Abstract—The diverse communication requirements of different vehicular applications and the innate dynamicity of VANET networks complicate the design of a network layer proposal adequate for all applications and network conditions. This paper highlights the necessity for adaptation of network protocols in VANETs, and describes our ongoing work on a platform (adaptation framework) that will provide VANET developers an environment where they can build network protocols that adapt based on context decisions. The design of our adaptation framework relies heavily on the separation of concerns principle by separating the adaptive protocol in subcomponents, and we model the adaptation as a combination of these subcomponents; for the network layer protocol subcomponent we define specific points where the adaptation can take place. These subcomponents will be developed and compiled independently of adaptation framework code, and linked to the framework at runtime, this thanks to the use of a plug-in platform and component-oriented programming. We describe the framework architecture and how the framework interacts with the framework users, applications and the network protocols. Lastly we briefly talk about the framework’s initial implementation and a case study that we are developing to test the adaptation framework.

Keywords-component; Adaptation; Framework; Network-Layer Protocols; VANET

I. INTRODUCTION

Currently, VANET applications range from simple exchanges of data describing the vehicle status, to highly complex, large-scale traffic management including the integration of fixed infrastructure. This diversity goes from active safety applications that are necessary life-saving applications, like local danger and emergency braking warnings, to traffic and parking applications that are less critical but can add value to a future installed VANET. In their core functionality, all these applications have one thing in common: the need for a network layer protocol to deliver relevant information (e.g., available parking space, traffic conditions, etc.) to potentially interested drivers.

It is clear that this network layer protocol must service multiple vehicular applications concurrently and be flexible enough to support the communication requirements of all these vehicular applications at the same time. Together, supporting all these potential applications will help promote the adoption of a real world VANET technology by offering a wealth of services that guarantee a good value from the initial investment of buying and installing the wireless VANET module in the vehicle. Nevertheless, for this to work, each application also needs to offer good service performance individually, in interaction with the other applications running concurrently in the same network and in different network conditions. Many other vehicular applications examples can be found in [11], in [10] and the Intelligent Transportation System (ITS) project webpage1.

Many vehicular applications require very different types of network layer protocols. For example, applications like post-crash or breakdown warnings are adequate to use geobroadcast or information dissemination protocols, because the information to transfer has to be delivered as quickly as possible to a subset of surrounding vehicles. On the contrary, applications like Internet service providing and road surface conditions to traffic authority center are not well suited to use dissemination protocols, because the destination of the message is of great importance and one-to-one communication is desired; in this case, using unicast routing network layer protocols seems more adequate. Some authors have made different categorizations of network layer protocols, for example in [10], [20] and [2]. To illustrate the difference between network layer protocols in VANETs we present in Table 1 a categorization mostly based on the categorization of [10].

It is also well known [10][7] that vehicular communication environments are characterized by highly mobile vehicles, extremely frequent topology changes, and a great variation in the number of vehicles in a certain region. These and other dynamic variations in network conditions (can also be seen as context variables) make the design of network layer protocols difficult and not generally applicable to applications that require efficiency under the dynamicity of the context variables. Schoch et al. [10] mention some key VANET network context variables (network density, node speed, node heterogeneity, movement patterns); the authors highlight the extremely opposite values that these context variables can get in the course of the network’s lifetime (e.g. Node velocity may

---

1 http://www.itsoverview.its.dot.gov/. This website provides an overview of the applications addressed by the ITS program
range from zero for stationary RSUs, or when vehicles are stuck in a traffic jam, to over 130 kmph on highways).

