Haaga-Helia
ammattikorkeakoulu Oy

Virtuaaliluonto - Nuotiovahti

IoT-sensorilaite anonyymin datan keräämiseen nuotiopaikoilla

Markus Pyhäranta
Mika Selvinen
Oliver Sirén
Vesa Valli

12.12.2017

Sisällys

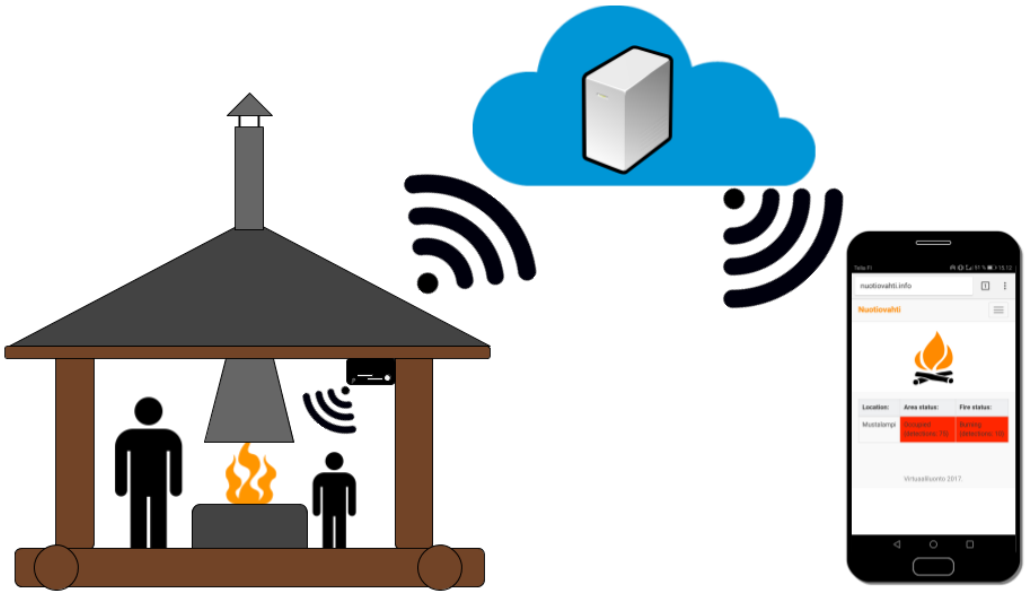
1 Johdanto

Virtuaaliluonto – Nuotiovahti projektissa suunniteltiin, toteutettiin ja testattiin yksi toiminta-valmis Internet of Things -tyyppisen sensorilaitteen prototyyppi. Tavoitteena oli kerätä laavu- ja nuotiopaikoilta dataa, joka lähetettään verkon yli loppukäyttäjän nähtäväksi netti-sivulle. Samalla mahdollistettiin statistiikkatiedon kerääminen kohteiden käytöstä, esimer-kiksi kansallispuisto- tai leirialueiden ylläpitäjien hyödynnettäväksi tulevaisuudessa.

Projektissa syntyneen sensorilaitteen pohjana toimii Raspberry Pi Model 3 B -tietokone. Laite kerää sensoreiden avulla tietoa ympäristöstään ja lähettää tiedot mobiiliverkon ylitse MQTT-protokollan avulla palvelimen tietokantaan, josta tiedot ovat esitettävissä webkäyt-töliittymässä. Kerättävä data sisältää tietoa liikkeestä ja valosta.

Kun sisätiloissa testattu IoT-laite oli todettu toimivaksi, siirryttiin ulkoilmatestaukseen. Tätä varten tuli selvittää laitteelle sopivat virransyöttö- ja datayhteysratkaisut. Prototyypin tulisi olla ulkoilmakelpoinen ja säänkestävä.

Laitteen keräämät tiedot päätyvät lopulta verkkosivulle, josta loppukäyttäjät voivat selvit-tää, onko esimerkiksi Nuuksion kansallispuistossa lähin nuotiopaikka jo matkailijoiden käytössä vai ei. Sen lisäksi sivulle eritellään palaako kyseisellä nuotiopaikalla tuli. Käyttä-jille esitetään siis tietoa sekä ihmisistä että nuotion tilasta. Laitteen infrapunasensoreilla ei pystytä identifioimaan henkilöitä, eli se kunnioittaa kävijöiden yksityisyyttä.



Kuva 1. Prototyypin konseptin perusidea.

1.1 Projektin GitHub-arkisto

Kaikki projektin aikana syntynyt lähdekoodi löytyy projektiryhmän virallisesta GitHub-arkistosta: https://github.com/wikkii/raspluonto

Suurin osa projektissa syntyneestä koodista on kirjoitettu Python-ohjelmointikielellä. Muita käytettyjä kieliä ovat HTML, CSS, JavaScript, Shell ja SQL.

Koodit, joissa on hyödynnetty kolmannen osapuolen lähdekoodia, sisältävät alkuperäiset lähteet koodin alun kommenteissa.

1.2 Projektin blogi

Sekä teknistä että vähemmän teknistä materiaalia projektin etenemisestä löytyy WordPress-blogistamme: https://raspluonto.wordpress.com/

1.3 Projektisivut

Prototyyppiämme varten luodut sivut: http://nuotiovahti.info/
(Ei välttämättä enää pystyssä projektin päättymisen jälkeen.)

2 Sensorilaite

Projektimme prototyypin laitealusta muodostui Raspberry Pi 3 Model B -tietokoneesta seuraavilla järjestelmätiedoilla:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- 40-pin extended GPIO
- 4 USB 2 ports
- 4 Pole stereo output and composite video port
- Full size HDMI
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- Micro SD port for loading your operating system and storing data
- Upgraded switched Micro USB power source up to 2.5A

2.1 Käyttöjärjestelmän asennus

Käyttöjärjestelmäksi asennettiin Raspbian Stretch Lite -käyttöjärjestelmä. Kyseisessä versiossa ei ollut graafista käyttöliittymää ollenkaan. Näin saatiin prototyypin virrankulutusta laskettua. Käyttöjärjestelmän levykuva ladattiin Raspberry Pi:n virallisilta verkkosivuilta:

Levykuva kirjoitettiin sitten SD-kortille käyttämällä Etcheriä.

2.2 Staattiset IP-osoitteet

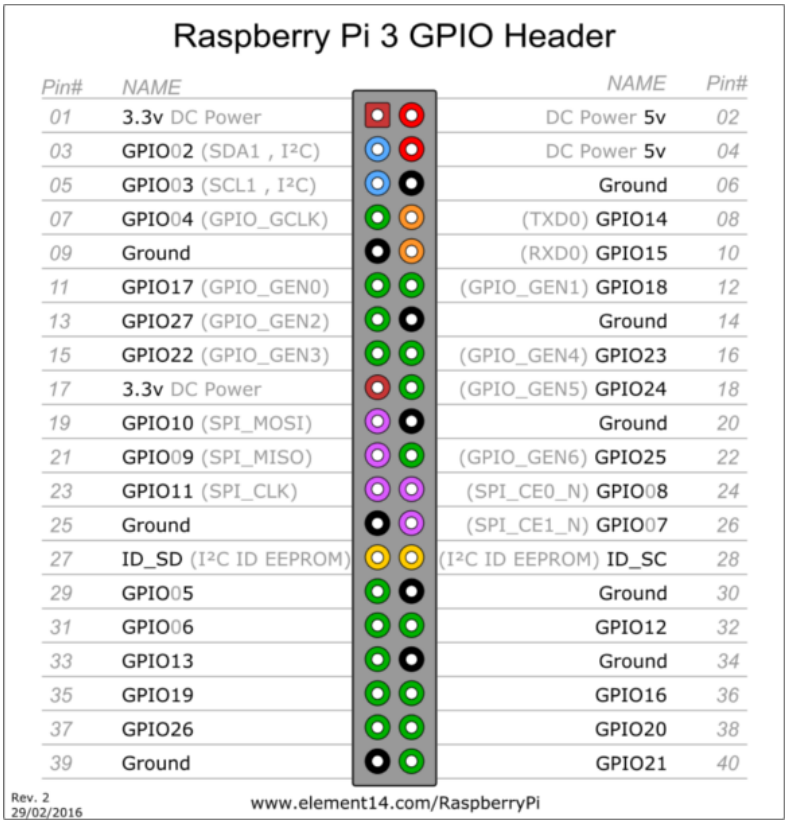
Käyttöjärjestelmän asennuksen jälkeen Raspberry Pi:lle annettiin staattinen IPv4-osoite Haaga-Helian laboratorioverkossa. Lisäsimme myös langattomalle wlan0-interfacelle staattisen IP-osoitteen mobiilidatayhteyden WiFi-verkon kautta jakavaa modeemia varten. Tämä tehtiin muokkaamalla **/etc/dhcpcd.conf** tiedostoa. Sieltä poistettiin kommentit ”static IP configuration” -otsikon alapuolelta **interface eth0** kohdalta ja lisättiin erikseen osoitetiedot **interface wlan0**:lle.

```
# Koulun verkkoa varten
interface eth0
static ip_address=172.26.177.25/16
static routers=172.28.1.254
static domain_name_servers=172.28.170.201 172.28.170.202

# Mobiiliverkkoyhteyttä varten
interface wlan0
static ip_address=192.168.8.102/24
static routers=192.168.8.1
static domain_name_servers=192.168.8.1
```

Muokkauksen jälkeen Raspberry Pi käynnistettiin uudelleen.

2.3 Sensoreiden asennus ja testaus

Kaikki projektissa käytetyt sensorit kiinnitettiin Raspberry Pi-tietokoneen GPIO-pinneihin (general purpose input/output). GPIO-pinnejen hyödyntämiseen käytetään monesti RPi.GPIO-moduulikirjastoa, mutta projektiryhmämme hyödynsi kurssin toisen opettajan suosituksesta hänen itse tekemäänsä Make: Sensors -kirjastoa, joka löytyi täältä: http://makesensors.botbook.com/code.html



Kuva 2. Raspberry Pi 3:n GPIO layout. (Element14 2017)

GPIO-pinnejä käytetään tyypillisesti **/sys/class/gpio/** kansion alaisuudessa, joka on root-käyttäjän ja root-ryhmän omistuksessa. Halusimme käyttää niitä ilman root-oikeuksia, mikä vaati kyseiseen kansioon pääsyn sallimisen uusilla udev säännöillä. Sääntöinä käytimme **88-gpio-without-root.rules**, joka tuli aiemmin mainitun kirjaston mukana.

Säännöt siirrettiin Raspberry Pi:lle SCP-yhteydellä, jonka jälkeen ne siirrettiin laitteen **/etc/udev/rules.d** hakemistosijaintiin. Uudet säännöt aktivoitiin käynnistämällä udev-demoni uudelleen ja syöttämällä sitten komento **sudo udevadm trigger --subsystem-match=gpio**.

Muutoksen jälkeen /sys/class/gpio/ kansion oikeudet olivat seuraavat:

```
$ ls -lR /sys/class/gpio/
/sys/class/gpio/:
total 0
-rwxrwx--- 1 root dialout 4096 Sep 15 08:27 export
lrwxrwxrwx 1 root dialout    0 Sep 15 08:27 gpiochip0 ->
 ../../devices/platform/soc/3f200000.gpio/gpio/gpiochip0
lrwxrwxrwx 1 root dialout    0 Sep 15 08:27 gpiochip100 ->
 ../../devices/gpiochip2/gpio/gpiochip100
lrwxrwxrwx 1 root dialout    0 Sep 15 08:27 gpiochip128 ->
 ../../devices/gpiochip1/gpio/gpiochip128
-rwxrwx--- 1 root dialout 4096 Sep 15 08:27 unexport
```

Kuten tuloksesta näkyy, dialout-ryhmälle annettiin kirjoitusoikeudet kansioihin.

Kummassakin sensorissa on kolme pinniä GPIO-liitäntää varten: virta, maadoitus ja digitaalinen ulostulo (digital out). Molemmat käyttämämme sensorit toimivat digitaalisella signaalilla. Analoginen signaali tarkentaisi havainnointia, mutta Raspberry Pi:n GPIO-väylä on täysin digitaalinen eikä tue analogisia signaaleja. Olisimme näin tarvinneet Pi:n ja sensorien välille jonkinlaisen signaalin analogisesta digitaaliseksi muuttavan A/D-muuntimen, esimerkiksi Arduinon.

2.3.1 Sensoriskriptit ja MQTT-protokolla

Molemmat sensoreistamme toimivat Python-skriptien avulla. Käytimme skriptiemme pohjana Make: Sensors -kirjan (ISBN 978-1449368104) esimerkkikoodeja, jotka latasimme samasta osoitteesta käyttämämme GPIO-kirjaston kanssa.

Muokkasimme liekki- ja PIR-sensorien esimerkkiskriptejä tarpeisiimme, esimerkiksi vaihtaen sensorin käyttämän GPIO-datapinnin. Käytimme skripteissä esimerkkikoodien mukana tullutta **botbook_gpio**-kirjastoa raportin kohdan 2.3 mukaisin muokkauksin. Skriptien avulla sensorien lähettämä digitaalinen signaali luetaan kahden sekunnin välein ja tulkitaan joko arvona 1 tai 0.

Myöhemmin muokkasimme skriptejä lähettämään datan suoraan palvelimellemme ilman välikäsiä **MQTT**-protokollaa käyttäen. MQTT toimii publish-subscribe periaatteella. Projektimme tapauksessa Raspberry Pi on ”publisher”, jonka viestit vastaanotetaan palvelimelta tilaamalla se ”topic”, johon sensorilaitteemme julkaisee viestejä. Viestit eivät mene suoraan julkaisijalta tilaajalle, vaan ne välitetään MQTT-brokerin kautta. Meidän projektissamme MQTT-broker ja -subscriber olivat samalla palvelimella, kuten raportin kohdassa 3.2.3 on esitetty.

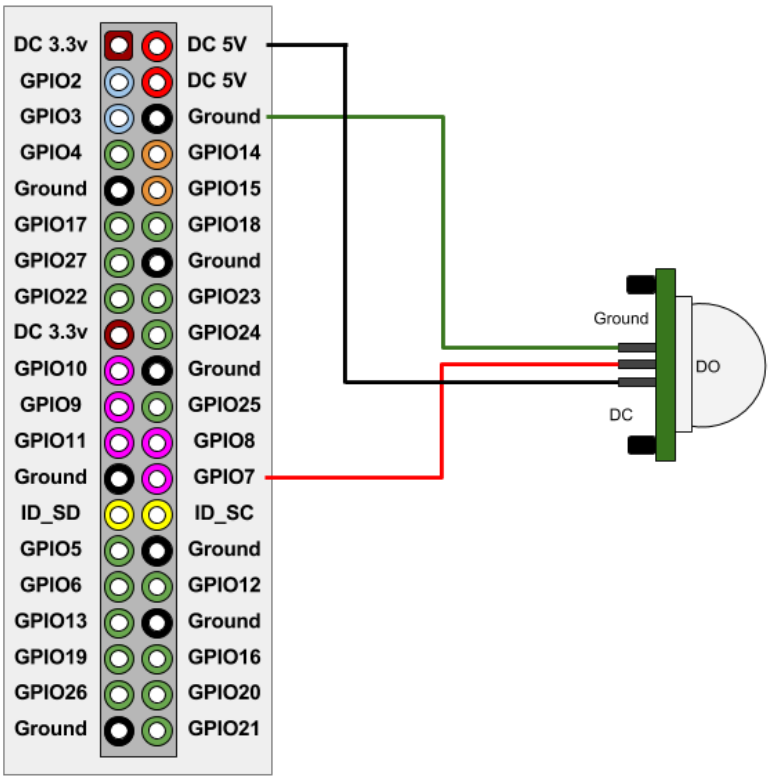
Kuva 3. MQTT-protokolla yksinkertaistettu toimintaperiaate.

