Low-cost wireless testbed for Internet of Things *ad hoc* networks prototyping and evaluation

Paulo Alexandre Regis*, Amar Nath Patra†, and Shamik Sengupta†
* Department of Computer Science
Southeastern Louisiana University–Hammond, Louisiana USA 70402
Email: pregis@southeastern.edu
† Department of Computer Science and Engineering
University of Nevada–Reno, NV USA 89557
Email: apatra@nevada.unr.edu, ssengupta@unr.edu

Abstract—**There is a distinct gap in IoT research between simulation and implementation of new networking protocols. While simulations allow fast validation of new ideas, it is not until implemented into a real system that a novel protocol gains tracking. We build an *ad hoc* network system to enable energy-aware protocol testing in real devices with reusable code. We demonstrate how to use low-cost components with open-source libraries to implement the system. We implemented models to allow the energy-awareness using a network simulator, and experiments to compare different routing protocols through indoor and outdoor experiments, showing the feasibility of the system.**

Index Terms—**Internet of Things, testbed, wireless *ad hoc* networks**

I. Introduction

The Internet of Things (IoT) is a new paradigm in which connected devices, embedded systems, and other forms of physical devices can interact with each other through the Internet. A "thing" in the Internet of Things is a specialized device with specifics objectives. It can range from connected home appliances, wearable devices such as smart-glasses, healthcare monitors, drones, and even smart-phones are part of the IoT universe. Similarly, there are several technologies that enable the communication aspect of the IoT. From near field communication to Bluetooth, they all provide communication access to a "thing" [1]. The growth of IoT is envisioned to trigger a massive demand for wireless communication due to technological advancements in this area. Smart-phones are becoming more and more accessible to the mass public, regulations for the use of Unmanned Aerial Vehicles (UAVs, also known as drones) by the general public are starting to emerge. The range of applications predicted for these new technologies can vary widely: smart-homes, temporary backbone network, search and rescue missions with the use of autonomous robots, traffic management, and so on and so forth.

The work-flow when designing new network protocols comprises of mathematical modeling and numerical results, protocol implementation and simulation validation, request for comments from the community (according to regulating agencies like IETF and IEEE), and finally, software integration in the operating system level (i.e., kernel module). However, this entire process can take a long time spanning many years. In this paper, we aim to close the gap between the theoretical and practical parts of the development process by providing tools to the community that make it easier to go from the simulation to experimental phase.

Autonomous and intelligent robots are becoming the norm due to the public interest, which encourages its commercialization. With the increased interest, new research challenges come to light such as security and data privacy, routing protocols for highly mobile vehicles, efficient spectrum utilization, and many others [2], [3], [4]. The spike in popularity brings to light the need to improve the quality of networking communications from different points of view. For example, novel routing methods have been proposed with different goals. One of which is to perform load-balancing to minimize congestion in the network [5]. By routing packets through different paths, a network might be able to reduce the total delay that affects the data streams. Another approach uses location-aware routing protocols, which utilizes the physical position of the nodes to assist with routing strategies [6], [7]. The basic idea is to use the proximity as a weight in the path computation: nodes that are closer to each other are less prone to external interference, can use less transmitting power to establish a link, and have less path-loss due to their proximity. Another objective might be the longevity of the network [8], [9], where protocols try to maximize the time the network can stay connected before nodes start running out of battery power. For example, in our previous works, we split a traffic flow into multiple distinct paths in order to increase the longevity of the network [10]. If all data flows were to route through the same path, the nodes in that path would run out of energy sooner, which in some cases may interrupt the entire traffic flowing in the network. Many other network optimization problems can be found in the literature, and one aspect that most of them have in common is the assumption that more information about the state of the system (energy, positioning, transmission power) is available. However, in practice, this is difficult to achieve due to current hardware and software limitations.

In this paper, we build a testbed system comprising of off-the-shelf components. We build our solution upon the existing *ns-3* network simulator framework to integrate hardware sensors, which in turn enables access to the state of charge of the energy source. We compare the performance of our testbed among three scenarios, using the proposed testbed setup in different environments. First, we design an indoor experiment setup that provides both the best and worst case benchmark results. We then compare the results with an external experiment executed in a wide open area field.

This paper provides a complete testbed system with novel features that enables energy-aware wireless *ad hoc* network protocols to be evaluated in a real experiment with physical devices. The main contributions of this paper are as follows.

