Design and development of a proof-of-concept platooning application using the HIDENETS architecture

Luís Marques
lmarques@lasige.di.fc.ul.pt
FC/UL*

António Casimiro
casim@di.fc.ul.pt
FC/UL

Mário Calha
mjc@di.fc.ul.pt
FC/UL

Abstract

This paper describes the design and development of a proof-of-concept platooning application, which operates in a mobile and dynamic environment and makes use of architectural and middleware solutions that were proposed in the scope of the HIDENETS project. With this application it is possible to demonstrate the practical feasibility of a hybrid system architecture, with realms of operation with distinct synchrony properties, and the benefits of adopting such architecture. In particular, we show that it is possible to improve the performance and behavior of the platooning application, which operates over an intrinsically uncertain environment (due to mobility and wireless communication), and still secure fundamental safety-critical requirements.

1. Introduction

The continuous advances in embedded, sensor and wireless technologies are key to the development of distributed applications and mobility-aware services in ubiquitous communication scenarios, such as car-to-car communication scenarios.

On the other hand, the typical characteristics of these scenarios, involving mobile nodes that communicate through wireless networks, possibly using general purpose operating systems and COTS-based devices, lead to uncertainty on the temporal behavior and on the reliability of the operations taking place therein. Therefore, in these scenarios it is not easy to provide highly resilient and available services and applications.

The well defined objective of HIDENETS [5], the IST-FP6 project in which our work has been developed, is precisely to address this problem, in particular through the definition of appropriate architectural constructs and resilience services.

In this paper we focus on the platooning application, which was one of the proof-of-concept applications that were developed to demonstrate the project results. Briefly, the objective of platooning is to ensure that a set of well-known cars follow each other as close as possible in a way that improves traffic flow, road capacity and fuel consumption [7]. In contrast with typical control approaches, which only make use of locally produced information (from internal car sensors), we consider a platooning application that uses vehicle-to-vehicle communication to obtain additional information and thus be able to make more "intelligent" decisions and achieve an optimized behavior with respect to the real surrounding context. Interestingly, and very importantly, this optimized behavior is achieved without compromising any safety-critical property.

Through the proof-of-concept platooning application described in this paper, we demonstrate the feasibility of some solutions developed in the scope of HIDENETS. In particular, we demonstrate:

- the feasibility of a hybrid system architecture in which different parts of the system have different synchrony properties;
- the benefits of using such hybrid architecture and some supporting services in the construction of a representative platooning application, showing how to achieve an implementation that combines efficiency with safety.

The paper is organized as follows. In Section 2 we briefly present the HIDENETS project and introduce the HIDENETS hybrid architecture. Then, Section 3 provides a general description of the platooning scenario and Section 4 presents the application design and implementation. The demonstration results are addressed in Section 5 and we make final remarks in Section 6.

2. HIDENETS Architecture

HIDENETS is a research project, funded by the European Union, which aims to develop and analyze end-to-end resilience solutions for distributed and mobile applications in ubiquitous communication scenarios, assuming highly dynamic, unreliable communication infrastructures [5]. The project addresses fault tolerance mechanisms at the middleware and communication layers, as well as methodologies to support their evaluation and testing [3].

* Faculdade de Ciências da Universidade de Lisboa. Bloco C6, Campo Grande, 1749-016 Lisboa, Portugal. Navigators Home Page: http://www.navigators.di.fc.ul.pt. This work was partially supported by the EC, through project IST-FP6-STREP-26979 (HIDENETS) and by the FCT through the Multiannual Funding and the CMU-Portugal Programs.

In HIDENETS we followed a hybrid distributed system model approach, with different parts of the system being described by different synchrony and fault models. This hybridization was inspired in the Wormholes model [8], which assumes that it is possible to construct a part of the system with improved properties (the wormhole), which is capable of performing some actions in a faster, more predictable or more secure way than it is possible outside of it. Figure 1 provides a simplified view of the HIDENETS architecture.

