Automatic Analysis of a Safety Critical Tele Control System*

Edoardo Campagnano[1], Ester Ciancamerla[1], Michele Minichino[1], and Enrico Tronci[2]

[1] ENEA CR Casaccia, Via Anguillarese, 301, S. Maria di Galeria, 00060, Roma, Italy
{ciancamerlae, minichino, edoardo.campagnano}@casaccia.enea.it
[2] Dipartimento di Informatica, Università di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Italy
tronci@di.uniroma1.it

Abstract. We show how the Murφ *model checker* can be used to automatically carry out safety analysis of a quite complex hybrid system tele-controlling vehicles traffic inside a safety critical transport infrastructure such as a long bridge or a tunnel. We present the Murφ model we developed towards this end as well as the experimental results we obtained by running the Murφ verifier on our model.

Our experimental results show that the approach presented here can be used to verify safety of critical dimensioning parameters (e.g. bandwidth) of the telecommunication network embedded in a safety critical system.

1 Introduction

Because of technological as well as economical reasons, the number of systems relying on wireless telecommunication (telco) networks is always increasing. This is also happening for *safety critical systems*. This poses new challenges to the safety analysis work. In fact, the telco network behaviour needs to be modeled in a fairly accurate way in order to formalize the relationship between telco network parameters (e.g. *bandwidth*) and the system safety property being investigated.

We show how the above is possible by presenting a case study on the analysis of a safety property for a *Tele Control System* (TCS), developed in the frame of the European project *SAFETUNNEL* [11].

The goal of TCS is to take active measures to improve safety in the *Critical Transport Infrastructure* (CTI) it controls, namely a tunnel. More specifically, TCS aims at reducing the number of accidents inside alpine road tunnels, exploiting GPRS (*General Packet Radio Service)*) communication between instrumented vehicles and a *Tunnel Control Centre* (TCC). TCS implements preventive safety functions, namely: vehicle prognostics, vehicle tunnel access control, vehicle speed and distance control, dissemination of emergency message.

* Contact Author: Michele Minichino, Tel.: +39 06 3048 3407, Fax: 06 3048 6511.

We present a model of TCS and an automatic analysis of it via model checking [12]. Our goal is to show that TCS operates in a safe way, that is no dangerous situation can arise from installation and usage of the TCS in our CTI.

More specifically, our analysis focuses on the interaction of TCS telco network dimensioning with TCS preventive safety functions. We formally check, via model checking, that the telco dimensioning, in terms of bandwidth, guarantees TCC ability to safely handle different tunnel scenarios. Namely: normal system operational mode (registrations, deregistrations, anomaly situations and emergency situations), emergency scenarios (i.e. dissemination of emergency information).

Basically, our present work is about TCS validation by modeling along the lines of [1]. In fact, in our case, only a limited number of field tests can be run on the actual system. This is because measures requiring long observation times inside the infrastructure (that has to be closed to the ordinary vehicular traffic, with loss of availability and money) should be kept to a minimum. Moreover measures which would require irreproducible infrastructure scenarios (i.e occurrence of incidents and emergency scenarios) cannot simply be done. From the above considerations stem the importance safety and performance analysis on the system model.

TCS is a quite large *hybrid system*, that is a system with continuous as well as discrete state variables. Automatic analysis of *Hybrid Systems* poses formidable challenges both from a modeling as well as from a verification point of view. In fact the simultaneous presence of continuous and discrete variables may lead very quickly to *state explosion*, thus preventing completion of the verification process.

Many verification tools (*model checkers*) are available for automatic verification of hybrid systems. Examples are: HyTech [9,3,2] and UPPAAL [10,18]. Also tools originally designed for hardware verification have been used for hybrid systems verification. E.g. in [17] SMV [12,16] has been used for verification of chemical processing systems.

In this case study we use the CMurφ [5,4] verifier since both HyTech and SMV could not complete the verification task because of state explosion. This is in agreement with our previous experience in hybrid systems verification [14].

CMurφ is the Murφ verifier [6,13] extended with (finite precision) real numbers [14], caching and disk based algorithms [15,5].

Automatic timeliness verification with the Murφ verifier and performability analysis of TCS telco network has also been studied, respectively, in [8], [7].

Our main contributions here can be summarized as follows. We sketch TCS features (Section 3), present our modeling of the TCS system (Sections 4, 4.1, 4.2, 4.3, 4.4), present a formalization of the main TCS safety requirement (Section 5) and finally give experimental results showing effectiveness of our approach (Section 6). Lack of space prevents us from giving the Murφ model of TCS.

