SensingBus: Using Bus Lines and Fog Computing for Smart-Sensing the City

Pedro Cruz
Universidade Federal do Rio de Janeiro

Felipe F. da Silva
Universidade Federal do Rio de Janeiro

Roberto G. Pacheco
Universidade do Estado do Rio de Janeiro

Rodrigo S. Couto
Universidade do Estado do Rio de Janeiro

Pedro B. Velloso
Universidade Federal do Rio de Janeiro

Miguel Elias M. Campista
Universidade Federal do Rio de Janeiro

Luís Henrique M.K. Costa
Universidade Federal do Rio de Janeiro

Collecting data is an important task for building smart cities. This article proposes SensingBus, a system to collect data from sensors carried by urban buses. Using buses to move sensors allows each node to cover a wider area, at a negligible cost. SensingBus is based on a three-level architecture. At the first level, sensing nodes collect and send data to the second level, consisting of fog nodes. The fog nodes preprocess data and deliver it to the third level, the cloud infrastructure, which stores and makes data externally available. The fog infrastructure, on the other hand, discards defective data, compresses information, and provides secure access points between the fog and the cloud. To validate SensingBus, the authors built a prototype and performed experiments to stress the fog nodes. They verified that each fog node can serve at least 20 simultaneous sensing nodes, an adequate number to sense a city such as Rio de Janeiro.

Smart cities aim at improving the quality of life of citizens by providing new public services or enhancing the efficiency of existing ones. The key idea consists of monitoring several aspects of the city, such as traffic, weather, and air quality, and using the collected information as the fundamental input to intelligent applications.

Copublished by the IEEE CS and IEEE ComSoc
2325-6095/18/$33.00 USD ©2018 IEEE

For example, consider a typical public service: street lighting. Municipalities are interested in keeping the number of faulty light poles as low as possible, because there is a relationship between dark areas and crime, and other problems such as car accidents. Hence, a smart city can use sensing devices to monitor light poles and detect problems in them. Another promising application is to predict flooding caused by storms and tell citizens to avoid areas that could be affected.

The Internet of Things (IoT) paradigm can be used to collect the data of interest for each smart-city application. The basic principle of the IoT consists of adding communication, processing, and sensing capabilities to everyday objects.[1] Nevertheless, to provide effective sensing in smart cities, one has to spread IoT devices over large geographic regions. Such a strategy might prove prohibitively expensive. For example, in the case of street lighting, every light pole would be equipped with sensors and communication interfaces. For a big city with several thousands of light poles distributed over a large area, adding IoT devices to every light pole might impose significant deployment and maintenance costs.

One alternative is to apply a mobile wireless sensor network that covers the entire region of interest. As sensors move through the region, the coverage area is enlarged, avoiding additional costs for IoT devices. This approach, called *mobile sensing* in the literature, creates a tradeoff between mobility cost and the time to cover a given area.[2]

One way of granting mobility to sensors is to embed them into buses, in line with the IoT paradigm. The advantage of using buses to transport sensors is threefold. First, buses cover a significant area of cities. Second, the additional cost of carrying IoT devices in buses is negligible. Third, the route of a given bus line is generally the same, with reasonably regular intervals, providing predictable coverage. Moreover, bus lines usually present some itinerary overlapping, which means that one location might be on the route of several buses. Thus, several measurements of this location can be collected, increasing the sensing reliability. In the example of street lighting, a single luminosity sensor in a bus can monitor the quality of all light poles on its route. Because employing a single mobile sensor can be unreliable, different buses passing near a given light pole can measure its luminosity quality.

Typically, IoT devices have limited processing and storage capabilities, while smart-city applications collect and analyze a large amount of data. Thus, a solution to store and process the collected data is to use a cloud-computing infrastructure.[3] IoT devices are responsible for sensing and actuation, while heavier processing tasks are performed in the cloud, which has more processing power and is able to gather all the data collected in a distributed fashion.

A major concern regarding this approach is the network traffic exchanged between devices and the cloud, which might be high. Furthermore, security is a concern, given that IoT devices may be unable to run secure protocols to communicate with the cloud. We overcome these limitations by employing a fog-computing infrastructure between IoT devices and the cloud. The role of the fog is to preprocess data before it reaches the Internet.[4] Hence, more sophisticated security protocols can be employed before the messages are sent over the Internet. Additionally, messages can be aggregated, filtered, and compressed in the fog, reducing both the traffic sent to the cloud and network costs.

