An Adaptive Failure Detection System for Vehicular Ad-hoc Networks

Eduardo Cambruzzi, Jean-Marie Farines, Raimundo Jose Macedo, Werner Kraus Jr.

Abstract— **The capacity to detect component failures is an important characteristic for fault-tolerant distributed systems. This article presents a failure detector suitable for Vehicular Ad hoc NETworks (VANETs) environments. The detector is composed of two parts: a detector for the communication link and a detector of process failure. While the link detector verifies the validity of a link between two processes, the process' failure detector adapts itself to the mobility of the vehicles and variations in the communication load in the network. Simulation results show that the detector presents a low number of false suspicions and a short detection time.**

I. Introduction

VANETs (Vehicular Ad hoc Networks) are dynamic, non-structured, self-organizing networks with asynchronous and distributed characteristics nodes (the vehicles) move at high speeds compared to other mobile ad hoc networks (MANETs). The main purpose of VANETs is to provide a medium for inter-vehicular communications allowing for inter-vehicle (V2V) and vehicle to roadside infrastructure (V2R) data exchange, with multiple applications to Intelligent Transportation Systems (ITS) [18]. For instance, a vehicle entering a freeway may automatically set its speed according to information received by surrounding vehicles.

In a VANET nodes can suddenly quit or enter the network due to their high mobility. Also, communication links among nodes may suffer from signal degradation due to obstacles, changes in vehicle densities, etc. Given these characteristics, ITS applications that rely on VANETs must be fault-tolerant. A failure detection system for VANETs is required in order to mitigate communication problems among network nodes so that decisions can be made with confidence and safety. It consists of a distributed algorithm that provides information about suspected faults in components being monitored on the network [6] so that corrective action can be performed as soon as possible.

There are a number of studies related to failure detection in MANETs (see, for instance, [3], [13], [14], [21], [15], [7]), however these are not adequate for the fast-changing configuration of VANETs. Although important for the latter, particularly when a distributed application is to be executed on a VANET scenario, the topic has not received specific attention in the literature.

In this paper, a scalable failure detection system that is suited to VANET characteristics is proposed. The main objective of this system is to provide fast and reliable information about faults so that distributed ITS applications can be executed. The accuracy and the speed of the proposed fault detector are evaluated by simulations performed in the OMNET++ *(Objective Modular Network Testbed in C++)* platform [20]. Its efficiency is assessed based on three measures: the number of false suspicions (detection of a node fault that actually did not happen), average time to detect fails that actually occurred, and average time to identify and recover from a false suspicion.

The paper is organized as follows. The communication environment in VANETs is outlined in Section II. Requirements for the implementation of failure detectors are presented in Section III together with some proposals. Sections IV and V present respectively the system model and the proposed failure detector. Simulation results appear in Section VI followed by concluding remarks in Section VII.

II. Communication in VANETs

The communication in vehicular ad hoc networks can be organized in two ways: flat and hierarchical organization. In the flat organization nodes do not occupy special functions within the network. In addition, the communication takes place through the dissemination of data, using, for example, mechanisms of flooding. Despite the simplicity of flat networks, its use in VANETs implies the loss of scalability, mainly due to the large number of transmissions and retransmissions generated during communication [22].

The hierarchical organization has nodes separated into clusters. Inside clusters, nodes assume different roles as, for example, a node can be the leader responsible for the control of communication within the cluster. Other members of the cluster can act as gateway nodes, allowing the communication between members of two clusters. The use of hierarchical communication in VANETs improves scalability, since the density changes are treated only within the cluster. In addition, it improves the use of the spectrum, offering the possibility of using different channels for communication inside the cluster and between clusters [17].

In recent years several VANET applications have used both flat and hierarchical organization in implementing services. [16] proposes a flat organization to coordinate the passage of vehicles at an intersection without traffic lights. In the proposal, vehicles approaching the intersection negotiate their way through it by exchanging messages. This negotiation occurs even before the drivers have visual contact with each other. Since each vehicle has identified its neighbors and possible conflicts to pass through the intersection, an access ordering is established so that a minimum delay takes place.

Eduardo Cambruzzi is with Federal Institute of Science and Technology of Bahia, Salvador, Brazil - email:ec@das.ufsc.br

Jean-Marie Farines and Werner Kraus Jr. are with Federal University of Santa Catarina, Florianopolis, Brazil - email:farines@das.ufsc.br, werner@das.ufsc.br

Raimundo Jose Macedo is with Federal University of Bahia, Salvador, Brazil - email:macedo@ufba.br

Another instance of flat organization is shown in [11], which describes a routing protocol for V2V communication in urban traffic environments, named “Traffic Aware Greedy Routing protocol” (Gytar). This protocol was designed to provide continuous services such as online games or Internet browsing. In GyTAR the choice of a communication route between two nodes is done dynamically based on each vehicle’s perception of other network nodes. Since this vehicle has obtained information about traffic density and distance between the source node and destination node, a route of communication between them is established. These routes can be changed as the nodes between the source and destination change their perceptions about the traffic around.

