IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 14 February 2024, accepted 4 March 2024, date of publication 13 March 2024, date of current version 21 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3377141

RESEARCH ARTICLE

A Scalable Server-Side Solution for the Real-Time Handling of Road Safety Notifications

MIGUEL PEREZ-FRANCISCO[1], PABLO BORONAT[2], CARLOS T. CALAFATE[3], (Senior Member, IEEE), JUAN-CARLOS CANO[3], (Senior Member, IEEE), AND PIETRO MANZONI[3], (Senior Member, IEEE)

[1]Department of Computer Science and Engineering, Universitat Jaume I (UJI), 12071 Castelló de la Plana, Spain
[2]Department of Computer Languages and Systems, Universitat Jaume I (UJI), 12071 Castelló de la Plana, Spain
[3]Department of Computer Engineering (DISCA), Universitat Politècnica de València, 46022 Valencia, Spain

Corresponding author: Miguel Perez-Francisco (mperez@uji.es)

This work was supported by the Research and Development Project funded by MCIN/AEI/10.13039/501100011033 under Grant PID2021-122580NB-I00, and in part by European Regional Development Fund (ERDF)—A way of making Europe.

ABSTRACT Accidents are the main hurdles for using bicycles to change our transport habits. After many studies, there is no generally adopted solution. Two (non exclusive) approaches can be applied: one is based on the direct detection among vehicles, usually requiring additional hardware. The other consists of communicating through an external server which sends alerts to the concerned vehicles in real-time. In the latter case, if smartphones are used as the only instrumentation, the adoption of the system could be straightforward. In a previous work we validated the usage of conventional smartphones to create the client-side of a warning system. Instead, in the current work we address the server part. Such a server has to meet several requirements, such as being scalable (a matter not previously addressed), and able to meet real-time constraints. To achieve our purpose, we first provide the algorithms needed to ensure scalability. The system is composed by a dynamic pool of region servers, which controls a defined geographic area. Then, we implemented a functional prototype of such servers; its performance has been tested under realistic conditions to find the saturation point, after which real-time constraints are no longer guaranteed. Finally, the saturation point has been tested along with different traffic densities. Results show that the region server is able to track up to 15,000 simultaneous users, while the best we have found in published results are less than 1,400 users in simulated scenarios.

INDEX TERMS Location based services, massive real-time applications, road safety, sustainable mobility.

I. INTRODUCTION

In order to reduce the ecological footprint, together with the search for a healthier life, people are changing transport habits, and the use of bicycles or other light vehicles such as scooters is increasing. For instance, a city such as Copenhagen is expected to reach half of the displacements with these vehicles, and in general it could be said that we are living a *bicycle renaissance* [1]. However, the coexistence with cars is not always easy. In urban areas, authorities are making clear efforts to adapt road layouts, but still the problem exits in crossings and junctions, or parts in which all type of vehicles share the road [2]. Cycling is a very popular sport, but up to this time there are many accidents with serious injuries and deceases in secondary roads produced mainly to the difference of speeds and weight between bicycles and cars. In [3] it is stated than, in 2020, 47% of the fatalities in severe accidents correspond to vulnerable road users (VRUs), and this figure increases up to 70% in urban areas.

Warning or alarm systems could help to alleviate this problem. Different projects and studies have been proposed during the past decade, but still a general solution has not been adopted [4].

In the literature, two different approaches for a traffic alarm system can be differentiated regarding how vehicles interact. One set of proposals is based on direct communications or detection between vehicles. These projects usually need extra hardware to be installed in vehicles such as sensors

The associate editor coordinating the review of this manuscript and approving it for publication was Jie Gao.

or wireless communication equipment, with the addition of batteries in the case of vehicles without electrical power. This is a major inconvenient for any technology to be massively adopted.

The other kind of solutions are based on sending data to an external server, which can analyze the received data, and send alarms to the concerned vehicles. Obviously the weak point of these solutions is the response time. It must be said that the two types of solutions are non exclusive, and they can (and even will) be mixed [5].

Some of the projects based on an external server propose using smartphones as the unique instrumentation, given that they are almost ubiquitous. Thus, the adoption of such systems would be straightforward [4]. In fact, smartphones have several wireless communication devices (Bluetooth, WiFi, and the cellular phone network), and a GPS interface providing acceptable accuracy [6].

In a previous work [7] we validated the use of smartphones communicating to a centralized server over the Internet when relying on a 4G phone network for our traffic alarm system. In the system, users send position messages to the server, and they choose to receive proximity warnings of the different type of users or vehicles. Typically, cars want to know if VRUs are close to them, but VRUs just want to warn about their presence. The distance to raise warnings was fixed to 150 m since, taking into account all times involved in the system, drivers should still be able to stop their vehicles before a possible collision.

In that work we tested the communications response time, the accuracy of the smartphone's GPS, and the coverage of the 4G cellular network for urban and inter-urban routes. The system reacted with success in different traffic scenarios and speeds, allowing the driver enough time to pay attention to VRUs. We tested relative speeds until 90 km/h, obtaining communication response times of about 0.01 seconds. In this paper we now center our attention on the server part of the alarm system.

The server part has severe restrictions. In particular, it has to be able to track the position of a great number of users, to detect alarms and send them to the appropriate users, and all tasks, including communications, have to be performed with response times lower than one second. To validate this part of the traffic alarm service, we provide measures of three square regions with sides 170, 17 and 3 kms, and with different traffic densities, increasing up to 20,000 simultaneous synthetic users. Finally, we propose an algorithm to scale-up the server by taking advantage of the fact that users are distributed geographically. This former part, as far as we know, has neither been addressed in previously published works. Yet, the ability to scale-up the server system is crucial given that, if saturation occurs, real-time performance would be compromised. Consequently, a load balancing technique is ineluctable.

To the best of our knowledge, there are no works that comprehensively address such extensive areas (tens of square kilometers) while simultaneously involving thousands of users. Many studies typically conduct tests with a small number of users, and only a few manage to work with several hundreds of users at most.

The rest of the paper is organized as follows: the following section provides an overview of relevant research works on this topic. Then, in Section III, we detail the proposed system architecture. Section IV details the structure and algorithms for the server system. Afterward, in Section V, we analyze the capacity of the server. Finally, Section VI is devoted to present some conclusions and future developments.

II. RELATED WORK

Several projects are dedicated to VRU alarm systems, given the number and severity of accidents in which these road users are involved. These projects could be classified into those which are based on the detection and communication between vehicles, and those which communicate through an external server, even if both techniques are not exclusive, as pointed out in [8].

A detailed review of the state-of-the-art technologies implemented in bicycles to enhance the safety of cyclists can be found in [9]. They provide a foundation for establishing a common language to be used in future research, aiming to prevent confusion among the various capabilities and levels of smart bicycles. According to the authors, advanced technologies are rarely employed in bicycles, and most systems are primarily based on smartphone functionality.

Many works which are based on direct detection use Dedicated Short Range Communication (DSRC) for collision warning systems (e.g. [10] developed an interface to connect bicycles to the vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) systems to deliver warning messages in hazardous situations). In [11] a platoon-based cyclists cooperative system is proposed, where the bikes communicate with each other using Xbee wireless connectivity modules. This allows cyclists to respond to a Cooperative Adaptive Cruise Control delivered through dedicated human-machine interfaces.

Other examples of direct interaction among vehicles and VRU are [12], [13], and [14]. However, these approaches usually need extra instrumentation, and they are too dependent on the detection and communication conditions, such as good line of sight, absence of obstacles for wireless beacons, or the time to establish a wireless connection. On the other hand, these solutions can scale-up easily thanks to their distributed nature, and they could have a reduced response time, conditioned to the margin of time since the conflict detection took place.

Other projects, including ours, propose to use smartphones as the basic instrumentation for both vehicles and VRUs, with the benefit of the massive usage of these devices. In this group of solutions, smartphones are used to determine and send the position to an external server, which then detects possible conflicts, and sends warnings to the concerned users [4], [6], [15]. In [16] a mere explanation of what the server should do is presented. Other authors like [17], [18], [19], [20], [21], and [22] perform experiments with one or two cars, and one or two pedestrians/bikers. In [23], simulations using ns2 are presented where a maximum of 400 cars are used with a maximum of 1000 pedestrians; yet, the authors have not conducted tests using a real implementation of the server.

The shared goal of all these studies is to validate the communications and the user part of the system, as we already did in [7]. Nevertheless, all of them fail to study the scalability of the system, which would be a main bottleneck in a real implementation, where thousands, or even tens of thousands of requests per second are potentially received.

Regarding the related work devoted to traffic warnings based on an external server, Table 1 summarizes the communication technology, the type of server, the number of users, and the type of implementation used in the tests made by different authors. It can be seen that most works provide a proof of concept with less than ten of user, but they do not address the problem of scalability. In our work, we provide algorithms to scale the system in order to cope with dynamic load. In addition, we have developed a functional implementation of the server, and it has been tested to detect the saturation load (which depends on the traffic density), arriving to serve up to 15.000 simultaneous users, respecting the real-time constraints.

TABLE 1. Summary of related works based on cooperative safety systems using external servers. *Ref* stands for reference, *Communications* is the type of communications system, *Server* is the type of server, *Users* is the number of users in their tests, and *Implementation* is the type of development.