This article introduces SensingBus, a system that uses the concept of mobile sensing in smart cities. To offer sensor mobility, SensingBus leverages the mobility of bus lines of public-transportation systems. Thus, using the IoT paradigm, buses in SensingBus are equipped with sensors and a communication interface to collect data and send that data to fog nodes located at bus stops. The fog sends the data to the cloud, after performing preliminary processing.

SensingBus is based on a three-level architecture, similar to the work of Flavio Bonomi and his colleagues[4] and Wei Li and his colleagues.[5] The first level is composed of IoT devices that collect data. The second level is the fog infrastructure, which lies between the IoT devices and the cloud, and is employed to improve security and reduce performance issues. The third level is the cloud, which receives this data and makes it available to users.

We also present the SensingBus prototype and a brief performance test of the fog nodes. Our results show that fog nodes can serve at least 20 simultaneous devices and that this number of devices is the maximum required to cover 88% of the bus stops in Rio de Janeiro.

MOBILE SENSING IN SMART CITIES

There are different projects that employ mobile sensing to monitor smart cities. BusNet is an architecture based on a delay-tolerant network used to monitor road conditions.[6] In this architecture, buses are equipped with sensors, and a main bus stop is responsible for gathering and analyzing the collected data. Data arrives at the main stop by using opportunistic communication. BusNet employs sensor units, installed on buses, which send data to auxiliary nodes spread over a road network. The auxiliary nodes then send the collected data to buses traveling in the direction of the main station. Hence, buses act as sensing nodes as well as data mules.

Mosaic is another mobile sensing architecture, which targets measuring air quality.[7] In Mosaic, sensor nodes are installed in vehicles and send raw data to the cloud using a GSM/GPRS (Global System for Mobile Communications / general packet radio service) module. The cloud is responsible for processing data and delivering it to users through an API. One of the cloud tasks is to calibrate the received information by removing inaccurate data. This calibration is performed using artificial neural networks and support vector machines.

The SensingBus communication architecture lies between the BusNet and Mosaic approaches. We employ opportunistic communication between buses and fog nodes that are located at bus stops. This approach is similar to the communication between sensor units and auxiliary nodes of BusNet. Nevertheless, fog nodes receive data and send it directly to a cloud, instead of using data mules. In Mosaic, the information is sent from sensing nodes directly to the cloud. Hence, differently from Mosaic, we employ a fog infrastructure between the sensing nodes and the cloud. This fog can perform preprocessing, such as the data calibration implemented in Mosaic.

The project SmartSantander[8] and the work of Rosa Ma Alsina-Pagès and her colleagues[9] also employed vehicles to perform urban mobile sensing. SmartSantander employed sensor nodes attached to urban vehicles in order to gather data about the city, together with other data sources. The project followed a three-tier architecture, with IoT, gateway, and server tiers. The work of Alsina-Pagès and her colleagues analyzed a wireless sensor network with urban buses for noise monitoring. Alsina-Pagès and her colleagues listed the requirements of this network and evaluated different options to perform signal processing using sensor nodes. The main difference between SensingBus and both of these projects is that the gateway nodes in SensingBus can preprocess data, composing a fog level and enabling new services and architectural possibilities.

Some research has employed mobile sensing to collect data and perform scientific analysis. Some examples addressed issues related to air pollution mapping. In these projects, data was collected to build models and infer the air quality in a city. Since their focus was data processing, they did not propose an architecture to store and make information available.

One such example is OpenSense, which used buses in Lausanne to acquire data and proposed models to estimate polluted locations.[10] Joshua Apte and his colleagues presented another approach focused on data processing, in which sensors were installed in Google Street View cars in Oakland, California.[11] Their work offered different conclusions about air quality in that city, as well as generic models or technical information that could be applied in other cities. Finally, the work by Joseph von Fischer and his colleagues employed Google Street View cars to detect gas leaks.[12]

The above-mentioned research confirmed the potential of mobile sensing for smart cities and indicated that an integrated strategy for city sensing can prove useful. SensingBus can improve upon these other approaches by applying the fog paradigm to mobile urban sensing with low-cost equipment. This strategy can enable new applications, reduce costs, and improve the quality of service.

