Self-stabilizing Manoeuvre Negotiation: the Case of Virtual Traffic Lights

António Casimiro, Univ. Lisboa, casim@ciencias.ulisboa.pt
Emelie Ekenstedt, and Elad M. Schiller, Chalmers Univ. Tech. {emeeke@student.,elad@}chalmers.se

Abstract—**The vision of automated driving promises to have safer and more cost-efficient transport systems. Automated driving systems have to demonstrate high levels of dependability and affordability. Recent advances of new communication technologies, *e.g.,* 5G, allow significant cost reduction of timely shared sensory information. However, the design of fault-tolerant automated driving systems remains an open challenge. This work considers the design of automated driving systems through the lenses of *self-stabilization*—a very strong notion of fault-tolerance. Our self-stabilizing algorithms guarantee, within a bounded period, recovery from a broad fault model and arbitrary state corruption. After this recovery period, our algorithms provide safe maneuver execution despite the presence of failures, such as unbounded periods of packet loss and timing failures as well as inaccurate sensory information and malicious behavior. We evaluate the proposed algorithms through a rigorous correctness proof and a worst-case analysis as well as a prototype that focuses on an intersection crossing protocol. We validate our prototype via computer simulations and a testbed implementation. Our preliminary results show a reduction in the number of vehicle collisions and dangerous situations.**

I. Introduction

We propose fault-tolerant services for maneuver negotiation.

Task definition. Traffic rules help to avoid ambiguity and collisions when vehicles cross intersections. In the absence of traffic lights, each vehicle has to approach the intersection border in a way that allows it to observe all other road users that it must give way to. It can then cross the intersection before any higher priority vehicles as long as it does not *violate their priority*, *i.e.,* cause a collision, cause them to change their velocity, or bring them to perceive risk. We define the task of a *virtual traffic light* (VTL) as one that considers the assignment of the intersection entry priorities, such that no two equal-priority vehicles have intersecting (yet different) paths, as well as the safe transition from one priority assignment to another. We consider two specifications of these assignments. The basic specification, at all times, allows only the highest priority vehicles to enter the intersection (provided nothing else endangers them). The extended specification allows the lower priority vehicles to negotiate their entry with the higher priority vehicles.

Safety criteria. Safe solutions always have bounded *levels of system safety* (LoSS), preferably zero. Our LoSS includes collisions and dangerous situations. A *collision* occurs when vehicles occupy the same space. We regard a situation as *dangerous* if the distance between two vehicles' fronts is $4\,m$ or less.

Vehicular model. We consider automated driving systems for which an *agent* has the sole access to the vehicular control module. This agent represents a subsystem that is capable of safely driving the vehicle through the intersection as long as it is provided with a correct stop or go input.

Lefèvre *et al.* [1] consider such an agent and use a *(statistical) risk estimator* (RE) for providing the input. Their RE uses only information received via periodic broadcasting of all vehicle locations. Machine learning is used for predicting the other agent's intention w.r.t. its forthcoming maneuvers without receiving any explicit declaration. According to the discrepancy between the estimated intentions and the behavior expected from agents that obey the traffic rules, the RE decides probabilistically whether to instruct stop or go.

Our negotiation protocol assumes local access to the maneuver intention declarations. It provides input to the agent via one of three methods. The $\mathcal{M}_{MN}$ method considers only the declared intentions, the $\mathcal{M}_{RE+MN}$ method uses a variation of Lefèvre *et al.*'s RE that has access to the declarations, and the $\mathcal{M}_{RE}$ method simply uses Lefèvre *et al.*'s RE [1].

Fault model. We consider a message-passing system that is prone to packet loss, duplication and reordering. We assume that all nodes have access to the coordinated universal time, *e.g.,* via GPS. However, the nodes are prone to timing failures, *i.e.,* a temporary inability to respond, within a bounded time, to an event raised by an expired timer. We refer to packet and timing failures as *benign failures* and assume that, in their absence, all messages arrive and are processed within a bounded time. We allow repeated occurrence of benign failures for an unbounded (yet finite) period, which implies unbounded communication delays and response times. We assume that, in the presence of benign failures, the system's core safety-critical functionalities are subject to *(bounded) accuracy degradation* of the location of nearby vehicles due to the lack of updated sensory information (either received from local sensors or from nearby vehicles). As *malicious behavior*, we consider only the case in which a low-priority agent that received the input stop decides to persistently use go instead.