Ref	Communications	Server	Users	Implementation
[4]	Cell phone 4G	Cloud server	2	Real prototype
[6]	Cell phone 3G	Cloud server	2	Real prototype
[15]	WiFi 802.11	Future work	3	Real prototype
[16]	Cell phone 3G, WiFi 802.11b	Cloud server	N/A	N/A
[17]	Cell phone 3G, WiFi 802.11b	Cloud server	3	Simulations and prototype
[18]	Cell phone 4G	Cloud server	2	Simulations
[19]	Cell phone 4G	Cloud server	2	Real prototype
[20]	Cell phone 4G, WiFi 802.11	Cloud server	2	Real prototype
[21]	Cell phone 4G	Cloud server	2	Real prototype
[22]	Cell phone 4G, WiFi 802.11	Cloud server	12	Simulations and prototype
[23]	Cell phone 4G, WiFi 802.11p	Road Side Units and cloud server	1400	Simulations

In [7] we proved that such a solution is feasible for a non-critical warning system communicating through the 4G cellular network. In the present paper we prove that an implementation of the server part of the traffic alarm system can be devised so as to support thousands of users whereas respecting the required real-time constraints. In addition, the required algorithms to scale-up the system according to the amount of users are provided. Even though there are several similar projects, we have not found references focusing on this essential part. Consequently, we have extended our research to real-time massive applications, and we found two scopes which could be of interest: the Internet of Things and Massive Multiplayer Online Games.

In general, the lack of studies focused on the server side for real-time applications in the field of the *Internet of things* (IoT) is a known issue [24]. In that paper, the authors analyse the state-of-art of data center networks and existing platforms. These platforms, for the time being, are devoted to web analysis, which have different requirements than real-time IoT applications. In our case, the analysis network will be of interest when implementing the system as a *global service*, in which the load is dynamically shared among a cloud of *region servers*, each one taking care of a geographic region, and having to cooperate with immediate neighbors.

Works such as [25], and [26] propose the reduction, filtering or pre-processing of data generated by sensors in order to further treat them in databases, also in the context of IoT. In the case of real-time applications, a cloud of edge computing nodes between the sensor cloud and the computing cloud could react faster whereas processing sensing data, and taking into account predictive information sent by the computing cloud. The case of our traffic warning system is different, as individual messages from vehicles cannot be accumulated or reduced; notice that we do not consider direct communication between vehicles given the requirements and cost to establish these links.

As mentioned previously, another related field could be that of Massive Multiplayer Online Games (MMOGs). MMOGs usually have millions of registered players, and they have to deal with several thousands of them concurrently.

Most of MMOGs servers achieve scalability by splitting their virtual world into smaller worlds, which are then managed by different servers. The main problem is that there are overlapped areas as the players move around the world. Different approaches have been developed to address this issue by balancing the load among servers. Reference [27] summarizes the two main solutions, where one of them consists of using *shards*. The idea is having several copies of the same world (called shards or instances), where each one hosts just a subset of the players. Shards do not communicate and, therefore, users on one instance cannot interact with users on another instance. This workaround is only useful for games or services that can integrate these constraints of limited interaction. This is the solution used in the well known *World of Warcraft* game [27].

The other solution, explained in [27], consists in the partitioning of the territory in contiguous non-overlapping zones, where each zone server receives the position updates of the objects within its zone, and informs players about the modifications occurring in their proximity. The partitioning can be done statically, which is easy to implement, although it is inefficient because several zone servers may be underloaded most of the time just waiting for players to join. For example, in the *Second Life* game, most regions are empty, 30% are never visited in a six day period, and only 1% are overloaded [28].

To mitigate this load imbalance, some works like [29] use dynamic space partitioning to adapt the allocation of resources according to player distribution and density fluctuations. The space is partitioned into a large amount of smaller parts called microcells. A computer node of the system may take care of one or several at a time, dynamically redistributing these microcells on the processing nodes. This approach has the drawback of a high cost for migrating entire zones between server nodes. In fact, what is actually made dynamically is the load balance among server nodes, and not the partition of space. In our proposal we apply dynamic load balance at two levels: region servers could be divided or fused depending on the load of the region server, and new server regions can be created (on demand) in the underloaded server nodes. A mixed solution is explained in [30], where also the behavior and distribution of players over time is depicted.

In [31] a technique to integrate dynamic partitioning into the *OpenSimulator* framework is presented. The system can take dynamic decisions to add additional resources, and reallocate regions as load (players in regions) increases. They presented the implementation of a scalable method comprising both an expansion and a contraction model.

For most of the cited works about MMOGs, the number of players is used as the server load index, and the players distribution decision is based on this index. In [32] a load balancing scheme for distributed MMOG servers is proposed taking into account not the number of players in a region, but the use of the upload bandwidth of the server nodes. The goal is to reduce the inter-server communications overhead. They also considered issues such as the quadratic growth of the traffic when the players are close, and the overhead due to the interaction of players allocated in different servers.

As a conclusion, considering MMOGs, there are similarities with the proposed real-time system. Some load balancing techniques used in these game servers could be applied in our case. However, a common practice is to replicate regions with different sets of gamers, and obviously this is not applicable in traffic applications, in which users are sharing a same *physical world*.

In the case of real-time IoT applications, few works are devoted to the server (or analysis) part. In this context, a common proposal consists in pre-processing sensing data in order to reduce the amount of messages being analyzed in the server part. This is not applicable in the traffic context, as all position messages must be treated independently. Also, some works use Apache Kafka to improve scalability [33]. In our system we have reduced communication overhead to a minimum, programming the application directly over UDP protocol, as it is explained in [7].

III. SYSTEM ARCHITECTURE

In this section we provide an overview of the entire system architecture. Our proposed solution is composed by vehicles

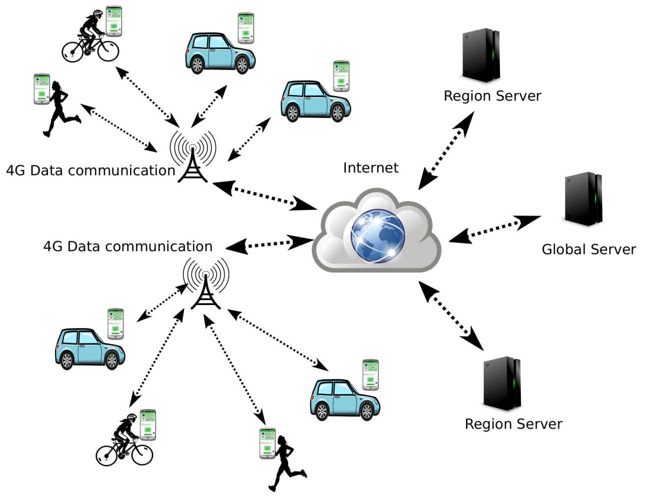

FIGURE 1. System architecture overview. Vehicles and VRUs are connected to the cloud of region servers through smartphones and the 4G cellular phone infrastructure. Global servers are in charge of the distribution of users among the region servers depending on their location.

and pedestrians carrying smartphones, the cellular network such as 3G, 4G or 5G, and a cloud service accessible via the Internet, as shown in Fig. 1.

The system has been designed for areas with mobile phone coverage given that the users send their position to a server on Internet. This is the case of most of the urban areas and main roads. The possibility of lacking phone infrastructure has not been considered. Yet, any alternative communications system could be adopted. In this regard, the size of the UDP messages used, 128 bytes, should not constitute any problem. Nevertheless, the network response time has to be tested in order to estimate the real-time conditions as in [7].

The cloud service is composed by servers which can take two roles: *global servers* and *region servers*. Global servers are in charge of system coordination, in particular they maintain the data structure of the set of region servers. They have to know in advance the address of all region servers, and the geographic region they cover. Users initially send their position to one of these servers, and they eventually obtain the region server they should work with.

Region servers provide the traffic warning system for all vehicles which are in the geographic region under their control. These servers will have to interact with other region servers for vehicles close the frontier of their region. Also they will have to interact with global servers for load balance matters, as overloaded region servers could not guarantee the real-time restriction of the system.

In the system, smartphones send UDP messages to the assigned region server. In response, region servers can answer with alarm messages, or they can answer to position messages if the smartphone asks for a connectivity confirmation.

Fig. 2 represents the different times involved in the communications between the clients and the server. The vehicles asking for warnings should receive an answer from the server before sending a new position message. This time is the position sending period, which is the GPS

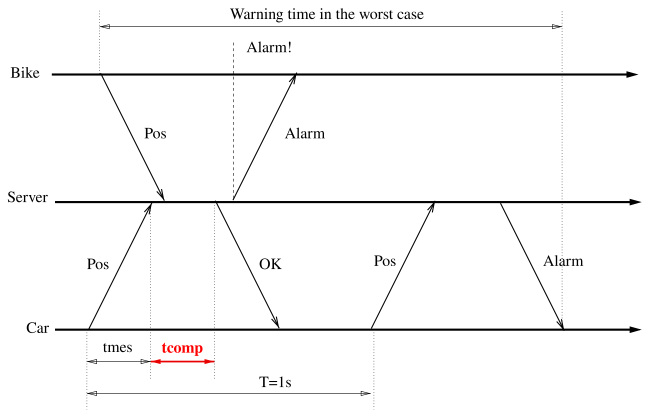

FIGURE 2. Worst case situation to receive a proximity warning: the server receives the bike's position update while already processing the car's position. The time used by the server to calculate possible alarms, *tcomp*, has been highlighted in red..

period ($T = 1s$), and it includes sending the position, the server's computing time, and the response from the server. Experimentally, we obtained response times of about 0.1 s corresponding mostly to the communication times, as in the experiments in [7] only two vehicles were being served.

