Rapidly Integrating Vehicle Computers Using Simulation

Matthew Rogers[*]
Department of Computer Science

Abstract

Millions of vehicles rely on embedded computers to function. Integrating these computers is a complicated process, more so as different manufacturers try to work together. We develop the CAN Simulation Format (CSF), publishable by these manufacturers. A designer can use this CSF to mix and match ECU configurations, simulating CAN traffic for each on the fly. This lets the designer test every configuration without access to prototypes of every other computer. We implement this simulation on hobbyist hardware, allowing us to test and develop with realistic CAN traffic. As different computers are updated, this technology enables the system maintainer to test different permutations of installed firmware versions, without modifying the actual computers.

1. INTRODUCTION

Millions of vehicles rely on embedded computers to operate. These computers often have limited resources, with specialized hardware for upgrading and connecting to them. A complicated supply chain, inherent to the multitude of manufacturers, results in a series of problems for both the system designer, and each manufacturer.

A core problem for any component developer is integration testing. There needs to be testing whenever any embedded component upgrades firmware, but for systems with loose configurations each of the embedded devices could be on any firmware version. To solve this, a developer needs to test against all permutations of major firmware versions for complete test coverage. For modern vehicles this is a pipe

[*]matthew.rogers@cs.ox.ac.uk.

dream. Testing against one configuration is complicated, much less systems in varying patch states.

Currently developers queue for access to a software integration lab, or real vehicle. More advanced groups may take a capture of the CAN Bus and replay it with their ECU connected to simulate real traffic, but they would need to do this for every possible configuration, or guess at pruning the data. In applications with a limited supply, such as military systems, getting any level of access is incredibly difficult.

To solve this problem we propose a standardized configuration document called the CAN Simulation Format (CSF), which all ECUs publish so that anyone can simulate their traffic. We make this feasible for existing systems by automatically converting captured CAN traffic into this format for each ECU on the network. Using the CSF a developer can simply decide on which ECUs they would like on the system, and generate simulated traffic specific to that configuration. Using a 90 minute data capture we produce the CSF, generate CAN messages, and transmit them in real time across a CAN interface.

We will open with a brief background on CAN how a CAN Bus functions in Section 2. This is followed by an analysis of different dynamic testing techniques, and different ways to generate our CSF in Section 3. We then describe our testbed, experiment and results in Sections 4, 5 ,6, before providing future work and concluding in Sections 7 and 8.

2. BACKGROUND

Since the 1980s, automobiles rely on a physical standard known as the Controller Area Network (CAN) [1]. Now, embedded computers known as Electronic Control Units (ECUs) control the vehicle. These computers must share data and receive commands, so they are all connected to the CAN network, broadcasting messages.

As vehicles get more complex to satisfy demands for safety, predictive maintenance, and other functionalities, the number of ECUs speaking also increases. Modern industrial standards such as J1939 [2] limit that number to 30 ECUs. Each ECU transmits any number of message IDs, with the possibility of two ECUs transmitting the same ID. This ID is 29 bits. During these 29 bits any number of ECUs may attempt to speak. However, in CAN 0 is considered the dominant bit, meaning any ECU trying to transmit a 1 while reading a 0 over the bus line will stop transmitting. This bit level arbitration process is a key feature of CAN, ensuring higher priority messages are transmitted in a timely manner instead of a straight queue.

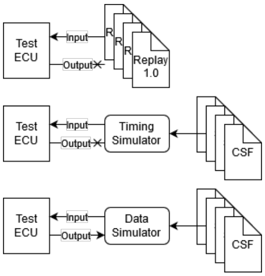

Figure 1: *This figure depicts differences between each method of CAN dynamic testing.*

3. DYNAMIC TESTING WITH CAN TRAFFIC

We identify three ways of doing dynamic testing with CAN Traffic:

- Replaying Traffic
- Simulating Traffic based on ECU timing: messages transmit at known time intervals based on documentation
- Simulating Traffic based on CAN data: the data going over the bus, such as RPM, modifies what simulated data should be sent

A simplified diagram of the different approaches can be seen in Figure 1. Replaying traffic is the simplest option, and provides realistic data inputs for the testing ECU, but the replay will not respond to data output by our ECU. Simulating traffic based on pure timing data has a similar problem. Data parameters can be specific for testing, but pure timing is a naive approach. The core difference is that capturing replays, with details on the ECU configuration, is difficult. We would need a replay for every possible system configuration to accurately reflect bus traffic, this approach does not scale. If we assume we have a CSF to read, timing based simulations provide more flexible input testing for our ECU. With vague requirements being a reality of any engineering task, being able to test several ECU configurations greatly reduces turn around time.

The third level of testing concerns a high fidelity simulation of CAN traffic, where the rest of the network modifies their messages in response to the output of our ECU. Hard coding this for one system is simple, but the nature of ECU configurations makes this complex to do dynamically. For example, an Engine Control Module (ECM) may send just engine data, brake and engine data, or a number of other auxillary data points. One ECU could handle everything, but the system configuration could call for a separate brake controller. This level of fidelity requires tying each data parameter to another. However, many of these data parameters change based off internal logic, not the data broadcast by another ECU. Identifying these parameter dependencies is perhaps possible, but prone to errors without reverse engineering the firmware of each ECU, or having data directly from the manufacturer. Given this, we focus on the second option, with the intent to upgrade to the third as more data is available.

