Multi-Bit Error Vulnerabilities in the Controller Area Network Protocol

Eushiuan Tran
Advisor: Dr. Philip Koopman
Carnegie Mellon University
Pittsburgh, PA

May 1999

Abstract

Embedded networks will increasingly be used in safety-critical applications such as drive-by-wire automobiles. Because of potentially high network noise in such systems, reliably detecting bit errors could become vital to preventing the dissemination of corrupted data. Unfortunately, an interaction between bit stuffing and use of a cyclic redundancy code (CRC) can create a vulnerability to undetected multi-bit errors. Simulations of the widely used Controller Area Network (CAN) protocol indicate that this problem can cause a double-bit error to result in a 1.3×10^{-7} probability of undetected corruption. This number, although small, becomes an issue when magnified by a fleet size of hundreds of millions of vehicles. This vulnerability and related CAN specification problems can be fixed, albeit at a cost. A generalized lesson is that transmission encoding can undermine the effectiveness of error detection codes to the point that a system might not provide a required level of robustness.

I. Introduction

Embedded communication networks are becoming prevalent in distributed embedded systems, and are poised for widespread use in safety-critical applications. In these days of increasing electronic content in many products, embedded networks provide greater design flexibility, reduce wiring complexity, and potentially reduce system cost compared to discrete wiring approaches. These advantages make it desirable to use embedded networks even for safety critical applications, but also require careful consideration of system safety issues. For example, the objective of the X-by-Wire project [Dilger 98] is to perform safety-critical vehicle control using redundant electronics connected by a reliable (and dual redundant) real-time embedded network, but dispensing with mechanical backup devices.

Unfortunately, traditional fault tolerant system design experience is not entirely applicable to mass-produced consumer applications. Techniques for constructing dependable networks have been generally developed in traditional critical application areas such as aerospace, nuclear, and military systems. In particular, the massive scale of deployment for consumer products means even very improbable events (from a single unit point of view) are likely to happen on a regular basis somewhere in a large deployed fleet. For example, the U.S. automotive fleet logs approximately four orders of magnitude more operating hours annually than the U.S. commercial aviation fleet. Thus, a failure that is extremely improbable in an aviation setting (a failure rate of 10^{-9} per hour) can be expected to happen approximately once in the fleet every 73 years; yet that same failure rate will result in a failure every 4.5 days in the automotive fleet. [Koopman 98] Thus, small, acceptable failure rates in traditional fault tolerance applications may not be small enough to ensure safety in consumer applications.

A particular source of potential problems is undetected network errors. The operating environment under the hood of a car is notoriously harsh and electronically noisy. Furthermore, the tight cost constraints on consumer products dictates that the bare minimum possible shielding and noise suppression hardware be used. Normally, error detecting codes can identify messages corrupted with modest numbers of bit errors, so current designers do not worry about this issue. But what if the assumption of effective error detection is incorrect? Designs could be produced in which corrupted data is mistaken for valid control information in a safety-critical system. Clearly, it is important to design systems so that the chance of corrupted data being mistaken for valid data is vanishingly small, even in the context of a large automotive fleet.

Unfortunately, the controller area network (CAN) protocol, which is specifically designed for automotive control applications, has a vulnerability to undetected multi-bit transmission errors. Despite the use of a Cyclic Redundancy Code (CRC), the CAN protocol can be expected to accept a small fraction of messages with double-bit errors (and, in general, any multi-bit error) as valid under realistic operating conditions. In this paper, we will demonstrate that CAN's use of bit stuffing (a typical bit-level encoding mechanism) actually undermines the effectiveness of the CRC error detection mechanism. While the effect is too small to be seen in most typical laboratory tests or small-scale field trials, it is possible that it could cause system failures when employed in a safety-critical role on a full-size automobile fleet.

The remainder of the paper describes the details of the vulnerability found in CAN, how frequently it can be expected to occur, and several potential solutions for systems using the existing protocol and any potential next-generation CAN protocol design. Section 2 presents a summary of the CAN protocol and previous work in CAN robustness and vehicle network failure rates. Section 3 shows how bit stuffing undermines CRC effectiveness. Section 4 describes an experimental methodology for predicting failure rates from the problem. Section 5 compares the simulation results to analytical results from previous research. Section 6 contains simulation results and analytic verification. Section 7 suggests potential improvements for current systems and future CAN protocols. Finally, Section 8 presents conclusions and opportunities for future research.

II. Background and Prior Work

A. Controller Area Network

The Controller Area Network is a low-level serial data communications protocol for embedded real-time applications. [Bosch CAN 91] [SAE CAN 90] [web CAN 98] It was originally developed by the German company Robert Bosch GmbH for use in cars as an alternative to expensive and cumbersome wiring harnesses. The transmission medium is usually a pair of copper wires. Although fiber optic implementations exist, they are too expensive for general automotive use. CAN operates at speeds of 100K to 1M bits/second. Data bits are sent in two states: dominant (a logic 0) and recessive (logic 1). Transmission hardware is designed in such a way that if two transmitters attempt to assert data simultaneously, any transmitter asserting a dominant bit will prevail over transmitters attempting to send a recessive bit.

CAN data is transmitted and received using formatted message frames. There are two protocol versions in widespread use: 2.0A which supports 11-bit message identifiers and 2.0B which supports both 11-bit and 29-bit identifiers. Without loss of generality, the work presented in this paper uses the 2.0A protocol with a frame format as shown in Figure 1. [Bosch CAN 91]

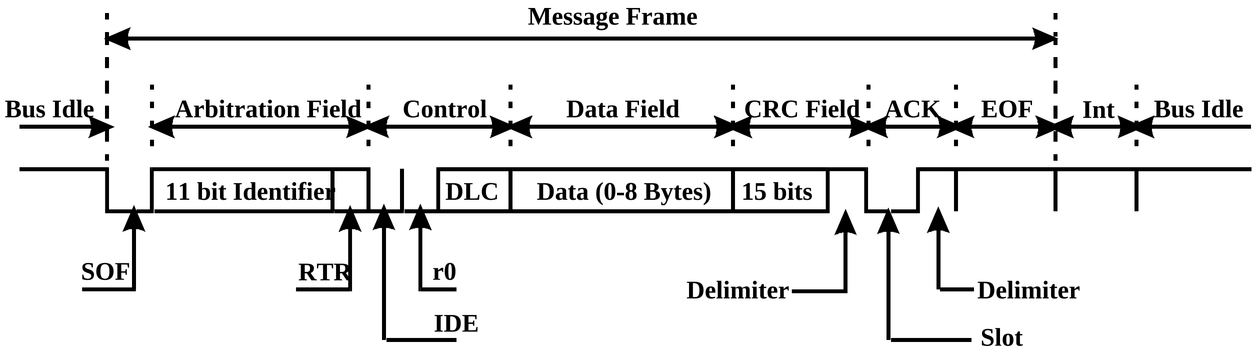

Figure 1: *CAN message format* [web CAN 98]

Below the message frame level, CAN performs bit stuffing. If the transmitter logic detects five consecutive bits of the same level, it will insert a sixth complementary bit into the transmitted bit stream to help the phase-locked bit timing loop maintain synchronization with the message bit stream. Thus if a CAN device receives five identical consecutive bits in the bit stream, the receiver logic will automatically delete the next incoming bit in a process called bit destuffing. Bit stuffing is performed on message frame bits from Start of Frame (SOF) through the CRC, and is done invisibly to the application. Therefore, while a CAN message may appear from Figure 1 to have a fixed number of bits given a fixed data field size, the number of bits actually transmitted on the physical network medium varies depending on data values and the associated need for stuff bits.

Even though CAN was specifically designed for safe operation in automobiles, previous studies have uncovered some design problems having to do with specific CAN features. Bit errors in the last two bits of the end-of-frame delimiter can cause inconsistent message delivery and generation of duplicate messages. For example, at a bit error rate (ber) of 10^{-4} and node failure rate of 10^{-3}, 2840 inconsistent message omissions will occur per hour and 3.94×10^{-6} inconsistent message duplicates will occur per hour. [Rufino 98]

A previous study was performed to analyze the possibility of undetected errors in CAN networks. [Unruh 90] That study classifies falsified messages that escape error detection into three types. Normal bit errors are errors that are due to the finite coverage of the CRC. Encoding related errors are bit errors that cause information bits to be interpreted as stuff bits and vice versa. Message length modifying errors either change the data length code of the message, generate end of frame marks, or change end of frame marks into stuffed sequences. That study concluded that encoding related errors were the most significant source of problems. An assumption made in their analysis was that bit errors in the identifier would cause application software to detect and reject messages based on improper data field lengths; however in our current work we have found that this assumption does not cover all likely failure scenarios and thus is unduly optimistic about the effectiveness of CAN error detection capabilities. In addition, their results were based purely on analysis and was not verified with simulation.