2 Basic Notions

A *Finite State System* (FSS) $\mathcal{S}$ is a 4-tuple (S, I, A, R) where: S is a finite set (of states), $I \subseteq S$ is the set of initial states, A is a finite set (of *transition labels* or *events* or *actions*) and R is a relation on $S \times A \times S$. R is usually called the *transition relation* of $\mathcal{S}$. We define the set $\texttt{next}(s)$ of successors of state s as follows: $\texttt{next}(s) = \{s' | \exists a R(s, a, s')\}$.

The set of *reachable states* of $\mathcal{S}$ (notation $\textbf{Reach}(\mathcal{S})$) is the set of states of $\mathcal{S}$ reachable in zero or more steps from I.

A *trace* π of $\mathcal{S}$ is a finite or infinite sequence $\pi \equiv s_0, a_0, s_1, a_1, \ldots$ s.t.: $s_0 \in I$ and for $i = 0, 1, \ldots$ $R(s_i, a_i, s_{i+1})$ holds. We also write $\pi(i)$ for $s(i)$.

In the following we will always refer to a given (once and for all) system $\mathcal{S}$ = (S, I, A, R). Thus, e.g., we will write **Reach** for $\textbf{Reach}(\mathcal{S})$. Also we may speak about the set of initial states I as well as about the transition relation R without explicitly mentioning $\mathcal{S}$.

Let $\mathcal{B} = \{0, 1\}$ the set of boolean values. An *invariant* for $\mathcal{S}$ = (S, I, A, R) is a map φ from S to $\mathcal{B}$. We say that $\mathcal{S}$ satisfies invariant φ iff for all $s \in \textbf{Reach}$ $\varphi(s) = 1$. That is, if for all reachable states of $\mathcal{S}$, φ holds.

Safety properties are modeled using invariants. That is, an *error state* or an *undesired state* is a state that does not satisfy the given invariant.

Basically, using a suitable high level language, a *model checker* takes as input the definitions of an FSS $\mathcal{S}$ and of an invariant φ for $\mathcal{S}$ an returns `PASS` if $\mathcal{S}$ satisfies φ, `FAIL` otherwise. Moreover, when a model checker returns `FAIL`, it also returns a finite trace $\pi \equiv s_0, a_0, s_1, a_1, \ldots s_k$, of $\mathcal{S}$ leading to an error state, that is we have $\varphi(\pi(k)) = \varphi(s_k) = 0$.

From the above follows that, *given* a system $\mathcal{S}$ and an invariant φ, a model checker *automatically* carries out a a *reachability analysis*, i.e. the computation of all reachable states, for $\mathcal{S}$, looking for undesired states (i.e. states not satisfying invariant φ).

We plan to use CMurφ [5,4] extended with real numbers [14] to analyze hybrid systems. For this reason we model hybrid systems as *Discrete Time Systems* (DTSs). We show the easy relationship between DTSs and FSSs using a toy example. Let us consider the DTS $\mathbf{x}$ defined by Equation 1, where $\mathbf{x}(t)$ is the state value at time t and $\mathbf{d}(t)$ is the disturbance value at time t.

$$\mathbf{x}(t+1) = \begin{cases} \mathbf{x}(t) + \mathbf{d}(t) \text{ if } \mathbf{x}(t) \leq 3 \\ \mathbf{x}(t) - \mathbf{d}(t) \text{ otherwise} \end{cases} \qquad \forall t[\mathbf{d}(t) \in \{0, 1, 2\}], \qquad \mathbf{x}(0) = 0. \quad (1)$$

$$\varphi_1(\mathbf{v}) = (\mathbf{v} \leq 5) \qquad\qquad \varphi_2(\mathbf{v}) = (\mathbf{v} < 5) \qquad (2)$$

Fig. 1 shows the FSS corresponding to the DTS defined by Equation 1. The initial state $\mathbf{x}(0) = 0$ is shown with an arrow in Fig. 1, where nodes are labeled with state values and edges are labeled with action (disturbance, in our case) values.

Equation 2 defines possible invariants for system $\mathbf{x}$ in Equation 1. A model checker taking as input the pair ($\mathbf{x}$, φ_1) will return `PASS` since all reachable states of $\mathbf{x}$ are less than or equal to 5. On the other hand a model checker with input ($\mathbf{x}$, φ_2) will return `FAIL` with the following trace (counterexample) $0, 1, 1, 2, 3, 2, 5$.

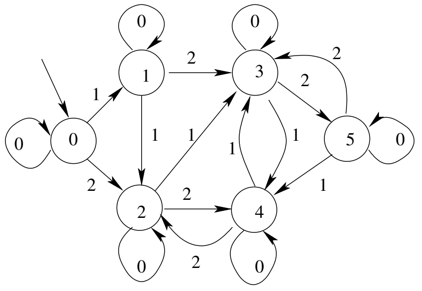

Fig. 1. FSS for the discrete time system in Equation 1

3 System Overview

In this Section we give a high level description of our TCS architecture. The remaining Sections will gradually zoom in TCS components showing how our Murφ model is organized.