In addition to handling the above failures, we also aim to recover from *transient violations* of the assumptions according to which the system was designed to operate, such as the corruption of control variables (*e.g.,* the program counter and timestamps), or violation of fault model assumptions (*e.g.,* cases in which a node fail-stops and then restarts). Since the occurrence of these transient violations can be combined, we assume that they can alter the system state in unpredictable ways. In particular, when modeling the system, we assume that these violations bring the system to an arbitrary state from which a *self-stabilizing algorithm* should recover the system. Therefore, starting from an arbitrary state, the correctness proof of self-stabilizing systems has to show the return to a "correct behavior".

Our research question and challenges. We answer the following question. Is there a self-stabilizing deterministic protocol for virtual traffic lights that follows the above vehicular and fault models, that also complies with existing traffic rules, and provides a clear system-safety analysis?

The fault-tolerance aspects of this question are important since negotiation and arbitration services must avoid the split brain phenomenon in which, due to network partitions, different nodes lead to an inconsistent state and by that jeopardize the system safety. More generally, dealing with communication failures, such as asymmetric (and unbounded number of) packet loss, implies the need to circumvent a number of well-know consensus impossibilities. For a worldwide acceptance of such systems it is imperative for the proposed solutions to have safety guarantees that comply with current traffic rules.

Our contribution. We present the first, to the best of our knowledge, system architecture, and matching self-stabilizing algorithms for maneuver negotiation services. We focus on the application of virtual traffic lights (VTLs) and solve both its basic and extended specifications. We give the correctness proof and validate it through computer simulations during which we demonstrated scalability of up to 300 vehicles and, in the presence of repeated benign failures, near-zero LoSS (Level of System Safety) w.r.t. the number of collisions and dangerous situations. A preliminary demonstration of our VTL prototype has a video. Due to the page limit, the detailed discussion and correctness proof appear in [2].

II. The Proposed Solution

The system is composed of nodes (see Figure 1). One of the nodes is called the *service* (or the infrastructure) and the others are called the *mobile nodes*. The system runs several client-server protocols, such as the *membership service* (MS), which we describe first, as well as the *virtual traffic light* (VTL) and *(V2V) maneuver negotiation* (MN) protocols. Similar to the traditional traffic lights, the VTL safely decides, according to congestion information, on the assignment of green and red lights for each path that crosses the intersection. To the end of making sure that all vehicles have sufficient time to leave the intersection before new updates are applied, an orange light period is used. This increases the latency, and thus, we use the MN for allowing safe intersection crossing for individual vehicles (without VTL updates).

The self-stabilizing spacial-tempo membership. This service monitors the spacial-tempo presence of the mobile nodes

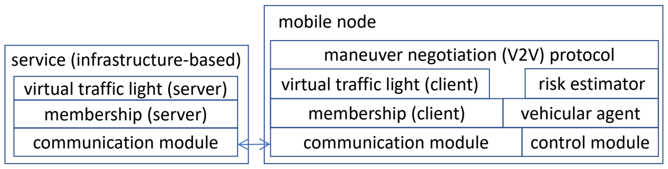

Fig. 1. The proposed architecture for self-stabilizing virtual traffic lights

based on the locations that their clients transmit periodically.

As a basic service, it lists all agents at the intersection along with their current and target lanes. We refer to these lists as the *(membership) view*, M. Every view M of an agent p_i is also associated with the *manoeuvre opportunity* indicator, MO, which is a Boolean that allows the VTL protocols to deal with benign failures. The service assigns False to MO, if and only if, an up-to-date view cannot be provided, say, due to a mobile node that is suspected to be within the intersection (computed using occupancy prediction) and yet fails to timely communicate with the service. In Section II-A, this basic service facilitates a basic VTL solution (Section I).

The extended task specification requires the following extensions to the membership service. A view, M, for an agent p_i contains a set of identifiers of agents p_j whose: (i) path can intersect with the one of p_i's paths, and whose (ii) lane has equal or higher priority than p_i's lane, and whose (iii) distance to p_i is at most d_{max} length units. In Section II-B, this basic service facilitates a basic VTL solution (Section I).