B. Electromagnetic Compatibility

The preceding studies have partially investigated what happens on a CAN network in the presence of bit errors. Of course the importance of these results depend on the ber that will actually be experienced. While it is generally acknowledged that bit error rates will be substantial, it seems to be difficult to find hard data on the subject of ElectroMagnetic Compatibility (EMC) inside vehicles. Nonetheless, at least two such studies have been published; one on a one-shot case study of a single vehicle, and another on general electromagnetic compatibility issues.

A one-shot case study of EMC for CAN demonstrated that for one vehicle, CAN seemed to remain robust in a harsh, real-world vehicle environment that had significant electromagnetic activity. In particular they state that when errors occur, the bus was able to recover rapidly so that there was no loss or significant delay of data. [McLaughlin 93] In addition, even though some of the systems that were controlled by CAN distributed information had failed, the information on the CAN bus was error free. Unfortunately, the amount of data that could be collected was limited by the realities of constrained resources and time available for testing. Thus, this report should be considered an indication that CAN will *typically* operate in a harsh EMC environment rather than a statement about any absence of low-probability events.

It seems likely that electric vehicles will provide a platform in which networks such as CAN be used in safety-critical control systems. However, the large drive motors in electric vehicles produce large amounts of electrical noise, making EMC problems an even greater concern than on current vehicles. For example, an EMC study of an electric vehicle revealed voltage spike levels up to 20V on the SAE J1850 serial communications bus (a low-speed bus that is otherwise generally CAN-like in nature). To reduce the spike levels down to 0.4V or less, EMC control features had to be installed, increasing vehicle cost. [Gaul 97] While adding cost is acceptable in a prototype, the harsh reality of the automotive industry is that even pennies count. Thus, in a production vehicle design it is almost a certainty that only enough shielding and similar features would be added to make the vehicle adequately operational, and in general this would not be enough to completely eliminate EMC problems.

From the previous research results, it can be seen that the CAN protocol is not perfect, but would appear to be reasonably robust to moderate bit error rates based on some simplifying assumptions that are not made in the analysis presented in this paper. There seems to be only limited data characterizing the bit error rate in current vehicles. However, it is clear that as electric vehicles (and, presumably hybrid combustion/electric vehicles) become more prevalent, electrical noise problems are likely to lead to significant bit error rates.

The results presented below describe a mechanism for failure rates worse than those discovered previously. (Unruh et al. report a subset of these failures, but did not discover the entire scope of the problem.) Given industry practice of studying network noise problems on a prototype basis as exemplified by the EMC studies above, it is important to have a tool that can project bit error rate to predict safety-critical failures without the need for full-scale fleet failure data. The results below provide such a predictive technique for multi-bit CAN errors using simulated fault injection.

III. Bit Encoding Undermines Error Detection Code Effectiveness

The CAN specification [Bosch CAN 91] states that the CRC field in each message will detect all burst errors up to 15 bits, and all errors with 5 or fewer disturbed bits. That specification also states that other multi-bit errors (with at least 6 disturbed bits or burst errors of more than 15 bits) will slip through undetected with a probability of 3×10^{-5}. These specifications seem adequate in environments with moderately infrequent individual bit error rates (because having a large number of bit errors quickly becomes improbable for any given message) and burst errors that are not longer than a few bits. And, in fact, if CAN simply used a CRC with no bit stuffing the specifications would fully be met by the CAN implementation.

Unfortunately, the CAN implementation does not meet the CAN specification. In particular, multi-bit errors may cause a cascading effect in which stuffing bits cause effective shifting of data patterns. This shifting leads to large numbers of bit errors being fed to the CRC. Thus, as few as two bit errors in the transmitted bit stream can appear as more than 6 disturbed bits to the CRC, causing false acceptance of a corrupted message.

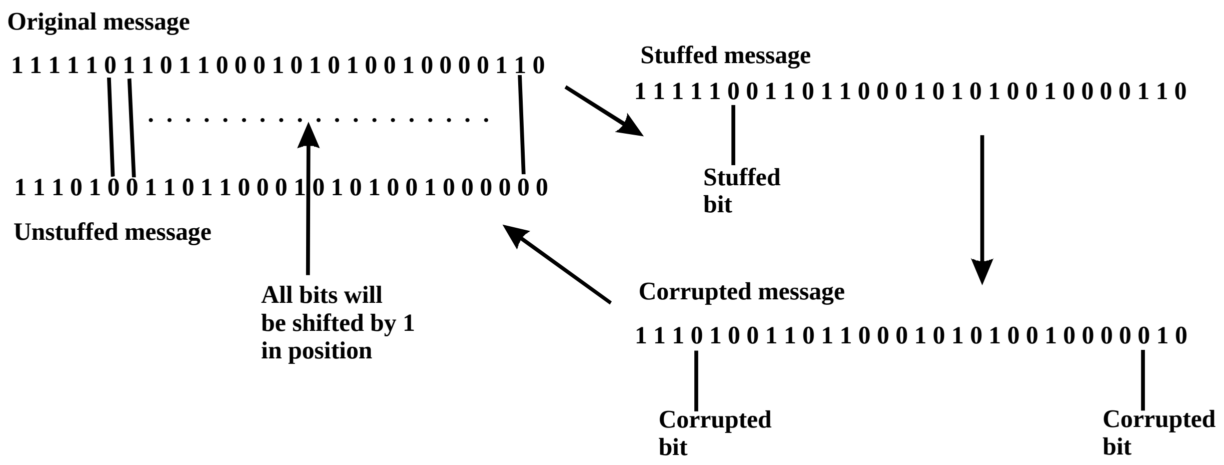

Figure 2: *Example of bit stuffing problem*

Figure 2 shows an example of the problem occurring in a message that has suffered two bit errors (2 flipped bit values). In this example a bit error breaks up a sequence of five ones in the data stream. Because of this, the following stuff bit is improperly interpreted as a data bit (a zero). For a single bit error this would eventually be caught as an illegal message length. However, in this example a second bit error occurs which creates a run of five zeros. This causes what was originally a data bit (the 1 at the end of the five zeros) to be interpreted as a stuff bit, subtracting a bit from the effective message length and therefore balancing the length problem created by the first bit error. Similarly, an apparent stuff bit might be created early in the message and a genuine stuff bit deleted later in the message (and, also, there is no reason that a message with more than two bit errors cannot suffer a similar problem).

The effect of the creation and deletion of balancing stuff bits in this failure mode goes beyond simply corrupting two data bits. In fact, this failure mechanism magnifies a pair of bit errors into creating a sequence of data and stuff bits that are shifted one bit position earlier or later in the data bit stream. This causes any place in the shifted bit pattern that transitions from a zero to one or a one to zero to experience an effective bit error. It should be noted that in general the result is not a burst error, because in places where the original data stream has multiple zeros or multiple ones in a row, effective bit errors only occur at the beginning and end of the shifted bit stream. Thus, the result of this failure mode is the random injection of a cluster of essentially independent bit errors in a region of the message.

IV. Experimental Methodology

An analytic approach to determining how often this problem would occur turned out to be extremely difficult. This is due to the variable length of a CAN message frame and the complex nature of a CAN frame when taking bit stuffing into consideration. Therefore, a simulation was developed to experimentally determine the likely occurrence of the problem. (Partial analytic results are described in the results and analysis section of the paper as a way to double-check simulation results.)

A program was developed that simulates sending CAN messages which are corrupted during network transmission (*i.e.*, corrupted at the physical message level including stuff bits). Two kinds of corrupted messages were simulated: randomly flipped bits and burst errors (where a burst error is defined as an injected stream of dominant or recessive bits). For comparative purposes, simulations can also be run with bit stuffing turned off (although a real CAN network could not be operated this way in practice). The simulator works by generating a message with random ID and data fields. A CRC is computed using the standard CAN CRC algorithm, and bit stuffing is then performed to create a transmitted bit stream. The message is then corrupted with an appropriate number and type of bit errors. The bit errors can be either randomly flipped bits or burst errors. For randomly flipped bits, the simulation user chooses the number of bit flips, and the simulator randomly generates the unique positions of the bit flips in the CAN message. For burst errors, the simulation user chooses the length of the burst error and the simulator randomly generates the starting position of the burst error.

