A Case Study on Using Probabilistic Verification to Find Failures in a Cooperative Driving Application

Shou-pon Lin
Department of Electrical Engineering
Columbia University
New York, USA
shouponlin@ee.columbia.edu

Nicholas F. Maxemchuk
Department of Electrical Engineering
Columbia University
New York, USA
nick@ee.columbia.edu

Abstract—**This paper introduces techniques that help proving the safety of cooperative driving applications. Most automotive applications are evaluated by means of simulations and test tracks, which cannot provide the degree of confidence that is demanded by the driving public. We introduce probability verification as an evaluation tool that provides greater degree of confidence over safety. This technique is applicable to systems with large state spaces that cannot be exhaustively verified by other model checking techniques.**

We present a protocol that assists drivers in merging between other vehicles. The protocol is built on a multiple stack architecture that partitions the system into smaller manageable components that can be tested and verified individually. A module is verified assuming that the modules that it depends on have been verified. The modules in different vehicles use synchronized clocks to reduce the number of states needed to be examined in the composite protocols that describe their interactions. We are able to show that system failures will occur with a sufficiently small probability. Since probabilistic verification is also used to verify the operation of the modules that provide services, the residual probability of failure of these modules must be considered in the probabilistic verification of any module.

I. INTRODUCTION

Sensing, communications, and vehicle control are being integrated in intelligent vehicles. Vehicles are evolving from autonomous vehicles that sense their environments and control their operation, to vehicles that communicate and coordinate their maneuvers. Vehicle platoons are being investigated in the Grand Cooperative Driving Challenges [1]. Other examples include [2], [3].

Before allowing these cooperative vehicles on public highways, we must guarantee that they will not cause accidents. Simulation and test tracks are the prevalent techniques that are used to verify that such systems are safe. However, the tolerance for failure in automobiles is much lower than can be guaranteed by these techniques. For example, recently, faulty ignition switches on several models of vehicles manufactured by General Motors resulted in a massive recall of vehicles [4]. According to the article, there were 52 crashes that are known to be caused by the faulty switch. At least 2.6 million vehicles were recalled in the year of 2014. Suppose each vehicle with a faulty switch spent at least 1,000 hours on public roadway from the time it left the auto plant. There were 52 crashes in $(2.6 \times 10^6 \times 10^3)$ hours of driving, or approximately 2 crashes every 10^8 hours of driving. It is unlikely that a failure this rare will be detected by either simulation or closed-track testing.

We use probabilistic verification [5], which belongs to a larger class of explicit-state model checking techniques [6],

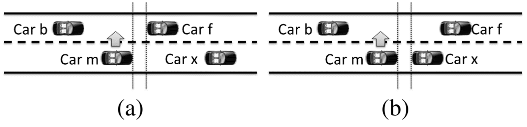


Fig. 1: Car m attempts to merge between car f and car b

to search for such failures. Explicit-state model checking techniques evaluate a system by examining all or some of the sequences of operations against a set of properties. Cooperative driving applications are time-critical and operate over imperfect communication channels. The size of the model of an automobile is large, and the complete model that describes the interactions between vehicles grows exponentially with the number of participants. Furthermore, automotive applications interact with the physical world in different ways, including sensing surroundings of a vehicle, controlling vehicle movements, exchanging messages through communication channels, and so forth. Each interaction with the physical world is susceptible to various types of failures and errors. For instance, transmitted messages may not be received in a timely manner or sensors may measure distances inaccurately.

Probabilistic verification is well suited for evaluating very large systems. It explores as many states as possible, given the limited memory and running time constraint imposed by the computer, and produces a bound for failure probability, as opposed to most explicit-state model checkers, which must explore all reachable states. In addition to its ability to tackle complicated systems, it is able examine combinations of infrequent failures, that arise because of the unpredictable nature of physical world, more quickly than conventional simulators. Probabilistic verification starts by examining system states that occur with higher probability, then it evaluates the states that are less likely to be encountered. More frequently occurring sequences of events, such as those that have no failures, or those that have a single failure, are only examined once before moving on to combinations of failures. As we explore less frequent sequences, we bound the probability of unexplored sequences and stop when we obtain a probability bound that satisfies the expected degree of confidence or run out of computing resources.