SENSINGBUS

The main goal of SensingBus is to provide a monitoring platform to facilitate the development of smart-city applications. SensingBus implements the whole monitoring process, which comprises data sensing, transmission, storage, and service. SensingBus has these main functionalities:

- *Data gathering*. SensingBus gathers data along the path of participating buses—i.e., the ones equipped with a sensing node.
- *Data preprocessing*. SensingBus preprocesses data before transmitting it through the Internet. For instance, inconsistency is filtered so that the links to the Internet are not overused. Additionally, using signed certificates, SensingBus prevents unauthorized devices from publishing or modifying the data before it reaches users.
- *Data transmission*. Gathered data is transmitted from buses to users. Users must be able to ensure that the data is not fake; i.e., it was gathered by SensingBus and was not modified during transmission.
- *Data storage*. All data is stored in a single database in the cloud.
- *Data service*. Urban data collected by SensingBus is made available to users.

To provide such functionalities, the architecture of SensingBus is divided into three levels: sensing, fog, and cloud. The sensing level is composed of sensing nodes located in the vehicles. This level performs data gathering. Data is then sent to the fog level, consisting of fixed nodes installed at strategic spots around the city, such as bus stops. The fog level preprocesses the data as a fog element in the network. The cloud level runs on a cloud service and receives, stores, and processes the data. The cloud level also gives users access to data. The architecture of SensingBus and its nodes is shown in Figure 1.

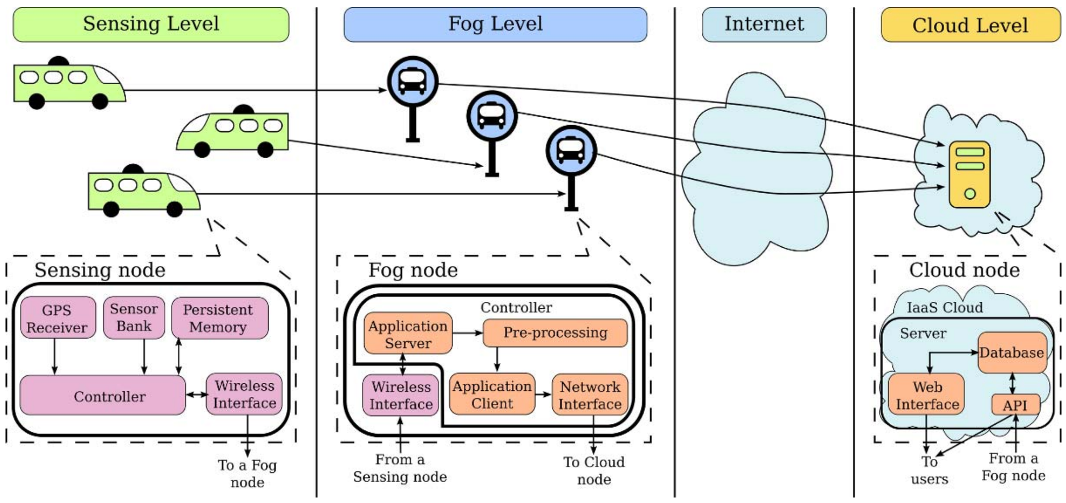


Figure 1. The three-level architecture of SensingBus.

The Sensing Level

The sensing level is responsible for gathering data about the city and sending it to fog nodes. Along with the sensed data, the sensing node registers the time stamp and geographic coordinates of each sample. This level is composed of the set of all sensing nodes located in the buses.

The controller of the sensing node (see Figure 1) manages all tasks of the node. At a specific sampling rate, the controller reads the coordinates and time indicated by the GPS receiver and reads the measurements of every sensor in the sensor bank. The coordinates, time, and measurements are associated and written into the persistent memory. At every iteration, the controller also queries the wireless interface to check whether a connection is established with a fog node.

The sensing node connects to the fog node using a private network where all the other devices are trusted. Whenever a connection is established, the controller sends all the data stored in the persistent memory to the wireless interface. This data is sent to a fog node and can thus be deleted from the persistent memory in the sensing node. The size of the persistent memory must be enough to hold all data gathered between any two consecutive encounters of the sensing node with a fog node.

One design objective of SensingBus is to have inexpensive sensing nodes because, ideally, every urban bus in the city will carry one. Thus, sensing nodes are simple and constrained in terms of computational resources.

The Fog Level

The fog level follows the fog-computing paradigm. Fog nodes are located at the edge of the network. Their computational power is at an intermediate level between the sensing nodes and nodes in the cloud level, which are cloud servers. Basically, fog nodes receive raw data from the sensing level, preprocess it, and send it over the Internet to the cloud level. The preprocessing is fundamental to our system since it can provide important functionalities that improve performance and security. These functionalities include data aggregation, data compression, cryptography, and others.