Skripteihin lisättiin tarvittavat tiedot, kuten MQTT-brokerin IP-osoite ja viestin topic. Lisäksi tarkensimme sensorien signaaliarvoja vastaavat viestit, esimerkiksi liekkisensorin havaitessa tyhjää MQTT-viesti olisi ”No flame detected.” Skriptejä viimeistellessämme muutimme koodia kuitenkin niin, että MQTT-viesti lähetetään vain, mikäli sensori havaitsee jotain oletusarvosta poikkeavaa. Näin tietokantamme ei täyty ”kaikki hyvin”-viesteistä, ja voimme käyttää tietueiden aika-arvoja havaintojen tiheyden laskemiseen.

Raspberry Pi:lle asennettiin Python PIP Debianin paketinhallinnan kautta, sekä Pythonin MQTT-toiminnallisuuden mahdollistava Eclipse Paho-MQTT Python Client -kirjasto PIP:illä, pakettinimeltään ”**paho-mqtt**”. Tämän kirjaston avulla Pi:n sensorien Python-skriptit voivat suoraan lähettää MQTT-viestejä sensorien outputin perusteella

Kunkin sensorin Python-koodin sisältö on kuvailtu tarkemmin raportin kohdissa 2.3.2 ja 2.3.3.

2.3.2 Liiketunnistin (PIR)



Kuva 4. PIR-sensorin kytkentäkaavio.

Ensimmäinen prototyyppimme kahdesta sensorista oli niin kutsuttu PIR-sensori (passive infrared sensor, mallinumero **HC-SR501**), joka havaitsee esineiden säteilemää lämpösäteilyä näkökentässään. Kun sensori on ensin totutettu tietyille tausta-arvoille, ja sen vastaanottaman säteilyn arvoissa tapahtuu poikkeamia oletusarvoista, lähettää sensori signaalin. PIR-sensorin näkökenttä on noin 110 asteen kartio, ja suurimmaksi havaintoetäisyydeksi osoittautui testeissä noin seitsemän metriä. PIR-sensori toimii 4.5V-20V virralla. Sensori lähettää tasaisen signaalin, koska se on digitaalinen; havaintojen määrällä ei ole merkitystä.

PIR-sensorissa on kaksi säädintä: herkkyys, joka säätää havaintoetäisyyttä (noin 3-7 metriä) ja aikaviive, joka vaikuttaa sensorin toimintaan (0,3-300 sekuntia). Lisäksi sensorin piirilevyllä on jumpperi sensorin toimintaperiaatteen valintaa varten:

- Non-repeatable: Kun sensori huomaa liikettä, se lähettää signaalia aikaviiveen pituisen ajan, jonka sensorin arvo muutetaan automaattisesti nollaan. Sensori ei havaitse liikettä uudelleen aikaviiveen pituisen ajan.
- Repeatable: Sensori voi jatkaa yhtä havaintosignaalia niin pitkään, kuin se havaitsee liikettä. Kun sensori lakkaa havaitsemasta liikettä, asettaa se signaalin nollaan aikaviiveen pituiseksi ajaksi.

Vaikka PIR-sensori on lämpösäteilyn tunnistin, eikä suoranaisesti tarkoitettu liikkeen tunnistamiseen, sopi se tarkoituksiimme. PIR-sensori ei havainnut ja tulkinnut liekkien lämpösäteilyä liikkeenä kuin vasta ison nuotion laukaistessa molemmat sensorit. Sinänsä se, että liiketunnistin havaitsee ”virheellisesti” liekin ei ole ongelma, sillä jos sensorilaite havaitsee PIR-sensorille tarpeeksi suuren liekin, on paikalla varmasti joku. Ultraäänipohjainen sensori olisi mahdollisesti toiminut paremmin liiketunnistimena, sekä täysin erillään liekkisensorista.

```
import time
import botbook_gpio as gpio
import paho.mqtt.client as mqtt                                    1)
#-------------------------------------------------------------------
------------------------
learningPeriod = 30

def main():

        broker_address="139.59.140.158"
        client = mqtt.Client("P1") #create new instance    2)
        client.connect(broker_address) #connect to broker
#-------------------------------------------------------------------
------------------------

        pirPin = 7                                                 3)
        gpio.mode(pirPin,"in")

#Learning period
#       client.publish("nuotiovahti","learning... " + str(learn-
ingPeriod) + " seconds")

#-------------------------------------------------------------------
------------------------
        time.sleep(learningPeriod) # <1>
        while (True):
                movement = gpio.read(pirPin) # <2>
                if(movement == gpio.HIGH):                         4)
                        client.publish("nuotiovahti/pir","1")
                else:

                  client.publish("nuotiovahti/pir", "No movement
detected")
#-------------------------------------------------------------------
------------------------

                time.sleep(2)                                      5)


if __name__ == "__main__":
        main()
```

Väliviivat ja vihreät numerot eivät kuulu koodiin, vaan ne on lisätty vain tähän raporttiin selityksiä varten. Koodi löytyy GitHubista nimellä ”**mqtt_pir_sensor.py**”.

1) Koodin alussa tuomme tarvitsemamme kirjastot. Jotta PIR-sensori tarvitsee aikaa oppia ympäristöään, ennen kuin se alkaa havainnoimaan liikettä datana. Asetimme oppimisajan 30 sekunttiin.

2) main()-funktion sisällä yhdistämme ensimmäiseksi MQTT-brokeriin, joka on siis se palvelimemme. Koska lähetämme kahdelta eri sensorilta, annoimme molemmille oman instanssin, joka oli PIR-sensorilla ”P1”. Näin broker osaa tunnistaa ne toisistaan.

3) Datapinniksi on määritelty 7, joka ei tarkoita fyysistä pinniä numero 7, vaan GPIO 7. Testausvaiheessa laitoimme koodin printtaamaan sen oppimisajan, jotta tiesimme sensorin oikeasti tekevän jotain. Se on sittemmin kommentoitu loppuratkaisua varten pois koodista.

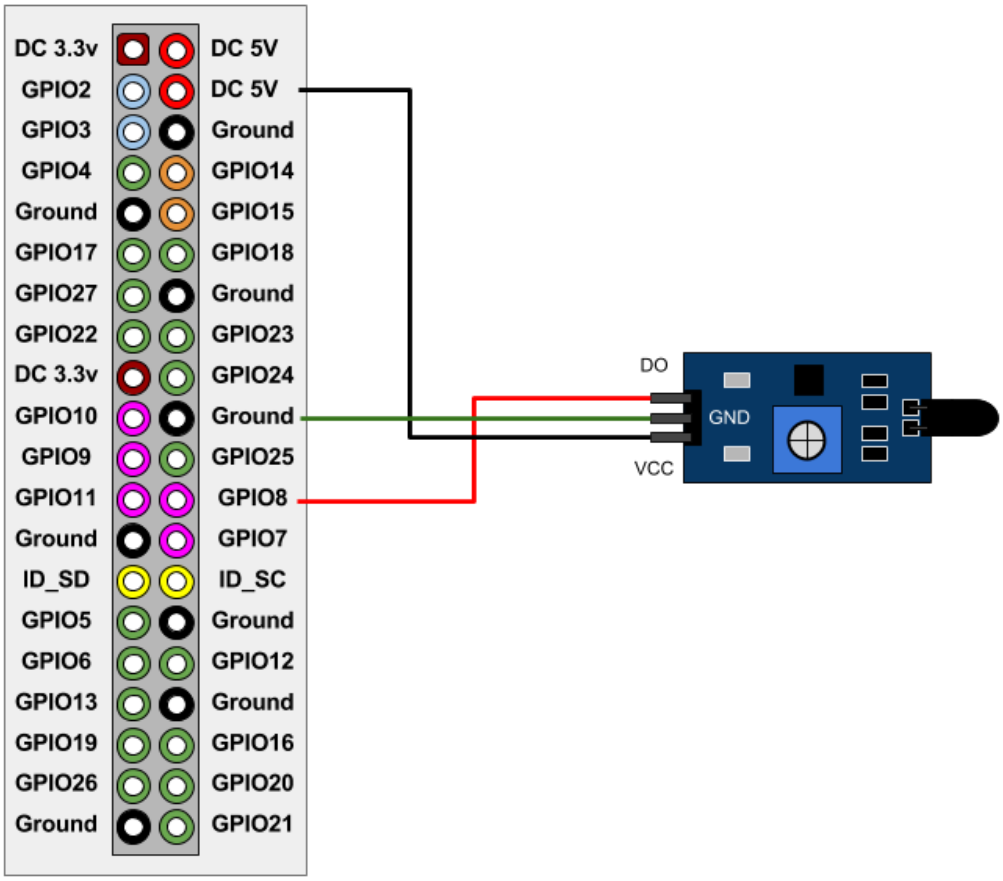
4) Kun sensori havaitsee arvoksi HIGH, eli liikettä havaittu, julkaisemme Paho-MQTT:llä ”nuotiovahti/pir/-topiciin viestin, jonka sisältö on ”1”.

   Kun sensori havaitsee arvoksi LOW, eli liikettä ei ole havaittu, julkaisemme Paho-MQTT:llä ”nuotiovahti/pir/-topiciin viestin, jonka sisältö on ”No movement detected”.

   Ainoastaan HIGH:n lähettämät arvot oikeasti talletetaan tietokantaan. Kun liikettä ei havaita, sitä ei tallenneta, mutta se lähetetään testaustarkoitusta varten.

5) time.sleep(2) määrittelee sensorin lähettämän datan tiheyden. Lähetämme dataa siis kahden sekunnin välein.

2.3.3 Liekkisensori



Kuva 5. Liekkisensorin kytkentäkaavio.

Toinen prototyyppimme sensoreista oli varsinainen infrapunasäteilyä havainnoiva liekki-sensori. Sensori tunnistaa tiettyjä infrapunasäteilyn aallonpituuksia, noin välillä 760-1100 mm, ja sen näkökenttä on noin 30 asteen kartio. Havaintoetäisyys riippuu eniten liekin koosta; tuikkukynttilä nähdään noin metrin päästä, ja suurempi nuotio noin parin metrin etäisyydeltä. Koska sensori on digitaalinen, lähettää se tasaisen signaalin millä tahansa havainnolla.

Liekkisensorissamme on kaksi LED-valoa, joista ensimmäinen on sensorin virtavalo, ja toinen ilmaisee sensorin havainneen liekin. Lisäksi sensorissa on herkkyyden säädin, joka maksimissaan tekee sensorista toimimattoman. Herkimmillään sensori on käyttökelvoton, sillä se väittää näkevänsä liekin vaikkapa pimeässä huoneessa. Taustasäteily luultavasti sotkee liekkisensorin tällöin.

Kuva 6. Liekkisensorin testausta. Vasemmanpuoleinen valo syttyi, kun tulta oli havaittu.

```
import time
import botbook_gpio as gpio
import paho.mqtt.client as mqtt
                                                        1)
#-----------------------------------------------------------------

def main():

        broker_address="139.59.140.158"
        client = mqtt.Client("P2")
        client.connect(broker_address)
        topic = "nuotiovahti/flame"
        flamePin = 8
        gpio.mode(flamePin, "in")
                                                        2)
#-----------------------------------------------------------------

        while(True):
                flame = gpio.read(flamePin)

                if(flame == gpio.LOW):
                        client.publish(topic,"2") 3)

#                if
                 else:
#                        client.publish(topic,"NO flame detected")
#-----------------------------------------------------------------
                time.sleep(2)                           4)

if __name__ == "__main__":
        main()
```

Liekkisensorin koodi löytyy GitHubista nimellä ”**mqtt_flame_sensor.py**”.

1) Koodin alussa tuomme tarvitsemamme kirjastot.

2) main()-funktion sisällä yhdistämme taas ensimmäiseksi MQTT-brokeriin. Liekkisensorille annettiin ID ”P2” ja topic, johon lähetetään, on ”nuotiovahti/flame”. Datapinniksi on määritelty 8, joka siis tarkoittaa GPIO 8, eikä sitä fyysistä.

3) Kun sensori havaitsee arvoksi LOW, eli tulta havaittu, julkaisemme Paho-MQTT:llä viestin, jonka sisältö on ”2”. Liekkisensori toimii jostain syystä eri tavalla kuin PIR-sensori, ja ilmaisee havaintonsa tulesta LOW-arvolla. Emme tiedä onko tämä virhe käytetyssä GPIO-kirjastossa, vai itse sensorissa.

   Kun sensori taas havaitsee arvoksi HIGH, eli tulta ei ole havaittu, julkaisemme Paho-MQTT:llä ”nuotiovahti/flame/-topiciin viestin, jonka sisältö on ”No flame detected”.

4) time.sleep(2) määrittelee sensorin lähettämän datan tiheyden. Lähetämme dataa siis kahden sekunnin välein.

2.4 Mobiilidatayhteys

Sensorilaitteemme tulee sijaitsemaan keskellä kansallispuiston metsää. Näin ollen tarvitsimme sille mobiilidataratkaisun, joka oli sekä halpa että resurssikevyt ja kestävä ulko-olosuhteissa. Tähän tarkoitukseen hankimme aluksi Iteadin RPI SIM800 GSM/GPRS lisämoduulin, joka on nimenomaan suunniteltu Raspberry Pi -tietokoneille. Se kuitenkin valitettavasti alkoi tuottamaan ongelmia, jonka jälkeen se palautettiin ja tilalle hankittiin langattoman verkon jakava 4G-modeemi. Selitämme tässä raportissa kuitenkin molempien laitteiden konfiguroinnin. SIM-kortti saatiin koulun IT-tuelta.

2.4.1 Itead SIM800

SIM800-modeemi tukee GSM-, GPRS- ja Bluetooth-yhteyksiä. Lisäksi se mahdollistaa puheluiden ja tekstiviestien lähettämisen ja vastaanottamisen. Meitä kuitenkin kiinnosti tätä projektia varten ainoastaan toimiva internetyhteys, eikä käyttämämme SIM-kortti tue kuin mobiilidataa. Moduulia on mahdollista hallita serial-väylän kautta AT-komennoilla. Tämä osoittautui meidän kannaltamme lopulta tarpeettomaksi, sillä AT-komennot eivät tarjonneet haluttua toiminnallisuutta käynnistää modeemi ilman virtakytkimen painallusta.

Laitteen dokumentaatiossa mainittiin, että SIM800-moduuli saattaa tarvita oman 5V/2A USB-tulovirtansa. Testeissämme emme kuitenkaan havainneet tarvetta lisävirralle. Valmistajan ilmoittama käyttölämpötila oli -40°C ja +85 °C välillä. Näin ollen laitteen tulisi kestää Suomen vaihtelevat vuodenajat.

Kuva 7. SIM800-moduli ilman antennia.

SIM800-mobiilidatamoduuli yhdistettiin Soneran langattomaan verkkoon **Sakis3G**-skriptillä. Vaihtoehtoinen ohjelma olisi ollut **wvdial**, mutta emme saaneet sillä mobiilidatamoduulia yhdistettyä mobiiliverkkoon.

