EWSN Tutorial: Programming the Internet of Things with Contiki and SicsthSense

Joakim Eriksson, Liam McNamara, Simon Duquennoy
and Thiemo Voigt
SICS Swedish ICT AB, Sweden
Email: joakime@sics.se ljjm@sics.se, simonduq@sics.se
and thiemo@sics.se

February 15, 2014

1 Introduction

This tutorial presents SicsthSense, an open cloud platform for the Internet of Things. SicsthSense enables low power devices such as sensor nodes and smartphones to easily store their generated data streams in the cloud. This allows the data streams, and their history, to be made permanently and globally available to users for visualisation, processing and sharing. Moving sensor data computation and monitoring into the cloud enables centralisation of control and redistribution of collected data.

We will now introduce SicsthSense incorporating real data collection and posting it to our datastore. This live data will then be visualised and made available for sharing between users of the platform. Our Android App will also be demonstrated to enable participants to stream their phone sensors into the system, demonstrating how simple it can be to start machine-to-machine interactions with SicsthSense.

We believe the following features are important for promoting easy development of applications in the realm of IoT, allowing developers to concentrate on their domain-specific problems rather than the "nuts and bolts" of making use of sensed data.

- Easily connect devices to the cloud;
- Store/retrieve sensor data in the cloud;
- Make decisions in the cloud;
- Visualise data;
- Actuate from the cloud.

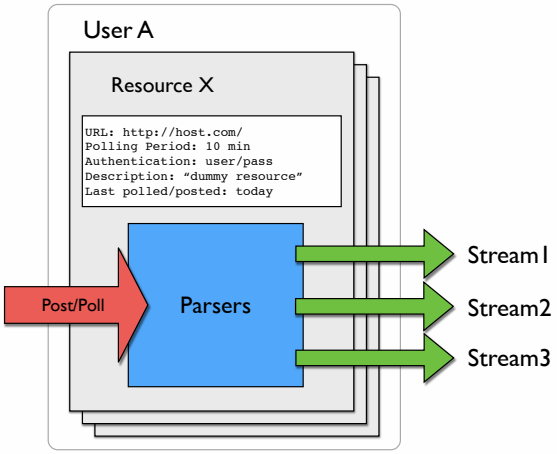

Figure 1: Overview

2 Overview

SicsthSense centers around the collection and processing of data *Streams*. These streams are a sequence of scalar numeric values arranged through time. Each stream is a logical grouping of some data point measurements that users are interested in (such as *temperature*).

Users can configure SicsthSense to populate these streams by its interaction with external *Resources*. A resource is a single source/device that provides information, such as a weather station. Each resource may provide multiple different data points upon each interaction with it. For example a weather station resource may provide a data payload containing the temperature, humidity and wind speed. This payload then has to be processed and split into three different streams for storage. This splitting procedure is performed by *Parsers* that specify which bits of data in a payload are of interest to the user. A diagram explaining the relationship between these entities is shown in Figure 1. In short, Users have many Resources, Resources contain Parsers and Streams. The Resource can be configured to interact with external devices and collect/receive data. This data is then processed by parsers and resulting values are added to time series of Streams.

Users can feed data into the system by either having Resources make periodic requests for data from a specified URL, whereby the system will *poll* it, or they can choose to perform their own HTTP POSTs of data to the Resource's own URL.

3 Architecture

The main SicsthSense platform is a Java-based server that was explicitly designed to be able to run on a wide range of devices, from powerful cloud-based servers to small home gateway machines. The platform has two components, a text-based backend *Engine* and a web-based visual interface. Both software servers are distributed as a single `.jar` files, to enable easy packaging and operation. All the logic, data processing and authentication is implemented in the *engine* sub-system which provides a RESTful HTTP interface for machine-to-machine interactions with SicsthSense on a separate network port from the website. The optional web server is provided so there is a user friendly interface to the system. This richer interface is particularly useful non-technical users and allows immediate visualisation of collected data in the SicsthSense system.

Communication with the API is performed using JSON to represent entities. When users need to request attributes of some part of our system (e.g. user definition or a data point), they simply perform an HTTP GET on a well-defined URL and receive a JSON representation of that entity. HTTP POSTs to the same URL can then reconfigure the system.