The goal of TCS is to monitor and control vehicle (mainly trucks) traffic inside the CTI area. This is done by equipping each vehicle with suitable sensors and actuators (e.g. to measure and control the distance from the preceding vehicle) and with telecommunication devices (to communicate with the control center).

TCS consists of three main subsystems: *Vehicles*, *Telecommunication network* (TLC) and, finally, the *Tele Control Center* (TCC).

The *Tele Control Center* (TCC) manages the vehicles in the CTI area. The TCC-vehicle communication protocol is defined with *Message Sequence Charts* (MSCs) which also define the telecommunication network load, since they define the number of bytes traveling in the communication channels. In case of an accident the TCC sends to all vehicles in the CTI suitable directives to escape from the accident area.

This is the most stressful situation for the telecommunication network. Since our main goal here is to verify the telecommunication network dimensioning, we will just focus on the case in which, for some reason (e.g. an accident), the TCC needs to send a given (*emergency*) message to all vehicles.

As far as we are concerned, vehicles are equipped as follows: 1) Fuel level sensors, distance sensors, oil level sensors, etc; 2) GPRS telecommunication devices; 3) Automatic Cruise Control (ACC), which takes from the TCC the max speed and min distance and actuates vehicle throttle and brakes accordingly.

A vehicle equipped with the above devices is also called a *mobile station.*

For safety reasons a vehicle must be an autonomous system, i.e. it should work safely also when the TCC or the telecommunication network are not working. This is why vehicles are equipped with an *Automatic Cruise Control* (ACC) that keeps the vehicle speed below a given threshold and the distance of the vehicle from the preceding one above a given threshold.

Communication between mobile stations (vehicles) and the TCC essentially exploits the GPRS technology used to support communication between mobile stations and TCC inside the whole CTI area.

CTI GPRS network consists of a set of *Base Stations* situated inside the CTI. Each Base Station supports traffic for a certain number of *Carriers*. The number of carriers per base station depends on the type and configuration of the base station model.

Using *Time Sharing* policies each carrier, in turn, is split into 8 *Time Slots*. This is the channel used for actual data transmission. The time slot channel has a transmission speed of 10.22 kbps. Theoretically a GPRS terminal can use up to 8 time slots in *uplink* (UL) plus 8 in *downlink* (DL). Typically, commercial terminals use 6 time slots for uplink and downlink.

As an example, assuming we have a 3 carriers base station, we have available $3 * 8 = 24$ time slots for each installation.

The following alternative working hypothesis have been considered in the GPRS dimensioning: 1) The max *bit rate* (UL + DL) for each mobile station is 5 kbps, thus 2 vehicles can share one time slot; 2) The max *bit rate* (UL + DL) for each mobile station is 2 kbps, thus 5 vehicles can share one time slot. Of course the first solution gives faster communication, but requires more carriers. The second solution saves on the number of carriers, yielding however slower communication.

4 TCS Model

We use the Murφ programming language to define our model and the Murφ verification engine to check that our model meets given safety requirements. Murφ uses a Pascal-like programming language to define model dynamics. This makes the definition of complex systems quite easy, since an *object oriented* modeling approach can be followed.

Because of lack of space we cannot present the actual Murφ code of our model. We will just describe the main subsystems forming our systems as well as their interactions.

Murφ constants are our TCS model parameters. Some of our constants are suggested by [11], others have been obtained from various (e.g. physical) considerations.

Murφ data structures are our TCS model *objects* (e.g. vehicles, etc). Murφ functions are used to define the dynamics of our TCS model.

As usual we follow the convention of ending function names with `()`. Function names used in this section correspond exactly to those in the Murφ model.

We model TCS as a *discrete time system* with sampling time $T = 100$ms [11].

A high level view of TCS consists of three main objects (Figure 2). Namely, (an array of) mobile stations (i.e. vehicles), the *Telecommunication Network* (TLC), the *Tele Control Center* (TCC).

Figure 2 shows some of the (Murφ) functions (`SendRequest()`, `AssignChannel()`, `CheckBarrier()` and `BlueToothTrigger()`) implementing

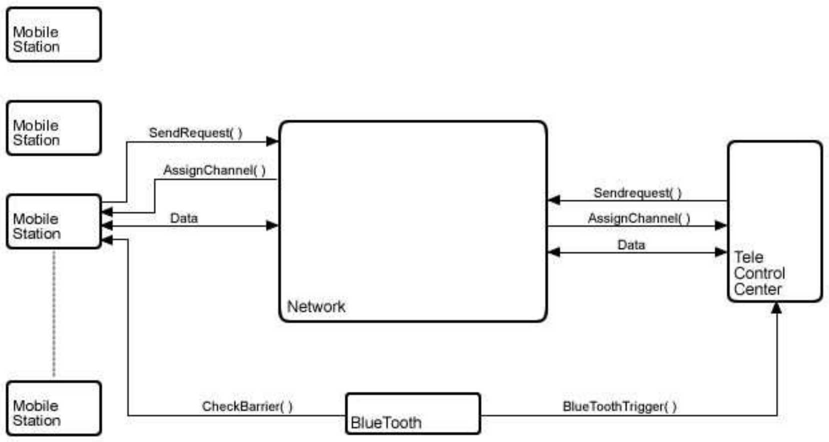

Fig. 2. Model top view

the interaction between (top) TCS objects, namely *Mobile Stations*, TCC and TLC.