Sakis3G ladattiin wget-komennolla seuraavasta lähteestä:

```
wget "https://www.modmypi.com/download/sakis3g.tar.gz"
```

Paketti purettiin ja skriptistä tehtiin suoritettava chmodilla. Sitten muutimme **/boot/cmdline.txt** -tiedostoa vapauttaaksemme moduulin tarvitsemat serial-portit.

```
$ cat /boot/cmdline.txt
dwc_otg.lpm_enable=0 root=PARTUUID=f3c42936-02 rootfstype=ext4 el-
evator=deadline fsck.repair=yes rootwait
```

Seuraavaksi muokkasimme **/lib/system/system/hciuart.service**, jotta se ei käyttäisi serialia. Kommentoimme tiedostosta pois ”After=dev-serial1.device” ja ”ExecStart=/usr/bin/btuart” -kohdat, jonka jälkeen konfigurointitiedosto näytti tältä:

```
$ cat /lib/systemd/system/hciuart.service
[Unit]
Description=Configure Bluetooth Modems connected by UART
ConditionPathIsDirectory=/proc/device-
tree/soc/gpio@7e200000/bt_pins
Before=bluetooth.service
Wants=dev-serial1.device
#After=dev-serial1.device

[Service]
Type=forking
#ExecStart=/usr/bin/btuart
ExecStart = /usr/lib/hciattach /dev/ttyS0 bcm43xx 460800 noflow -
[Install]
WantedBy=multi-user.target
```

Lopuksi muokkasimme vielä **/boot/config.txt**, johon lisäsimme seuraavat kolme kohtaa tiedoston loppuun:

```
dtoverlay=pi3-miniuart-bt
enable_uart=1
force_turbo=1
```

Serial oli näillä muokkauksilla vapautettu, jotta SIM800 pystyi yhdistämään mobiiliverkkoon.

Tämä skripti ajettiin sudo-oikeuksin Cronilla. Cronin käytöstä selitämme lisää raportin kohdassa 2.7.2. Komennon voi toki ajaa manuaalisestikin, mutta automatisointia varten tarvitaan Cron. Kannattaa huomata, että laite ei voi yhdistää mobiiliverkkoon, jos virtanappia ei ole ensin painettu pohjassa muutaman sekunnin ajan. Virtavalo palaa, vaikka laite ei olisikaan päällä.

```
#!/bin/bash
# Connects to Sonera mobile network.
# Note: Make sure the SIM800 module is powered on!

sudo /home/pi/sakis/sakis3g connect --console --nostorage --pppd
APN="internet" BAUD=9600 CUSTOM_TTY="/dev/ttyAMA0" MODEM="OTHER"
OTHER="CUSTOM_TTY" APN_USER="" APN_PASS="" SIM_PIN="" --noprobe
```

Tuo skripti ajettiin siis Cronin toimesta käynnistyksen yhteydessä. Mobiilidataan yhdistämisen onnistuminen näkyi siitä, kun SIM800-moduulin aktiivisuus-LED alkoi välkkymään nopeammin, sekä käyttöjärjestelmästä ifconfig-komennon kautta ppp0-yhteytenä.

```
$ ifconfig
...
ppp0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST>  mtu 1500
       inet 100.65.176.12  netmask 255.255.255.255  destination
10.64.64.64
       ppp  txqueuelen 3  (Point-to-Point Protocol)
       RX packets 6  bytes 72 (72.0 B)
       RX errors 0  dropped 0  overruns 0  frame 0
       TX packets 7  bytes 111 (111.0 B)
       TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
...
```

SIM800 toimi meillä ison osan projektista odotetulla tavalla. Kuitenkin pian Witty Pi-virranhallintamoduulin käyttöönoton jälkeen huomattiin, ettei SIM800 toiminut enää oikein. PIR-sensori alkoi lähettämään virheellistä dataa, eikä havainnut enää oikeaa liikettä.

Ensin, epäilimmekin syyksi juuri pari päivää ennen ongelmien esiintymistä asennettua Witty Pi-virranhallintamoduulia, tai pikemminkin sitä varten asennettua ohjelmistoa, minkä seurauksena asensimme uudelleen Raspberry Pi:lle käyttöjärjestelmän ja kaikki ohjelmat, pois lukien Witty Pi:n ohjelmiston. Näin poissuljimme mahdollisen ohjelmistojen välisen konfliktin. Uudelleen asennuksen jälkeen Raspberry oli aiempaa vastaavassa tilassa, jolloin SIM800 oli toiminut PIR-sensorin kanssa normaalisti.

Havaitsimme kuitenkin, että ongelmat sensoridatan kanssa jatkuivat. Kokeilimme erilaisia kokoonpanoja ja havaitsimme, että mobiilidatayhteyteen yhdistäminen johti sensorien toimimattomuuteen. Sensori lähetti dataa oikein aina, kun laitetta ei ollut yhdistetty Soneran mobiiliverkkoon Sakis3G:llä. Jostain syystä mobiiliverkkoon yhdistäminen sai Raspberry Pi:n menettämään virtaa, jolloin se ei kyennyt syöttämään sensorille sen tarvitsemaa tasaista volttimäärää, mikä taas johti vialliseen dataan. Tämä näkyi Raspberryn oman virtavalon pätkimisenä hetkinä, jolloin mobiilidatayhteyteen yhdistettiin tai sillä lähetettiin dataa.

Totesimme mobiilidatamoduulin olevan fyysisesti vioittunut. Tarkkaa syytä vian aiheutumiselle ei saatu selville, ja koska projekti läheni loppuaan, tarvitsimme pikaisesti vaihtoehtoisen mobiiliyhteysratkaisun prototyypillemme.

2.4.2 Minicom ja AT-komennot

Vaikka hylkäsimmekin SIM800-moduulin sen yllättävän toimimattomuuden vuoksi, ehdimme kuitenkin tutustua sille tarkoitettuihin AT-komentoihin, eli Hayesin komentokieleen, lyhyesti ennen sen hajoamista. Jos jatkossa uusi projektiorganisaatio hankkii vastaavan laitteen mobiilidataratkaisuksi, olisi hyvä tiedostaa myös kyseisen mobiilidataratkaisun hallintatavat mm. sen virranhallinnan suhteen.

Sensorilaitteemme toimii akun varassa. Siksi etsimme kaikenlaisia tapoja hallita Raspberry Pi:n virrankulutusta. Yksi loogisimpia tapoja olisi ollut sammuttaa laite yöaikaan. Automaattinen, ajastettu sensorilaitteen sammutus ja käynnistys olisi säästänyt virtaa huomattavasti, ja siihen pyrittiinkin, kuten raportin kohdassa 2.8 selitetäänkin. SIM800-moduuli vaatii käynnistyäkseen fyysisen virtakytkimen painalluksen, mikä osoittautui kaavaillulle virranhallinnalle esteeksi.

Etsimme mahdollisia tapoja käynnistää SIM800-moduuli ohjelmiston kautta. Selvitimme, että sitä on mahdollista hallita AT-komennoilla serial-yhteyden kautta. Pelkkä Raspbianin komentotulkki ei riittänyt tähän, joten etsimme sopivan ohjelman. Päädyimme valitsemaan Minicom-terminaaliemulaattorin, joka kykenee serial-pohjaiseen modeemien hallintaan.

Kun myöhemmin hylkäsimme SIM800-moduulin, emme palauttaneet asetuksia, sillä tämä ei ollut tarpeen. Yhtäältä siksi, etteivät asetukset häiritse minkään muun asian toimintaa ja toisaalta siksi, että nollasimme ja uudelleenasensimme Raspberry Pi:n muutaman kerran. Jatkokehitystä ajatellen nämä vaiheet ovat kuitenkin turhia ellei mobiilidataratkaisu ole tuolloin muutettu vastaavaksi.

Aloitimme kytkemällä pois serial-yhteyden kautta kirjautumisen (**serial login shell**) raspi-configissa. Tämän avulla vapautimme serial-portin omaan käyttöömme modeemin hallintaa varten. Se tehtiin raspi-configin interfacing-asetusten serial-asetuksissa, jossa kytkettiin pois ”serial login shell”.

Mikäli emme olisi aiemmin muokannut **/boot/config.txt**-tiedostoa kuten edellä, täytyisi varsinaisen serial-kommunikaation mahdollistamiseksi tehdä samat lisäykset. Nyt ne oli jo kuitenkin valmiiksi tehty, joten käynnistimme Pi:n uudelleen, jotta muutokset tulevat voimaan.

Sitten asensimme minicom-ohjelman normaalisti apt-getin avulla. Minicom käynnistettiin komennolla ”**sudo minicom -o -s**”. Lippu -o ohittaa initialisoinnin. Käytimme tätä koska

edellinen istunto epäonnistui satunnaisesta syystä. Lippu -s avaa Minicomin setup-tilan, jossa sijaitsevat haluamamme asetukset. Valitsimme kohdan ”**serial port setup**” ja teimme tarvittavat asetukset:

```
Serial Device: /dev/ttyAMA0
Bps/Par/Bits: 9600 8N1
```

Kokoonpano tallennettiin vaihtoehdosta “save as dfl”, jonka jälkeen Minicom käynnistettiin uudelleen. Minicomin konsoli ei oletuksena näytä kirjoitettuja komentoja tai niiden tulosteita. Jouduimme siksi sokkona kirjoittamaan komennon, joka kytkee Minicomin ”local echo”:n päälle:

```
Welcome to minicom 2.7

OPTIONS: I18n
Compiled on Apr 22 2017, 09:14:19.
Port /dev/ttyAMA0, 09:17:01

Press CTRL-A Z for help on special keys

AT+ECHO=1
```

Kuva 8. Esimerkki Minicomiin syötetystä komennosta. Local echo on nyt päällä.

Nyt näimme, mitä teemme Minicomissa. AT-komentojen syntaksi on <alkuosa>+<komento>=<arvo>. Yleisin alkuosa on AT. Tässä komentona on melko itsestään selvä ECHO. Arvolla 1 taas kytketään asioita päälle.

Nyt kun pystyimme hallitsemaan modeemia, tajusimme, ettei modeemin komentojen lista sisällä komentoa käynnistämiseen. Modeemin voi kyllä sammuttaa AT-komentojen avulla, mutta käynnistämiseen tarvitaan edelleen virtakytkimen painallus. Sen perusteella mitä siis testasimme, niin SIM800-moduulin sammuttaminen yön ajaksi, on haastavaa, sillä sen käynnistäminen taas vaatii manuaalisen virtanapin painalluksen. Raportoimme tämän kokeilun kuitenkin, sillä perustein, että siitä voi olla hyötyä joissain vastaavanlaisilla ratkaisuissa, jotka hyödyntävät myös SIM800-piiriä.

Mobiilidatamoduulin sammutus AT-komennolla: **AT+CPOWD=<n>**, jossa ”n”:n arvoksi annetaan joko 0 tai 1, joista 0 tarkoittaa, että modeemi sammutetaan välittömästi ilman normaaleja sammutus -menettelyitä. 1 taas tarkoittaa normaalia sammutusta ja sen pitäisi olla turvallisempi käyttää.

2.4.3 Huawei E5577Ts-321 LTE-modeemi & WiFi-tukiasema

SIM800:n näennäisesti hajottua hankimme tilalle Huawein E5577Ts-321-tukiaseman, joka jakoi Raspberrylle langattoman WiFi-verkon. Ratkaisu ei ollut täysin optimaalinen, sillä se oli kallis verrattuna moniin muihin vaihtoehtoihin. Päädyimme kuitenkin siihen, koska SIM800-moduulin hajotessa oli jo kiire päästä testaamaan laitetta ulkotiloissa.

Huawein modeemin saimme hankittua hyvin nopeasti, ja sen käyttöönotto oli suoraviivaista. Haittapuolia olivat lähinnä suurempi hinta ja kovempi virrankulutus, joka on seurausta langattoman verkon jakamisesta Raspberry Pi:lle. Myöhemmin ilmeni myös, että modeemissa on mobiilidatan pakotettu aikakatkaisu, jos liikennettä ei ole. Aikakatkaisua ei voi säätää kokonaan pois ja maksimiaika, jonka sille voi asettaa, on kaksi tuntia. Käytännössä se tarkoittaa, että vähintään kerran kahdessa tunnissa pitää lähettää jotain, sillä muuten modeemi sammuttaa virtansa automaattisesti. Modeemia ei voi käynnistää ohjelmistoteitse, vaan se vaatii manuaalisen käynnistyksen virtanapista. Tämä oli harmillinen piirre, sillä tarkoituksenamme oli sammuttaa sensorilaite kokonaan yön ajaksi.

Huawei E5577Ts-321 tuli Soneran asetuksin esikonfiguroituna, mutta se tukee myös muiden operaattoreiden liittymiä ja hakeekin niiden asetukset automaattisesti.

Modeemi konfiguroitiin verkkohallintasivun kautta osoitteessa http://192.168.8.1, mikä tietenkin edellytti, että käytettävä laite on samassa modeemin oletusasetuksilla jaettavassa WLAN-verkossa. Hallintapaneelin kautta muutettiin laitteen tunnukset, verkon SSID ja salasana. Jaettava verkko myös piilotettiin, eli verkon SSID ei näy muille laitteille.

Langatonta verkkoa varten asetettiin staattinen IPv4-osoite raportin kohdassa 2.2. Enää piti vain yhdistää Raspberry Pi kyseiseen langattomaan verkkoon. Avoimia verkkoyhteyksiä voi skannata komennolla ”sudo iwlist wlan0 scan”. Koska olimme piilottaneet verkon SSID:n, ei kyseinen komento auttanut meitä. Tiesimme kuitenkin verkon SSID:n ja salasanan, sillä konfiguroimme ne modeemin asetuksista.

Raspberry Pi yhdistettiin langattomaan verkkoon muokkaammalla tiedostoa **/etc/wpa_supplicant/wpa_supplicant.conf**. Tiedoston loppuun lisättiin:

```
network={
        ssid="nuova"
        scan_ssid=1
        psk="vahtiVerkko33!"
}
```

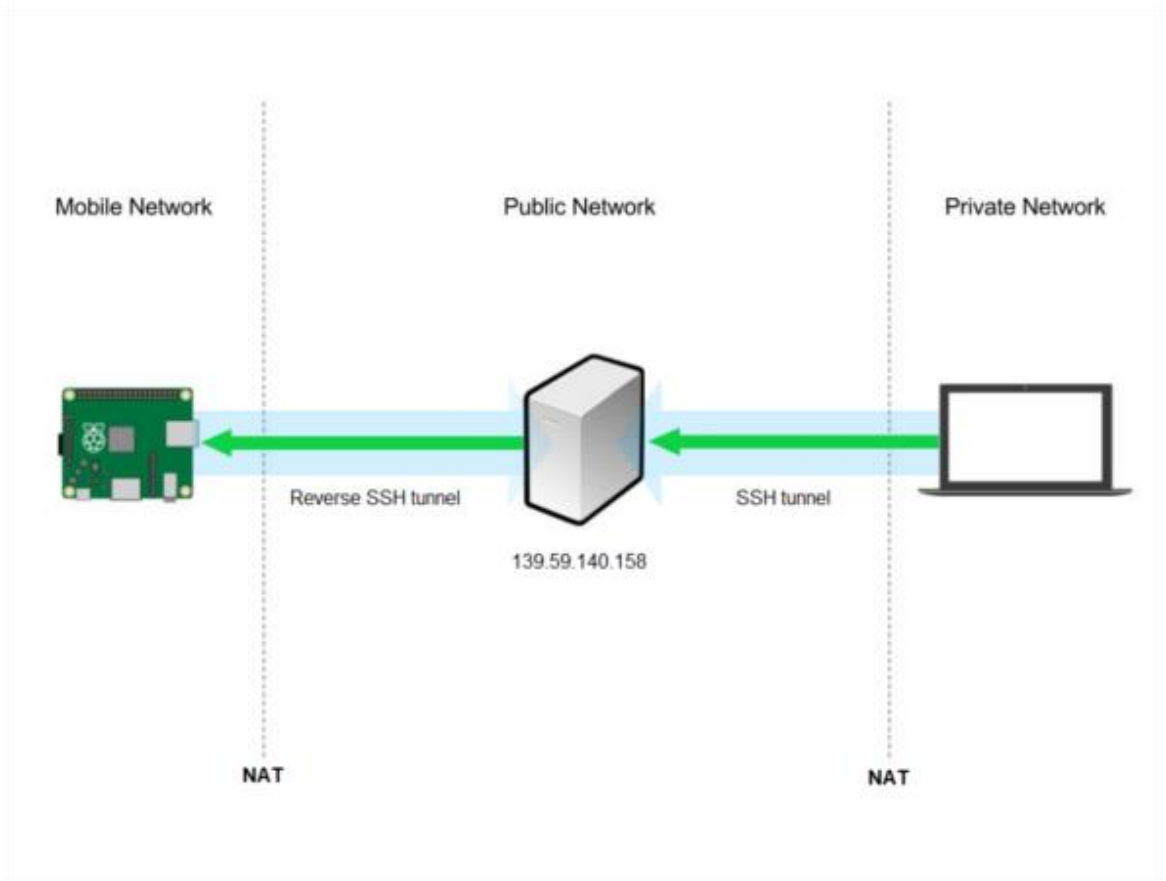
Koska verkkomme oli piilotettu, tarvitsimme asetuksen ”scan_ssid=1”. Käynnistimme Raspberry Pi:n uudelleen, jonka jälkeen se oli saanut IP-osoitteen Huawein modeemilta.

