An Integrated Testbed Environment for the Web of Things

Mina Younan
Computer Science Department
FCI - Minia University
Minia, Egypt
E-mail: mina.younan@mu.edu.eg

Sherif Khattab, Reem Bahgat
Computer Science Department
FCI - Cairo University
Cairo, Egypt
E-mails: {s.khattab, r.bahgat}@fci-cu.edu.eg

Abstract – **This paper proposes a testbed architecture for the Web of Things (WoT) using simple components. Sensor networks have become one of the most researched topics currently, due to proliferation of devices equipped with sensors and actuators for monitoring and controlling their surrounding environment (e.g., places and devices). Although simulators, like Cooja, and Web sites, like Thingspeak, give the ability to build simple Internet of Things (IoT) and WoT applications, they are not compatible with many testing purposes in WoT. Getting real datasets that cover the main features of WoT is one of the most important factors in WoT testing and research. The proposed testbed environment allows for generating datasets and using them offline and online. It integrates small equipment elements (sensors and actuators) that convert things into Smart Things (STHs). Moreover, it augments IoT by STHs virtualization through Web applications. The main components and detailed design of the testbed are described. Then, a case study of searching for STHs and Entities of Interest (EoIs) is explained using a real dataset generated from the proposed testbed. The proposed testbed architecture incorporates the sense of smart environments and, hence, is expected to enhance testing results in WoT.**

Keywords – Internet of Things (IoT); Web of Things (WoT); searching in WoT; Smart Things; Test Environment.

I. INTRODUCTION

The number of devices and things connected to the Internet will be increasing and is expected to reach the order of billions by 2020 [1][2], as soon as the Internet Protocol (IP) becomes the core standard in the fields of embedded devices. As a result, the number of Internet users will be less than the number of devices connected to it. The Internet of Things (IoT) focuses on the infrastructure layer needed for connecting things and devices to the Internet. IoT addresses the connectivity challenge by using IP and IPv6 for embedded devices (i.e., 6LoWPAN) [3]. Sensor networks have become one of the most researched topics currently [2] [4]. This is due to the proliferation of devices equipped with sensors and actuators that provide information about and control of their surrounding environments. Sensors allow the state of things (e.g., places, devices, etc.,) that sensors represent to be inferred. In a sense, sensors and actuators convert things to Smart Things (STHs) and things' environments to smart spaces.

The Web of Things (WoT) virtualizes the IoT and focuses on the application layer needed for building useful applications over the IoT. Services, such as searching for STHs and Entities of Interest (EoIs) in the WoT, in addition to Web-based applications for controlling and monitoring services in smart spaces using friendly user interfaces are core power features in the WoT. However, there is no general method for testing and benchmarking research in IoT and WoT [4][5][6][7].

Muhammad et al. [6] summarize differences between concepts of emulators, simulators, and physical testbeds. They concluded that physical testbeds provide more accurate results. MoteLab [4] is a testbed for Wireless Sensor Networks (WSNs). It addresses challenges related to sensors' deployment and the time consumed for building a WSN. It features a Web application to be accessed remotely. The need for WSN testbeds is highlighted by challenges and research topics, which shed light on a specific set of features to be embedded within the testbed and its tools [6][7]. For instance, not only datasets about sensor readings are needed but integrating the readings with information about the underlying infrastructure (i.e., the IoT layer) is needed as well; this integration is the goal of the testbed proposed in this paper.

This paper proposes a testbed architecture for WoT. It addresses the general needs of WoT testing and focuses on the Web search problem and its related issues, such as crawling (i.e., preparing WoT pages for crawling). The problem of how to find STHs and EoIs that have dynamic state that change according to environment events [8][9] has sheer importance in drawing conclusions, deductions, and analysis in various fields. The proposed testbed can be used as a WoT application, which monitors real devices in real-time and can be used as a WoT simulator to do the same process on WoT datasets instead of devices. It aims at collecting datasets that contain information about things (i.e., properties and readings) formatted using multiple markup languages. The collected datasets are designed to help in testing in many problem domains [8][10].

The remainder of the paper is organized as follows. The next section defines dataset requirements. In Section 3, the related work of creating searchable IoT and WoT domains using IoT and WoT simulators and datasets is discussed. Section 4 describes the proposed system architecture. In

 ISBN: 978-1-61208-404-6

Section 5, the implementation of the proposed system is described followed by a case study. Finally, conclusions and important ideas for future work are presented in Section 6.

II. DATASET REQUIREMENTS

Things, SThs, resources, and EoIs are main concepts in IoT and WoT. They have differences in meaning but the main goal is that they are used for integrating the physical world into the virtual world [11]. In WoT, what needs to be retrieved (i.e., searched for) includes SThs (e.g., TV sets), EoIs (e.g., buildings), IP addresses, current values [8], and general information like device's web banners [9]. Generally, searching is done on SThs and EoIs that have dynamic locality [8][12][13] caused and fired by other events or objects in the network. For testing and evaluating the search process in IoT and WoT, simulators should reflect as many IoT and WoT challenges to achieve accurate results. To achieve this, datasets are used and replayed by applications that act as emulators of WoT.

In general, the main challenges that face testing of smart spaces, IoT, and WoT are: (1) the huge number of sensors and SThs, which makes communication services, monitoring, and analysis of sensor information require non-trivial amounts of CPU time and storage, (2) the dynamic state of SThs, which means that sensor readings and information about SThs properties and devices to which they are attached are in continuous change, and (3) the non-standardized naming of SThs properties (e.g., name, services, and location) and formats (e.g., microformats and microdata) that are used in WoT applications, which makes retrieval of information about objects and their attached sensors difficult.

