A scalable communication abstraction framework for internet of things applications using Raspberry Pi

Conference Paper · May 2018

DOI: 10.1117/12.2306596

CITATIONS
0

READS
85

5 authors, including:

Anke Meyer-Base
Florida State University
349 PUBLICATIONS **2,289** CITATIONS
SEE PROFILE

Simon Y. Foo
Florida State University
67 PUBLICATIONS **826** CITATIONS
SEE PROFILE

Behshad Mohebali
Florida State University
5 PUBLICATIONS **12** CITATIONS
SEE PROFILE

Amirhessam Tahmassebi
Florida State University
26 PUBLICATIONS **52** CITATIONS
SEE PROFILE

A Scalable Communication Abstraction Framework for Internet of Things Applications using Raspberry Pi

Behshad Mohebali[a,*], Amirhessam Tahmassebi[b], Amir H. Gandomi[c], Anke Meyer-Baese[b], and Simon Y. Foo[d]

[a]Department of Electrical Engineering, Center for Advanced Power Systems, Florida State University, Tallahassee, Florida, USA
[b]Department of Scientific Computing, Florida State University, Tallahassee, Florida, USA
[c]School of Business, Stevens Institute of Technology, Hoboken, New Jersey, USA
[d]Department of Electrical and Computer Engineering, FAMU-FSU College of Engineering, Tallahassee, Florida 32310-6046, USA

ABSTRACT

The Internet of Things concept is described as a network of interconnected physical objects capable of gather, process, and communicate information about their environment, and potentially affect the physical world around them through their sensors, embedded processors, communication modules, and actuators, respectively. Such a network can provide vital information on events, processes, activities, and future projections about the state of a distributed system. In addition, it can give the devices inside the network awareness about their environment far beyond the range of their dedicated sensors through communication with other devices. In most cases, such network consists of devices with different processing and communication capacities and protocols, from a variety of hardware vendors. This paper introduces an abstracted messaging and commanding framework for smart objects, aimed towards making the network capable of including various communication standards. This issue is addressed by proposing a messaging structure based on JavaScript object notation (JSON) format so the new devices connecting to the network can introduce themselves to the central coordinator. The introduction includes a list of functionalities that the device is capable of, and the information it needs to carry out those tasks. This platform makes the network capable of incorporating different devices with various purposes and functions with ease and flexibility. Having a fast, reliable, and scalable communication scheme is critical for realization of a robust and flexible network.

Keywords: Raspberry Pi, Arduino, Internet of Things, RESTful Web Services

1. INTRODUCTION

1.1 Raspberry Pi System on a Chip

Raspberry Pi, as a fully functional and ridiculously inexpensive little computer has lots of potentials and applications in real life to be used as desktop PC, wireless server, media center, gaming machine, game server, robot controller, stop motion camera, radio station or in wider fields such as cloud computing, multi-core distributed scheduling, parallel computing, and Internet of Things.

In 2013, Tso et al.[1] used 56 Model B Raspberry Pi to construct a 4-rack Cloud Data Center cluster. The multi-root tree topology is used to form the cluster, in which all the Raspberry Pi devices in the same rack are connected to a designated Top of Rack (ToR) switch. Each Pi supports three concurrent hosts having isolated file systems. The cluster is introduced as a cost-effective platform for research on various aspects of Cloud Computing. One advantage of using PiCloud is that the novel approaches can be operated on actual (yet scaled down) hardware platform instead of simulation tools, which usually isolate a certain aspect of the Cloud and may make assumptions that are far from reality. Virtual Machine management, resource allocation, novel

* Corresponding Author: Behshad Mohebali
E-mail: bmohebali@fsu.edu

network architectures, and power consumption measurement and optimization (which is difficult to predict using simulators)[2] are among the Cloud Computing research topics that can benefit greatly from PiCloud as a testbed. In addition, the researchers are considering the possibility of Cloud Computing without virtualization, in which the actual physical node is rented out to the client instead of a virtual node. This is feasible only with smaller and cheaper processors such as Raspberry Pi. At last the cost of a 56 machine testbed using commodity x86 servers and PiCloud are compared to show the cost-effectiveness of PiCloud as a replication of the architecture of a Cloud Data Center. The performance of the cloud is not measured or presented in this work.