In [8], hierarchical communication is used in a proposed medium access (MAC) protocol. Initially, vehicles exchange messages so that after some time groups with one leader and member nodes are formed. The leader node is responsible for coordinating the communication of all members of its group. This coordination is done by determining a slice of time in which a member of the group has access to the physical environment. Using this mechanism allows a better use of the physical environment, as it reduces the number of collisions. Moreover, it offers improved network scalability, because communication problems are treated within each group, avoiding the constant reorganization of the network.

[1] also proposes a hierarchical organization of the network, with vehicles traveling in the same direction organized in clusters whose leaders are responsible for collecting data about the members’ state. Later, these data are sent to traffic light controllers for traffic management purposes. The formation of clusters occurs soon after nodes have identified their respective neighborhoods, ie, vehicles that are within their coverage area. This identification is made from the signaling messages sent periodically by vehicles. Once a node has information about its neighbors, for example, identifier, position, direction and speed, it calculates a weight for itself. The node with lower weight in a neighborhood is chosen leader and other vehicles traveling in the same direction are affiliated to form the cluster.

Although the proposals outlined above indicate the viability of application in VANETs, all require a stable communication environment, without latencies or at least with known latencies and without node faults as, for instance, crash-faults. However, these assumptions are at best artificial and cannot be adopted in a VANET environment. Thus, to minimize the problems brought by the VANET environment to ITS applications, these applications should be designed to be fault-tolerant with a well-defined behavior in the occurrence of a fault in order to maintain service availability to users. The next section presents some of the more recent and well known proposals for implementing failure detectors in mobile ad hoc networks that can inspire VANET counterparts.

III. Failure Detectors in Mobile Ad-hoc Networks

Failure detection is an essential service to the development of fault tolerant applications such as in group communication, replication services, and distributed consensus, and has been extensively studied in the last decade. Traditional systems for failure detection [3], [5], [15] use periodic signaling (*heartbeats*) between the processes and generally assume that these processes are in a fully-connected network. If a node (process) p does not receive a signal from another node q of the network within a certain fixed timeout, p considers q suspected of failure.

The use of fixed timeouts leads to many suspicions because it does not consider the changes in charge of communication nor the frequent link loss in mobile networks. Recently, new proposals more suitable to the environment of mobile networks have introduced the concept of adaptive timeouts [4], [19], [9].

In [4], fixed timeouts are replaced by *freshness points*. In this proposal a new time value τ_i^v is estimated for the arrival of the *i*-th signal from process v, regardless of the time when the previous one occurred. Using an estimate for the arrival of the next signaling increases the reliability and robustness of the detector, since it makes the detection time of independent of the last signaling, thus reducing the number of false suspicions generated by network delays.

An adaptive failure detector is proposed in [14]. In this proposal, the detector uses SNMP communication and artificial neural networks for training MIB patterns. These patterns are used to identify failures and adapt the timeouts, effectively estimating the arrival of the next signal.

Other proposals, such as in [19], [9], have a hierarchical concept for failure detection. The network is divided into clusters and each cluster provides a local view of the network, within which nodes identify failures by message exchanges between its members. Failures identified in the clusters are propagated through the network by inter-cluster communication channels so that a suspicious or faulty node is known by other nodes of the network.

Failure detection systems based on adaptive signaling and hierarchical organization are well suited to the environment of mobile ad hoc networks. They make failure detectors apt to avoid the two main problems of these networks: variable communication time and message loss [19]. However, failure detection systems for mobile ad hoc networks assume little variation in the network density and low mobility of nodes. Furthermore, they commonly assume that all nodes of the network are reachable from each other. These assumptions are unlikely in VANETs due mainly to the high mobility of vehicles. Thus, the current proposed failure detectors are not well suited for the environment of vehicular ad hoc networks. The following sections describe a system model and a proposed failure detector designed for the VANET environment.

IV. System Model

The system consists of a set $P = \{p_1, p_2, \dots, p_n\}$ of process in which n is unknown, but finite. Each process has a unique identifier p and represents an instrumented vehicle, which in turn represents a node in a flat network. In this model the terms node, process and vehicle are interchangeable. A vehicle is said to be *instrumented* when equipped with communication capabilities, embedded processing, GPS, and digital road maps.