The worst case to receive a warning happens when the message of one of the vehicles arrives just after the message of another close-by vehicle, for which the server is already searching for its alarm state. Then, we can assume a worse case scenario of 2 s (two GPS periods). Consequently, we assume that, during 2 s, the vehicles are approaching at a maximum of 100 km/h (for instance, a car at 80 km/h, and a bicycle in the opposite direction at 20 km/h), which results in a traveled distance of 55.5 m. In Fig. 2 *tmes* represents the communication time for messages (to/from the vehicle), and *tcomp* is the server's computation time; such a time involves updating the database and checking if the sender vehicle has proximity alerts. Note that the response time ($2tmes + tcomp$) has to be less than T. If the vehicles travel faster, and given that the communication time can not be tuned, the computation time should decrease (or alternatively increase the alarm distance to more than 150 m).

IV. THE SERVER SYSTEM

The server system we envision is composed of two parts in order to geographically distribute the workload: a redundant *global service* and *region servers*.

The pool of *global servers* implementing the global service is in charge of receiving the initial position message from vehicles, and then to retrieve the region server currently covering their location. In addition, they have to initiate region servers, and to coordinate the load balancing strategy of region servers.

Region servers cover a specific geographic region. They are the base of the system, as they receive position messages from vehicles which are traveling in their region. These servers support most of the load, as position messages are sent each second, as well as eventual alarms, which should be sent with a real-time restriction of also one second.

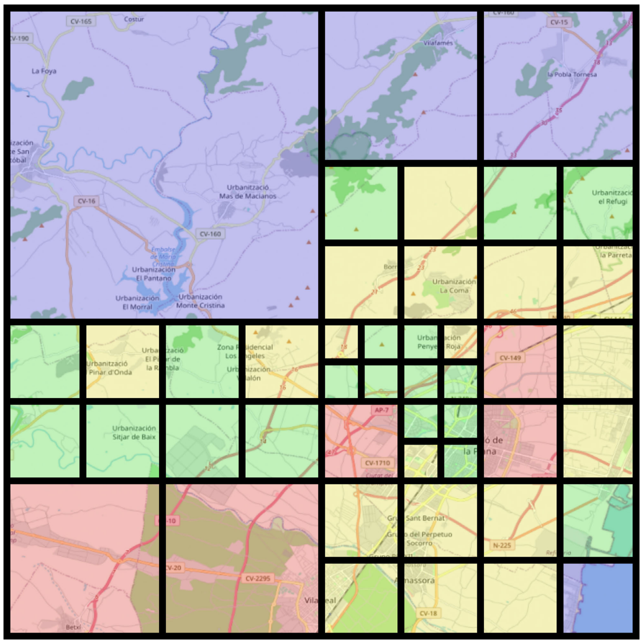

FIGURE 3. Schema detailing how a part of the Earth's surface is covered by region servers. The structure is a quadtree where blue regions are not covered, yellow regions are covered by servers with a reasonable load level, green regions are covered by servers with a low load, and red regions are covered by overloaded servers.

A. THE GLOBAL SERVICE

As stated above, a pool of replicated global servers offers the global service. Basically, these servers have two tasks: receiving initial messages when vehicles start travelling, and coordinating region servers.

The replicated *global servers* have a copy of the data structure with all existing region servers. The set of region servers is initially empty, and they are created when a vehicle visits a region not yet covered by any server.

When a vehicle starts using the application, it first sends its location to one of the pre-configured global servers. Then, this server answers with the address of the region server covering the client's position.

As region servers have real-time constraints (they have to be able to send alarms to vehicles in less than one second as a response to position messages), a load balancing strategy has to be implemented given the unpredictable fluctuations of the number of connected vehicles in a region. The idea is that region servers could be splitted or fused depending on their load. This produces a recursive quadtree data structure. Fig. 3 shows the evolution of geographic regions regarding the state of the respective region server. Blue regions are not yet covered (no region server was assigned), yellow regions are covered by servers with reasonable load levels, and finally green and red regions are covered with under or overloaded servers, respectively.

The procedure which global servers will apply when they receive an initial position message can be seen in

Algorithm 1. This algorithm is recursive, as it enquires in the quadtree data structure. The dimension arguments (*dimLat* and *dimLon*) are the dimension of regions in degrees at a given recursion level. In each recursion, these dimensions are half of the precedent level. Initially, the maximum pre-defined dimensions are used. The first level in the data structure is a bidimensional array, which is supposed to cover the whole earth surface. Many elements of this data structure will be empty, as they are never visited by vehicles. For visited regions, the corresponding element of the array will have a region server or a quadtree. In this algorithm, it is assumed that vehicles are able to determine the element of the initial bidimensional array to which the query starts. The interactions between a vehicle and both type of servers are shown in Fig. 4.

The mentioned maximum dimensions used in the first level of the region servers' data structure depend on the error, which can be assumed when calculating distance in degrees, as explained further in Section V-A. It must be noted that the maximum dimensions are scaled for different ranges of latitudes, given that the same longitude degrees represent different distances for different latitudes.

Algorithm 1 Determining the Region Server Covering a Location

Input	*node*	A node in the quadtree data structure storing the region servers
	(lat, lon)	A location on Earth in the geographic coordinate system
	dimLat	Latitude dimension of the region in degrees
	dimLon	Longitude dimension of the region in degrees
Output	*node*	The node in the quadtree data structure serving the location (lat, lon)

```
1:  algorithm GetRegionServer(node, (lat, lon), dimLat, dimLon)
2:      $y = \lfloor (lat - node.lowLat)/dimLat \rfloor$
3:      $x = \lfloor (lon - node.lowLon)/dimLon \rfloor$
4:      if $node[y][x].type$ is SERVER then
5:          return $node[y][x]$
6:      else if $node[y][x].type$ is Null then
7:          $lat_1 = node.lowLat$
8:          $lon_1 = node.lowLon$
9:          $lat_2 = lat_1 + dimLat/2$EndOutput
10:         $lon_2 = lon_1 + dimLon/2$
11:         $node[y][x]$ = CreateRegServ(($lat_1$, $lon_1$), ($lat_2$, $lon_2$))
12:         return $node[y][x]$
13:     else
14:         return GetRegionServer($node[y][x]$, ($lat$, $lon$), $dimLat/2$, $dimLon/2$)
15:     end if
16: end algorithm
```

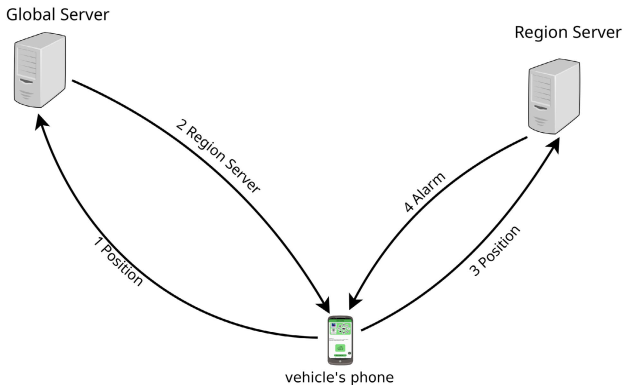

FIGURE 4. Message exchange sequence between a vehicle's smartphone and the two types of servers to start using the alarm system.

For the coordination of region servers, a load balancing system is needed because, as mentioned, overloaded region servers could loose real-time constraints. Thus, under region servers demand, they can be divided (or fused) to serve smaller (or bigger) regions. The process to split a server (and its region) in four child nodes is shown in Algorithm 2. Fig. 5 depicts the split-up process in four new region servers (following the quadtree structure).

Algorithm 2 Splitting a Region Server

Input	*node*	A node to be divided in the quadtree data structure
Output	Null	

```
1:  algorithm SplitRegionServer(node)
2:      $dimLat = (node.upperLat - node.lowLat)$
3:      $dimLon = (node.upperLon - node.lowLon)$
4:      $newDimLat = dimLat/2$
5:      $newDimLon = dimLon/2$
6:      $serverA$ = CreateRegServ(($node.lowLat$, $node.lowLon$), ($node.lowLat + newDimLat$, $node.lowLon + newDimLon$))
7:      $serverB$ = CreateRegServ(($node.lowLat + newDimLat$, $node.lowLon$), ($node.upperLat$, $node.lowLon + newDimLon$))
8:      $serverC$ = CreateRegServ(($node.lowLat$, $node.lowLon + newDimLon$), ($node.lowLat + newDimLat$, $node.upperLon$))
9:      $serverD$ = CreateRegServ(($node.lowLat + newDimLat$, $node.lowLon + newDimLon$), ($node.upperLat$, $node.upperLon$))
10:     node.CreateChildren(serverA, serverB, serverC, serverD)
11:     node.DeleteServer()
12: end algorithm
```

To merge the four child nodes of a quadtree node it is necessary to know the cumulative load of the respective region servers in order to be sure that the new server will

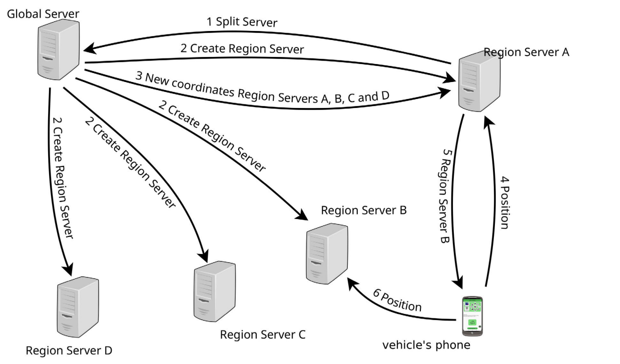

FIGURE 5. Messages and steps to split an overloaded region server. In message 5 the smartphone receives the address of the new region server if its last location belongs to the region of the new server (Region server B).

be able to cope with the resulting load. To do this the nodes are consulted, as it is shown in Algorithm 3. For the sake of simplicity, a maximum load for servers is considered (constant *MERGE_THRESHOLD*). Moreover, it is possible that any of the child nodes is also a parent node itself. In that case, a recursive call is done to check if they can also be merged. Finally, if the merge process is feasible, the new server in charge of the region is created (or assigned), and its children are deleted. Fig. 6 illustrates the process to merge four region servers.

