International Journal on Advanced Science Engineering Information Technology

Vol.10 (2020) No. 2
ISSN: 2088-5334

An Intelligent Transportation System: the Quito City Case Study

Ana Zambrano[a1], Marcelo Zambrano[b,] Eduardo Ortiz[a2], Xavier Calderón[a3], Miguel Botto-Tobar[c,d]

[a] *Departamento de Telecomunicaciones y Redes de Información, Escuela Politécnica Nacional, Quito, Ladrón de Guevara, 17051, Ecuador*

E-mail:[1]ana.zambrano@epn.edu.ec; [2]eduardo.ortiz@epn.edu.ec; [3]xavier.calderon@epn.edu.ec

[b]*Departamento de Telecomunicaciones, Universidad Técnica del Norte, Ibarra, Av 17 de Julio 5-21, 100110, Ecuador*
E-mail: omzambrano@utn.edu.ec

[c] *Eindhoven University of Technology, Eindhoven, The Netherlands*
E-mail: m.a.botto.tobar@tue.nl

[d] *University of Guayaquil, Cdla. Universitaria, Guayaquil, Ecuador*
E-mail: miguel.bottot@ug.edu.ec

***Abstract*— Managing traffic in a large city has become a topic of great interest in both politics and science. The costs of poor traffic management have been quantified as losses equal to millions of dollars, not counting the unquantifiable value of the time that a person loses in traffic jams. Intelligent transport systems (ITS) offer a set of innovative solutions specific to the management of different modes of transport. This article focuses on the development of an ITS for the city of Quito that allows smart decision-making to direct heavy haul transporters that want to enter the city via one of its main access routes. Technologies such as Sensor Web Enablement (SWE), in association with the Message Queuing Telemetry Transport (MQTT) communication protocol, facilitate the development of a vehicular management platform/system capable of sending notifications in real-time and issuing instructions to drivers regarding traffic delays along routes, average speeds, etc. The system supports a network of heterogeneous sensors accessible through the web. It can integrate any device that uses HTTP protocol. Time interval and location range testing have been undertaken to refine the accuracy of the system and make it adaptable to any geographic situation. The system allows communicate with the server through MQTT or through web services, using technologies such as: MongoDB and GeoJSON. One of the most relevant results is that the degree of accuracy of the system is within appropriate ranges when compared to commercial applications such as Google Maps and Waze.**

***Keywords*— internet of things; sensor web enablement; message queue telemetry transport; intelligent transport system; crowdsensing.**

I. INTRODUCTION

We are in the midst of the digital age and are witnesses to the exponential growth in the development and use of personal devices such as smartphones, tablets, the Rasberry, etc. Together, these devices constitute a potential network of heterogeneous sensors that enable the expansion of device-based information exchange, thus improving autonomous decision-making. From this digital environment, the IoT (Internet of Things) has emerged as a platform for interconnecting sets of autonomous digital systems to society, and collecting and providing information between end users [1]–[5]. The integration of heterogeneous sensors (sources of information) with data management systems has created intelligent environments capable of responding to all types of imminent risk [6]–[8]. Big cities are host sites of this new integration [9], [10]. In order to tackle issues arising from the need to manage urban environments, big cities have implemented a series of broadcasting technologies that transmit relevant notifications that benefit society at large. By adopting specified technologies as their preferred tools, these cities have entered the ranks of the smart cities. For example, using a set of sensors at the main access routes to a city allows it to predict the duration of intercity delays. These sensor networks are called SmartRoads [11]. Within this context, our study focuses on optimizing the flow of heavy transport into the city of Quito, a project we have identified as ESR–Q (Ecuador Smart Roads - Quito). Transportation is a key topic that plays a crucial role in commerce and industry. During the past several years, the number of vehicles in Ecuador has grown. Currently,

301,806 trucks with a minimum load-bearing capacity of 3.5 metric tons now circulate in Quito [12]. In 2016, traffic flow into the city increased by 31,761 vehicles. This amount jumped to 70,203 vehicles in 2017, representing a 121% growth rate [12]. At the same time, the capacity of the access roads to handle traffic flows into the most important cities in Ecuador has remained the same. Among the most representative effects of excess traffic flow are the inability to make deliveries and delays in delivery schedules. On a related topic, truck drivers in Ecuador are generally unaware of motor vehicle accidents, highway blockages due to landslides, and many other problems that complicate the traffic flow panorama.

In the city of Quito, the 2016 statistics [13] show that 213,932 vehicles entered the city by the various routes available. In 2017, there were 227,187 vehicles entered the city, representing a 6.19% increase in traffic flow. This problem entails a commute of between 1-2.5 hours for travelling between work and home [14]. The newspaper El Comercio reports that, on average, a person spends 28 hours per month stuck in traffic in Quito [15], equal to 336 hours or 14 days per year. However, among the alternatives that Quito's public officials have proposed to improve mobility, there is no mention of developing an Intelligent Transportation System (ITS) that takes advantage of the population's widespread access to smartphones. To address this problem, our research project proposes an ITS to manage the main access roads into Quito adapted to the city's unique characteristics (such as its challenging geography). This system broadcasts real-time notifications of traffic-related events on Quito's main arteries. It presents to the end user the most recent data about conditions on selected routes, such as average traffic speed, delay time, average traffic congestion (according to IMT, or International Mobile Telecommunications), and ETA (Estimated Time of Arrival). These parameters are assessed in Section III in order to verify their effectiveness. To complete this analysis, we used data obtained from heterogenous sensors available throughout the community (smartphones), along with electronic modules designed exclusively to fulfill prototype testing. Based on these data, we made improvements that improve drivers' access to information regarding the real-time conditions on favored routes, thus enhancing their ability to make better decisions regarding travel plans.

The prototype uses a distributed communications architecture that integrates heterogeneous devices and handles both data acquisition and results notification for end users. The system allows any user, regardless of device, to find, link to, and query any sensor in order to expand their sources of information. Utilizing a sensor network allows our system to cover a large area in which the system can then strategically pinpoint sensors in order to take measurements of other roadway conditions. In summary, our system has been designed to incorporate new sensors, regardless of whether they are components of mobile or fixed devices, and regardless of their software or hardware constraints. Incorporating new sensors increases data volume and therefore allows the system to make new estimates that help drivers navigate access into the city. According to our study, the more information is generated about roadway access, the better forecasts mimic actual roadway conditions—this trend is examined in Section III. The widening of the sensor network optimizes arrival times, and minimizes produce loss and travel operating costs.

Even though there are off-the-shelf applications that fulfill similar objectives, our project takes advantage of new ITS tools and can be customized to the needs of specific cities. Our pilot project produced minimal error percentages in regard to predicted times compared to measurements of actual highway conditions, and compared to the most commonly used WAZE (community-based GPS navigation) traffic management applications for commerce [16].

This article is divided into four sections. Section I (Introduction) focuses on the different angles to ITS implementation and synthesizes the general features of our research. It also describes the research projects aimed at developing proposals for transportation management that are most relevant to our project. The second section presents the technical architecture of our system (ESR-Q), focusing on the information, sensor system, and notification blocks used in this prototype. Next, it presents the results of trial run with the software developed. Finally, it offers the conclusions and follow-up research plans.

A. State-of-the-Art

We focused our study on heavy haul transport in Ecuador, since heavy haul drivers experience ongoing economic losses because they lack knowledge about roadway conditions [17]. For example, lack of knowledge about a landslide can result in a traffic jam that lasts hours and, consequently, leads to loss of merchandise and failure to meet the scheduled delivery. Ultimately, lack of situational awareness significantly affects the economic development of the country. Currently, the driver community uses social networks such as WhatsApp, Facebook, and Twitter to stay informed about roadway conditions. However, most notifications are decontextualized, that is, they are not real-time alerts, and are sometimes so out-of-date that they are misleading. By applying ITS technology, a system tries to reduce the impact that urban traffic has on a city [18], [19]. Several ITS-based traffic management projects have been developed [20]–[22]. In the state of Michigan (USA), various projects under the umbrella term, ITS-Michigan, are underway. Their goal is to improve urban safety and viability in Michigan [23]. One of the most characteristic projects uses video cameras and adaptive traffic lights to optimize and control intercity traffic. The Russian city of Moscow has created a group of related projects, called ITS-Russia, to address traffic management problems [24]. ITS-Russia used the latest advances in technology, such as outdoor camera detection, to identify and locate vehicles that have broken down, vehicles moving in the opposite direction to the traffic, etc. In contrast, Ecuador currently has no projects that are using traffic management technology, except for an isolated project in Ambato, Ecuador [25] that measures vehicular traffic by counting passing vehicles by means of cameras installed on city streets. The goal in Ambato is to determine the number of autos travelling along a certain route in order to optimize traffic light signal changes. However, for a developing country such as Ecuador, acquiring more cameras to cover a wider area is not feasible.

