Keepalive service for VANET applications

Farah El Ali, Bertrand Ducourthial

Keepalive service for VANET applications

Farah El Ali and Bertrand Ducourthial

***Abstract*—Many applications are designed for vehicular networks, like chat, dynamic map update or touristic information. The need of services used as building blocks for these applications is then emerging: localization, map matching, data collect, and so on. When a unicast communication is required, applications rely generally on naming, localization and routing services. This approach consumes many network resources and leads to poor performances when the network dynamic is high. We then propose an efficient service, named keepalive, to ensure the continuity of a communication that began between two mobile neighbors. The major strengths of our proposal are that it relies only on local exchanges of identifiers and it is less resource consuming than usual networking services. Moreover, it is beneficial for a large set of applications, including download from road side units or follow me application for instance. Road tests and performance study confirm the efficiency of the keepalive service.**

***Index Terms*—vehicular networks, applications, services**

I. Introduction

APPLICATIONS for vehicular networks in particular, and dynamic networks in general, are expanding widely. These applications are designed for costumer services, road safety issues or traffic management. We can name the *follow me* application that helps the exchange of information between two cars moving towards the same destination, or the *RSU-download* application that ensures the data transfer between a road side unit and a car, in order to receive information.

All these applications rely on underlying *services* to be able to function properly and satisfy the needs of the users. Some of them need to retrieve information from the Controller Area Network (CAN) bus of the vehicle through a gateway to receive data from embedded sensors and calculators. Other applications require a map matching service to localize the vehicle on a map with a confidence indication. Others need to periodically collect data from the environment, and so on.

Usual networking services have been proposed too, such as naming, localization services, routing, and so on. However some of these services may consume resources when the network is highly dynamic. In some situations, the traffic road admits a reasonable dynamic (e.g., highways or country roads). This is not the case in a city with dense traffic, crossroads, traffic lights and so on. In such a situation, when two vehicles wish to maintain a communication (as in the *follow me* application), many networking resources will be used. It is the same with the *RSU-download* application when the vehicle cannot stay enough time in the range of the RSU. In these cases, an efficient dedicated services would be preferable.

The above mentioned applications – *follow me* and *RSU-download* – need to maintain the started communication between the source (first vehicle or RSU) and its destination. A keepalive service is then essential when the entities are mobile, to prevent the communication from being broken.

When applications rely on usual networking services, many control messages are generated. Let take the example of the *follow me* application and consider two instances A and B of this application running on two different vehicles. These instances are known to each other by their identifiers defined at the application level. To establish and maintain the communication, the routing layer requires network addresses. Thus, another service is called, a naming service or a localization service depending on the kind of addressing scheme. However, these addresses will change because of the mobility of the cars running A and B, and then the naming/localization service requires regular updates of its data structures (local caches or remote database). Generally such services need broadcast communications and consume bandwidth [2].

Suppose now that the address resolution is solved. The problem is then to route messages between two nodes known by their network addresses. In a dynamic network, this task is not simple and, in some situations, can generate many control messages. For instance, broadcast are generally used to repair a route or when the destination is not known precisely [17].

Hence, when an application requires to establish a route between two moving vehicles, naming/localization and routing services may lead to a large control overhead. However, some applications could perform better when relying on new services. In this paper, we propose a *keepalive service* that replaces the naming/localization and routing services for a large set of applications with the particularity to establish the communication while the nodes are neighbors. We believe that this will be the case for most of the unicast communications in dynamic networks. Therefore, our service will be beneficial for a large set of applications.

The *keepalive* service relies on cars identifiers to manage the communication and proceeds only to local exchanges of messages in the neighborhood of the concerned vehicles. Therefore, remote information and broadcast communications are not needed and applications' performances increase.

In the next section, we show that in most applications some optimizations are possible by replacing usual networking service with our keepalive service. Then in Section III, we detail our protocol. We show its interest using road tests and performance studies in Section IV. Comparison with related work and concluding remarks end the paper.

II. Keepalive Service Definition

A. Applications needs

As explained in the previous section, initiating a communication with a remote node at the application level solicits

Laboratory Heudiasyc UMR CNRS UTC 7253, Université de Technologies de Compiègne, 60200 Compiègne France.

Labex MS2T *Maîtrise des Systèmes de Systèmes Technologiques*

services such as naming or localizing that consumes resources when running in dynamic networks. Moreover, routing can overload the network in some situations.