After the message is corrupted, the stuffed message goes through a destuffing process. A number of different scenarios can happen during the destuffing process. There are certain bits in a CAN message that define the message format and length. If any of these bits are flipped, the message format and/or length will be changed. The first bit in a CAN message, the SOF, must be a logic 0. If this bit is corrupted, the CAN receiver will continue to look for a logic 0 until it is found. This will alter the values of all future fields in the message. The RTR bit is used to differentiate between a transmitted data frame and a request for data from a remote frame. If the RTR bit is 1, then the receiver will accept a regular data frame. If the RTR bit is flipped to 0, the receiver will accept a remote frame and look for the request for data from a remote node. This means that the message received will be interpreted as not containing any data bytes. The IDE bit identifies whether the message is standard format or extended format. If the IDE bit is 0, the message is standard format. If the IDE bit is flipped to 1, the received message will be interpreted as an extended format message. The DLC field defines the length of the data field in the message. The corruption of any bits in the DLC field will alter the message length and the receiver interpretation of all fields after the data field.

Any of these potential bit changes makes the receiver interpret a very different message from the one that was originally sent by the transmitter. The receiving algorithm checks for illegal stuffing patterns (*i.e.*, six or more zeros or ones in a row) and other message format violations (called form errors in CAN; requiring correctly formatted bit values in the end-of-frame, INT, ACK and CRC delimiter bits). In the majority of the cases, the corrupted message will be dropped due to either stuff or form errors. But in a small number of cases, the message may still pass both the stuff and form error checks.

If a stuff error or form error occurs, the error is considered successfully detected. If no error is detected at that stage, a new CRC is calculated from the corrupted unstuffed message. If that new CRC does not exactly match the information recovered from the CRC field of the corrupted message, then the corruption is considered to be successfully detected by the CRC check. If the computed and transmitted

CRC are the same, then the corrupted message has not been detected by the CAN protocol, and a failure is declared to have occurred in the form of a corrupted message being accepted as valid information. All original messages in the simulations used 8-byte data fields and 11-bit ID fields without significant loss of generality.

The simulation results presented here are more detailed than previous analyses because they take into account not only the effects of bit errors on the message ID field making failure rates higher, but also the effects of bit errors on fixed frame fields and stuff errors serendipitously reducing susceptibility to multi-bit errors. All simulations were run for extended periods of time to ensure representative results (at least 100 failures to detect corrupted messages for all runs, and more than 1000 such failures in selected instances to check accuracy). A run to detect 1000 failures consumed approximately 80 hours of CPU time on a 600 MHz Alphaserver.

V. Comparison of Simulation Results to Previous Analytical Results

A previous study analyzed the probability for errors to be undetectable at receivers (residual error probability) assuming a two-state symmetric binary channel model for the physical transmission medium. This model is used because there has been no tractable models that have been published for dominant/ recessive channels. Therefore, the channel is approximated as a binary symmetric channel where the probability for a bit to change from dominant to recessive is the same as the probability for a change from recessive to dominant. The temporal sequence of bit errors is described by a model containing good and bad states which allows for a decomposition of the residual error probability into contributions from the good and bad phases. The distribution of bit error occurrences at different stations is also modeled. [Charzinski 94] Using purely theoretical analysis based on these models, the author presents the contributions of different error cases to the residual error probability for a standard CAN frame with 8 data bytes per frame in a system with 10 stations. Specifically, the error cases are cases where the corrupted bits can only be contained in the data and CRC field, the ID field, the DLC field, the SOF field, or the IDE field. Due to the complex nature of CAN, assumptions and simplifications must be made in determining the residual error probability using only analysis.

An attempt was made to recreate the graphs presented in the paper using the CAN simulation. However, several obstacles occurred along the way. In the paper, the only equation given was for the residual error probability when only the data and CRC field bits can be corrupted. But for this case, the pictured graph in the paper does not match the graph generated from its equation given in the paper. This could be because the equation in the paper is an approximation of a more exact equation from a previous paper. The pictured graph does not deviate considerably from the graph generated from the approximated equation, so the approximation could account for this difference. Figure 3 shows a graph of the residual error probability versus bit error rates. The curves in Figure 3 represent a duplication of the curve presented in the original paper (legend label drawn), the curve generated from the approximated equation (legend label equation), and the curve generated from the CAN simulation (legend label simulation). As it is seen clearly from the graph, the simulated curve does not match either of the analytical curves.

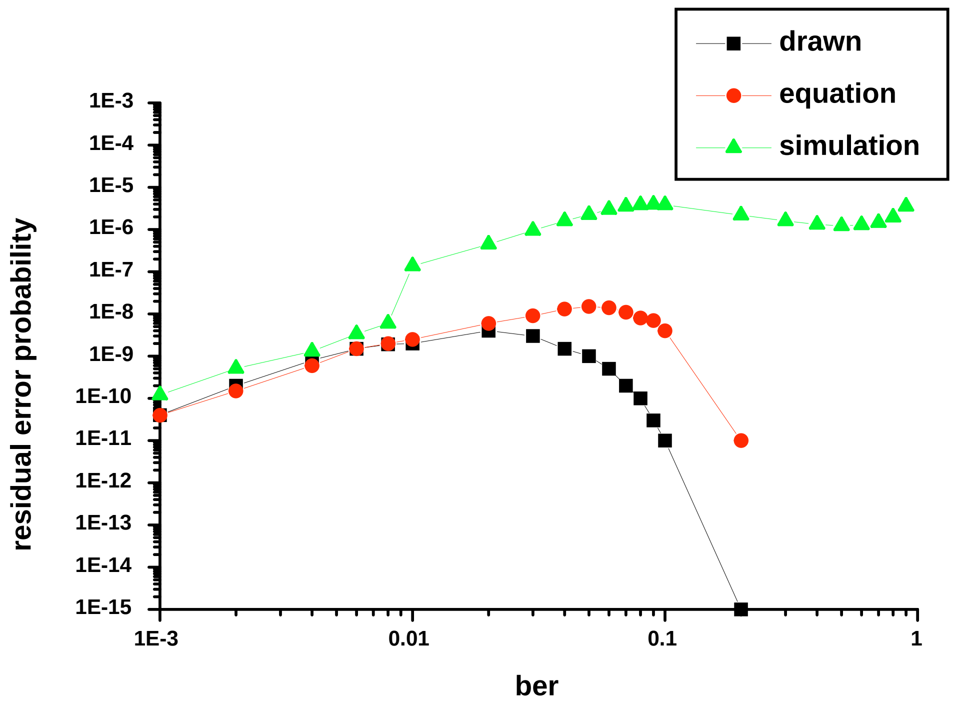

Figure 3: *Residual error probability vs. ber: simulation results compared to [Charzinski 94] graph and equation.*

An investigation was made into why the simulated curve does not match the analytical curves, and whether the simulated curve makes logical sense. The difference could be due to several assumptions that were made in the paper to simplify the analysis but could also cause possible errors in the results. One assumption made was that the frame length was calculated neglecting the number of stuff bits in the message frame. Neglecting this factor does make the analysis simpler as the number of stuff bits in a message changes based on the bits values in the frame. However, as anywhere from one up to six or seven stuff bits can be inserted into a message, this can cause the frame length to increase by up to 10%. Another assumption made was that for an error to be considered a residual error in the network, the error must be undetectable by every station in the system. A model was given that gives the probability that this scenario happens, but not enough information was available in the paper to recreate this situation. The paper also states that due to the bit stuffing effects, the CRC can safely detect two bit errors per message. However, this is clearly not true because of the ripple effect discussed previously. Finally, the last assumption made is that for low bit error probabilities, the residual error probability is dominated by cases where there are only two bit errors per message frame. But what defines a "low" bit error probability was never specified in the paper.