Mobile Stations and TCC communicate via the TLC which consists of a GPRS network and a system of antennas used to check vehicle parameters (e.g. position) at the CTI barriers (`CheckBarrier()` in Figure 2) and possibly to send messages to the TCC (`BlueToothTrigger()` in Figure 2).

For GPRS communication a channel must be assigned to the peers. This is modeled as follows (Figure 2). The sender asks for a communication channel `SendRequest()` to the *Network Manager*. Once such channel is assigned to the sender (using `AssignChannel()`) the communication can take place. That is the sender can send its message to the receiver (`Data`).

Note that the CTI itself does not appear in Figure 2. This is because the *CTI* status does not change over time. Thus it can be simply modeled using its physical constants.

For example, constant `TUNNEL_LENGTH` defines the physical length of the CTI under consideration. Constant `APPROACHING_LENGTH` defines the distance outside CTI entrances that we still consider relevant for our modeling (*CTI area*). Constant `TOOTH_DISTANCE` gives the distance of the first Bluetooth barrier from the CTI entrance (of course, on both sides of the CTI). As a result, position (in meters) of the four CTI barriers can be easily computed. Thus, in our TCS model, to formalize the fact that a vehicle has passed a certain barrier it suffices to compare the vehicle position with the barrier position.

4.1 Vehicles

A single vehicle is modeled using a record (named `Vehicle`). Each record `Vehicle` field models a vehicle feature (e.g. `position`, `speed`, etc) needed in order to define the dynamics of our model. In other words, record `Vehicle` holds the vehicle state information.

Our CTI has one lane for each direction. Each lane is modeled with an `array` of size `NUMBER_OF_VEHICLES_PER_LANE` of vehicle records.

A vehicle can be a car or a truck. Each vehicle is equipped with suitable communication devices [11]. For this reason, in our context, vehicles are also called *mobile stations* or *terminals* (when dealing with TLC network issues).

When modeling a vehicle dynamics we also take into account its acceleration and deceleration characteristics.

4.2 The Tele Control Center

The TCC consists of four interacting subsystems: 1) Communication devices; 2) Constant directives (storing system parameters)p 3) Registered vehicle data (storing information about registered vehicles); 4) Right monitoring devices (handling the *tight monitoring* procedure to be describe din Section 4.4).

For example, among the TCC constants directives (parameters) we have `STANDARD_RECOMMENDED_SPEED` (70 Km/h) `STANDARD_RECOMMENDED_DISTANCE` (150 m). If an anomaly occurs in the monitored area TCC suitably recomputes these values.

For each vehicle v, TCC stores information about v as well as information about the messages exchanged between TCC and v.

Our model for the Tele Control Center consists of: 1) *CTI Status Variables*, storing all information (directives) to be sent to vehicles (e.g. *Recommended Speed* AND *Recommended Distance*); 2) The I/O system handling GPRS communication with the mobile stations; 3) Administrative information to make decisions about messages to be sent to vehicles.

4.3 The Telecommunication Network

The TLC network is one of the main target of our analysis. More specifically, our goal is to check that TLC dimensioning guarantees TCC ability to safely handle emergency situations. In fact, when an emergency occurs, TCS sets up a particular emergency procedure involving the TCC as well as many vehicles. This is the more demanding situation for the telco network.

Figure 3 shows our model for the telecommunication network. We view the telecommunication network as a set of (*virtual*) channels and a manager that handles virtual channel assignments and releases.

To save on the state space dimension, we only model the GPRS network and ignore other components.

In the GPRS architecture each base station can have up to 12 carriers, although typically a base station has 3 or 4 carriers. In our setting we can assume that each base station can have at least 6 carriers because of the high expected traffic volume. In the following we denote with C the number of carriers for each each base station.

Each carrier can have up to 8 time slots to be used for communication. However usually at most 6 are used. In the following we denote with N_{slots} the number of time slots for each carrier.

In the following we denote with T_{speed} the number of bits per second that a time slot can transmit. With the network configuration envisaged in [11] we have: T_{speed} = 10.22 kbps = 10465 bps.