However, vehicular networks are not a general purpose network. Applications requiring to establish a communication with a remote node will certainly use an operated network, as provided by 4G for instance. Many applications designed for the ad hoc vehicular networks will be specific. We believe that most of them will initiate unicast communication by evidence and naturally when the two protagonists are neighbors. In that case, they share a common interest, such as traffic information, touristic information, and so on.

For instance, when the *follow me* application starts, the vehicles are supposed to be neighbors, one behind the other at the beginning of the trip. In the same way, the *RSU-download* application will certainly starts when the vehicle requests it while it is in the range of the road side unit (illustrated in Figure 2). Similarly, chats (illustrated in Figure 1), images or video transfers will certain be done while the users are in interaction due to the proximity.

Fig. 1. Convoy of vehicles in Compiègne. The chat application starts a communication between two cars in the convoy and uses the keepalive service.

Fig. 2. An RSU built at UTC and used to exchange information with passing vehicles. The *RSU-download* application allows for instance to update maps.

Hence, it is possible to exploit this specificity to design an efficient *keepalive* service, offering the continuity of the unicast communication to the application layer. Such a service could only rely on identifiers exchanged in the neighborhoods instead of network addresses. Identifiers remain stable during the application (to the contrary of network addresses) and can be managed at the application layer easily.

B. Principle

The keepalive service is illustrated in Figure 3. The first step is to initiate a communication between two neighbor

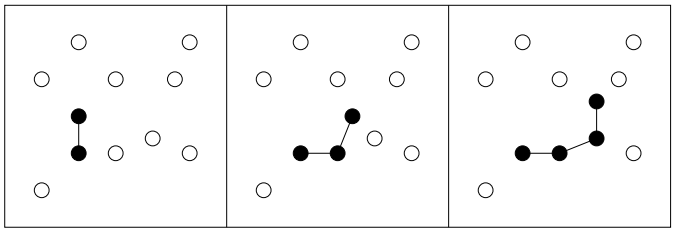

Fig. 3. When a link breaks, a neighbor is inserted to prevent the communication from being broken. Only local adjustments are required.

nodes. This can be done for instance using a local broadcast at the network layer followed by a pseudonym filter at the application layer. For some applications, cryptographic keys or peering codes can be exchanged using another channel, as it is used for peering Bluetooth devices. For instance, codes could be exchanged before starting the vehicles in the *follow me* application for securing the communication. In other applications, the license plate number could be entered after having been seen by the close users. In case the IEEE WAVE stack is used, the communication could be set using the Wave Short Message Protocol (WSMP) and a Provider Service Identifier (PSID) identifying the application [21].

Then messages can be exchanged between two instances of the service, running on neighbor nodes (Figure 3-left). If the distance between the nodes increases up to the communication range, other instances of the service running in close vehicles will propose to become relay. Note that in case there is no such nodes, then either the ad hoc network is disconnected or there is a longer path that should be discovered. In the first case, the communication is impossible. In the second case, discovering such a path will require to flood the network. Moreover the duration of the path is generally short because each of its links shares the same characteristics as the broken one. Hence, the keepalive service will only rely on close nodes.

The path will then grow (Figure 3). When the vehicles come closer to each other, useless relays must be removed.

The path adjustments are done on the basis of local information exchanged in the neighborhood of the concerned nodes only. Hence the load on the network is drastically reduced. Moreover, when messages from the application layer are sent over the path, the control information is piggybacked to the messages. When a node forwards the received message, the previous sender receives an indirect acknowledgment. The control information consists in the local knowledge about the path as well as some flags allowing to exchange states about neighbors: sender, receiver, uncertainty about its identifier and so on. It is important to note that only local information is used, avoiding overhead due to remote information update.

III. Keepalive Protocol

In this section, we describe the underlying protocol of the keepalive service. For sake of simplicity a single source-destination communication is considered. The protocol is based on three mechanisms described hereafter: local periodic messages, path extension and path reduction.

A. Periodic messages

The local periodic updates allow to deal with neighborhood changes and link failures. Each node involved in the communication sends periodically a message in its neighborhood for announcing its position in the path. In case a node does not receive this message anymore in a certain time interval, it concludes that the link is broken. These periodic sending are local, and involve only nodes of the path.