4 Web Interface

The web interface can be accessed through pointing your web browser (modern ones preferred for HTML5 features) at the IP/host and port number of the SicsthSense instance you would like to interact with. A global one is provided at `http://sense.sics.se` and a development version is running at `http://presense.sics.se`. Signing up is simplified through the use of OpenID[1], which is an open standard that allows users to be authenticated by certain cooperating sites (known as Relying Parties or RP) using a third party service. If you have an account with any of the listed providers (Google, Yahoo, etc.), just select them and authenticate through this 3rd party. Alternatively you can register with an email address and password combination.

The menu has three main sections in the top left of the screen – *Home* containing your list of favourite streams, *Resources* to list and add your own resources and *Streams* to list your streams. There is also a link to your user account page in the top right (see Figure 2). The user account page contains userID, email address and other personal details, many will be filled in by OpenID. For interacting with the system it is also useful to notice your personal authentication *key* is also listed here.

4.0.1 Exercise

- Register on the SicsthSense instance running at `HOST` and configure your user details.
- View public streams on the `Home` page.

[1] Website: `http://openid.net`

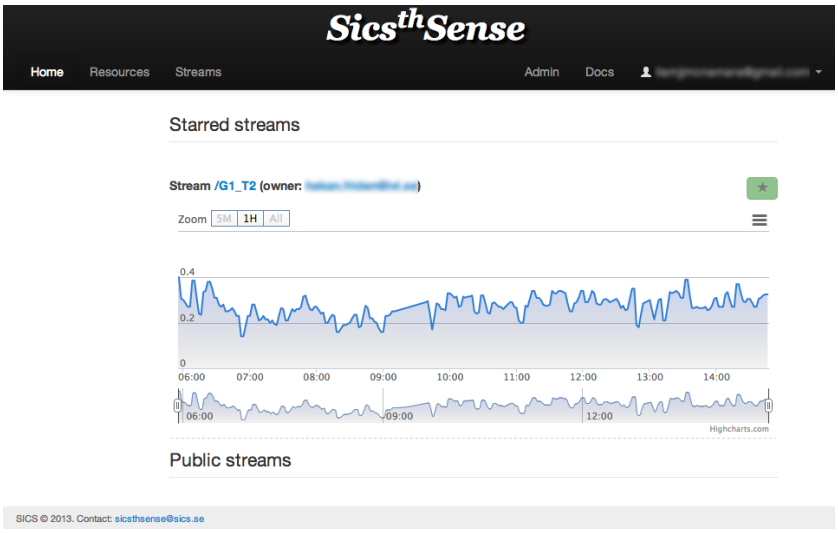

Figure 2: SicsthSense Website with some data streams displayed.

4.1 Making a resource

Resources handle the polling of external sources of information and organise similar groups of data streams together. Creating a resource only requires input of a unique *label* in order to identify it. At the top of the Resources page there is space to enter a label and URL of a new resource. The URL is optional and represents a possible location for SicsthSense to poll information from. Enter a label of your choice and click **Add**. A new resource has now been created. Click on its name from the Resources list to see detailed configuration information about this new resource. Obviously the resource configuration can be changed and saved or even deleted. The *API URL* gives the location of the resource in the Engine, if you would like to access it through the programmatic interface. An authorisation *key* for the resource is also listed. Using the URL and the key it is possible to POST data into the system. Through the assignment of a *Resource URI* and a Polling Period it is possible to have the resource periodically poll an external data source.

4.2 POSTing Data

Data can be sent into SicsthSense through the HTTP based Engine API. If Parsers or Streams have been created inside a new Resource then SicsthSense will automatically create them the first time a JSON document [2] is posted to the Resource. First make a note of the API URL from your new resource and append */data* to it. A JSON document can now be sent to this URL in order

[2]Basic introduction: http://www.regular-expressions.info/

to have data recorded. It is possible to POST JSON documents to this server using basic command line tools such as cURL or wget[3] or through any HTTP library in the programming language of your choice. In this tutorial we will use cURL for basic examples and Python for more advanced ones. This can easily be accomplished by the following command on a unix-like system:

```
$ curl -XPOST -H "Content-Type: application/json" -d '{"temperature":
20.1, "humidity": 10}' http://HOST/users/X/resources/Y/data
```