A. Basic virtual traffic lights

Protocol 1 is responsible for (i) scheduling the assignment of priorities for the different paths that cross the intersection and (ii) communicating to all vehicles this schedule in a way that allows them to adapt to updates and tolerate benign failures. The protocol considers three parts of the schedule (line 4): now, $next$ and $prelim$. The protocol makes a preliminary plan according to the number of agents that wish to cross the intersection (line 9). The plan includes a set of non-intersecting directions that have the maximum number of pending agents, and the plan period. After that, the coordinator (on the server-side), $p_{coordinator}$, repeatedly broadcasts the plan (line 12) and collects feedback for it from the mobile nodes (line 13) until it is time to change the phase (line 9). *I.e.,* the $next$ field takes the preliminary plan, $prelim$, and the currently executed plan, now, takes the $next$ plan.

Our experiments included a Python implementation using Asynchronous I/O. We used a Raspberry Pi 3 B for the coordinator (the service node) and up to 600 clients (mobile nodes) using 4 PCs that have emulated up 150 clients each. These clients have communicated with $p_{coordinator}$ over Wi-Fi at the rate of 10 Hz per client. The experiments showed that for up to 300 clients, we could not observe a case in which the default plan was used (when looking at the first 100 phase changes). At 400 clients, the success rate has plunged to 10%.

B. Extended virtual traffic lights

All mobile nodes play both roles of requesters and requestees, see the pair of distributed state-machines in Fig. 2.

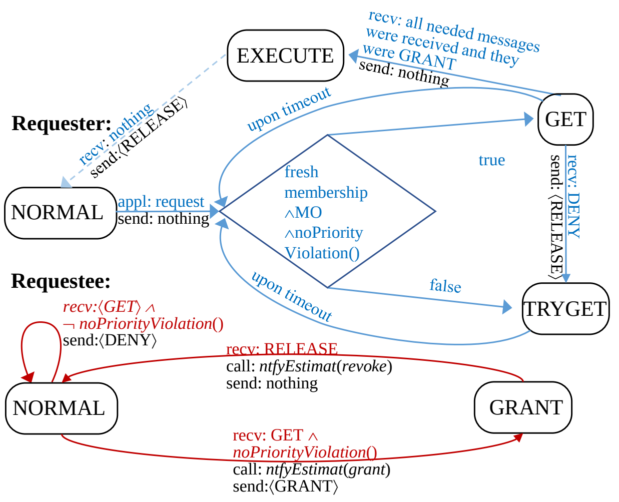

Fig. 2. The dialog between the requester and the requestee depicted by the state-machines above and below respectively.

The function myPriority(m) arbitrates between concurrent requests (even for equal-priority cases), say, by considering the request time, direction, and agent identifiers. The function noPriorityViolation determines if a priority violation can occur between two vehicles whose priorities are decided by the basic virtual traffic light (Section II-A). The function can be based on occupancy prediction or Lefèvre *et al.* [1].

The requester automaton includes a pair of states for facilitating a reliable transmission of the request message, $\langle GET \rangle$. Whenever it is time to send a request, the requester enters the state GET directly or passes through state $TRYGET$ first before reaching state GET. From state GET, p_i periodically multicasts $\langle GET \rangle$ messages. Once requester p_i receives a $\langle DENY \rangle$ message, p_i defers communication by entering the state $TRYGET$ until it is time to return to GET. The automaton of requestee p_j calls ntfyEstimat(grant) and replies with a $\langle GRANT \rangle$ to p_i if, and only if, noPriorityViolation$()$ = True and: (i) p_j's status is $NORMAL$ or $TRYGET$, (ii) p_j's status is $GRANT$ and its state encodes that the grant is already given to p_i, or (iii) p_j's status is GET (p_i and p_j have concurrent requests) and p_i has the higher priory according to myPriority(m). Otherwise, p_j replies to p_i with a $\langle DENY \rangle$ message. The requester p_i multicasts $\langle RELEASE \rangle$ messages in the following cases: (i) p_i's status is GET and it receives a message $\langle DENY \rangle$ from p_j, (ii) p_i's status is GET and its timer expires so that it needs to defer its requests by changing status to $TRYGET$, (iii) p_i's status is $EXECUTE$ and it has just left the intersection. Node p_j also calls ntfyEstimat(revoke) when p_j status is $GRANT$ (its state encodes that it has granted permission to agent p_i) and it receives the message $\langle RELEASE \rangle$ from agent p_i. It also calls ntfyEstimat(revoke) when p_j's status is $EXECUTE$ and p_j has left the critical section.