The periodic messages contain the list of nodes' identifiers belonging to the path (e.g., $u_0u_1u_2u_3u_4u_5$) along with two flags (Figure 4). The first flag (represented by symbol $>$) indicates the sender of the message along with its successor. For instance, if u_2 is the sender and u_3 is its successor in the path, the message looks like $u_0u_1u_2{>}u_3u_4u_5$, as in Figure 4. When resent by u_3 to its neighbors, the message will contain $u_0u_1u_2u_3{>}u_4u_5$. Node u_2 will also receive it and will deduce that its last message has been received by its successor. The first message between, say nodes u and v is $u{>}v$ and the acknowledgment sent by v is $uv{>}$.

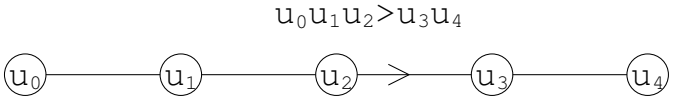

Fig. 4. Messages periodically sent by u_2 to its neighbors indicate its position in the path as well as the identity of its current predecessor and successor.

The second flag, called uncertainty flag and represented by the symbol ?, is used in case of problem regarding a node. For instance, if u_2 has a doubt on its successor u_3, its message looks like $u_0u_1u_2{>}u_3?u_4u_5$ (see Figure 5).

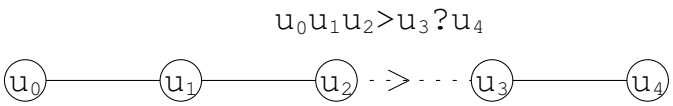

Fig. 5. The message sent by u_2 indicates that it suspects a problem on the link (u_2, u_3).

The uncertainty flag can also be used on the preceding node. Indeed, a message sent by u_3 should be received by any node in its neighborhood, including both u_2 and u_4. By sending $u_0u_1u_2?u_3{>}u_4u_5$, node u_3 informs its neighbors that it suspects u_2 to have left the neighborhood (see Figure 6).

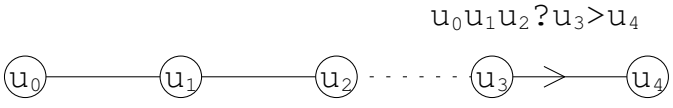

Fig. 6. The message sent by u_3 indicates that u_3 suspects a problem on the link (u_2, u_3).

All this control information is sent periodically using an adjusted timer, either by piggybacking of the application messages or in standalone messages when they are not enough application layer messages.

B. Path extension

The path extension mechanism permits to deal with link breaking, which can happen when two neighbors go far from each other. When receiving messages from both extremities of a link containing uncertainty flags on one another, the neighbor nodes can propose to become relays of the broken communication.

Consider for example Figure 4 and suppose that link (u_2, u_3) breaks. Since node u_2 sends periodically messages containing $u_2{>}u_3$ (as illustrates in Figure 4), it expects implicit acknowledgment of u_3, that is a message without uncertainty, containing $u_2u_3{>}$. Such messages indicate that u_3 well receives the message of u_2 and forwards it with the path $u_2u_3{>}$ to the next node in that path.

If node u_2 does not receive such messages from u_3, it sets the uncertainty flag on its successor u_3 in its next messages, containing $u_2{>}u_3?$ as in Figure 5. Similarly, if u_3 does not receive other messages from u_2, it will send messages containing $u_2?u_3{>}u_4$ as in Figure 6. This leads to the situation shown in Figure 7.

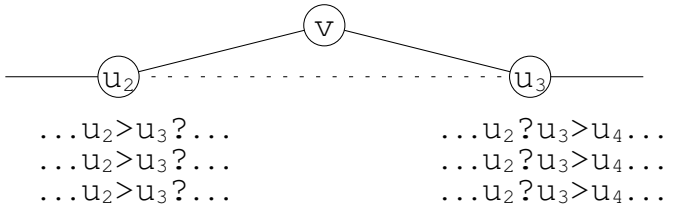

Fig. 7. Extremities of the broken link add an uncertainty flag in their periodic messages. This process is repeated a limited number of times.

If a neighbor node v notices such messages containing the uncertainty flags, it counts them. When a given threshold is reached, it proposes to be relay of the communication by sending a message containing $u_i?v{>}u_j?$.

As several neighbors of both u_2 and u_3 may propose to be relays, u_2 (closer to the source of the path) chooses the first (say v) and sends a message containing $u_2{>}vu_3?$. Then v sends $u_2v{>}u_3?$. Then u_3 sends $u_2vu_3{>}$ and the path is repaired: there are no more uncertainty flags (see Figure 8).