In the simulated curve, the graph peaks at a ber of 0.09 then decreases down to a low point at a ber of 0.5. The curve then rises again until a ber of 0.9. For ber lower than 0.01, the simulated and analytical lines, although different in values, are parallel. However, for ber’s greater than 0.01, the simulated and analytical line deviations are much larger. This could be due to the fact that for ber’s greater than 0.01, there is often more than one bit error per frame, causing the "low bit error probability" assumption to be invalid. But for ber’s lower than 0.01, the "low bit error probability" assumption is valid, so the curves are parallel. The difference between the two parallel lines could be due to the assumptions as stated in the previous paragraph.

To gain further insight into the shape of the simulated graph, several other graphs were generated. Figure 4 shows the number of messages dropped from stuff errors, form errors, and the sum of stuff and form errors for ber’s from 0.01 to 0.9 where one billion messages were simulated per ber value. The number of messages dropped from both stuff and form errors peaks at a ber of 0.5. This peak corresponds to the dip in the graph for the residual error probability. This dip makes sense because where there is the largest number of messages dropped from stuff and form errors, there will be the lowest number of messages susceptible to not being caught by the CRC.

Figure 5 shows a graph of the number of messages undetected when sending one billion messages for each ber value simulated. This graph also shows the calculated number of corrupted messages undetected.

The calculated numbers are generated by taking one billion and subtracting the number of corrupted messages dropped by stuff and form errors, and then multiplying by 3 x 10^{-5}. The multiplication by 3 x 10^{-5} is the fraction of messages undetected by the CRC as the number of corrupted bits increases. Comparing the simulated and calculated curves, the estimation works well for ber’s greater than 0.1 and fails for ber’s less than 0.1. This is because the 3 x 10^{-5} estimation works well for large number of bit errors and fails for small number of bit errors. This estimation can therefore be used to verify that the values for ber’s greater than 0.1 are accurate, which is what is seen in Figure 5.

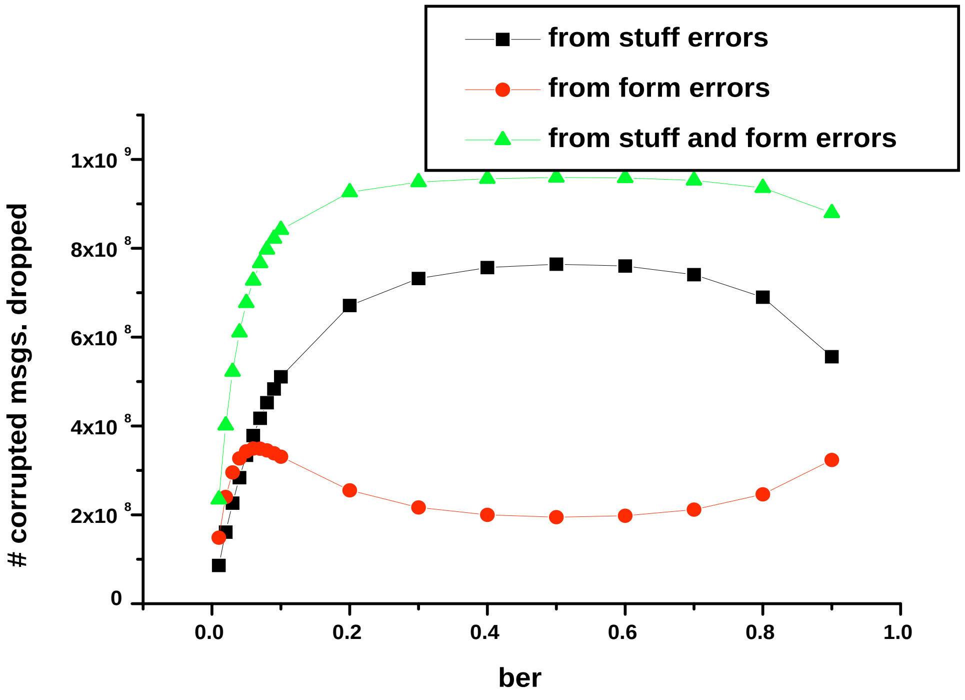

Figure 4: *Simulation: number of messages dropped due to stuff and form errors (1 billion corrupted messages sent per ber)*

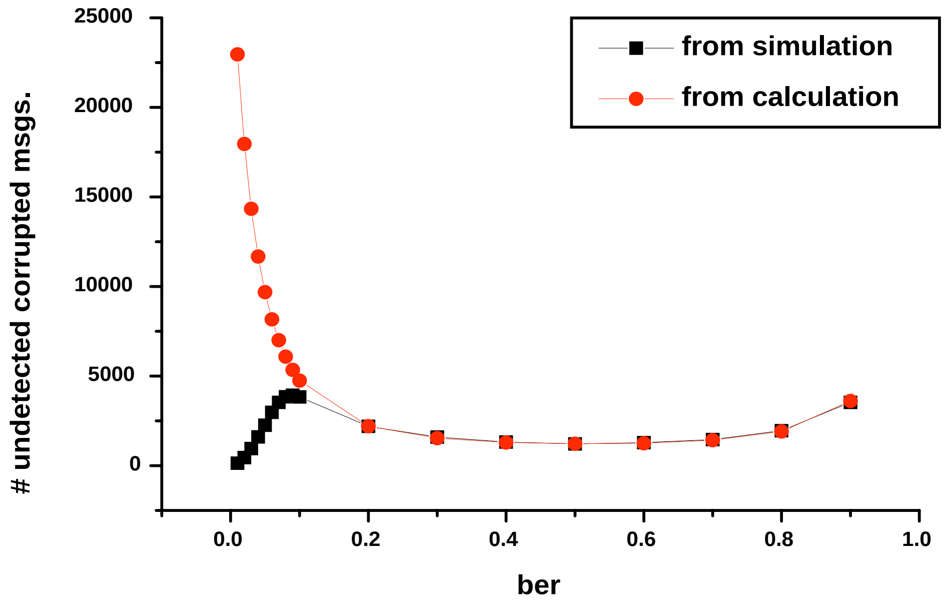

Figure 5: *Simulation: undetected messages vs. ber (1 billion corrupted messages sent per ber)*

VI. Simulation Results and Analysis

We ran several sets of simulation experiments to understand CAN multi-bit error detection failure rates. These experiments first quantify the magnitude of the problem of messages slipping past the CRC, then explore the effects of bit stuffing and CRC in isolation as they contribute to the overall CAN error detection capabilities.

A. Results For Randomly Corrupted Bits

The CAN specification states that all messages with fewer than 6 corrupted bits will always be detected and messages with 6 or more corrupted bits will be detected with a given probability, but this did not correspond with simulation results. Figure 6 show the ratio of undetected corrupted messages found when simulating CAN message transmissions having various numbers of corrupted bits. For each data point in Figure 6, 250 million message transmissions were simulated, each with a given number of randomly selected bits being flipped to an opposite value from the correct value (this included stuffing bits and all fields within a CAN message being subject to corruption). For example, in Figure 6 the data point for 4 bits flipped indicate the probability of a message being accepted as valid given that 4 distinct and independent bit errors have occurred in that message.