2.5 Sensorilaitteen etähallinta

Yksi haaste projektin aikana oli keksiä, miten mobiiliverkossa olevaan IoT-sensorilaitteeseen päästäisiin halutessa kiinni julkisesta verkosta. Mobiilidatalla toimiva Raspberry Pi on osoitteenmuunnoksen (NAT) takana, ilman omaa yksilöllistä ja julkista IP-osoitetta. Etäyhteyttä kuitenkin tullaan tarvitsemaan mahdollisten vikatilojen selvittämiseen, jos vaikka sensoreita pyörittävä Python-skripti kaatuisi. Näin ollen prototyyppiin tulisi saada etäyhteys milloin vain.

Lopulta päädyimme seuraavanlaiseen ratkaisuun: Raspberry Pi luo käänteisen SSH-tunnelin julkisessa verkossa olevaan Ubuntu-palvelimeemme. Tunnelia ylläpidetään Shell-skriptillä Raspberryn päässä, jolloin ylläpitäjä voi palvelimelta kirjautua Raspberry Pi:hin tunnelin kautta. Harkitsimme myös keskitetyn hallinnan työkaluja, kuten Puppet tai Salt, mutta päädyimme siihen, että tavallinen SSH-yhteys on monipuolisin lähestymistapa etähallinnan toteuttamiseen. Projektin loppupuolella konfiguroimme kuitenkin SaltStackin käänteisen SSH-tunnelin lisäksi.

2.5.1 Käänteinen SSH-tunneli



Kuva 9. Etähallintaratkaisun yksinkertaistettu toimintaperiaate.

SSH-avaimet

Automatisointia varten loimme SSH-avaimet palvelimen ja sensorilaitteen välille. Avaimet luotiin Raspberryllä seuraavasti:

```
cd ~/.ssh
ssh-keygen -t rsa
```

Emme luoneet avaimille passphrasea. Julkinen avain kopioitiin palvelimelle scp:llä.

```
scp id_rsa.pub markus@139.59.140.158:.ssh/authorized_keys
```

Tunnelin luominen

Tunneli luodaan **remoteconnection.sh** -skriptillä, jota ajetaan 30 minuutin välein Cronin toimesta. Cronin toiminnasta selitämme tarkemmin raportin osiossa 2.7.2. Kaikille skriptille annettiin tietenkin chmodilla suoritusoikeudet.

```
#!/bin/bash
# Script for creating a reverse SSH tunnel from Raspberry Pi to
the server

        ssh -f -N -o BatchMode=yes -R 2222:localhost:22
markus@139.59.140.158
```

Skripti ottaa SSH-yhteyden palvelimen IP-osoitteeseen käyttäjän ”markus” tunnuksin. Samalla se käskee palvelinta avaamaan portin 2222. Kaikki SSH-yhteydet palvelimen porttiin 2222, ohjataan automaattisesti tunnelia pitkin Raspberry Pi:n porttiin 22.

- **-f** tarkoittaa, että haluamme SSH:n pyörivän taustalla käyttäjän todentamisen jälkeen. Tunneli pysyy näin pystyssä, vaikka mitään liikennettä tunnelin lävitse ei tapahtuisikaan.
- **-N** tarkoittaa, että emme ole suorittamassa mitään komentoja yhteyden jälkeen.
- -**o BatchMode=yes** tarkoittaa, että yhteys epäonnistuu, jos SSH-avaimet eivät ole kunnolla konfiguroitu. Yhteys ei siis jää roikkumaan, jos käyttäjän todennus epäonnistuu.
- **-R** tarkoittaa, että luomme tunnelin, jonka sisäänpääsypiste on yhteyden etäpäässä, eli tässä tapauksessa palvelimella. Luomme siis käänteisen SSH-tunnelin.

Kun Raspberry Pi on luonut SSH-tunnelin, pääsemme siihen palvelimelta käsin kiinni ajamalla manuaalisesti **tunnelconnection.sh** -skriptin.

```
#!/bin/bash
# Connects to remote tunnel.

    ssh -l pi -p 2222 localhost
```

Otamme skriptillä SSH-yhteyden localhostiin, eli palvelimen porttiin 2222. Tämä toimii, koska portti kuuntelee tulevia SSH-yhteyksiä. Jos se löytää sellaisen, palvelin ohjaa kaiken liikenteen jo aiemmin yhdistettyyn SSH-tunneliin.

Teimme palvelimelle myös skriptin **checkconnection.sh**, jolla voidaan tarkistaa, onko tunneli yhdistynyt.

```
#!/bin/bash
#Check alive remote connections.

          sudo lsof -i TCP:2222
```

Loimme skriptejä noinkin yksinkertaisista komennoista, koska sillä säästettiin aikaa. Niitä kuitenkin ajettiin testikäytössä jatkuvasti. Jos tunneli on pystyssä, portin 2222 pitäisi olla kuuntelevassa tilassa. Monesti kuitenkin kävi siten, että Raspberryn uudelleenkäynnistymisen jälkeen portin yhteys näytti yhä olevan pystyssä. Tosiasiassa palvelin kuitenkin yritti yhä kuunnella sitä yhteyttä, joka oli pystyssä ennen kuin Raspberry sammui. Vanhat yhteydet kannattaakin aina tappaa kyseisestä portista, jotta sensorilaite pystyy yhdistämään porttiin sen tullessa takaisin päälle. Sitä varten loimme skriptin nimeltä **killconnections.sh**.

```
#!/bin/bash
# Script for killing old connections to port 2222. Run this after
remote connection is over.
# Note! This is run on the server! You can check if (CLOSE WAIT)
connections are still alive with: "sudo lsof -i TCP:2222".

          sudo lsof -t -i tcp:2222 -s tcp:listen | sudo xargs kill
```

Kun vanhat yhteydet on tapettu, uusi tunneli luodaan Cronin pyörittämällä aikavälillä, eli 30 minuutin välein.

Tätä etähallintatapaa käytettiin onnistuneesti lähes koko projektin ajan. Vaikka se onkin ideana kömpelö, ei sen tuomia ongelmia ehditty havaita ajoissa. Vasta myöhemmin, kun palvelimella alkoi olla enemmän sisältöä, mm. tietokanta, ongelmat alkoivat esiintyä. Tunneliskripti ottaa yhteyttä säännöllisin väliajoin huolimatta siitä, onko toimivaa tunnelia jo pystyssä. Tämän ei ajateltu haittaavan, sillä yhteydenotto epäonnistuu, jos portti kuuntelee jo jotain aiempaa yhteyttä. Palvelimelta alkoi kuitenkin loppumaan muisti, minkä seurauksena MySQL-tietokanta kaatuili. Tajusimme, että etähallintaratkaisumme luo palvelimelle jatkuvasti uusia sshd-prosesseja, vaikka yhteydet eivät menekään läpi. Nuo prosessit veivät merkittävän osan palvelimemme keskusmuistista.

Ongelmaa alettiin ratkoa muokkaamalla tunnelia luovaa skriptiä fiksummaksi. Siihen pyrittiin laittamaan tarkistus, joka selvittää, onko toimiva tunneli jo pystyssä, ennen kuin se luo uuden. Tarkistusta yritettiin mm. tarkastelemalla Raspberry Pi:n prosesseja ja netstat-komennon tuloksia. Toimivan yhteyden varmistaminen luotettavasti osoittautui yllättävän vaikeaksi. Prosessit eivät kerro mitään siitä, että yhteys toimii yhä päästä päähän, sillä taustalle jää aina zombieprosesseja, jotka on ajettu aikoja sitten. Mikään niistäkään ei välttämättä enää ylläpidä toimivaa SSH-yhteyttä. Netstat näytti myös, että palvelimen IP-osoitteeseen oli luotu ”Established”-yhteys, vaikka tunneli oli kaatunut palvelimen päässä jo ajat sitten. Seurauksena oli se, että tunnelia ei luotu, sillä Raspberry luuli, että se on pystyssä.

```
#!/bin/bash
# Script for creating a reverse SSH tunnel from Raspberry Pi to
the server

output=$(netstat | awk '$5 ~ /^139.59.140.158:ssh/ && $6 ~ /ESTAB-
LISHED/')
connection=$output

if [ "$connection" ]; then
:
else
ssh -f -N -o BatchMode=yes -R 2222:localhost:22
markus@139.59.140.158
fi
```

Esimerkiksi ylläolevan skriptin tarkoituksena oli juurikin tarkistaa ensin netstatilla, onko palvelimen osoitteeseen toimivaa SSH-yhteyttä. Jos kyllä, niin mitään ei tehdä. Jos ei, se luo tunnelin. Emme vain kyenneet mitenkään luotettavasti saamaan varmistusta siitä, että tunneli oikeasti toimii myös palvelimen päässä. Autossh-ohjelmaa testattiin, mutta sekään ei kyennyt itsenäisesti nostamaan tunnelia takaisin ylös muutaman minuutin nettikatkoksen jälkeen. Olisimme toki voineet tappaa timeout:illa SSH-tunnelin tietyn ajan päästä. Se olisi vain tuntunut ratkaisuna kömpelöltä, jos käyttäjät heitetään juuri silloin etäyhteydestä kesken töiden. Päätimme sitten lopulta tehdä **terminatesshd.sh** -skriptin palvelinpäähän, jolla tapetaan ne sinne muodostuneet turhat sshd-prosessit. Se tarkistaa ensin, mikä sen toimivan SSH-tunnelin prosessi ID (PID) on. Sen jälkeen se selvittää, onko kukaan käyttäjä SSH-yhteydessä palvelimelle. Jos käyttäjiä ei ole palvelimella, kaikki sshd-prosessit tapetaan paitsi se, jonka ID selvitettiin heti skriptin alussa.

```
#!/bin/bash

CheckPID=$(sudo lsof -ti TCP:2222)
PID=$CheckPID

CheckSession=$(who)
session=$CheckSession

#CheckSession = $(ps aux | grep '[m]arkus@pts/0')
#session = $CheckSession

if [ "$session" ]; then
:
#echo 'Someone is logged in! Aborting...'

else
ps aux | grep 'sshd: markus' | awk '{print $2}' | grep -v $PID |
sudo xargs kill -9
#echo 'Just killed all processes!'

fi
```

Nopealla testauksella, juuri raportin palautusta edeltävällä viikolla, skripti testattiin toimivaksi. Sitä voi kokeilla ensin echo-komennoin, jos ei uskalla alkaa tappamaan prosesseja automaattisesti. Tuota skriptiä olisi sitten tarkoitus ajaa Cronin toimesta sudo-oikeuksin. Prosessien tappaminen ei onnistu, jos sitä ei ajeta sudon crontabissa. Skriptiä voisi tehdä fiksummaksi siten, että se tarkistaa myös mm. markus-käyttäjän PID:n ja suodattaa sen kanssa tuon else-lauseen komennossa. Tapettaisiin siis kaikki sshd-prosessit paitsi se, joka liittyy toimivaan etäyhteystunneliin tai palvelimen tavallisten käyttäjien ssh-istuntoihin. Ideaa ei kuitenkaan ehditty implementoida, joten sitä voi miettiä jatkokehityksen kannalta. SSH-tunnelia luovassa skriptissä olisi voinut toimivaa yhteyttä myös testata netcat-komennolla, mutta sitäkään emme ehtineet testaamaan ennen kurssin loppumista.

2.5.2 SaltStack

SSH-pohjainen etähallintatapa ei ole tuotantokelpoinen eikä skaalautuva. Jos sensorilaitteiden määrää kasvatettaisiin, niin niiden ottaminen samaan etähallintaan vaatisi järjettömän määrän manuaalista konfigurointia. Etähallintaa ei myöskään kyettäisi toteuttamaan keskitetysti kaikille laitteilla samanaikaisesti. Tästä syystä konfiguroimme SaltStackin.

Salt on keskitetyn hallinnan työkalu, joka toimii master-minion periaatteella. Salt-minion ylläpitää yhteyttä julkisessa verkossa olevaan Salt-Masteriin tämän IP-osoitteen perusteella. Masterin ei itse tarvitse tietää missä sen minionit sijaitsevat, vaan se tunnistaa ne yksilöllisesti salt-avainten (salt-key) perusteella. Masterilta voidaan sitten ajaa komentoja joko yksittäiselle minionille tai kaikille samanaikaisesti.

Meitä kiinnosti lähinnä sensorilaitteen ongelmanratkonta mm. lokitietoja lukemalla. Tarvitsimme myös kyvyn käynnistää Raspberry Pi uudelleen tarvittaessa. Salt mahdollistaa tavallisten shell-komentojen ajamisen minioneille. Sillä pystytään myös määrittelemään, millaiseksi minionin halutaan mukauttavan itsensä (Salt State). Voidaan siis esimerkiksi määritellä, että kaikilla minioneilla tulisi olla LAMP asennettuna, jolloin ne asentuvat kaikille minioneille, joilla se ei ole jo asennettuna. Hyöty näkyisi varsinkin sitten, kun useaan uuteen Raspberry Pi-tietokoneeseen tulisi asentaa kaikki tarvittavat skriptit ja ohjelmat, jotta se kykenisi toimimaan sensorilaitteena. Saltilla voidaan keskitetysti asentaa ja konfiguroida uudet sensorilaitteet toimintakuntoon.

Salt-Master

Salt-Master asennettiin aluksi samalle palvelimelle prototyyppimme webpalvelimen kanssa. Saltia ei kuitenkaan saatu konfiguroitua oikein, ja myöhemmin syyksi selvisi keskusmuistin määrän loppuminen. Muistin loppumisen seurauksena, poistimme Salt-Masterin palvelimelta ja loimme DigitalOcean-pilvipalveluun täysin uuden palvelimen isommalla muistimäärällä (3GB RAM / 1 CPU / 20 GB Disk).