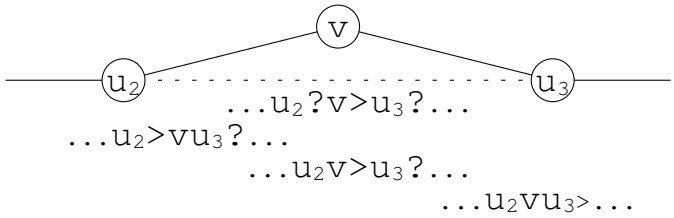

Fig. 8. A neighbor node v proposes to become a relay between u_2 and u_3. Node u_2 will confirm to a single relay. When node u_3 confirms in turns v by removing the last uncertainty flag, the path is repaired.

C. Path reduction

The third mechanism, called path reduction, consists in reducing the path whenever it is possible.

Consider the case depicted in Figure 9. By receiving messages sent by u_1 and u_4 (which contain path information), node v observes that it can reduce the path, becoming a new relay node for the path. Subsequently v proposes the reduction by placing in the path the shortcut denoted by $u_1{-}?v{-}?u_4$, with an uncertainty flag on the links (u_1, v) and (v, u_4). The message sent by v contains then $u_1{-}?v{-}?u_4u_2u_3v{>}u_4$ (Figure 9).

When u_4 receives this message, it confirms to v the possibility of reduction, removing the uncertainty flag on the reduction link (v, u_j) (message 2 in Figure 9). Then messages from v will no more contain the uncertainty on link (v, u_4). Node u_1 will consider the reduction proposed by v and will propagates it along the path by sending from this time forward messages containing $u_1 - v - u_j{>}u_2u_3u_4$. The reduction information will then be forwarded along the path by u_2 and u_3 and will eventually reach u_4. However, in case one of the nodes between u_1 and u_4 is already engaged into a reduction, it will not forward the reduction by v. This prevents any path disconnection.

When receiving the reduction proposed by v from its predecessor u_3, node u_4 knows that this reduction has been accepted by u_1 and all other nodes between u_1 and itself in the path. There is then no risk of path disconnection. Finally, node v sends a message containing $u_1vu_4{>}$, confirming to v that the reduction is effective.

By using this acceptance mechanism on all the nodes in the path between u_1 and u_4, the protocol prevents any path disconnection due to several overlapping reductions occurring simultaneously. If a node is already engaged in another reduction, it will not forward the validation message containing the reduction. Hence, any conflict between reductions is solved using a lock on the common sub-path: the winning reduction is the first that will be received by nodes of the common sub-path.

Through the same mechanism, the reduction can be performed by a node belonging to the original path. In case of conflict, priority is given to the node closer to the destination (it reaches the common sub-path first because it is itself in the common sub-path). The rest of the analysis is similar to the one shown in Figure 9.

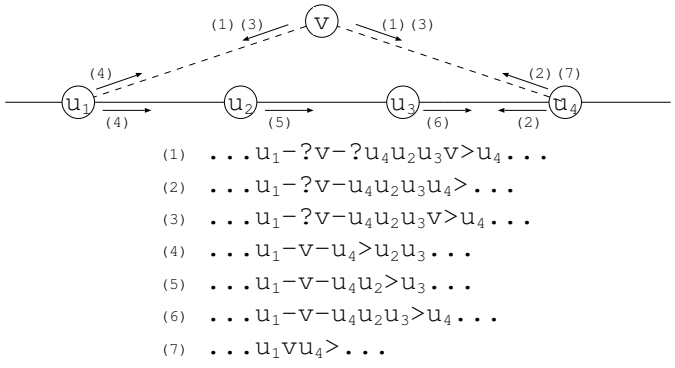

Fig. 9. Messages exchanged during the path reduction proposed by node v.

IV. Experimental validation

The protocol described in the previous section has been studied formally [7] and has been implemented using the Airplug framework [6]. The framework permits road experiments and performance studies through network emulation.

A. Airplug framework

Airplug is a light communication middleware designed for dynamic networks [6]. It is characterized by its simplicity and robustness in the messages exchange for in-vehicle and inter-vehicle communications. Any language can be used within the Airplug framework. However Tcl/Tk is generally preferred for compatibility with ns-2 [8]. The keepalive service has been implemented in Tcl/Tk as an Airplug application named PTH.