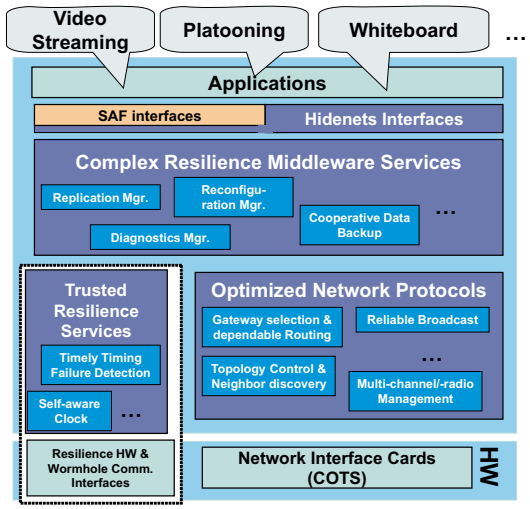

Figure 1. HIDENETS architecture.

There are two distinct domains in this architecture: the bulk or "payload" part of the system, implemented in a general purpose computing node, includes the applications, complex middleware functions and optimized network protocols over standard communication infrastructures; the "wormhole" part of the system consists in a set of simple and resilient services that must be implemented separately, typically within a subsystem with its own computational resources, possibly dedicated communication channels and a clearly defined interface to the payload part. In this paper, and in the context of the platooning application, we refer to a particular implementation of a wormhole and we briefly describe the services that it provides to the platooning application software running in the payload.

We must note that this architecture is well suited to real vehicles. Modern cars are equipped with a huge amount of sensors, connected to central control units through one or several networks, and grouped in accordance to their purpose and/or criticality. A car can be seen as a heterogeneous computational environment, with distinct parts, each with their own synchrony and reliability properties and with their own purpose. In such real car, the wormhole part of our system would have to be integrated as an extension of the critical and predictable car control subsystems, from where it could reach the relevant actuators and perform timely operation. On the other hand, the payload part would be the general purpose computational infrastructure, where non-critical applications (e.g., infotainment, entertainment, passive security) are executed and provided to passengers.

3. Platooning scenario and assumptions

We consider a platooning application for a predefined set of cars. Each car is equipped with a front proximity sensor, communicates over a wireless network and obtains its position through a GPS receiver. For simplicity we only consider positioning on a single dimensional axis and we assume that lateral steering is not required, or is automatically granted.

In the specification of the platooning application we define two fundamental requirements. A safety-critical requirement is that cars have to maintain a minimum distance to the front car, defined as Δ_{min}. This distance will vary over time, depending on the car speed and environment conditions. A non safety-critical requirement is that each car has to maintain a maximum distance to the rear car, defined as Δ_{max}, to avoid platoon fragmentation. The Δ_{max} requirement depends on the maximum communication range assumed for the wireless network.

We further consider that the driver has control of when the platooning application becomes active, by pushing a platoon activation button (similarly to a traditional cruise control system).

4. Design and Implementation

4.1. Overview of platooning scenario

In the conceptual platooning scenario we have a set of mobile entities that are at the same time computational elements (distributed nodes that communicate with each other) and physical entities (cars) that obey physical laws of movement, are operated by a human (driver) and interact with the physical environment through sensors (speed, distance, position, etc) and actuators (accelerator and brakes). All these components are represented in Figure 2, which we now explain.

The bottom part of the figure represents the physical environment, emulated using The Open Racing Car Simulator (TORCS) [6], which offers a graphical interface to observe representations of the cars, their positions and movements (see Section 4.2).

On top of that we have the computational representation of the cars where five cars are illustrated. Each car (a HIDENETS node) is composed by an embedded real-time device (the wormhole) and a generic laptop PC or PDA (the payload), which are connected through a serial RS232 interface. This hybrid node executes all the software that controls the car behavior (the platooning application) and provides interfaces for the driver (accelerator, brake and platoon activation). Sensing and actuation interactions take