The same time slot can be used by more than one terminal (vehicle). We denote with V_{ehic} the number of vehicles sharing the same Time Slot.

For example if the max bit rate per vehicle (UpLink + DownLink) is 5kbps we can allocate 2 vehicles on the same time slot.

We define as *Virtual Communication Channel* or just *channel* the transmission bandwidth *ideally* allocated to each terminal (vehicle). In the previous example we have 2 channels with a transmission speed of 5kbps for each slot.

We denote with B the number of base stations available.

Given the network technology (e.g. GPRS for us) the number of channels `NUMBER_OF_CHANNELS` and their speed `CHANNEL_CAPACITY` are project requirements for the network design. The following relations hold:

$$\texttt{NUMBER\_OF\_CHANNELS} = BCN_{slots}, \ \texttt{CHANNEL\_CAPACITY} = T_{speed}/V_{ehic}.$$

For example, with our data (T_{speed} = 10.22 kbps, $V_{ehic} = 5$) we have: `CHANNEL_CAPACITY` $= T_{speed}/V_{ehic} = 10465/5 = 2093$ bps.

Here we are only interested in transmission capacity. For this reason we consider channels as basic elements of our modeling.

Communication channels, of course, can be implemented with many technologies. The only difference resides in the network architecture (e.g. number of base stations, carriers, etc) needed to meet the given network specifications, `NUMBER_OF_CHANNELS`, and `CHANNEL_CAPACITY` for us.

In other words, `NUMBER_OF_CHANNELS` and `CHANNEL_CAPACITY` define the *external* view of the telecommunication network and are indeed the design parameters of the network itself. Since our goal is to study the interaction of the telecommunication network with the other TCS subsystems `NUMBER_OF_CHANNELS` and `CHANNEL_CAPACITY` are indeed a good abstraction of the network. That is, they are what the other TCS subsystems *see* of the network.

Of course the above computation of B assumes that each base station covers most of the area of our interest. This is a reasonable assumption in the case of CTI area.

Communication set up is done, once and for all, from each vehicle upon entering CTI area. This establishes a *communication link* between the terminal (vehicle) and the TCC. During this setup the vehicle sends to the TCC administrative information such as vehicle identifier, etc. When a terminal (vehicle) wants to communicate with the TCC it must look for an available channel, that is a channel not in use by another vehicle. Only once such available channel is found communication can take place. Thus each communication round is preceded by a *channel search* phase.

A terminal (vehicle) may loose its communication link with the TCC. In such cases the interaction protocol with the TCC is such that the lost link cannot be recovered. Thus if a vehicle looses its communication link, it is no more connected to the TCC.

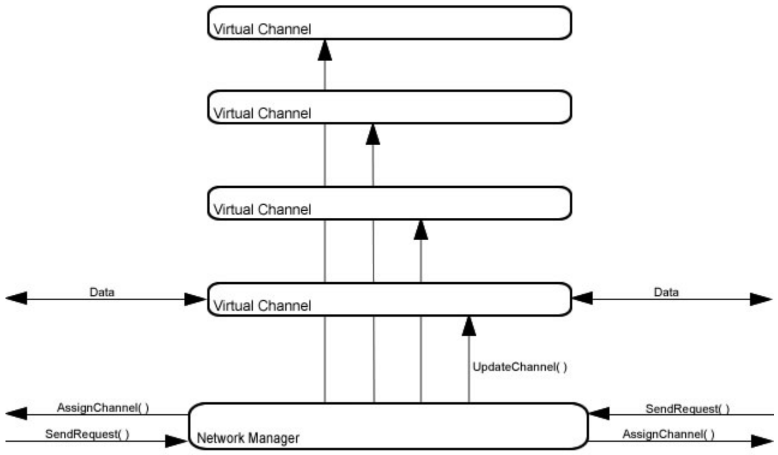

Fig. 3. Telecommunication Network

4.4 Communication Protocols

The protocols used in the TCS are defined by using *Message Sequence Charts* (MSCs). In particular we have a *Vehicle Registration Procedure* (VRP), a *Vehicle Deregistration Procedure* (VDP), a *Tele Control Application Procedure* (TAP), a *CTI Exit Procedure* (CEP), an *Emergency Procedure* (EP). To make our model working we have to model all such procedures. For space reasons, however, here we only show (Figure 4) the *Emergency Procedure* which is needed to define our safety requirement.

Many different kind of emergencies, with different severity levels, each requiring specific recovery procedures, are considered in CTI.

However, emergency ranking often requires a human intervention. This is hard (if possible at all) to model in our framework. On the other hand our goal here is to evaluate safety of the Tele Control System consisting of the TCC, the TLC network and the vehicles. For this reason we just consider the emergency situation that is more demanding for the TLC network. This happens when the TCC has to broadcast an emergency message to all vehicles in the CTI area, Figure 4.