The Airplug middleware offers several modes, allowing to use a given application in different ways without further modifications:

- airplug-live: environment for real tests on vehicles or other dynamic networks (as illustrated in Figure 1) [5];
- airplug-emu: environment for emulating dynamic networks in a personal computer (PC). It permits, for instance to replay real tests while changing parameters or adding vehicles [3].
- airplug-ns: adds-on for ns-2 allowing to reuse Tcl code of the application in a large simulation [8];
- airplug-rmt: environment allowing to export an application on another computer while still being connected to the framework.

B. Proof of concept on the road

In order to test the interest of our approach as well as to validate our implementation, we designed a chat application running on Android phones connected through bluetooth to the vehicles' on-board computer. The Airplug applications running on this embedded computer are:

- the Airplug core program (APG) that manages message exchanges and network interfaces;
- the Airplug Bluetooth application (BTH) that is in charge of communicating with the users' devices;
- the Airplug Chat application (CHT) that requires the keepalive service;
- the Airplug PTH application implementing the keepalive service, used by CHT.

The CHT application can either be used on the PC or from the Android phones (illustrated in Figure 1). This is a generic application for exchanging messages between remote users and it could be used as a basis for the *follow me* application. Each message sent by CHT are relayed by the local PTH application to neighbors until the destination. In case no message is produced by CHT, PTH will produce its own to check the path. Else the control information is sent with the CHT payload. In case a link breaks, a relay node is inserted. In case useless relays are present, they are removed as explained in the previous section.

The proof of concept involved six vehicles. The embedded computers were Dell mini-9 Model DP118 under Ubuntu

v8.04 Hardy Heron, external USB GPS devices and external USB WiFi antennas Alfa AWUS036EH. The Android phones were Motorola Atrix under Android 2.3.

During the scenario, two cars started a communication while they were close to each other. Then the first car accelerates, creating the space for other cars to be inserted into the path between the first two vehicles, in order to keep their communication alive. Next the first vehicle slows down, allowing a path reduction. Two traffic conditions have been used: Compiègne down-town and high-speed road close to Compiègne.

This proof of concept confirmed that our protocol is able to maintain a communication between two vehicles in different real traffic conditions, showing its usability and interest.

C. Measure of performances

1) Method: Analytical studies validated our protocol [7] and on-road tests validated its implementation (previous section). However, in order to complete the evaluation, it is interesting to estimate the gain in terms of number of messages compared to usual networking services. As it is not possible to obtain repeatable experiments on the road (due to traffic among other parameters), we used the network emulation mode of the Airplug framework [3].

With Airplug-emu, scenarios are described in an XML file, GPS traces logged during road tests are replayed for vehicles moves and real applications are used. The Airplug-emu application manages vehicles moves and communications between applications involved in the emulation, according to the geographic positions. Figure 10 shows a large scenario (described hereafter) obtained from several real road tests.

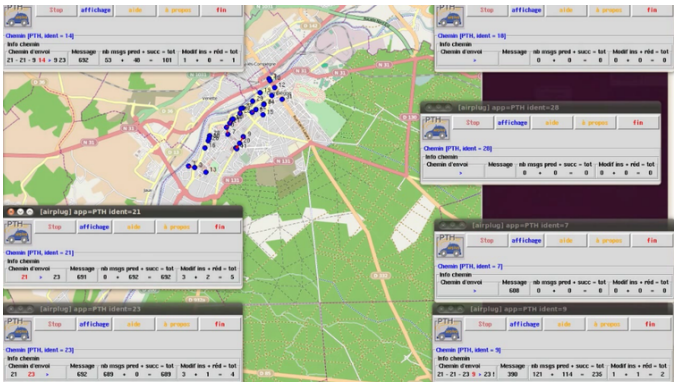

Fig. 10. A screenshot of the execution of the keepalive service using the vehicular networks emulator Airplug-emu on the down-town scenario.

2) Scenarios: Two scenarios have been used for the performance study. First, we replayed by emulation the high-speed road scenario from the proof of concept. In this scenario (named *high-speed road* scenario in the following) traffic is sparse and regular. Then, we mixed several road tests in order to obtain a realistic down-town scenario with larger density (Figure 10) and less regularity. In this scenario (named *down-town scenario* in the following), 31 vehicles are involved. The communication begins between two vehicles moving from the Research Center to the train station in Compiègne. The path involves up to four vehicles (a screenshot movie of this emulation is available on-line [1]).