The load definition, upon which the balancing system is based on, could be a function of the UDP buffer used for the application and other variables, such as the evolution of CPU usage. In our experiments we have observed a regular growth of this buffer when servers start being saturated. In addition, the load function should perform an exponential moving average (EMA), which will be continuously updated taking care of the history of the variable to prevent an excess of reconfigurations triggered by punctual load pics.

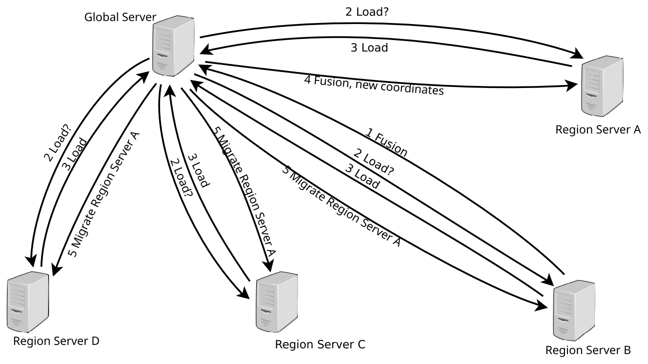

FIGURE 6. Messages involved in the fusion of four region servers. The process is started by demand of Region server B based on its load. The Global server asks region servers A to D about their load because they are siblings nodes of B. Server A is chosen to be the new server of the fused region. Servers B, C, and D are informed about the new server. These servers will answer their clients for a few more seconds to notify them about new server, and then vanish.

B. REGION SERVERS

Region servers support the heaviest part of all the system, handling all client messages having real-time restrictions, reason why we will focus on them in more depth.

Algorithm 3 Merging the Nodes of a Quadtree Node

```
Input   node    A node in the quadtree data structure to be merged with its other sibling region servers.
Output  load    If the merge is feasible, the load supported by the new merged region server.
1:  algorithm MergeRegionServers(node)
2:    $dimLat = (node.upperLat - node.lowLat)$
3:    $dimLon = (node.upperLon - node.lowLon)$
4:    $newDimLat = dimLat \cdot 2$
5:    $newDimLon = dimLon \cdot 2$
6:    parentNode = node.Parent()
7:    $load = 0$
8:    for $n \in parentNode.children$ do
9:      if n.type is SERVER then
10:       load = load + n.GetLoad()
11:     else
12:       if n.type is not Null then
13:         firstChild = n.children[0][0]
14:         load = load + MergeRegionServers(firstChild)
15:       end if
16:     end if
17:     if load >= MERGE_THRESHOLD then
18:       return
19:     end if
20:   end for
21:   newServer = CreateRegServ(
      (parentNode.lowLat, parentNode.lowLon),
      (parentNode.upperLat, parentNode.upperLon) )
22:   parentNode.AddNewServer(newServer)
23:   parentNode.DeleteChildren()
24:   return load
25: end algorithm
```

Region servers cover a geographical area defined by two geo-locations (the opposite left-lower and upper-right corners). They can ask the global service for a division or fusion depending on their workload given that, at a certain load level, its response time could be compromised or, in the opposite case, there is an undesirable atomization of region servers. Too many region servers are harmful because looking for alarms of vehicles near the region frontiers is a more costly procedure.

The creation of region servers and their coverage area could be decided by a mechanic division of the region, based on socioeconomic aspects such as the population of the area, or it could be decided by the entities which cooperate with the project. In the algorithms presented previously we propose a global coverage.

To provide an idea about the amount of simultaneous clients that a region server can reasonably take care of, we have compared two algorithms. In the first one we use a simple approach. In this case, all clients are stocked in a single list. We call it *list-algorithm*. The algorithm to process a position message from a vehicle is shown in Algorithm 4. In this algorithm we suppose that client messages are composed by the identity of the vehicle, its position, and a boolean value indicating if it requires alarm notifications. The *vehicleList* argument is the list with all vehicles in the region and their positions.

Algorithm 4 List-Algorithm

```
Input   vehicleList  A list of all vehicles in the server region.
        message      A position message from a vehicle, which includes the intention to receive alarms.
Output  alarm        A boolean indicating if another vehicle is within the security distance.
1:  algorithm ProcessMessageList($vehicleList$, $message$)
2:    $alarm = False$
3:    $inserted = False$
4:    for $v \in vehicleList$ do
5:      if $v == message.vehicle$ then
6:        $v.position = message.position$
7:        $inserted = True$
8:        if $\neg message.asksAlarm$ then
9:          return
10:       end if
11:     else if $message.asksAlarm$ & Distance($v.position$, $message.position$) $<$ $securityDistance$ then
12:       $alarm = True$
13:     end if
14:   end for
15:   if $\neg inserted$ then
16:     $vehicleList$.Append(($message.vehicle$, $message.position$))
17:   end if
18:   return $alarm$
19: end algorithm
```

In the list-algorithm, as shown in Algorithm 4, when a client sends its position, the list is sequentially traversed until the vehicle is found or (if demanded) an alarm is detected. In fact, in our implementation, for each vehicle in the list, the distance with the current vehicle is computed only if it asks for alarms, and if their locations are closer than the pre-defined *security distance*.[1]

The cost of this algorithm is quadratic with the amount of simultaneous vehicles because the list has to be passed through for every single message of every vehicle ($\Theta(n^2)$). In this algorithm, outdated information is purged for the selected elements when looking for alarms, but also, periodically, a complete purge should take place as the list could contain zombie vehicles (i.e. vehicles that are no longer active/connected).

[1] 150 meters is the security distance we determined in the communications part of the system.

Algorithm 5 Cells-Algorithm

```
Input   grid     A matrix containing all cells in the region. Each cell has a list of vehicles visiting the cell.
        message  A position message from a vehicle, including the intention to receive alarms.
Output  alarm    A boolean indicating if another vehicle is within the security distance.
1:  algorithm ProcessMessageCells($grid$, $message$)
2:    $cell_y = (message.position.lat - grid.base\_lat)/grid.step\_lat$
3:    $cell_x = (message.position.lon - grid.base\_lon)/grid.step\_lon$
4:    $grid[cell_x][cell_y].Append(message.vehicle, message.position)$
5:    if $message.askAlarm$ then
6:      for $x \in (cell_x - 1, \dots, cell_x + 1)$ do
7:        for $y \in (cell_y - 1, \dots, cell_y + 1)$ do
8:          for $v$ in $grid[x][y]$ do
9:            if Outdated($v.date$) then
10:             $grid[x][y].delete(v)$
11:           else if $v.vehicle == message.vehicle$ & Distance($v.position$, $message.position$) $<$ $securityDistance$ then
12:             return $True$
13:           end if
14:         end for
15:       end for
16:     end for
17:     return $False$
18:   else
19:     return
20:   end if
21: end algorithm
```

In the other algorithm, which we call *cells-algorithm*, the region handled by the server is divided into square cells. Each cell has a list of vehicles which are visiting its area. As it can be seen in Algorithm 5, when a position message arrives from a client, it is quite easy to find the respective cell (representing a cost of $\Theta(1)$). For each dimension (latitudes and longitudes), the coordinates in the bidimensional array of cells are obtained by the integer division $(position - base\_position)/step$, where *position* is the position of the vehicle, *base_position* is the left lower corner of the region covered by the server, and *step* is the fixed width of the cells in both dimensions, latitudes and longitudes (all these values are in degrees).

Once the cell is found, the vehicle is added to its vehicle list. Then, if the vehicle intends to receive alarms, the list of that cell, and those of its eight immediate neighbor cells, are sequentially traversed, and the process stops when an alarm is found.

In the cells-algorithm, the purging process is done while looking for alarms. When outdated vehicles are found in the lists, they are removed. It should also be emphasized that zombie vehicles in non-visited cells do not affect the performance.

The cost of this algorithm is quadratic with the amount of simultaneous vehicles in a cell. If we consider a uniform distribution of the vehicles in a square region, the computational complexity will be $\Theta(k^2)$ where $k = n/c^2$, being c the number of cells in each dimension. This version represents a significant improvement with respect to the list-algorithm: however a considerable data structure is needed. For instance, for cells of side 150 meters (i.e. the security distance), a region of $250 \times 250\ km^2$ would need a two-dimensional array of 1667×1667 items.

V. EXPERIMENTS

In this section a description of the test environment we have used and the results we have obtained is presented. The experiments have been conducted to compare the two presented server algorithms, the amount of vehicles that can be assisted, and the influence of different aspects such as the amount of vehicles demanding alarms or the size of cells in the cells-algorithm.