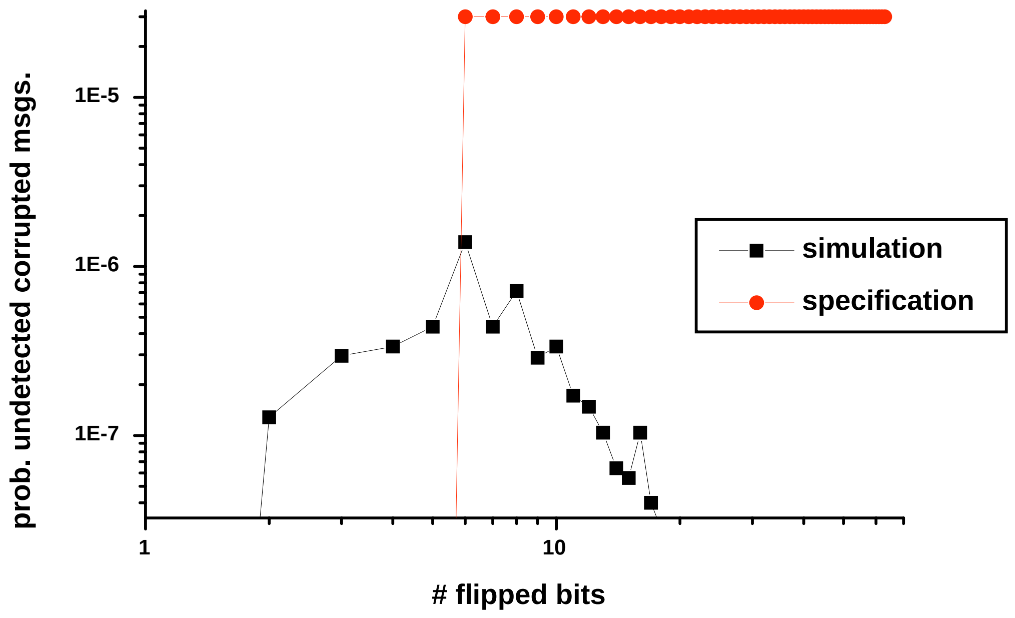

Figure 6: *Simulated performance of CAN compared to CAN specifications*

A.1. Full CAN Error Checking

For messages with fewer than 6 corrupted bits, it is evident from the data in Figure 6 that CAN does not detect all corrupted messages as specified. For example, with only 2 flipped bits, 32 of 250 million corrupted messages would have been undetected by CAN, yielding a detection failure rate of 1.25 x 10^{-7} rather than the specified failure rate of zero. This is due to the interactions of bit stuffing with CRC error detection as discussed above. The number of messages that slip by the CRC decreases with an increasing number of corrupted bits (rather than increasing as one might expect, and as is implied by the CAN specification) because of the increased probability that the message will cause a stuff error or form error. For large numbers of flipped bits the ratio of corrupted messages that go undetected actually drops to almost zero because a very large percentage of the corrupted messages are detected by the stuff and form error detection. These findings differ significantly from the CAN specification also shown in Figure 6 that messages with fewer than 6 corrupted bits have a zero failure rate and that messages with higher levels of corrupted bits have a higher failure rate of 3 x 10^{-5}.

A.2. CRC-only Error Checking

Even in light of the double-bit error failure mode discussed above, it was surprising that the CAN implementation is both worse for small numbers of bit errors and better for large numbers of bit errors than stated by the CAN specification. Thus, separate simulations were run to understand the relative contributions of CRC error detection, form error detection, and bit destuffing error detection. The simulations were run under the same conditions as Figure 6, but with various error detection mechanisms disabled.

When a CAN message is corrupted, there are several checks that can detect the corruption and drop the message. The maximum length of data for a CAN message is 8 bytes, so any corruption that generates a message with greater than 8 bytes of data will be dropped. The message can also be caught by either a stuff error or form error check. Any message not caught by these checks will fall through to the CRC check. The higher line in Figure 7 shows the probability of a message undetected using the calculated method as follows. The total number of messages dropped by data length error, stuff error, and form error were subtracted from 250 million. This result is then multiplied by 3 x 10^{-5} to yield the line shown in Figure 7. The fraction 3 x 10^{-5} comes from the average probability that a message will not be detected by the CRC. As seen from Figure 7, the calculated values and simulated values converges as the number of flipped bits increases. This is because the average CRC probability is only valid for large number of flipped bits and not for small number of flipped bits (as shown in Figure 6).

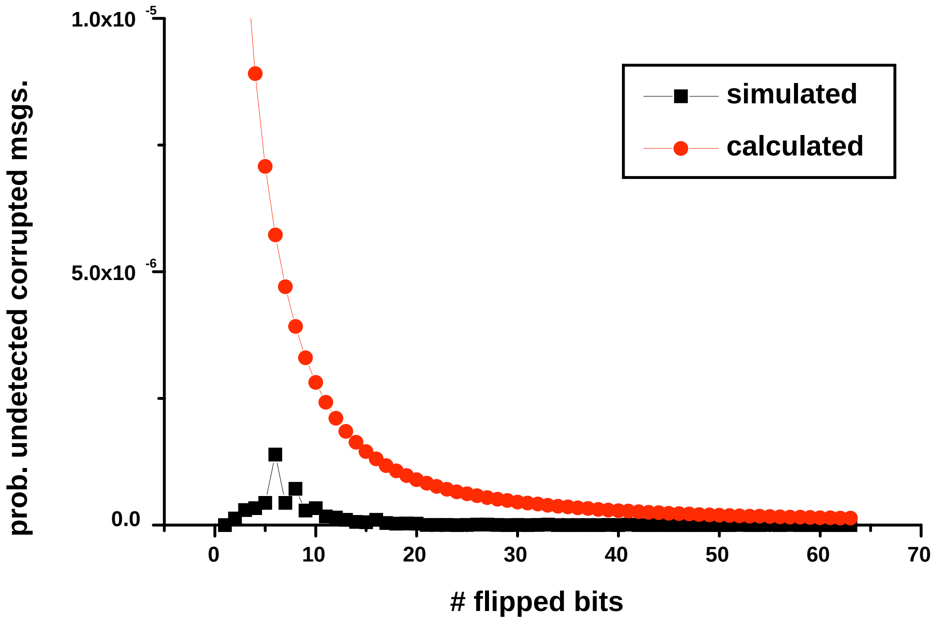

Figure 7: *Comparison of simulated and calculated probability of undetected messages*

Figure 8 shows the probability of a message passing undetected when only the CRC check is implemented for 2 cases. The first case illustrates results when running the CAN simulation with unstuffed messages and only flipping bits before the CRC field, but certain CAN features are not implemented. These features not implemented are: the SOF bit is always correct, the message cannot be interpreted as being extended format, the message can only be interpreted as having 8 data bytes, and the message cannot be interpreted as a remote frame. The second case illustrates results when running the CAN simulation with unstuffed messages and only flipping bits before the CRC field, with all other CAN features implemented. For the first case, any configuration with an odd number of bit errors is always detected (as one would expect from a well-constructed CRC having the term (x+1) as a polynomial factor). With CRC-only, even numbers of bit errors slip past at a rate of approximately 6.0 x 10^{-5}, which is twice the specified CAN rate of 3 x 10^{-5}. Thus it would appear that the CAN specification for 6 or more bit errors is an average for CRC-only detection that does not take into account the CRC's ability to detect all odd numbers of bit errors as well as does not take into account form errors and bit stuffing effects.

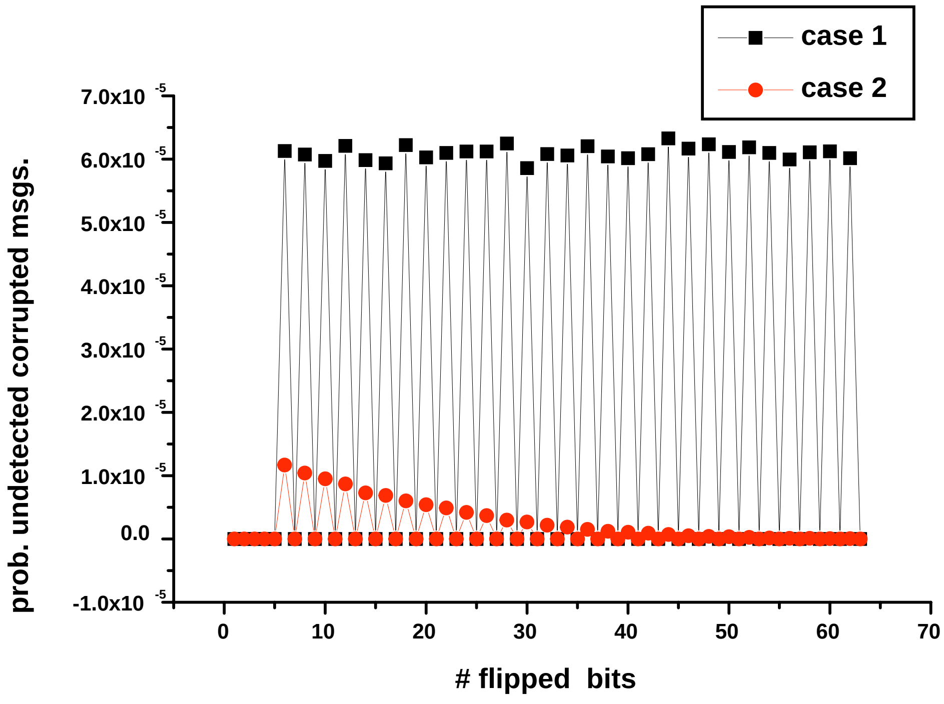

Figure 8: *Probability of undetected corrupted messages under the assumptions of case 1 and case 2. Case 1: Sending unstuffed messages and only flipping bits before the CRC field, but certain CAN features are not implemented. Unimplemented features are: the SOF bit is always correct, the message cannot be interpreted as being extended format, the message can only be interpreted as having 8 data bytes, and the message cannot be interpreted as a remote frame. Case 2: Sending unstuffed messages and only flipping bits before the CRC field, with all other CAN features implemented.*

