Minimizing electricity theft by Internet of Things

Y Rithvik, 15085076, [y.rithvik.eee15@itbhu.ac.in](mailto:y.rithvik.eee15@itbhu.ac.in)

Sai Sumanth Narkedamilli, 15084010, [nsai.sumanth.eee15@itbhu.ac.in](mailto:nsai.sumanth.eee15@itbhu.ac.in)

Yenuganti Sai Rithwik, 15085079, [ysai.rithwik.eee15@itbhu.ac.in](mailto:ysai.rithwik.eee15@itbhu.ac.in)

V Hemanth Sai, 15085073, [vhemanth.sai.eee15@itbhu.ac.in](mailto:vhemanth.sai.eee15@itbhu.ac.in)

# Problem: Electricity Theft

Theft of electricity is the criminal practice of stealing electrical power. It is a crime and is punishable by fines and/or incarceration.

According to the annual Emerging Markets Smart Grid: Outlook 2015 study by the Northeast Group, LLC, the world loses US$89.3 billion annually to electricity theft. The highest losses were in India ($16.2 billion), followed by Brazil ($10.5 billion) and Russia ($5.1 billion). President of Northeast Group Ben Gardner stated, "India loses more money to theft than any other country in the world. The state of Maharashtra—which includes Mumbai—alone loses $2.8 billion per year, more than all but eight countries in the world. Nationally, total transmission and distribution losses approach 23% and some states' losses exceed 50%."

## Types of electricity theft

There are various types of electrical power theft, including Tapping a line or bypassing the energy meter. According to a study, 80% of worldwide theft occurs in private dwellings and 20% on commercial and industrial premises. The various types of electrical power theft include:

### Direct hooking from line

Hooking is the most used method. 80% of global power theft is by direct tapping from the line. The consumer taps into a power line from a point ahead of the energy meter. This energy consumption is unmeasured and procured with or without switches.

### Bypassing the energy meter

In this method, the input terminal and output terminal of the energy meter is short-circuited, preventing the energy from registration in the energy meter.

### Injecting foreign element into the energy meter

Meters are manipulated via a remote by installing a circuit inside the meter so that the meter can be slowed down at any time. This kind of modification can evade external inspection attempts because the meter is always correct unless the remote is turned on.

### Physical obstruction

This type of tampering is done to electromechanical meters with a rotating element. Foreign material is placed inside the meter to obstruct the free movement of the disc. A slower rotating disk signals less energy consumption.

### ESD attack on electronic meter

This type of tampering is done on electronic meter to make it either latent damage or permanent damage. Detection can be done correctly in high end meters only.

# Existing Solutions

## Manual Inspections

Energy meters are inspected for any signs of tampering by billing agents and inspectors during surprise inspections. These are not effective against line hooking cases and are often subject to non-cooperation by locals.

## Complaint Portals

These portals are hosted online where consumers can report any suspicious activities by their neighbours regarding electricity theft. This method is not effective at least in India.

# Proposed Solution

We propose replacing current energy meters with smart energy meters. Smart energy meters are always connected to a central server and any tampering can be detected easily. These meters also send real time power readings. Except for a few losses, it is simple matter to detect energy theft when there is more electricity delivered to the substation than the sum of all the smart meters.

Identification still must be done manually once energy theft has been detected but can be deeply localized geographically.