In contrast to the studies mentioned above, our project does not require a large budgetary investment, since we are basing our technology on a set of devices already deployed in our community, i.e. smartphones, that host the on-demand mobile application we have designed and developed.

There are off-the-shelf solutions, such as Waze, that address traffic management concerns. These platforms allow end user interaction via mobile applications. They can both collect end user data and transmit results. For example, Waze recompiles traffic data and classifies it into areas of interest. Therefore, it can transmit detailed traffic data in real-time to a community. Projects such as that of Banner and Orda [26] have examined the various algorithms that Waze uses in order to make traffic predictions, including the Bayesian Nash Equilibrium, Price of Anarchy (PoA [27]), and Price of Stability (PoS [26]). These algorithms are key to describing traffic behavior and transmitting relevant results to the community. In contrast, our research bases itself on concepts in the SWE-SOS heterogeneous standard, in combination with IoT MQTT protocol, that take advantage of their crowdsensing capabilities. Our research suggests that the more sensors there are in the system, the lower the percent error in the analyzed data (velocities, times, etc.) will be. (See Section III Results.) In short, the results of this research are on par with or even exceed results using Waze, an application adapted for global use. Another advantage that our application delivers is that it adapts to a wide variety of sensors; it is not limited to mobile applications only. Our project is customized and adjustable to conditions in Quito, whereas Waze might include outdated routes that are no longer available, especially given the high pace of change in developing countries, such as Ecuador.

Within this field of research, we have identified several projects that monitor vehicular traffic. One of the most prominent is the Collaborative System for Monitoring Vehicular Traffic developed by the Instituto Politécnico Nacional (IPN, or National Polytechnic Institute) in Mexico City [28]. This project addresses three challenging aspects of monitoring traffic. First, data must be collected from mobile users. Second, the data are stored in the Postgresql repository. Finally, the data are processed to identify the intensity of traffic congestion on selected routes in the city. Technologies employed in the solution include PostGresSql for data storage, Apache Tomcat for web service hosting, and Android Studio for mobile applications development. In contrast, we use a SWE-SOS standard interface, which accepts web messages from a wide variety of sensors, in such a way that it broadens the source base beyond just smartphones operating with Android [28].

Another project aimed at improving urban mobility through traffic monitoring is McGill University's "Towards a WIFI-Bluetooth system for traffic monitoring in different transportation facilities" [29]. This project measures traffic parameters such as travel time, average speed of travel, and traffic volume. It employs wireless technology, due to its low cost, and can collect large amounts of data in a short period of time. The system uses at least two Bluetooth sensors on major routes separated by a specified minimal distance to measure variable shifts in vehicle times. However, the project is limited by Bluetooth's maximum coverage—for Bluetooth 5.0, approximately 240 meters [30]—and by the number of sensors that can be connected to a single network. Regardless, the results closely approach real-time conditions. For example, for a designated section of the route, where the real speed was 27.34 km/h, the calculated speed was 28.33km/h. This calculation represents a low 3.49% margin of error. The limits of the system such as its range, however, make the project unscalable to larger cities such as Quito. Our current research project uses a standard web interface that gives users more flexibility, since sensors are not limited to certain geographical distances, as in the Lesani et al project [29]. Our system can have nearly ubiquitous reach, allowing it to send and receive data independent of the geographic location of sensors. Another research project known as "VANET based Real-Time Intelligent Transportation System" [31] takes advantage of the rapid growth of ad-hoc networks in vehicle network technology (VANET). These systems use RFID (Radio-Frequency Identification) and ARM (Advanced RISC machine) technology to display the least congested routes to drivers. But as is the case with the Lesain et al project [29], we must analyze the technological limits of this system. RFID, for example, has a restricted range that makes it unsuitable for use in remote locations such as interprovincial highways. Moreover, any improvements in measuring devices directly affect the cost of implementing a system. In contrast, in our project, display costs are nearly zero for users equipped with smartphones.

II. Material and Methods

ESR–Q is built on a client-server architecture in which the server uses a SWE-SOS interoperable web interface to manage the remote queries of heterogeneous clients. It also incorporates a REST software architecture style that uses HTTP (Hypertext Transfer Protocol) and JSON data type format for client-server messaging. We decided to use the latest version (2.0) of the SWE (Sensor Web Enablement) standard to implement the web interface, since it defines specifications in XML and JSON for Given our desire for a real-time system, we used the Message Queue Telemetry Transport (MQTT) as our main communications protocol for sending notifications to system users. MQTT manages both the sending and receiving of notification messages. To implement messaging, we installed a client MQTT on each device to give it notification capabilities. As an alternative, Arduino and Raspberry devices can be installed in certain vehicles in which drivers do not have smartphones, or where drivers are unable or unwilling to use mobile applications due to confidentiality concerns. (See Figure 1.)

ESR-Q is adapted for use on the major routes into Quito. Figure 2 identifies the routes targeted in our research. We have taken into consideration differences in flow on round-trip routes, e.g., traffic flow from Santo Domingo to Quito is not the same as flow from Quito to Santo Domingo. Otherwise, we would have client-server messaging that maintain compatibility with the REST architecture. To implement the messaging service, we used the SWE-O&M (SWE Observations and Measurements) standard. SWE-O&M uses a JSON schema to provide document data that allows objects and locations stored in the database to be georeferenced using spatial and temporal filters.