In the second case, the numbers still do not match the CAN specification. However, this line does give us insights into the performance of CAN. This line shows that all corrupted messages with less than 6 bit errors per message are detected by CAN, confirming that bit stuffing is working to undermine CRC effectiveness (and behaving in accordance with the CAN specification). For number of flipped bits greater than 5, practically all messages with an odd number of flipped bits are detected and messages with an even number of flipped bits are not necessarily detected. This line can be used to explain the performance of the full CAN error checking case. There is a big jump in the probability of undetected messages from 5 to 6 flipped bits due to the decreased effectiveness of the CRC check for cases with greater than 5 flipped bits. The spikes at even numbers of flipped bits is in accordance with the CRC effectively catching an even number of bit errors. Finally, for cases with an even number of flipped bits, the probability of undetected messages decreases for increasing bit flips because more messages are being caught by format error checks and form error checks.

A.3. Verification of Results

Beyond digging for reasonable explanations for behaviors observed in Figures 6, 7, and 8, we were able to perform limited verification of independent bit error results. The verification was performed for the double-bit error case because this would be the most probable multi-bit error scenario in practice. We ran a long simulation of CRC and bit stuffing to make sure we were getting representative results in the preceding set of simulations that varied the number of corrupted bits.

Software testing was performed on the simulation code. Each subroutine was individually tested with test cases to cover the range of possible input values. Additionally, output values of the bit stuffing subroutine were used as input values of the bit unstuffing subroutine to verify that the same original result was obtained. The CRC function was verified by comparing 2 different algorithms obtained from different sources. Other subroutines were verified with hand calculated results. In addition, a code walkthrough was performed on all key portions of the simulation. Verification by physical hardware fault injection was not done due to lack of a budget for a hardware injection tool (although hand comparison of stuffed message formats and CRC values was performed using a logic analyzer for several messages).

Analysis was performed as a "sanity check." To make the analytic verification tractable, we used a set of simplified approaches to build suggestive evidence that the results were correct. The verification was also performed taking into consideration the 4 assumptions in the previous section which are, once again, that the SOF bit is always correct, the CAN message is only in standard format, the CAN message always has 8 bytes of data, and the CAN message cannot be interpreted as a remote frame by the receiver. Using these assumptions, in an extended simulation of 10 billion double-bit error messages, 5010 messages were undetected, yielding the fraction of undetected messages to be 5.00 x 10^{-7}. As discussed previously, when a message has 2 flipped bits, certain bit patterns cause a stuffed bit to look like a data bit and vice versa. The problem is to determine for how many cases this would happen. If we can calculate this value, then we can compute the probability of a corrupted message slipping past the CRC. Several simplifications were made to aid in the analysis. First, we are assuming that only the 83 data and message ID bits before the CRC can be corrupted (taking the CRC into consideration would significantly increase the complexity of the problem). Second, 83 bits was used as a length value without including message lengthening that would take place with bit stuffing without significant loss of generality for the calculation.

In order for a stuffed bit to look like a data bit and vice versa, the patterns in Figure 9 need to occur at 2 locations in the original message. Each set of pattern consists of 6 bits so we group them together and designate them Group A and Group B. For a failure to occur, a pattern from Group A must be followed by a pattern from Group B. The first step is to determine the total number of possible locations for A and B. For example, keeping a window size between groups A and B in Figure 9 constant at 2 bits, there are 4 x 5 x 69 different locations. (where 83 - 12 - 2 = 69). Keeping the window size constant at 3, there are 4 x 5 x 68 different locations. A pattern develops until a window size of 71 (at which point bits from Group B overflow the message length), where there are 4 x 5 x 1 different locations.

Next we need to determine how many different corruption patterns might be undetected by the CRC. As stated previously, with two corrupted bits in a particular location, intervening bits appear to be shifted by one bit position, and bit value transitions will produce apparently corrupted bits. So the key is to determine the total number of transitions or bit change for each window size. Summing for each window size the total number of transitions greater than 2 and knowing the total number of locations from above gives enough information to determine the probability of messages that may be corrupted compared to a total number possible messages of 2^{83}. The following equations yields this value.

Let p = probability of message undetected for 2 corrupted bits

Let i = window size

Let t = number of patterns for window size i which has greater than 1 transition

Simplified Approximation:

$$p = \frac{\left(\sum_{i=1}^{69} 4 \cdot 5 \cdot i \cdot t \right)}{(2^{83} \cdot 2^{15})}$$

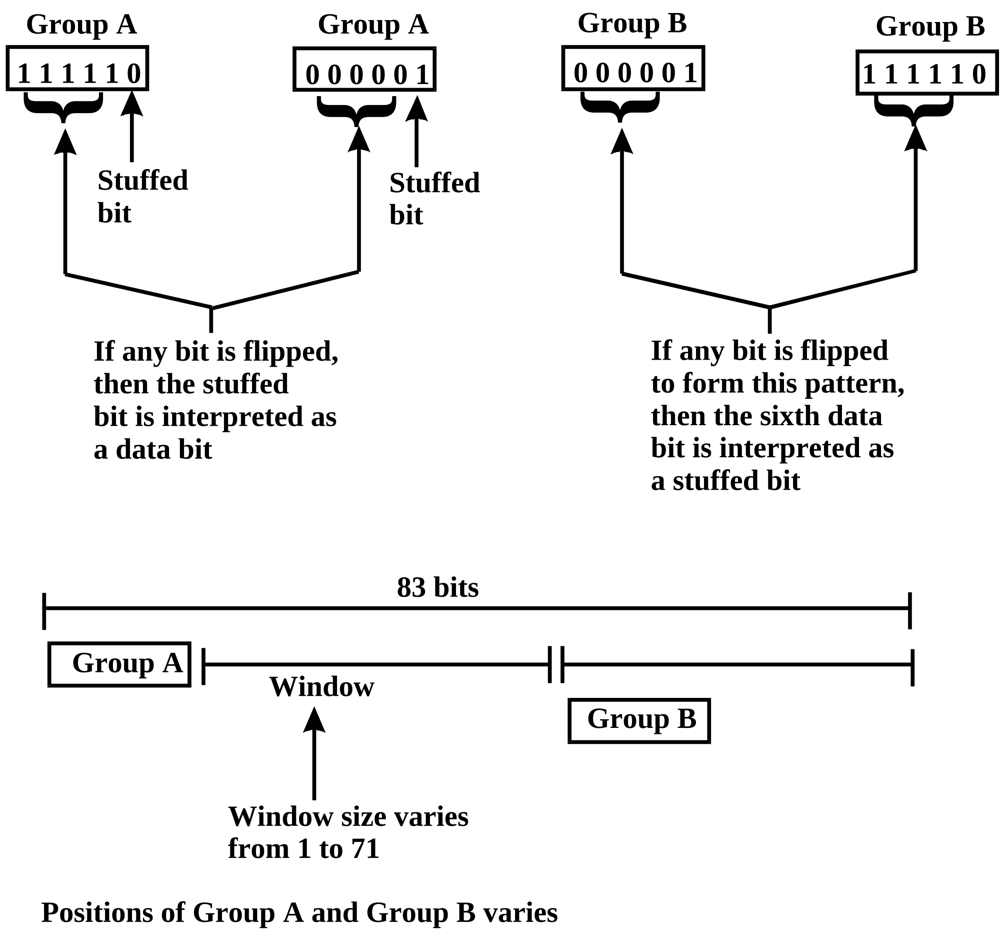
Group A
1 1 1 1 1 0
Stuffed bit
Group A
0 0 0 0 0 1
Stuffed bit
Group B
0 0 0 0 0 1
Group B
1 1 1 1 1 0
If any bit is flipped, then the stuffed bit is interpreted as a data bit
If any bit is flipped to form this pattern, then the sixth data bit is interpreted as a stuffed bit
83 bits
Group A
Window
Group B
Window size varies from 1 to 71
Positions of Group A and Group B varies

Figure 9: *Bit stuffing problem*

Since the analytical probability does not exactly match the simulated probability, we ran a simplified simulation result to determine if the values would converge. We simulated 6 billion stuffed messages with 2 corrupted bits picked from the first 83 bits after stuffing. However, to our surprise the result was further from the analytic result than the original simulation results. A resulting insight was that when any of the bits in the message can be flipped, some of the corrupted messages will be caught by the form error and stuffing error checks. But when we are only able to corrupt any of the first 83 bits, then the probability of corrupted messages being undetected increases because none of them will be caught by the other checks. Other analytic attempts also failed to get us any closer to our simulated results because of the complexity of interactions among form error checking, stuffing error checking, and CRC error checking. In the end we seemed to have re-learned a lesson that characterizing the results of undetected errors for a CRC usually cannot be determined exactly, but can only be bounded from below [Wells 99]. Eventually taking this hint, we can compute a lower bound on the probability of undetected errors as 3.0 x 10^{-7} from equation 1, which in fact is below the simulated probability of 5.0 x 10^{-7}.