Each process p has a local clock that represents time through a sequence T, in which an instant of time t_p is an element of T and t_p^i represents the p-ith signaling broadcasted by process p. Furthermore, it is possible to assume that local clocks are synchronized by GPS and that the derivation between them is not significant for the application.

Vehicles move along the same one-way track. These vehicles can communicate directly with each other through communication channels, provided they are within the coverage area of each other. The coverage area of a node p is a circle of radius r_p with p at the center. Since two nodes p and q are in a coverage area of the other, they are *neighbors* or members of a neighborhood. If two neighboring nodes p and q move away from each other beyond one's coverage area, both eliminate the other from their respective list of neighbors.

A. Communication Channels

Communication between nodes in a neighborhood takes place through communication channels using *send()* and *receive()* events. To send message m, a process p triggers an event *send(m)* and when q receives the message it triggers an event *receive(m)*. These messages are sent by broadcast and a message from a node may or may not be received by its neighbors.

It is also assumed a transmission delay $a_p(q)$ for the signaling messages received by a process p of a neighbor q. This delay is not constant and can be calculated by:

$$a_p(q) = t_p - (t_q^i + D_q), \tag{1}$$

where t_p is the moment that p receives a signaling message from q, t_q^i is the timestamp of the last signaling received from process q and D_q is the latency in seconds needed for q to send the message through the communication channel. D_q can be calculated from the function used to measure the quality of the link between two nodes in the 802.11s standard [2] as:

$$D_q = \left\lceil H + \frac{B}{C} \right\rceil, \tag{2}$$

where H is a constant overhead of the MAC layer and indicates the time in seconds to process and post the application to the physical medium; B is the message size in bits; and C is the transmission rate in bits/s of the communication channel.

In this model the communication channels are unreliable, but a message obtained through an event *receive()* is always correct. It is also defined that a process p broadcasts periodically at every Q seconds a signaling message m to all its neighbors. The fields that make up this message are shown in Table I.

As shown in Table I, each process p broadcasts periodically its unique identifier (Id_p), instantaneous velocity (v_p), current time (t_p^i), current position ($p(x, y)$) on the digital map at the moment of the signaling, and a list containing the Ids and the $timestamps$ most recently received from its neighbors (N_p).

TABLE I
Fields of a signaling message m.

Field	Description
Id_p	identifier of node p
v_p	speed of p in m/s
t_p^i	i-th timestamp sent by process p
$p(x, y)$	current position of vehicle p
N_p	list containing the *Ids* and *TimeStamps* of the last signaling broadcast from p neighbors

B. Fault model

It is assumed that a process can produce two types of faults: those caused by an equipment crash (*crash-fault*) and those caused by the vehicle exiting from the road (*exit-fault*). A process that produces a crash-fault is not able to send and receive messages and will not return to the system until its communication equipment is fixed.

An exit-fault occurs when a process (vehicle) leaves the road as, for example, when parking or entering a building. In this case, the process keeps sending or receiving messages. A process p can determine if another process q in its neigborhood is on the road by comparing the received data from its neighbors (see Table I) with the information from its digital road map. A process that is faulty due to an exit becomes part of the network whenever it returns to the road.

This model does not consider or treat malicious or Byzantine faults, in which the processes that fail continue to send messages, impairing the functioning of the system. To avoid such failures, an authorization mechanism can be used to verify the existence of network intruders, but the study of such mechanisms is not the focus of this work. The next section describes the system for detecting failures proposed in this paper.

V. The Proposed Failure Detection System

VANETs are subject to connection breakages and variations in the quality of communication between its nodes due mainly to the high mobility and sudden changes in the density of vehicles on the roads. Thus, a desirable feature for failure detectors to be used in these networks is that they adapt their timeout intervals along with the variations in communication loads as proposed in [13].

This paper proposes a failure detection system in which a monitor process p determines if a monitored process q should be considered suspect, ie, if process q may be experiencing a faulty behavior. For a monitor process to determine if a monitored process is suspicious, the monitor process uses information about the state of the monitored process received directly and indirectly. A monitor process p gets direct information about a monitored process q through the signaling that process q periodically broadcasts to its neighbors. Indirect information is obtained through the data contained in the lists N_p (see Table I) sent by the other neighbors of p along with their signals.

When a monitor process p does not receive within a given timeout a direct or indirect signal of the monitored process q, process q is considered suspicious and added to the list of suspects in the monitor process p. A suspicious process q remains on the list of suspects until the monitor process p receives a new signaling directly from q. A correct process, ie, one that does not have a fault and even so is inserted in the list of suspects is called a *false suspicious*. Such situation indicates an error of the failure detection system.