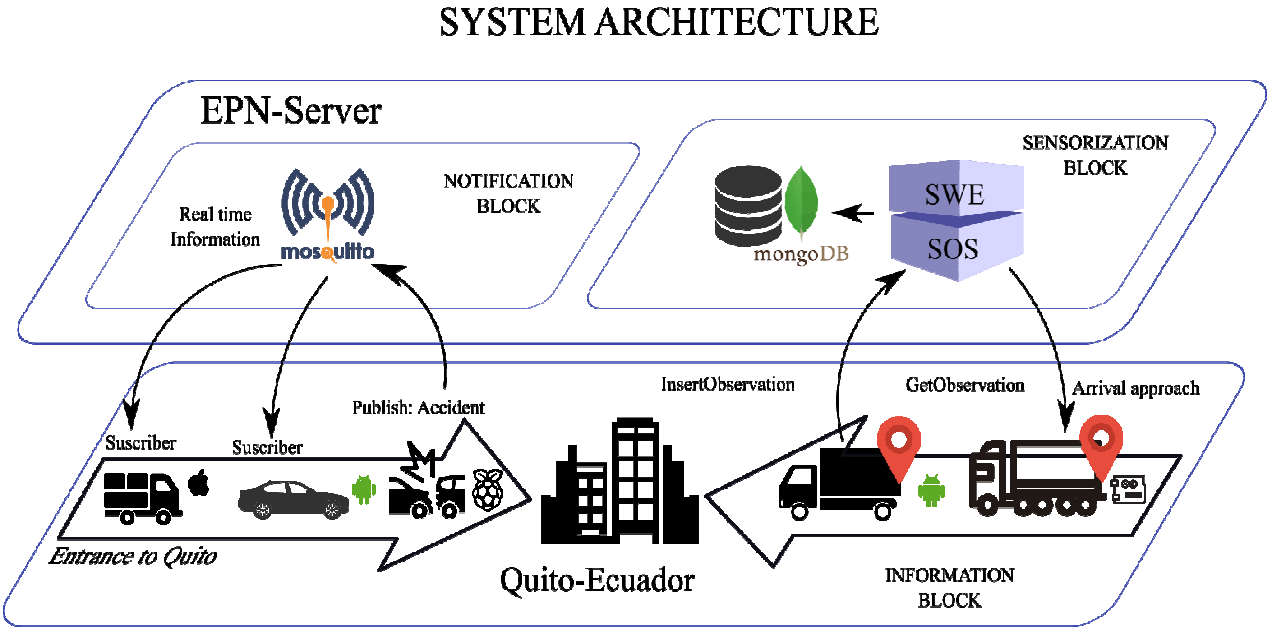
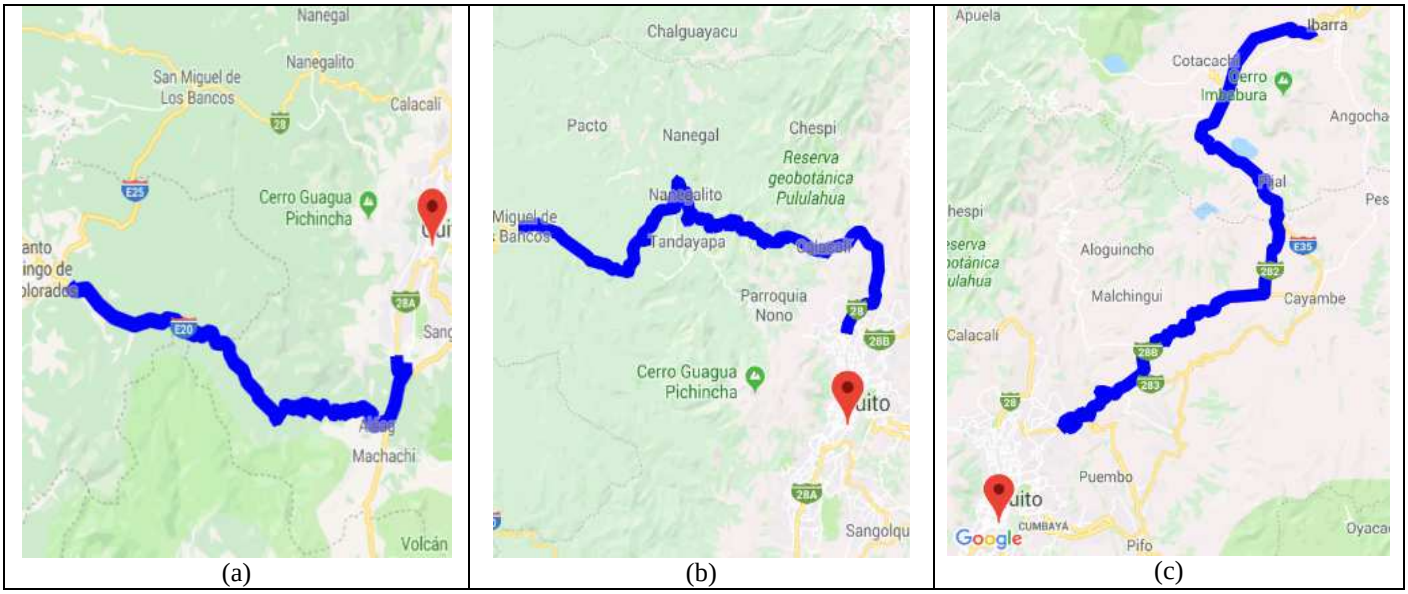


Fig. 1 System Architecture

Figure 1 shows the role that the web interface plays in the ESR-Q system. The system collects data from the sensors (smartphones, the Raspberry, and the Arduino), which represent system users that are travelling on the main access routes into Quito. These sensors automatically send their observations of average transit speed and location coordinates (in latitude and longitude) to the server.

Given our desire for a real-time system, we used the Message Queue Telemetry Transport (MQTT) as our main communications protocol for sending notifications to system users. MQTT manages both the sending and receiving of notification messages. To implement messaging, we installed a client MQTT on each device to give it notification capabilities. As an alternative, Arduino and Raspberry devices can be installed in certain vehicles in which drivers do not have smartphones, or where drivers are unable or unwilling to use mobile applications due to confidentiality concerns. (See Figure 1.)

ESR-Q is adapted for use on the major routes into Quito. Figure 2 identifies the routes targeted in our research. We have taken into consideration differences in flow on round-trip routes, e.g., traffic flow from Santo Domingo to Quito is not the same as flow from Quito to Santo Domingo. Otherwise, we would have seen inconsistencies in the traffic data returned by the sensors along the same route. The MQTT application uses the host to store the traffic parameters that are captured, such as average speeds and transit times. Thus, each driver becomes an information source for the system at zero cost.



(a) (b) (c)

Fig. 2 Sensorization Block

The system architecture is composed of three main blocks: The Information Block, the Sensorization Block, and the Notification Block. The combination of these three blocks gives the system the ability to be heterogeneous, opportunistic, and to operate in real-time. The following sections explain in detail how each of these blocks' functions.

A. *The Information Block*

This research project included both mobile and fixed devices as sources of information: smartphones, the Arduino Uno [32] and the Raspberry Pi 3 [33]. All these devices made onboard measurements of transit speeds and then sent these data to the main server. To transmit these data, we integrated a GPS (GY-GPS6MV2) [34] module into the Arduino and Raspberry devices in order to boost their georeferencing capabilities and thus enable them to determine transiting speeds. We also added a GSM 900 module [35], [36] for sending data to the remote server via the web. Both the Raspberry and the Arduino devices were installed in vehicles whose drivers did not have access to smartphones, or those who were reluctant to use a mobile application on their smartphone due to concerns about security and/or confidentiality. We should mention that we recaptured the energy needed to operate these devices from lost vehicle power.

The system captures two types of data/information. The first is the variation in average speed of the final user ($\Delta \vec{v}$), and the second is an informative message about traffic incidents along the route, for example, accidents, damaged vehicles, etc. The system calculates the speed in conjunction with the user's geographic location (latitude and longitude). First, the application determines the variation in distance travelled ($\Delta \vec{d}$), which are two coordinates taken over a defined change in time ($\Delta \vec{t}$), and is used as a configuration parameter for sensors such as smartphones, the Raspberry, or the Arduino. Combining these two data gives the transit speed, which is then sent to the server to be stored in a standard specified by SWE-SOS. The speed is calculated and automatically sent to the server with a period (T) of 1 minute in order to avoid draining the device's battery, as well as to capture data that better approximate real time. In contrast, the event notification message is sent manually by end user when she or he runs into a traffic problem on the roadway.

B. *The Sensor Block*

This block captures data from all kinds of devices in the system, not just from the mobile application. It uses an SWE-SOS web interface to mediate between the heterogeneous clients and the MongoDb data repository. The SWE-SOS standard allows the system to query registered sensors, roadway measurements, etc. It includes a database that, together with the SWE-SOS interface, registers the sensors along with their metadata. This register facilitates sensor identification and allows the system to simplify data search and filtering operations. The standardized SWE-SOS interface is described in Figure 3.

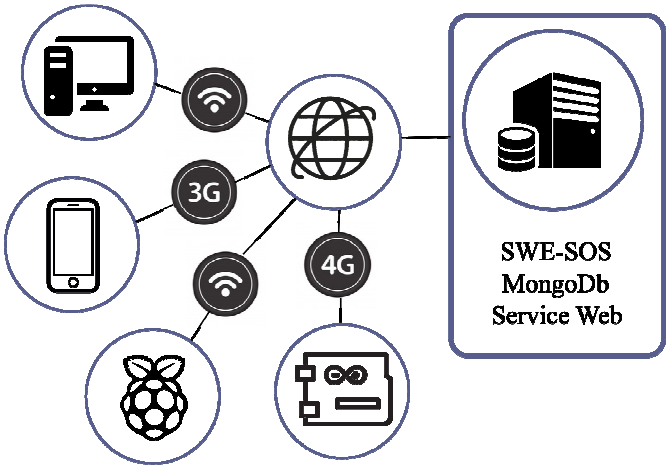


Fig 3. Sensorization Block

Figure 3 illustrates the standardized SWE-SOS interface, which is the main gateway to server resources. It manages remote sensor queries. Using PHP, we designed the SWE-SOS interface so that it could accept and process JSON messages according to the guidelines documented in OGC Observations and Measurement – JSON Implementation. This standard note that data should include metadata, such as geographic location. The web interface is stored on a physical server at EPN (Escuela Politècnica Nacional in Quito) that has been assigned a public IP address so that it can be accessed via the web, thus making the system nearly ubiquitous.

1) Sensor Web Enablement: The design for the standardized SWE-SO interface uses a series of client-server messages for data transfer.