This command will create two new streams under the resource and two new parsers that feed into their respective streams and then POST the data points. Due to the two primitive numeric elements in the POSTed JSON document the system will create a parser for `/temperature` and `/humidity`. Any subsequent JSON posting will be checked for these two matches. Previously unseen identifiers will **not** cause new parsers and streams to be created, this process only happen on the first POST to an unconfigured resource. POST a few more data points to populate the graphs:

```
$ curl -XPOST -H "Content-Type: application/json" -d '{"temperature":
23, "humidity": 20}' http://HOST/users/X/resources/Y/data
$ curl -XPOST -H "Content-Type: application/json" -d '{"temperature":
21.1, "humidity": 25}' http://HOST/users/X/resources/Y/data
```

Viewing your *Streams* webpage will now list both automatically created streams and show the data that they contain. This data will only be visible to your user account, unless you enable the *public* options of the streams (the globe icon).

4.2.1 Exercise

- Create a new resource with just a label and no polling URL.
- POST a json document at the new resource's API data URL from the command line.
- POST a few more times with different values and view the results through the web interface.

4.3 Polling Data

Poll events can be scheduled by configuring the Resource. Go to a specific Resource's configuration page and simply specify the *Polling URL* and the *Polling Period*. SicsthSense will then perform an HTTP GET on that url every period seconds. Failed GETs will be ignored. If you specify an URL to a JSON document, you may use the *Auto add Parsers* button on the relevant Resource page which will create streams and parsers for all primitive elements in the retrieved JSON document.

[3]Command line HTTP clients - cURL website: http://curl.haxx.se/ wget website: https://www.gnu.org/software/wget/

4.4 Parsing

Upon receipt of a HTTP POST to the resource (or a periodic poll) each of the resource's parsers are applied to the whole data payload. The parsed data is then added to the data stream given by the parser' stream_id (the stream_id will be automatically matched if it was automatically generated).

The parsing system provides two methods to interpret values from a resource's pull/push data payload. First and most straight-forward is JSON parsing, where a parser JSON primitive has its complete path stated in a filesystem path style. For example, the floating point primitive called "temperature" at the top level of a JSON string would be referenced as "/temperature". If the primitive was contained in a JSON associative array called "readings" it would be called "/readings/temperature". Through this method, many different values can be parsed from a single JSON payload and turned into a stream in SicsthSense. This can be set in the parser representation through setting the primitive's path in the parser's Parser field and setting the content type to "application/json".

The other, more powerful, technique uses Regex pattern matching to enable parsing of arbitrary text payloads. The parser field need only be set to the regex that you would like to attempt to match against the payload. The content type should be set as "text/plain". Both a *value* and a *timestamp* can be captured by the given regex through the use of groups. By default, the first group is assumed to be the data value and the second group to be the timestamp.

These parsing methods provide both an easy and a powerful method to parse data entering the SicsthSense system.

4.4.1 Exercise

- Setup a new resource to poll JSON data from `http://HOST/data.json` (Use 'Auto add parsers' to simplify).
- (*Optional and only if online*) Setup a new resource to poll web page from `http://www.yr.no/place/United_Kingdom/England/Oxford/`. This will require a regex similar to: "(\\d+)°"

5 API

The programmatic interface to SicsthSense is through a HTTP server that represents data as JSON documents. It is running on the same host/IP but a different port number than the webserver.

5.1 URL structure

The structure of the URLs in SicsthSense reflects the relationship between entities. If `HOST` represents the hostname of the SicsthSense server, the user with userID X is located at `HOST/users/X`. All of their registered entities are hosted

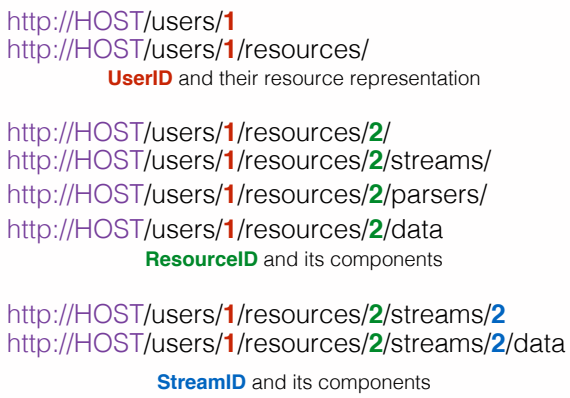

Figure 3: API URL structure arranged into groups for clarity.

under this URL. If they registered a resource which was given ID Y then the resource URL would be `HOST/users/X/resources/Y`, similarly a stream Z belonging to resource Y could be located at: `HOST/users/X/resources/Y/streams/Z`.