# Prototype Implementation

## Components used

* Energy meter
* Raspberry Pi 3 (Model B)
* Opto-isolator 4N35
* Bulb Load
* Connecting wires

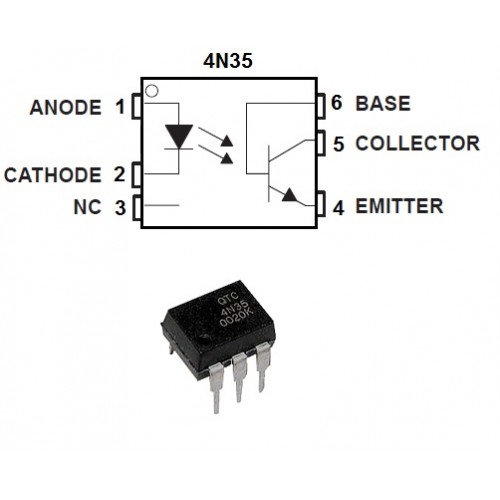
 

Figure 1: Conventional Energy Meter Figure 2: 4N35 Opto-coupler

## Image result for Raspberry pi 3 model b

Figure 3: Raspberry Pi 3 Model B

## Energy meter and Raspberry Pi interfacing

A prototype of Smart energy meter was made by interfacing a Raspberry Pi to a conventional Energy meter. Conventional Energy meters have an LED which blinks a constant number of times for every unit of power consumed. We captured the pulses to this LED through an opto-isolator to the GPIO pins of Raspberry Pi. The pulse is obtained across pins 1 and 2 of the opto-isolator which makes pins 4 and 5 short circuited. Since pin 4 is connected to GND, Raspberry Pi GPIO pin connected to pin 5 of opto-isolator picks up 0V as input. This is detected as a pulse in our program. Whenever the pulse is detected the python program increments the energy reading accordingly. Then it calls a function to send data to the server.

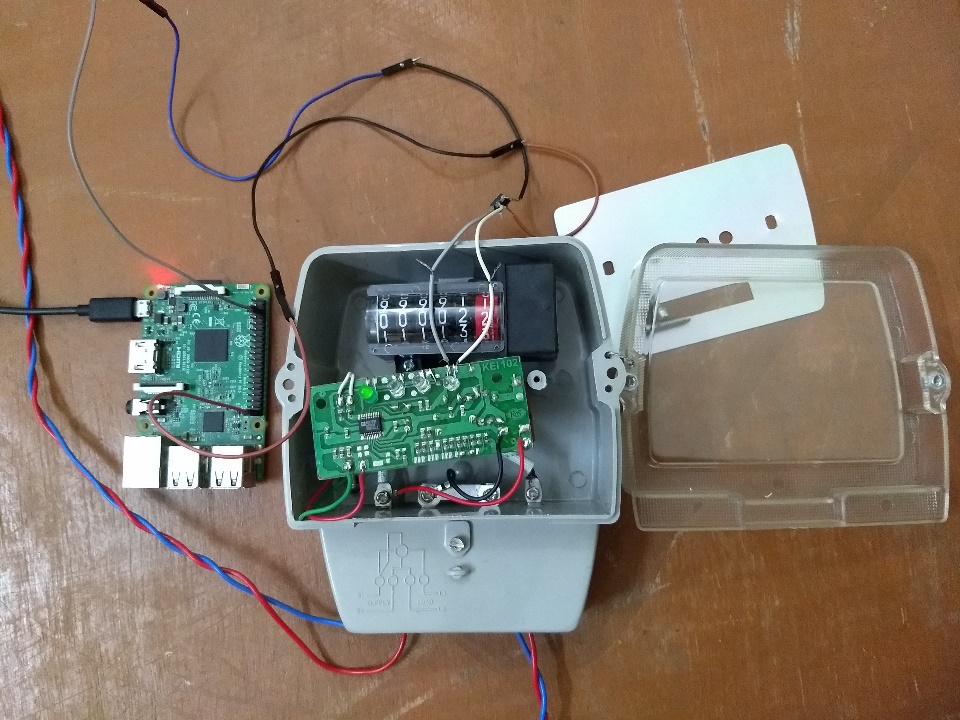


Figure 4: Energy Meter interfaced to Raspberry Pi

### Raspberry Pi Code

#Desktop\energy\_meter.py

**import** RPi**.**GPIO **as** GPIO

**import** time

**import** requests

meter\_reading **=** 0

debug **=** **True**

**def** senddata**():**

url **=** "http://192.168.137.102/readings/1/update/"

data **=** **((**"energy\_reading"**,** meter\_reading**), (**"username"**,** "hemanthsai"**), (**"password"**,**"ilovepython"**))**