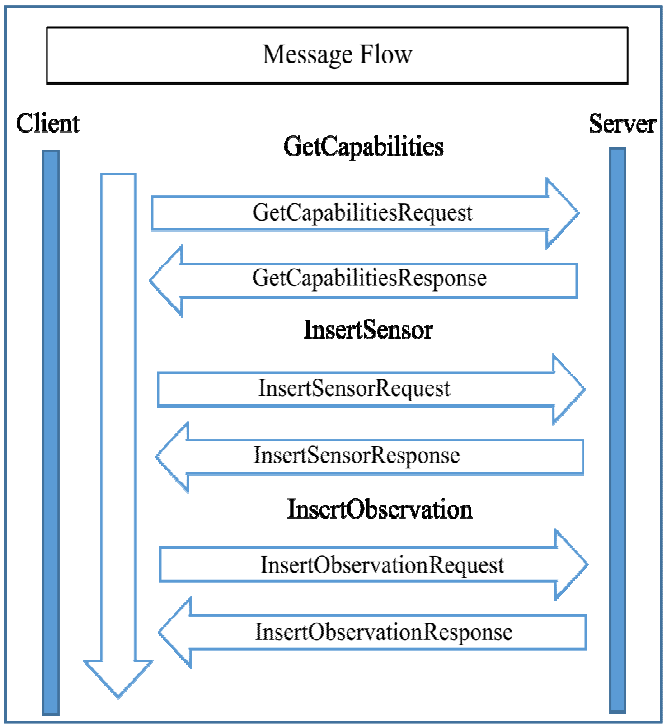


Fig 4. SOS Message Flow

Figure 4 outlines the implementation procedure for a new device (including hardware and software characteristics), i.e., how to register it in the system so that it can begin to send

observations to the server. SWE-SOS manages the observation format (observation), which is a property measured within the object under investigation, for example, transit velocity. This observation is a numeric value (int, double, float) that measures a specified property, for example, a vehicle's average speed. At the client end, sensors include any device that can carry out these measurements and send the results to the remote server.

Figure 4 illustrates the message relay. The first message sent is GetCapabilitiesRequest(), which is used to find the operations that the SWE-SOS implementation supports. These operations include insertSensor(), updateSensor(), deleteSensor(), insertObservation(), getObservation() and getCapabilities(). After identifying the SOS operations that are available, if the insertSensor() is available, the sensor is registered in the system using insertSensorRequest()—see Figure 4. If the operation is structured according to specifications, the data from that sensor is stored and its identification is transmitted. At this point, the sensor is enabled and can generate its own observations and send them to the communications server.

The operation GetCapabilities() displays the operations available to the sensor. If the InsertObservation() is available, a sensor can send data to the server. For this project, a device periodically calculates its average speed ($\Delta\vec{v}$). These speed data are then sent to the server in observation format. This set of observations is used to determine the average speed for the whole route ($\vec{V}$) and the average time of any delay. If needed, the users can manually send incident type observations to the server. An incident is any unfavorable happening that drivers experience such as vehicular accidents, landslides, road construction, or other problems.

The insertObservation() message registers the speed measured at a defined geographic location along the route. There are several main parameters needed to assemble a message of type observation. For example, the featureOfInterest property identifies the target object and, in this case, is assigned the value "SDaQ", representing the Santo- Domingo-to-Quito route. Likewise, procedure and offering procedures represent the physical identity of a device (MAC address). The OM_Observation property is a JSON object that contains the most important metadata of the main observation, including phenomenonTime, which registers the instant when the observation is made. We should note that date and time formats conform to the ISO 8601 standard [37], specifically year, month, day, and hour.

insertObservation() also includes a procedure that is used to identify the device that makes the measurement. The observation datum (speed or incident) is stored in result, which stores the final result of the roadway measurement. For example, if the measurement is 25km/h, the data type will be double with a value of 25 and units of measurement km/h. Or if the value has an incident type of "accident," the data type will be string with a value of "accident" without units of measurement. Result is a JSON object that contains the name of the observation, its data type, the units of measurement, and its value. Finally, the geographic location is stored in location for georeferencing purposes. Location identifies the exact site where the measurement is taken. These metadata allow users to understand the context of the observation, that is, the system can identify the place, the route number, sensor details, the exact time when the observation is made, etc.

2) Heterogeneous Communications: As Figure 5 illustrates, all devices have two ways to communicate with the server.

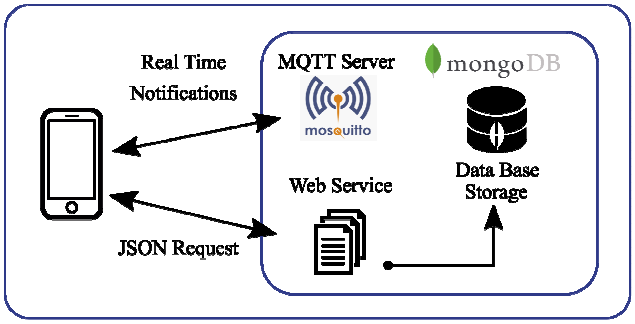


Fig 5. Communication pathways to the server

The first use MQTT server to receive messages in real-time. The second uses web services to store data in MongoDb; these data are stored on the same server. MongoDb provides massive data storage capability for mobile device observations, since it uses a flexible, NoSQL database manager for storage. To query observations, we used GeoJSON [38] format. GeoJSON is an open standard that allows systems to code geographic identifications with special features derived from JavaScript (JSON) notation. GeoJSON allows developers to differentiate between documents by geographic dimensions (circle, rectangle, or any polygon) circumscribed by the user. For example, with JSON the system can derive observations that are distant from a certain point-of-interest. We implemented time filters in order to request observations within time ranges specified by the user. By integrating GeoJSON with the time filters, we can create queries that represent observations about a certain section of a route over a given interval of time. The query results are processed by the mobile application and displayed to the user, along with average speed ($\Delta\vec{v}$).

Fig 6. Implementation of GeoJSON of MongoDB

Figure 6 illustrates the GeoJSON MongoDb implementation using spatial (rectangle) and a time filter of type combo Box, with options to select the times of the last observations. Inclusion of these features gives the ESR-Q system the real-time capabilities of intelligent transportation operations. By combining the SWE-SOS and MongoDb technology, we can create a sensorization block that can capture relevant roadway data and broadcast predictions during trips.

C. The Notification Block

The notification block sends real-time messages to devices linked into the ESR-Q system. One of its main components is the messaging system used to communicate with end user devices. The messaging function was implemented using the *Message Queue Telemetry Transport (MQTT)* communications protocol on the client side and the *Mosquito Broker* [39] server on the server side.

The real-time notification system is based on publish-subscribe architecture. This relationship is client-dependent; once an object's status is known, the object must register itself with a publisher in order to pass on the latest updates on events. The client receives notifications and messages from all objects to which it is subscribed, and thus "learns" the object's status. The MQTT communications protocol allows our system to split the communications channels into various subchannels that can be used according to a project's needs. In our case study, each communications channel was assigned to each of the available routes, that is, there are as many communications subchannels as there are routes into the city. For example, for the Santo-Domingo-to-Quito route, the subchannel is assigned the code "SDaQ". (For more details, see Table 1 in Section II.).

TABLE I
ACCESS ROUTES INTO QUITO

Route Identification	Point of Origin (city)	Destination Point (city)
SDaQ	Santo Domingo	Quito
QaSD	Quito	Santo Domingo
IaQ	Ibarra	Quito
QaI	Quito	Ibarra
LBaQ	Los Bancos	Quito
QaLB	Quito	Los Bancos

The subchannels referred to above are mapped to the Topics filter of MQTT. Topics identifies a communications channel from its alphanumeric string. (By the way, the string is hierarchical, that is, it divides communications media in a granular way.) Thus, the hierarchical string can be used to generate different notification systems for the same route. For example, we can create an emergency communications channel for the Santo-Domingo-to-Quito route and call it "SDaQ/Emergency." On this channel, we can classify emergencies according to type, e.g., "SDaQ/Emergency/ Accident." Using this schema, we can subdivide channels according to the system's design.

Figure 7 shows how a message circulates from a message publisher to subscribers. In order for a device to receive a message using the MQTT protocol, the device must enable an MQTT client that has the capability to establish a network connection with the Mosquito Broker [39]. The connection parameters must specify the listening port number, the username and password for the server, the service quality (QoS), a level for MQTT message delivery, and an SSL security certificate for channel encryption (if the server requests it). In addition, the connection must specify the message sending (PING) frequency—known as "Keep Alive"—for the server. MQTT uses this message to maintain a permanent network connection with Mosquito.