To identify dataset requirements for testing the Web search process in WoT, WoT data are classified according to type, static or dynamic. The first type is **static information (IoT level)**, which includes (1) information about *sensors* (e.g., information about sensor properties like ID, name, brand, image, description, authoritative URL, manufacturer, and list of services that it offers) and (2) information about *entities* (e.g., device, thing, and place) including entity properties, such as logical paths, list of hosted devices, and possible states by which the entity is described. The second WoT data type is **dynamic information (WoT level)**, which includes (1) sensor readings or state, (2) current entity state, which changes according to sensor reading or other factors that the entity state depends on [8][14]. A dataset for testing Web search in WoT should ideally contain the following items: (1) files that contain schematics of the buildings and locations of sensors, (2) files that contain other static information about sensors (written in different formats), (3) a file for each sensor type that contains a table for readings of all sensors that have that type as a time series to aid in testing sensor similarity search and analysis [15], and a file for all devices in the network that contains sensor readings as a time series so that it can be used for browsing the WoT. Examples of these files will be described later. For accessing these files, headers of tables (sensor definitions) should follow a certain structure for creation and accessing (e.g., sensor name and virtual and physical location).

III. RELATED WORK

In the light of the previous requirements, this section discusses the usage of sensor datasets in the literature. To summarize our observations, if the research is only interested in values measured by sensors or in states of EoIs (e.g., being online or offline), then the used dataset is based on the WoT level, whereas if the research is interested in the sensor network infrastructure, then the used dataset is based on the IoT level. An integrated dataset contains information about both sensor readings and network infrastructure, that is, it is based on both IoT and WoT levels.

A. IoT Simulations

There is no general way for simulating IoT [5][6][16]. Moreover, there are situations in which simulators and real datasets containing raw information (e.g., sensor readings [17]) or information about the IoT layer are not enough for modeling an environment under testing, as the datasets miss the sense of one or more of the challenges mentioned earlier and thus, miss the main factors for accurate WoT evaluation. Also, many datasets are not actually related to the problem under investigation, but were generated for testing and evaluating different algorithms or methods in other researches. For instance, an evaluation of WSNs' simulators according to a different set of criteria, such as Graphical User Interface (GUI) support, simulator platform, and available models and protocols, concludes that there is no general way for simulating WSNs, and hence IoT and WoT [5][16]. None of these criteria address the previous challenges. So, it is desirable to embed the unique IoT and WoT challenges within datasets and to make simulators support as much of these challenges.

WSN Simulators. Several studies [5][6][16] summarize the differences between existing simulators according to a set of criteria. The Cooja simulator is one of the most valuable tools [5][16] in WSNs that aids researchers to simulate WSNs relatively easily using a supported GUI. Cooja allows to add different types of sensors (motes) and to attach them to binaries or source codes that have been previously developed. Cooja supports applications (e.g., written in the nesC [18] language after building in the TinyOS [19]) for different sensor targets. For example, the RESTful client server application [20] simulates a simple IoT. Cooja has a main advantage that it allows users to create their network using a non-trivially large number of sensors with different types, to get information about sensors (readings and properties), to control sensors, and to change their states as well. However, there are limitations and difficulties for testing the extensible discovery service [10] and sensor similarity search [15] in Cooja, because there is no information about

 ISBN: 978-1-61208-404-6

network infrastructure and entities, in particular static information about sensors, written in different formats, and schematics information of the buildings and locations of sensors.

WSN Physical Testbeds. Physical testbeds produce accurate research results [6]. Different testbeds are found in this field due to different technologies and network scales. Providing a Web interface for users is a main feature in testbeds. MoteLab [4] supports two ways for accessing the WSNs, (1) offline, by retrieving stored information form a database server and (2) online, by direct access to the physical nodes deployed in the environment under test. Datasets can be downloaded from MoteLab's Web site. However, the WoT challenges mentioned previously are not fully supported in MoteLab. User accessibility in MoteLab is similar to what is done in the proposed testbed.

SmartCampus [21] tackles gaps of experimentation realism, supporting heterogeneity (devices), and user involvement [7] in IoT testbeds. CookiLab [22] is another WSN testbed. It gives users (researchers) the ability to access real sensors deployed in Harvard University. However, it does not support main WoT features, such as sensor formats and logical paths as a property for sensor nodes and entities.

Nam et al. [23] present an Arduino [24] based smart gateway architecture for building IoT. Their architecture is similar to the architecture of the testbed environment proposed in this paper. For example, both their approach and ours use periodic sensor reporting. The Sense Everything, Control Everything (SECE) server stores information sent by Arduinos to be accessible anywhere at any time. Also, they provide the 'Bonjour' application that discovers all connected Arduinos and lists the devices connected on each Arduino. The information is sent in JavaScript Object Notation (JSON) format back to the application [23]. However, the framework does not cover all scenarios that WoT needs, especially for searching. For example, information of logical paths and properties of entities and information of the devices that the components simulate or measure are missing. At Intel Berkeley research lab [17], 54 sensors were deployed, and sensor readings were recorded in the form of plain text, which can be used as a dataset for sensor readings.

B. WoT Simulations

Using websites (e.g., [25][26][27]), a WoT environment can be built online by creating channels then attaching them to SThs like Arduinos or other devices equipped with sensors. The devices send information to the attached channels using private keys that are generated by the website. The website receives information from the attached resources to monitor the states of devices or entities that the resources represent. These websites provide RESTful services (GET, PUT, UPDATE, DELETE) [28] for uploading and accessing reading feeds. Moreover, the values (sensor readings) are visualized for users.

The services and design of the aforementioned websites is similar to our proposed testbed environment. However, these websites are limited by available service usage and formats of the responses, which are hardcoded and embedded within website code or at least not exposed to users. The proposed testbed architecture, which is built specially for testing WoT, provides more general services, such as monitoring live information fed from attached SThs, visualizing sensor readings and states of EoIs over time, controlling actuators, triggering action events, and periodic sensor reporting.