In this paper, we design and verify a conceptual three-party merge protocol. It assists a driver (car m) who attempts to merge between the other two vehicles (car f and car b), as shown in Fig. 1a. The protocol considers semi-automated vehicles that are fully automated in the longitudinal direction, using existing adaptive cruise control (ACC) [7]. The merge protocol creates a safe gap and notifies the driver when it is safe to merge. The driver retains steering decisions and is responsible for switching lanes. Semi-automated merge

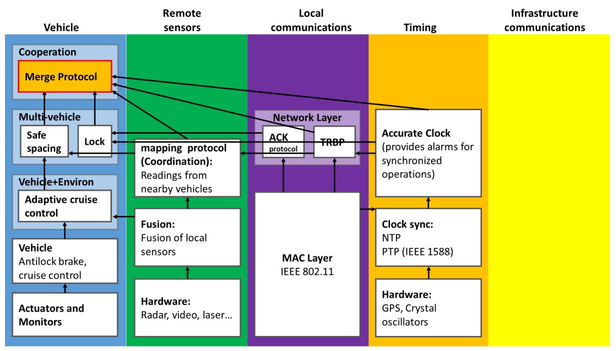


Fig. 2: Merge Protocol in the Architecture

protocols are a crucial step toward fully automated protocols. The public should be convinced that these applications improve safety before they are asked to give up control of the vehicle.

The merge protocol is defined within a multiple stack architecture [8] that addresses the multiple interactions with the physical world. The architecture allows the designer to concentrate on the coordination between participating vehicles, without being concerned with the details of measuring distances, communicating over unreliable channels, or the physics of controlling the movement of the vehicle. These implementation details are addressed in other modules, and are accessed through a well-defined interface. The merge protocol interacts with the physical world through the stacks defined in the architecture. Each module in the stack is verified separately. The probability of failure, or the occurrence of an unexplored state, of a module depends on the states within the module and the failure modes and unexplored states of the modules from which it receives service. For instance, the merge protocol that we evaluate is simple enough to examine all of the states, and guarantee that it will operate safely when the other modules operate correctly, but the merge process can fail if the sensors fail. Probabilistic verification evaluates the entire system, considering the failure modes exhibited by the service-providing modules. It also provides an indication of which modules should be improved to reduce failure probability.

Reasonably accurate clocks are used to simplify the design of time-critical protocols. The protocol relies on the timing information obtained from Global Positioning System (GPS) rather than on timers. Using clocks that read accurate time, we are able to avoid the interleaving of timer expirations, which reduce the number of reachable states and hence reduces the problem size of verifying the correctness of protocol design.

This paper is organized in the following order. Section II briefly describes the multiple stack architecture. In Section III, we present the specification of the merge protocol. How verification is carried out and its result are presented in Section IV. Finally, Section V concludes the paper.

II. MULTIPLE STACK ARCHITECTURE

The multiple stack architecture is shown in Fig.2, and is described in detail in [8]. It isolates the physical devices that perform environment sensing, vehicle control, communications, and clock maintenance from the intelligent vehicle applications. The architecture consists of five interacting stacks: the vehicle stack, the remote sensors stack, the local and infrastructure communications stacks, and the timing stack. The components in the architecture provide services to the intelligent vehicle applications via well-defined interfaces, which allows component-wise design and testing, as in communication networks.

Fig.2 shows how the modules that the merge protocol depends on are located in the architecture. The merge protocol itself belongs to the cooperation layer of the vehicle stack. It depends on modules of the safe spacing systems, the lock protocol, the mapping protocol, a simple acknowledgement protocol, the timed reliable broadcast protocol (TRBP) [9], and accurate clocks. The required services of these modules are specified in Section III. There is no modules shown in the infrastructure communications stack, because the merge protocol does not involve communications with the infrastructure.

Note that the dependency among components in the architecture is acyclic. This implies that we are able to verify and test a particular component given that the functions it depends on have been verified and tested.

III. SPECIFICATION OF MERGE PROTOCOL

The merge protocol is the main logic that dictates the actions of the three participating vehicle, namely car m, car f and car b shown in Fig.1. It consists of four finite-state machines (FSMs) that jointly describe the interaction with the other modules in the system in order to assist the driver in changing lanes. The implementation details of these modules are abstracted out by the multiple stack architecture, and are replaced by interfaces consisting of simple messages. The FSMs of the merge protocol operate by passing messages across these interfaces. The evolution of events can then be described in three phases.