The failure detector proposed in this paper can make mistakes, adding to the list of suspects some processes that are correct. Also, it may consider faulty processes as correct. Each monitor process p has therefore a module for failure detection that adds or removes a monitored process q from the list of suspect processes using different pieces of information in order to correct itself: an adaptive *timeout* and information obtained directly or indirectly about the state of the monitored process.

A. Timeout computation

Adaptive failure detectors cope better with changes in the conditions of network communications, dynamically changing the waiting interval for a signal. In this proposal, a process p computes a timeout β_q for each of its neighbors q by:

$$\beta_q = Q + A_q + \Delta_q, \tag{3}$$

where Q is the period of signaling (heartbeat), A_q is the mean square of the delays of the n last signalings of q and Δ_q is a time in seconds that varies according to the distance between two neighboring processes.

To calculate the component A_q, each process p maintains a list L_n with the delay of the last n signals of q. From these delays, A_q can be calculated by:

$$A_q = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left[a_p(q)\right]^2}, \tag{4}$$

where n is the number of signals received from q and stored by process p in the list L_n and $a_p(q)$ is the delay between each of such signalings.

The quadratic mean is used in this proposal because it is more sensitive than the arithmetic mean to changes in average values (in this case, the delays between signal arrivals). Thus, it is possible to adaptat more rapidly the timeout with respect to variations in the communication load without disregarding the history of previous signalings.

The size of L_n affects the behavior of the detector. If it is very small, the detector will have little tolerance to small variations in network load; if it is too large, abrupt changes in communication load will be averaged out and hence do not necessarily represent the current communication state between processes. Thus, the size of L_n must be adjusted taking into account some parameters such as mobility of nodes, the time of signaling and time constraints of applications that rely on the detector accuracy.

The Δ_q component adds to the timeout a time value that increases as two neighboring nodes move away from each other. This increase in the timeout reduces the impact of signal attenuation generated by the increased distance between the vehicles and possible variations in the density of vehicles (nodes), for example, at traffic lights or speed bumps. These density variations cause the neighboring processes that do not share the same neighborhood may have very different medium access conditions. This difference in access time to the physical environment can affect communication by increasing packet loss and delays. Moreover, as two nodes move away from each other, the probability of loss of messages due to the physical attenuation of the signal also increases.

Considering these factors, Δ_q consists of a term α which defines a minimum coefficient of delay between the signals and a second term associated with the variable distance between vehicles:

$$\Delta_q = \alpha + k\frac{d(p,q)}{r_p} \tag{5}$$

where α is a calibration constant, allowing a minimum of delay in the signaling between two processes. The second term of the equation defines a ratio between the distance $d(p,q)$ of processes p and q and their communication radius r_q, so that the closer these processes are the smaller the resulting value and vice-versa. The gain k is used to ajust the relative importance of the ratio. The limits of Δ_q are defined as follows:

$$\forall(p,q)|d(p,q) = 0 \rightarrow (\Delta_q = \alpha)$$
$$\forall(p,q)|d(p,q) = r_p \rightarrow (\Delta_q = \alpha + k)$$

.

It is also defined that, for $d(p,q) > r_p$, $k = 0$, thus avoiding an indefinite increase in Δ_q.

B. Neighborhood perception

The perception that a process p has of its neighborhood depends on the signals received from its neighbors q and the information contained in the list N_q of each process q.

Thus, a node p stores information about every signal received directly from its neighbors, in addition to information from the list N_p of Ids and timestamps of the neighborhood of its neighbors. With this, a vehicle p can build an image of the network beyond its radius of communication, allowing: i) if it fails to receive a signal from a neighbor within its timeout, data contained in the list N_p can be used to avoid that p insert a neighbor q in its list of suspects; and ii) if a p suspects of a neighbor q, the time of suspicion may be reduced since p can see that q is active through messages sent by another neighbor.

Using information about processes that are at more than one hop makes the failure detection more reliable and robust in the face of mobility and delays in communication.

C. Conectivity detector

In vehicular ad hoc networks nodes can reach relative speeds in excess of 60 m/s generating a lot of broken links. Link breakages cause many applications to overload the network with control messages when attempting to reorganize the network, to restore communication routes or to rediscover lost services. In this context a mechanism to detect connectivity has two functions: i) anticipate a possible loss of connectivity between two processes and ii) prevent that the link loss be considered a failure, which would be interpreted by the application as a service outage.