We performed a comparative evaluation of PTH against a greedy geographic routing in the two scenarios: high-speed road and dense down-town. The geographic routing relies itself on a location service to know the position of the destination. When this position is known, the sender sends its message towards the geographic zone containing the destination. When the message reaches the borders of this zone, it is broadcast to reach the destination.

3) Results: We denote by N the number of generated messages on the source node. These messages are forwarded to the destination using the keepalive services. Consequently, we express the total number of messages sent on any link of the network as a function of N.

a) High-speed road scenario: In this sparse and regular network, all nodes are involved in the path. As shown in Table I, the geographic routing produces $2.3N$ messages, whereas our implementation of the keepalive service produces $2.6N$ messages. This difference is explained by the fact that the destination must acknowledge the received messages by retransmitting a message (to its predecessor). To the contrary, this retransmission is not required with the geographic routing. Hence the destination node sends less messages over the network. Finally, the geographic routing involved one more vehicle than PTH to ensure the communication (Table I).

Service	# Messages produced	# Cars involved
Geo. routing	$2.3N$	6
Keepalive	$2.6N$	5

N is the number of messages generated by the source node.

TABLE I
KEEPALIVE VS. GEOROUTING IN SPARSE AND REGULAR NETWORK

b) Down-town scenario: In this dense and irregular scenario, 31 vehicles are present with a path increasing up to length 4. The geographic routing with its location service produces $5N$ messages. On the same scenario, our implementation of the keepalive service produces only $2.4N$ messages, avoiding to load excessively the network.

Furthermore, in this scenario, all the 31 vehicles are involved in the operation of the geographic routing, whereas only 6 are used by PTH (Table II). As we can see, PTH is advantageous when it comes to dense networks, compared to other routing protocols because very few vehicles are concerned by the communication.

Service	# Messages produced	# Cars involved
Geo. routing	$5N$	31
Keepalive	$2.4N$	6

N is the number of messages generated by the source node.

TABLE II
KEEPALIVE VS. GEOROUTING IN DENSE NETWORKS

As a conclusion, our keepalive service admits similar performances compared to geographic routing in sparse networks and reduces drastically the load over the network in dense scenarios. We see that the keepalive service not only reduces the total number of messages produced in the network for the same demand by the application (here the application wishes

to send N messages), but also reduces to the minimum the number of cars involved in the transmissions.

V. Related works

Applications for mobile networks have grown in number and in variety. Some of these services concern entertainment applications, like video streaming for example [18], [15], [19], and some others concern road safety or safety critical applications, like exchanging traffic information for example. Other applications concern commercial purposes [22], or data dissemination [4]. A survey of these applications is detailed in [10]. These applications usually use the services deployed in these networks.

New infrastructures are being proposed to offer the cooperation between cars as a service and allow the sharing of many services. In [14], the authors propose a novel architecture that permits to share a set of services for free without any additional infrastructure.

In terms of safety critical applications and services, in [13], the purpose is to ensure the reliability of one hop safety critical broadcast services. For video conferencing, the aim is to be able to distribute contents to all participants. In [9], the authors propose an evaluation platform for scalable video conference service. On the other hand, in [23], the authors propose a dynamic differentiated service management to provide QoS for delivering IP applications on the IP-Broadcasting Gateway equipment.

Many services and applications use the location service. Securing these location based applications is essential. In [12], the authors study a new approach for efficient privacy preserving and improving the key update efficiency of Location Based Services. In [16], a secure location aware services using geographical secure path routing is proposed.

Many work deal with routing in vehicular networks to support the ITS services. Geographic routing relies on a location service (which generally uses flooding) to discover the position and the address of its destination. When the address is well known to the source, the relay node is the closest neighbor to the destination (greedy routing). Two surveys of protocols designed for vehicular networks are presented in [11] and [2]. In [20], the authors propose a new scheme for a stable routing protocol to support ITS Services in VANET Networks.

VI. Conclusion

Many applications are emerging in VANET. They rely on the underlying services deployed in the network. However, usual networking services such as naming, localization or routing are not always pertinent in these networks because they turn to be costly in terms of messages and resources when the network dynamic becomes high.

In this paper, we proposed a new service named *keepalive*, which is advantageous for a large set of applications, including *chat*, *follow me*, *RSU-download*, *map sharing* among others. This service relies on identifiers to manage the communication and proceeds only to local exchanges of messages in the neighborhood of the concerned vehicles. Therefore, addresses, remote information or broadcast communications are not needed and applications' performances increase.