- Coulomb counter (fuel gauge) model in *ns-3*, using a customized sensor library. The model allows exploration of various scenarios regarding the state of charge. The module updates the state according to real consumption values obtained from an I2C sensor device.
- Proof-of-concept testbed and guidelines to reproduce both the system configuration as well as the experiments. The reusable code enables testing new energy-aware protocols in real devices without extensive re-implementation.
- Comparison between the simulation and testbed scenarios, giving insights on how to improve the simulation models to better reflect the behavior of real devices.
- Experiments and analysis of the performance of the system in both indoor and outdoor scenarios, providing benchmark results as well as a starting point for future performance experiments. We list the benefits as well as the shortcomings of using the proposed approach.

The remainder of the paper is organized as follows. Section II presents a literature review in the field. Section III describes the development design process, from the hardware selection to the software models implemented for the system. Section IV compares and evaluates experiments performed with testbed and compares with similar simulation scenarios. Finally, Section V gives the final remarks and concludes this article.

II. Related work

There have been many different testbed systems proposed to aid the development and testing of wireless networks protocols. They provide the infrastructure with different capabilities that enable the deployment of custom protocols. Alternatively, they can be built with a specific set of standards in mind. We list features and shortcomings of some testbed systems.

The ORBIT project is perhaps the most notable testbed platform [11]. The main system is an indoor static grid comprised of several nodes. It supports a variety of radio interfaces, such as GNU radio, WiFi networks, and it also has an outdoor mobile environment that allows mobility in the network. The experiments rely on operating system images to be deployed, meaning that for each experiment, the researcher must implement the protocols to be tested and specify the OS image to be used. Several base images that serve as starting points are provided. However, the system requires knowledge of the development system, as well as the scripting language used to run the experiments. It is not straightforward to jump from a simulation to an experimental testbed environment.

The PhantomNet testbed provides a large number of devices to researchers. It uses the Emulab software platform to provision the testbed experiments [12]. Its array of hardware include real eNodeB devices, common-off-the-shelf end devices (Android cellphones), software defined radios, and it also has an RF attenuator matrix that allows the insertion of controlled loss. Even though the PhantomNet utilizes well-known software components and a variety of physical devices, its primary focus is on the cellular networks. Therefore wireless mesh networks experiments are not feasible in its current setup.

One of the objectives of our work is to enable a fast transition for researchers from simulations to a testbed implementation. There are several network simulators available to the community. For example, the OMNeT++ is a modular event driven simulator adapted for networking applications [13]. OPNET simulator has a large set of models available [14]. However, most of the system implementations focus on smart-grid backbone communications and would require the acquisition of expensive equipment. An open-source alternative is the network simulator *ns-3* network simulator is an open-source alternative [15]. It is in active development and has a large user community. The simulator also has some features that allow testbed implementations. We explore these features in this work as well as implement new ones.

TABLE I: Network simulators comparison

	ns-3	ns-2	OPNET	OMNeT++
Language	C++	C++/oTcl	C/C++	C++
Operating System	Win/Linux/ macOs	Win/Linux/ macOs	Win/Linux	Win/Linux/ macOs
License	Open-source (GPLv2)	Open-source (GPLv2)	Commercial	Commercial/ Academic
GUI	Limited	Limited	Extensive	Extensive
Hardware interface	Yes	No	Yes[1]	Yes[1]

In Table I we see a brief comparison between the simulators mentioned above. While *ns-3* does not offer a great variety of hardware integration (i.e., it has support for NIC integration), we developed a new library for the framework as a new module. OPNET, on the other hand, offers a great variety of hardware integration; however, the commercial aspect discourages its use in this particular work. There are many tools available to the scientific community. Even though they offer a vast amount of opportunities, they also lack interoperability. In this work, we provide new libraries implemented in *ns-3* that enable energy-aware protocols to be easily ported using the same code base from the simulation.

III. System design and proposed mechanism

Two high-level components enable the testbed system to function correctly. The hardware component is comprised of the logic board that runs the experiment logic, the current sensor that enables power-aware experiments, and the power supply. The software components are the programmable/configurable parts of the testbed and are mainly comprised of

[1]Offers hardware-in-the-loop features

the operating system running in the logic board, and the user-space library models that replicate the network stack and make it manageable to manipulate from the user level.