The merge protocol interacts with the physical world through the following modules that are implemented in the architecture. They are feasible given current technology.

- The safe spacing system controls the distance between vehicles. It uses ACC to adjust the distance according to the output of the mapping protocol.
- The lock protocol [10] creates a cooperating group among the three vehicle and ensures that the group only expires at a prespecified time.
- TRBP detects communication failure [9]. It periodically circulates token, broadcasts and recovers messages among the three vehicles. Communication failure is detected if the token is not recovered within specified time.
- The mapping protocol presents a list of objects in the surroundings of the participating vehicles.
- The timing stack provides accurately synchronized clocks. Clocks are reliably synchronized using a combination of GPS, crystal oscillators, and the precision time protocol as described in [8].

Table 1 summarizes the messages that are being exchanged and the behaviors exhibited by the modules through the simplified interfaces. In the parentheses are the relevant interfacing messages that are available to the FSMs of the merge protocol. Some behaviors of the modules are significantly less likely

Clock

- accepts a value t for the alarm (set(t))
- sends alarm when the clock reads t (alarm(t))
- detects a failure and report (clock-fail, $< p^2$)
- an undetected failure; the clock goes out-of-sync or stops ($< p^4$)

Mapping protocol

- the gap is less than sufficient for merging
- the gap is large enough for merging (gap-ready)
- there is an interfering vehicle entering the group (interference)
- fails to provide a mapping (inconsistent, $< p$)
- provides a wrong mapping, with the gap in fact too small ($< p^4$)

Lock protocol exhibits the following sequence:

- Succeeds with both car f and car b cooperating
- reports failure because either car f or car b is unable to cooperate
- reports failure because experienced message loss ($< p$)
- unexplored behavior, if any, occurs with low probability ($< p^4$)

Safe spacing system

- aligns the merging vehicle with the gap (align)
- increase the headway of the back vehicle to create a gap (create)
- instruct the front vehicle to maintain speed (maintain)
- cruise control dysfunction

TRBP

- detects communication loss (comm-loss)
- distributes urgent message (emergency)

TABLE I: Simplified interfaces for the modules

than the other, and they are annotated with probability $< p^k$ for some integer k, with $p = 10^{-4}$. The behaviors that are not labeled with probability are the more common events to which we cannot associate a small probability bound. We conservatively assume that it occurs with some probability ≤ 1. Assignment of probability to events is discussed in Section IV.

On each vehicle there are implementations of all four FSMs of the merge protocol, namely M_{merge} (Fig.4a), M_{front}, M_{back} (Fig.4c), and M_{HMI} (Fig.4b). M_{merge} is activated whenever a driver switches on the turn signal so as to change lanes. M_{HMI} represents human machine interface (HMI) between the logics of the merge protocol and the driver who attempts to change lanes. M_{front} and M_{back} are nearly identical in their FSM specification, except that they issue different commands to the safe spacing system. M_{front} and M_{back} are activated when a vehicle that is not participating in another merge receives a request from the merging vehicle indirectly through the lock protocol; M_{front} is activated when the participating vehicle is in front of the gap, while M_{back} is activated when the participating vehicle is behind the gap.

Fig.3 shows that how messages are passed between interfaces and the FSMs across different vehicle. Although the mapping protocol, TRBP, and the lock protocol appear as a single block across three vehicles, each of them consists of a three distinct implementations, one on each of the three vehicles. They become a congruent block as the communications between the physically-separated implementations are hidden from the merge protocol. M_{merge}, M_{front} and M_{back} do not talk to each other directly. M_{merge} communicates with M_{front} and M_{back} by using the lock protocol to create a group until a specified deadline. The lock protocol then exchanges message via a communication channel that provides at least best-effort message delivery. M_{front} and M_{back} create gap by giving instructions to safe spacing systems on the respective vehicles in which they reside.