C. Services Architecture for WoT

Web services are considered as the main method for accessing WoT devices [15]. Mayer et al. [14] propose a hierarchical infrastructure for building WoT to enhance the performance of the searching service. Nodes receive queries then pass them to the right nodes in the network to answer the queries. The searching scenario starts by getting a list of sensors that can answer a query according to their static properties and predicted values. After that, the identified sensors are queried to check their current values, which are used for ranking the search results. The searching scenario is integrated into the proposed testbed.

Mayer and Guinard [10] and Mayer [29] provide a method for solving the problem of using multiple formats (e.g., microformat and microdata) in the WoT. They propose to add multiple strategies for parsing and producing information in the intended format. However, their work does not result in a dataset. They implemented an algorithm [10], called extensible discovery service, as a Web application that asks users about sensor page URL and retrieves information about devices if and only if the page is written in one of a set of pre-defined formats. Our proposed testbed allows such an algorithm to be tested to measure its performance. The required dataset contains sensor information written in different formats so that the algorithm is tested in parsing and retrieving information about sensors and entities.

To summarize, none of the datasets or testbeds used in the literature fulfills the full requirements for testing and evaluating the Web search process in the WoT as mentioned in Section II. Our proposed testbed environment aims at filling this gap. It is not the main focus of this paper to propose a new WSN testbed. Our main goal is to integrate WoT features above the layer of the IoT for visualizing things and entities, retrofitting on the benefits of existing physical testbeds.

IV. TESTBED ARCHITECTURE

The proposed testbed architecture transforms the physical control of devices in a surrounding physical environment to an emulated control for those devices keeping the same sense of events and features that existed in the physical environment. These events and features are embedded in datasets that can be later replayed. The

 ISBN: 978-1-61208-404-6

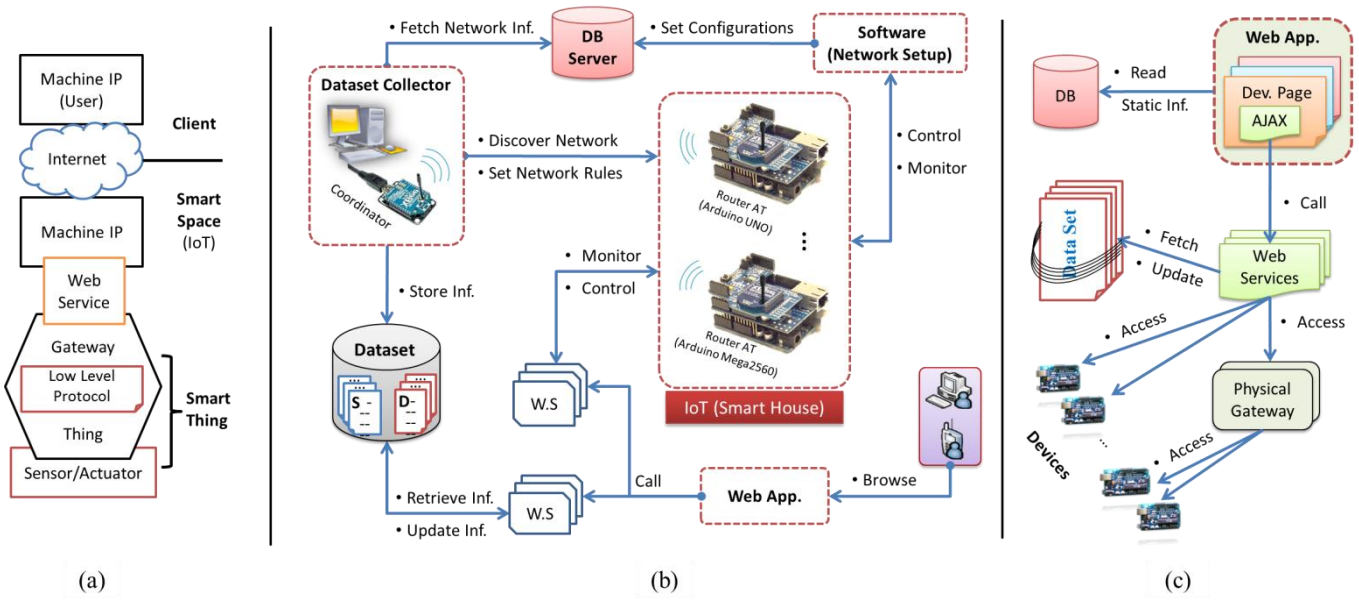


Figure 1. Testbed Architecture: (a) Integrating smart things (STHs) in the IoT - (b) Testbed environment architecture for simulating a physical environment - (c) Web services fetch data from real devices and gateways (online mode), or from dataset files (offline mode).

proposed architecture has two modes of operation: online and offline (Figure 1 (c)). In online mode, datasets are generated, "real" physical information is recorded, and a Web application offers WoT services by accessing the real devices for monitoring and controlling them. In offline mode, the Web application accesses the datasets to replay the events monitoring information.

The testbed architecture, shown in Figure 1 (b), is divided into five parts, as follows**.**

An IoT infrastructure (e.g., modeling a smart home). To build the IoT [12], the steps are briefly as follows. First, things are converted to STHs by attaching smart equipment (e.g., sensors and actuators), as shown in Figure 1 (a). Second, the static and dynamic information of STHs is described. STHs representation specifies URLs to invoke STHs services and their parameters and response format [29]. Third, RESTful APIs for accessing the STHs are built. Fourth, communication protocols between STHs and gateways are developed. Fifth, the STHs are connected to the Internet using physical and virtual gateways. STHs integration is done in the form of (1) direct integration, for STHs that support IP address for connection or (2) indirect integration using gateways, for STHs that use low-level protocols [13][30].