A. Hardware components

Each node in the testbed is made of three components: logic board, current sensor, and power supply. The requirements necessary to enable desired features for the system are as follows.

- ***Ad hoc* mode**: the wireless NIC driver must be able to work in *ad hoc* mode because the software component will need to process packets even if it is not addressed to the node itself.
- **State of charge (SoC)**: nodes must have access to power supply state information. SoC itself is a vast research problem, and it is out-of-scope to evaluate different mechanisms related to it. Energy-aware protocols can be implemented if SoC estimation is present.
- **Modularity**: the hardware components should be replaceable to allow future system extensions. For example, multi-radio multi-channel radio interfaces.

Given the listed requirements, we opted to use the Raspberry Pi 3 Mobel B (Pi) as the logic board of out testbed nodes [16]. It has built-in WiFi interface that enables wireless communications in the $2.4GHz$ frequency range that can be configured in *ad hoc* mode and allows to change the transmitting power of the interface. We used the INA219 current sensor to provide power supply readings to our logic board [17].

B. Software components

The goal of the testbed is to provide a fast-lane between prototyping a new protocol in simulation and testing in real devices. We chose to use the network simulator *ns-3* due to its open-source nature and some features already available [15]. It is an event-driven simulator with libraries developed in C++, and the maintainers release new versions with updates and fixes in a six-month cycle basis. It also provides python bindings for researchers that prefer that programming language. However, not all modules have this capability implemented.

We take advantage of mainly two *ns-3* features: the $FdNetDevice$ module, and the $RealtimeSimulatorImpl$. The $FdNetDevice$ is a high-level device implementation that binds a socket (file descriptor) to the hardware (a given network interface), which enables the software to write and read bytes directly to the NIC buffer. The interface then transmits those bytes through the medium. The $RealtimeSimulatorImpl$ is a real-time scheduler that is attached to the operating system internal clock. Meaning that instead of running event after event as soon as possible, the simulator will wait for the amount of time between consecutive events before executing them. Implying that a simulation configured to run for T seconds will actually run for T seconds. In this paper, we extend the functionality of the $FdNetDevice$ to enable transmitting power management. We also developed a new module that interfaces with the INA219 current sensor. These new features allow researchers to test energy-aware protocols and validate them in a real testbed.

For the $FdNetDevice$ to receive packets properly, it needs the NIC to be previously configured in *promiscuous mode* (the driver must have this mode implemented). This allows the operating system to receive packets even if they were not addressed to the node. In practice, the $FdNetDevice$ will be configured with a different address than the one specified in the experiment script. The $FdNetDevice$ transmits packets by injecting frames into the NIC buffer. The injected frames will have the MAC and IP addresses according to the script, and will not use the OS configured ones. The program then uses the *ns-3* implementation of the network stack to perform other tasks (routing, sending packets up in the stack, etc.). $FdNetDevice$ provides many features that enable the development of physical testbeds. However, it does not have the interface to change the NIC configuration at runtime, which would enable energy-aware protocols to run in the testbed.

We implemented four new class methods to the original $FdNetDevice$: `GetDeviceName()`, `SetDeviceName(std::string devName)`, `GetTxPower()`, and `SetTxPower(int power)`. We also adapted the helper class $EmuFdNetDeviceHelper$ to store the device name ($wlan0$, $eth0$, etc.) in the class object, which initially does not have any reference to the device name. To avoid any software or library dependency, we opted to use `ioctl` calls to the wireless card.

Both `GetDeviceName()` and `SetDeviceName(std::string deviceName)` methods simply keep track of an internal object attribute. The `GetTxPower()` method can be called anytime during the experiment. It will perform a read `ioctl` operation on the $deviceName$, and return the transmitting power configured. Since it is a read operation it does not require administrative privileges by the OS. However, the `SetTxPower(int power)` does require it since it will write the new value to the NIC configuration. In order to avoid giving $root$ access to the entire simulation program, we created an auxiliary program that needs to have the *sudo* bit turned on. With this, only the auxiliary program needs to execute as root. It is the same approach used by the simulator to create the raw sockets to interact with the NIC. The $FdNetDevice$ will fork a process that will, in turn, execute the helper program. This program (`tx-power-setter`) accepts two arguments: an integer representing the new transmitting power, and a string which is the device name. Then the program will try to change the value according to the arguments received.