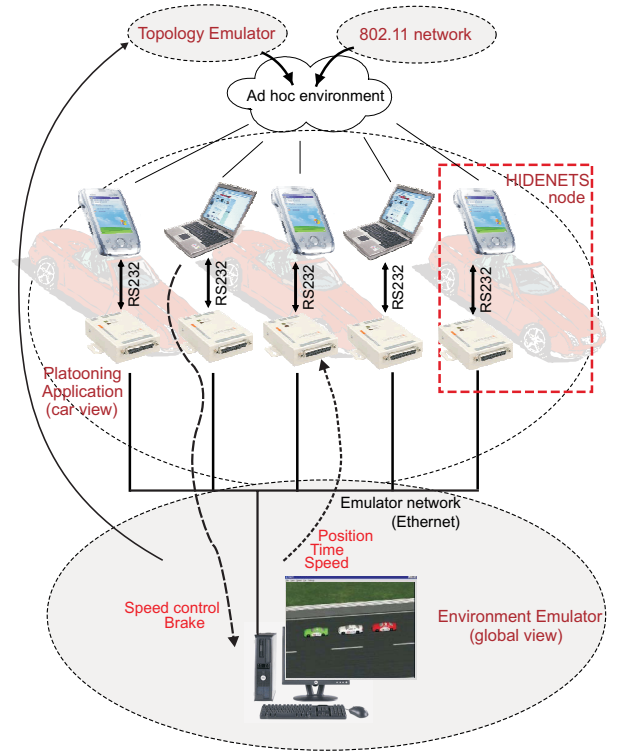

Figure 2. Implementation overview.

place between each node (car) and the environment emulator. Actuation commands include acceleration and brake control, while sensing information includes position, time and speed.

Finally, the demonstration setup also includes the wireless communication network through which the cars exchange information (top part of Figure 2). For the purposes of the demonstration a standard 802.11 network could be used. However, since the nodes in the demonstration are static and physically close to each other, the communication properties would not depend on the distance between cars, maintained by the environment emulator. Therefore, in the demonstration we use a topology emulator (also developed in HIDENETS), which models a 802.11 network and is connected to TORCS to receive information from the positioning of communication entities in run-time.

4.2. Environment emulator

For the simulation of the platooning environment and physics we adapted an open source 3D racing simulator, named The Open Racing Car Simulator (TORCS) [6]. TORCS simulates a race where a configurable number of human and computer drivers each control a chosen car. In every iteration, the simulator provides information about each car situation and evaluates available commands for acceleration and braking.

Several changes were made to better fit the simulator to the platooning scenario. For instance, no steering commands are considered since all cars steer to follow a common path on the right side of the road (controlled by the emulator on its own).

4.3. Platooning application architecture

The platooning application itself is split in two parts: a major part residing on the payload subsystem (on the laptop PC or PDA) and a simpler part in the wormhole subsystem (the real-time embedded device that represents the predictable part of the car infrastructure). This separation can be seen in Figure 3.

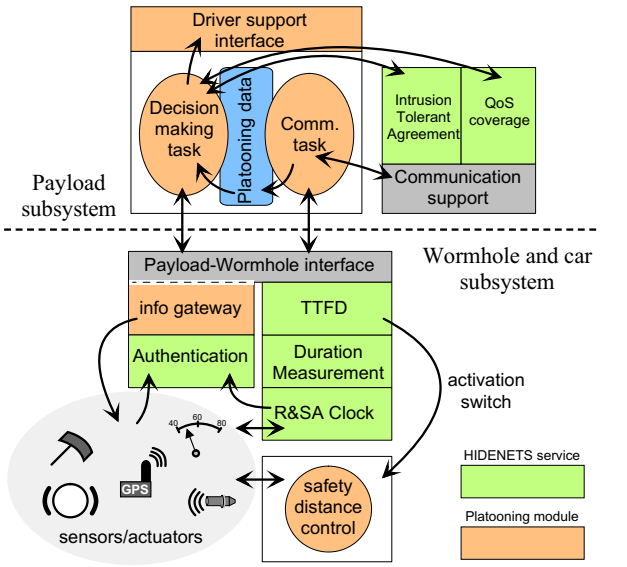

Figure 3. Application architecture.

According to the previously described HIDENETS architecture, the payload is where generic applications execute and where arbitrarily complex computations can take place. Therefore, the platooning application can be made as sophisticated as needed in this part of the system, by executing complex tasks for decision making and for communication, dealing with potentially large amounts and variety of data. The trade-off is the lack of timeliness guarantees, typical in asynchronous systems where an uncertain number of applications compete for the available resources (CPU, memory and network).

Because of that, the platooning application is complemented by baseline mechanisms, implemented within the wormhole subsystem, which assure a safe and timely response to relevant events, namely timing failures caused by the uncertain temporal behavior of the payload part, as we explain in Section 4.4.3. The wormhole implements a payload-wormhole information gateway, though which sensor information is provided to the payload and platooning speed control commands are received.

4.4. Wormhole subsystem

The wormhole was implemented using a dedicated real-time device and operating system. The hardware employed was Lantronix's UDS100 device, which is driven by a 48MHz 8086 compatible CPU and has 256Kb of RAM. The device provides Ethernet and serial connections, which were used to communicate with the environment emulator and the payload, respectively. Additionally, two programmable LEDs, a green and a red one, were used to show the status of the Timely Timing Failure Detection service.