Asennus tapahtui salt-bootstrap-skriptillä, jonka uusin versio asennettiin curlilla.

```
curl -L https://bootstrap.saltstack.com -o install_salt.sh

sudo sh install_salt.sh -P -M git v2017.7.2
```

Asennuksen yhteydessä tuli käyttää “-M” parametria, jos halutaan asentaa pelkän Salt-Masterin. Asentumisen jälkeen, muokkaisimme Masterin config-tiedostoa sijainnissa **/etc/salt/master**, ja laitoimme masterin kuuntelemaan omaa IP-osoitettaan. Annoimme Masterille myös oman ID:n:

```
# Set and ID for your master:
master_id: NuotiovahtiMaster

# The address of the interface to bind to:
interface: 46.101.235.80
```

Muutosten jälkeen salt-master service käynnistettiin uudelleen. Salt-Master kuuntelee oletuksena portteja 4505 ja 4506. Tulikin siis varmistaa, että portit ovat palomuurissa auki. Master-palvelimen palomuurisäännöt:

```
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW IN    Anywhere
4505                       ALLOW IN    Anywhere
4506                       ALLOW IN    Anywhere
22/tcp (v6)                ALLOW IN    Anywhere (v6)
4505 (v6)                  ALLOW IN    Anywhere (v6)
4506 (v6)                  ALLOW IN    Anywhere (v6)
```

Salt-Minion

Myös Salt-minion olisi kannattanut asentaa salt-bootstrap skriptillä. Jostain syystä uusimman version asennus ei sillä kuitenkaan mennyt Raspberry Pi:lle läpi, vaan se valitti jostain riippuvuusongelmista, eli yhteensopivuudessa oli puutteita sillä kyseisellä versiolla. Asensimmekin minionin lopulta eri lähteestä. Ensin importtasimme SaltStackin repositorio-avaimen wget:illä.

```
wget -O - https://repo.saltstack.com/apt/debian/8/armhf/lat-
est/SALTSTACK-GPG-KEY.pub | sudo apt-key add -
```

Ylläolevan komennon jälkeen, muokkasimme sijaintia **/etc/apt/sources.list.d/salts-tack.list**, jonne lisäsimme seuraavan rivin:

```
deb http://repo.saltstack.com/apt/debian/8/armhf/latest jessie
main
```

Päivitimme sitten apt-get pakettivarastot ja asensimme salt-minionin normaalisti paketinhallinnan kautta.

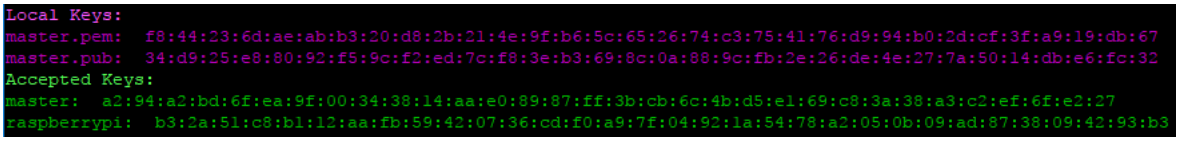
Asennuksen jälkeen, asetimme minionin config-tiedostoon masterin nimen tilalle sen julkisen IP-osoitteen, johon minion, eli Raspberry Pi, tulee jatkossa ottamaan yhteyttä. Hakemistosijaintiin **/etc/salt/minion** tuli siis:

```
# Set the location of the salt master server. If the master server
cannot be
# resolved, then the minion will fail to start

master: 46.101.235.80
```

Enempää minionilla ei tarvitse konfiguroida. Käynnistimme vaan salt-minion servicen uudelleen, jonka jälkeen minion pyrki ottamaan masteriin yhteyttä.

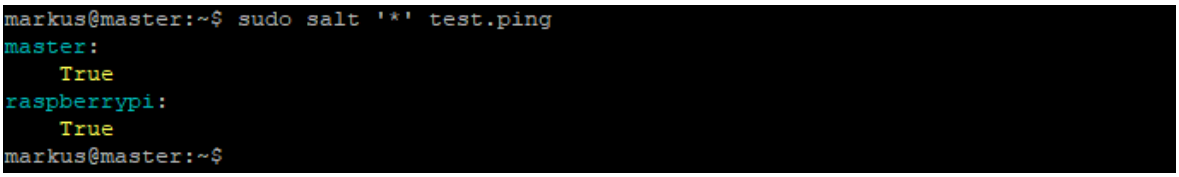
Yhteyden muodostaminen Salt-Masterilla

Salt-minion otti nyt yhteyttä Salt-masteriin, mutta minionin yksilöllinen avain piti vielä hyväksyttää masterilla, ennen kuin yhteys on muodostettu. Listasimme kaikki avaimet komennolla ”**sudo salt-key--list all**”. Raspberrypi-minionin avain näkyi kohdan ”unaccepted keys” alapuolella. Kaikki hyväksymättömät avaimet hyväksyimme komennolla ”**sudo salt-key -A**”, jonka jälkeen raspberryn avain näkyi hyväksyttyjen avainten alla:



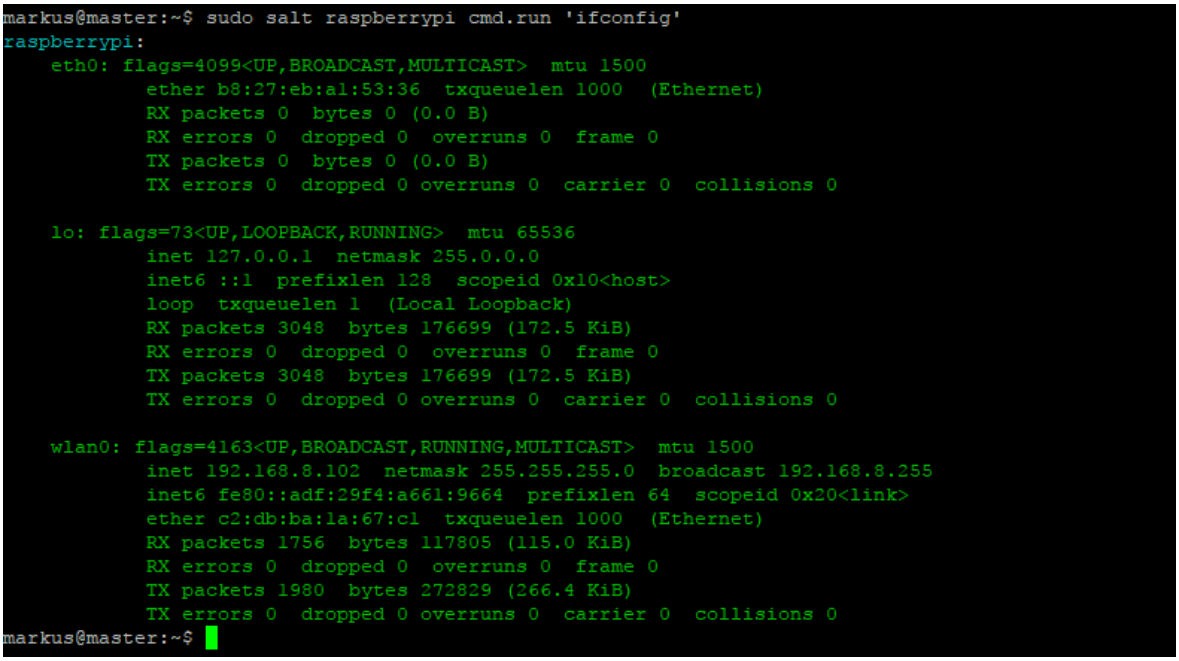
Kuva 10. Salt-masterin hyväksymät avaimet.

Meillä oli aluksi testitarkoituksessa master ja minion samalla palvelimella. Siksi hyväksytyissä avaimissa näkyy myös master-niminen tietokone. Testasimme yhteyttä pingaamalla Raspberryä masterilta komennolla ”**sudo salt raspberrypi test.ping**”, ja minion vastasi:



Kuva 11. Minion-koneiden yhteystesti.

Pääsimme nyt ajamaan sensorilaitteelle shell-komentoja Saltia käyttäen. Komentoja ajettiin muodossa ”**sudo salt raspberrypi cmd.run ’command’** ”, jossa ”raspberrypi” viittaa minionin nimeen ja ”command” suoritettavaan shell-komentoon. Jos haluaa ajaa komennon kaikille minioneille samanaikaisesti, niin se tapahtui laittamalla minion nimen kohdalle **’*’**.
Esimerkiksi ifconfig-komennon ajaminen Raspberrylle komennolla **sudo salt raspberrypi cmd.run ’ifconfig’**



Kuva 12. Raspberryn verkkointerface-tiedot Salt-masterilta pyydettynä.

2.6 Sensorilaitteen automatisointi

Prototyypin toimivuuden kannalta on oleellista, että Raspberry Pi-pohjainen sensorilaite operoi ydintoimintonsa automaattisesti ilman tarvetta manuaaliselle säädölle. Ylläpitäjän tulee kyetä käynnistämään laite uudelleen ilman vaativampaa tietoteknistä osaamista. Prototyyppimme itsetoimintakykyä ylläpidettiin kahdella eri työkalulla: rc.localilla ja Cronilla.

2.6.1 rc.local

Rc.local-tiedostoon voi lisätä komentoja, jotka haluaa ajattavaksi Raspberry Pi:n käynnistymisen yhteydessä. Ajamme sensoreiden Python-skriptit tällä tavoin. Vaihtoehtoinen tapa olisi ollut Cron, mutta testauksessa havaitsimme, että rc.local suoriutui luotettavimmin käynnistyksen yhteydessä ajettavista skripteistä. Tämä johtuu osittain siitä, että se on viimeisin Raspbianin käynnistyksen yhteydessä suorittama palvelu, mikä onkin hyvä, sillä erityisesti PIR-sensorin skripti on herkkä keskeytymään jonkin toisen käynnistyvän prosessin toimesta, sillä siinä on sisäänkirjoitettu 30 sekunnin oppimisaika.

Muokkasimme **/etc/rc.local** -tiedoston loppuun seuraavat rivit. Huomaa, että ajettavat komennot laitetaan konfigurointitiedostoon ennen ”exit 0” riviä.

```
#Start sensors on startup

(sleep 90
python /home/pi/sensors/mqtt_pir_sensor.py) &

(sleep 120
python /home/pi/sensors/mqtt_flame_sensor.py) &
```

Koska PIR-sensorin skriptissä on se sisäänrakennettu oppimisaika, laitoimme liekkisensorin skriptin ajettavaksi 30 sekuntia PIR-sensoria myöhemmin. Tällöin molemmat sensorit aloittavat toimintansa samanaikaisesti. Jätämme varmuudeksi muille prosesseille tarpeeksi aikaa suorittaa itsensä, jotta ne eivät keskeytä taustalla pyöritettäviä sensoriskriptejä.

Komentojen lopussa on ”&”, jotta skriptit eivät pyöri loputtomassa loopissa, ja ne suoritetaan taustalla erillisenä prosessina samalla, kun Raspberry käynnistyy.

2.6.2 Cron

Cronilla voidaan ajoittaa laitteella pyöritettäviä tehtäviä, kuten skriptejä tai komentoja. Siinä, missä rc.local toimii ainoastaan käynnistyksen yhteydessä, cron pystyy ajamaan skriptejä käynnissä olon aikana tasaisin väliajoin. Cronilla voidaan myös ajaa skriptejä käynnistyksen yhteydessä ”@reboot”-vaihtoehdolla. Siten mm. ajoimme Sakis3G:tä suorittavaa skriptiä, joka yhdisti SIM800-mobiilidatamoduulin mobiiliverkkoon aina käynnistyksen yhteydessä. Siinä Cronin @reboot toimikin luotettavasti, mutta sensoriskripteihin ei juurikaan.

Käytimme Cronia jatkuvasti toistuvien tehtävien suorittamiseen. Tehtävät ajoitettiin Cronin ajettaviksi komennolla ”**crontab -e**”. Jos haluaa käyttää Cronia käynnistyksen yhteydessä, niin silloin crontabia tulee ajaa sudo-oikeuksin.

Ajoitimme seuraavat skriptit Cronin suoritettaviksi:

```
# crontab -e

*/30 * * * * /home/pi/automation/remoteconnection.sh >/dev/null
2>&1
*/15 * * * * /home/pi/automation/pingserver.sh >/dev/null 2>&1
```

Cron yrittää oletuksena lähettää outputit tehtävien suorittamisesta käyttäjän sähköpostiin, mikä on estettävissä lisäämällä loppuun ">/dev/null 2>&1". Cronin luotettavuuden ja toimivuuden takaamiseksi seurasimme lokitietoja taustalla suoritetuista tehtävistä läpi projektin. Cronin ajamat prosessit löytyvät sijainnista **/var/log/syslog**.

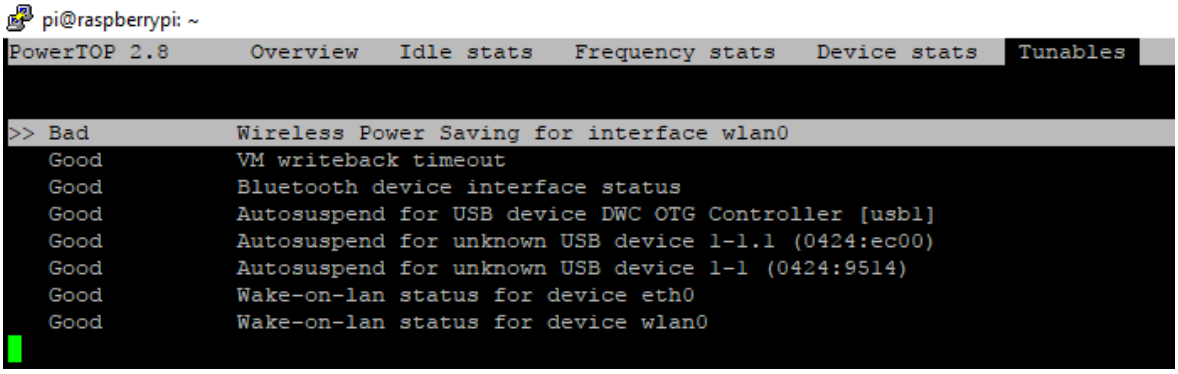
Pingserver.sh-skripti pingaa palvelintamme 15 minuutin välein ja tallentaa tulokset uptime.txt nimiseen tekstitiedostoon. Tämä tehtiin ulkoilmatestausta ajatellen, sillä sensorilaitteen tarkka sammumisaika ei ole kovin helposti selvitettävissä. Virran menettäminen akun loppumisen vuoksi ei tuota samoja lokitietoja, kuin tavallinen shutdown. Skriptin tulosten pohjalta voidaan päätellä, milloin laite on sammunut, tai jos verkkoyhteydessä on ollut häiriöitä. Tulokset sisältävät aikaleiman, jolloin ICMP-pakettiin on tullut vastaus.

```
#!/bin/bash
#Ping nuotiovahti.info and output to uptime.txt.

ping -c 1 139.59.140.158 | while read pong; do echo "$(date):
$pong"; done >> /$
```

Tulokset säilytettiin paikallisesti Raspberryllä, eli niiden lukemiseksi pitää päästä käsiksi itse laitteeseen. Output voitaisiin lähettää ylläpitäjän sähköpostiin, mutta meille riitti se, että tulokset tallennetaan paikallisesti.

2.7 Virransäästö

Projektin ja konseptin toimivuuden oleellisena osana on laitteiden virransaanti ja kulutus. Meidän prototyyppimme toimii puhelimelle tarkoitetulla 30000 milliampeerin varavirtalähteellä. Käytössämme oli myös pienempi 2600 mAh akku. Sillä tehdyt testit osoittivat, että teoreettinen maksimi yhtäjaksoiselle toiminta ajalle olisi 42 tuntia. Lähdimme selvittämään ja löytämään mahdollisia ratkaisua pidempään toimintaa aikaan. Linuxille löytyi kaksi suositeltua virranoptimointiohjelmaa, Powertop ja TLP. Testasimme ja kokeilimme molempia ohjelmia seuraavin tuloksin.



Kuva 13. Powertop.