Thus, to anticipate whether a link will be valid at a given moment can benefit the application in various ways such as allowing alternative communication routes between origin and destination to be found before the link is broken. Moreover, it can also work to identify real failures, in which the processes are still within the area of communication of each other but there is no communication between them. In both instances, applications can save time and increase reliability for users.

Although it is not possible to anticipate the actions of individual drivers, traffic behavior follows a set of known rules, such as vehicle traveling in the same direction along a road do not experience instantaneous changes in speed and position. Thus, it is possible to use some of these features along with information broadcast periodically by vehicles to determine whether a link between two neighboring processes remains valid.

The connectivity detector $DC_p(q)$ proposed in this paper is executed in all vehicles. Using the connectivity detector, a vehicle p estimates if a neighbor q is still within its coverage area of communication. For this end, p uses the following information: the last known speed of the neighbor q, the communication radius r_p, the last known position of both vehicles $P^p_{current}$ and P^q_{last} and the time of the last signaling t^i_q sent by q and the current time t_p at p. The following algorithm describes in a simplified manner the operation of the connectivity detector.

```
1  begin
2      $P^p_{current} \leftarrow$ current position of $p$
3      $P^q_{last} \leftarrow$ last known position of $q$
4      $P^q_{estimated} \leftarrow$ calculate the estimated position of $q$
5      if $(|P^p_{current} - P^q_{estimated}|) < r_p$ then
6          return true;
7      else
8          Process $p$ eliminates $q$ of its neighbors list;
9          return false;
10     end
11 end
```

Algorithm 1: Conectivity detector

In algorithm 1, ($P^p_{current}$) indicates the current position of p and the last position (P^q_{last}) sent by process q which was received by p (lines 2 to 3). Next the estimated position of q ($P^q_{estimated}$) is calculated using the data of the last signal that p received from q together with information from the GPS and digital road map (line 4). If the difference between the positions of p and q is less than r_p, then the link is valid, otherwise the node is removed from the list of neighbors (lines 5 to 10).

The next section presents the algorithm for failure detection in which the connectivity detector is used.

D. The Proposed Failure Detection Algorithm

The proposed failure detection system runs three parallel tasks, T1, T2 and T3. Task T1 is responsible for receiving the signals and update the data of each neighbor of the process p. In task T2, the failure detector continuously checks if a process q should be added to the list of suspects and task T3 is responsible for the recovery of false suspicions.

The simplified algorithm of the proposed failure detector is described below, and works as follows: consider two processes p and q, where p monitors q, which in turn broadcasts a message m at every Q seconds.

```
1  Task: T1
2  repeat
3      When receive(m)
4      Update data of neighbor $q$ using signaling message
5      Calculate $\beta_q$
6      $t^i_q \leftarrow$ more recently timestamp from the process $q$
7  until forever;
8  Task: T2  /* Failure detection */
9  repeat
10     When $(t_p - t^i_q) > \beta_q$: timeout expired
11     if ($q \notin$ suspect list) then
12         if ($DC_p(q)$=True) then
13             Insert $q$ in suspect list of $p$
14         end
15     end
16 until forever;
17 Task: T3 /* Failure recovery */
18 repeat
19     if (($DC_p(q)$=True) and ($q \in$ suspect list) and
       $(t_p - t^i_q) \leq \beta_q$) then
20         Remove $q$ from suspect list
21
22 until forever;
```

Algorithm 2: Failure detector

In task T1, when a process p receives a signaling message from a neighbor q, the process p updates the data about this neighbor (lines 3 to 4). Then, process p estimates for this neighbor the new timeout β_q (line 5). In line 6, the process p updates the variable t^i_q with the more recently timestamp received from process q. That is, process p updates the variable t^i_q with the timestamp received either directly or indirectly from the process q, whose value is closest to the current time t_p in the p. The *timestamps* indirectly received from the process q are stored in the list N_p, which is sent by its neighbors along with the signals (see Table I).

The generation of failure suspicions occurs in task T2. Upon starting the detection, p verifies if there is a non-suspicious neighbor q that exceeded its timeout (lines 10 and 11). When q does exceed its timeout β_q, process p runs the connectivity detector and, if the link is still valid, process p inserts q in the suspect list (lines 12 to 14). Finally in T3 a failure recovery process is executed. If a process p receives a signal from a suspicious process q, either directly or through one of its neighbors, p verifies if the signal is within the timeout β_q; if this is true, q is removed from the list of suspects (lines 19 e 20). In both T1 and T2, the connectivity detector prevents processes whose link was not considered valid to be interpreted as suspicious processes.