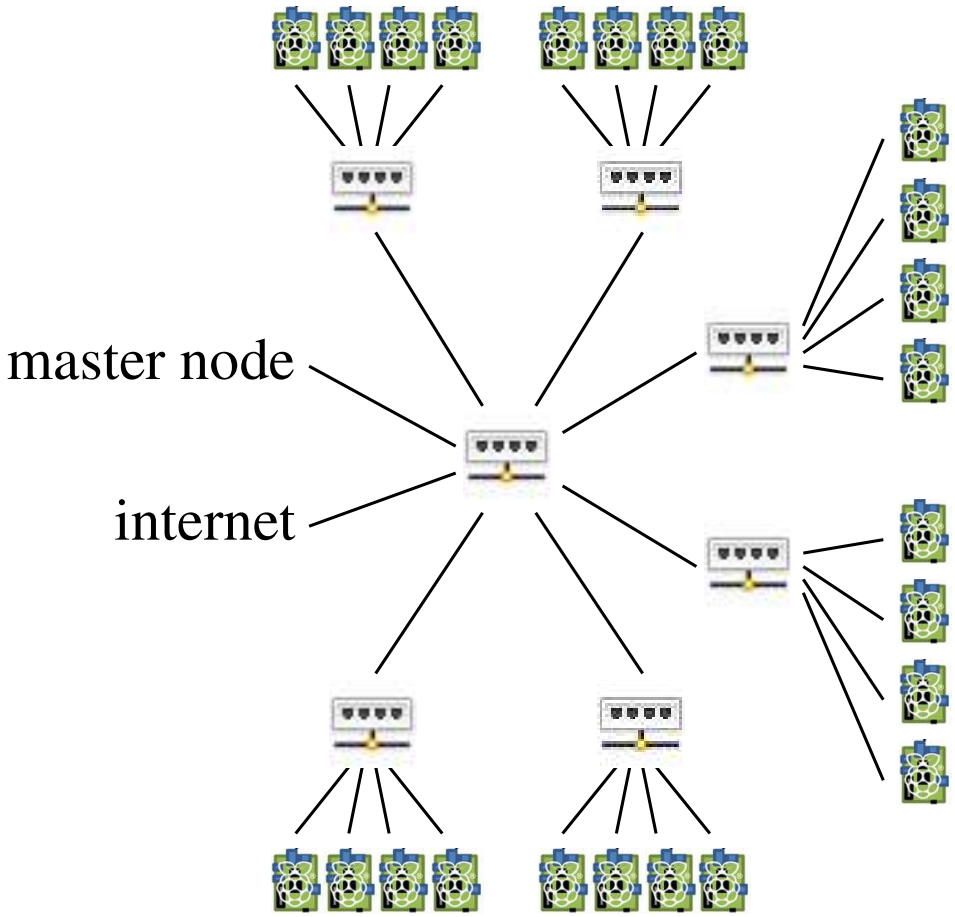


Figure 1: Network topology used in[3] to form a cluster with 300 Raspberry Pie devices.[3]

While the cloud DC proposed in[1] is meant mostly for cloud computing research,[3] investigates the benefits and challenges of having a practical cloud made of Pies capable of handling computationally intensive operations. The limited processor speed and SD card speed and life span are cited as such challenges. However, the affordability and energy efficiency are listed as the benefits. In addition to their low speed, the SD card will distribute the storage capacity over the whole network. To address this issue, a common file system is provided for the cluster using a Network Attached Storage. The cluster is made of 300 RPies in a star network as depicted in Figure 1. Different parts of the software running on the nodes are optimized to reduce the load on each RPi. The OS used in this project is a minimal version of Debian 7, created by the authors, which has only the necessary features. Also, it is argued that some of the open-source cloud computing platforms such as OpenStack can put strain on the scarce resources of RPi. As a result, it makes sense to develop customized platforms that can centralize the management of the resource pool, configure the network dynamically, monitor the whole network, and provide online access to the resources. The authors envisioned using this cluster as an inexpensive testbed for research[1] and also a mobile data center cluster that can be deployed in remote locations.

In 2015, Schot[4] investigated the possibility of implementing a Hadoop server on a RPi cluster. Apache Hadoop is a widely used open-source framework that uses map/reduce methods, developed for the first time at Google,[5] to process large distributed data on a cluster.[6] Hadoop framework has four major modules:

- Hadoop common libraries and utility packages used by other modules
- Hadoop Distributed File System (HDFS), that stores the data on commodity hardware.
- Hadoop YARN, the resource management layer of Hadoop
- Implementation of map/reduce methods.