Our goal here is to simulate an accident blocking traffic on both lanes. In this case TCC sends to all vehicles a request to stop. Thus in our model we have a procedure `SimulateAccident()` that stops (suddenly) a given vehicle at a given point in the CTI. That vehicle then sends a `DetectedAnomalyMessage` to the TCC. Such message send to the TCC the vehicle id, the kind of accident, etc.

Upon receiving the `DetectedAnomalyMessage` message the TCC, once it has determined the nature of the emergency, starts the procedure in Figure 4. More specifically, once the TCC has determined the nature of the emergency, it sends a recovery strategy using the message `ActivateRepairingPlanningRequest`

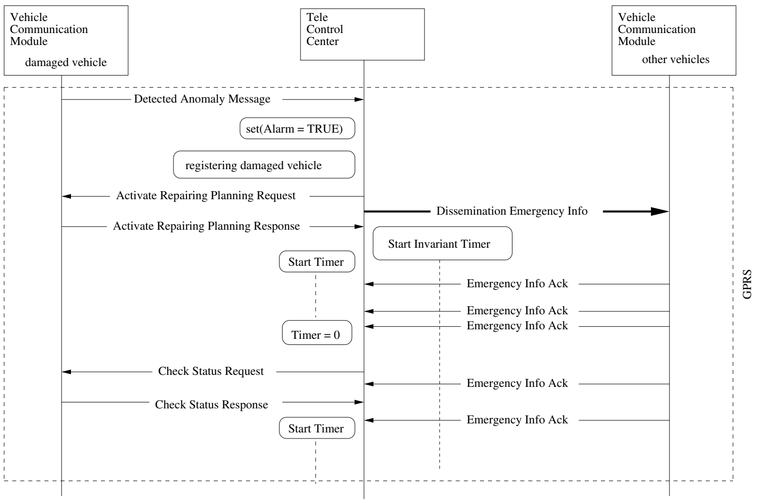

Fig. 4. Emergency Procedure

(left side of Figure 4). At the same time TCC sets to `true` the TCC alarm field and registers the vehicle involved in the emergency in order to activate a *Tight Monitoring* (TM) procedure. The TM procedure tells TCC to check the vehicles status with a higher frequency than usual. Moreover one (virtual) channel is reserved for each vehicle under tight monitoring. The mobile station (vehicle) answers the `ActivateRepairingPlanningRequest` message with a `ActivateRepairingPlanningResponse` message.

The right side of Figure 4 shows that notice of a serious emergency (incident) must be broadcasted to all vehicles in the CTI area. This, together with the ongoing TM puts a nontrivial load on the TLC network. Checking that the TLC network, under such condition, can deliver the emergency notification, to ALL vehicles in the CTI area within an assigned time constraint, is the safety requirement what we want to verify here.

Of course satisfaction of such requirement depends on the number of virtual channels available, which in turn depends on the TLC network dimensioning.

Moreover we must consider that GPRS technology, used for our TCS, does not allow *one to many* communications. Thus the broadcast needed in case of the above mentioned emergency is simulated by the TCC by sending sequentially to each vehicle in the CTI area the emergency message.

The message to be *broadcast* to all vehicles is `Dissemination_Of_Emergency_Info` and its length is 200 bytes (the longest message of all here). This message transmit an updated version of the emergency exits map, a strategy to leave the accident area as well as TCC notes (if any).

Summing up, we are going to analyze the scenario in which there is one vehicle that requires tight monitoring and TCC that broadcasts the `Dissemination_Of_Emergency_Info` message. This is the most stressful situation (assuming *single vehicle failure*) for the TLC network.

Note that, the case in which we have n channels and $(k + 1)$ vehicles requiring tight monitoring and (at the same time) broadcasting of the `Dissemination_Of_Emergency_Info` can be treated as the case in which we have $(n - k)$ channels and one vehicle requiring tight monitoring *and* broadcasting. This is because each vehicle under tight monitoring reserves a channel which is not released until the tight monitoring is over.

5 Requirements

Murφ defines requirements by using *invariants*. An invariant is a condition that all reachable states must satisfy. In other words, if a reachable state does not satisfy the given invariant we have a reachable undesired (error) state. The verifications task is to check if it is possible for the given system to reach an error state, i.e. to reach a state that does not satisfy the given invariant.

In general there are many invariants to check, one for each requirements. Here we will discuss only the main invariant for our system.

Our invariant asks that the time needed by the TCC to broadcast the `Dissemination_Of_Emergency_Info` message (Section 4.4) be below given threshold `TIME_TO_FAULT`.