VI. Evaluation of the Proposed Failure Detector

This section presents implementation results for the algorithms described in previous sections in a simulation scenario. The efficiency of the proposed failure detector was measured by analyzing its performance in relation to three metrics: i) number of false suspicions, ii) average time for failure detection due to failures, and iii) average time to recover a false suspicion. A fourth metric evaluates the influence of sample size L_n used to calculate the quadratic mean delay A_q between signals in the generation of false suspicions.

The number of false suspicions is the sum of individual suspicion produced by the processes during the simulation subtracted by the number of failures that actually occurred in the neighborhood of these processes. This metric indicates the reliability of the detector in face of mobility and changes in the load of network communication.

The metrics that evaluate the average time to detect a failure and the mean time to the recovery of false suspicions indicate the speed of detection, ie, the response time of the proposed failure detector. The time for failure detection consists of an interval between the time when the fault happened and the moment when it was detected by a correct process. The time for fault recovery is the time elapsed between the insertion and withdrawal of a process from the list of suspects.

A failure detector is considered more reliable as the number of suspects indicated by it gets closer to the actual number of faulty processes in a certain time interval. The efficiency of the detector is related to its response time in terms of (i) detecting fault processes in a short time in relation to the heartbeat, and (ii) recovering a false suspicion quickly.

A. Simulation Model

Simulations were performed using the discrete event simulator OMNET++ (Objective Modular Network Testbed in C++) [20]. The communication radius is defined as $r_p = 150$. The communication standard used is the IEEE 802.11 [10] with a model of signal attenuation (path loss, fading and shadowing) implemented in the Mixim package [12]. The rate of communication and the overhead of the MAC layer are adopted respectively as C = 2 Mbps and $H = 0.01$ s.

The simulation scenario consists of a road with 4000 m in length, on which vehicles move in the same direction with speeds ranging from 10 m/s to 22 m/s, or about 36 km/h to 80 km/h. Four experiments were conducted, each with four different densities of vehicles on the road: 50, 150, 250 and 350 vehicles.

The duration of each experiment is 100 s with signaling period Q = 0.1 s. $\alpha = 0.02$ s is constant in all experiments, while the values of k were varied between 0 s and 0.06 s, so as to assess the influence of mobility on the performance of the detector. The window L_n, which stores the delay between the signals of a vehicle, has size $n = 100$ for all processes in the system. In the simulation, 20% of processes exhibit faults; these faults occur randomly throughout the simulation and a faulty process does not return to the system.

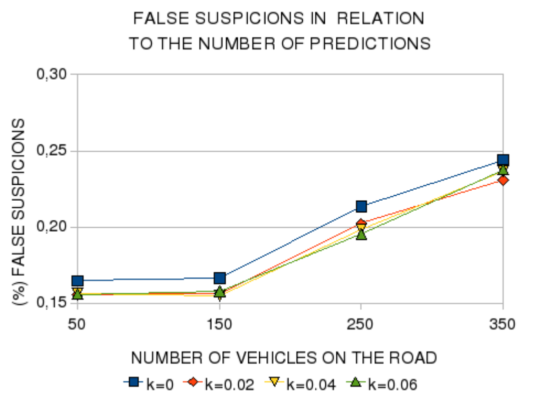
B. Experimental results

Figures 1 and 2 show results related to the reliability of the proposed fault detector.

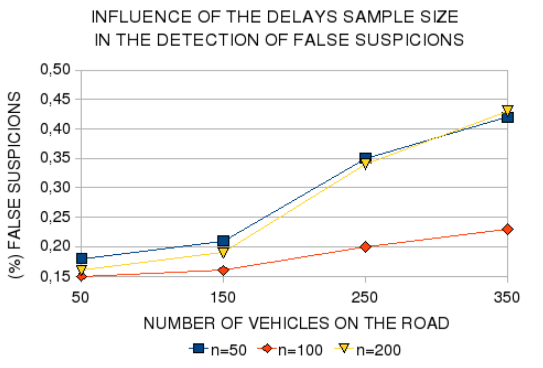
Figure 1(a) shows the percentage of false suspicions in relation to the number of predictions made by the processes throughout the simulation. Note that increasing vehicle density on the road increases the number of false suspicions. This increased density is produces an increase in the number of heartbeat signals received by each of process. However, the absolute number of false suspicions does not increase proportionally to the absolute number of signals. These results show that the proposed fault detector provides good reliability in the VANET scenario in which density variations are frequent. The stability of a fault detector in face of the density variations is a very important feature for fault detectors in VANETs, as the traffic density can vary sharply, for example, at traffic lights or other bottleneck.