Network setup software, after building IoT, a program is built for configuring the IoT network. It assigns locations to STHs in the hierarchical structure of the simulated building or environment shown in Figure 2. This allows for using the generated logical path as attributes for the STh.

Web services for each device are used for executing WoT services directly and for feeding back users with information about STHs, such as indicated in Figure 1 (a). The web services are hosted on machines that support IP connection, either the STh itself or a physical gateway for accessing STHs that use low-level protocols.

A **Web application** offers WoT services like monitoring and controlling. The application loads information by calling web services, which pull information from devices (online mode) or from WoT dataset files (offline mode), as shown in Figure 1 (c).

The **dataset collector** discovers all available gateways and list of devices connected on each one, sets rules by which

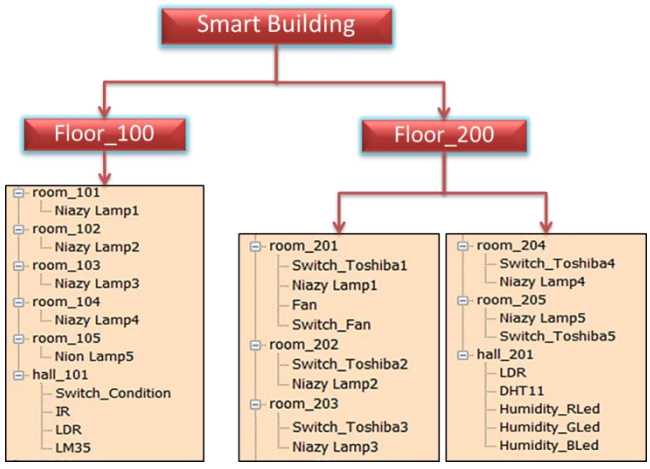


Figure 2. WoT graph for locating devices at specific paths in the hierarchical structure of a building.

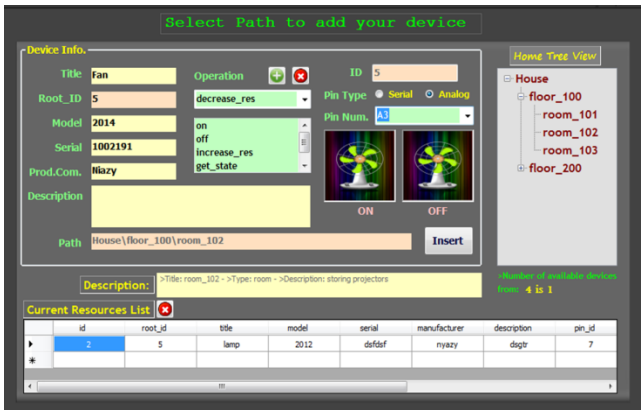


Figure 3. Locating and configuring a fan device at logical path 'floor_100\room_102.'

```
void loop(){
  ...
            else if(strcmp(buffer,"GET# ") == 0)
                        send_sr_get(client);
            else if(strcmp(buffer,"POST# ") == 0)
                        send_sr_post(client);
  ...  }
void send_sr_get (EthernetClient client)  {
            // send a standard http response header
             client.println("HTTP/1.1 200 OK");
             client.println("Content-Type: text/html");
             client.print("Value of ");
             client.print(default_Dpin);
             client.print(" is ");
             client.print(digitalRead(default_Dpin));
  ...  }
```

Figure 4. Device network protocol for handling incoming requests of monitoring, controlling, and crawling services (RESTful service).

data are collected from them, and sets the format by which the datasets are generated.

V. TESTBED IMPLEMENTATION

According to the testbed architecture presented in the previous section, testbed implementation was done along four axes.

A. *Building the IoT infrastructure*

This step will be executed the first time around; but, if a dataset that is generated by this testbed exists, then building WoT begins from the next step by attaching the dataset with the web application to work in offline mode.

Building IoT was done in a simple way [31] using widely-available components. The SECE server [23] gets information from the IoT according to events and actions that happen in the environment. It offers the collected information in a friendly user interface. A testbed environment for the WoT is built using these connections.

Building the IoT infrastructure was done in three steps: (1) connecting devices, (2) building the network setup software, and (3) implementing device communication protocols. Whereas it is desirable to build IoT using devices that support direct IP connection rather than devices that support only low-level protocols, the latter devices needed gateways for integrating them into the IoT. The IoT infrastructure was built using Arduinos, on which sensors and actuators were connected. Arduino has two interfaces: a Serial Peripheral Interface (SPI) bus and an Internal Integrated Circuit (I2C), which allows modules, like Ethernet and Secure Digital (SD) cards, to communicate with the microcontroller [32]. The Arduinos connected more than one device using digital and analog pins. In a sense, the Arduinos acted as physical gateways and IP addresses were set for them. They were attached to the network using Ethernet or XBee [33] connections.

Network setup software was written in C# for locating, managing and configuring resources for each virtual gateway. A virtual gateway represents a location, such as floor_100 and floor_200. For example, Figure 3 shows the process of adding a new device to the testbed using the software. Logical paths in the building hierarchical structure are very important for accessing devices.