With only eight Pies, size of the cluster used in[4] is considerably smaller than the former examples. However, it serves the purpose as a proof of concept. The Pies run DietPi, a lightweight version of Raspbian optimized for minimal usage of hardware resources.

1.2 Internet of Things

The Internet of Things is described as a smart infrastructure that connects the objects and people.[7] Since no specification about the communication protocol or the type of devices that should be used is included, it is logical to expect that any communication protocol might be utilized to connect different types of electronic devices depending on the circumstances. Although this can liberate the design engineers and network architects, it brings up new challenges that must be addressed. Therefore, heterogeneity (Having the ability to support and manage variety of technologies, devices, services, and environments)[8] and scalability (ability to handle large number of devices and services into the network) are listed as the main requirements of an IoT system[9] which should be dealt with on architecture level.

In addition, an IoT system should require minimal (if any) human intervention for configuration and setup. This means that more effort should be allocated to make the devices self-governing in terms of communication. In other words, the ability to set up a new device by merely connecting the device to the network. This feature can decrease the money and time spent for setting up a network in environments where there is a need for high number of simple sensors, such as a shipboard power system,[10,11] or a mobile robot swarm swarm.[12]

Previously, Borgia,[9] Taleb, and Kunz[13] compiled a list of the IoT communication features, which can be summarized as:

- Reliability: The ability to ensure accurate delivery of the data from the source to the destination.
- Security: Having safe connection between the devices with "anonymity of identity and location"[9] while being able to detect abnormal behavior or attack.
- Scalability: Ability to incorporate large number of devices into the network and handle transmission between them.
- Robustness: Ability to maintain connection under adverse circumstances such as bad weather, disruption of the primary communication path, or mobility of one or more devices in the network.
- Flexibility: Ability to support and incorporate multiple types of networks and different communication modes. Ability to expand the functionality in case of needing special purpose packages.

Choosing a network type for establishing connection between two devices is a trade-off between these features. The reliability and security of a network can be mostly addressed in lower layers such as data link layer and network layer, respectively. This can be seen in CAN bus[14] , Ethernet, Zigbee, WiFi, etc. The higher layers, such as application layer, can affect the scalability and flexibility of the network. This paper focuses on a communication establishment approach that can set up newly connected devices and make them ready for operation in a dynamic fashion.

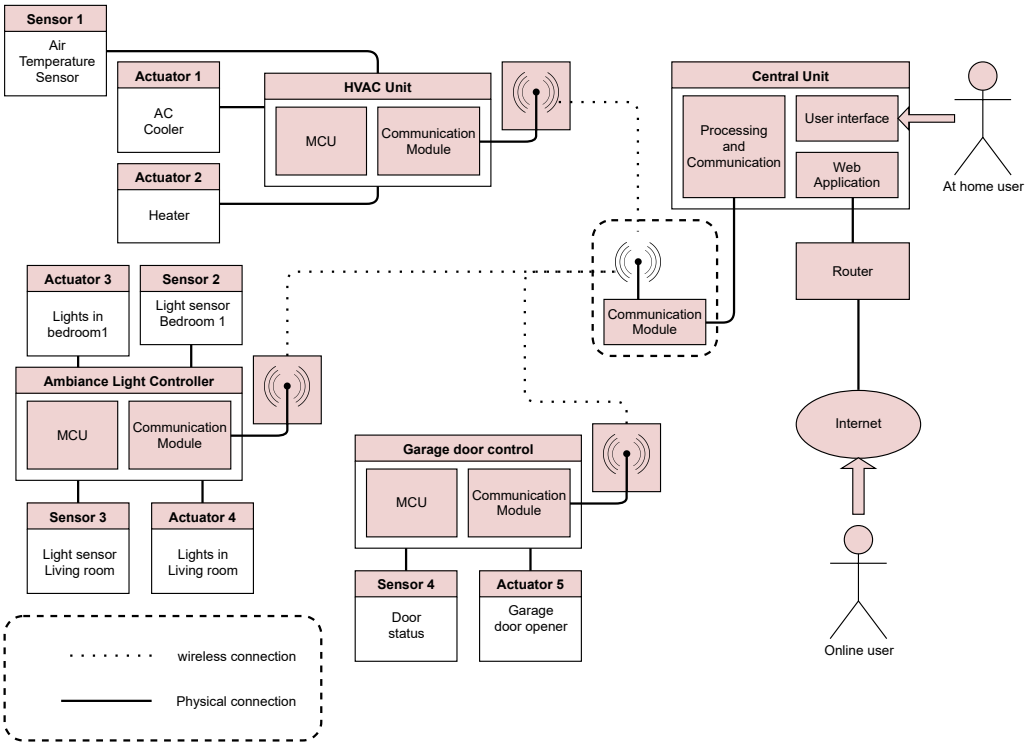


Figure 2: An example of an IoT home automation network. The network consists of different types of devices communicating on different channels.