Figure 1(b) shows that the sample size of the delays between signalings stored in L_n affects the performance and hence that the choice of size should be careful. Criteria for determining the sample size depend on factors such as signaling period, time constraints of the applications, network density, etc. It can be seen in this figure that i) very small samples leave the failure detector very sensitive to load changes in the communication network, and ii) very large sample sizes atenuate the sensitivity of the detector against sudden changes in load. Therefore, both cases lead to a lower reliability of the fault detector.

Figure 2(a) shows the average time to detect faults during the simulation. Again, the proposed detector shows stability in face of variation of density in the network, which is essential for the communication environment of VANETs. Note also that the increase in k does not significantly affect the average time of detection. On the other hand, while the average time of detection is not affected by the increase in k, the percentage of false suspicions decreases, as shown in Figure 1(a). That is, the increase in values of k does not reduce the response time of detector and promotes an

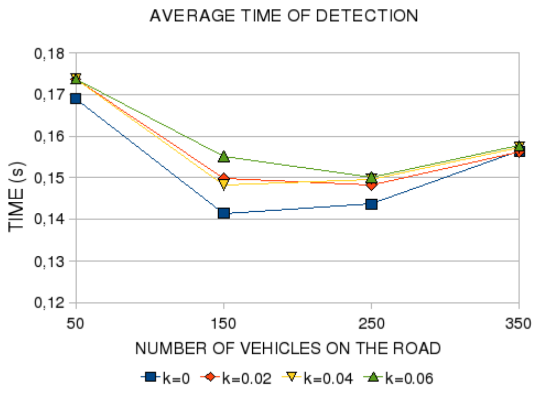


(a) *False suspicions percentage in relation to the total number of predictions.*

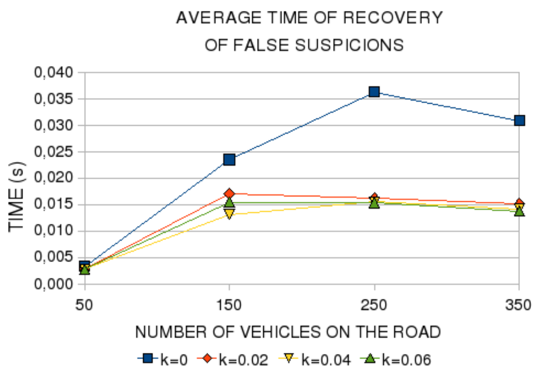


(b) *Influence of L_n size in the percentage of false suspicions.*

Fig. 1. Performace evaluation of the fault detector.



(a) *Average time for fault detection*



(b) *Average time for recovering from a false suspicion.*

Fig. 2. Response times of the fault detector.

increase in the reliability of the detector, ie, it reduces the number of false suspicions.

Figure 2(b) indicates the average recovery time of false suspicions. Note that the average time for fault recovery increases with increasing density. However, its value is relatively low when compared to the heartbeat. This is because although the density affects the load of communication and, therefore, the delay between the signals, the increase in the number of neighbors helps a monitor process to recover quickly from a false suspicion using data sent by processes being monitored (See Table I).

Observing the results shown in Figures 1 and 2, the proposed fault detector not only adapts quickly to the conditions of mobility and network density, but also has a low error in the fault detection. Thus, it shows promissing possibilitis for creating a more reliable network so that distributed applications in VANETs can operate with improved performance.

VII. Conclusion

The ability to detect faults and to reduce the number of false suspicions, adapting the fault detection system to the varying loads in network communications is vital for distributed applications with critical time constraints, especially those that use information about the vicinity of a vehicle for making their decisions. Applications such as, for example, real-time traffic light control, collision avoidance systems, and vehicle-infrastructure cooperation cannot disregard the existence of a vehicle on the road after the loss or delay of a message. Even other network applications such as P2P communication need mechanisms to detect failures that minimize the need for the reorganization of the network or the reconstruction of communication routes.

Despite its importance, current proposals for failure detectors for mobile networks are not designed for the environment of high mobility and rapid variations in network density found in VANETs. This paper proposed a fault detector adapted to the VANET environment. This detector has a mechanism of connectivity detection that attempts to minimize the impact of mobility in the process of fault detection. Simulations show that the proposed fault detector is well suited to density variations, generating a small number of false suspicions and a low time for the recovery of these false suspicions. In future work new mechanisms will be added to the fault detection so that it can understand the context of the surrounding traffic. For example, if the vehicle moves on the road with other vehicles forming a platoon, the detector can be used to form and maintain clusters of vehicles on urban roads. Such clusters are useful for gathering traffic data from individual vehicles for traffic management purposes, as in the work proposed in [1].