<table>
<thead>
<tr>
<th>Network layer protocol categories</th>
<th>Characteristics</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Beaconing</strong></td>
<td>Single hop link layer broadcast</td>
<td>Continuous update of information among all neighboring nodes</td>
</tr>
<tr>
<td></td>
<td>Unidirectional</td>
<td>Many applications have mid-range latency requirements</td>
</tr>
<tr>
<td><strong>Geocast</strong></td>
<td>Multihop dissemination within a geographical region. Attention in retaining dissemination in geographical area</td>
<td>Dissemination of information in a target area, for a specific period of time</td>
</tr>
<tr>
<td></td>
<td>Unidirectional</td>
<td>Often require very low latency, some applications may need best possible delivery success</td>
</tr>
<tr>
<td><strong>Unicast Routing</strong></td>
<td>Single-multi hop route messages towards destination (single node or destination region)</td>
<td>Priority of lower than safety apps. Some apps can tolerate high delays and retransmissions (reliability)</td>
</tr>
<tr>
<td></td>
<td>Uni- or bidirectional (connection-oriented comm.)</td>
<td>Transport data through ad hoc network to a destination (vehicle or RSU)</td>
</tr>
<tr>
<td><strong>Advance Information Dissemination</strong></td>
<td>Unidirectional</td>
<td>Widespread and time-stable dissemination of messages are more important than a low latency. Prioritization is implicitly included</td>
</tr>
<tr>
<td><strong>Carry and Forward</strong></td>
<td>Multihop routing of messages to destination. Node retains message and forward decision based on the vehicle mobility, driver navigation information</td>
<td>Dissemination of information among vehicles enduring a certain time, capable to bridge network partitions and prioritizing information</td>
</tr>
<tr>
<td></td>
<td>Unidirectional</td>
<td>Not for time sensitive applications, dependency of human factor. Great for delay tolerant apps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transport of connectionless oriented data though vehicle network in sparse conditions</td>
</tr>
</tbody>
</table>

Beside the network-specific context variables, application-specific context could also change the way the message is communicated in the network. For example the type of information (or event) to disseminate can affect the area of relevance in a dissemination protocol, also the amount of data (text, simple event, multimedia) to be transferred can greatly influence a given protocol, and lastly the QoS demands of specific messages and data types.

The network and application context variables can be modeled and implemented as an independent module whose output is the context values. These context values outputs can be offered to applications (see Context-Aware Computing [8], [5]) and can serve as input to other subsystems such as those who can take decisions to adapt the behavior of network layer protocols depending on context.

These adaptation subsystems implement what we call here an “Adaptation solution”. We define this concept as the particular way in which an adaptation is made to a process, based on certain contexts. An adaptation solution needs to respond to the following:

- What information is acquired in order to make the adaptation?
- How is the adaptation made?
- What behavior of the process does the adaptation changes?

When examining network layer protocol proposals having in mind the concepts of the adaptation solution, we can find such protocols exist in the literature, to give them a short name we call them “adaptive protocols”. Some examples of adaptive protocols are mentioned in the related work section.

However the problem with these adaptive protocols is that the adaptation solution is tightly integrated with the protocol-specific functionality, and cannot be easily separated to be reusable in other network protocols and promote the generalization of an adaptation solution independent of the network layer protocol. Also, they usually cannot be extended to use other context variables; the context variable retrieval code is also integrated in the adaptive protocol, meaning difficulty in reusing the context variable code.

Another approach can be to have an upper layer platform in which the developer can add context retrieval code modules and protocol implementations on to the platform, and in a later step combine them by an adaptation solution in order to create adaptive protocols. Allowing concurrent execution of multiples network protocols in such platform can produce adaptation solutions where there is a choice of what protocol to use depending on current context variables.

That is why the goal of this work is to design a framework for adaptation of network layer protocols in VANETs. This framework will provide VANET developers an environment where they can build an adaptation solution, and integrate it into a protocol to form an instance of an adaptive protocol.

The rest of the paper is organized as follows: section II introduces the related work and in section III we present the design of our adaptation framework separated into four general ideas. Section IV illustrates the framework architecture and how the framework interacts with the applications, protocols and the framework user. In section V, we briefly talk about the initial implementation of the framework guided by the implementation of a first case study. Finally we present a short summary and future work in section VI.

II. RELATED WORK

As we stated earlier, the proposed adaptation framework helps in the creation of adaptive protocols instances. This work started by searching and examining some adaptive protocols examples in the literature from VANET, MANETs and other networks, some of those representative examples are:

- Adler et al. [7] present a protocol that adapts the area/time of dissemination and the priority to send a message based on general network and application context.
- Delot et al. [14] propose a solution for dissemination of safety messages that adapts relaying decisions based event/vehicle mobility and direction.
• Eichler et al. [18] show a protocol that adapts the priority to send message based on the potential benefit of the message (message context).

• Bako et al. [4] proposed an adaptive gossiping (dissemination) protocol for VANET that adapts the dissemination probability to different network topologies and node densities based on position information from beacon messages.

• SHARP [21] is an ad-hoc routing protocol balanced between proactive dissemination and reactive discovery of routing information by dynamically adapting to changing network characteristics and traffic behavior.

Besides the adaptive protocol proposals, there is some related work from proposals in the spirit of the adaptation framework that we are proposing.