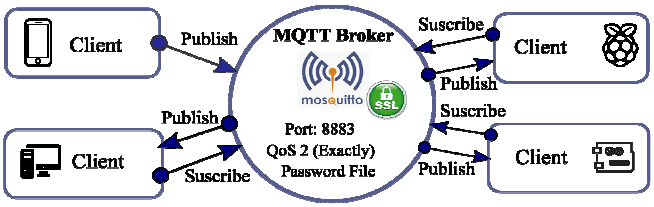


Fig 7. Real-time notification system.

As shown in Figure 7, there is no limit to the number of devices that can connect to the server, since the protocol accepts connects from all types of sensors (such as smartphones, the Raspberry, or the Arduino) that have the capability to implement an MQTT client and use a web connection to access the Mosquito server. We should point out that, in the future, we will be able to incorporate new sensors of types yet to be developed into this flexible system.

The main parameters to implement an MQTT client on a smartphone include the IP address of the MQTT server (broker) and the access credentials (username and password). The message service quality is 2 (QoS=2), which indicates that, for all messages sent, the protocol guarantees that the message will arrive exactly one time at the target. Once configured, the smartphone is enabled for publishing and receiving messages.

III. RESULTS AND DISCUSSION

The test scenario follows the steps outlined in Figure 2 of the Technical Architecture Section II. Figure 8 illustrate the Santo-Domingo-to-Quito route (SDaQ). This route is traced onto an interactive Google Map that the end user can navigate.

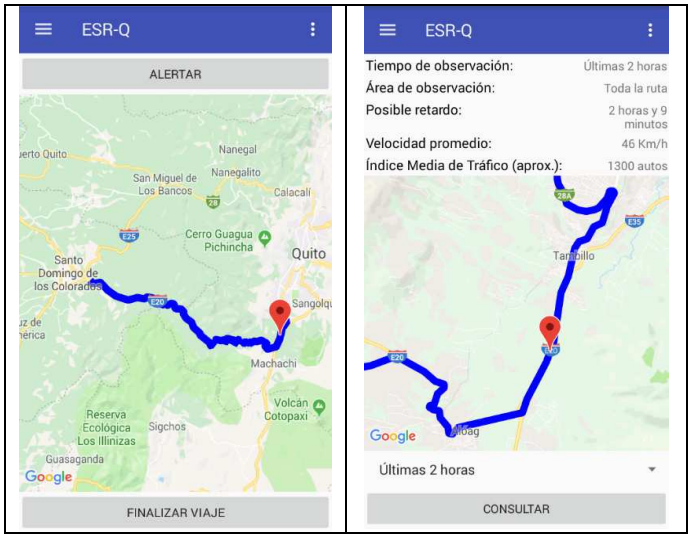


Fig 8. Details of the Route of entrance to Quito.

The map identifies any ongoing traffic incidents and corresponding relevant data, such as the average transit speed on the route (km/h), delay time (in hours: minutes), and the (approximate) Average Traffic Density (IMT, or Intensidad Media de Tráfico in Spanish)—that is, the average number of vehicles on the route. There are three routes registered in the ESR-Q system. Details can be found in Table 1 of the System Architecture section.

The ESR-Q system is based on requirements specified by the heavy transportation companies associated with FENATRAPE (Federación Nacional de Transporte Pesado del Ecuador, or The National Federation of Heavy Transportation of Ecuador) [40]. For the test phase, we contacted two Ecuadorian heavy haul enterprises through this federation. We wanted each truck in their fleet to become a separate data source using the modules we developed for the Raspberry and Arduino, or through an application installed on the mobile device (smartphone) of each driver. Our sample included 50 vehicles tested over a period of 30 days. Our goal was first, to determine if the predictive values generated by the system were within acceptable ranges of true values and, second, to verify whether the instantaneous messaging system enhanced driver awareness about roadway conditions so that they could make better decisions in time. We tested our system progressively, that is, we started with a sample of five drivers in order to validate errors and improve the ITS application After that, we expanded the test to 10 drivers, then continued expanding until we included all 50 truckers. The data were then tabulated for analysis. We calculated the percent error deviation from real conditions. Then the data obtained from the ESR-R system were compared to data from the commercial applications, Waze and Google Maps, in order to verify whether the ESR-Q system's degree of accuracy fell within expected ranges.

We also evaluated the performance of the database server's hardware and the mobile application. We needed to verify that host resources could handle operations, e.g., that there was enough RAM memory, processing power, and bandwidth for operability. The results are presents in the next section (4.1)

A. System Performance

In this section, we analyze the feasibility of the ESR-Q system. That is, we discuss our results in terms of our predictions for average speed and delay times in comparison with real conditions. Table 2 shows the results for the tests undertaken on the entry routes into Quito.

TABLE II
% ERROR FOR PREDICTED TIME VS ACTUAL TIME

	SDaQ	IaQ	LBaQ
Predicted time (min)	174,96	125,46	151,44
Actual time (min)	182,25	123	157,74
% Error	4%	2%	4%

As one can see, predicted delay times are congruent with real delay times. Also, the percent error between the predicted measurement and the actual one is minimal, which indicates that the system under development fulfills drivers' expectations. We can also say that the test results match times predicted by the commercial applications that are widely available. Once can even say they are better, since Waze, for example, is a non-flexible application, in the sense that its route catalogue is not up-to-date, nor can it offer customized services.

We have been able to validate, as seen in Table 2, that predicted times closely match actual times, since the percent error is low. Undoubtable, the percent error depends on crowdsensing scale. That is, the more observations are available as input to the calculations, the closer the projected times will approach actual times. It is quite different to base predictions on dozens of user observations versus thousands of observations. The more that data accurately reflect true roadway conditions from different perspectives, the more we can will significantly reduce possible sampling errors. We should emphasize that that the ESR-Q system can link to any roadway incident, such as accidents or landslides, since it captures the average speed of every device under any circumstance in real time. For example, when traffic is intense, the inserted observations will record a low average speed (<10 Km/h), which is reflected in the delay time on the route.

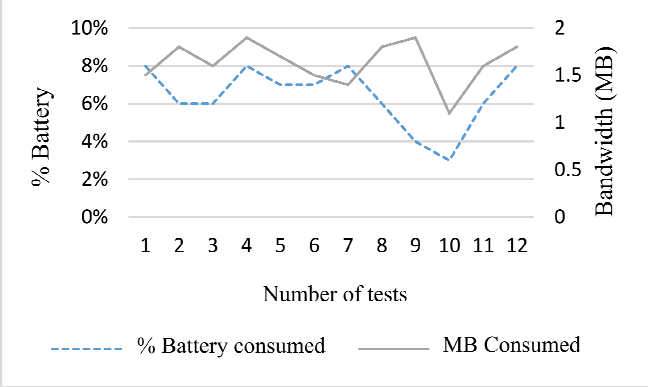


Fig 9. Resources consumed by the mobile application

As Figure 9 shows, the application uses a minimal level of resources; bandwidth use does not exceed 2 megabytes for a two-and-a-half-hour trip, indicating that the application consumes a minimal amount of resources. Resource consumption is an important consideration for this application, since excessive use of the system's energy or storage resources can cause the mobile application to deinstall, and the Rasberry and Adruino platforms to be disconnected. We should emphasize that both the Raspberry and the Arduino devices are permanently connected to a vehicle's power outlets, which eliminates any risk to the device's autonomy.

Battery longevity, on the other hand, is directly related to the amount of data sent to the server. If the rate of data transmission increases, battery consumption also increases. Before data is sent from the server, the mobile application uses a device's sensors to determine the average speed of host transmission. The most frequently tapped sensor is the global Geo Positioning System (GPS), since the GPS is periodically measuring geographic location in order to determine average speed. Our project, therefore, proposes that drivers use either the Raspberry o Arduino devices as an alternative to the mobile application, since we noticed some

reluctance in some end users to use the mobile application because of its resource consumption and/or lack of privacy or confidentiality.

Both battery consumption and storage space are determinant factors for end users. A real-time application will consume host resources in order to keep the client informed of system events. Smartphones need to establish permanent connections with the Mosquito server in order to receive real-time notifications. Figure 9 shows the minimal levels of resource consumption on a mobile application. The results show that resource consumption does not have a significant impact on devices. Our anonymous questionnaire to drivers revealed that they did not experience any type of technical problem with the mobile application, nor did they experience excessive loss of storage space. They noted, on the contrary, that the mobile application was intuitive and easy to use.