The update to the $FdNetDevice$ enables changing the transmitting power during the simulation. However, this is only the first step to enable energy-aware protocols. We still need to enable *ns-3* experiments to have access to reliable energy usage status. As mentioned previously, we selected the INA219 current sensor to measure the current flowing through the node hardware. We implemented `ioctl` calls to read, write, and calibrate the sensor according to its technical specifications [17]. These calls are independent of the *ns-3* environment, and can be used in any C++ project; the source-code, as well as usage instructions, can be found in [18].

The energy module in *ns-3* provides several model implementations for different kinds of batteries (base class $EnergySource$) and consumption rates (base class $DeviceEnergyModel$). They all rely on having the initial state of charge of the energy source. To allow multiple scenarios to be tested, we follow the same concept, where the initial energy is provided to the model. However, instead of depleting the battery according to a model, our module depletes the current state by the amount of energy read from the sensor, giving a more realistic view of the system. We created the $Ina219Source$ module that extends the basic $EnergySource$ class. We implemented a basic coulomb counting algorithm [19]. The $Ina219Source$ object instance will read the current value every δt interval from the sensor and deplete the energy by $\delta t * current$. The value of interval δt is $1s$ by default but can be modified by the script. Note that determining the state of charge is a research topic by itself. Even though we chose to implement the simple coulomb counter method, other methods could be implemented as well.

1) Other configurations: In our testbed we used *Linux kernel 4.14.50-v7+*. Each wireless NIC in the node is configured to be in *ad hoc* mode, and all nodes should be configured to connect to the same network ESSID, prior to the experiment. To avoid any unwanted traffic to go through the wireless interface being used for the experiment, we also configured the default behavior of ARP and DHCP protocols. By default, the system is configured to broadcast ARP requests in all interfaces of a computer (Pi), which would result in packets outside the scope of *ns-3* being collected and processed by the experiment script. We also disable any power management feature that the NIC driver may provide. Instead, power management can be seen as a new possibility of the testbed. Since the INA219 sensors use $I2C$ to communicate with the computer, this option must be enabled before the execution of any script.

IV. Experimental evaluation

In this section, we explain how we measure the performance of the proposed system. First, we compare the energy consumption behavior of an *ns-3* simulation with single node with a single node of the testbed. This comparison provides insights about the consumption model implemented in *ns-3* and how it could be further improved. Then, we designed a simple indoor experiment that forces a multi-hop straight line network topology. We use the indoor experiment as a benchmark scenario. The 1-hop experiment gives the best performance for each protocol tested, while the 4-hop indoor experiment returns the worst results. We compare the benchmark results with an outdoor experiment. The outdoor scenario provides a more realistic behavior since the topology cannot be controlled. All experiments took place between $July1^{st}$ and $August31^{st}$ 2018, during non-commercial hours ($9pm-5am$) to minimize external interference. Each experiment setup was run five times, and the results were averaged.

A. Experiment parameters

In our experiments, we compare the performance of five different routing protocols: Ad-hoc On-demand Distance Vector (AODV), Destination-Sequenced Distance Vector (DSDV), Optimized Link State Routing (OLSR), OLSR with expected transmission count metric (ETX), and Distributed Split-path (SPLIT). We use our previously proposed SPLIT protocol as it uses the remaining battery power as part of the implemented link metric. Among the listed protocols it is the only energy-aware one. The experiments run for $600s$. All nodes use channel 11, with a transmitting power of $10dBm$. The client node generated traffic using a Constant Bit Rate model, with packet size of $1024B$.

B. Benchmark

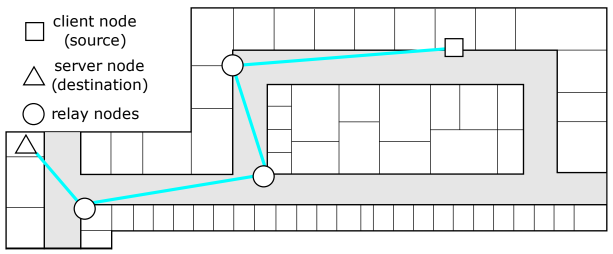

Fig. 1: The indoor experiment floor plan. The topology of the network is a linear due to the materials of the building.