The rest of the paper is organized as follows. Section two discusses the objectives that the proposed approach is aspire to achieve and how it does it. Section three gives a brief description of the hardware used for testing the communication approach and the developed software for two communicating systems in the network. Section four illustrates the way that the the network coordinator recognizes a new device and adapts its user interface according to the device capabilities. Lastly, some suggestions are offered for the future work.

2. METHODOLOGY

The objectives that are being pursued are:

- To form a mechanism for communication between smart objects that is independent of the technology or hardware vendor as long as both sides of the connection support it.
- To be able to dynamically and seamlessly integrate objects into the network at run time.
- To be able to handle new types of sensors or actuators in the future without applying changes to the underlying framework.

To achieve these goals an approach inspired by Service-Oriented Architecture (SOA)[15] web services is adopted. Although SOA is not a new concept, applying its principles to IoT applications have several advantages. Such an approach lets the designer hide the details of the functionality of a given device behind an interface that is shared with the network coordinator in the beginning of the communication. By using SOA, all the functionalities in the network (such as sensing, moving, recording, etc.) can be treated as loosely coupled services. In addition, adhering to SOA principles means having a mechanism with which the devices can publish their available functionalities and their inputs, outputs, and calling structure in the form of services.

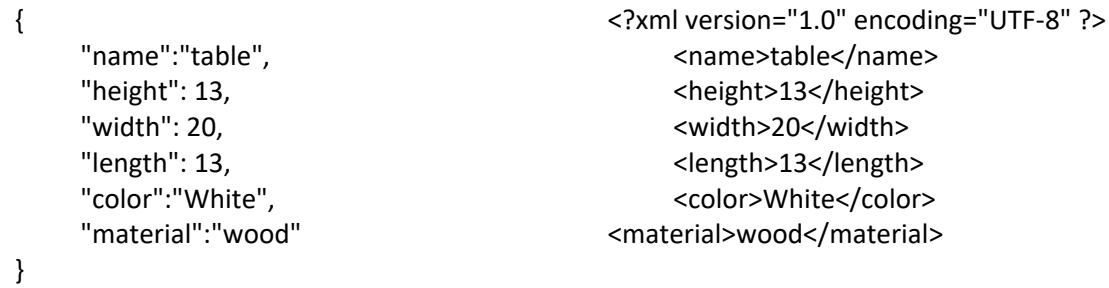


Figure 3: A comparison between JSON and XML. Both of these examples are representing the same information. The JSON has 85 characters while XML has 154 characters, counting out the spaces.

A self-configuring process was developed to facilitate the connection of a new device into an already operating network. This process can be summarized as:

1. New device is connected to the network (all the transmission connections are in place) and is powered on.
2. The device starts by sending generic packages to request permission to publish the service description. This is done periodically with low frequency so it does not occupy much bandwidth from the network.
3. The coordinator receives the request or permission and sends "permission granted" package.
4. Upon receiving the permission, the device sends a "service description" package containing its name, description (to show to the human user), and a full description of services. The service description contains the name of the service, the name and type of the inputs and outputs, and a brief description about the service.
5. The coordinator receives the service description package and updates the list of available services on the UI and web service.

The conventional format used for communication data in SOA web services is Extensible Markup Language (XML). However, due to limited bandwidth and speed in IoT networks, the XML seems unnecessarily verbose. To address that issue, JavaScript Object Notation (JSON) is used in this project. In addition, due to popularity of JSON format, its parser tools are available in most of the programming languages. Figure 3 shows a comparison between a JSON file and an XML file which represent the same object but the JSON uses 69 characters less than XML to carry the same amount of information.

Note that some services, such as reading a sensor or a parameter, may not have any inputs from the coordinator (or user). However, another group of services that are meant to control an actuator may only have inputs and no output. Below is an example of an introduction JSON message for a device that controls the speed of a fan in a bedroom and also has a temperature sensor. This device offers two services. One for setting the speed of the fan, which takes the speed and direction of rotation (true for Clock-wise, false for Counter clock-wise) as its inputs and has no output, and one for reading the temperature sensor that only has an output for the obtained temperature.