A. IMPLEMENTATION OF REGION SERVERS AND THE TEST ENVIRONMENT

Both versions of the server have been implemented in Python 3.7.3 to simplify prototyping. Messages are received in a main thread, and then they are processed in a different thread. However, all is executed in only one processor at a time due to the Python's GIL (Global Lock Interpreter). Even if Python is not the best choice for a high performance application, it is enough to study the main hints of these parts of the system.

We have instrumented the code of the server to generate an execution trace at the end of each experiment with the parameters of our interest.

All the tests have been executed in a server with twelve Intel Core(TM) i7-8700 CPU 3.20GHz with a RAM of 24 Gbytes running Debian GNU/Linux 10.10. To avoid an early overflow of the UDP buffer, we have increased it from 208 Kbytes, which is the default value, to 64 MBytes.

IMPROVING DISTANCE COMPUTING

Calculating the distance in kilometers between two geo-positions with latitudes and longitudes needs costly trigonometric operations. A common method to address this is to use the haversine formula (shown in (1) where $r = 6371\ km$ is the mean radius of the Earth).

$$H((lat_1, lon_1), (lat_2, lon_2)) = 2r\, sin^{-1}\left(\left(sin^2\left(\frac{lat_2 - lat_1}{2}\right) + cos(lat_1)\, cos(lat_2)\, sin^2\left(\frac{lon_2 - lon_1}{2}\right)\right)^{1/2}\right) \quad (1)$$

We reduce the computation time of this part by calculating the distance in both dimensions in degrees, and then converting these distances to meters given that, for rectangular regions of the earth that are not excessively large (what we call a cell), we can approximate the correspondence between degrees and meters.

This procedure introduces an error, specially for longitudes because the same degrees in longitude represents a different distance in meters depending on the latitude. For instance, in a region at latitude 39.5 degrees, we have an horizontal step of 0.001744 degrees for 150 meters, and at latitude 41 degrees, the same horizontal step in degrees represents 146.72 meters. To distribute this error we calculate the step in degrees for the cells' side in the middle of the region and, in addition, the algorithm to create region servers should limit the error to a maximum of 5 meters. This means that the total error between the border latitudes of the region should be less than 10 meters. In latitudes close to 40 degrees, this makes it possible to work with regions of about 500 km of side. In case of a massive use of the system, regions will probably be much smaller to support the real-time requirements. Also, it must be noted that this error is additional to the GPS error, which is accepted to be usually lower than 5 meters [6]. Finally, it must be noted that the calculated distances of our interest are limited to 424 meters (the worst case in which two vehicles are in the diagonal of two cells with 150 meters of side).

Given the lower left and upper right corners in degrees, (Lat_1, Lon_1) and (Lat_2, Lon_2), and being *side* the cells' side in meters, the steps in degrees are initially calculated as follows, when the region server is configured. First of all, the dimension in meters of both sides of the region are computed, as shown in (2) and (3) (where $H()$ is the haversine formula).

$$dim_{lat} = 1000 \cdot H\left(\left(Lat_1, Lon_1 + \frac{Lon_2 - Lon_1}{2}\right), \left(Lat_2, Lon_1 + \frac{Lon_2 - Lon_1}{2}\right)\right) \quad (2)$$

$$dim_{lon} = 1000 \cdot H\left(\left(Lat_1 + \frac{Lat_2 - Lat_1}{2}, Lon_1\right), \left(Lat_1 + \frac{Lat_2 - Lat_1}{2}, Lon_2\right)\right) \quad (3)$$

Then the number of degrees equivalent to a meter, both in latitude and longitude, is computed (see (4) and (5)).

$$step_{lat} = \frac{regLat_2 - regLat_1}{dim_{lat}} \quad (4)$$

$$step_{lon} = \frac{regLon_2 - regLon_1}{dim_{lon}} \quad (5)$$

Thus, to compute the distance in meters between two geo-positions, (lat_1, lon_1) and (lat_2, lon_2) using the region approximations $step_{lat}$ and $step_{lon}$, we use the pythagoras formula shown in (6).

$$distance((lat_1, lon_1), (lat_2, lon_2)) = \sqrt{\left(\frac{lat_1 - lat_2}{step_{lat}}\right)^2 + \left(\frac{lon_1 - lon_2}{step_{lon}}\right)^2} \quad (6)$$

We have used a profiler to check the benefit of calculating the distance between vehicles applying pythagoras in degrees rather than using the haversine formula (through Geopy Python module) in the cells-algorithm. In a test with 2000 clients for 300 seconds, the time to compute the distances (around 670,000 calls) accumulated 2.341 and 109.961 execution seconds, respectively, being that our approach represents an improvement of about 47 times. Consequently, in the rest of the tests performed in the paper, the pythagoras method has been used for distance computing.

Fig. 10 (explained further in Section V-C) shows the impact of using both distance computing methods on the server load.

B. SYNTHETIC WORKLOAD

For the experiments, a synthetic load has been developed in which vehicles are emulated with a random movement inside the geographic area covered by the server. These simulated vehicles start their motion at different random points, and send a position message every second. After sending its simulated position, each vehicle chooses an increment or decrement of their next direction and speed, and then calculate what will be its future position in the next second. Speeds are chosen uniformly from 0 to 80 km/h. Fig. 7 shows the starting and arrival locations (red and blue dots respectively) of a hundred of synthetic vehicles in a 17 × 17 km^2 area. They were moving for 600 seconds covering an average distance of 6.4 km with a standard deviation of 2.2 km.

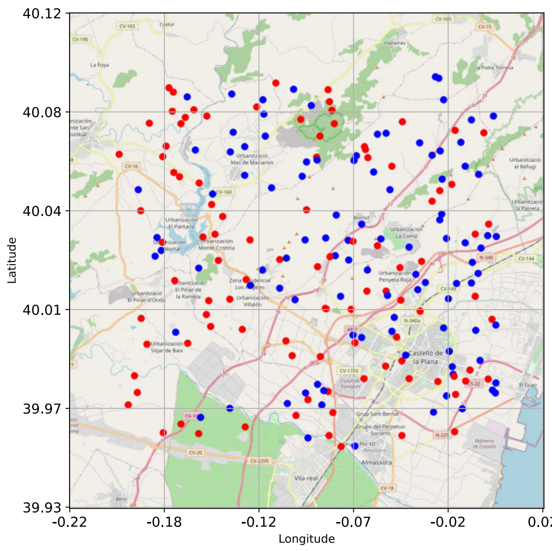

FIGURE 7. Origin (red dots) and destination (blue dots) of 100 synthetic vehicles traveling for 600 seconds in an area of 17 × 17 km^2.

In the results presented below, except in Section V-F, all vehicles ask for proximity alarms of all other vehicles. This is not the expected use of the application, as probably cars want to be alerted of the presence of VRUs, but not of other cars, and cyclists want to communicate their position, but do not ask for any alarm. We have made this simplification assuming that it represents a heavier load (worst-case scenario).

To execute the synthetic load, 28 computers have been used, each one with four Intel Core(TM) i5-7400 CPU 3.00GHz with 15 Gbytes of RAM running Ubuntu 20.04.1. Each of these computers is able to execute up to 950 simultaneous virtual clients.

The set of computers executing the clients is connected to the server through our university campus network, which at the endpoints is a Fast Ethernet network (100 Mbps).

C. SATURATION LOAD

We have made tests to detect the saturation load for region servers. As the service time depends on traffic density and the amount of vehicles, so does the saturation point. To detect the load level which saturates the server, we have registered the time between the moment at which clients send position messages, and the time when the server software receives the messages (it takes the messages from the operating system UDP buffer). When the capacity of the server is exceeded, the percentage of messages that are older than one second raises abruptly. It must be noted that, when the server handles messages older than one second, the real-time restriction fails.

The time elapsed between the instant at which a client sends a message and the server receives it, is approximate, as we compare the server clock when it takes a message against the timestamp defined by the client, and the clocks of both computers can not be exactly synchronized. This can happen even if all computers periodically adjust their clocks using an external time server, for instance using the Network Time Protocol [34]. As a consequence, it is possible to obtain negative values for this variable. However, this variable is accurate enough to give us an idea about the saturation point of the system.

As we have said previously, in these tests we have incremented the UDP buffer to detect the saturation point when messages start to be outdated, rather than when messages are discarded due to UDP buffer overflow. It must be noted that the tests have been done on the wired campus network, and the number of lost messages is negligible.

Fig. 8 shows the saturation point of the list-algorithm version of the server for square regions of sides 160, 17, and 3 km. In these tests, the regions of side 160 and 17 km saturate with 1750 clients. The region of 3 km of side saturates later, with 2500 clients. This latter region has a higher traffic density and, when walking through the list, alarms are detected with less iterations. For the amount of clients in these tests, this time is close to zero for the cells-algorithm version.