Peddemors et al. [3] proposed a platform to express and provide context information to mobile applications. The platform allows applications to read the context variables or act upon them by adapting their behavior. Salber et al. [8] proposed the use of a context widget to represent and provide applications access to context information. The context widget platform provides a standard subscription mechanism and a polling mechanism. Internally, a context widget consists of any number of three types of components: generators, interpreters, and servers. The previous context platforms only focus on offering context to applications and not network layer protocols as we are proposing, and the authors do not give details on how to perform the adaptation of the application behavior once the context information is obtained.

Other platforms proposals offer a way to adapt the communication system by switching network protocols or reconfiguring them from within, this is closer to the spirit of our adaptation framework.

Nundloll et al. [20] proposed a framework for runtime reconfiguration of routing protocols for VANETs. This framework encapsulates a network layer protocol in a component and also separates it in subcomponents using a Control-Forward-State (CFS) pattern. The result of adaptation is making changes at two levels (add / remove protocol components, change subcomponents CFS within the protocol). The authors also identify some common operations of network layer protocols to perform adaptation (send, receive messages, scheduling tasks, neighbor discovery). This framework for protocol reconfiguration has also been applied to other types of networks (WSN [15] and MANETs [16]). Unfortunately, adding an existing protocol implementation to this framework requires re-implementing the entire protocol code based on components, this requires more time and effort that what we are proposing (see next section). This framework proposal also does not offer a concrete solution on how to express and use the context and obtain information, and lastly only offer two methods for adaptation (change the whole protocol or reconfigure it internally).

III. FRAMEWORK DESIGN

Having stated the problem of this work, we made the design of our adaptation framework under four general ideas presented next.

A. Separate the functionality of the adaptation solution from the functionality of a network protocol

As we mentioned earlier, one of the main ideas of this proposal is to separate the functionality of an adaptation solution from the specific functionality of the network protocol. This idea relies on the separation of concerns principle, one of the key principles in software that indicates that the software must be broken so that different "concerns" or aspects of the problem at hand are resolved in separate modules or parts of the software [6]. For us, the two concerns to separate are the adaptation solution functionality and the network layer protocol functionality (e.g. messages format, message sending/receiving operations, suppression mechanism for repeated messages, topology update), and by solving them separately our framework will offer the means to combine them to form a complete solution (see Fig. 1), in our case an adaptive protocol.

![Fig. 1: a) Adaptation solutions are tightly integrated into the protocol specific functionality. b) Separation of concerns of network protocols from adaptation solutions](image)

B. Separation and modeling of the adaptation

Applying again the principle of "separation of concerns", we divided the adaptation solution into three elements: adaptation policy and mechanism [12], and adaptation metric [13]. These three adaptation elements act as follows:

- **Adaptation metric**: encapsulates the context information to be used in order to make the adaptation, here is located the context retrieval code.

- **Adaptation policy**: encapsulates the algorithm that uses the adaptation solution, which is based on the context information (metrics) to produce an adaptation as a result.

- **Adaptation mechanism**: encapsulates characteristics, elements that the adaptation solution modifies, activates or deactivates in response to information from the metrics and policy decisions.

The benefits of separating the adaptation solution into adaptation elements are:
• Facilitates the deduction of the most appropriate adaptation solution to the specific process wanting to adapt by solving these sub-problems separately and then combine them to create an adaptation solution.

• Facilitates the reuse of the elements of an adaptation solution as part of another adaptation solution.

• Facilitates the exchange of adaptation elements in the same adaptation solution in order to find the right combination for the process that we want to adapt.

The adaptation solution is modeled as a mathematical function (1), which receives N values as input, and produces R output values.

\[ F( N \text{ input values} ) \rightarrow R \text{ output values} \]  

The similarity between the relationship of the adaptation elements and a mathematical function is shown in (2). The adaptation policy takes as input a list of adaptation metrics and their values are used by the policy algorithm, the adaptation policy produces as a result a list of new values for the associated adaptation mechanisms.

\[ \text{Policy( N Metrics values )} \rightarrow R \text{ Mechanisms values} \]  

An important concept here is how the network layer protocols are integrated into the adaptation solution and elements. Network layer protocols will offer a number of adaptation mechanisms to the framework that will allow other components to adapt it (i.e. network layer protocols will contain adaptation mechanisms). This is why in the next sections we do not talk about adaptation mechanisms anymore, but instead we refer to network layer protocols as extension elements. Also, there is a special kind of adaptation mechanism that is not associated to a specific protocol, we call this mechanism “protocol selector”, and it will be provided by default by the adaptation framework implementation.