{ "name": "Bedroom ventilation control", "type":"introPackage", "desc": "Control the bedroom ventilation.", "services": [{ "name": "Change Speed", "desc": "Change the speed of the bedroom fan (RPM).", "inputs": [{ "name": "speed", "type": "int" }, { "name": "direction CW", "type": "boolean" }], "outputs": [] }, { "name": "Read Temperature", "desc": "Measure the room temperature", "inputs": [ ], "outputs": [{ "name": "Temperature", "type": "double" }] }] }

Figure 4 shows the generic structure of this JSON message. The number of the services and their parameters depend on the functionality and role of the device in the network.

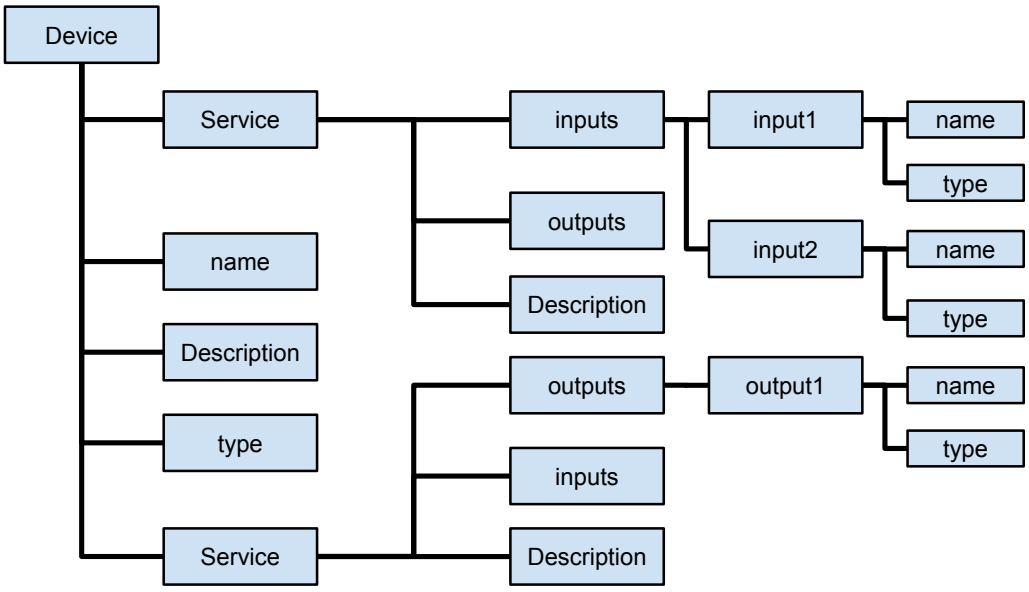


Figure 4: The structure of the JSON message used for publishing the available services by the newly connected device. The name and description are used for prompting the user when getting the inputs for the services. The type property declares that this JSON is an introduction package that contains the information on the device services.

After the coordinator (in this case the Raspberry Pi) establishes the connection and becomes aware of the presence and available services of the newly connected device, it can call for execution of a service using this JSON package:

{ "name": "Change Speed", "type": "request", "parameters": [{ "name": "speed", "value": 60 }, { "name": "direction CW", "value": true }] }

This JSON instructs the bedroom fan controller device to set the speed of the fan to 60 RPM clock-wise. The type "request" is used for commands coming from the coordinator. After getting the request for executing the service, the device sends a package informing the coordinator of succeeding (or failing) in executing the command:

{ "name": "Change Speed", "type": "response", "status": true }

3. SYSTEM DESCRIPTION

To demonstrate the efficacy of the approach a simple testbed that can be used in home automation applications is assembled. Figure 5 shows the schematic of the system, which consists of:

- A Raspberry Pi SoC, which acts as the coordinator in the network.
- An Arduino Uno module, which acts as the local controller that will be connected to the network.
- A step motor driver, which gets the clock signal and the direction of rotation from Arduino and generates signal for four inputs of the step motor.
- Step motor, which is the actuator.