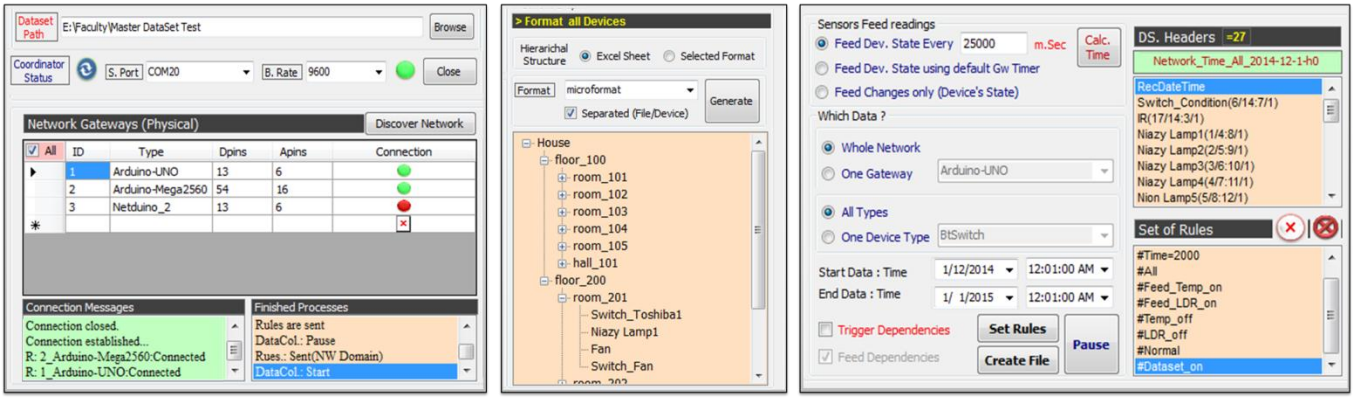
The **protocols**, written in Arduino Sketches [24], were used to get and set the state of devices that are connected to the Arduinos, whereby get and set requests were sent within the body of the protocol messages. When the special symbol '#' is found within the body of the message, as shown in Figure 4, the spider gets the current device's states. The crawling case involved only getting information, not controlling or changing device states.

B. *From IoT to WoT*

Building Web pages in the testbed followed standard features for dealing with dynamic information. The common way for developing dynamic websites depends on AJAX. AJAX is used for live update of some parts in the sensor's pages. The dynamic parts typically include STHs readings or entity states, which indirectly depend on sensor readings [25][27].

However, pages with dynamic content built using AJAX cannot be crawled by traditional search engine crawlers. Some search engines, such as Google, suggest practical solutions for optimizing the crawling process [34] of dynamic content. Alternative URLs that lead to pages with static information are indexed by default or instead of pages that contain dynamic information. According to Google optimization rules, Web sites in our testbed use AJAX in some parts in device's web page but for crawling, corresponding Web services are accessed instead to get current STh value or EoI state, in addition to all possible states with corresponding occurrence probabilities. Another technique not implemented in our testbed is to render pages on the fly (i.e., crawlers have browsing processes embedded in their code [35]). Still, it is difficult to crawl pages that need to send some data first before loading their content. Moreover, the time consumed by the crawling process itself becomes high and the crawling process needs to be done

 ISBN: 978-1-61208-404-6



(a) (b) (c)

Figure 5. Dataset collector software: (a) Dataset collector program discovers gateways in WoT loading a list of devices connected on them. (b) Static information about the IoT testbed are generated in different formats. (c) Rules are defined to control the way gateways send dynamic information (readings).

frequently; information in WoT may be updated in less than a minute.

Using Ethernet, RESTful APIs can access Arduino components. Devices are programmed as *clients* to push sensor data to some services and as *services* to enable remote control of devices over the Web. Because it is desirable to have a Web page for each device and because Arduino acts as a gateway for managing at least one component, a website is built and can be hosted on an SD card connected to the Arduino. The website is accessed using an IP address, assigned to the Arduino. Another alternative is to host the website on a different server for adding more capabilities like storage capacity. In the latter case, Arduinos are accessed using RESTful APIs. The selection of either alternative is determined by the amount of information that needs to be stored and accessed over time.

Two steps were done to add WoT layer to the testbed. First, a **Web application** was written in Asp.Net (Figure 6). The main services of the Web application are monitoring sensors, controlling actuators, triggering action events, and

Figure 6. Web pages of virtual gateways get their information from a database server. Sensor Web pages get their information either from direct access to devices or from the offline dataset.

periodic sensor reporting [12][23][30]. The WoT application was built according to the building hierarchical structure configured by the network setup software. The homepage shows general information and allows users to perform general tasks, such as monitoring room status. The user selects a logical path to browse, then, for each room, a list of devices and their states appear. The user selects a device to access. The device page loads the RESTful services dynamically (using WSDL [36]) according to the Arduino IP and selected device ID. Second, a set of **Web services** were written in C#. The Web application loads the available RESTful services dynamically for each device. A special tag *'GET#'* is added as an additional service that is executed by default for the device webpage. The crawling process returns the current sensor value or the state of the device and all possible states with their probabilities.

C. Dataset Collector (DsC)

In Figure 5 (a), using Zigbee connection, the WoT coordinator (gateway that acts as a base station) discovers all available gateways, getting a list of connected devices on each gateway. The dataset collector (DsC) program generates files written in different formats for the static information of the IoT testbed including the building hierarchy and the devices located in the hierarchy (Figure 5 (b)). The dynamic information is collected using a set of rules, as in Figure 5 (c). The rules instruct the gateways to send back specific information about a specific list of devices according to a specific action or event done by other devices. The gateways feed the DsC with device readings according to these rules. If the rule '*ChangesOnly*' is selected, the DsC stores only changes on device state. If the rule '*TimeSlot*' was selected, the DsC stores periodical feeds of device state. One of the most important rules is that if a certain device type is selected for analysis of device readings and making decisions according to the analysis results, rules can be set to collect data from all devices of that type across all the gateways in WoT.

 ISBN: 978-1-61208-404-6

```
<div class ="hproduct ">
        <span class =" fn">22_Fan</span>
        <span class =" identifier">,
            <span class =" type ">Fan</span> Sfan123
            <span class =" value ">0</span>
        </span>
        <span class =" category ">
            <a href =http://www.XXX.com   rel =" tag"> Fan </a>
        </span>
        <span class =" brand ">Brand Name</span>
        <span class =" description "> characterized by …
        </span>
        <span class =" Photo ">Fan</span>
            <a href =http://www.XXX.com/?s=wsn
                class =" URL">
                    More information about this device.
             </a>
</div>
```

Figure 7. Static information about a fan written in microformat.