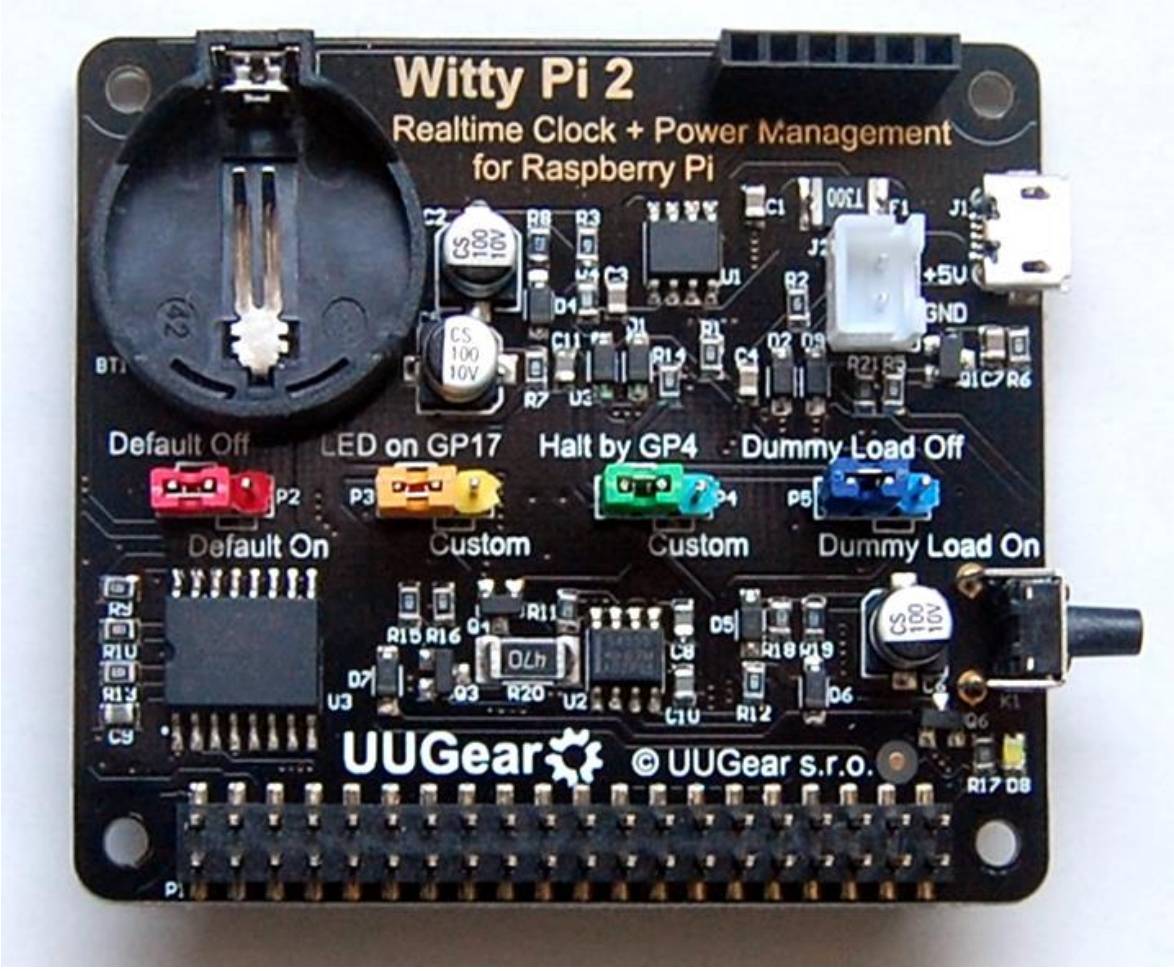
Kuten ylläolevasta kuvasta näkyy, ainoa virransäästöön vaikuttava ”Bad” arvo on WLAN, jota tarvitsemme projektin aikana. Mikäli arvon muuttaisi ”Good” tilaan, lakkaisi wlan0 toimimasta.

TLP:n ainoa ero oli, että ohjelma ehdotti Bluetoothin sammuttamista, mikäli se olisi ollut päällä. Tulimme hyvin nopeasti tulokseen, että Raspberry Pi on jo valmiiksi hyvin optimoitu laite ja virrankäyttö on minimaalista. Mietimme myös, että kannattaako Raspberrystä löytyviä lisälaiteportteja sammuttaa. Löysimme useamman vaihtoehdon poistaa USB-portit käytöstä, mutta Raspberryn kaikki portit ovat kytketty yhteen, mikä tarkoittaa, että yksittäisiä portteja ei pysty erikseen sammuttamaan. Mikäli portit olisi sammutettu, laite olisi jäänyt kokonaan ilman virtaa. Vaihtoehtoisesti aikomuksena oli alikellottaa Pi. Epäonneksemme laiteongelmien ja ajan puutteen takia tämä jäi kokonaan testaamatta. Projektin alussa mietimme myös lisälaitteita, jotka mahdollisesti herättäisivät laitteen horroksesta, nämä tosin tarvitsivat oman virtalähteensä, joten hylkäsimme idean.

2.7.1 RTCWake

Normaaleissa tietokoneissa on mahdollisuus ajastaa toimintoja koska emolevyssä on paristo, joka ylläpitää sisäistä kelloa, vaikka itse laite on sammutettu. Raspissa ei ole tällaista ominaisuutta. Olisimme muuten voineet käyttää RTCWake-ohjelmaa (Real Time Clock), jolla voidaan tehdä ajastettuja käskyjä tietokoneelle. Lähdimme etsimään mahdollista vaihtoehtoa Raspberrylle. Parhaassa tapauksessa laitteemme voitaisiin automaattisesti sammuttaa ja käynnistää ennalta määritettyyn aikaan.

2.7.2 Witty Pi 2



Kuva 14. Witty Pi 2 – kuva otettu UUGearin nettisivuilta

Löysimme tšekkiläisen yrityksen nimeltä UUGear (http://www.uugear.com) – he valmistavat pääasiassa Raspberry Pi:lle ja Arduinolle lisälaitteita. Heillä oli projektin aikana myynnissä virranhallintamoduuli nimeltä Witty Pi 2.

Päätarkoitus laitteella oli prototyyppimme automaattisammutus ja käynnistys. Ideana oli, että laite sammuisi kello 22.00 ja uudelleen käynnistyisi 08.00. Näin säästäisimme kymmenen tuntia vuorokaudessa, sillä Nuuksiossa ei liikkuisi ihmisiä talvella yöaikaan.

Bonuksena laitteesta löytyy akun ylläpitomoduuli, sininen jumpperi. Käytännössä tämä auttaisi projektissa käytettyä litiumioniakkua toimimaan normaalia paremmin. Litiumioniakkuihin on sisäänrakennettu ominaisuus, joka säätelee akun käyttölämpötilaa. Kun akku on päällä, se pitää itse itsensä sopivan lämpöisenä. Esim. kylmässä akku ei anna itsensä jäätyä. Mikäli sinisen jumpperin, ”dummy load” -toiminnan ottaa käyttöön, Witty Pi antaa akulle valekäskyjä eli estää akkua hyytymästä Raspin ollessa pois päältä. Valeviestien sykli varmistaa, että akku ei pääse jäätymään kylmässä ja toimintakyky jatkuu pidempään.

Witty Pi:n ohjelmisto asennettiin UUGearista löytyvällä paketilla. Saatavilla oli myös grafisen käyttöliittymän versio, joka ei ollut meille tarpeellinen. Itse ohjelmisto on rakennettu Bashillä, ja virranhallinta tapahtuu ennalta määriteltyjen komentojen avulla.

```
$ wget http://www.uugera.com/repo/WittyPi/installWittyPi.sh
$ sudo sh installWittyPi.sh
```

Laiteongelmien vuoksi pääsimme itse Witty Pihin kiinni vasta myöhäisessä vaiheessa projektia ja ehdimme juuri ja juuri testata skriptien toimintaa. Testien aikana laite päästi savua pihalle ja rikkoutui.

Todennäköisesti laite oli niin kutsuttu maanantaikappale ja meillä kävi vain huono tuuri. Tämä tapahtui viikko ennen lopullisia testejä ja tässä vaiheessa oli selvää, että emme ehdi saamaan korvaavaa laitetta tilalle. Jouduimme tekemään projektin ilman virransäästöä ja toivomaan parasta, että ei tulisi liikaa pakkasta, ettei akku jäätyisi heti kättelyssä.

2.8 Prototyypin kotelo- ja akkuratkaisut

Aikeemme ulkoilmatestata prototyyppiä tarkoitti, että tarvitsimme mahdollisimman säänkestävän kotelon. Samalla laatikon tulisi olla tarpeeksi suuri, jotta akkumme mahtuisi sisään ja laitteiden asennus ja irrotus onnistuisi mahdollisimman helposti.

2.8.1 Virransyöttö

Virtaratkaisuksi hankimme 30 000 mAh litium-ioni-varavirtalähteen, jossa oli kolme USB 2.0 porttia, joista kukin oli kykenevä syöttämään 5.0V/2.1A edestä virtaa. Useamman USB-portin olemassaolo oli suunnitelmallisesti tärkeää, sillä projektin alussa mobiilidataan käytetty SIM800-moduli olisi saattanut vaatia oman 5V/2A virtansa. Virtalähde oli sopivin vaihtoehto projektiimme kapasiteettinsa, fyysisen kokonsa ja hintansa perustella. Akun latausaika oli 2,5 ampeerin laturilla 12 tuntia, tehottomammalla laturilla latausaika nousee merkittävästi. Testeissä käytimme Vartan valmistamaa 3.7V 2600 mAh litium-Ioni akkua, joka kykeni 1 ampeerin syöttövirtaan.

Kuva 15. Prototyypin 30 000 mAh varavirtalähde.

Tarkoituksena oli ladata sekä Raspberry Pi 3 että Huawei E5577Ts-321 LTE-modeemi samasta isosta akusta. Huaweissa on oma 3000 mAh akku, jota ei valitettavasti kyetä hyödyntämään, sillä virran loppuessa isosta akusta Raspberry Pi sammuu, mutta modeemi jää päälle oman täyden akkunsa varaan. Modeemi ei toimi ilman omaa akkuansa.

Prototyypin pääakku on tuntemattomalta valmistajalta, joten laadusta emme osaa sanoa mitään. Käytön aikana huomasimme, että laitteen numeronäyttö ei pitänyt paikkaansa ja jälkiviisaasti ei olisi pitänyt ostaa akkua, jossa on metallikuori. Metallin lämpöjohtavuuden vuoksi akkumme on alttiimpi jäätymiselle

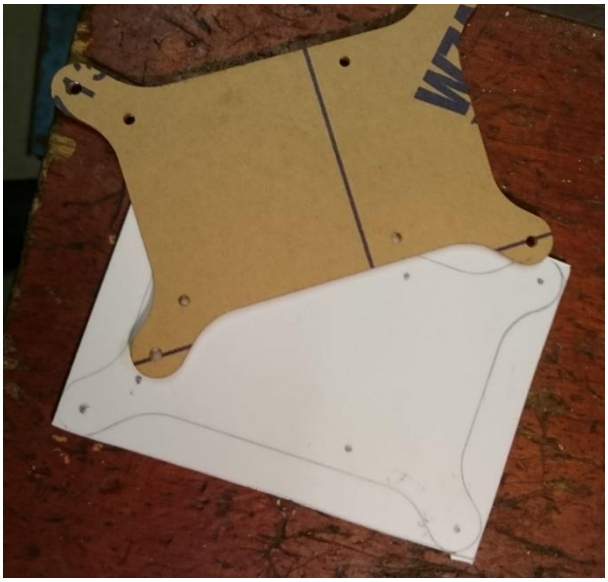
2.8.2 Prototyypin fyysinen rakenne

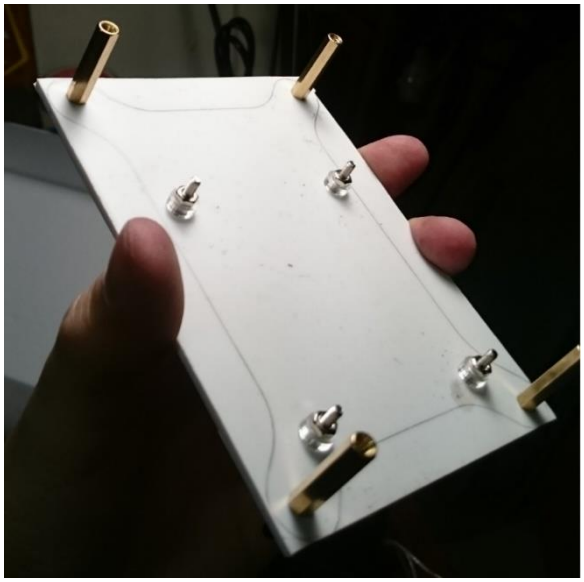
Prototyypin laatikon kriteerejä olivat muun muassa:

- Sopiva koko
- Tarpeeksi tukeva rakenne
- Vedenkestävyys
- Halpa hinta

Näiden kriteerien pohjalta valitsimme pakasterasian, joka maksoi noin kahdeksan euroa.

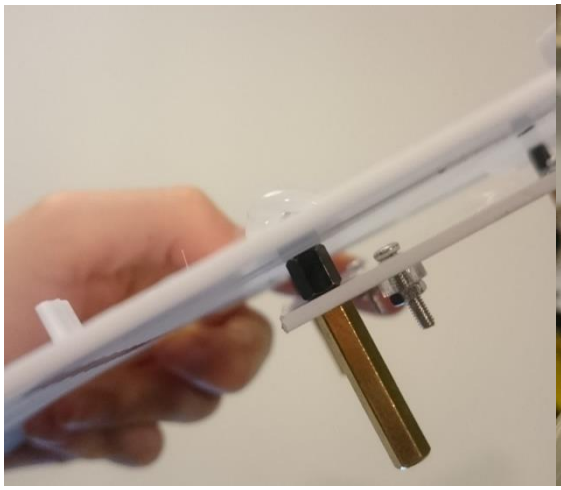
Kuva 16. Laatikko ilman muokkauksia.

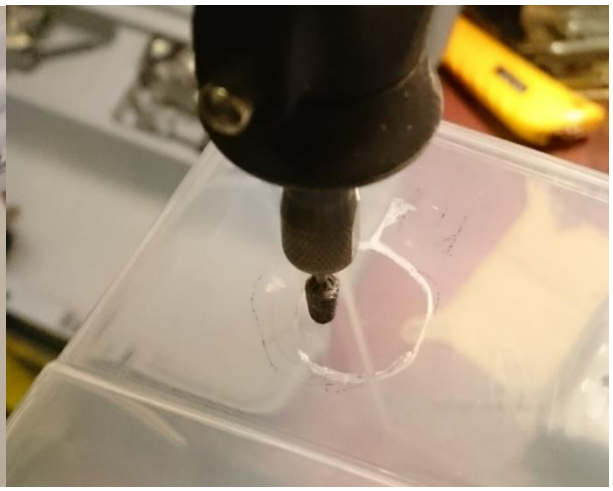




Kuvat 17 ja 18. Valmis pohja ja kiinnityspaikat.

Prototyypin laatikkoratkaisun rakennus alkoi Raspin kiinnityksen valmistuksella. Olimme hankkineet ”rack”-tyylisen avoimen kotelon raspille. Emme halunneet tuhota koulun omistamaa koteloa tai ruuveja, joten teimme kotelon pohjalevystä kopion käyttäen 2 mm paksuista polystyreenilevyä.





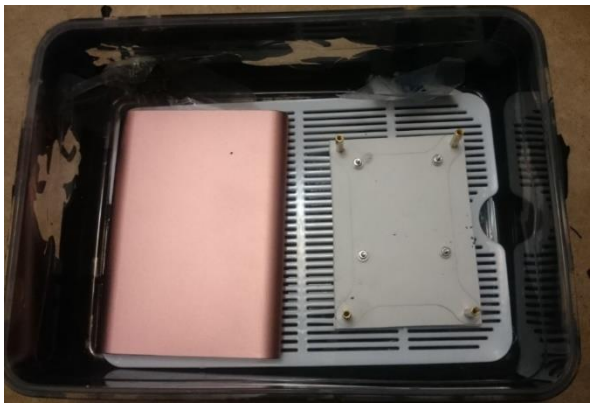
Kuvat 19 ja 20. Pohjan kiinnitys laatikon muoviritilään ja sensoriaukkojen teko.