B. Burst Errors

Further simulations were performed to measure behavior in the presence of burst errors, which can be caused in real systems from components such as electric motors that have electromagnetic fields with time constants as long as several bit times. Similar to our work with independent bit errors, we performed simulations and attempted analysis.

Surprisingly, the first problem encountered was that there does not appear to be an agreement as to the definition of exactly what constitutes a burst error! The CAN specification simply uses the term without definition. Furthermore, an attempt to pin down the definition for simulation purposes revealed that the definition of burst error differs between mathematical texts and engineering practice. The engineering definition is that a burst error appears as a series of consecutive ones or zeros injected into the message (corresponding to a long voltage spike induced in a baseband communication network, or, more loosely, a transient stuck-at condition on the network). This definition is supported by one reference that states that a burst error occurs when many bits are damaged due to a brief power surge or static electricity [Shay 99] and our industry experience. However, the mathematical definition defines burst errors as a number of consecutive bits that are flipped. This seems to be supported by another reference that states that burst errors are characterized by a pattern of flipped bits having an initial 1, a mixture of 0s and 1s, and a final 1, with all other bits being 0. [Tanenbaum 96] While certainly a series of flipped bits can occur from a noise source, it seems improbable that a noise phenomenon would get lucky enough to flip a set of sequential bits in just the right way to complement their somewhat arbitrary data pattern. No doubt the mathematical definition makes analysis simpler, but it would stand to reason that a stuck-at model more closely approximates what happens in the real world. (For truly random noise sources instead of voltage spikes, one would expect to actually get a cluster of concentrated but separated bit errors depending on whether the noise source gets lucky or unlucky in forcing a bit to its original value or not but that debate is beyond the scope of this paper.)

According to the CAN specifications, burst errors up to a length of 15 are detected in any message (using the CRC given in the specifications). This turns out to be a true statement, but possibly not for reasons one might expect. For the mathematical definition of burst errors and a CRC-only system, the specified behavior is what one would expect and was confirmed via simulation (in fact a greater than expected/ specified number of 16-bit and greater burst errors were caught because of form errors).

Using the engineering definition of burst errors, the CRC alone would not catch a significant number of burst errors below the length of 15. This is because a stuck-at burst error model is very likely to force some subset of the burst error field to correct data values, changing the effect of a burst error into clusters of some bits flipped and other bits not flipped. When the total number of bits flipped in this manner is more than 5, the CRC would be partially ineffective. However (fortunately for CAN), destuffing error detection is guaranteed to detect all burst errors of this type of length 6 bits or greater. Thus, for the engineering definition of burst errors, CAN detects all burst errors, not just burst errors of length 15 or less.

C. Implications of the Results

We will now examine the CAN failure mode in the context of a fleet of automobiles in the real world. As stated earlier in the paper, the probability that a message with 2 flipped bits will go undetected is 1.29 x

10^{-7}. Although this probability is small, the number will be magnified by the size of the vehicle fleet. For example, in round numbers there are probably approximately 200 million ground vehicles in the U.S. and 2.469 trillion vehicle miles [Koopman 98] traveled per year at an average speed of approximately 30 miles per hour. Assume that there is one safety-critical CAN bus in each vehicle operating at 50% utilization, or 500 kbps total utilization, and that each message is 117 bits in length including bit stuffing. In one year, this works out to an accumulated operating period of 2.96 x 10^{14} seconds traveled. At 500 kbps this is a total of 1.48 x 10^{20} bits per year or 1.27 x 10^{18} messages per year subject to corruption. Based on these assumptions, Table 1 and Figure 10 shows the frequency at which undetected corrupted messages are sent at different ber values. If it were desired that undetected corrupted messages were to occur less frequently than once per year for the vehicle fleet, the ber would need to be better than 1 x 10^{-6}. To obtain the equivalent incident rate as an extremely improbable aerospace failure of once every 73 years, the ber would have to be below approximately 1 x 10^{-9}. However, it is known that typical ber for a copper cable is on the order of 1 x 10^{-6} or 1 x 10^{-7}. [Peterson 96] This number is generally only valid for a general-purpose computer-grade network, so the performance for a network in a noisy environment is likely to be considerably worse. Therefore, it seems likely from this example that the vulnerability in CAN could cause potential problems in a fleet of vehicles unless corrective measures are taken in the form of improved noise resistance or better error detection capabilities.

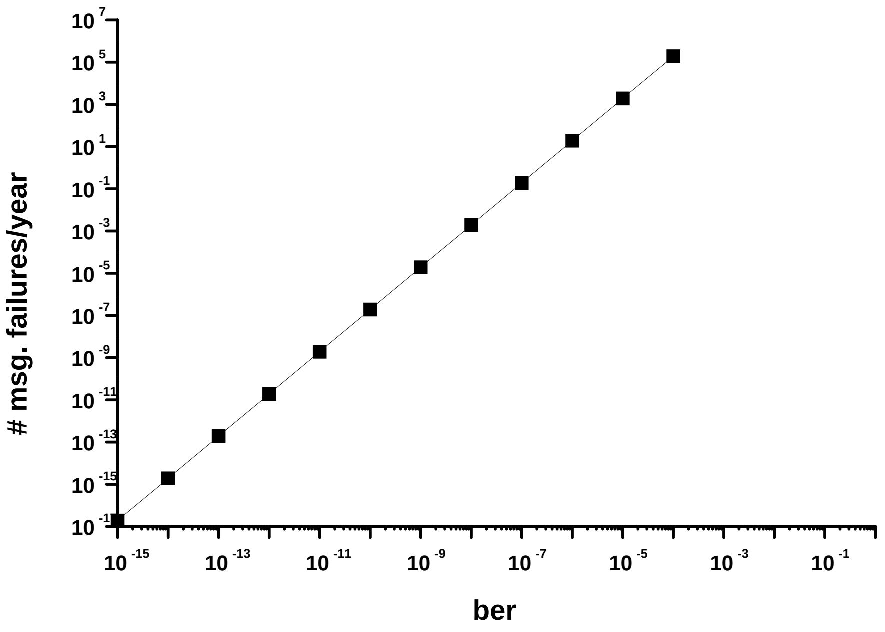

Figure 10: *# corrupted messages sent/year vs. ber for a fleet of 200 million vehicles traveling for 2.469 trillion miles per year at 30mph.*

ber	**# undetected corrupted messages**
1 x 10-4	21.8 msgs/hr
1 x 10-5	5.2 msgs/day
1 x 10-6	0.05 msgs/day

Table 1: *Frequency of undetected corrupted messages at varying bit error rates for a fleet of 200 million vehicles traveling for 2.469 trillion miles per year at 30mph.*

VII. Potential Solutions

There are several ways, both in software and hardware, to improve the error detection capability of a CAN system. Software is obviously easier and cheaper, and one very simple way to improve CAN is to perform additional error checks on top of the ones that are currently in CAN.

One way to improve CAN error detection is to use the last data byte or two in the data field to compute an additional CRC or checksum on the message. Using two bytes to compute an additional 16 bit CRC, all corrupted messages were detected by the CAN simulation. Unfortunately, this costs 25% of the data payload of a CAN message, and is likely to be too expensive for real applications. Figure 11 shows the results of implementing three different ways to improve CAN error detection using only one byte of the data field.