B. Server Performance

We conducted performance tests on the data server in order to determine the number of transactions that it could handle. We used JMeter [41] software to simulate a large flow of queries to the server. These tests were conducted progressively, that is, the number of connections was increased gradually by 1,000 queries per pass. We began with 1,000 simultaneous requests, and reached a maximum of 20,000 requests. Results are given in Figure 10. The server performed according to expectations.

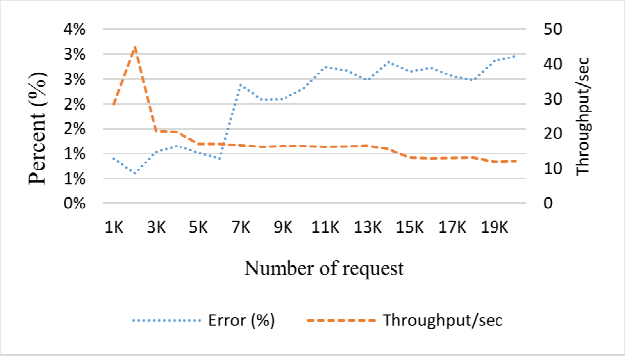


Fig 10. Data server performance

For the initial tests, the percent error was small, and when the number of queries was increased, server performance declined gradually. Since the error rate was not exponential, we were able to adjust server hardware resources to the number of clients that needed to be managed on the system.

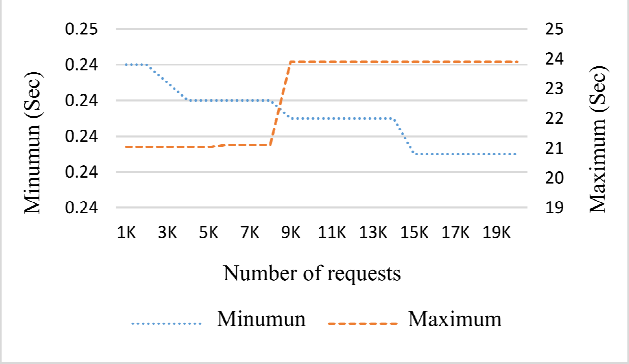


Fig 11. Maximum and minimum times for server request

Figure 11 illustrates how much time a server request may be delayed. Results are given in milliseconds; the maximum wait time for a server response falls within normal tolerances for an end user. Slow system response could cause users to stop using the ESR-Q system, which would affect the system's crowdsensing scale. Delays in information processing can also diminish the system's ability to mimic real time performance. During the test execution, the system maintained optimal conditions for end users. These test show that the server is capable of handling massive amounts of simultaneous queries through the SOS web interface. The number of requests is given in increments of ten thousand, which offer an advantage to deployment in the general community.

In addition, server performance does not degrade significantly when the number of queries increases. This indicates that the server maintains its capacity to handle real-time responses for many users. We also monitored the Mosquito server, focusing on the service implemented on the server at EPN (Escuela Politécnica Nacional), in order to determine how much RAM, the server uses to handle one client-server connection. We found that Mosquito needs about 5KB to manage a network connection, as shown in Figure 12.

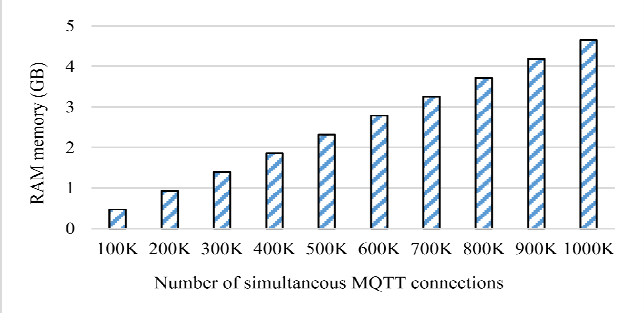


Fig 12. Memory required for MQTT connections

These data imply that, for a server with approximately 2GB of RAM memory, Mosquito has the capacity to handle 430,000 simultaneous network connections. These data indicate that a Mosquito-based implementation would have a light footprint, and the system would not need lots of hardware resources to handle thousands of network connections. Ultimately, the physical characteristics of the server will define the limits of the application solution.

C. Mobile Application Performance

The performance of the mobile client was evaluated, comparing it to other mobile applications with global distribution such as Waze and Google Maps. Our objective was to validate our proposal against two heavyweight commercial transportation applications networked for smartphones. We should point out that all types of sensor-based devices, not just smartphones, can use the ESR-Q system, which gives it an advantage over these other mobile systems. This adaptability implies that it is theoretically possible to add any type of sensor to the system and capture all types of measurements, e.g., asphalt temperature, humidity, wind speed, precipitation levels, etc. Adaptability allow the ESR-Q system to scale modularly with time. ESR-Q also avoids being limited to only one type of data.

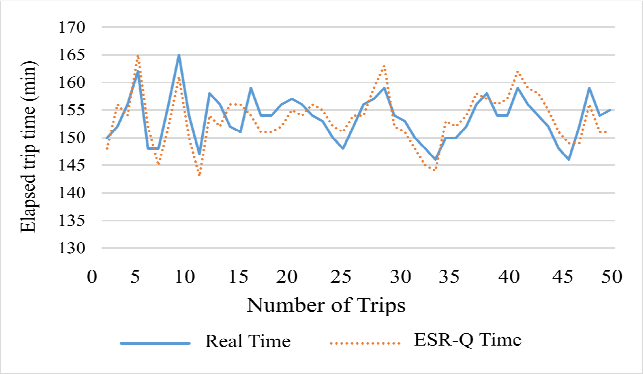


Fig 13. Predicted time vs. real-time on Route SDaQ

Figure 13 shows a comparison between times predicted by the ESR-Q systems and the actual time needed to complete one route. The graph makes clear the fact that the times coincide closely, proving that our prototype simulates real conditions and can be reliably used by end users. As shown in Figure 14, the ESR-Q system produces results that are comparable to those produced by Waze and Google Maps. Our tests were undertaken with a sample of 50 users who completed 90 trips in 30 days.

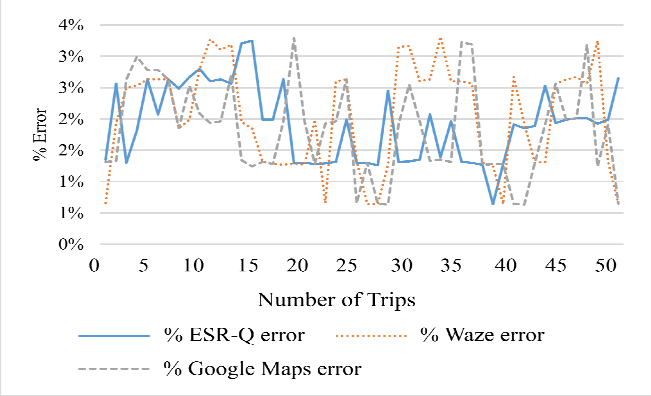


Fig 14. % Error between predicted time and actual time on the SDaQ route

The times forecasted by ESR-Q closely approximate the actual time taken to complete one trip into Quito. The percent error between real and forecasted time is less than 4% for a trip that takes 150 minutes. Clearly, these results will improve with denser crowdsensing. Increasing the number of users will increase the amount of data sent to the server. These data, in turn, feed back into the calculations of delay times on each of the routes.

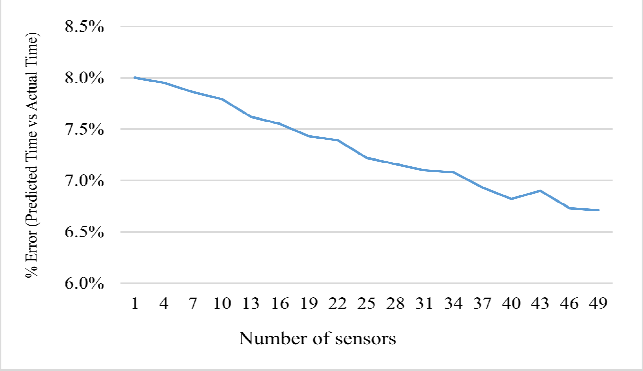


Fig 15. % Error vs Number of sensors

Figure 15 shows the effect of crowdsensing on the system. Basically, the greater the number of sensors in the system, the more the percent error between forecasted time vs. actual time decreases. Nevertheless, towards the end of the test cycle, the percent error no longer decreased, indicating that there remains a minimal percent error associated with the forecasting system. But we can expect that when the number of users is significantly high (in the tens of thousands), the percent error for minimal tolerance will also decrease.