**try:**

r **=** requests**.**post**(**url**,** data**=**data**)**

**if** debug**:** **print(**r**.**text**)**

**except:**

**if** debug**:** **print(**"Error occurred"**)**

GPIO**.**setmode**(**GPIO**.**BOARD**)**

GPIO**.**setup**(**40**,** GPIO**.**IN**)**

pulses **=** 0

prevprev**,** prev**,** curr **=** 1**,** 1**,** 1

**while** **True:**

prevprev**,** prev**,** curr **=** prev**,** curr**,** GPIO**.**input**(**40**)**

**if** **not** **(**prevprev **or** prev **or** curr**):**

meter\_reading **+=** 1**/**3200

senddata**()**

**while** **not** curr**:**

curr **=** GPIO**.**input**(**40**)**

time**.**sleep**(**0.01**)**

time**.**sleep**(**0.01**)**

GPIO**.**cleanup**()**

## Setting up Web Server

A development web server was hosted on a laptop built using Python Django framework. It had a Django model “meter” to represent a smart energy meter. Each meter has a unique meter id, date\_created, last\_updated datetime fields and meter\_reading, power\_reading decimal fields. The date\_created field is initialized the datetime when the meter object is created. The power\_reading, meter\_reading and last\_updated fields get updated whenever a http POST request is received by the server for a particular meter. All these fields are stored efficiently in a database which was the Django default SQLite3 database in our prototype.

#server\readings\models.py

**from** django**.**db **import** models

**from** django**.**utils **import** timezone

**class** **Meter(**models**.**Model**):**

meter\_id **=** models**.**AutoField**(**null**=False,** blank**=False,** unique**=True,** primary\_key**=True)**

date\_created **=** models**.**DateTimeField**(**default**=**timezone**.**now**,** editable**=False)**

last\_updated **=** models**.**DateTimeField**(**default**=**timezone**.**now**)**

meter\_reading **=** models**.**DecimalField**(**default**=**0**,** max\_digits**=**12**,** decimal\_places**=**7**)**

power\_reading **=** models**.**DecimalField**(**default**=**0**,** max\_digits**=**12**,** decimal\_places**=**7**)**

**def** \_\_str\_\_**(**self**):**

**return** 'Meter ID: {}'**.**format**(**self**.**meter\_id**)**

The urls.py file is used to elegantly map between URL patterns and our python functions which are defined in the views.py file.

#server\readings\urls.py

**from** django**.**conf**.**urls **import** url

**from** **.** **import** views

app\_name **=** "readings"

urlpatterns **=** **[**

url**(**r'^$'**,** views**.**index**,** name**=**'index'**),**

url**(**r'^powers/$'**,** views**.**powers**,** name**=**'powers'**),**

url**(**r'^(?P<meter\_id>[0-9]+)/detail/$'**,** views**.**detail**,** name**=**'detail'**),**

url**(**r'^(?P<meter\_id>[0-9]+)/update/$'**,** views**.**update**,** name**=**'update'**)**