5.2 RESTful Interface

Representational State Transfer uses the powerful notion of information sources termed "resources" (though meaning something different from SicsthSense Resources). These entities are referenced with a global identifier (e.g., an URL) and correspond to our Users, Resources and Streams. In order to manipulate these entities, users can communicate over HTTP and exchange *representations* of these resources (documents containing all the entity attribtues). We use JSON as the default format of these representations. In short, configuring the system involves requesting, modifying and sending back JSON documents that describe Users, Resources and Streams. There are collection URLs such as `HOST/users/1/resources` (similar to directories) and element URLs such as `HOST/users/1/resources/9` (similar to files). HTTP request methods are use: GETs will simply return data and are idempotent, whereas POST's involves the modification of data.

5.3 Authentication

Data is not public by default, so users must provide credentials when accessing or changing a configuration or non-public data. Any configuration requires people to provide a key in the URL `http://URL?key=XXXXX`. A user's key works for any entities that they own, alternatively a resource's key or a stream's key can be used to only provide access to that specific entity. This enables you to give others access to specific parts of your data. There is a `Regenerate` button to enable key's to be remade in case you would like to revoke access.

5.4 GETing data from the system

A user's resources can be listed via an HTTP GET on the URL (replace X, Y or Z): `http://HOST/users/X/resources` or individual resources by appending their resource ID to the URL `http://HOST/users/X/resources/Y`. It is possible to request variable amounts of data points by using the query parameter `limit`. For example, to only get the last 10 data points, GET the following:

```
curl HOST:8080/users/X/resources/Y/streams/Z/data?limit=10
```

All data since a given timestamp can also be requested using the query parameter `from`. The supplied value should be a time in milliseconds since the beginning of 1970 UTC:

```
curl HOST:8080/users/X/resources/Y/streams/Z/data?from=1385763270458
```

The query parameter `until` can also be supplied to constrain the time period:

```
curl HOST:8080/users/X/resources/Y/streams/Z/data?
from=1385763000000&until=1385764000000
```

If both limit and from/until parameters are given, the limit option will take precedence. If you would like data streams to be sent in CSV format, then simple append the following to the GET URL: format=csv For instance requesting the URL:

```
curl HOST/users/X/resources/Y/streams/Z/data?token=KEY&format=csv
```

5.4.1 Exercise

- Use your web browser to view one of your streams and its data. Navigate to its API URL `HOST/users/X/resources/Y/streams/Z?token=KEY` to see the stream's JSON representation. Then append /data such as `HOST/users/X/resources/Y/streams/Z/data?token=KEY` to see its data contents.
- Change the amount of data points viewed: `HOST/users/X/resources/Y/streams/Z/data?token=KEY&limit=2`

5.5 Post to a resource

Data enters the system via SicsthSense receiving a POST or by it performing a Poll. Poll events can be scheduled by configuring the Resource. Simply specify the fields polling_url and polling_period, SicsthSense will then perform an HTTP GET on that url every period seconds.

```
$ curl -XPOST -H "Content-Type:  application/json"
-d '{"label":  "resourceLabel", "polling_url":"http://url.com/test.json",
"polling_period":100}' http://HOST/users/X/resources
```

5.6 New resources

A new resource can be added by POSTing a JSON representation of a resource to the URL: `http://HOST/users/X/resources`, whereas they can be deleted

by performing an HTTP DELETE request to their relevant URL (also works on contained streams and parsers): `http://HOST/users/X/resources/Y` It is possible to modify the representation stored on the server by PUTing a new JSON representation of that resource. Alternatively a new resource may be created by POSTing a JSON representation of the new resource.

Resources contain Parsers that make sense of data and output Streams of data points. The parsers can simply be a JSON field path, allowing a given JSON field to be parsed as a datapoint from the incoming data and then stored in SicsthSense.