D. Generated Dataset Files

A simple dataset was generated by the testbed according to the rules: (1) '*every_2500 msc*' for updating dataset every 2500 millisecond (i.e., DsC pulls information from the network), it could be replaced by rule '*ChangesOnly*' for storing changes on devices' states only (i.e., devices push information to DsC), (2) '*All_Network*' for pulling information from all discovered gateways in the network, (3) '*All_types*' means all devices on selected gateways, (4) '*2014-12-1-h0_to_2015-1-1-h0*' for storing dataset from '1/12/2014' to '1/1/2015', and (5) '*TD*' for triggering all dependences related to selected devices. The dataset generated according to limited time slot (date and time) by DsC, as shown in Figure 5 (b), contains static information about IoT infrastructure and dynamic information about sensing and actuating activities.

The static information of each device, such as logical path and device type, is stored in a file named using the device ID, the EoI ID, and device name (e.g., *22_9_Fan*). Static information about a fan written in microformat is shown in Figure 7.

The dynamic information, such as sensor readings, is stored in a file named using the collection-rule title and the date and time of collection (e.g., *Network_Time_All_2014-12-1-h0*). This file contains readings collected from all devices in the WoT testbed. A subset of data stored in that file would look like Table 1, where monitoring is set to rule '*time only'*.

As mentioned before, sensor definition contains information about sensor so that it can be accessed easily through built web application that simulate the WoT. In sensor definition: '*Xlamp2(2/5:9/1)*', X is sensor's brand name, lamp2 is sensor title, (2/5) is the virual location sensor id and hosting room id, and (9/1) is the physical location where 9 is pin number and 1 is gateway id. Column '*Time*' is the response time. Arduinos support 5 Voltages as maximum; they convert voltage range (0:5V) to be (0:255) using built in analog to digital converter. Values recored for each sensor definition, are current voltages consumed by that sensor (0 : 255).

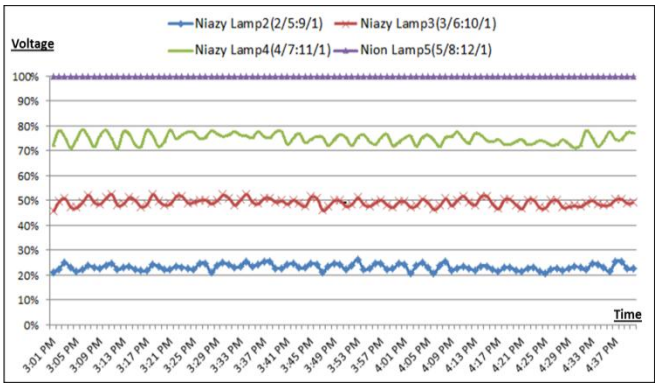


Figure 8. A graph generated out of a dynamic data file collected from devices of the same type (lamp in this graph). Voltages consumed by devices are represented in percentage at Y axis.

TABLE 1. SAMPLE READINGS GENERATED FROM ALL SENSORS OF TYPE 'LAMP' IN THE ENTIRE NETWORK AS A TIME SERIES. CONSUMED VOLTAGES ARE MAPPED FROM (0:5) TO (0:255).

Time	XLamp2 (2/5:9/1)	XLamp3 (3/6:10/1)	XLamp4 (4/7:11/1)	XLamp5 (8/8:12/1)
03:01 PM	66	77	83	71
03:02 PM	66	80	83	68
03:04 PM	66	68	65	69
03:06 PM	67	71	69	67
03:08 PM	68	80	85	70
03:10 PM	69	81	86	70
…	…	…	…	…
10:12 PM	65	80	85	70

VI. CASE STUDY

In this section, a case study of the proposed WoT testbed is described.

A. Using WoT Dataset for information analysis

Using the generated dataset, researchers can analyze sensor data collected using multiple controlled scenarios. A lot of experiment scenarios can be achieved on the testbed, such as comparing the state of devices on certain gateways (e.g., gateways of a room), comparing state of devices on all gateways (e.g., all gateways of a building), getting live time of each device and high level power consumption in daily live to provide suggestions related to energy efficiency achievement (i.e., Energy awareness through interactive user feedback). Figure 8 shows a comparison between devices of type '*Lamp*' for analysis of power consumption. Y-axis represents consumed voltages percent and X-axis represents time. Estimation on timing accuracy of the data hasn't been measured yet, but after enlarging WoT scale, DsC can estimate timing accuracy by calculating request and receive time for each device. In general, such a dataset, especially composed of dynamic information, will be helful for computing Fuzzy-based sensor similarity [15], and for running prediction algorithms on real information that are used in searching about STHs and EoIs in the WoT.

 ISBN: 978-1-61208-404-6

```
Select   [Device_Header]
From     [Sheet_Title]
Where    [RecDateTime] = (Select     min ([RecDateTime])
                          From       [Sheet_Title]
                          Where      [RecDateTime]
                          Between    @Date_1 and @Date_2)
```

Figure 9. Accessing dataset files using web services (offline mode): Selecting column 'Device_Header' from sheet 'Sheet Title' where its time = current system time (hours and minutes) using OleDbCommand.

Providing information about SThs and EoIs in multiple formats with additional attributes like logical paths expands experimental work in this area.

B. Browsing WoT

Building simple and physical WoT (offline or online) will be helpful and more accurate than using simulators. Figure 1 (c) shows a scenario of calling RESTful web services for pulling information about buildings and their devices from the generated dataset (offline). Sensor pages call Web services that fetch information from a dataset file *'Network_Time_All_2014-12-14-h17.xlsx'* using command of type *OleDbCommand*. Web services called in the testbed (Figure 1 (c)) execute the command string shown in Figure 9. *'Device_Header'* and *'Sheet_Title'* were sent by calling pages to the Web service *monitorin*g. The special character @ before variables *'Date_1'* and *'Date_2'* means that they are initiated within the Web service.