To measure the performance of the mobile application, we used the application App tune-up kit (or similar) [42], which monitors an application's performance in terms of power consumed, percent CPU used, and percent GPU used. Mobile application performance was measured in two modes: first, when the application was used in the foreground (intensive use), and second, with the application running in the background, i.e., with only MQTT and the sensorization block actively waiting for MQTT protocol messages. In the second mode, the end user is not interacting with the system. Testing was conducted on the drivers' smartphones, meaning that the efficiency and performance of these devices depended on how long they were used, as well as their maintenance and care.

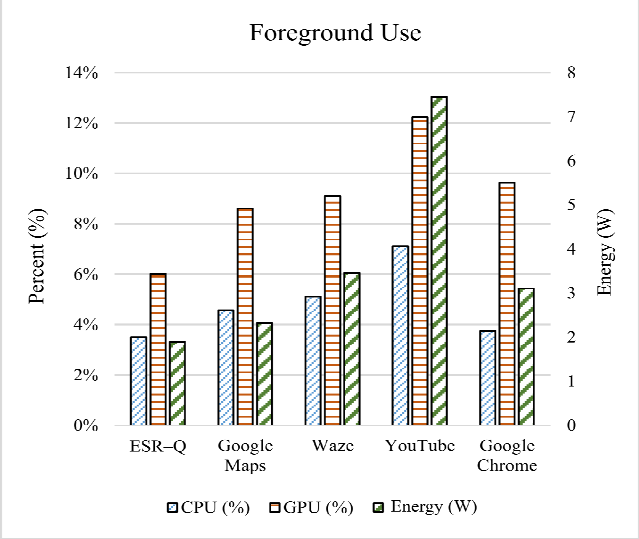


Fig 16. Use of hardware resources when application is foregrounded

Figure 16 depicts performance when the mobile application is used in the foreground, with energy consumption measured in Watts (W). ESR-Q did not exceed 2W of energy use, implying that our application will not significantly drain a smartphone's battery. In terms of consuming CPU and GPU resources, energy use is around 4% and 6%, respectively. These latter figures show that hardware resource consumption is minimal in comparison with the other applications. We should underscore the fact that this consumption relates to patterns of foregrounded use, i.e., times when the user is interacting intensively with the application.

Figure 17 show hardware resource consumption while the application is running in background mode. This is a measurement of background processes that remain active, specifically, real-time reception of messages and delivery of observations to the data server. These modules were

programmed as services (service) on the Android platform, so they operate independently of the mobile application's main thread. In short, these background processes keep the server informed of the relevant data used to describe the route, and they maintain a permanent host connection for receiving MQTT messages.

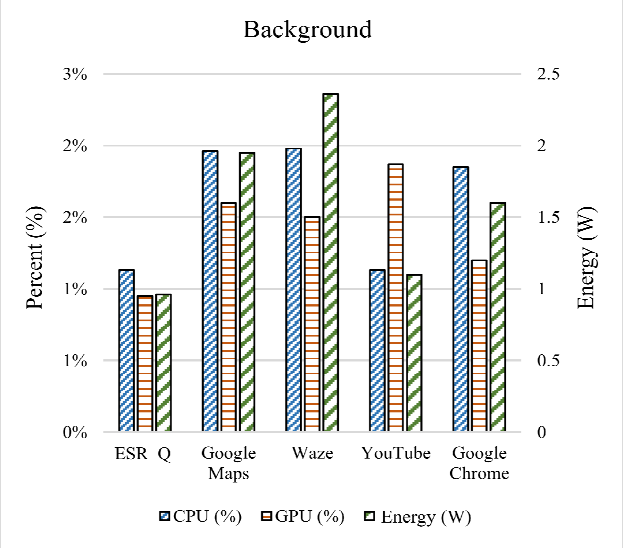


Fig 17. Use of hardware resources when application is backgrounded

Comparing Figure 17 (foreground use) to Figure 16 (background use), we see that ESR-Q uses far fewer resources when running in background mode. For example, energy consumption falls below 1 W, and CPU and CPU resource consumption remain below 1. In both modes, consumption is below that of the social networking applications. The results indicate that our application will not significantly or negatively impact either the end user's economic resources or his or her hardware resources.

Overall, the results show that our platform, developed to manage traffic in the city of Quito, can be successfully integrated into its heavy transportation system. Drivers in Quito have remarked that they do not have any other software package with these characteristics that alerts them about the status of the main access routes into the city. Survey participants also noted that the data that are available via social networking applications is mostly decontextualized, thus engendering distrust in users and doubts about receiving true information based on real circumstances.

The results obtained from our integration testing parallel those derived from actual conditions that drivers experience when trying to enter Quito. The ESR–Q system competitively in communications server. In comparison with the other applications mentioned, like the state-of-the-art systems ITS-Michigan or ITS-Rusia, our solution costs very little to implement, thus making it an attractive option for traffic managers in other cities. In contrast to the research projects undertaken by Olvera [28], Lesani et al [29], and Pallavi and Satone [31], our project uses a standard web interface based on SWE-SOS for data capture, thus making it much easier to simultaneously collect data and metadata. Under test conditions, our pilot project registered low percentages of error [29], even for a roadway that is hundreds of kilometers long, compared to the roadway studied by Lesani et al [29], which only incorporated a few hundred feet. Furthermore, our solution offers a notification system in real-time that keeps users informed of changing conditions along the route. Real-time capability improves the situational awareness of drivers, thus also improving the decisions that drivers make.

IV. Conclusion

Cities whose populations and vehicular traffic flows are increasing should implement ITS (Intelligent Transportation Systems) that allow them to balance traffic flows and improve mobility throughout these cities. Quito, for example, has experienced a disproportionate growth in parking lots, which has encouraged even more traffic and subsequently increased travel times within the city. Our research offers an ITS solution for Quito called ESR–Q. ESR-Q issues real-time notifications to drivers about traffic problems occurring on roadways. It also broadcasts data on roadway conditions such as average vehicular speed, approximate delay times, and Average Traffic Density (ATD). With these data, drivers can develop and enhance a reality-based, situational awareness that lets them make timely decisions and optimize their operating costs for trips to and from Quito. The ESR-Q system meets the needs of the heavy haul drivers who are likely enter Quito along its major routes.

The technology employed for our pilot project allows developers to integrate all types of heterogeneous sensors into the system, since the application is based on an SWE-SOS standard that includes a web interface for handling data interoperability between all types of devices. A real-time platform for traffic notifications that uses MQTT messaging services was implemented. This configuration allowed us to send notification messages to everyone in the driver community. Clearly, enhanced community awareness increases overall situational awareness. Now end users can always be informed of roadway conditions and in all places.

Our use of a non-relational database such as MongoDb increases the communications system's flexibility, since it is possible to store observations in SOS format directly into the database. The database's GeoJSON capability allows users to query the database using geospatial and time-based filters. This means that the system can obtain observations for s specific place over a specific interval of time.

Our systems testing was performed on a sample of 50 users who completed 90 trips on major access routes into Quito. Tests were carried out over a 30-day period, a enough time to make improvements to the system based on the truckers' expressed needs. We used the data generated by the system to successfully optimize the budgets assigned to operating costs for travel to Quito. The trucker community (test base) affirmed that, during the test phase, our application met their expectations and successfully gave them useful information about roadway conditions and alerted them to traffic problems on the route before they were directly encountered, thus fulfilling the goal of raising situational awareness on the road. Ecuador does not have an ITS system for vehicular management; this software really represents a pioneering effort to solve traffic management in this country.

The test results confirm that we have developed an intelligent transportation system for Quito, Ecuador that is capable of broadcasting relevant data that assists drivers to optimize their decision-making and thus reduce operating costs related to long haul transportation. It should be noted that the forecasts delivered by the system have an error rate of 3.21% compared to actual roadway times. We have noted that when the number of sensors incorporated into the system increases, the closer the forecasting approaches real-time conditions, which affirms that crowdsensing density plays a fundamental role in system deployment. In addition, we have presented evidence that shows that results from our system are comparable to results obtained from commercial applications such as Google Maps and Waze. To summarize, this system fulfills driver expectations for predicting arrival times and for sending real-time notifications that alert drivers about any type of incident that affects traffic flow, thus aiding drivers in planning their trips. This application has proven to be a viable and useful tool to meet logistical challenges.