The merge protocol operates in three phases, each phase accompanied by a lock request which allocates time for the

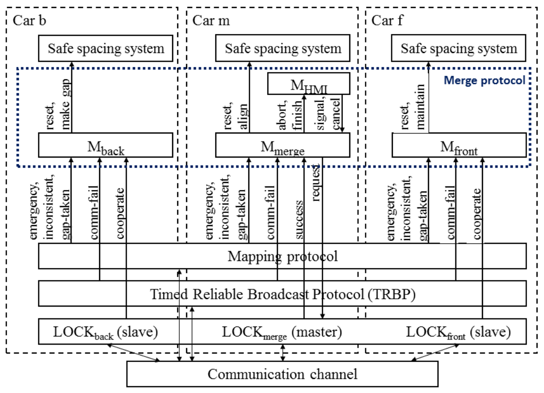
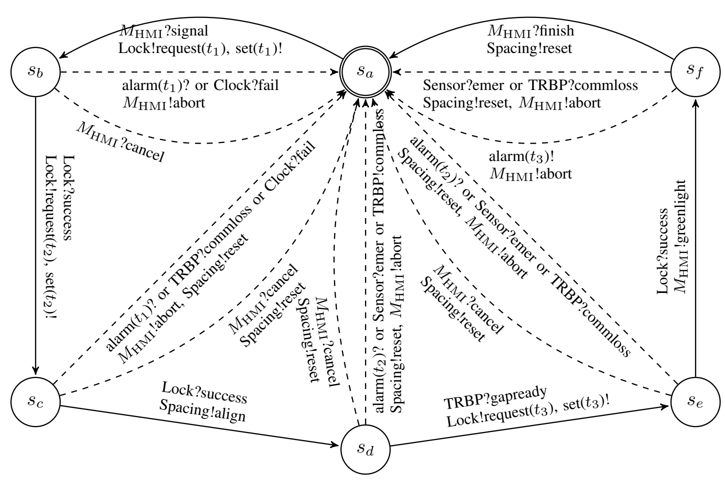


Fig. 3: Interactions between FSMs and the interfaces

merge protocol to complete some actions during the phase. The solid arrows in the specification diagrams corresponds to these phases. In the parentheses are the associated transitions in each state machine.

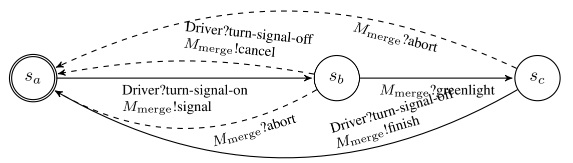
1) Phase 1 starts when the driver switches on the turn signal ($M_{\text{HMI}} : s_a \to s_b$). As M_{merge} being activated, it uses the lock protocol to form a group until time t_1, and it also sets an alarm that will expire at time t_1 (M_{merge}: $s_a \to s_b$). If both car f and car b have the merge protocol available and not participating in any action, M_{front} and M_{back} are then activated and agree to cooperate (M_{front}/M_{back}: $s_a \to s_b$). The lock protocol returns success message and the merge protocol moves on to phase 2.
2) In phase 2 the merge protocol attempts to create the gap. It uses the lock protocol to extend the group to expire at $t_2 > t_1$, and it sets an alarm at time t_2 (M_{merge}: $s_b \to s_c$). On receiving cooperate request from the lock protocol, M_{front} instructs the safe spacing system to maintain speed while M_{back} instructs the safe spacing system to increase headway (M_{front}/M_{back}: $s_b \to s_c$). When notified of the success, M_{merge} instructs the safe spacing system to align itself with the gap by following car f or car x, whichever is closer, as shown in Fig.1a and Fig.1b (M_{merge}: $s_c \to s_d$). When the mapping protocol reports that the gap is large enough for merging, the protocol moves on to phase 3.
3) The driver is required to steer the vehicle into the gap in phase 3. It again uses the lock protocol to extend the expiration of the group to $t_3 > t_2$ and sets the alarm at t_3 (M_{merge}: $s_d \to s_e$). After both M_{front} and M_{back} agree to cooperate until t_3 (M_{front}/M_{back}: $s_c \to s_d$), the M_{merge} issue a greenlight message to the HMI that notifies the driver that it is clear to change lanes (M_{merge}: $s_e \to s_f$, $M_{\text{HMI}} : s_b \to s_c$). After steering into the target lane, the driver completes the action by switching off the turn signal. M_{merge} returns to the initial state (M_{merge}: $s_f \to s_a, M_{\text{front}}/M_{\text{back}} : s_d \to s_a, M_{\text{HMI}} : s_c \to s_a$).

