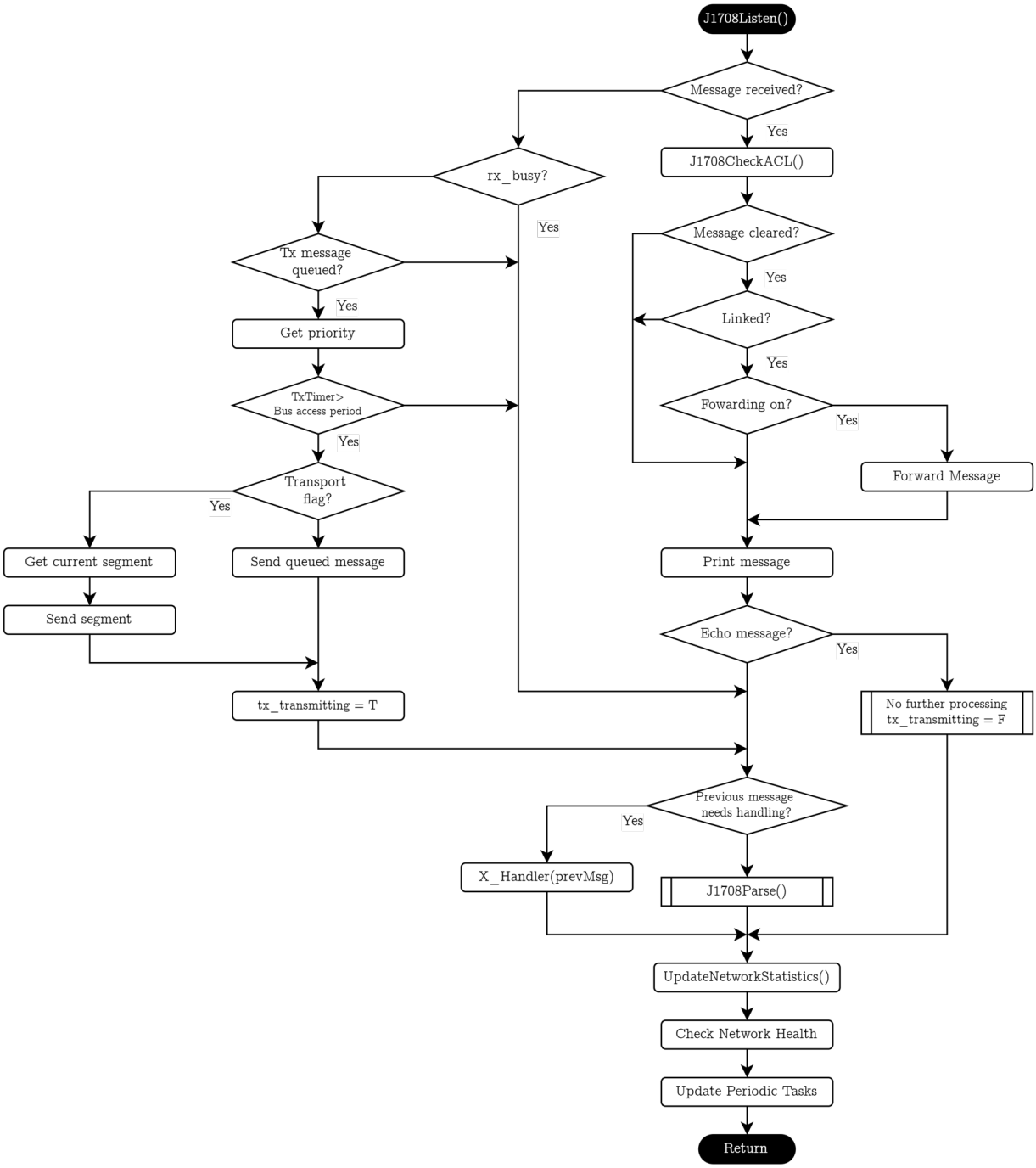


Figure B.6: J1708 Message Processing and Handling Flowchart

Table B.5: Software Unit Tests

Category	Unit(s)	Related SWR	Status
J1708 Object Setup	begin()	-	✓
J1708 Object Setup	link()	-	✓
J1708 Object Setup	unlink()	-	✓
Configuration and Test Controls	J1708Settings(), Command parsing	6,8	✓
Configuration and Test Controls	J1708Settings(), send command	1,3,6	✓
Configuration and Test Controls	J1708Settings(), config command	6,7,8	✓
Configuration and Test Controls	SerialEvent()	6	✓
Configuration and Test Controls	getValue()	6	✓
Configuration and Test Controls	stringtoHex()	6	✓
Configuration and Test Controls	hex2int()	6	✓
Receive J1708	J1708RX() Normal	1	✓
Receive J1708	J1708RX() ERR1	1,4	✓
Receive J1708	J1708RX() ERR2	1,4	✓
Schedule and Transmit J1708	J1708Send()	1,3	✓
Schedule and Transmit J1708	J1708Tx() Normal	1,3	✓
Schedule and Transmit J1708	J1708Send() ERR3	-	✓
Schedule and Transmit J1708	J1708Tx() ERR4	1,2,3	✓
Schedule and Transmit J1708	J1708Tx() ERR5	1,2	✓
Message Firewall	checkACL()	7,10	✓
Message Firewall	resetACL()	7	✓
Message Firewall	updateACL()	7	✓
Message Firewall	checkACL() ERR7	11,12	✓
Message Processing and Handling	Parse() ERR7	14	✓
Message Firewall	checkACL() ERR8	13	✓
Message Processing and Handling	Parse() ERR8	15	✓
Bus Load Monitoring /Message Rate Enforcement	J1708RX() Byte Counter	16	✓
Bus Load Monitoring /Message Rate Enforcement	UpdateNetworkStatistics(), Baud Calculation	16	✓