Fog nodes (see Figure 1) act as both application clients and servers, depending on the context. From the viewpoint of sensing nodes, fog nodes are application servers receiving data. Additionally, sensing nodes also view fog nodes as access points, since fog nodes create private networks for communication between fog and sensing nodes. On the other hand, to the nodes in the cloud level, fog nodes are application clients. They connect to the cloud level using the Internet and forward data received from the sensing nodes.

Communication between the sensing nodes and fog nodes happens on a private network; therefore, a certain security level is assured. Since messages between fog nodes and cloud nodes travel over the Internet, it is important that fog nodes use protocols that implement authorization and data integrity. For this reason, it is expected that every fog node has a certificate, signed by a certificate authority known and trusted by the cloud nodes.

The Cloud Level

The cloud level is the final destination of the collected data. Inside it, data is stored, processed, and made available to users. There are many reasons for having these tasks performed by a cloud service, but the most important are the elasticity and availability of the cloud. Elasticity is important because we expect intensive algorithms to use this data as input from time to time. Availability is important because SensingBus might act as the core of fundamental services, such as flood warnings.

The webserver of a cloud node waits for requests from fog nodes and users. Fog nodes can only add data to the database, whereas users can only read data from the database. A cloud node consists of a virtual machine running on a distributed IaaS (infrastructure as a service) cloud. This arrangement provides elasticity, allowing the processing and storage of the cloud node to grow and shrink, on demand. Moreover, availability is favored by the distributed aspect of the IaaS cloud. Different sites of the distributed cloud can replicate the same cloud node, improving the resilience of the system.

Cloud nodes must hold a certificate signed by a certificate authority trusted by the nodes of the fog level, in such a way that fog nodes know if they have their messages intercepted and even discarded. Additionally, cloud nodes must be capable of checking the certificates presented by fog nodes. This prevents malicious devices from inserting data into the cloud node database.

The main limitation of SensingBus is related to its spatial and temporal coverage. The spatial coverage is limited because SensingBus can only cover the streets that are visited by buses. The temporal coverage is limited because buses travel through the city, covering different areas at each moment. Therefore, SensingBus can serve as a main source of data, as other networks cover the areas SensingBus cannot reach.

THE PROTOTYPE

We built a prototype for SensingBus in order to confirm its feasibility and scalability. The equipment and software used in the sensing, fog, and cloud nodes are listed in Table 1. Our source code is available at http://github.com/pedrocruz/sensing_bus. The next sections explain in detail the different parts of the prototype.

Table 1. The equipment and software used in the SensingBus prototype.

Type of node	Module	Equipment	Manufacturer or publisher	Hardware or software
Sensing node	Controller	Arduino UNO R3	Arduino	Hardware
	GPS receiver	GS-96U7	Guangzhou Xintu	Hardware
	Persistent memory	GS-96U7	Guangzhou Xintu	Hardware
	Wireless interface	ESP8266	Espressif	Hardware
	Sensor bank	DHT11 humidity and temperature sensor	DFRobot	Hardware
		GL5528 light intensity sensor	GBK Robotics	Hardware
		YL-38 rain intensity sensor	GBK Robotics	Hardware
Fog node	Controller	Raspberry Pi II Model B	Raspberry Pi Foundation	Hardware
	Wireless interface	WiPi	element14	Hardware
Cloud node	IaaS (infrastructure as a service) cloud	OpenStack	OpenStack Foundation	Software
	Server	Apache 2.4.18	Apache Software Foundation	Software
	Database	MySQL 5.7	Oracle	Software
	Web interface	Django	Django Software Foundation	Software
	Web API	Django REST (Representation State Transfer) framework	Tom Christie	Software

Sensing Nodes

The sensing nodes employ an Arduino Uno as the controller, owing to its low cost. The wireless interface is an ESP8266, a programmable IEEE 802.11b/g/n interface. Physically, this node is composed of two parts: one is kept inside the bus, and the other is installed outside the vehicle. This division is performed since some modules must be exposed to the environment, such as sensors and transmitters, while other components must be protected from harsh environment conditions. For example, rain can harm some electronic equipment, but a rain sensor must be exposed to it. The part inside the vehicle contains the controller, GPS receiver, and persistent memory. The GPS receiver and persistent memory are on the same board, the most expensive in this node. The external part contains the wireless interface, GPS receiver antenna, and sensor bank.

The external part is protected by an acrylic case measuring 7 × 7 × 4 cm, on the rooftop of the bus. Since sensors need to detect environmental issues, the protective case has a transparent surface and some holes, allowing sensors to effectively sense the environment. The effect of the protective case and mobility on the measurements were tested in previous work, using a similar component layout.[13]

The following functionalities are currently implemented in the sensing nodes:

- *Data gathering*. Data is gathered by sensors in the sensor bank.
- *Data temporary storage*. Data is stored until a connection with a fog node is possible.
- *Data delivery*. Data is delivered to a fog node when a connection is available.