Laatikkomme mukana tuli korotettu muoviritilä, jonka tarkoituksena on pitää säilöttävä ruoka irti itse laatikon pohjasta, näin välissä on ilmatilaa. Tämä levy toimi sikäli hyvin, että mahdollinen kondensoitunut vesi ei pääsisi osumaan laitteisiimme. Ritilä on kiinnitetty kuumaliimalla rasian pohjaan.

Molemmat sensoreistamme tarvitsivat aukot, jotta ne näkisivät laatikon ulkopuolelle. Itse reiät tehtiin ensin porakoneella ja muokattiin oikean kokoisiksi Dremelillä.

Kuvat 21 ja 22. Yllä PIR-sensorin sopivuuden testausta ja oikealla tuki liekkisensorille.

Tiesimme, että liekkisensori tarvitsi noin 45 asteen kulman havaitakseen liekin Nuuksiossa. Sensori on sen verran hauras, ettei sitä voinut vain liimata paikalleen, joten sensorille tehtiin sopiva tuki.




Kuvat 23 ja 24. Vasemmalla laatikon ensimmäinen valmis versio ja oikealla laatikon viimeisin versio.

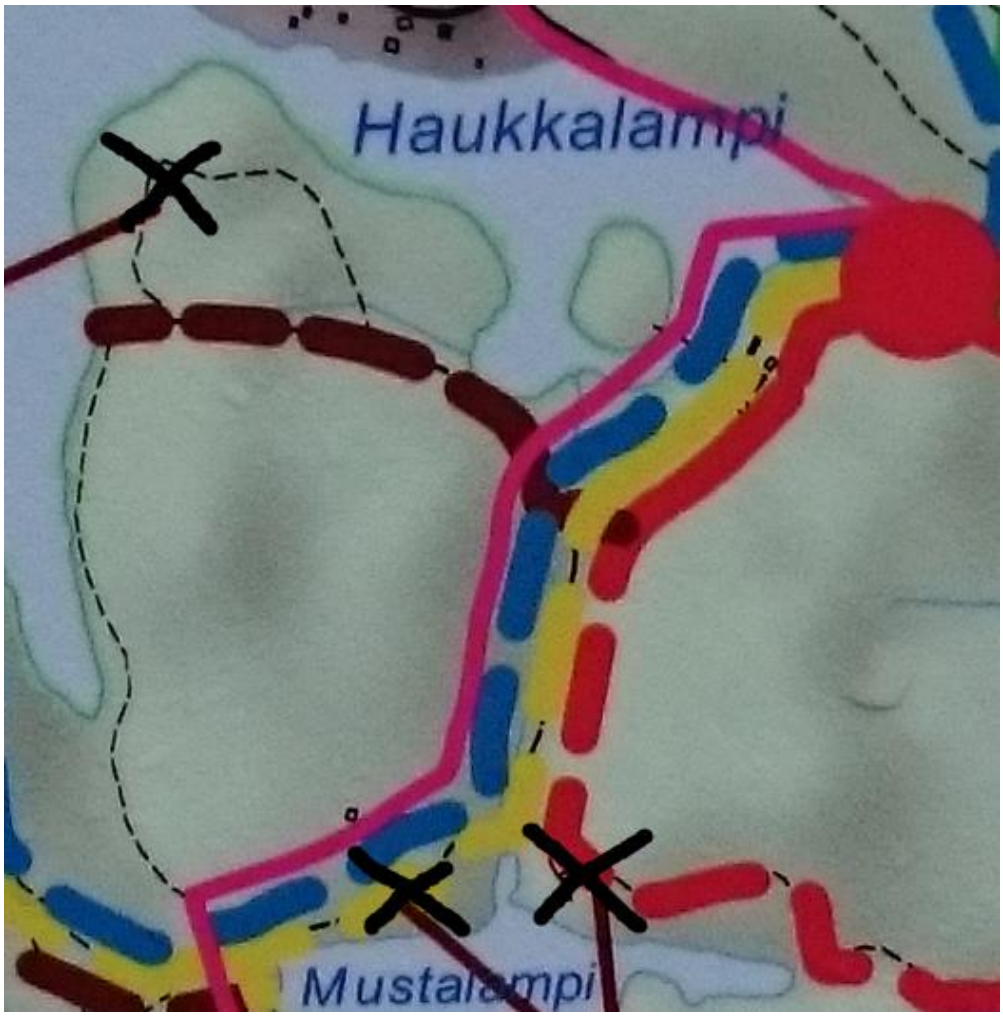
Lopuksi laatikko maalattiin mustaksi spraymaalilla ja lakattiin. Tosin hyvin nopeasti selvisi, että laatikon muovi hylkii maalia, vaikka sen hioisi. Laatikosta puuttui vielä kiinnityspalat, sillä tässä vaiheessa projektia emme vielä olleet saaneet Metsähallitukselta lupaa porata laatikkoa kiinni nuotiopaikan kattoparruun. Laatikkoon tuli muutamia muutoksia projektin aikana. Kiinnityspaikat vaihtuivat ja WLAN-mokkula tarkoitti, että itse raspin kiinnityslevyn paikkaa piti siirtää, jotta ylläoikean kuvan mukainen konfiguraatio olisi mahdollinen. Samalla spraymaali vaihdettiin mustaan ilmastointiteippiin.

2.9 Testipaikan valitseminen ja testauslupa

Projektin alussa oli hyvin tärkeää löytää mahdollisimman ajoissa sopiva testipaikka prototyypille. Vaatimuksina oli muun muassa:

- Katettu tulisija
- Hyvät kulkuyhteydet
- Sijainti, jossa paljon vierailijoita

Näiden vaatimusten pohjalta valikoitui kolme vaihtoehtoa: Haukkalampi, Mustalampi Länsi ja Mustalampi Itä.



Kuva 25. Mahdolliset testausvaihtoehdot kartalla.

Yllä kuvassa näkyy punaisella pallolla merkittynä parkkipaikka, josta lähti kolme eri reittiä. Ruskea johti suoraan Haukkalammella, keltainen Mustalampi Länteen ja punainen Mustalampi Itään, jokainen nuotiopaikka on kuvaan merkattu mustalla ruksilla. Etäisyys parkkialueesta on alle kilometri.

Kuvat 26 ja 27. Vasemmalla Haukkalampi ja oikealla Mustalampi Itä.

Haukkalammen kohde oli lähimpänä parkkialuetta, kyseessä on mökkimäinen levähdyspaikka, jossa on yksi umpinainen seinä. Tämä kohde karsiutui, sillä sopiva kiinnityspaikka prototyypille olisi ollut noin 5 metrin päässä tulisijasta. Mustalampi Itä oli hyvin samanlainen kuin Haukkalampi, erona parempi maisema ja tuulelta suojaisampi kohde.

Viimeisenä kohteena oli Mustalampi Länsi. Tämä kohde erosi muista paikoista fyysisesti erilaisella mökillä ja etuna oli myös, että prototyypin kiinnityspisteitä oli paljon. Nuotiopaikalla oli kiinnityskohdista lyhin etäisyys tulipesään, noin kaksi ja puoli metriä. Valitsimme tämän kohteen testipaikaksemme.

Kuva 28. Mustalampi Länsi valikoitui lopulliseksi testauspaikaksi.

Olimme yhteydessä Nuuksioon Haltian luontokeskuksen kautta, ja Haltiassa oltiin hyvin kiinnostuneita projektista. Sieltä saimme yhteystiedot lupahakemusta varten. Projektin lopussa Haltia ei valitettavasti vastannut yhteydenottoihin, jotta olisimme voineet esitellä heille paikan päällä prototyyppiä toiminnassa.

Otimme yhteyttä myös Metsähallitukseen ja saimme ohjeet tehdä virallisen lupahakemuksen projektia varten. Virtuaaliluonto-projekti jatkuu vielä pitkään meidän jälkeemme, joten pyysimme projektin virallista edustajaa, Ari Alamäkeä tekemään lupahakemuksen ja saimme luvat testaukseen välille marras-joulukuu. Valittavasti myöskään Metsähallituksesta ei kukaan ehtinyt tulla katsomaan projektimme esittelyä Nuuksiossa.

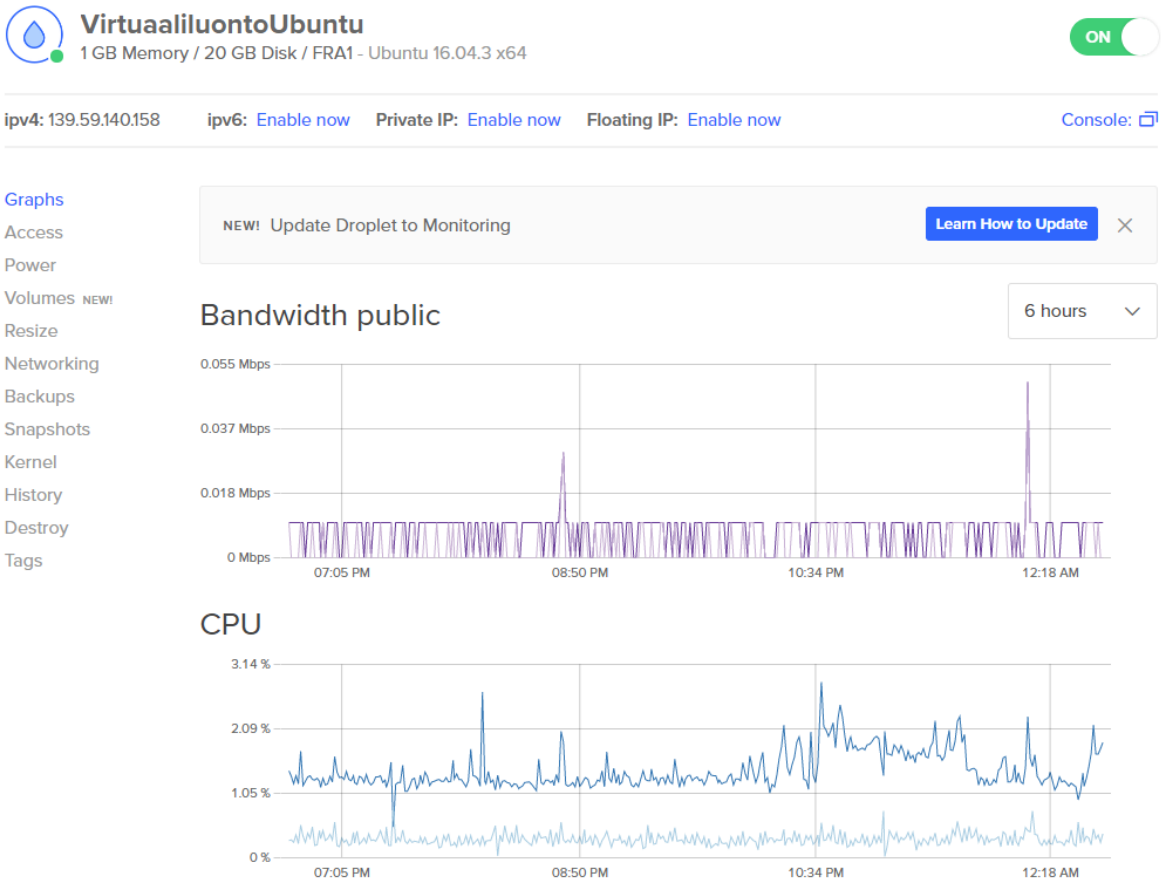
3 Palvelintietokone

3.1 Virtuaalipalvelimen konfigurointi

Prototyypin verkkosivuja ja tietokantaa datan tallennusta varten ylläpidettiin Ubuntu virtuaalipalvelimella, jonka hankimme palveluntarjoaja DigitalOceanilta. Kyseisen palveluntarjoajan edut ovat helppo ja nopea käyttöönotto sekä halvat hinnat. Aluksi tarkoituksena oli käyttää Haaga-Helian omia protopalvelimia, mutta ajatuksesta luovittiin niissä palvelimissa esiintyneistä ongelmista tietokantojen asennuksen ja konfiguroinnin kanssa.

Hankimme DigitalOceanista 10 euroa kuukaudessa (24/7 käytöllä) maksavan VPS:n seuraavilla tiedoilla:

- Nimi: VirtuaaliluontoUbuntu
- Sijainti: Frankfurt (FRA1)
- IPv4-osoite: 139.59.140.158
- Käyttöjärjestelmä: Ubuntu 16.04.3 x64
- Prosessoriytimiä: 1
- Keskusmuistia: 1 GB
- Tallennustilaa: 20 Gt

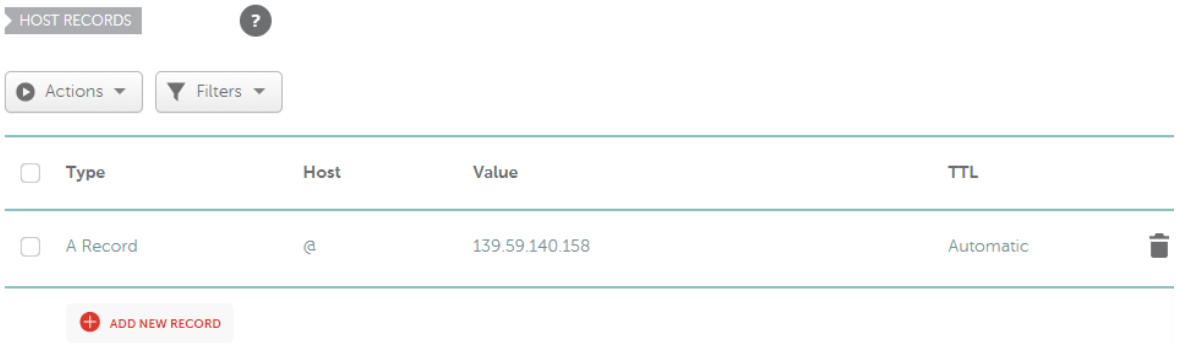


Kuva 29. DigitalOceanin dashboard-näkymä palvelimen hallintaan.

3.1.1 Domain-nimen osoittaminen palvelimeen

Jotta käyttäjät löytäisivät palvelumme helposti, tarvitsimme sille domain-nimen. Sen vuokrasimme NameCheapilta, joka tarjoaa aika-ajoin kohtuullisen edullisia nimiä. Nimi vuokrattiin vain yhdeksi vuodeksi, mutta se voidaan tarvittaessa uusia automaattisesti pitemmällekin ajalle, mikä tyypillisesti onkin kannattavaa, ettei oikeutta nimeen menetä vahingossa. Nimeksi valikoitui **nuotiovahti.info**, joka maksoi hankintahetkellä 1,99 euroa vuodessa.

Osoitimme hankkimamme Domain-nimen palvelimeen luomalla sille uuden A Recordin NameCheapin Advanced DNS-valikosta.



Kuva 30. A Record NameCheapin Advanced DNS-valikossa.

Kun uusi A record oli tehty, poistimme NameCheapin automaattisesti oletuksena luomat recordit. Nimen ohjauksessa kesti noin tunnin verran, minkä jälkeen palvelimelle ja verkkosivulle pääsi ostamamme domain-nimen kautta.