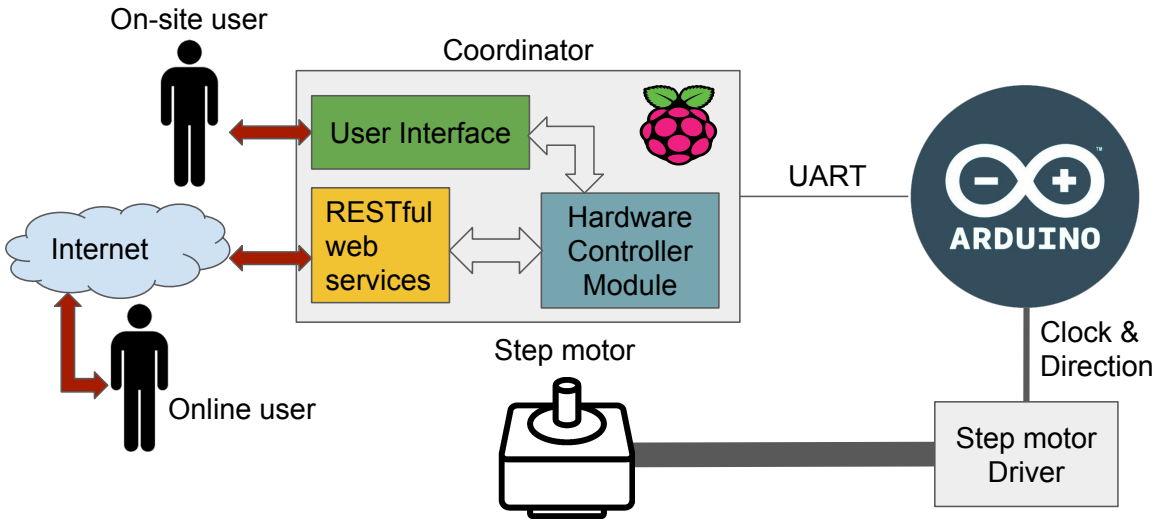


Figure 5: A schematic view of the experiment. The Raspberry Pi SoC runs a RESTful API that makes it able to receive requests from Internet. If the user is at the same location as the also can interact with the system using the graphic interface.

Figure 6 shows the hardware setup. The software on the RPi has three major parts:

- Hardware Controller Module (HCM): This is the core of the software which consists of three modules. First, the logic for organizing the information about the current status of each local device connected to the network, their available services, and their means of communication (whether it is serial port, CAN, LAN, etc.). Second, the command translator, which receives the calls for services from user interface or the web and constructs the request JSON. Third, a general utility module that is responsible for interacting with the hardware. This module is used as an interface between hardware and software by getting the requests for controlling the General Purpose Input/Output (GPIO) pins or sending data over communication ports.