5.7 POST to a Stream

Actual data payloads can be represented through any format, though the preferred one is JSON. Any type of JSON primitive value can be easily consumed by Resources, depending on how the parsers are configured. If Streams are being directly interacted with, the format needs to be slightly more strict. It needs to contain `value` and can optionally contain a `timestamp`. Rather than POSTing to the stream representation, the child `/data` should append, hence in the following form: `http://HOST/users/X/resources/Y/streams/Z/data`.

```
1 { "value": 2.5, "timestamp": 1384249393 }
```

5.8 Functions

Streams can be populated be processing functions of other streams. For example, a stream may be formed from the mean of other input streams. Whenever a data point is added to one of its predefined input streams, the most recent value is taken from each input and the mean of them all is taken.

The current predefined functions are:

- mean
- median
- min
- max

To register a stream that is populated by a function, simply POST a new stream representation into one of your existing SicsthSense resources (e.g. `POST http://HOST/users/X/resources/Y/streams/`) with the following additional attributes:

```
1 {"function": "mean", "antecedents": ["1", "2"]}
```

This will form a new stream under that resource that takes stream IDs 1 and 2 as input. Assuming you own streams 1 and 2, the new stream will then contain the mean of the two antecedent streams.

5.8.1 Exercise

- POST to one of your resources that has parsers defined (or become auto defined when you post).
- POST directly into one of your streams using the more specific value/-timestamp format.
- Create a new stream that is a function of some of your other streams.

5.9 Summary

This should have given you the basics of how to interact with the SicsthSense system through the web interface and the purely textual machine-to-machine API.

6 Practical uses

Rather than handcrafting JSON documents it is obviously much easier to use a library. We have created a simple Python library for interacting with the m2m engine. The relevant files can be found in the `SiscthSensePython` archive in the shared directory/flash drive[4]. The python module is just called `Engine.py` and functions as a wrapper around the HTTP interface.

The script `demoAutoParse.py` shows how to initialise the Engine and perform User registering, Resource creation and automatic parsing of JSON documents. Modify this file to point at the running SicsthSense instance and configure your userID and authentication key. Then the resources that you register and post to will appear on the running web interface for inspection.

6.1 Android Application

We have also created a basic Android app to POST data at a SicsthSense URL (see Figure 4). The app is called **PhoneSense** and is available on Google Play: `https://play.google.com/store/apps/details?id=se.sics.phonesense` The sourcecode for this app is also available in our Github:
`https://github.com/sics-iot/sicsthsense/tree/master/android`

The application will periodically read some of the on-board sensors from the Android phone and construct a JSON containing the values. When the toggle button is active the JSON will then be POSTed to the URL written in the text box. This doesn't involve any state or negotiation by the Android app. The posted JSON or possible communication errors will be displayed in the status box at the bottom of the screen. Feel free to checkout the code from the Github and modify the code to create more advanced interactions.

[4] https://drive.google.com/?tab=mo&authuser=0#folders/0B4GZtU3sa5OpeG5mcGoyMEFFUjQ

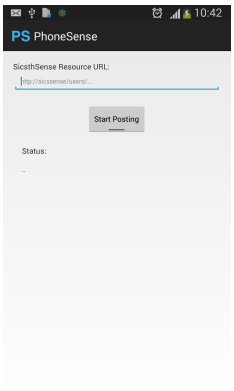

Figure 4: PhoneSense Android application screenshot.

6.1.1 Exercise

- Examine the scripts python that manage posting and scraping data from SicsthSense.
- Build your own application (in the language of your choice) to store and retrieve data in SicsthSense. Preferably using data from a stream populated from one of the Motes.
- Use acceleration or light data collected from your phone app to actuate tasks, possibly from the provided Yanzi devices.

7 Future Work

SicsthSense is being actively developed to offer useful features to IoT system developers and users of home automation technologies. Other commercial IoT data platforms, such as Xively.com, Open.sen.se and ThingSpeak.com, are appearing on the market and are becoming a promising area of development in the future of highly networked sensed data collection, collation, presentation and actuation.

Other research institutes and industrial partners use of the system is driving development of new features. The code is available on Github: `https://github.com/sics-iot/sicsthsense` and we welcome contributors, there are more features than presented here, such as CoAP[5] support and Websocket communication.

[5] http://tools.ietf.org/search/draft-ietf-core-coap-18