C. Use of plugIn and components to encapsulate the adaptation parts

One of the objectives of this proposal is to give the framework user a method to create various adaptation elements to add them to the platform as a first phase, and then as a second phase combine them to create adaptation solutions. There is also the problem of adding a whole network protocol implementation to the framework. Moreover, we want to promote the sharing of adaptation elements created by third party framework users (or separate team in the project) to accelerate the development of adaptation solutions by reusing already implemented elements.

However these adaptation elements and network protocols are not data or information that can be easy to integrate; they are instead pieces of code (whole existing protocol implementations, policy algorithms, context retrieval code) that are more difficult to integrate to the core framework implementation.

To integrate code blocks to the core implementation one solution is to offer the framework as source code to the user; then compile the extended customized framework with whatever language and compiler the framework was made. This solution is easier for the framework maker to develop, but transfers difficulty to the framework user as he will need to be familiar with the source code of the framework, learn how to compile the source code (most times it’s not as trivial) and to a lesser extent (depends on the choice of the language) learn the programming language used for the framework implementation; all this requires time and effort from the framework user before being able to use the framework.

Another solution is for the user to develop and compile the extendable elements (code blocks) and somehow integrate the binaries to the framework to be used by it at runtime. This solution transfers the difficulty to the framework maker, because he has to create such binary integration mechanisms; however the framework user is freed from working with the framework source code, and compiling the whole framework by itself. Also, if the framework integration mechanism is made language-independent the user can create and integrate his extendable elements in his language of choice. The user only needs to put time and effort to learn how to create the extendable elements and their interaction with the framework; this can be made easier by offering code examples of extendable elements to learn from.

Fortunately there exists in the software development community technologies that help in dealing with this predicament; these technologies are component oriented programming and plug-ins platforms. The idea is to design the interaction of the adaptation elements, network protocols and the adaptation framework under these technologies. Plug-In platforms and architectures, like the ones used in the Eclipse platform\(^2\), are used in the adaptation framework as the means for binary distribution and encapsulation of extendable elements. The concept of separation of interface from implementation (from component oriented programming) will serve to clearly define the communication between the extension elements and the framework to the user creating an extension element, and give more implementation freedom to develop extension elements. The components concept is well known in modeling languages such as UML, so we will represent the extension elements as components to ease the modeling of the framework architecture.

The relationship between plug-ins, components and the adaptation framework is described in Fig. 2. There are three types of components, the metric component, the policy component and the protocol component, each with a unique interface defined by the framework maker. The extension element is encapsulated in a component and the user implements the interface of that component type. We also define three types of plug-ins, and each type of plug-in encapsulates a component of the same type.

---

\(2\) The most prominent representative of plug-in systems, and it’s also popular nowadays in web browsers
Fig. 2: Relationship between the adaptation framework, the components and plug-ins.

The internal structure of a plug-in is defined by slots and extensions [17]. In our case one of the slots is going to be the interface of the internal component and its extension the class that implements the interface. The other slots will have meta-information about the plug-in and the extension element, such as version, author name, name or plug-in type, etc. (see Fig. 3) that will serve to identify the plug-in. There will be also specific meta-information unique to every plug-in type, e.g. for the metric plug-in a slot can be the update rate of the context, the error rate of the context, that can serve to give information about the extension element to the framework user when he is using it to create an adaptation solution.

Fig. 3: Plug-ins slot and extension structure and the adaptation framework with a metric plug-in example.

D. Definition of adaptation points in the network protocols

The object of this work is the adaptation of network layer protocols, however the question also arises: Exactly at what point in the protocol code is there a need to implement adaptation solutions?

On a first approach, the framework could be very general and flexible by supporting the addition of adaptive solutions at every point in the protocol. This however, will result in a very complex framework and will require that protocols implementations require substantial code modification to be compatible with the framework. This shortcoming is not adequate if one of our goals is to support already implemented network protocols reported in the literature without requiring substantial modification to their code.

That is why we lean towards a second approach, where we only allow certain points for adaptation inside the protocols (in our framework architecture we refer to these points as adaptation modes). Although this approach will limit flexibility to produce adaptation to the protocols, thus getting a less general proposal, we believe that finding only a set of points in the right places can result in a compromise between flexibility and complexity. We chose the following adaptation points for our framework approach:

- The send operation
- The receive operation
- The forward operation
- An all-purpose mode called update/notify

Adaptation points were deducted from adaptive protocols proposals: in [14] the authors perform adaptation from the receive network layer operation, in [7] and [14] the authors perform adaptation in the forward operation of a network layer protocol, in [18] the authors perform adaptation in the sending of the messages in the network layer protocol, lastly the work in [19] performs adaptation in another place of a network layer protocol (with a timer every T seconds).