The TCC, Upon receiving the `Detected_Anomaly_Message` from the vehicle:

– handles, if possible, vehicle involved in an accident;
– sends (broadcast) to all N registered vehicle a `Dissemination_Of_Emergency_Info` message;
– sets to 0 the value of our auxiliary variable `ReceivedAcks` counting the number of ack's (`Emergency_Info_Ack`) received in response to the `Dissemination_Of_Emergency_Info` message;
– initialize our timer `timer` to `TIME_TO_FAULT`.

Depending on channel availability some messages will get sent immediately, some will have to wait accordingly to the rules described in Section 4.3.

Upon receiving message `Dissemination_Of_Emergency_Info`, each mobile station will send to the TCC a `Emergency_Info_Ack` message. The TCC, in turn, increments by 1 counter `ReceivedAcks` for each `Emergency_Info_Ack` message received.

At each sampling time, variable `timer` is decremented by `SAMPLING_TIME`.

Our invariant asks that *it does not take too much to broadcast the emergency info to all vehicles in the CTI area.* That is, (`timer` $\neq 0$ or `receivedAcks` $= N$).

Using Murφ syntax this is written as follows.

```
Invariant "Too much time to deliver"
!(timer = 0.0) | receivedAcks = registeredVehicles;
```

That is, not too much time is elapsed (`!(timer = 0.0)`) or all vehicles have got the `Dissemination_Of_Emergency_Info` message (`receivedAcks =` registeredVehicles).

Of course the more virtual channel we have, the more chances we have to make our invariant true.

6 Experimental Results

In this Section we describe our verification experiments and show our experimental results.

Our invariant has been defined in Section 5.

What remains to be defined is the constant `TIME_TO_FAULT` denoting the maximum time by which all emergency messages have to be sent.

Taking `TIME_TO_FAULT` too large would make our verification uninteresting. On the other hand, taking `TIME_TO_FAULT` too small would give us *false positives* i.e. errors that indeed do not occur in the actual system.

We estimate a reasonable value for `TIME_TO_FAULT` as follows. Let v be the vehicle suggested speed inside the CTI and let d be the minimum distance among vehicles in the CTI. Suppose that a vehicle speed suddenly drops to 0 (stop). The following vehicle will bump into such stopped vehicle after

$$T_{bump} = d/v$$

Assuming (our case) that $v = 70\text{Km/h}$ and $d = 150\text{m}$, we have $T_{bump} = 7.7$ seconds. Considering some *lead time* the above calculation suggests us to set `TIME_TO_FAULT` to 5 seconds. That is we ask that within 5 seconds all vehicles in the CTI are reached by the emergency message broadcasted by the TCC.

Two parameters can (and do) lead to state explosion: the number of vehicles and nondeterminism in the inter-arrival times between vehicles.

Thus, to avoid state explosion, we scale down our model as follows.

- We limit the number of vehicles in the tunnel area.
- We set the inter-arrival time (`ENQUEUING_TIME`) to 5 seconds for 70% of all vehicles. The remaining 30% vehicles have a non deterministic interarrival time in the interval [`ENQUEUING_TIME` - 1, `ENQUEUING_TIME` + 1].

Figure 5 shows the experimental results we obtained with Murφ.

Column *Vehicles* gives the total number n of vehicles in the CTI area (namely we have $n/2$ vehicles per lane).

Column *Channels* gives the minimum number of virtual channels needed to pass verification. For example with 10 vehicles we need at least 4 channels to satisfy our invariant. If we use 3 channels our invariant fails.

Column *Rules* gives the number of rules fired by the Murφ verifier during verification.

Column *Time* gives the time (in seconds) needed to complete our verification.

Column *Reach* gives the number of reachable states.

Column *State Size* gives the number of bit used by Murφ to represent each state.

The results in Figure 5 have been obtained using Murφ 4.2 [4] with 200 MB RAM (option `-m200`) bit compression and hash compaction enabled (options `-b -c`) on a 800 MHz Pentium 3 Linux PC. Note that computation times in Figure 5 depend on the size of the set of *reachable* states (column *Reach*). The latter, in turn, depends on *both* number of vehicles and number of channels.

Of course we may use Figure 5 to dimension our TLC network. It is interesting to compare the dimensioning obtained from Figure 5 with that obtained from the approximate worst case analysis of the TLC network.

Figure 6 plots our results from Figure 5 (bottom curve) as well as the curve obtained from the TLC network dimensioning (top curve) suggested in [11]. On the x axes we have the number of vehicles in the CTI area (column *Vehicles* of Figure 5). On the y axes we have the minimum number of virtual channels that the TLC network should have (column *Channels* of Figure 5).

The exact analysis via model checking shows (Figure 6) that we may save on the virtual channel (and thus on the TLC network size) without compromising safety. In other words, our analysis allows us to estimate the *robustness* of our dimensioning, i.e. how many channel we may lose without compromising safety.