The operating system of the device is proprietary and not fully disclosed, but it is documented as an O.S. with real-time capabilities, which we have explored in the wormhole implementation.

The wormhole essentially includes a task to process incoming requests arriving at the payload-wormhole interface (where admission control is performed), a task to process the periodic interactions with the environment emulator (read sensors and write to actuators) and a task to process internal events, such as timing failures detected by the Timely Timing Failure Detection (TTFD) service. The maximum execution time of these tasks can be bounded, rendering the wormhole a synchronous component, as assumed in the HIDENETS model.

In the wormhole implementation we included three basic services: an Authentication service, a Reliable and Self-Aware Clock service and a TTFD service (which uses Duration Measurement facilities). These are presented in the following sections.

4.4.1. Authentication In real platooning scenarios it may be necessary to deal with malicious behaviors and with false data originating from non-trusted sources. Although this was not the main concern of the HIDENETS project, we exploited the availability of a wormhole subsystem with better properties to add an Authentication service.

In essence, the idea is to use a scheme based on a public-key infrastructure model, in which the wormhole keeps a private key that is used to sign sensible information (e.g., position, speed) collected from sensors before this information is sent to the payload subsystem. Given that, payload applications, and the platooning in particular, may authenticate received information before using it. We note, however, that we are not concerned with security issues in this demonstration.

4.4.2. Reliable and Self-Aware Clock The Reliable and Self-Aware Clock service provides both current time and current synchronization uncertainty, i.e. an estimation of the distance of the local clock from an external global time reference [1]. This allows a client to query not only the (most likely) current time, but also to obtain assured minimum and maximum time values.

In our implementation the R&SA clock is synchronized periodically through the GPS receiver (emulated in TORCS and read as another internal car sensor), thus using a global time reference. In the platooning application the R&SA Clock service is used to timestamp the positioning data shared with other cars. This allows the cars to relate all the received positioning information in a coherent timeline, while also using the most conservative timestamp values, for maximum safety.

4.4.3. Timely Timing Failure Detection In order to ensure the safety properties of platooning, it is essential that actuation commands that determine the speed of the car are executed in a timely way. However, it is clear that the commands issued by the platooning decision control task in the payload subsystem may be subject to unpredictable delays before being executed by the wormhole subsystem.

Because of this, the TTFD service is used to observe timed executions and react upon the occurrence of timing failures. It works like a switch that determines if the car can be controlled by the payload decision control task or, instead, if the safety distance control mechanism becomes active (see Figure 3).

While the car is under the drivers' control, only the safety distance control mechanism is active. Then, when platooning is activated, the payload control task sets up the TTFD service, specifying a deadline for the next control command that will be sent to the wormhole. If the command indeed arrives before the specified deadline, then it is safe to send this command to the actuators (which is done, for predictability reasons, at the deadline instant). While the payload is timely, the information gateway is allowed to accept commands received from the payload and the safety distance control mechanism is disabled. At the same time, the TTFD is re-armed for a new deadline (specified by the payload, along with the actuation command). This process is continuously repeated until a timing failure occurs. The timing failure is timely detected by the TTFD, which immediately activates the safety distance control mechanism and prevents any further commands from the payload to be accepted.

Note that the Duration Measurement service is simply used to calculate the time elapsed since the TTFD was requested to observe the timeliness of a timed action, and the instant in which the action terminates.

4.5. Payload subsystem

The payload is structured as two main tasks, which share an internal data structure with platooning-related information. One task is concerned with exchanging information with the remaining cars (each message contains an identifer of the car and time stamped information about position and speed), storing this information in the shared data structure. This task is always running and disseminating this local information. The other task runs a periodic decision control algorithm, which implements the optimized behaviors of the platooning using the information contained in the shared data structure and interacting with the Intrusion Tolerant Agreement and the QoS Coverage services. These will be explained next.