Bus Load Monitoring /Message Rate Enforcement	J1708CheckNetwork() ERR6	17	✓
Bus Load Monitoring /Message Rate Enforcement	UpdateNetworkStatistics(), MID% Calculation	18,19	✓
Bus Load Monitoring /Message Rate Enforcement	J1708CheckNetwork() ERR9	18,19,22	✓
Bus Load Monitoring /Message Rate Enforcement	J1708CheckNetwork() ERR9 Reporting	18	✓
Message Processing and Handling	Parse() ERR9	20	✓
Bus Load Monitoring /Message Rate Enforcement	J1708CheckNetwork() ERR10	19,22	✓
Bus Load Monitoring /Message Rate Enforcement	J1708CheckNetwork() ERR10 Reporting	19	✓
Message Processing and Handling	Parse() ERR10	21,22	✓
J1587 Transport Protocol Utility	J1708Transport(), RTS	3,5	✓
J1587 Transport Protocol Utility	J1708Parse(), Transport Message Parsing	3,5	✓
J1587 Transport Protocol Utility	CTS_Handler()	3,5	✓
J1587 Transport Protocol Utility	RTS_Handler()	3,5	✓
J1587 Transport Protocol Utility	CDP_Handler()	3,5	✓
J1587 Transport Protocol Utility	EOM_Handler()	3,5	✓
J1587 Transport Protocol Utility	Abort_Handler()	3,5	✓
J1587 Transport Protocol Utility	J1708CheckNetwork() Connection Status	3,5	✓
J1587 Transport Protocol Utility	J1708CheckNetwork() Connection Abort	3,5	✓
Message Processing and Handling	J1708Listen() Transport Segment Tx Queue	3,5	✓
Message Processing and Handling	J1708Listen() Serial output	6,7,8	✓
Message Processing and Handling	J1708Listen() Security LED	9	✓
Message Firewall	J1708Listen() Message Forwarding	7,10	✓