3.1 CAN Simulation Format

For each source address seen in a data capture we identify the message ID and corresponding average interval between messages, destination address, and priority; these features represent all of the data we need to craft the arbitration ID of a CAN packet. We prepend the ECU name to each source address. If we assume industrial standards such as J1939 we can automatically parse this data by sending a request to with data 0x00EE00 to identify the manufacturer, while using the standardized functions for source address numbers to determine the purpose (e.g., Engine, Brakes, Transmission). This data is sufficient for emulating bus controller message prioritization while evaluating the test ECU's ability to read and react to normal message flow from the CAN Bus.

If we do not have a data capture we have a two options. If we have temporary physical access to a system we can extract the firmware, performing static analysis to extract the messages it sends. For consumer CAN this would be impractical, as the purpose of each message ID are non-standardized. For industrial standards like J1939, we could look for the 1892 provided message IDs. But perceiving anything beyond the sent IDs would take a great deal of time, requiring the manual reverse engineering of each data parameter. Extracting only message IDs implies we do not have any design requirements pertaining to what data is sent over the bus.

Our second option assumes we have design requirements. If we only know the messages we are expected to read, and the messages we are expected to send across the bus, we can still recreate the CSF. We know the relevant arbitration ID fields, can test a frequent, moderate, and delayed timing interval, and can randomize each parameter in the data field. If each developer did this, then combined their CSFs, we would generate a full system simulation.

As existing systems develop a backlog of CSFs, additions and upgrades to that system can rely on them for development. Meanwhile the system maintainer can ensure different permutations of ECU firmware don't result in bugs. Normally each ECU would be attached to the analog components for a software integration lab. Instead our simulator, powered by CSFs, could connect to every analog component.

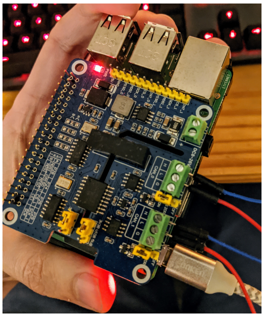

Figure 2: *Raspberry Pi 3 testbed with Waveshare 2-CH CAN FD HAT. Both CAN channels are connected to each other to detect transmitted messages over the CAN interface.*

This allows for quickly changing between ECUs without modifying any real hardware, applying DevOps techniques to embedded system development.

3.2 Emulating a CAN Bus

To make the ordering of messages realistic, we must emulate the collision avoidance done by multiple ECUs, despite being one device transmitting. To do so efficiently we use an event manager to enqueue messages as their average timing approaches. The queue is a priority queue based on a minheap, as it provides O(log N) time complexity for our important operations: appending messages, and extracting the minimum message. Our CAN message object uses an overwritten less than operator, such that the minheap uses the arbitration ID like a normal CAN collision avoidance system [3]. A test ECU connected to our CAN network will use the normal arbitration process. From the physical layer this will look like two ECUs communicating, towards the application layer the packets will appear the same.

4. METHODOLOGY

We extracted our CSF from 90 minutes of truck data using a Python script. To test our generated CAN messages we used a 2 channel CAN hat on a Raspberry Pi. We transmitted our generated CAN messages across channel 0, monitoring channel 1 to analyze our simulated traffic. A picture of this test bed can be seen in Figure 2.

5. EXPERIMENT

To evaluate the accuracy of our data we compare the timing interval of each message against that of the original data capture. If all of the same message IDs are visible across the network, at a rate frequent enough to prevent data starvation by the test ECU, we believe the CSF can be extended to any permutation of ECU configurations for testing and development against representative CAN data.

6. RESULTS

We observed a similar distribution of message timing between our 90 minute data capture and simulation. However, the simulation was unable to account for messages with timing intervals that changed with the data going over the bus. An example being a message varying the transmission rate with the speed of the engine. As long as the test ECU is able to process all simulated messages at the fastest transmission speed possible, usually set at 10ms [3], the test ECU will be adequately tested for input handling.

7. FUTURE WORK

Easier testing leads to greater software quality assurance. We achieve this with a relatively simple simulation now, but can do more with a responsive, data based solution. The benefits to testing the full system are obvious. Instead we focus on the cyber security elements for automotive security systems.

If high fidelity CAN traffic can be simulated, is it possible to distinguish between simulated traffic and real car traffic in real time? Without this capability an attacker can feed fake CAN data to any patch-based security solutions. By continuously confirming the CAN data is real, future work could prevent dynamic testing, and tampering of automotive security systems by attackers. This has potential auxiliary benefits for research that relies on data coming from a real vehicle. Spoofing insurance data, or location data is more difficult if it has the assurance of coming from a real vehicle.

Simulating the entire software integration lab means our simulator is capable of acting as the computer for any device attached to the CAN Bus. This lets us act as a redundant device. This enables system recovery post an intrusion prevention system being activated, or for any ECU faults, without the cost of installing another engine block.

8. CONCLUSION

Testing each configuration of the ECUs that control modern vehicles is difficult. To counter this problem we propose a standardized file format, published by each manufacturer containing all elements of the arbitration ID, a distribution of any message timings, and initial setup information. Until manufacturers publish this standardized file, defenders extract the relevant fields from a packet capture of an existing vehicle. However, capturing this data does not scale for complex systems which update frequently, or have multiple ECU configurations.

We demonstrate the ability to extract this information from an existing capture, and create a simulation of traffic based on the created CSF. This simulation ignored data dependencies, but otherwise matched timing distributions, while simulating the message arbitration process of multiple ECUs speaking over each other. As we gain more data to create more CSFs we can simulate more traffic. When we encode how different data fields change into our format, we will simulate the entire system, simplifying application layer testing.

References

[1] Robert Bosch. 'CAN specification version 2.0'. http://esd.cs.ucr.edu/webres/can20.pdf.
[2] Society of Automotive Engineers standard.
[3] Wilfried Voss. Guide to SAE j1939 - controller area network and j1939, 2018.