We presented a proof of concept on road and a performance study using network emulation to replay road tests with some variants while using the same code as the one deployed on the road. The proof of concept showed the interest of such a service. The performance study proved its efficiency in terms of messages saved and vehicles involved.

In future work, we plan to organize new large road tests, to improve our prototype and to measure the performance gain offered by the keepalive service with the *RSU-download* application.

The authors wish to thank their colleagues for their help during road tests.

References

[1] PTH emulation screenshots. https://www.hds.utc.fr/airplug.

[2] M. Altayeb and I. Mahgoub. A survey of vehicular ad hoc networks routing protocols. *IJIAS*, 3(3):829–846, July 2013.

[3] A. Buisset, B. Ducourthial, F. El Ali, and S. Khalfallah. Vehicular networks emulation. In *ICCCN*, Zurich, Switzerland, 2010.

[4] M.O. Cherif, S.M. Senouci, and B. Ducourthial. Efficient data dissemination in cooperative vehicular networks. *Wireless Communications and Mobile Computing*, 13(12):1150–1160, 2013.

[5] B. Ducourthial and S. Khalfallah. A platform for road experiments. In *IEEE VTC 2009*, Barcelona, Spain.

[6] B. Ducourthial. Designing applications in dynamic networks: The Airplug Software Distribution. In *Proc. of ASCoMS*, France, 2013.

[7] F. El Ali. *Communication unicast dans les réseaux mobiles dynamiques*. PhD thesis, Université de Technologie de Compiègne, 2012.

[8] S. Khalfallah and B. Ducourthial. Bridging the gap between simulation and experimentation in vehicular networks. In *IEEE VTC Fall 2010*.

[9] T. Le and H. Nguyen. Application-aware cost function and its performance evaluation over scalable video conferencing services on heterogeneous networks. In *IEEE WCNC 2012*, pp 2185–2190, France.

[10] U. Lee, R. Cheung, and M. Gerla. *Emerging Vehicular Applications*. Taylor & Francis Group, 2008.

[11] Y. Lin, Y. Chen, and S. Lee. Routing protocols in vehicular ad hoc networks: A survey and future perspectives. *J. Inf. Sci. Eng.*, 26(3):913–932, 2010.

[12] R. Lu, X. Lin, X. Liang, and X. (Sherman) Shen. A dynamic privacy-preserving key management scheme for location-based services in VANETs. *Trans. Intell. Transport. Sys.*, 13(1):127–139, March 2012.

[13] X. Ma, J. Zhang, and T. Wu. Reliability analysis of one-hop safety-critical broadcast services in VANET. *IEEE TVT*, 60(8):3933–3946, 2011.

[14] H. Mousannif, I. Khalil, and H. Al Moatassime. Cooperation as a service in VANETs. *J.UCS*, (8):1202–1218, 2011.

[15] J.S. Park, U. Lee, S.Y Oh, and M. Gerla. Emergency related video streaming in VANET using network coding. In *ACM VANET06*, pp 102–103, 2006.

[16] V. Pathak, D. Yao, and L. Iftode. Securing location aware services over VANET using geographical secure path routing, 2008.

[17] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing, 2003.

[18] P. Piñol, O. López, M. Martínez, J. Oliver, and M.P. Malumbres. Modeling video streaming over VANETs. In *Proceedings of the 7th ACM PM2HW2N*, pages 7–14, New York, NY, USA, 2012.

[19] N. Qadri, M. Altaf, M. Fleury, M. Ghanbari, and H. Sammak. Robust video streaming over an urban VANET. In *IEEE WIMOB 2009*, pages 429–434, Washington, DC, USA.

[20] T. Taleb, E. Sakhaee, A. Jamalipour, K. Hashimoto, N. Kato, and Y. Nemoto. A stable routing protocol to support its services in VANET networks. *IEEE TVT*, 56(6):3337–3347, 2007.

[21] R. A. Uzcátegui and G. Acosta-Marum. Wave: a tutorial. *IEEE Comm. Mag.*, 47(5):126–133, May 2009.

[22] M. Watfa. *Advances in Vehicular Ad-Hoc Networks: Developments and Challenges*. Information Science Reference. 2010.

[23] J. Zhang, H. Jiang, Z. Xu, J. Li, X. Ye, and Y. Sun. Dynamic differentiated service management for ip over broadcasting network. In *IEEE WCNC 2010*, pages 1–6, Sydney, Australia.