The send, receive and forward adaptation modes are placed on top of the send, receive, and forward operations that a great majority of network layer protocols have (in accordance with the OSI standard) and are called by the application layer. The adaptation framework will instead be place at a middleware layer and using the interceptor pattern [9] intercepts these network operations before the application to perform adaptation in between (see Fig. 4).

Fig. 4: (a) shows the normal access to the network layer operations by the application layer. (b) Shows the use of the interceptor pattern by the adaptation framework for these operations.

The last adaptation mode “Update/Notify” is a mode that serves for general situations of adaptation. The idea of this mode is for the framework to allow the automatic update of a specific protocol variable by monitoring the context.
information. This will serve as a flexible mode for the protocol developer to use a custom point of adaptation in the protocol (defined by the variable to update), and especially useful for extending already implemented protocols that report that a constant can be tweaked to adapt to some context conditions. Going back to the adaptation elements concepts, the protocol variable to update will be represented in the framework by an adaptation mechanism.

Once one configures an update notify mode instance, the adaptation framework will automatically take care of updating the protocol variable (see Fig. 5), freeing the protocol from addressing the problem of calculating the new value from context variables.

![Fig. 5: Framework take care automatically of updating the protocol variable in the update/notify mode](image)

The time to update the protocol variable is configured in the framework when creating the update/notify instance, and time is independent for each instance. Each time to update interval the framework will collect context values, calculate the new value and finally notify the new value to the protocol.

The protocol implementation will need to adequate its code to offer this variable as an adaptation mechanism to the adaptation framework, forcing the protocol developer to add code to already existing implementation. However, this framework proposal aims at diminishing this extra work by offering an easy and clear way to do it thanks to the benefits of using software components.

IV. FRAMEWORK ARCHITECTURE AND UTILIZATION

In this section we illustrate the framework architecture and how the framework interacts with the users (vehicular system developers), applications and the network protocols.

A. Framework architecture

As we stated earlier, the adaptation framework is situated on a middleware layer; therefore there will be no direct communication between applications and the network protocols but through the services offered by the framework. First, the framework will register all the plug-ins that are available, then applications will view and subscribe to network protocols, view the metric plug-ins for available context variables, and with the available extension elements create the adaptation solutions by using the framework services. The framework takes charge in automatically connecting the receive, forward events from the network protocols to the applications, and the framework send command service called by application to the network layer protocols.

The architecture of the adaptation framework will support multiple applications, network layer protocols, policies and metrics (see Fig. 6). However having to many network layer protocols is not desirable, because they are complex modules and may quickly bloat the framework; finding a optimum set of network layer protocols that service all the applications communications needs is out of the scope of this work.

![Fig. 6: Adaptation Framework main architecture](image)

There is no component for encapsulating the applications nor does the framework deal with the application code, applications are represented in the framework by an applications id that they can register in the framework, this application id is analogous to the TCP port of the TCP/IP stack. These applications ids serve to redirect the services to the required application, and to associate an adaptation configuration to a specific application.

B. The adaptation configuration

An adaptation solution will be created in the framework when the user adds an "adaptation configuration". This configuration registers in the framework the combination of adaptation elements that will be forming the adaptation solution plus the mode of operation where it will be located (see Fig. 7). The adaptation configuration associates a specific adaptation element from an id number given by the framework when the element is registered.

![Fig. 7: The adaptation configuration definition](image)

The configuration contains the same elements as the definition of the function for an adaptive solution; however one new parameter is introduced: the mode of operation. This parameter tells the adaptation point (Send, Receive, Forward and Update/notify) where this adaptation solution will be associated. The adaptation configuration is associated to only
one specific application by storing this application id, and an application can have multiple adaptation configurations in various modes of operations.

C. Utilization of the framework by the user

To clarify, when we mention the framework user, we mean the developer constructing the vehicular communication system and vehicular applications with the help of the adaptation framework. The use of the adaptation framework is carried out in two stages:

- Configuration stage: This is the initial stage; it starts when the framework begins executing until the user decides to move to the next stage. In this stage the following actions are performed: add and register the adaptation elements, add and register the network protocols, register the applications, create the adaptation configurations. Some of these actions can be performed automatically, e.g., the registration of adaptation elements by reading the plug-ins from a specific directory.