Taking advantage of the programmable wireless interface, we implement a simple protocol for communication between the controller and the wireless interface. In this protocol, the controller asks periodically whether there is a connection, and the wireless interface answers. When the answer is positive, the controller sends enough data to fill the buffer of the wireless interface, waits for data to be sent, and initializes a new iteration, asking again whether there is a connection.

Regarding implementation of this system, messages have constant values per node, such as a header, indicating the sensors installed in the bank. Since the SRAM (static RAM) of the Arduino Uno is limited, we treat all the constants as strings and store them in the flash memory. In the first implementation on the ESP8266 wireless interface, every message sent opened a TCP connection and closed it after the message was sent. Even though this approach saved lines of code, every new connection created new objects that were not discarded immediately when the connection closed, causing instability to the ESP8266. To solve this issue, the connection status is kept, and the ESP8266 performs several requests over the same connection.

The wireless interface holds the SSID (service set identifier) and password of a WPA2 (Wi-Fi Protected Access 2) network created by the fog nodes. The wireless interface searches periodically for a wireless network with the preconfigured SSID, and answers queries from the controller. After the wireless interface signals the controller about a new connection, the controller sends only application-specific data to the wireless interface. The wireless interface prepares the HTTP headers and executes a POST method to the fog node. According to our experiments, this approach is two times faster than preparing the HTTP headers inside the controller. The codes used in this test can be found in the "sensing/tests" folder in the repository.

Fog Nodes

The equipment used in fog nodes is shown in Table 1. A Raspberry Pi II Model B was chosen as the controller because of its computing power, size, cost, and available interfaces. WiPi was chosen as the wireless interface owing to its cost and because of the number of simultaneous connections that it supports. Some other options with greater communication range were considered but discarded because of the low number of simultaneous connections supported. The effect of simultaneous connections is further discussed in the following sections.

The wireless interface of the fog nodes acts as an IEEE 802.11 access point protected with WPA2. The SSID and password are configured into the controller. In the controller, an HTTP server implemented in Python serves incoming requests and preprocesses data before sending it to the cloud node. Fog nodes have the following preprocessing functions currently implemented:

- *Error detection*. Data is checked for inconsistencies, and defective data is discarded.
- *Data concentration*. Data is stored and sent periodically. We use a single connection at the end of each interval, reducing the number of connections to the cloud level.
- *Data compression*. Data is compressed before it is sent, reducing the traffic to the Internet and, consequently, the cost of subscriptions.

For every request, the fog node controller checks data and discards inconsistent measurements. After that, data is accumulated in a concentration queue. At regular intervals, data is compressed and sent to the cloud node. To send data, the fog node acts as an HTTPS (HTTP Secure) client, getting authorization by presenting its certificate to the cloud node.

Cloud Nodes

The prototype of a cloud node is implemented in software. The software used is shown in Table 1. An Apache (httpd.apache.org) server runs on a virtual machine, instantiated on an IaaS cloud. The Apache server executes a Django (www.djangoproject.com) application, which relies on MySQL (www.mysql.com) to store data, and on the Django REST (Representational State Transfer) framework (www.django-rest-framework.org) to create API endpoints.

The cloud node exposes a URL for data insertion. This URL is protected by Apache, which authorizes clients only if they present valid certificates, generated by a trusted certificate authority. Therefore, on SensingBus as a whole, data integrity and authorization between the sensing level and fog level rely upon WPA2, whereas between the fog and cloud levels, data integrity and authorization are ensured by HTTPS.

The cloud node prototype also provides API endpoints for querying data. Data can be queried by date, time, location, and sensor type. Thus, users can use SensingBus to develop their own applications. The documentation of the API can be found together with the available code.

The following functions are currently implemented in cloud nodes:

- *Data insertion endpoints*. API endpoints are exposed for data insertion.
- *Data query endpoints*. API endpoints are exposed for data query. Data can be queried by date, time, location, and sensor type.