Figure 1 illustrates the floor-plan of the Computer Science and Engineering department in which the indoor experiment was performed. The position of each node is explicit in the figure. Nodes are placed in such a way that the topology is forced to be a single straight line. The triangle represents the server node, or the sink node, which is the destination of all the data traffic in the network. This scenario can be compared to a command and control center, where sensor nodes flush data to a central node that in turn consumes the received data by performing some computation or analysis with it. In our case, we compute the amount of application data received as a performance indicator.

We run two different scenarios in the indoor environment. First, the best case scenario, in which there are only two nodes (i.e., client and server). The server is in the same triangle position, but the client is placed in the first relay node position. The positioning of the nodes in the worst case scenario is shown in Figure 1. In both cases, the client tries to send as much data to the server as possible.

C. Energy consumption

The goal of this experiment is to illustrate how the implemented consumption model in *ns-3* differs from an actual physical hardware. Both simulation and experiment contain a single node that starts a CBR application that broadcasts packets using UDP. The experiment runs for $100s$, while we measure the total energy consumption of the node.

The simulation updates the consumption value using the TraceSource system, provided by the simulator, each time the value changes (it might not be in uniform time intervals since the model updates the value every time the device state changes). For the testbed node, on the other hand, we update the total energy consumption at fixed time intervals of $100ms$ becausesince it does not respond to the *ns-3* model events. In addition, other hardware components might also affect the device energy consumption (I/O device interfaces, the operating system itself, and so forth) that *ns-3* is not aware of its existence. In the testbed we use the Coulomb counter (fuel gauge) method to count the amount of energy drained from the source.

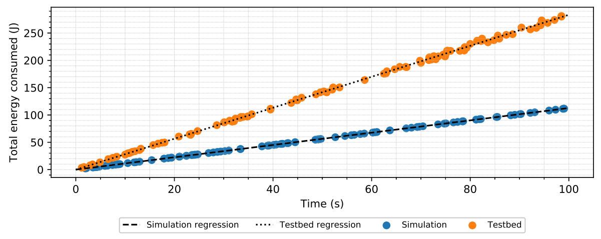

Fig. 2: Total consumption comparison between simulation and testbed nodes. Both present similar linear behavior with different slopes, indicating some component(s) of the hardware is(are) not accounted for in the consumption model of the simulator. Each dataset was sub-sampled for better visualization.

Figure 2 shows the consumption trend for each experiment, simulation and testbed. We see a considerable difference in absolute values between them. However, the trend hints that the simulated model trend is quite similar, and could be potentially adjusted to reflect the consumption of a real world device (not just the radio installed in the device).

The difference in energy consumption is different because, for the simulator, it is hard to capture the behavior of other components that comprise a node. It is possible to replicate such behavior if prior to the simulation, there is knowledge of what activities the node will be performing, such as applications that require more energy (video capturing, machine learning tasks). However, in practice, this prior knowledge might not be available when designing the system. Thus, having the ability to evaluate the network system in a testbed such as this, can be beneficial when developing new systems [20].

D. Outdoor experiment

In order to evaluate the testbed in a more realistic application scenario, we selected a wide open area in our university campus. We chose the area referred as the Quad, a $45m$ by $146m$ grass field surrounded by buildings and trees. Figure 3 shows the exact positioning of the nodes, as well as the point-of-view from the perspective of each node. The overview shows the field surrounded by the buildings.

Due to the open area nature of the outdoor area, the topology of the network is unknown, which is the intended behavior. Thus throughout the experiment, the routing protocols may opt for multiple different routes. Since this is a popular area

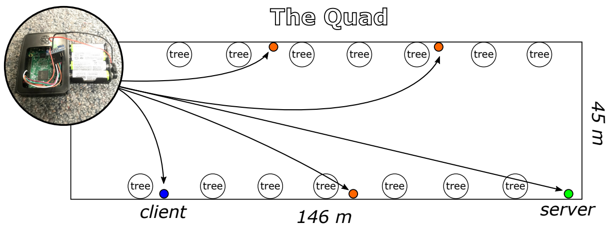

Fig. 3: Outdoor experiment node placement. Nodes were positioned between trees, and the entire setup was surrounded by buildings.

where students walk by, the experiment was conducted during the night time to minimize interference from smart-phones and other devices that may use the same frequency channel as the testbed.