**]**

#server\readings\views.py

**from** django**.**shortcuts **import** get\_object\_or\_404**,** render

**from** django**.**http **import** HttpResponse

**from** **.**models **import** Meter

**from** django**.**views**.**decorators**.**csrf **import** csrf\_exempt

**from** django**.**utils **import** timezone

**from** django**.**contrib**.**auth **import** authenticate

**def** index**(**request**):**

meter\_list **=** Meter**.**objects**.**order\_by**(**'meter\_id'**)**

context **=** **{** 'meter\_list'**:**meter\_list **}**

**return** render**(**request**,** 'readings/index.html'**,** context**)**

**def** detail**(**request**,** meter\_id**):**

meter **=** get\_object\_or\_404**(**Meter**,** pk **=** meter\_id**)**

**return** render**(**request**,** 'readings/detail.html'**,** **{**'meter'**:**meter**})**

**def** powers**(**request**):**

meter\_list **=** Meter**.**objects**.**order\_by**(**'meter\_id'**)**

context **=** **{** 'meter\_list'**:**meter\_list **}**

**return** render**(**request**,** 'readings/powers.html'**,** context**)**

@csrf\_exempt

**def** update**(**request**,** meter\_id**):**

**try:**

**assert** request**.**method**==**'POST'

user **=** authenticate**(**username**=**request**.**POST**[**'username'**],**password**=**request**.**POST**[**'password'**])**

**if** user **is** **not** **None:**

meter **=** get\_object\_or\_404**(**Meter**,** pk**=**meter\_id**)**

energy\_reading **=** request**.**POST**[**'energy\_reading'**]**

nowtime **=** timezone**.**now**()**

meter**.**power\_reading **=** **(**energy\_reading**-**meter**.**meter\_reading**)\***3600000**/(**nowtime**-**meter**.**last\_updated**)**

meter**.**meter\_reading **=** energy\_reading

meter**.**last\_updated **=** nowtime

meter**.**save**()**

response**=**"Success"

**else:**

response**=**"Authentication Failed"

**except:**

response**=**"Post Failed"

**return** HttpResponse**(**response**,** content\_type**=**"text"**)**

The views.py file contains all our views and functions to handle requests and render webpages. The templates used for rendering these webpages are in a separate folder and written in HTML and CSS.