C. Reusing Testbed for Different IoT

The proposed testbed architecture allows the implementation of different purposes in the WoT. If someone has to operate the testbed for a certain environment (for example, energy saving of smart home, detect something unacceptable happening at a shopping mall, etc.), and because the proposed testbed operates in two modes (online and offline), then reusing this testbed is restricted with operation mode; For online mode, new IoT infrastructure, which is built by attaching resources support information about measuring physical phenomena and actuating EoIs, is replaced by the IoT part shown in Figure 1 (b) and registered by **Network setup software**. New IoT should speak the same language as the DsC (gateways make it easy for supporting heterogeneity in devices); But for offline mode, such as shown in Figure 10, because the dataset represents the IoT itself where it hosts information about SThs, EoIs, and sensing and actuating processes, then IoT part will be replaced by that dataset to be accessed by web services as indicated in Figure 1 (c), so Offline mode could be used for retesting previously built IoT.

VII. Conclusion and Future Work

WoT has become one of the most trendy research directions due to facilities and services provided in many

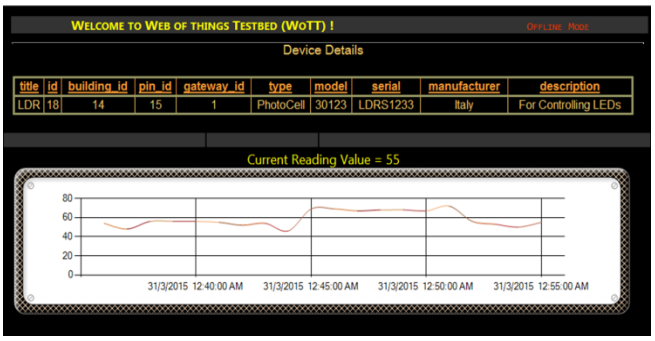


Figure 10. Testbed is running in offline mode (attaching IoT dataset).

domains. Sensors can provide great benefits if their readings are presented in a meaningful and friendly way to users and machines. Searching for SThs and EoIs is one of the most important services in the WoT. But, most of the work in this area focuses on searching in a single environment. In other words, the WoT is built using single formatting and network infrastructure. In this work, a WoT testbed is proposed to be built in a simple way with readily-available physical components. The proposed testbed allows capturing - and maintaining for offline usage - the sense of events and actions in the environment under test. The testbed allows for building a WoT environment according to a hierarchical architecture, providing description for components in a way that gives search engine spiders the ability to crawl them in addition to the ability given to users to perform live monitoring of their environment. The dataset generated from the testbed is expected to help research on the crawling, indexing, and searching processes in WoT in general.

The problem of searching about SThs depends on the standardization of formats used for representing SThs (properties and services they offer). So, providing semantic discovery services based on application of multiple discovery strategies [10] and enriching SThs metadata may enhance results of searching and lookup services in the WoT. Creating standardized RESTful service description embedded in HTML representation using microdata is feasible and desirable [29]. Still a few important questions remain here: what is the timing accuracy of the data, what information needs to be indexed, and how to index WoT data streams.

References

[1] M. Blockstrand, T. Holm, L.-Ö. Kling, R. Skog, and B. Wallin, "Operator opportunities in the internet of things – getting closer to the vision of more than 50 billion connected devices," [Online] Feb. 2011, http://www.ericsson.com/news/110211_edcp_244188811_c, ,(accessed: 10 Feb. 2015)

[2] L. Coetzee and J. Eksteen, "The Internet of Things – Promise for the Future ? An Introduction," in IST-Africa Conference, Gaborone, May 2011, pp. 1-9.

[3] Ch. Lerche, "WS4D-uDPWS - The Devices Profile for Web Services (DPWS) for highly resource-constrained devices," WS4D Initiative, [Online] Aug. 2010, http://code.google.com/p/udpws/wiki/IntroductionGeneral, (Accessed: 10 April 2015).

[4] G. Werner-Allen, P. Swieskowski, and M. Welsh, "MoteLab: A Wireless Sensor Network Testbed," in Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on, Boise, ID, USA, 2005, pp. 483-488.

[5] H. Sundani, H. Li, V. K. Devabhaktuni, M. Alam, and P. Bhattacharya, "Wireless Sensor Network Simulators - A Survey and Comparisons," International Journal Of Computer Networks (IJCN), vol. 2, no. 6, pp. 249-265, Feb. 2011.

[6] I. Muhammad, A. Md Said, and H. Hasbulla, "A Survey of Simulators, Emulators and Testbeds for Wireless Sensor Networks," in nformation Technology (ITSim), 2010 International Symposium in, vol. 2, Kuala Lumpur, June 2010, pp. 897-902.

[7] A. Gluhak, S. Krco, M. Nati, D. Pfisterer, N. Mitton, and T. Razafindralambo, "A Survey on Facilities for Experimental Internet of Things Research.," IEEE Communications Magazine, Institute of Electrical and Electronics Engineers (IEEE), no. <10.1109/MCOM.2011.6069710>. <inria-00630092>, pp. 58-67, 2011, 49 (11).

[8] B. Ostermaier, B. M. Elahi, K. Römer, M. Fahrmair, and W. Kellerer, "A Real-Time Search Engine for the Web of Things," in The 2nd IEEE International Conference on the Internet of Things (IoT), Tokyo,Japan, Nov. 2010., pp. 1-8.

[9] Shodan, "The search engine for the Internet of Things, " [Online] 2015, https://www.shodan.io/, (Accessed: 10 April 2015).