When any of the undesirable events occurs, such as communication loss, an interfering vehicle that does not belong to the group, or driver ignoring the prompt to switch lanes, the protocol executes escaping transitions (dashed arrows in the specification), notifies the driver, reset the safe spacing system



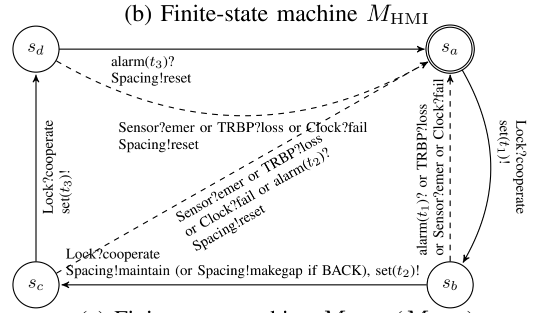
s_a
s_b
s_c
s_d
s_e
s_f
M_{HMI}?signal
Lock!request(t_1), set(t_1)!
alarm(t_1)? or Clock?fail
M_{HMI}!abort
M_{HMI}?cancel
M_{HMI}?finish
Spacing!reset
Sensor?emer or TRBP?commloss
Spacing!reset, M_{HMI}!abort
alarm(t_3)!
M_{HMI}!abort
Lock?success
Lock!request(t_2), set(t_2)!
alarm(t_1)? or TRBP?commloss or Clock?fail
M_{HMI}!abort, Spacing!reset
M_{HMI}?cancel
Spacing!reset
M_{HMI}?cancel
Spacing!reset
alarm(t_2)? or Sensor?emer or TRBP?commloss
Spacing!reset, M_{HMI}!abort
alarm(t_2)? or Sensor?emer or TRBP?commloss
Spacing!reset, M_{HMI}!abort
M_{HMI}?cancel
Spacing!reset
Lock?success
M_{HMI}!greenlight
Lock?success
Spacing!align
TRBP?gapready
Lock!request(t_3), set(t_3)!

(a) Finite-state machine M_{merge}



s_a
s_b
s_c
Driver?turn-signal-off
M_{merge}!cancel
M_{merge}?abort
Driver?turn-signal-on
M_{merge}!signal
M_{merge}?greenlight
M_{merge}?abort
Driver?turn-signal-off
M_{merge}!finish

(b) Finite-state machine M_{HMI}



s_a
s_b
s_c
s_d
alarm(t_3)?
Spacing!reset
Sensor?emer or TRBP?loss or Clock?fail
Spacing!reset
Sensor?emer or TRBP?loss
or Clock?fail or alarm(t_2)?
Spacing!reset
Lock?cooperate
set(t_3)!
alarm(t_1)? or TRBP?loss
or Sensor?emer or Clock?fail
Lock?cooperate
set(t_1)!
Lock?cooperate
Spacing!maintain (or Spacing!makegap if BACK), set(t_2)!

(c) Finite-state machine M_{front} (M_{back})

Fig. 4: Specification of the merge protocol

for a safe, autonomous driving mode in which the vehicle maintains its lane and safe spacing.

Note that determining the value of t_1, t_2, and t_3, which are essentially the respective duration of phase 1, phase 2, and phase 3, is beyond the scope of this paper. Phase 1 can be short, within which the merge protocol only has to establish a group. The duration of phase 2 must be long enough to account for the time needed for the ACC to create a gap, and it depends on the vehicles speed, acceleration, traffic condition, etc. The duration of phase 3 must be long enough for the driver to react to the green light and steer into the gap without haste.

The merge protocol is expected to satisfy the following properties. We use probabilistic verification to establish the bound for the probability that these properties are violated.

1) Creates the gap and notifies the driver only if none of the undesirable events occurs
2) If any of the undesirable events occurs, the driver is notified and the ACC is reset to conservative mode for emergency maneuver

IV. VERIFYING THE MERGE PROTOCOL

To verify the merge protocol we construct execution sequences of system states and search through states that are reachable from the initial state, looking for system failure. A system state consists of states of the four FSMs and the interfaces of other modules. We may identify some events exhibited by the interfaces to occur with probability significantly lower than the other, and vice versa. The transitions among system states can thus be labeled as either high probability transitions and low probability transitions due to this events with skewed probability. Consequently, probabilistic verification can start by exploring states that are more likely to be reached from the initial states, then the states that are less likely to be reached. Even when only a subset of reachable states have been explored, probabilistic verification is able to obtain a bound for the failure probability. The bound for failure probability is a measure of confidence in the protocol.