Tests with a greater number of clients have been done to saturate the cells-algorithm version of the server. For the three mentioned regions (squares of sides 160, 17, and

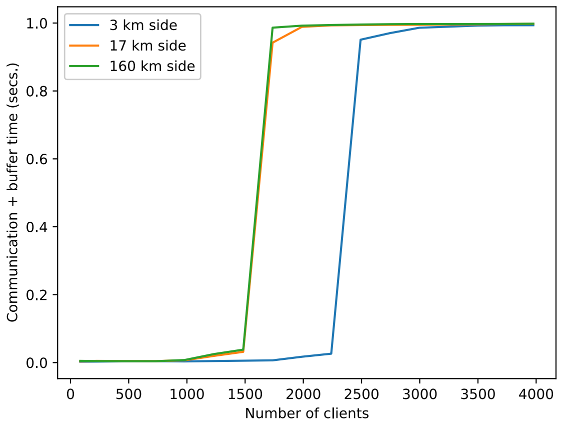

FIGURE 8. Communication plus UDP buffer time for square regions of 3, 17 and 160 km of side for the list-algorithm as function of the number of clients.

3 km), the saturation point can be seen in Fig. 9. The saturation for the three regions takes place at 13,000 clients with 2,88% of outdated messages, 13,500 clients with 7,2% of outdated messages, and 15,000 clients with 3,02% of outdated messages, respectively. Up to a certain limit of traffic density, in which the time to serve the messages stabilizes, increasing traffic density actually reduces the service time, meaning that the saturation takes place at a threshold higher than expected, hence allowing the system to support more vehicles.

Fig. 9 shows a step where the communications time suddenly increases by about 1 second. For a range of load, the server can deal with a percentage of non-outdated messages because outdated messages are just discarded. Notice that, beyond a certain load level, the server is no longer able to process messages, and it can just receive outdated messages. After this point, the time in the UDP buffer starts growing until a buffer overflow occurs.

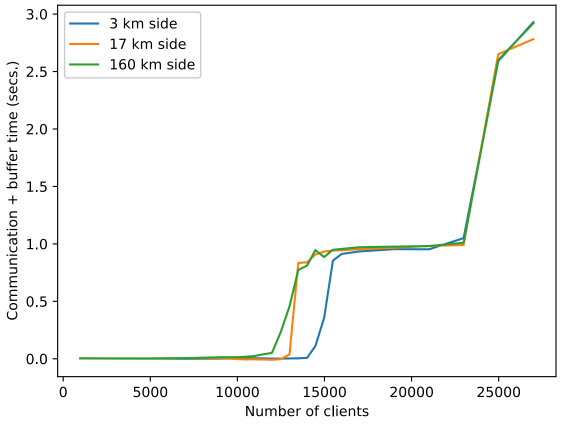

FIGURE 9. Communication plus UDP buffer time for square regions of 3, 17 and 160 km of side for the cells-algorithm.

Fig. 10 shows the impact of using both methods (pythagoras and Geopy library) to compute the distance between vehicles, as explained in Section V-A. It can be seen that, using the haversine formula, the saturation of the server starts even before the 3000 client threshold is reached. These tests have been done with the cells-algorithm version of the server with a region of $17 \times 17\ km^2$. However, when using our proposed pythagoras formulation, the server is able to serve up to 13,000 clients for the same region, as it can be seen in Fig. 9.

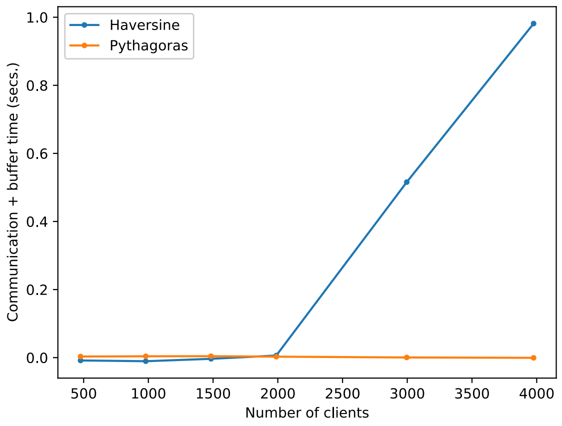

FIGURE 10. Communication plus UDP buffer time to show the effect of distance computing method on the saturation of the cells-algorithm version of the server in a region of $17 \times 17\ km^2$.

In the experiments three different traffic densities have been tested, showing that, for highers densities, slightly more load is supported. In any case, saturation detection must be checked independently of the traffic density. This applies for regions in which there are unbalanced densities, as it can be the case of a region including several urban agglomerations. The number of users saturating the servers may change, but the proximity of saturation events have to be detected in any case.

D. SERVICE TIME

The *service time* we have measured is the time between the start and end of the thread to process a client position message. That is, to update the position of the client in the data structure of the server, and to look for and notify alarms. To check the dependence of this time and the traffic density we have tested the service time of both algorithms as function of the amount of clients in three square areas, with sides of about 160, 17, and 3 km. The results of the average service time are shown in Fig. 11, 12, and 13. These figures show, as expected, the benefits of dividing the region in cells and working with many reduced lists. The list-algorithm curve (in blue) in these figures is interrupted for the number of clients which saturates this version of the server, given that beyond this amount of clients, there are many outdated messages and they are not processed by the server.

The server based on the list-algorithm is saturated around 2000 clients. When saturation is reached, each second the server receives more messages than it can process. Consequently, UDP buffer starts growing, and there is more and more messages older than one second; when this happens, real-time restrictions cannot be guaranteed. On the other hand, the cells-algorithm version of the server largely

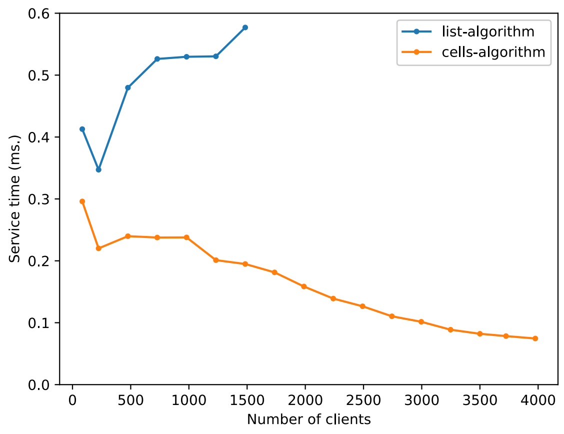

FIGURE 11. Service time as function of the number of clients for a square region of 160 km of side for both versions of the region server. The times for the list-algorithm (in blue) are shown until saturation is reached. The cells-algorithm is not saturated for the amount of clients in the figure.

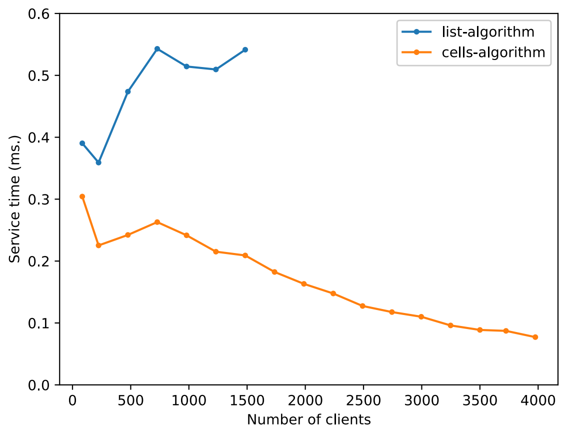

FIGURE 12. Service time as function of the number of clients for a square region of 17 km of side. The times for the list-algorithm (in blue) are shown until saturation is reached. The cells-algorithm is not saturated for the amount of clients in the figure.

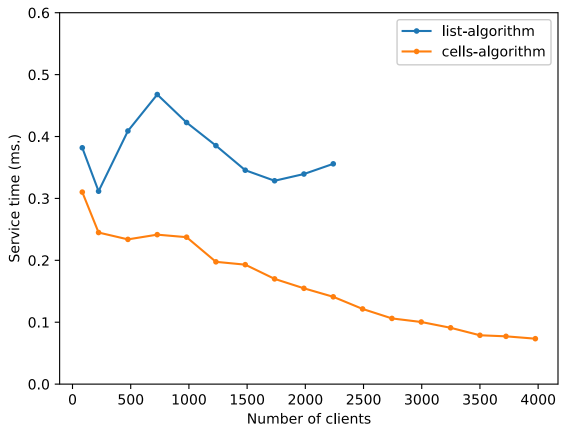

FIGURE 13. Service time as function of the number of clients for a square region of 3 km of side. The times for the list-algorithm (in blue) are shown until saturation is reached. The cells-algorithm is not saturated for the amount of clients in the figure.

outperforms the list-algorithm one as the list is distributed among the cells.

Fig. 14 shows the average service time as function of the number of clients for the cells-algorithm server, and

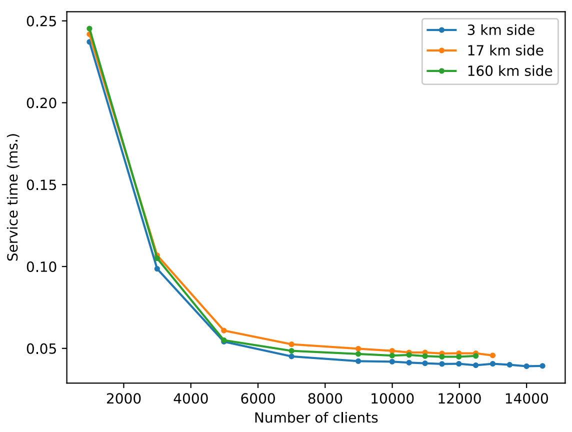

FIGURE 14. Service time for the cells-algorithm server as function of the number of clients for square regions of 3, 17 and 160 km of side.