E. Results and discussion

As mentioned before, we compare the obtained results between the 1-hop (best case scenario), the 4-hop (worst case scenario) indoor experiments with the outdoor experiment. The performance indicators are defined as follows:

- **Average link throughput**: the average amount of bytes transferred among all the links between nodes in the network.
- **Application throughput (goodput)**: the number of bytes that the server node consumed per second throughout the experiment run.
- **Average consumption**: the amount of current measured by the sensor at each second (by definition it is in *coulombs (C)*), which is used to implement the coulomb counter mechanism. The result is the average between the readings among all nodes in the network.

In Figure 4 we can see the link throughput comparison. Keeping in mind that the underlying protocol mechanisms do work, we can see that among the routing protocols, OLSR offered the best performance in the outdoor experiment, while the others had relatively similar performances. ETX demonstrated the lowest decline when compared with the indoor benchmark counterpart.

Comparing the *goodput* in Figure 4, on the other hand, showed that OLSR had the lowest decline. Due to the route changes promoted by ETX and SPLIT, and the route discovery/maintenance mechanism from both distance-vector protocols, their performance gap is considerably higher.

Perhaps the most reliable and closest to an actual production system metric is the battery consumption, shown in Figure 4. For each protocol, the best case benchmark scenario is the one that used more energy. This behavior is due to the lower delay (due to a direct link connection between source and destination), and the lower number of total nodes. Consequently, the client was able to transmit much more packets to the destination. The consumption during the outdoor experiment, on the other hand, is similar to the worst case scenario. This pattern is a good sign, especially for the SPLIT protocol, since the amount of data transferred is larger while consuming a similar amount of energy.

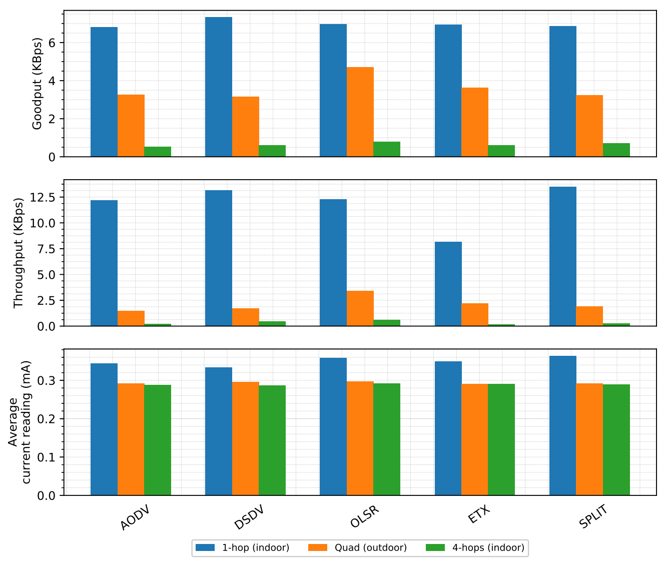

Fig. 4: Testbed network performance.

The initial results obtained show the viability of the testbed. Even though the network performance might not translate to a network in production performance, it does, however, allow the fast prototyping of new energy-aware network protocols. The most significant factor that undermines the network performance is the packet input/output that the software has to perform. Currently, due to its development design, the *ns-3* module can only execute a single packet I/O operation at a time. However, in the future, if the system allows to skip the kernel operation by allowing direct access to the hardware queue, this issue can be minimized.

V. Conclusion

In this article, we designed an energy-aware wireless network testbed using off-the-shelf components. We developed new modules to fill the necessary gap between energy-dependant simulations and experiments in the testbed using ns-3. The new modules and features are freely available for the community to reuse it [21]. Experimental results prove the viability of the testbed. While simulations give more control over various parameters, the testbed experiments enable real-world performance measurements. Although the viability of the testbed is proven through experimental results, the results may not reflect the scenario in which the protocol is deployed in production. One possible solution would be the integration of the *netmap* device, which allows direct access to queue buffers of the hardware in the userspace, which in turns enable read and write operations of packets in batches directly to the buffer [22].

References

[1] V. Coskun, B. Ozdenizci, and K. Ok, “A survey on near field communication (nfc) technology,” *Wireless personal communications*, vol. 71, no. 3, pp. 2259–2294, 2013.