Protocol 1: Self-stabilizing virtual traffic light

```
1  **Constants:** dfltPhs := $(0, \bot, \mathsf{False})$: default phase;
2  dfltSch = (dfltPhs, ..., dfltPhs): default schedule;
3  **Structures:** $phase := (set : \text{int}, till : \text{time}/\bot, done : \text{Boolean})$: a
     schedule phase (planned set, expiration time and termination bit);
4  $schedule(now : phase, next : phase, prelim : phase)$;
5  %———————— server-side ————————
6  **Variables:** $state[n] :=$ [dfltSch, ..., dfltSch]: schedule of all agents,
     where $state[i]$ is the schedule of coordinator $p_{coordinator}$;
     $queues[k] := (0, \dots, 0)$: queues for each direction;
7  **once in every (predefined)** $T_{period}$ **time units begin**
8      **if** $\neg state[i].(prelim.till \geq next.till \geq now.till)$ **then**
         $state[i] \leftarrow$ dfltSch;
9      **if** $state[i].now.till = \bot \vee state[i].now.till <$ clock() **then**
10         $state[i] \leftarrow state[i].(next, prelim, (set, (T_{phase} +$
             $state[i].prelim.till)))$ **where** $T_{phase}$ is a predefined value
             and $set$ includes non-intersecting paths for which there is a
             maximum number of pending to cross agents
11     $state[i].prelim.done \leftarrow (state[i].prelim.till \neq \bot) \wedge$
         $\bigwedge_{k \neq i} state[k].prelim.(set, till) = state[i].prelim.(set, till)$;
12     **foreach** *mobile node* $p_k$ **do send** $\langle state[i] \rangle$ **to** $p_k$;
13 **upon** message $\langle m \rangle$ **received from** $p_j$ **do** $state[j] \leftarrow m$;
14 %———————— client-side ————————
15 **Variables:** $localState =$ [dfltSch]: schedule of the client;
     $localOutput : (phase, phase) =$ (dfltPhs, dfltPhs); $interPhs$
     $: phase =$ dfltPhs: intermediate phase between 2 primary phases
16 **do forever (once in every** $T_{period}$ **time units): begin**
17     **if** clock() $> (localState.now.till + T_{period})$ **then**
         $localState \leftarrow localState.(next, prelim,$ dfltPhs$)$;
18     **let** $interPhs :=$ (getInterSet$(localState.now.set,$
         $localState.next.set), (localState.now.till + T_{intermPhase})$
         **where** $T_{intermPhase}$ is a predefined value;
19     **write** $(localState.now, interPhs, localState.next)$ **to**
         $localOutput$;
20     **send** $\langle state \rangle$ **to** $p_{coordinator}$;
21 **upon** $\langle m \rangle$ **arrival from** $p_{coordinator}$ **do** $\{localState \leftarrow m;\}$
```

Our evaluation test-case consider low (V_L) and high (V_H) priority vehicles in which V_L turns left across the priority lane on which V_H is approaching. There were 12 experiments that differ in their level of accuracy of the sensory information, the starting distance from the intersection and when benign failures occurred. We also consider malicious behavior. Each experiment was run 10 times with different seeds for the random number generator. $\mathcal{M}_{MN}$ and $\mathcal{M}_{RE+MN}$ (Section I) yielded lower values than $\mathcal{M}_{RE}$ in all cases. $\mathcal{M}_{MN}$ resulted in less collisions than $\mathcal{M}_{RE+MN}$ while $\mathcal{M}_{RE+MN}$ resulted in less dangerous situations than $\mathcal{M}_{MN}$. Other than the case in which V_L acts maliciously, for $\mathcal{M}_{MN}$ and $\mathcal{M}_{RE+MN}$, we have observed zero collisions and up to 2 dangerous situations, whereas for $\mathcal{M}_{RE}$, there were between 1 to 7 collisions and 7 to 10 dangerous situations. When V_L acted maliciously, $\mathcal{M}_{MN}$, $\mathcal{M}_{RE+MN}$ and $\mathcal{M}_{RE}$ had 1, 2, and 2 collisions and 4, 2, and 4 dangerous situations, respectively.

References

[1] S. Lefèvre, *et al.* "Intention-Aware Risk Estimation for General Traffic Situations, and Application to Intersection Safety," INRIA, Research Report RR-8379, Oct. 2013.

[2] A. Casimiro, *et al.* "Membership-based manoeuvre negotiation in autonomous and safety-critical vehicular systems," *CoRR*, vol. abs/1906.04703, 2019.