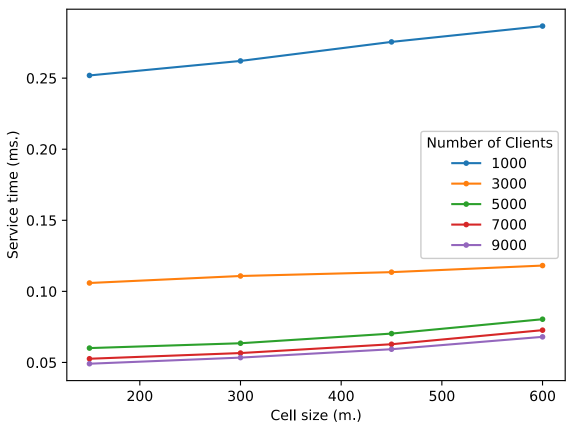

FIGURE 15. Service time for a region of 17 km of side as function of the size of cells in the cells-algorithm. There is a line for each number of clients, ranging from 1,000 to 9,000 clients.

for the three considered regions. For each region, only the values before the saturation of the server are shown. It can be seen that the service time of the cells-algorithm version reduces progressively with the increment of vehicles, stabilizing at around 0.05 milliseconds. This is a consequence of the increment of traffic density, given that alarms are found early.

E. EFFECT OF CELLS SIZE

In the cells-algorithm we have chosen the alarm distance, 150 meters, as the cells' side. The idea was that if a cell has more than one vehicle, it could be possible to activate the alarm without computing the distance for the sake of efficiency, even if it is not exact as vehicles could be on the extremes of the diagonal of the cell. In the tests, this capability has not been used, and it can be of interest to compare the service time of the algorithm with different cell's size.

Fig. 15 shows a comparison of the service time for different sizes of cells and for different numbers of clients. The region used in these tests is a square of 17 km of side. It can be seen that, independently of the number of clients, the best choice is to use the security distance for the cells' side, even if the previously mentioned acceleration is not applied.

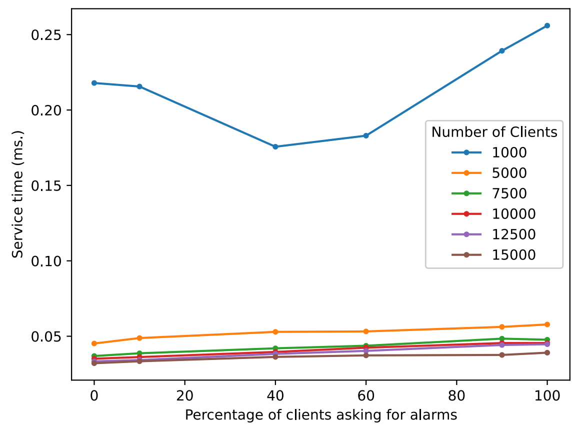

FIGURE 16. Service time as a function of the percentage of clients demanding alarms in a square region of 160 km of side. There is a line for each number of clients, ranging from 1,000 to 15,000 clients.

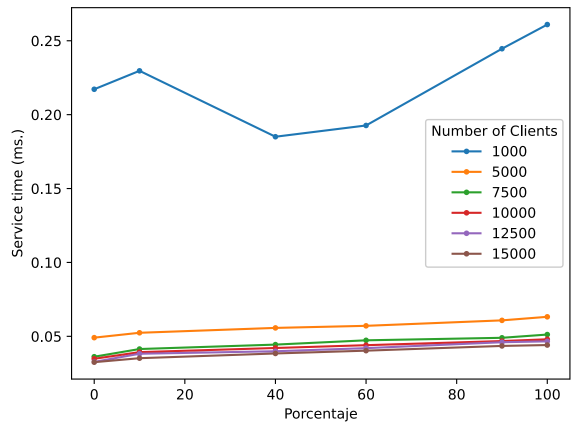

FIGURE 17. Service time as function of the percentage of clients demanding alarms for a square region of 17 km of side. There is a line for each number of clients, ranging from 1,000 to 15,000 clients.

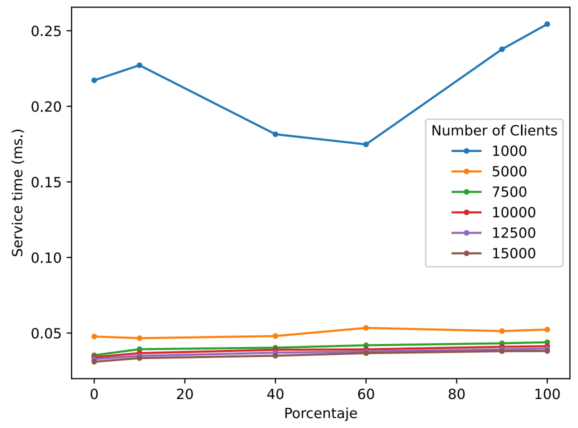

FIGURE 18. Service time as function of the percentage of clients demanding alarms for a square region of 3 km of side. There is a line for each number of clients, ranging from 1,000 to 15,000 clients.

F. NUMBER OF VEHICLES DEMANDING ALARMS

Previous tests have been done assuming that all vehicles ask for alarms of all other vehicles, as it seems to be the heaviest workload (worst-case scenario). So, we have made tests with different percentages of clients asking alarms. Vehicles who do not ask for alarms just send position messages to warn the other vehicles.

Fig. 16, 17 and 18 show the service time (time to handle the messages on the data structure) for the cells-algorithm server with different percentages of vehicles demanding alarms. Figures correspond to regions of 160 × 160, 17 × 17 and 3 × 3 km^2, respectively. In these figures a light increment of the service time when increasing the number of vehicles asking for alarms. Thus, we conclude that in the cells-algorithm, the main part of the work is due to the management of position messages.

VI. CONCLUSION

This paper builds upon previous research to explore if a traffic alarm system can rely on 4G cellular infrastructure and smartphones. The prior study [7] confirmed the viability of the communication aspect, whereas the present paper focuses on the server component. Our proposal consists of a cloud of *coordination servers* and *region servers* which oversee vehicles in specific geographic areas, and we've implemented two algorithms to assess real-time constraints.

Region servers process vehicle location data, issuing collision warnings in less than a second for those vehicles closer than a security distance. Two server versions were tested: one with a single list of vehicles, and another one where the target region is divided into cells. The cell-based version significantly outperforms the single-list one, achieving service times below 0.1 ms, and supporting 13,000 vehicles for a region up to 25,600 km^2, a value that could be slightly increased to 15,000 vehicles when the region was limited to 9 km^2.

However, the algorithms for implementing a global service are still to be tested. Our future work involves evaluating response times when multiple region servers collaborate in their borders. In that regard, we plan to study how the load balance system affects the real-time constraints.

REFERENCES

[1] M. Léo, "A study on bicycle and public transportation synergy based on a cross-analysis between Europe and Japan," Ph.D. thesis, Inst. Urban Innov., Yokohama Nat. Univ., Yokohama, Japan, 2021.

[2] C. Zhao, T. A. Carstensen, T. A. S. Nielsen, and A. S. Olafsson, "Bicycle-friendly infrastructure planning in Beijing and copenhagen–between adapting design solutions and learning local planning cultures," *J. Transp. Geography*, vol. 68, pp. 149–159, Apr. 2018.

[3] European Comission. *2019 Road Safety Statistics: What is Behind the Figures?*. Accessed: Jan. 18, 2021. [Online]. Available: https://ec.europa.eu/commission/presscorner/detail/en/qanda_20_1004

[4] A. Kourtellis, P.-S. Lin, and N. Kharkar, "Smartphone-based connected bicycle prototype development for sustainable multimodal transportation system," Center Urban Transp. Res., Univ. South Florida, Tampa, FL, USA, Tech. Rep. CUTR-NCTR-RR-2018-03, 2019.

[5] F. Dressler, F. Klingler, M. Segata, and R. L. Cigno, "Cooperative driving and the tactile internet," *Proc. IEEE*, vol. 107, no. 2, pp. 436–446, Feb. 2019.

[6] M. Liebner, F. Klanner, and C. Stiller, "Active safety for vulnerable road users based on smartphone position data," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2013, pp. 256–261.

[7] P. Boronat, M. Pérez-Francisco, C. T. Calafate, and J.-C. Cano, "Towards a sustainable city for cyclists: Promoting safety through a mobile sensing application," *Sensors*, vol. 21, no. 6, p. 2116, Mar. 2021.

[8] S. Herrnleben, M. Pfannemüller, C. Krupitzer, S. Kounev, M. Segata, F. Fastnacht, and M. Nigmann, "Towards adaptive car-to-cloud communication," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops)*, Mar. 2019, pp. 119–124.

[9] G. Kapousizis, M. B. Ulak, K. Geurs, and P. J. M. Havinga, "A review of state-of-the-art bicycle technologies affecting cycling safety: Level of smartness and technology readiness," *Transp. Rev.*, vol. 43, no. 3, pp. 430–452, May 2023.

[10] M. Jenkins, D. Duggan, and A. Negri, "Towards a connected bicycle to communicate with vehicles and infrastructure: Multimodel alerting interface with networked short-range transmissions (MAIN-ST)," in *Proc. IEEE Conf. Cognit. Comput. Aspects Situation Manage. (CogSIMA)*, Mar. 2017, pp. 2–4.

[11] S. Céspedes, J. Salamanca, A. Yañez, C. Rivera, and J. C. Sacanamboy, "Platoon-based cyclists cooperative system," in *Proc. IEEE Veh. Netw. Conf. (VNC)*, Dec. 2015, pp. 112–118.