A system state is essentially a snapshot of the collective behavior of every contributing component in the system. This includes the states of four FSMs and the interfaces of TRBP, mapping, lock protocol, timing stacks and safe spacing systems. An execution sequences is a series of system states, e.g. $(M_{\mathrm{merge}} = s_a, M_{\mathrm{front}} = s_a, M_{\mathrm{back}} = s_a, M_{\mathrm{HMI}} = s_a, \ldots) \to (s_a, s_b, s_a, s_b, \ldots) \to (s_a, s_b, s_b, s_b, \ldots) \to \ldots$

Note that the merge protocol is a time-critical protocol. It employs synchronous clock to realize time-sensitive actions, such aborting the merge if cruise control fails to create the gap within a specified time limit. In some transitions it sets alarms, which then trigger other transitions at some later time. The value of the alarm can be delivered to physically-separated state-machines, in which the alarms can trigger transitions simultaneously as accurate clocks are provided everywhere. This reduces the number of sequences needed to be considered in verification [10].

The merge protocol itself is relatively simple. However, its simplicity does not guarantee that the merge protocol is safe because of the other modules that it depends on. These modules either directly or indirectly interact with the physical world, which contribute uncertainty.

For instance, since the mapping protocol uses multiple sensors to estimate a distance, it is resistant to single sensor failures. The fusion algorithm can reject an incorrect sensor readings by comparing it with other sensors. However, it is possible on some rare occasions for all of the sensors to provide incorrect measurements, which can result in an incorrect mapping of the surrounding objects. The mapping protocol still renders the mapping, but in fact it fails silently without notifying the merge protocol. Even though the probability of such rare events can be decreased by either introducing more sensors or employing a more sophisticated fusion algorithm, the probability of failure cannot be reduced to zero. In the analysis of the merge protocol, we assign the probability of a silent mapping failure a bound of p^4, with $p = 10^{-4}$. The bound p^4 must be verified by performing probabilistic verification on the mapping protocol. It can be viewed as a design constraint, in that any mapping protocol must be verifiable to this level for the merge protocol to be safe. The mapping protocol can also report an inability to generate a map when sensor readings are inconsistent. Depending on the mapping protocol, these reports are more likely than silent failures, but less likely than single failures. In this analysis, we assume that the probability of this report is less than p^2.

We assign a probability bound to the events in table 1 which are less likely to occur than the others. Similarly, the timing stack can fail, too. The timing stack may have one or more free-running circuits that detect and report failure of the GPS-disciplined clock. The clock may also fail silently if the clock and all circuits fail. The lock protocol, which was formally proven by probabilistic verification in [10], has unverified sequences with a very small probability. The assignments of probability bounds to the events in table 1 is qualitative at the time of this writing. They are not bounds obtained from the probabilistic verification of a specific protocol, except for the lock protocol. We have assigned these probabilities to show how they are incorporated into the verification of the merge protocol. For the events that are not associated with a small probability, we assume that they occur with probability ≤ 1.

Once the low probability events are labeled, probabilistic verification is able to differentiate between states that are more likely to be reached than others. Initially, it verifies the sequences that do not contain any low probability events. Next, it verifies the sequences that contains one low probability transition with probability $< p$. Then the sequences either contain two transitions with probability $< p$, or transition with probability $< p^2$, and so on. Among the states it searches for the states that violate the properties presented in table 2. Note that we do not assume that we can determine the exact probability of low probability events, but only require that their probability be less than p, p^2, or p^4, for some value of p.

We have found 402 states that can be reached via sequences containing 3 transitions with probability $< p$, or 1 transition with probability $< p$ and one with probability $< p^2$, among which there is no violation of properties, deadlock, or live-locks. While there are still an unknown number of states yet to be explored by probabilistic verification, the procedure is able to compute the bound for probability of reaching those unknown states, which is upper-bounded by $199p^4$. We may conclude that the protocol is safe if the bound is sufficiently small to provide the confidence required by the cooperative driving application.