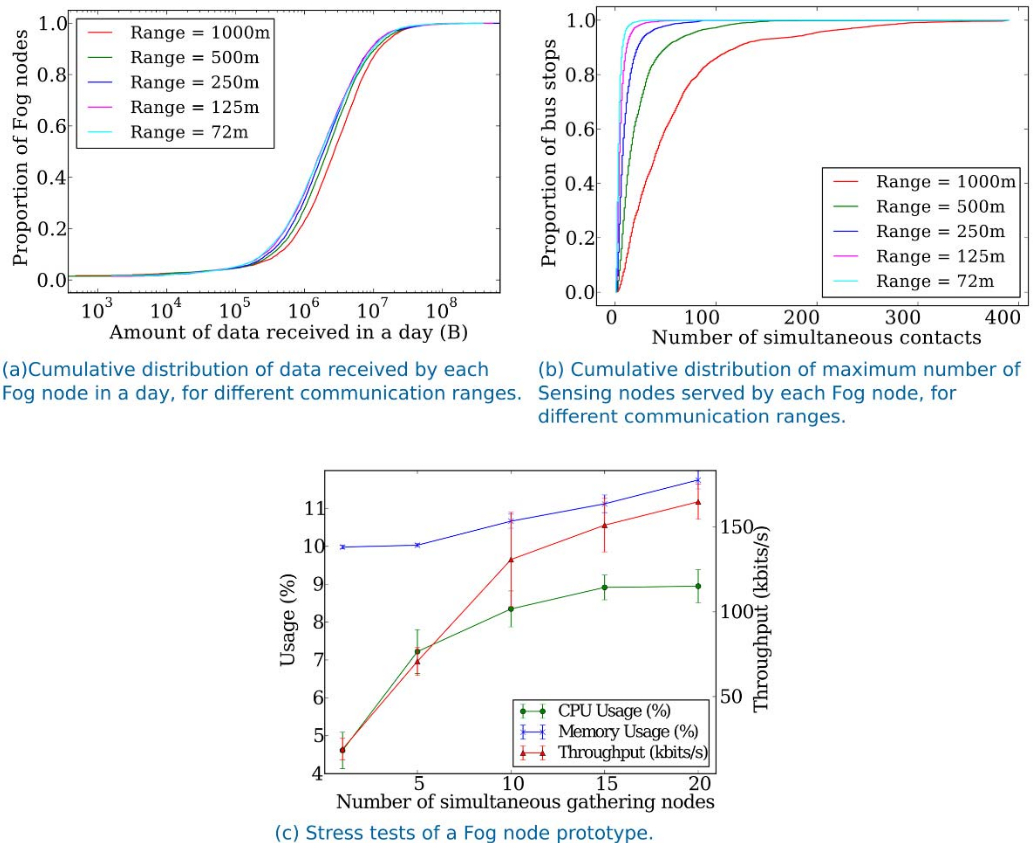
The communication between different nodes is performed using HTTP or HTTPS, and users fetch data using RESTful APIs. This ensures uniform interfaces, to deal with the heterogeneity of devices and applications.

Performance Analysis

We conducted a number of experiments to test the scalability of our fog node prototype. We aimed to evaluate the number of requests a single fog node has to serve simultaneously. To estimate this number, an analysis of two real datasets was performed. The first dataset contained the positions of all bus stops in Rio de Janeiro (https://www.gta.ufrj.br/~cruz/datasets/bus_lines_and_stops.csv). The second dataset contained the positions of all buses in Rio de Janeiro, refreshed every minute (http://dadosabertos.rio.rj.gov.br/apiTransporte/apresentacao/rest/index.cfm/obterTodasPosicoes). We collected the positions of all buses from 8 AM to 10 PM on 30 November 2016. The early and late hours of the day were discarded because a great number of buses were parked, creating distortions in our evaluation.

We assumed that every bus carried a sensing node and every bus stop had a fog node installed. For every minute, the positions of buses and bus stops were compared. If the distance between a bus and a bus stop was shorter than a given threshold that represented the communication range, we considered that the sensing node on the bus could send data to the fog node at the bus stop. We defined five communication ranges: 1,000, 500, 250, 125, and 72 meters.

Using this dataset, it was possible to evaluate the amount of data received by every fog node. Figure 2a illustrates the cumulative distribution of the amount of data received by each fog node throughout a day. For all ranges, less than 40% of the fog nodes received less than 1 Mbyte of data. In addition, for every range, 10% of the fog nodes concentrated more than 47% of the traffic.



(a)Cumulative distribution of data received by each Fog node in a day, for different communication ranges.

(b) Cumulative distribution of maximum number of Sensing nodes served by each Fog node, for different communication ranges.

(c) Stress tests of a Fog node prototype.

Figure 2. Analysis of the fog nodes of the prototype.