- Execution stage: This stage starts when the user decides to finish the configuration stage. In this stage the following actions are performed: the protocols begin to function, the adaptation framework stays alert to make the adaptations through the execution of the added adaptation configurations when required, the framework connects the send commands from applications to the proper network protocol and the network protocols receive events to applications.

D. Example of interaction of framework and network protocols in the Runtime Stage

In order to explain how the framework does the adaptation (i.e., executes an adaptation configuration) in the executing phase, we present an example (see Fig. 8) where we adapt a protocol in the adaptation point “send”. The goal is to show the flow of operations that takes place, numbered from 1 to 4.

![Diagram](image)

Fig. 8: Example of the execution of a configuration associated to the operation mode “send”.

In this example holds other configurations put by other applications (in this case represented as the configurations X and Z). The adaptation mechanism used is the special “protocol selector mechanism” that is protocol independent and is provided by the framework.

We briefly explain the flow of operations taking place to perform the adaptation: 1) first the application i calls the send command in the framework, 2) the framework then goes into the operation mode “send” and searches a configuration associated to the application i, and finds the configuration X, 3) the framework then executes the configuration (adaptation solution) by collecting the metric values, executing the policy and getting the return value of the mechanism, 4) lastly the ‘protocol selector’ mechanism receives the resulting mechanism value and chooses the protocol k and redirects the message to the sending operation of such protocol.

V. FRAMEWORK INITIAL IMPLEMENTATION AND CASE STUDY

After defining the general architecture, we are currently implementing some aspects of the adaptation framework. Using a spiral development process, the implementation of early prototypes will allow us to continue refining the architecture details and have a more solid framework. In this section we present our ongoing work in this process.

Our current efforts on the implementation of the framework focus on the development of a plug-in platform, and the definition of the component interfaces and plug-in metadata. For the framework implementation we have chosen Java and C# .NET, mainly due to the features in the languages and environments that promote code re-use; there is a lot of information and code about creating plug-ins and they are popular languages to find code blocks from context variables and network protocols.

The binary plug-in can be implemented as a DLL file in .NET and as a JAR file in Java (more specifically an OSGi bundle platform). To identify that a binary file is a plug-in instance and to manage plug-in metadata, .NET attributes and reflection services can embed metadata directly in the DLL file; Java relays on reading meta-information defined in a manifest text file alongside the binary file inside the jar file.

Once the plug-in platform is ready, the next step is to implement some real plug-ins of adaptation elements; specifically we will implement a metric plug-in that reads location information from a GPS device and a protocol plug-in that encapsulates the OLSR network protocol. We chose to create these two plug-ins because we aim to construct our first adaptive protocol based on a first case study, where we extend an already created network protocol to make it more adaptable for VANET (improve scalability versus changes in vehicle speed).

We chose the OLSR protocol, because there is no standard network protocol in VANET just yet, but in MANETs OLSR is already a well-known protocol with a lot work around it and code availability. We also chose the OLSR protocol because the adaptation mechanism that we will use (modifying the HELLO_INTERVAL constant of OLSR) is already
documented and experimented [22] to bring improved performance if it’s dynamically changed.

VI. SUMMARY AND FUTURE WORK

This paper highlights the need for adaptation of network layer protocols in dynamic networks such as VANETS; we mention that this dynamicty comes from the diverse applications requirements and the very dynamic network variables; this motivated our proposed framework that deals with this problem. This is still ongoing work, and in this paper we described the design and initial implementation of an adaptation framework for network layer protocols in VANETs; this framework allows the creation of adaptation solutions and integrates them to the network layer protocols. For the framework design we presented four general ideas where the separation of concerns principle and the reusability of components of an adaptation solution created by third party guides our design.

As future work, beside of what we mention in the previous section and finishing up the implementation, we intend to propose other case studies for adaptation of network layer protocols (apart from the one that uses the OLSR protocol from the previous section) and use the framework to solve the adaptation problems presented in the case studies. This will exhibit even more the functionality and flexibility of the adaptation framework. We will also make a performance evaluation of the adaptation framework by evaluating memory overhead (RAM, ROM) and processing time overhead (time a message passes through the framework and route establishment delay overhead).

ACKNOWLEDGMENT

Partial funding for this work was provided by the Mexican Council for Science and Technology (CONACyT). This work is also funded by the ANR research agency (France) within the scope of the Optimacs project.

REFERENCES