[12] S. Smaldone, C. Tonde, V. K. Ananthanarayanan, A. Elgammal, and L. Iftode, "The cyber-physical bike: A step towards safer green transportation," in *Proc. 12th Workshop Mobile Comput. Syst. Appl.*, Mar. 2011, pp. 56–61.

[13] W. Jeon and R. Rajamani, "Rear vehicle tracking on a bicycle using active sensor orientation control," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 8, pp. 2638–2649, Aug. 2018.

[14] Volvo Cars. (2017). *Collision Warning System—Detection of Cyclists*. Accessed: Jan. 14, 2021. [Online]. Available: https://www.volvocars.com/lb/support/manuals/v60-cross-country/2016w17/driver-support/collision-warning-system/collision-warning-system—detection-of-cyclists

[15] M. Dozza, P. Gustafsson, L. Lindgren, C.-N. Boda, and J. C. Muñoz-Cantillo, "Bikecom—A cooperative safety application supporting cyclists and drivers at intersections," in *Proc. 3rd Conf. Driver Distraction Inattention*, Gothenbrug, Sep. 2013, pp. 4–6.

[16] K. David and A. Flach, "CAR-2-X and pedestrian safety," *IEEE Veh. Technol. Mag.*, vol. 5, no. 1, pp. 70–76, Mar. 2010.

[17] C. Sugimoto, Y. Nakamura, and T. Hashimoto, "Prototype of pedestrian-to-vehicle communication system for the prevention of pedestrian accidents using both 3G wireless and WLAN communication," in *Proc. 3rd Int. Symp. Wireless Pervasive Comput.*, May 2008, pp. 764–767.

[18] R. Bastani Zadeh, M. Ghatee, and H. R. Eftekhari, "Three-phases smartphone-based warning system to protect vulnerable road users under fuzzy conditions," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 7, pp. 2086–2098, Jul. 2018.

[19] C.-H. Lin, Y.-T. Chen, J.-J. Chen, W.-C. Shih, and W.-T. Chen, "PSafety: A collision prevention system for pedestrians using smartphone," in *Proc. IEEE 84th Veh. Technol. Conf. (VTC-Fall)*, Sep. 2016, pp. 1–5.

[20] Z. Liu, L. Pu, Z. Meng, X. Yang, K. Zhu, and L. Zhang, "POFS: A novel pedestrian-oriented forewarning system for vulnerable pedestrian safety," in *Proc. Int. Conf. Connected Vehicles Expo (ICCVE)*, Oct. 2015, pp. 100–105.

[21] C.-Y. Li, G. Salinas, P.-H. Huang, G.-H. Tu, G.-H. Hsu, and T.-Y. Hsieh, "V2PSense: Enabling cellular-based V2P collision warning service through mobile sensing," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.

[22] U. Hernandez-Jayo, I. De-la-Iglesia, and J. Perez, "V-alert: Description and validation of a vulnerable road user alert system in the framework of a smart city," *Sensors*, vol. 15, no. 8, pp. 18480–18505, Jul. 2015.

[23] H. Artail, K. Khalifeh, and M. Yahfoufi, "Avoiding car-pedestrian collisions using a VANET to cellular communication framework," in *Proc. 13th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Jun. 2017, pp. 458–465.

[24] S. Verma, Y. Kawamoto, Z. Md. Fadlullah, H. Nishiyama, and N. Kato, "A survey on network methodologies for real-time analytics of massive IoT data and open research issues," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1457–1477, 3rd Quart., 2017.

[25] Y. Elbanoby, M. Aborizka, and F. Maghraby, "Real-time data management for IoT in cloud environment," in *Proc. IEEE Global Conf. Internet Things (GCIoT)*, Dec. 2019, pp. 1–7.

[26] T. Yu and X. Wang, "Real-time data analytics in Internet of Things systems," in *Handbook of Real-Time Computing*. Singapore: Springer, pp. 541–568.

[27] R. Diaconu, "Scalability for virtual worlds," Ph.D. thesis, École Doctorale Informatique, Télécommunications et Électronique, Université Pierre et Marie Curie-Paris VI, Paris, France, 2015.

[28] A. Yahyavi and B. Kemme, "Peer-to-peer architectures for massively multiplayer online games: A survey," *ACM Comput. Surveys*, vol. 46, no. 1, pp. 1–51, Oct. 2013.

[29] B. Van Den Bossche, B. De Vleeschauwer, T. Verdickt, F. De Turck, B. Dhoedt, and P. Demeester, "Autonomic microcell assignment in massively distributed online virtual environments," *J. Netw. Comput. Appl.*, vol. 32, no. 6, pp. 1242–1256, Nov. 2009.

[30] D. Pittman and C. GauthierDickey, "A measurement study of virtual populations in massively multiplayer online games," in *Proc. 6th ACM SIGCOMM workshop Netw. Syst. Support Games*. New York, NY, USA: Association for Computing Machinery, Sep. 2007, pp. 25–30.

[31] U. Farooq and J. Glauert, "Integrating dynamic scalability into the OpenSimulator framework," *Simul. Model. Pract. Theory*, vol. 72, pp. 118–130, Mar. 2017.

[32] C. E. B. Bezerra and C. F. R. Geyer, "A load balancing scheme for massively multiplayer online games," *Multimedia Tools Appl.*, vol. 45, nos. 1–3, pp. 263–289, Oct. 2009.

[33] A. Akbar, G. Kousiouris, H. Pervaiz, J. Sancho, P. Ta-Shma, F. Carrez, and K. Moessner, "Real-time probabilistic data fusion for large-scale IoT applications," *IEEE Access*, vol. 6, pp. 10015–10027, 2018.

[34] J. Martin, J. Burbank, W. Kasch, and P. D. L. Mills, *Network Time Protocol Version 4: Protocol and Algorithms Specification*, document RFC 5905, Jun. 2010.

MIGUEL PEREZ-FRANCISCO received the bachelor's degree in computer science from Universitat Politècnica de València (UPV), in 1992, and the Ph.D. degree in computer science from Universitat Jaume I (UJI), Spain, in 1998. He is currently an Associate Professor of computer science with the Department of Computer Science and Engineering, UJI. His main research interests include wireless communications, mobile sensor networks, and frailty detection in elderly people.

PABLO BORONAT received the bachelor's degree in computer science from Universitat Politècnica de València (UPV), in 1991, and the Ph.D. degree in computer science from Universitat Jaume I (UJI), Spain, in 2001. He is currently an Associate Professor of computer science with the Department of Computer Languages and Systems, UJI. His main research interests include community networks, wireless communications, mobile sensor networks, and frailty detection in elderly people.

CARLOS T. CALAFATE (Senior Member, IEEE) received the Graduate degree (Hons.) in electrical and computer engineering from the University of Oporto, Portugal, in 2001, and the Ph.D. degree (cum laude) in informatics from Universitat Politècnica de València (UPV), Spain, in 2006. Since 2002, he has been with UPV, where he is currently a Full Professor with the Department of Computer Engineering. His research interests include ad-hoc and vehicular networks, UAVs, smart cities and the IoT, QoS, network protocols, video streaming, and network security. To date he has published more than 500 articles, several of which in journals, including IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE/ACM TRANSACTIONS ON NETWORKING, *Ad hoc Networks* (Elsevier), and *IEEE Communications Magazine*. He is a founding member of the IEEE SIG on Big Data with Computational Intelligence and the IEEE SIG on Green Internet of Vehicles. He is an associate editor of several international journals from editorials, including Elsevier, Hindawi, MDPI, IET, and SAGE, and the Section Editor-in-Chief of the *Drones* (MDPI) journal. He has participated in the TPC of more than 250 international conferences. He is ranked among the World's Top 2% Scientists, and also among the top 100 Spanish researchers in the computer science and electronics field.

JUAN-CARLOS CANO (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in computer science from Universitat Politècnica de València (UPV), Spain, in 1994 and 2002, respectively. From 1995 to 1997, he was a Programmer Analyst with IBM's Manufacturing Division, Valencia. He is currently a Full Professor with the Department of Computer Engineering, UPV. His current research interests include wireless communications, vehicular networks, mobile ad hoc networks, and pervasive computing.

PIETRO MANZONI (Senior Member, IEEE) received the master's degree in computer science from Università degli Studi di Milano, Milan, Italy, in 1989, and the Ph.D. degree in computer science from Politecnico di Milano, Milan, in 1995. From November 1992 to February 1993, he interned at Bellcore Labs, Red Bank, NJ, USA. From February 1994 to November 1994, he was a Visiting Researcher with the International Computer Science Institute (ICSI), Berkeley, CA, USA. He is currently a Professor of computer engineering with Universitat Politècnica de València, Spain. He is also developing solutions for the Internet of Things using LPWAN networks and Pub/Sub systems. These solutions have various applications, including environmental intelligence by integrating TinyML-based solutions, sustainable and green IoT, and smart tourism. In addition, he is interested in exploring different aspects of network pluralism and finding ways to provide integrated connectivity in the edge-cloud continuum. His research interest includes mobile wireless networks to create dynamic systems. He is the Coordinator of the Computer Networks Research Group (GRC) and a member of the IEEE Technical Committee on Hyper-Intelligence, the IEEE SIG on Metaverse, and the ACM SIGCAS-Computers and Society.