In order to test the scalability of the fog node prototype, we evaluated the maximum number of sensing nodes served simultaneously by each fog node during the day. Figure 2b shows the cumulative distribution of the maximum number of sensing nodes served simultaneously by each fog node, for different communication ranges. According to Marcelo Rubinstein and his colleagues, the range of IEEE 802.11g in traffic speed is less than 250 m.[14] Since our prototype could use IEEE 802.11b/g/n, we adopted 250 m as a reference. Figure 2b shows that, for a communication range of 250 m, around 88% of the fog nodes served fewer than 20 sensing nodes simultaneously. Therefore, we adopted 20 as the maximum number of sensing nodes to use in the stress test.

We used the same datasets to verify the intercontact times of buses with bus stops, considering a communication range of 250 m. We observed that, in more than 98% of the samples, a bus waited less than 10 minutes between contacts. Therefore, we configured every sensing node in our stress test with data equivalent to 10 minutes of sensing. Given that the sensor bank in our prototype generated 53 bytes every second, 10 minutes of sensing produced 31,800 bytes. In our implementation, this was equivalent to 15 HTTP POSTs from the sensing node to the fog node. The data generation rate of our prototype indicated that the persistent memory had to store less than 5 Mbytes in order to store data for a whole day. In addition, the maximum expected time a bus traveled with no fog contact was much shorter than a day.

To simulate the simultaneous arrival of sensing nodes in the communication range of a fog node, we simultaneously turned on 1, 5, 10, 15, and 20 sensing nodes within the range of the fog node.

After the sensing nodes were on, we waited for all data to be received by the fog node and transmitted to the cloud node of the prototype. For each number of sensing nodes, the test was repeated 30 times. Throughout the tests, we did not observe any instability in our nodes. The codes used in this experiment are available in the "fog/stress_tests" folder in the repository.

Figure 2c shows the results for the average memory and CPU usage from when the first POST arrived at the fog node until the last POST arrived at the fog node, for all the participating sensing nodes. The measurements are presented with a 95% confidence interval. In Figure 2c, you can observe that the fog node could serve up to 20 sensing nodes without exhausting its memory and CPU resources. It is worth noting that the average throughput obtained by each sensing node dropped as the number of sensing nodes increased, as a consequence of competition for the medium.

In previous research, we studied the intercontact times between buses and bus stops.[15] Considering the data generation rate of the sensor bank, we concluded that the throughput per sensing node must be at least 3 Kbps to deliver all data gathered in a single contact opportunity. Our experiments showed that our prototype suited such requirements, even in the worst case, with 20 simultaneous sensing nodes sharing the same fog node.

CONCLUSION

This article presented SensingBus, a system for smart-city sensing based on urban buses. SensingBus combines the concepts of the IoT and fog and cloud computing in order to collect, preprocess, process, store, and serve sensed data. One of the main goals of SensingBus is to provide a low-cost general-purpose solution that collects and provides to users information from different locations in a city.

We also demonstrated the feasibility of SensingBus through the results obtained using a prototype, built using low-cost hardware. Those experiments analyzed the scalability of the SensingBus prototype. Using data from real bus lines, we showed that the fog node prototype supported at least 88% of the bus stops in Rio de Janeiro. Note that in a real implementation, we might not install fog nodes at all bus stops. Hence, being able to choose 88% of the bus stops is more than enough.

As future work, we plan to build a pilot version of the system for the internal bus fleet of the Universidade Federal do Rio de Janeiro campus. We are also investigating promising user applications that can benefit from data gathered from SensingBus, as well as a list of sensors that should be added to the sensor bank. Another plan is to use fog nodes to coordinate cooperation between buses to eliminate, or at least reduce, the amount of duplicate data.

ACKNOWLEDGMENTS

This work was partly funded by FAPERJ, CAPES, CNPq, and grants 15/24494-8 and 15/24490-2, São Paulo Research Foundation (FAPESP).

REFERENCES

1. J. Gubbi et al., "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, 2013, pp. 1645–1660.
2. B. Liu et al., "Mobility improves coverage of sensor networks," *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing* (Mobihoc 05), 2005, pp. 300–308.