The performance of our ESR-Q application was optimal. Test results show that demand on device resources, including the battery, CPU, and GPU, is less than that for commonly used mobile applications such as Google Maps and Waze. Regarding Google Maps, ESR-Q makes 0.8% less demand on device CPU and 2.1% less on the GPU. In comparison to Waze, ESR-Q makes 1% less demand on CPU and 2.5% less on GPU. In terms of energy consumption (battery), the ESR-Q application uses less than 2W for a 1 hour 45-minute trip (approximately). We can thus conclude that our system does not make intense demands on the hardware resources of the host device. These are useful conclusions for developers, who know that straining hardware resources can cause N application to deinstall itself, and thus bring down the whole communications system.

In the future, we plan to incorporate new technology into our system related to data visualization. For example, we want to research an implementation that uses Lambda Architecture [40], in which data processing operations run in parallel with data production. In our case, notification alerts on roadway changes would be transmitted while data was being generated. In addition, we want to implement other features offered by the MQTT protocol, such as hierarchies for communications media and connectivity to remote MQTT servers, in order to eliminate any potential single points of failure in the system.

ACKNOWLEDGMENTS

We express our gratitude to the Escuela Politécnica Nacional (National Polytechnic Institute) for financing the following project: PIJ 15-20 “E-iRoads: Ecuador - Intelligent Roads. Un Sistema inteligente para la gestión de tráfico en las periferias de grandes ciudades (Caso de Estudio: Quito)” [“E-iRoads (Intelligent Roads) in Ecuador: An Intelligent System for Traffic Management on the Outskirts of Large Cities. The Quito Case Study”].

REFERENCES

[1] S. Singh and N. Singh, “Internet of Things (IoT): Security challenges, business opportunities & reference architecture for E-commerce,” in *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, 2015, pp. 1577–1581.

[2] R. Gunasagaran *et al.*, “Internet of things: Sensor to sensor communication,” in *2015 IEEE SENSORS*, 2015, pp. 1–4.

[3] G. B. Satrya, H. T. Reda, K. J. Woo, P. T. Daely, S. Y. Shin, and S. Chae, “IoT and Public Weather Data Based Monitoring & Control Software Development for Variable Color Temperature LED Street Lights,” *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 7, no. 2, p. 366, Apr. 2017.

[4] A. I. Niculescu, B. Wadhwa, and E. Quek, “Smart City Technologies: Design and Evaluation of An Intelligent Driving Assistant for Smart Parking,” *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 6, no. 6, p. 1096, Dec. 2016.

[5] L. Benny and P. K. Soori, “Prototype of Parking Finder Application for Intelligent Parking System,” *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 7, no. 4, p. 1185, Aug. 2017.

[6] S. Kaur, S. Jain, and D. Virmani, “Deployment of Wireless Sensor Networks for intelligent information retrieval in marine environment,” in *2015 IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, 2015, pp. 371–376.

[7] Q. Wang, J. Zheng, H. Xu, B. Xu, and R. Chen, “Roadside Magnetic Sensor System for Vehicle Detection in Urban Environments,” *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 5, pp. 1365–1374, May 2018.

[8] Y. Liang, X. Meng, Y. Hu, and K. Zhang, “Design and implementation of an ultra-low power wireless sensor network for indoor environment monitoring,” in *2017 IEEE 17th International Conference on Communication Technology (ICCT)*, 2017, pp. 937–940.

[9] B. S. Leelar, E. S. Shivaleela, and T. Srinivas, “Cognitive Sensing in Smart Cities Using Optical Sensors,” in *2015 International Conference on Advanced Computing and Communications (ADCOM)*, 2015, pp. 13–15.

[10] V. H. Gonzalez-Jaramillo, “Tutorial: Internet of Things and the upcoming wireless sensor networks related with the use of big data in mapping services; issues of smart cities,” in *2016 Third International Conference on eDemocracy & eGovernment (ICEDEG)*, 2016, pp. 5–6.

[11] T. Nguyen, “Ahead of the Curb: Smart Roads,” in *2018 IEEE International Smart Cities Conference (ISC2)*, 2018, pp. 1–2.

[12] AEADE, “Sector Automotor en Cifras,” 2018. .

[13] R. Mena, “Análisis, caracterización y simulación del transporte de vehículos de carga pesada (caso de estudio: Quito),” Escuela Politécnica Nacional, 2018.

[14] G. Coba, “Dos ciudades ecuatorianas entre las 100 urbes principales con más horas perdidas en el tráfico,” *El Comercio*, Quito - Ecuador, 2017.

[15] D. Bravo and A. Carvajal, “¿Cuántas horas al año pasan los quiteños atascados en el tráfico?,” *El Comercio*, Quito - Ecuador, 2018.

[16] Waze, “Waze official page,” 2019. .

[17] I. Thomson and A. Bull, *La congestión del tránsito urbano: causas y consecuencias ecnonómicas y sociales*. Santiago de Chile: United Nations, 2001.

[18] R. Sanchez-Iborra, J. F. Ingles-Romero, G. Domenech-Asensi, J. L. Moreno-Cegarra, and M.-D. Cano, “Proactive Intelligent System for Optimizing Traffic Signaling,” in *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, 2016, pp. 544–551.

[19] G. Baban, A. Iovanovici, C. Cosariu, and L. Prodan, “Determination of the critical congestion point in urban traffic networks: A case study,” in *2017 IEEE 14th International Scientific Conference on Informatics*, 2017, pp. 18–23.

[20] A. El Mrini and A. Ghacham Amrani, “Wireless Sensors Network for Traffic surveillance and management in Smart Cities,” *MATEC Web Conf.*, vol. 200, p. 00024, Sep. 2018.

[21] F. Zhu, Z. Li, S. Chen, and G. Xiong, “Parallel Transportation Management and Control System and Its Applications in Building Smart Cities,” *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 6, pp. 1576–1585, Jun. 2016.

[22] A. Dubey, M. Lakhani, S. Dave, and J. J. Patoliya, “Internet of Things based adaptive traffic management system as a part of Intelligent Transportation System (ITS),” in *2017 International Conference on Soft Computing and its Engineering Applications (icSoftComp)*, 2017, pp. 1–6.

[23] ITS-Michigan, “Sociedad Inteligente de Transporte de Michigan,” 2018. .
[24] ITS-Rusia, “Intelligent Transport Systems of Russia.” .
[25] P. Mendez, “Red privada virtual como alternativa para el respaldo de información digital en el ilustre municipio de Baños,” UNIANDES, 2017.
[26] R. Banner and A. Orda, “Bottleneck Routing Games in Communication Networks,” *IEEE J. Sel. Areas Commun.*, vol. 25, no. 6, pp. 1173–1179, Aug. 2007.
[27] H. Youn, M. T. Gastner, and H. Jeong, “Price of Anarchy in Transportation Networks: Efficiency and Optimality Control,” *Phys. Rev. Lett.*, vol. 101, no. 12, p. 128701, Sep. 2008.
[28] X. Olvera, “Sistema colaborativo para el monitoreo de tráfico vehicular,” Instituto Politécnico Nacional, 2014.
[29] A. Lesani, S. Jackson, and L. Miranda-Moreno, “Towards a WIFI--Bluetooth system for traffic monitoring in different transportation facilities.” McGil University.
[30] Bluetooth official page, “Bluetooth 5 | Bluetooth Technology Website.” .
[31] P. A. and M. P., “VANET based Real-Time Intelligent Transportation System,” *Int. J. Comput. Appl.*, vol. 145, no. 4, pp. 34–38, Jul. 2016.
[32] Arduino official website, “Arduino UNO R3,” 2014.
[33] Raspberry official website, “Raspberry Pi 3.”.
[34] Synacorp Trading & Services, “Arduino GY-NEO6MV2 GPS Module c/w Antenna & Flight Control EEPROM.”.
[35] Rhydo Technologies (P) Ltd, “SIM 900-RS232 GSM/GPRS Modem User Manual,” 2011.
[36] Itead Studio, “Raspberry PI GSM Datasheet,” 2013.
[37] ISO, “ISO 8601 Date and time format.”.
[38] MongoDB Manual, “GeoJSON Objects.”.
[39] Eclipse Mosquito official page, “Eclipse Mosquito,” 2018.
[40] FENATRAPE official page, “FENATRAPE.”.
[41] Apache JMeterTM official page, “Apache JMeter.”.
[42] Qualcomm Developer Network, “App Tune-up Kit.”.