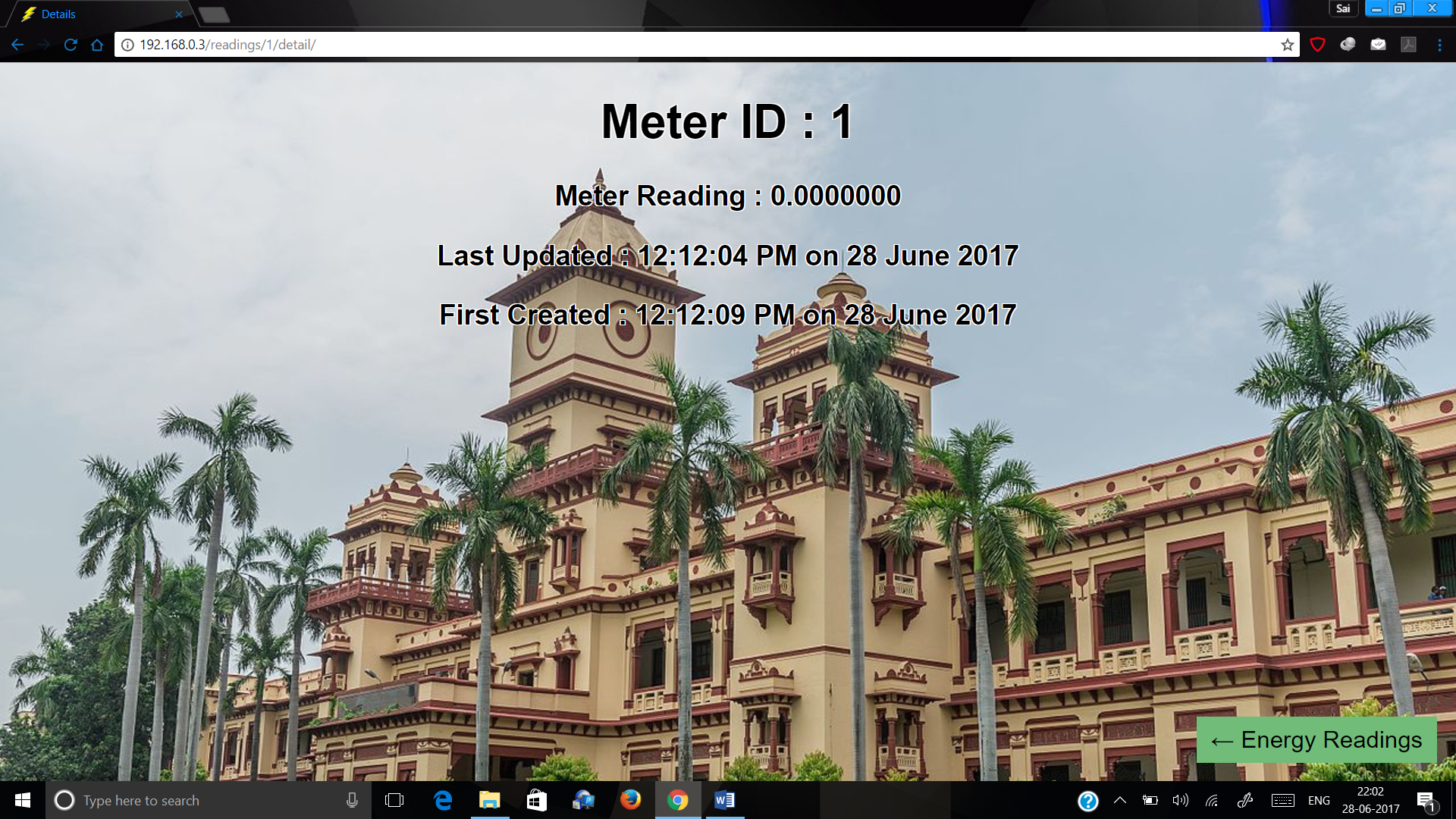


Figure 5: Energy meter detail webpage

## Communication between Raspberry Pi and Web Server

For this prototype Raspberry Pi was connected to the development web server via Wi-Fi. Energy meter readings are sent from Raspberry Pi to the Web Server at regular intervals of one second. This communication is done by a http POST request to the Web Server host. The request contains the energy meter id, meter reading and username and password for authentication.