[2] N. Komninos, E. Philippou, and A. Pitsillides, “Survey in smart grid and smart home security: Issues, challenges and countermeasures,” *IEEE Communications Surveys Tutorials*, vol. 16, pp. 1933–1954, Fourthquarter 2014.

[3] Y. Chen and H. Oh, “A survey of measurement-based spectrum occupancy modeling for cognitive radios,” *IEEE Communications Surveys Tutorials*, vol. 18, pp. 848–859, Firstquarter 2016.

[4] S. Batabyal and P. Bhaumik, “Mobility models, traces and impact of mobility on opportunistic routing algorithms: A survey,” *IEEE Communications Surveys Tutorials*, vol. 17, pp. 1679–1707, thirdquarter 2015.

[5] H. Hassanein and A. Zhou, “Routing with load balancing in wireless ad hoc networks,” in *Proceedings of the 4th ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pp. 89–96, ACM, 2001.

[6] K. Katsaros, M. Dianati, R. Tafazolli, and R. Kernchen, “Clwpr — a novel cross-layer optimized position based routing protocol for vanets,” in *2011 IEEE Vehicular Networking Conference (VNC)*, pp. 139–146, Nov 2011.

[7] C. Prodhon and C. Prins, “A survey of recent research on location-routing problems,” *European Journal of Operational Research*, vol. 238, no. 1, pp. 1–17, 2014.

[8] C. E. Jones, K. M. Sivalingam, P. Agrawal, and J. C. Chen, “A survey of energy efficient network protocols for wireless networks,” *wireless networks*, vol. 7, no. 4, pp. 343–358, 2001.

[9] Y. Yao, Q. Cao, and A. V. Vasilakos, “Edal: An energy-efficient, delay-aware, and lifetime-balancing data collection protocol for heterogeneous wireless sensor networks,” *IEEE/ACM Transactions on Networking (TON)*, vol. 23, no. 3, pp. 810–823, 2015.

[10] P. A. Regis and S. Sengupta, “Distributed split-path routing strategy for multi-hop mesh networks,” in *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, pp. 575–580, Oct 2017.

[11] M. Ott, I. Seskar, R. Siraccusa, and M. Singh, “Orbit testbed software architecture: Supporting experiments as a service,” in *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. Tridentcom 2005. First International Conference on*, pp. 136–145, IEEE, 2005.

[12] A. Banerjee, J. Cho, E. Eide, J. Duerig, B. Nguyen, R. Ricci, J. Van der Merwe, K. Webb, and G. Wong, “Phantomnet: Research infrastructure for mobile networking, cloud computing and software-defined networking,” *GetMobile: Mobile Computing and Communications*, vol. 19, no. 2, pp. 28–33, 2015.

[13] A. Varga and R. Hornig, “An overview of the omnet++ simulation environment,” in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, p. 60, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[14] X. Chang, “Network simulations with opnet,” in *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future-Volume 1*, pp. 307–314, ACM, 1999.

[15] ns-3 Consortium, “Network simulator, ns-3.” https://www.nsnam.org/.

[16] Raspberry Pi Foundation, “Raspberry pi.” https://www.raspberrypi.org/.

[17] T. I. Inc., “Ina219 zero-drift, bidirectional current/power monitor with i2c interface,” 2015.

[18] Regis, Paulo Alexandre, “Raspberry pi c++ library for voltage and current sensors using the ina219.” https://github.com/regisin/INA219.

[19] S. Piller, M. Perrin, and A. Jossen, “Methods for state-of-charge determination and their applications,” *Journal of Power Sources*, vol. 96, no. 1, pp. 113 – 120, 2001. Proceedings of the 22nd International Power Sources Symposium.

[20] G. P. Perrucci, F. H. Fitzek, and J. Widmer, “Survey on energy consumption entities on the smartphone platform,” in *2011 IEEE 73rd vehicular technology conference (VTC Spring)*, pp. 1–6, IEEE, 2011.

[21] Regis, Paulo Alexandre, “Testbed ns-3 github public repository.” https://github.com/regisin/ns3-hw.

[22] L. Rizzo, “Netmap: a novel framework for fast packet i/o,” in *21st USENIX Security Symposium (USENIX Security 12)*, pp. 101–112, 2012.