Figure 11 shows that adding an 8-bit CRC only results in a very slight decrease in corrupted messages slipping past both the 15-bit CRC and this new 8-bit CRC (in general one could say that slightly extending a weak technique already in use is not the most effective approach). However, performing an computationally less expensive checksum operation improves the error detection of messages slipping past the CAN CRC by 2 orders of magnitude. This performance improvement is probably due to the different mathematical basis of a checksum operation, and an apparently greater ability to detect clustered bit errors. A Logitudinal Redundancy Code (LRC a bit-wise XOR of the message) was also tried because it has a cheaper hardware implementation than an adder. The 8-bit LRC was better than an 8-bit CRC but not quite as good as an 8-bit checksum. Performing an additional check results in dramatic improvement if the bit error rate is moderate and a 1-byte loss in data payload is acceptable to the application. If the application requires an extremely robust network protocol that cannot tolerate any corrupted messages passing through, then a 16-bit additional error detection code will probably be required.

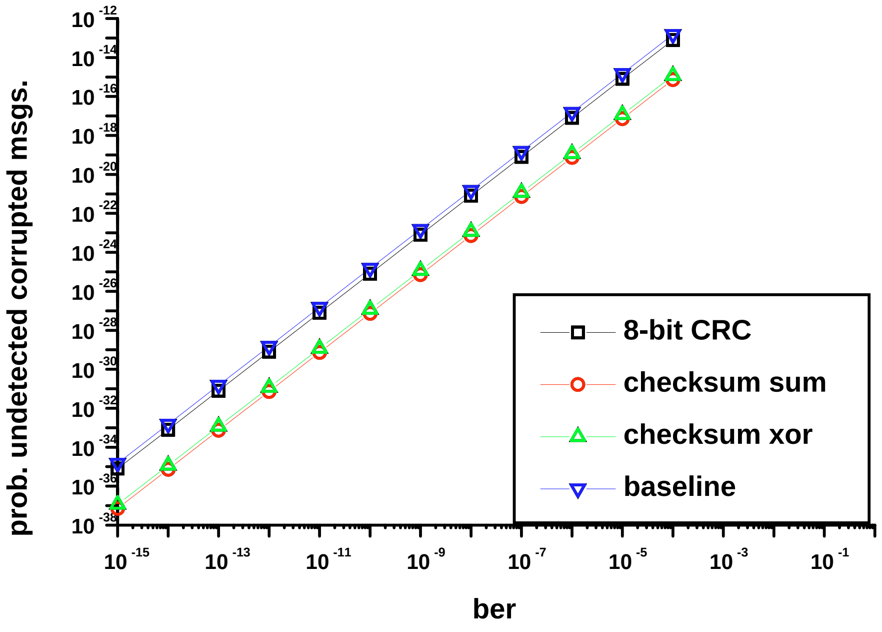

Figure 11: *Improvement of undetected error rate by putting an 8-bit additional error detection code in the message data field.*

In addition to a software solution, there is also a potential hardware solution. A modified version of the CAN protocol could compute the CRC *after* bit stuffing is performed instead of before bit stuffing. This would eliminate the possibility that a stuffed bit looked like a message bit and vice versa. This approach was simulated and resulted in no 2-bit corrupted messages passing the CRC, as one would expect. There is a minor complication in that the message format must be adjusted to ensure that the CRC field does not itself need bit stuffing. This solution would obviously not be compatible with current CAN hardware, but is a plausible candidate for some new, future CAN standard or enhanced operating mode for near-term CAN chips being used in proprietary applications.

VIII. Conclusions

Even though the Controller Area Network protocol is specifically designed for critical automotive applications, there is a vulnerability to undetected multi-bit data errors. While the probability of encountering these errors in prototype installations is low (so they will probably not be seen in the lab except under extremely noisy electrical conditions), they appear very likely to happen in full-size deployed fleets. Under a set of assumptions representative of a possible future national automotive fleet, it was found that bit error rates must be reduced to 10^{-8} or 10^{-9} to essentially eliminate the problem, which is significantly below the usual bit error rate of 10^{-6} or 10^{-7} for typical copper-based computer networks, and even further below the elevated bit error rates likely to be seen in a vehicular network. Fortunately, there are some near-term software fixes involving adding an extra check byte to the data message that may help in some situations. Further, a relatively straightforward change to the CAN protocol to compute the CRC after bit stuffing instead of before bit stuffing would completely eliminate the problem if adopted in some future incarnation of the CAN standard.

Beyond the specific simulation results presented here, there are some higher level issues that should be noted. One is that error detection cannot be designed layer-by-layer in a system or communication protocol in the absence of knowledge about other system layers. In this case bit stuffing undermined CRC effectiveness. In fact, this exact problem might manifest in other networks. For example the HDLC protocol also implements bit stuffing and uses a CRC. While HDLC is generally not used in safety-critical systems, it is certainly possible that it might someday be, or that some new protocol might be invented that also makes this design decision and is in fact used in safety-critical systems. No doubt there are other similar interactions in critical systems that have yet to be discovered because they do not occur frequently enough to be noticed, or are blamed on other causes.

With the current push to use off-the-shelf software and hardware, another high level issue is that even an apparently tried-and-true technology might have safety-critical problems that were not envisioned by the original designers and not found by subsequent users (or even researchers). The research reported herein was spurred on by a desire to completely understand the safety-critical implications of embedded networks in the context of automated vehicle operation. One unhappy conclusion is that now that we have found one problem and spent significant effort to understand and characterize it, we are daunted by the task of even thinking of where to look for problems next.

Acknowledgments

This work is sponsored by ONR contract N00014-96-1-0202, and by USDOT under Cooperative Agreement Number DTFH61-94-X-00001 as part of the National Automated Highway System Consortium (NAHSC).

References

[Charzinski 94] Joachim Charzinski, "Performance of the Error Detection Mechanisms in CAN," *Proceedings of the 1st International CAN Conference*, Mainz, Germany, September 1994, pp. 1.20-1.29.

[SAE CAN 90] "Controller Area Network (CAN), An In-Vehicle Serial Communication Protocol," *1998 SAE Handbook, Volume 2 Parts and Components*, SAE J1583 MAR90.

[web CAN 98] Controller Area Network, http://www.omegas.co.uk/CAN/, accessed October 19, 1998.

[Dilger 98] Elmar Dilger, Thomas Fuhrer, Bernd Muller, Stefan Poledna, "The X-By-Wire Concept: Time Triggered Information Exchange and Fail Silence Support by New System Services," SAE 98-PC124.

[Gaul 97] Harry W. Gaul, Tom Huettl, Chuck Powers, "Electromagnetic Compatibility of an Experimental Electric Vehicle," *EE Evaluation Engineering*, Vol. 36, No. 2, February 1997, pp. 114-121.

[Koopman 98] Philip Koopman, Eushiuan Tran, Geoff Hendrey, "Toward Middleware Fault Injection for Automotive Networks," Fast Abstract, *28th International Symposium on Fault-Tolerant Computing Systems*, Munich, Germany, June 1998.

[McLaughlin 93] R. T. McLaughlin, "EMC Susceptibility Testing of a CAN Car," SAE 932866.

[Peterson 96] Larry Peterson, Bruce Davie, *Computer Networks: A Systems Approach*, Morgan Kaufmann, San Francisco, 1996.

[Bosch CAN 91] Robert Bosch GmbH, "CAN Specification Version 2.0," September 1991.

[Rufino 98] Jose Rufino, Paulo Verissimo, Guilherme Arroz, Carlos Almeida, Luis Rodrigues, "Fault-Tolerant Broadcasts in CAN," *Proceedings of the 28th International Symposium on Fault-Tolerant Computing Systems*, Munich, Germany, June 1998, pp. 150-159.

[Shay 99] William Shay, *Understanding Data Communications & Networks*, Brooks/Cole Publishing Company, Pacific Grove, CA, 1999.

[Tanenbaum 96] Andrew S. Tanenbaum, *Computer Networks*, Prentice Hall, New Jersey, 1996.

[Unruh 90] Jan Unruh, Hans-Jorg Mathony, Karl-Heinz Kaiser, Robert Bosch GmbH, "Error Detection Analysis of Automotive Communication Protocols," SAE 900699.

[Wells 99] Richard B. Wells, *Applied Coding and Information Theory for Engineers*, Prentice Hall, New Jersey, 1999.