Vehicles	Channels	Rules	Time (Sec)	Reach	State Size (bits)
10	4	77942	1798	26332	2890
20	9	58312	4960	19702	4366
30	12	48477	8443	16379	6294
40	16	98730	31041	33354	8381
50	19	79100	37915	26724	10322
60	22	59470	40916	20094	12263
70	25	170875	152150	57735	14306
80	28	58730	65170	19844	16260
90	31	54085	77480	18273	18214

Fig. 5. Murphi TCS model experimental results

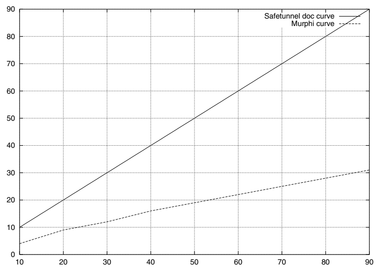

Fig. 6. Comparison between Murphi TCS model results from Figure 5 and TLC dimensioning in [11]

7 Conclusions

Our experimental results (Section 6) show that the approach presented here can be used to verify safeness of critical TLC network dimensioning parameters (namely bandwidth) as well as *robustness* w.r.t. safety of the TLC network dimensioning.

The main obstruction to be overcome is *state explosion*. Thus, in order to verify larger hybrid systems more efficient model checking algorithms are needed.

References

1. A. Bobbio, E. Ciancamerla, M.Minichino, and E. Tronci. Stochastic and functional analysis of a public mobile network in a safety critical context. In *European Safety & Reliability Conference (ESREL'05)*, June 27-30 2005.
2. A user guide to hytech: http://www.eecs.berkeley.edu/~tah/HyTech.
3. Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. Softw. Eng.*, 22(3):181–201, 1996.
4. Cached murphi web page: http://www.dsi.uniroma1.it/~tronci/cached.murphi.html.

5. G. Della Penna, B. Intrigila, I. Melatti, E. Tronci, and M. Venturini Zilli. Exploiting transition locality in automatic verification of finite state concurrent systems. *STTT*, 6(4):320–341, 2004.
6. David L. Dill, Andreas J. Drexler, Alan J. Hu, and C. Han Yang. Protocol verification as a hardware design aid. In *Proceedings of the 1991 IEEE International Conference on Computer Design on VLSI in Computer & Processors*, pages 522–525. IEEE Computer Society, 1992.
7. E. Ciancamerla and M.Minichino. Performability measures of the public mobile network of a tele control system. In *23rd International Conference on Computer Safety, Reliability and Security (SAFECOMP'04)*, volume 3219 of *Lecture Notes in Computer Science*, pages 142–154. Springer, September 21-24 2004.
8. E. Ciancamerla, M.Minichino, S. Serro, and E. Tronci. Automatic timeliness verification of a public mobile network. In *22nd International Conference on Computer Safety, Reliability and Security (SAFECOMP'03)*, volume 2788 of *Lecture Notes in Computer Science*, pages 35–48. Springer, September 23-26 2003.
9. T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1(1):110–122, dec 1997.
10. Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL: Status and developments. In Orna Grumberg, editor, *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*, volume 1254 of *Lecture Notes in Computer Science*, pages 456–459. Springer, 1997.
11. Project IST – 1999 – 28099 SAFETUNNEL. `http://www.crfproject-eu.org`.
12. Kenneth L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.
13. Murphi web page: http://sprout.stanford.edu/dill/murphi.html.
14. G. Della Penna, B. Intrigila, I. Melatti, M. Minichino, E. Ciancamerla, A. Parisse, E. Tronci, and M. V. Zilli. Automatic verification of a turbogas control system with the murφ verifier. In Oded Maler and Amir Pnueli, editors, *Hybrid Systems: Computation and Control, 6th International Workshop, HSCC 2003 Prague, Czech Republic, April 3-5, 2003, Proceedings*, volume 2623 of *Lecture Notes in Computer Science*, pages 141–155. Springer, 2003.
15. G. Della Penna, B. Intrigila, I. Melatti, E. Tronci, and M. V. Zilli. Integrating ram and disk based verification within the murφ verifier. In Daniel Geist and Enrico Tronci, editors, *Correct Hardware Design and Verification Methods, 12th IFIP WG 10.5 Advanced Research Working Conference, CHARME 2003, L'Aquila, Italy, October 21-24, 2003, Proceedings*, volume 2860 of *Lecture Notes in Computer Science*, pages 277–282. Springer, 2003.
16. Smv web page: http://www.cs.cmu.edu/∼modelcheck/.
17. A. L. Turk, S. T. Probst, and G. J. Powers. In Oded Maler, editor, *Hybrid and Real-Time Systems*, volume 1201 of *Lecture Notes in Computer Science*, pages 259–272. Springer, 1997.
18. Uppaal web page: http://www.docs.uu.se/docs/rtmv/uppaal/.