Furthermore, if the $199p^4$ still does not give the expected degree of confidence, we can continue probabilistic verification further to consider states that are reached via sequences containing 4 transitions of probability $< p$, 2 transitions of probability $< p^2$, a single transition of probability $< p^4$, and so forth. If we find that an error states occurs among these newly reached states, the cause of error can be identified, and the necessary algorithm or hardware can be modified. For the conceptual protocol in this paper, the mapping protocol failing silently causes the merge protocol to fail. We now know that we can improve the mapping protocol by introducing redundancy to the set of sensors or using a more advanced fusion algorithm, and drive the probability of silent failure smaller.

V. CONCLUSION

Vehicles require a higher degree of reliability than most engineering systems. Errors can result in the loss of life. We cannot achieve the required degree of reliability with test tracks and simulation alone. We must model the systems and determine the effects of the interplay of failures due to direct and indirect interactions between the physical world and the modules. This work is more than model checking a conceptual merge protocol. We use an architecture to address how the application interacts with the physical world through modules in multiple stacks. These modules are not perfect and they sometimes fail. Their failure modes have to be considered when evaluating the system, as there can be multiple ways to enter the failure modes of a particular module.

The result of probabilistic verification implies that the probability of an unexplored sequence is less than $199p^4$, or less than once every 5×10^{13} protocol invocations when $p < 10^{-4}$. Suppose there are 250 million vehicles [1] equipped with the merge protocol, each of which travels 200,000 miles during its lifetime [12]. If the owner of every vehicle drives aggressively, and attempts to change lanes once every mile [2], on the average less than one unexplored state will occur during the lifetime of all of these vehicles. Clearly, this level of confidence will never be achieved by driving vehicles on a test track.

REFERENCES

[1] J. Ploeg, *et al.*, "Introduction to the special issue on the 2011 grand cooperative driving challenge," *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 3, pp. 989–993, 2012.

[2] M. Segata, *et al.*, "Supporting platooning maneuvers through ivc: An initial protocol analysis for the join maneuver," in *Wireless On-demand Network Systems and Services (WONS), 2014 11th Annu. Conf. on*. IEEE, 2014, pp. 130–137.

[3] L. Guvenc, *et al.*, "Cooperative adaptive cruise control implementation of team mekar at the grand cooperative driving challenge," *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 3, pp. 1062–1074, 2012.

[4] J. Dye, "Deaths linked to GM ignition-switch defect rise to 27," *Reuters*, Oct. 2014.

[5] N. F. Maxemchuk and K. Sabnani, "Probabilistic verification of communication protocols," *Distributed Computing*, vol. 3, no. 3, pp. 118–129, 1989.

[6] J.-P. Katoen and C. Baier, "Principles of model checking," 2008.

[7] A. Vahidi and A. Eskandarian, "Research advances in intelligent collision avoidance and adaptive cruise control," *IEEE Trans. Intell. Transp. Syst.*, vol. 4, no. 3, pp. 143–153, 2003.

[8] S.-p. Lin, Y. Gu, and N. F. Maxemchuk, "A multiple stack architecture for intelligent vehicles," in *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*. IEEE, 2014, pp. 268–273.

[9] N. F. Maxemchuk and D. H. Shur, "An internet multicast system for the stock market," *ACM transactions on computer systems*, vol. 19, no. 3, pp. 384–412, 2001.

[10] S.-p. Lin and N. F. Maxemchuk, "The fail-safe operation of collaborative driving systems," *Journal of Intelligent Transportation Systems*, pp. 1–14, 2014. [ahead-of-print]. Available: http://www.tandfonline.com/doi/abs/10.1080/15472450.2014.889932

[11] (2015, Feb.) 2013 production statistics. [Online]. Available: http://www.oica.net/category/production-statistics/2013-statistics/

[12] D. Ford, "As Cars Are Kept Longer, 200,000 Is New 100,000," *The New York Times*, Mar. 2012.

[13] (2015, Feb.) A comprehensive examination of naturalistic lane-changes. [Online]. Available: http://www.nhtsa.gov/DOT/NHTSA/NRD/Multimedia/PDFs/Crash%20Avoidance/2004/Lane%20Change%20Final.pdf

[1] 87 million vehicles are produced in the single year of 2013 [11]

[2] US drivers average one lane change every 2.8 miles [13]