4.5.1. Decision control algorithm The decision control task wakes up periodically, with a period that is made short enough to allow a smooth control of the vehicle, but sufficiently large to accommodate the reception of new information from other vehicles in between. In each iteration, the task starts by requesting to the wormhole updated information about local time, position and speed, which is stored in the shared information structure. Then, it uses all the available information concerning vehicle positions and speeds, in order to determine a new speed command to be sent to the wormhole. The algorithm makes worst case assumptions about the behavior of neighbor vehicles (in particular, assuming that they have braked to stop just after sending their last positioning/speed information). It also considers the deadline for sending the current actuation command (this deadline has been previously set on the TTFD service), and the deadline for sending the next actuation command (when there will be a new opportunity for setting a new speed, or when will the car will switch into safe mode after receiving the current command). The latter will be sent along with control command to re-arm the TTFD service.

The communication task is simpler. It is activated whenever some message is received from another node, and periodically disseminates local information that it reads from the shared data repository.

4.5.2. Intrusion tolerant agreement The Intrusion Tolerant Agreement service is used, in essence, to allow platoon members to agree on a common platoon speed. It is invoked whenever a driver activates the platooning mode. In the implementation we used an adapted version of the randomized, intrusion-tolerant, asynchronous vector consensus, from the RITAS protocol stack [4]. Although this service is meant to deal with malicious participants, in this specific demonstration we have other goals and therefore do not consider these capabilities, just the possibility of reaching consensus on a value.

4.5.3. QoS coverage As mentioned, the nodes of the platoon exchange messages through the wireless network. These messages are subject to unpredictable delays, and may even be lost. On the other hand, the "freshness" of these messages is important in the decision control algorithm: the optimality of the control decisions depends on this freshness. Since the decision control task executes periodically, and in each execution it sets the next control set-point (deadline for actuation), this deadline should be calculated using a prediction on when there will be new information available from the remaining cars.

The QoS Coverage service is helpful as it provides hints on the message delivery delay that should be considered, holding with a certain specified probability. Using this service, any changes in the communication environment affecting communication delays will be reflected in the considered value and hence in the algorithm deadlines. A detailed description of the probabilistic framework used in this service can be found in [2].

5. Demonstration

In this section we describe and analyze the results obtained by running our proof-of-concept platooning application in a test-bed with 3 nodes, plus the environment emulator node, as shown in Figure 4.

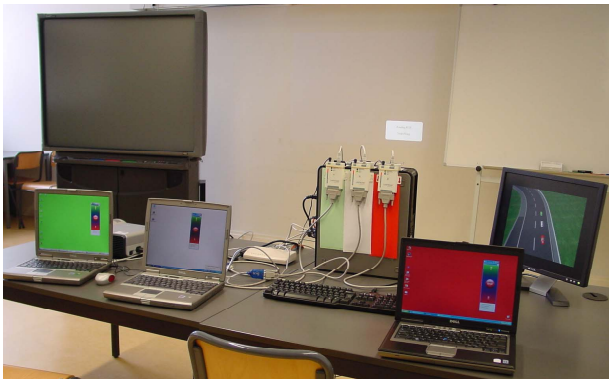

Figure 4. The complete demo setup.

5.1. Normal platooning behavior

When the demonstration is initiated, cars are controlled by their drivers. In this state the safety distance control mechanisms is active, but driver commands always take precedence. In the demo we show that a driver has free control of the car and is able to crash into the front car. Without driver intervention, and given the safety distance control, the car will autonomously brake. We also show that in this state there is no active mechanism in effect to prevent platoon fragmentation.

Once the drivers activate the platooning, the cars agree on a speed value and the platoon is formed. We show that cars can follow each other very closely, even at high speeds, and when the car in the middle of the platoon brakes at full intensity the follower will also stop, without colliding. On the other hand, the decision control algorithm in preceding car will eventually stop this cars also, to secure the maximum distance property.

5.2. Dealing with timing failures

To observe the behavior of the TTFD service, and the detection of timing failures, we use the green and red LED lights of the embedded device. When the TTFD service is activated and does not detect any timing failure, the green LED is blinking. But when a timing failure is detected, the red LED is turned ON. Therefore, it is possible to know if a car is being controlled by the platooning application or by the safety control algorithm, just by observing the LEDs.

time.

When a heavyweight task is ran on the payload system it will compete with the platooning application for CPU time,

and may cause timing failures to occur. In the demonstration we reproduce this behavior and we show that a timing failure is promptly detected. The red LED turns on and the safety distance control mechanism is activated, forcing the car to brake in order to preserve the baseline, non optimal, safety distance.