[10] S. Mayer and D. Guinard, "An Extensible Discovery Service for Smart Things," in Proceedings of the 2nd International Workshop on the Web of Things (WoT 2011), ACM, San Francisco, CA, USA, June 2011, pp. 7-12.

[11] S. Haller, "The Things in the Internet of Things," Poster at the (IoT 2010). Tokyo, Japan, vol. 5, no. 26, p. 4, Nov. 2010.

[12] D. Guinard, "A Web of Things Application Architecture - Integrating the Real-World into the Web," PhD Thesis, Computer Science, Eidgenössische Technische Hochschule ETH Zürich, Zürich, 2011.

[13] D. Guinard, V. Trifa, S. Karnouskos, and D. Savio, "Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services," Services Computing, IEEE Transactions on, vol. 3, no. 3, pp. 223-235, Sep. 2010.

[14] S. Mayer, D. Guinard, and V. Trifa, "Searching in a Web-based Infrastructure for Smart Things," in Proceedings of the 3rd International Conference on .,the Internet of Things (IoT 2012),IEEE, Wuxi, China, October 2012, pp. 119-126.

[15] C. Truong, K. Romer, and K. Chen, "Sensor Similarity Search in the Web of Things," in In World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium, San Francisco, CA, June 2012, pp. 1-6.

[16] J. Miloš , N. Zogović, and G. Dimić, "Evaluation of Wireless Sensor Network Simulators," in the 17th Telecommunications Forum (TELFOR 2009), Belgrade, Serbia, 2009, pp. 1303-1306.

[17] P. Bodik, C. Guestrin, W. Hong, S. Madden, M. Paskin, and R. Thibaux. "Intel Lab Data," [Online] Apr. 2004, http://www.select.cs.cmu.edu/data/labapp3/index.html, (Accessed: 10 April 2015).

[18] D. Gay, P. Levis, D. Culler, E. Brewer, M. Welsh, and R. von Behren, "The nesC language: A holistic approach to networked embedded systems," in PLDI '03 Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation, New York, NY, USA, May 2003, pp. 1-11.

[19] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, and A. Woo, "TinyOS: An Operating System for Sensor Networks," in Ambient Intelligence, W. Weber, J. M. Rabaey, and E. Aarts, Eds. Springer Berlin Heidelberg, 2005, ch. 2, pp. 115-148.

[20] Hosted by Thingsquare, "Contiki: The Open Source OS for the Internet of Things, " [Online] 2012, http://www.contiki-os.org/, (Accessed: 10 April 2015).

[21] M. Nati, A. Gluhak, H. Abangar, and W. Headley, "SmartCampus: A user-centric testbed for Internet of Things experimentation," in Wireless Personal Multimedia Communications (WPMC), 2013 16th International Symposium on, Atlantic City, NJ, June 2013, pp. 1-6.

[22] G. Mujica, V. Rosello, J. Portilla, and T. Riesgo, "Hardware-Software Integration Platform for a WSN Testbed Based on Cookies Nodes," in IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society , Montreal, QC, October. 2012, pp. 6013-6018.

[23] H. Nam, J. Janak, and H. Schulzrinne, "Connecting the Physical World with Arduino in SECE," Computer Science Technical Reports, Department of Computer Science, Columbia University, New York, Technical Reporting CUCS-013-13, 2013.

[24] Arduino, "Arduino," [Online] 2015, http://www.arduino.cc/, (Accessed: 10 April 2015).

[25] LogMeIn, Inc., "Xively," [Online] 2014, http://www.Xively.com, (Accessed: 10 April 2015).

[26] XMPro, "Intelligent Business Operations Suite For The Digital Enterprise," [Online] 2015, http://xmpro.com/xmpro-iot/, (Accessed: 10 April 2015).

[27] IoBridge, "ThingSpeak- The open data platform for the Internet of Things," [Online] 2015. http://www.thingspeak.com, (Accessed: 10 April 2015).

[28] M. Elkstein, "Learn REST: A Tutorial," [Online] 2008. http://rest.elkstein.org/2008/02/what-is-rest.html, (Accessed: 10 April 2015).

[29] S. Mayer, "Service Integration - A Web of Things Perspective," in W3C Workshop on Data and Services Integration, Citeseer, Bedford, MA, USA, October 2011, pp. 1-5.

[30] D. Guinard and V. Trifa, "Towards the Web of Things: Web mashups for embedded devices," in in Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain, 2009, p. 15.

 ISBN: 978-1-61208-404-6

[31] C. Pfister, "The Internet of Things," in Getting Started with the Internet of Things: Connecting Sensors and Microcontrollers to the Cloud, B. Jepson, Ed. United States of America.: O'Reilly Media, Inc., 2011, ch. 4, pp. 29-41.

[32] A. McEwen and H. Cassimally, "Designing the Internet of Things," 1st ed., C. Hutchinson, Ed. John Wiley & Sons, ISBN: 1118430638;9781118430637, November 2013, https://books.google.com.eg/books?id=oflQAQAAQBAJ, (Accessed: 10 April 2015).

[33] Digi International Inc., "Official XBee website- Connect Devices to the Cloud," [Online] 2015. http://www.digi.com/xbee, (Accessed: 10 April 2015).

[34] Google, "Search Engine Optimization (SEO) - Starter Guide," Jan. 2010.

[35] P. Suganthan G C, "AJAX Crawler," in Data Science & Engineering (ICDSE), 2012 International Conference on. IEEE, Cochin, Kerala, July 2012, pp. 27-30.

[36] wikipedia, "Web Services Description Language," [Online] Apr. 2015, http://en.wikipedia.org/wiki/Web_Services_Description_Language , (Accessed: 10 April 2015).

 ISBN: 978-1-61208-404-6