3. K. Xu, Y. Qu, and K. Yang, "A tutorial on the Internet of Things: From a heterogeneous network integration perspective," *IEEE Network*, vol. 30, no. 2, 2016, pp. 102–108.
4. F. Bonomi et al., "Fog computing and its role in the Internet of Things," *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing* (MCC 12), 2012, pp. 13–16.
5. W. Li et al., "System modelling and performance evaluation of a three-tier Cloud of Things," *Future Generation Computer Systems*, vol. 70, May 2017, pp. 104–125.
6. K.D. Zoyza et al., "A Public Transport System Based Sensor Network for Road Surface Condition Monitoring," *Proceedings of the 2007 workshop on Networked systems for developing regions* (NSDR 07), 2007, p. Article No. 9.
7. W. Dong et al., "Mosaic: Towards City Scale Sensing with Mobile Sensor Networks," *IEEE International Conference on Parallel and Distributed Systems* (ICPADS 15), 2015, pp. 29–36.
8. L. Sanchez et al., "SmartSantander: IoT experimentation over a smart city testbed," *Computer Networks*, vol. 61, March 2014, pp. 217–238.
9. R.M. Alsina-Pagès et al., "Design of a mobile low-cost sensor network using urban buses for real-time ubiquitous noise monitoring," *Sensors*, vol. 17, no. 1, 2016, p. 57.
10. A. Marjovi, A. Arfire, and A. Martinoli, "High Resolution Air Pollution Maps in Urban Environments Using Mobile Sensor Networks," *International Conference on Distributed Computing in Sensor Systems* (DCOSS 15), 2015, pp. 11–20.
11. J.S. Apte et al., "High-Resolution Air Pollution Mapping with Google Street View Cars: Exploiting Big Data," *Environmental Science & Technology*, vol. 51, no. 12, June 2017, pp. 6999–7008.
12. J.C. von Fischer et al., "Rapid, Vehicle-Based Identification of Location and Magnitude of Urban Natural Gas Pipeline Leaks," *Environmental Science & Technology*, vol. 51, no. 7, March 2017, pp. 4091–4099.
13. P. Cruz et al., "On the Accuracy of Data Sensing In the Presence of Mobility," *7th International Conference on Network of the Future* (NOF 16), 2016.
14. M.G. Rubinstein et al., "Measuring the capacity of in-car to in-car vehicular networks," *IEEE Communications Magazine*, vol. 47, no. 11, November 2009, pp. 128–136.
15. P. Cruz, R.S. Couto, and L.H.M.K. Costa, "An algorithm for sink positioning in bus-assisted smart city sensing," *Future Generation Computer Systems*, 2017; doi.org/10.1016/j.future.2017.09.018.

ABOUT THE AUTHORS

Pedro Cruz is a DSc student in electrical engineering at Universidade Federal do Rio de Janeiro. His research interests are cloud computing, fog computing, and the Internet of Things. Cruz received an engineer degree in computer and information engineering from Universidade Federal do Rio de Janeiro. Contact him at cruz@gta.ufrj.br.

Felipe F. da Silva is pursuing a BSc in electronic and computer engineering at Universidade Federal do Rio de Janeiro. His research interests include cloud computing, the Internet of Things, and network virtualization. Contact him at felipe@gta.ufrj.br.

Roberto G. Pacheco is pursuing a BSc in electrical engineering at Universidade do Estado do Rio de Janeiro. His research interests include fog computing and the Internet of Things. Contact him at roberto.pacheco@uerj.br.

Rodrigo S. Couto is an associate professor in the Department of Electronics and Telecommunications Engineering at Universidade do Estado do Rio de Janeiro. His research interests include cloud computing, the Internet of Things, and network virtualization. Couto received a DSc in electrical engineering from Universidade Federal do Rio de Janeiro. Contact him at rodrigo.couto@uerj.br.

Pedro B. Velloso is an associate professor in Universidade Federal do Rio de Janeiro's Electronic and Computer Engineering Department. His research interests include content

distribution, wireless networks, the Internet of Things, and cloud computing. Velloso received a doctorate in computer science/telecommunications from Université Pierre et Marie Curie—Paris VI. Contact him at velloso@gta.ufrj.br.

Miguel Elias M. Campista is an associate professor in Universidade Federal do Rio de Janeiro's Electronic and Computer Engineering Department and a full professor in the university's electrical-engineering program. His main research interests are computer networking, cloud computing, and network science. Campista received a DSc in electrical engineering from the Federal University of Rio de Janeiro. Contact him at miguel@gta.ufrj.br.

Luís Henrique M.K. Costa is an associate professor in Universidade Federal do Rio de Janeiro's Electronic and Computer Engineering Department. His research interests include routing on wireless networks, group communication, quality of service, multicast, and stateless routing. Costa received a DSc in computer science from the University Pierre et Marie Curie. He's an associate member of IEEE and a professional member of ACM. Contact him at luish@gta.ufrj.br.