5.3. Dealing with communication uncertainty

To illustrate the role of the QoS Coverage service, we first show what happens when we artificially inject some delays in the ad-hoc network, while not using the QoS Coverage service. In this situation the decision control algorithm sometimes issues braking commands, even when not necessary (i.e., even when the front car is moving with a constant speed). This happens because control decisions are taken using "old" information about the front car, since updated information is delayed.

When the QoS Coverage service is activated, we observe that when delays are injected the decision control algorithm sends the necessary commands to force a greater distance to the front car. This extra distance allows to extend the control period and the time during which cars can wait for incoming "fresh" information, reducing the probability that communication delays will force unnecessary, inconvenient and constant braking manoeuvres.

5.4. Dealing with temporal inconsistency

In the event of a network crash the information about neighbor cars will not be updated. In practice, as we show in the demonstration, the effect is that cars will eventually stop, either because they believe (from the old available information) that they have a car in front of them, or waiting for a car that they believe has stopped way behind.

Note that in the case of this platooning application, temporally inconsistent information does not affect any safety-critical property – it only compromises the liveness of the system. In fact, this is true because we assume that cars always move forward, but never backwards. In other scenarios, using temporally inconsistent information may have very harmful effects.

Interestingly, the hybrid system architecture that we use to implement the platooning application also facilitates dealing with temporal inconsistency. In fact, the problem mentioned above occurs because the decision control task is running in a timely way, and therefore is allowed to control the car, despite the outdated view of the surrounding environment. The solution consists in simply stopping the decision control task, automatically transferring the control to the safety distance control mechanism. For doing that, the decision control task evaluates the freshness of the available information in order to decide when to silently stop.

In the demonstration we show the behavior of the platoon in the two situations, with and without this temporal inconsistency detection mechanism activated. Therefore, when we crash the network preventing any further communication, we observe that with the temporal inconsistency detection the cars keep on moving, just ensuring a (non-optimistic) safe distance to the front car, until they eventually stop if the driver does not take get in control.

6. Conclusions

In this paper we analyzed the design and implementation of a distributed platooning application, built using the HIDENETS architecture. We described the several components that are involved in the proof-of-concept prototype demonstration and showed how the use of the HIDENETS hybrid architecture and its supporting services enabled us to solve the difficult problem of constructing an arbitrarily complex, temporally uncertain control application while securing fundamental safety-critical requirements.

We presented a hybrid system design using an inexpensive, real-time embedded wormhole, which nevertheless provides the necessary characteristics that allow our objectives to be fulfilled.

Finally, we described the steps of a demonstration that shows the correct operation of the application in realistic scenarios and highlights the improvements that may effectively be achieved with our approach.

Acknowledgments

We wish to warmly acknowledge the comments received from our colleagues in the HIDENETS project, in earlier stages of this work. Special acknowledgments must be given to Lorenzo Falai and Andrea Bondavalli, who proposed and defined the R&SA Clock service, to José Rufino, for his work in the definition of the TTFD service, to Mônica Dixit, for her work in the QoS Coverage service and to Henrique Moniz, for his work in the Intrusion Tolerant Agreement service.

References

[1] A. Bondavalli, A. Ceccarelli, and L. Falai. Assuring resilient time synchronization. In *Proc. of the 27th International Symposium on Reliable Distributed Systems*, pages 3–12, Napoli, Italy, oct 2008.

[2] A. Casimiro, P. Lollini, M. Dixit, A. Bondavalli, and P. Veríssimo. A framework for dependable qos adaptation in probabilistic environments. In *23rd ACM Symposium on Applied Computing, Dependable and Adaptive Distributed Systems Track*, pages 2192–2196, Fortaleza, Ceara, Brazil, Mar. 2008.

[3] J. A. et al. Revised reference model. June 2007. EU FP6 IST project HIDENETS, Deliverable D1.2.

[4] H. Moniz, N. F. Neves, M. Correia, and P. Veríssimo. Randomized intrusion-tolerant asynchronous services. In *DSN '06: Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*, pages 568–577, jun 2006.

[5] H. W. site. http://www.hidenets.aau.dk/.

[6] T. W. site. http://torcs.sourceforge.net/.

[7] P. Varaiya. Smart cars on smart roads: Problems of control. *IEEE Transactions on Automatic Control*, 38(2):195–207, 1993.

[8] P. Verissimo. Travelling through wormholes: a new look at distributed systems models. *SIGACT News*, 37(1):66–81, 2006.