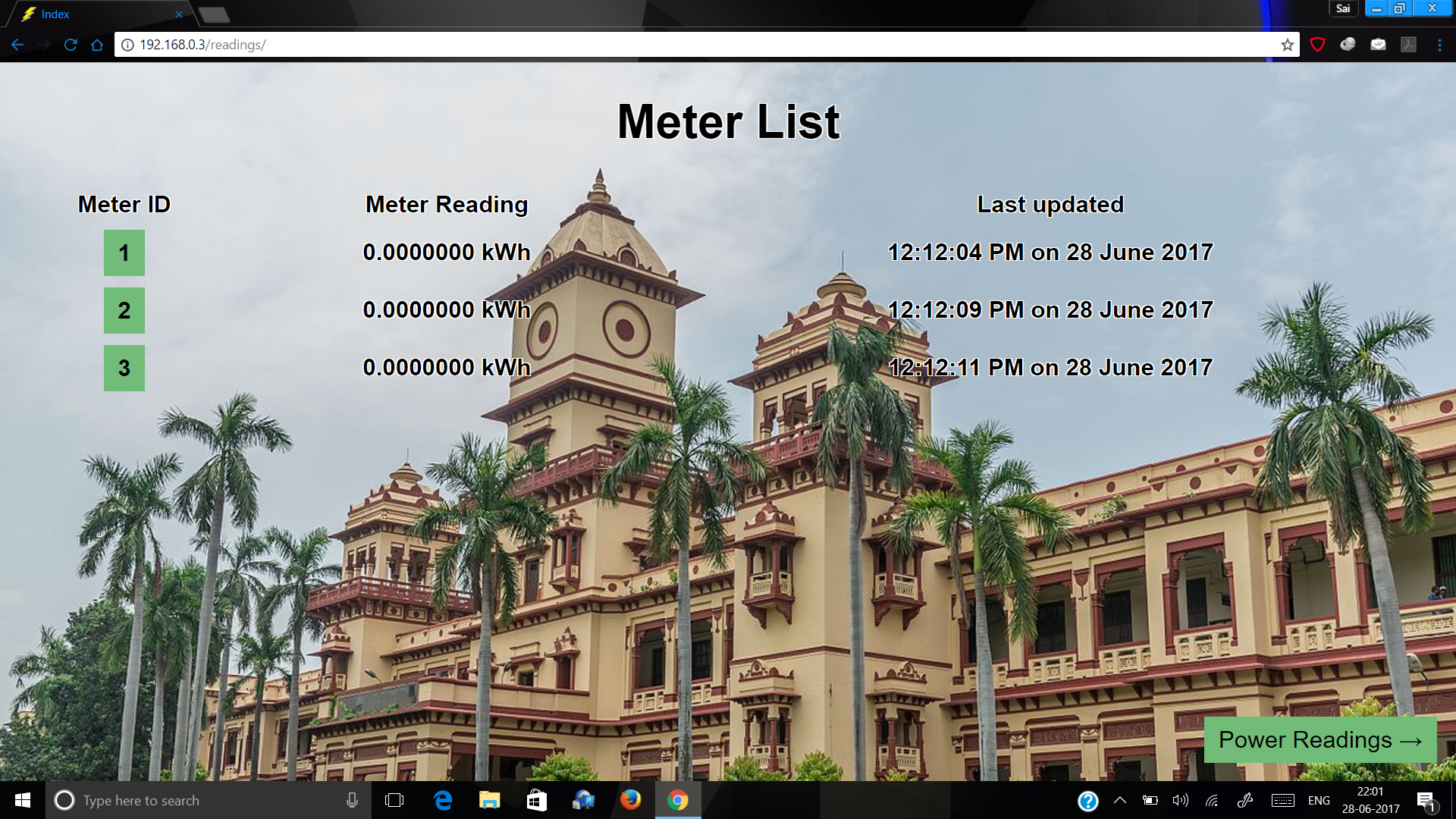


Figure 6: Meter List webpage

Using the new and old energy readings of the meter and also the exact time of measurement, the average power over the time interval is calculated. This reading is then displayed for that meter on another webpage.

# 

Figure 7: Power Readings webpage

## Theft Detection

The difference between the power reading of a substation/distribution transformer and the sum of power readings of all the energy meters that belong to that line will be the criterion for theft detection. Under normal conditions, this difference is because of technical losses over the line which is always a small fraction of the total power being delivered. However, this difference will be quite larger if energy is being stolen. So, we set a decision threshold on this difference to detect energy theft. If the difference is above the threshold then we assume that there is an energy theft happening somewhere beyond that substation/transformer.

# Issues with prototype

## Issues with Raspberry Pi

During the absence of a pulse from energy meter, Raspberry Pi GPIO pins pick up garbage values which can be misinterpreted as a pulse. We avoid this by checking for a continuous pulse in order of 50ms. However, there may still be some false positives in our prototype.

## Low range network

The network used in the prototype was a Wi-Fi Hotspot created on a laptop which has a range of about 10 m. This range is nowhere near the required range for the full-scale model which should be about 1 km.

# Full Scale Expectations

## Smart Energy Meter

Instead of using a Raspberry Pi along with the energy meter, a smart energy meter embedded with a communication system can be used. These meters are all on the same network and can establish an encrypted connection with the central server to send their energy readings at regular intervals. These meters should also be tamper proof and isolated from external electromagnetic waves. These will be far more economical than the prototype we proposed.

## Network Expectations

Since the energy meter and the central server will be very far apart geographically, short range communication methods like Wi-Fi will not be feasible. The safest method would be to set up a separate wireless network having long range transmitting antennas/towers specifically for this purpose. In this way, the meters will send data to a closely located tower which in turn will send it to the central server.