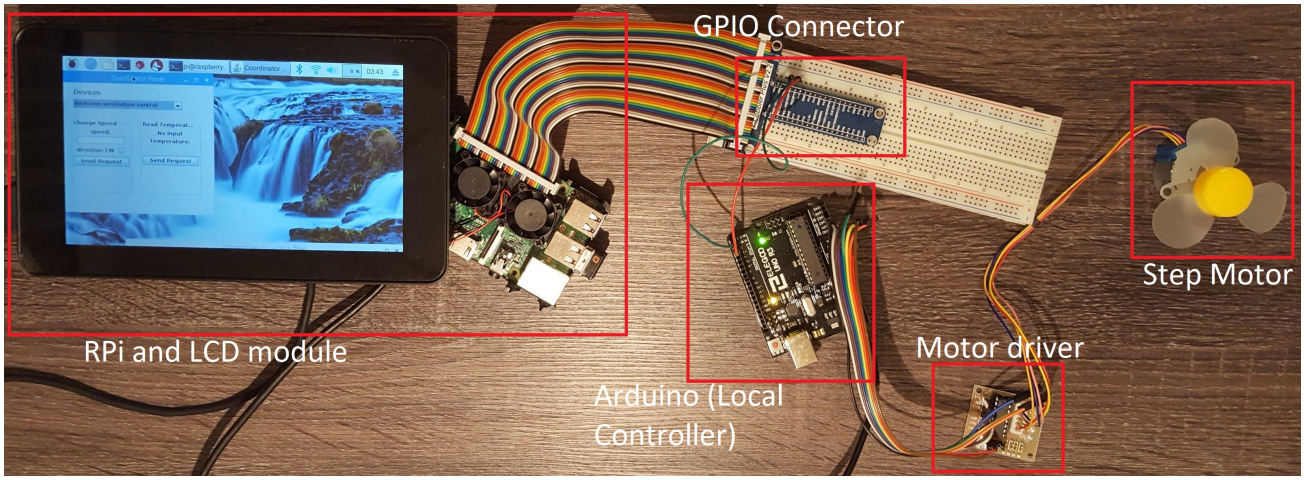


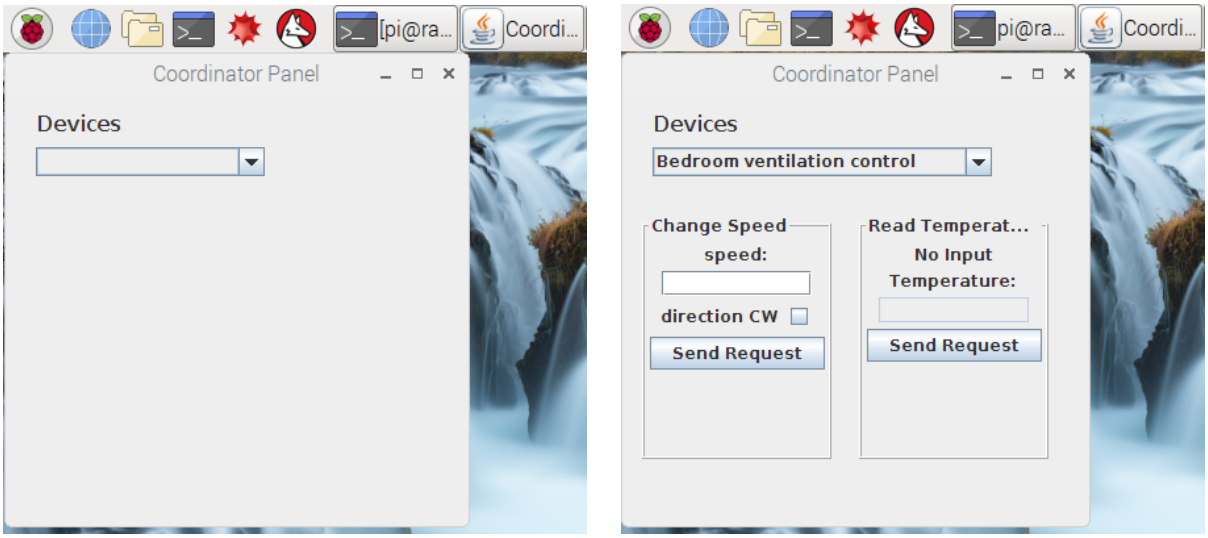
Figure 6: The hardware setup used to demonstrate the application of the communication framework. The Arduino module controls the step motor driver and is connected to RPi GPIO via its UART serial port. Step motor shaft is connected to a fan which is controllable by the user input into the RPi user interface.

- User Interface (UI): The UI has three responsibilities. First, presenting the active devices and their available services to the user along with the needed parameters for each service. Second, collecting the user inputs and communicating them with the HCM. Third, getting the information on active and newly connected devices as well as their available services and updating the interface according to those services and their parameters. The UI is developed using Java Swing widget toolkit.
- RESTful web services: A collection of services designed using REST architecture[16] and exposed to the web that can be invoked from the Internet. This will give the user the ability to call any service in the network from any location provided Internet connection is available. An embedded instance of Apache Tomcat

The software on the Arduino module, written in C, is simpler than the RPi software due to its limited role and responsibilities. It has a block for capturing, recognizing, and constructing a JSON package, and also an interpreter that gets the request from the coordinator, determines which service is called, and calls the service with given parameters.

4. RESULTS AND DISCUSSION

Figure 7 (a) shows the initial version of the coordinator UI before the local controller is connected. Since no device is present in the network the device menu does not have any option available. After connecting the local controller to the network, the coordinator will dynamically update the UI and gives user access to the services of the controller. The final state of the UI after the update is illustrated in Figure 7 (a).



(a) Before (b) After

Figure 7: The coordinator UI (a) before and (b) after connecting the local controller device. The coordinator will dynamically update the UI based on the information that comes from the local controller. Every time a new device is connected to the network the "Devices" drop down menu is updated. By selecting each device from that menu the user can access the services that the local controller offers.