References

[1] E. Cambruzzi, J. M. Farines, and W. Kraus Jr., "Um Algoritmo Baseado em Peso para Formação e Manutenção de Agrupamentos em Redes Veiculares," *27o-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos-Recife-Brazil*, 2009.

[2] J. Camp and E. Knightly, "The ieee 802.11 s extended service set mesh networking standard," *IEEE Communications Magazine*, vol. 46, no. 8, pp. 120–126, 2008.

[3] T. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of the ACM (JACM)*, vol. 43, no. 2, pp. 225–267, 1996.

[4] W. Chen, S. Toueg, and M. Aguilera, "On the quality of service of failure detectors," *IEEE Transactions on computers*, pp. 561–580, 2002.

[5] D. Dolev, R. Friedman, I. Keidar, and D. Malkhi, "Failure detectors in omission failure environments," *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, vol. 16, 1997.

[6] P. Felber, X. Defago, R. Guerraoui, and P. Oser, "Failure detectors as first class objects," in *Distributed Objects and Applications, 1999. Proceedings of the International Symposium on*, 1999, pp. 132–141.

[7] R. Friedman, G. Tcharny, and I. LTD, "Evaluating failure detection in mobile ad-hoc networks," *Int. Journal of Wireless and Mobile Computing*, vol. Vol. 1, no. 8, 2005.

[8] Y. Gunter, B. Wiegel, and H. Grossmann, "Cluster-based medium access scheme for vanets," *IEEE Intelligent Transportation Systems article (ITSC)*, pp. 343–348, October 2007.

[9] G. Gupta and M. Younis, "Fault-tolerant clustering of wireless sensor networks," *Proceedings of IEEE WCNC*, vol. 3, p. 1, 2003.

[10] IEEE, "(draft) standard for information technology - telecommunications and information exchange between systems-local and metropolitan area networks - specific requirements," 2007, iEEE Computer Society, New York, USA - http://standards.ieee.org/getieee802/download/802.11-2007.pdf.

[11] M. Jerbi, S. Senouci, and Y. Ghamri-Doudane, "Towards Efficient Routing in Vehicular Ad Hoc Networks," *UBIROADS 2007 workshop, GIIS*, 2007.

[12] A. Kopke, M. Swigulski, K. Wessel, D. Willkomm, P. Haneveld, T. Parker, O. Visser, H. Lichte, and S. Valentin, "Simulating wireless and mobile networks in OMNeT++ the MiXiM vision," *Proceedings of the 1st international article on Simulation tools and techniques for communications, networks and systems & workshops*, 2008.

[13] R. Macêdo, "Failure Detection in Asynchronous Distributed Systems," *2nd Workshop on Tests and Fault-Tolerance. Curitiba-PR, Brazil*, July 2000.

[14] R. Macêdo and F. Lima, "Improving the quality of service of failure detectors with SNMP and artificial neural networks," *Anais do 22o. Simpósio Brasileiro de Redes de Computadores, Gramado, RS, Brazil*, pp. 583–586, 2004.

[15] A. Mostefaoui, E. Mourgaya, and M. Raynal, "Asynchronous implementation of failure detectors," *Proc. IEEE article on Dependable Systems and Networks*, 2003.

[16] E. O'Gorman, "Using Group Communication to Support Inter-Vehicle Coordination," 2002.

[17] T. Ohta, S. Inoue, and Y. Kakuda, "An adaptive multihop clustering scheme for highly mobile ad hoc networks," *The Sixth International Symposium on Autonomous Decentralized Systems (ISADS)*, pp. 293–300, April 2003.

[18] M. Taha and Y. Hasan, "VANET-DSRC Protocol for Reliable Broadcasting of Life Safety Messages," in *2007 IEEE International Symposium on Signal Processing and Information Technology*, 2007, pp. 104–109.

[19] A. Tai, K. Tso, and W. Sanders, "Cluster-Based Failure Detection Service for Large-Scale Ad Hoc Wireless Network Applications," *Proceedings of the 2004 International article on Dependable Systems and Networks*, 2004.

[20] A. Varga *et al.*, "The OMNeT++ discrete event simulation system," *Proceedings of the European Simulation Multiarticle*, pp. 319–324, 2001.

[21] S. Wang and S. Kuo, "Communication strategies for heartbeat-style failure detectors in wireless ad hoc networks," *Proceedings of the International article on Dependable Systems and Networks,(San Francisco, CA)*, pp. 361–370, 2003.

[22] K. Xu and M. Gerla, "A heterogeneous routing protocol based on a new stable clustering scheme," *MILCOM 2002. Proceedings*, vol. 2, pp. 838–843, Oct. 2002.