This paper introduces an expandable framework for dynamically establishing connection between a coordinator and a local controller device in an IoT network. Each device was modeled as a collection of services that can be called by the coordinator (which takes inputs from user). The structure of the message that carries the information on available services that a newly connected device will offer was discussed. JSON format was used to build the messages between devices to save development time and communication bandwidth due to its popularity and relatively low character count compared to XML. A testbed was designed and built to test the functionality of the introduced method using hardware tools that are widely popular in IoT applications. Finally, the state of the user interface before and after connecting the local controller (Arduino) to the coordinator

(Raspberry Pi). The UI software module was able to update the available services according to the introduction package received from the local controller.

Even though this approach is a move towards making the IoT self-governing it faces serious challenges along the way. One might argue that the most important concern about this framework is security. The JSON format is mainly meant to be an application to application communications. However, it can be easily interpreted by human if it gets intercepted. In that case the interceptor party has a description of all the available services that the local controller offers and might hijack the line and pretend to be the coordinator invoking a particular service.

Another feature that might be useful is the bidirectional request ability. So far only the coordinator can request a services from local controllers and not the other way around. There should be a similar mechanism for the local controllers to call for information relevant to their role. In such a scenario, the privilege management, in the sense that no device should be allowed to access data beyond its need, will be of great importance.

In addition, not all the parameters have primitive data types. Being able to accept more complex data types can significantly increase the versatility and applicability of the framework.

5. ACKNOWLEDGEMENTS

This work is partially supported by ONR Grant "Gulf of Mexico Spring School (GMSS) Deep Learning Workshop".

REFERENCES

[1] Tso, F. P., White, D. R., Jouet, S., Singer, J., and Pezaros, D. P., "The glasgow raspberry pi cloud: A scale model for cloud computing infrastructures," in [*2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops*], 108–112 (July 2013).

[2] Mohebali, B., Breslend, P., Graber, L., and Steurer, M., "Challenges of frequency domain measurement for modeling the components of shipboard power systems," *Naval Engineers Journal* **126**(4) (2014).

[3] Abrahamsson, P., Helmer, S., Phaphoom, N., Nicolodi, L., Preda, N., Miori, L., Angriman, M., Rikkil, J., Wang, X., Hamily, K., and Bugoloni, S., "Affordable and energy-efficient cloud computing clusters: The bolzano raspberry pi cloud cluster experiment," in [*2013 IEEE 5th International Conference on Cloud Computing Technology and Science*], **2**, 170–175 (Dec 2013).

[4] Schot, N., "Feasibility of raspberry pi 2 based micro data centers in big data applications," in [*Proceedings of the 23th University of Twente Student Conference on IT, Enschede, The Netherlands*], **22** (2015).

[5] Dean, J. and Ghemawat, S., "Mapreduce: simplified data processing on large clusters," *Communications of the ACM* **51**(1), 107–113 (2008).

[6] Tahmassebi, A., "ideeple: Deep learning in a flash," in [*Disruptive Technologies in Information Sciences*], **10652**, International Society for Optics and Photonics (2018).

[7] Brock, D. L., "The electronic product code (epc)," *Auto-ID Center White Paper MIT-AUTOID-WH-002* (2001).

[8] Gama, K., Touseau, L., and Donsez, D., "Combining heterogeneous service technologies for building an internet of things middleware," *Computer Communications* **35**(4), 405–417 (2012).

[9] Borgia, E., "The internet of things vision: Key features, applications and open issues," *Computer Communications* **54**, 1–31 (2014).

[10] Mohebali, B., Breslend, P., Graber, L., and Steurer, M., "Validation of a scattering parameter based model of a power cable for shipboard grounding studies," in [*ASNE Electric Machines Symposium (EMTS)*], 28–29 (2014).

[11] Mohebali, B., "Characterization of the common mode features of a 3-phase full-bridge inverter using frequency domain approaches," (2016).

[12] Kia, M., Rezayieh, K. R., and Mohebali, B., "A novel position sensor in mobile robots motion control," (2012).

[13] Taleb, T. and Kunz, A., "Machine type communications in 3gpp networks: potential, challenges, and solutions," *IEEE Communications Magazine* **50**, 178–184 (March 2012).

[14] Bosch, "Can specification," *Robert Bosch GmbH, Postfach* **50** (1991).

[15] Papazoglou, M. P., "Service-oriented computing: Concepts, characteristics and directions," in [*Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*], 3–12, IEEE (2003).

[16] Fielding, R. T. and Taylor, R. N., [*Architectural styles and the design of network-based software architectures*], vol. 7, University of California, Irvine Doctoral dissertation (2000).