3.1.2 Alustavat toimet palvelimella

Ennen kuin aloitimme prototyypin vaatimien palvelinohjelmistojen asennuksen, tuli palvelimesta tehdä ensin lähtökohtaisesti tietoturvallinen. Ensimmäisen kerran kirjauduimme palvelimelle SSH-yhteydellä pilvipalvelutarjoajan antamin root-tunnuksin. Jatkossa halusimme kuitenkin käyttää omia käyttäjätunnuksia, joten root-tunnuksilla kirjautuminen estettiin tietoturvasyistä. Lisäksi palvelin tarvitsi palomuurin suodattamaan saapuvaa liikennettä.

Palomuurina käytimme Ubuntun oletuspalomuuria UFW, jolla sallimme liikenteen portteihin 22 (SSH), 80 (HTTP), 443 (HTTPS), 1883 (MQTT) ja 2222, jota käytimme prototyyppilaitteen etähallintaan, kun se oli mobiilidatan päässä.

UFW:n sääntölista oli siis seuraavanlainen:

```
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To                         Action    From
--                         ------    ----
22/tcp                     ALLOW IN  Anywhere

80/tcp                     ALLOW IN  Anywhere

443/tcp                    ALLOW IN  Anywhere
1883/tcp           ALLOW IN  Anywhere
2222/tcp           ALLOW IN  Anywhere

22/tcp (v6)                ALLOW IN  Anywhere (v6)

80/tcp (v6)                ALLOW IN  Anywhere (v6)

443/tcp (v6)               ALLOW IN  Anywhere (v6)
1883/tcp (v6)              ALLOW IN  Anywhere (v6)
2222/tcp (v6)              ALLOW IN  Anywhere (v6)
```

Seuraavaksi päivitimme Raspbian käyttöjärjestelmän pakettivarastot ja paketit uusimpiin versioihin. Kun palvelin oli suojattu palomuurilla ja paketit oli päivitetty, loimme kaksi uutta käyttäjää: ”nuotiovahti”-käyttäjälle ei annettu sudo-oikeuksia laisinkaan, sillä sitä tultiin käyttämään ainoastaan prototyypin websivujen hostaamiseen kyseisen käyttäjän kotihakemistosta. Nuotiovahdin lisäksi loimme toisen käyttäjän sudo ja adm-oikeuksin, jota käytettiin sitten palvelimen ohjelmistojen asennuksiin ja yleiseen ylläpitoon.

DigitalOceanin root-tunnuksille ei enää käytetty kirjautumiseen, joten ne lukittiin täysin. Sitä ennen testasimme tietenkin, että kirjautuminen luomillamme kahdella uudella käyttäjällä onnistui. Root-tunnukset suljettiin komennolla **sudo usermod --lock root**.

Komennon lisäksi root-kirjautuminen estettiin kokonaan muokkaamalla konfigurointitiedostoa **/etc/ssh/sshd_config**, jossa ”PermitRootLogin” arvoksi annettiin ”no”:

```
# Authentication:

LoginGraceTime 120
PermitRootLogin no
StrictModes yes
```

SSH-palvelu käynnistettiin uudelleen muutoksen jälkeen, ja root-kirjautuminen ei enää onnistunut.

3.2 Back-End

Projektin back-end on toteutettu LAMP (Linux, Apache, MySQL, Python/Python-Flask)-pinolla. Linux koneelle on asennettu Apache, joka on konfiguroitu pyörittämään Python Flask-kehyksellä toteutettuja verkkosivuja. Tietokannasta haetaan dataa Python Flaskilla toteutetun REST API:n kautta. REST on siis HTTP-protokollaan perustuva arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen ja se yleisin webkehitysstandardi. Tietokantaan viedään dataa erillisellä Paho-MQTT Python-skriptillä joka vastaanottaa MQTT viestejä ja tallentaa ne kantaan.

3.2.1 Apache

Apache-webpalvelinohjelmistolla ajettiin prototyypin verkkosivuja, jonne sensorien keräämä tieto sitten luetaan. Apache asennettiin normaalisti paketinhallinnan kautta. Koska ajoimme verkkosivuja käyttäjän omassa kotihakemistossa, otimme Apachen **userdir-modulin** käyttöön. Lisäksi otimme käyttöön Apachen **wsgi-moduulin** käyttöön, jotta pystyimme liittämään Python Flask -kehyksen siihen. Otimme **/etc/apache2/sites-available/000-default.conf** tiedoston pois käytöstä ja loimme **nuotiovahti.conf** sen tilalle, jossa asetimme palvelun oletushakemistoksi käyttäjän kotihakemistossa olevan public_html/nuotiovahti kansion.

Alla on nuotiovahti.conf sisältö:

```
<VirtualHost *:80>
        ServerName mywebsite.com
        ServerAdmin admin@mywebsite.com
        WSGIScriptAlias / /home/mint/pub-
lic_html/nuotiovahti/nuotiovahti.wsgi
        <Directory /home/mint/public_html/nuotiovahti/nuotiovahti/>
            Order allow,deny
            Allow from all
        </Directory>
        Alias /static /home/mint/pub-
lic_html/nuotiovahti/nuotiovahti/static
        <Directory /home/mint/pub-
lic_html/nuotiovahti/nuotiovahti/static/>
            Order allow,deny
            Allow from all
        </Directory>
        ErrorLog ${APACHE_LOG_DIR}/error.log
        LogLevel warn
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

3.2.2 Mosquitto-broker ja Paho-MQTT

Käytimme Mosquitto-nimistä demonia, joka toimii MQTT-brokerina vastaanottaen ja välittäen sensorien lähettämää dataa. Mosquitto asennettiin normaalisti pakettienhallinnasta ja se alkoi pyörimään heti palvelimen portissa 1883. Asensimme myös mosquitto-clients-paketin testausta varten, koska sillä pystyi helposti lähettämään ja kuuntelemaan brokerin MQTT-viestejä.

Viestien vastaanottamiseen ja tallentamiseen tietokantaan käytimme Paho-MQTT Python-pakettia, joka on Pythonille tehty MQTT-client. Paho-MQTT-paketin asennuksessa käytimme **Python-pip**-työkalua, joka on tarkoitettu Python-pakettien hallintaan ja asentamiseen. Python-pip asennettiin normaalisti paketinhallinnan kautta. Pip toimii samalla tavalla kuin tavallinen paketinhallintatyökalu (esim. apt-get), mutta vain pythonin paketeille. Pip:n asennuksen jälkeen se tarvitsi ensimmäisenä päivittää.

```
$ sudo pip install --upgrade pip

#Useat paketit vaativat myös setuptools paketin asennuksen, ennen
kuin niitä pystyy asentamaan
$ sudo pip install setuptools

#Kun setuptools oli asennettu, pystyimme asentamaan paho-mqtt-pa-
ketin.
$ sudo pip install paho-mqtt

#Tarvitsimme myös mysql-connector paketin, jotta pystyimme yhdis-
tämään mysql tietokantaan. Käytimme mysql-connector versiota
2.1.4.
$ sudo pip install mysql-connector==2.1.4
```

Loimme uuden python skriptin **mysql_mqtt.py** internetohjeiden pohjalta (EV3Dev 2016). Taitojen ja ajan puutteen takia emme liittäneet skriptiä Python Flask -applikaatioomme vaan se jäi erilliseksi kokonaisuudeksi, joka pitää laittaa päälle erikseen. Käymme koodia läpi kohta kohdalta alapuolella, koodi löytyy myös GitHubista.

Tuomme ensin tarvitsemamme paketit Python-skriptiimme:

```
import paho.mqtt.client as mqtt
import mysql.connector
from mysql.connector import errorcode
```

Määrittelimme ensimmäisenä **addtobase** funktion, joka ottaa vastaan topic muuttujan. Funktio avaa tietokantayhteyden hakemalla tietokannan login-tiedot erillisestä **connectors.cnf** tiedostosta, ja alustaa cursor-objektin kun yhteys on luotu.

```
def addtobase(topic):
          #connecting to the database
          try:
                    cnx = mysql.connector.connect(op-
tion_files='connector.cnf')
          except mysql.connector.Error as err:
                    if err.errno == errorcode.ER_ACCESS_DENIED_ER-
ROR:
                              print("Something is wrong with your
user name or password")
                    elif err.errno == errorcode.ER_BAD_DB_ERROR:
                              print("Database does not exist")
                    else:
                              print(err)

          cursor = cnx.cursor()
```

Tarkistamme if-lauseella metodin vastaanottaman muuttujan topic (joko “pir” tai “flame”) ja valitsemme sen perusteella, kumpaan tietokannan tauluun lisäämme rivin (Pir tai Flame taulu, kts. kohta 3.2.4 MySQL).

```
add_sensordata = ‘’
          if topic == "Pir":
                    add_sensordata = ("INSERT INTO Pir(pir_id)
VALUES(1)")
          elif topic == "Flame":
                    add_sensordata = ("INSERT INTO Flame(flame_id)
VALUES(1)")
```

Funktion lopussa toteutamme ja committaamme itse lisäyksen kantaan ja suljemme cursor-objektin ja yhteyden.

```
          cursor.execute(add_sensordata)
          cnx.commit()
          cursor.close()
          cnx.close()
```

Koodissa määrittelemme Paho-MQTT:n mukana tulleen **on_connect** funktion, joka suoritetaan, kun skriptimme yhdistää mosquitto-brokeriimme. Funktio tulostaa terminaaliin yhteyden muodostuttua tuloskoodin, ja tilaa brokerilta topicilla **nuotiovahti/#** tulevat viestit, jotta se voi vastaanottaa ja käsitellä niitä.

```
def on_connect(client, userdata, flags, rc):
          print("Connected with result code "+str(rc))
          #subscribes to the broker topic nuotiovahti/# so it can
receive both pir and flame sensor messages
          client.subscribe("nuotiovahti/#")
```

Toinen Paho-MQTT:n mukana tullut funktio, jonka määrittelimme, on ”**on_message**”. Tämä funktio suoritetaan silloin kun skriptimme vastaanottaa brokerilta MQTT-viestin. Funktio tarkistaa if-lauseella viestin sisällön ja kutsuu sitten yllä mainittua **addtobase** funktiota. Jos viestin sisältö on numero **1**, funktio kutsuu addtobase-funktiota muuttujalla **“pir”**, ja jos viestin sisältö on numero **2** se taas kutsuu funktiota muuttujalla **“flame”**. Jos viestin sisältö ei ole 1 tai 2, viesti hylätään.

```
def on_message(client, userdata, msg):
          #check the message and set the topic for which table to
add the data and then calls the add to base function.
          # if message contains something else prints the message
instead
          if msg.payload == "1":
                    addtobase("Pir")
          elif msg.payload == "2":
                    addtobase("Flame")
```

Alustamme Paho-MQTT kirjastosta saatavan client-objektin ja annamme sille Mosquitto-brokerimme osoitteen, portin ja keep alive-ajan.

```
client = mqtt.Client()
client.connect("127.0.0.1",1883,60)
```

Kerromme skriptille, mitä funktiota kutsua, kun yhteys muodostetaan tai viesti vastaanotetaan.

```
client.on_connect = on_connect
client.on_message = on_message
```

Lopuksi asetamme Paho-MQTT-clientin pyörimään niin kauan kuin itse skripti on pyörimässä:

```
client.loop_forever()
```

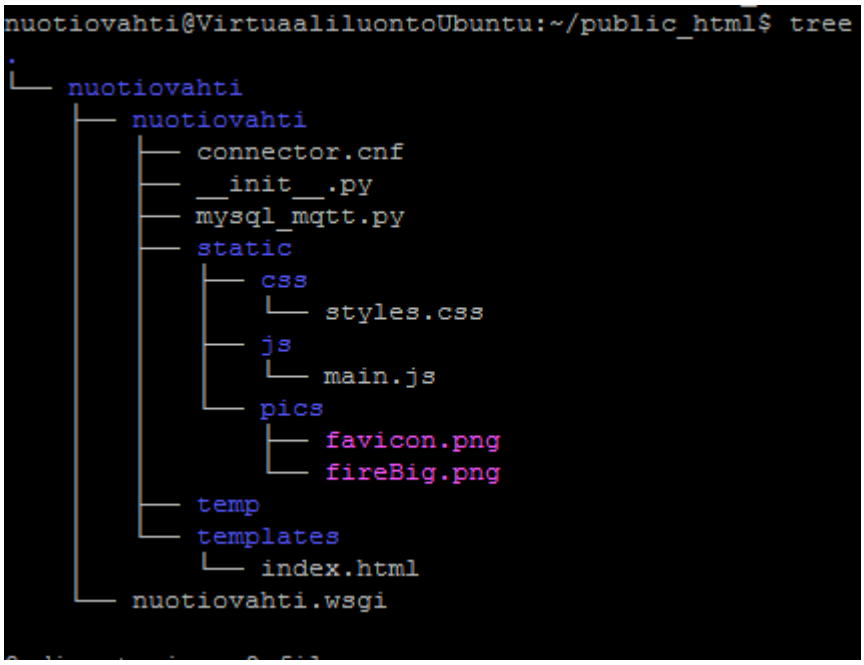
3.2.3 Python Flask

Sensorien lähettämän datan käsittelyyn ja lukemiseen tietokannasta verkkosivuille käytimme Python Flaskia, joka on python pohjainen ohjelmistokehys webkehitykseen. Flaskillä on toteutettu REST periaatteen mukainen rajapinta, jonka kautta palvelumme toimii. Palvelulle on määritelty kaksi toimivaa URLia: **nuotiovahti.info** ja **nuotiovahti.info/data**. Nuotiovahti.info on palvelumme kotisivu, ja nuotiovahti.info/data palauttaa vastauksena GET pyyntöön viimeisen viiden minuutin datan tietokannasta.

Flask paketin asennus tapahtui pip työkalulla samalla tavalla kuin edellisten python pakettien asennus, eli **sudo pip install flask**.
Flaskin paketti oli asennettu ja loimme web applikaatiomme käyttäen DigitalOceanin Flask-tutorialia (DigitalOcean 2013) Huomioi, että emme käyttäneet ohjeessa mainittua **virtualenv**-ympäristöä. Jotta Python Flask toimisi Apachen wsgi-moduulin kanssa, teimme **nuotiovahti.wsgi** tiedoston edellä mainittuun public_html/nuotiovahti tiedostoon. Tiedosto liittää Flask-applikaatiomme Apachen wsgi-moduulin kautta Apachen palvelimelle. Alla nuotiovahti.wsgi:

```
#!/usr/bin/python
import sys
import logging
logging.basicConfig(stream=sys.stderr)
sys.path.insert(0,"/home/nuotiovahti/public_html/nuotiovahti/")

from nuotiovahti import app as application
application.secret_key = 'Add your secret key'
```

Flask-applikaatio tarvitsi oikeanlaisen tiedostorakenteen, jonka teimme nuotiovahti-käyttäjän kotihakemistoon public_html/nuotiovahti -kansion alle. Seuraavassa kuvassa näkyy tiedostorakenne applikaatiolle.



Kuva 31. Flask-applikaation tiedostorakenne.

Käyttäjälle näkyvät verkkosivut asetettiin templates-kansioon. JavaScript, CSS sekä kuvatiedostot löytyvät omista kansioistaan static-kansion alta. Itse Flask-applikaation koodi on __init__.py tiedostossa. Koodissa käytetään pääasiassa normaalia Pythonin syntaksia, muutamia Flaskin kehykseen vaadittavia kohtia lukuun ottamatta. Alla käymme __init__.py koodin läpi kohta kohdalta. Kaikki koodi löytyy myös GitHubista.

Tuomme Flask-paketista applikaatiomme tarvitsemat toiminnot, jotta pystymme tarjoamaan sivun käyttäjälle ja hakemaan kannasta dataa sivulle.

```
from flask import Flask, render_template, jsonify, json, request
```

Määrittelemme kotisivun URLin. Kun käyttäjä selaa nuotiovahti.info osoitteeseen, Flask etsii **@app.route("/")** alle määritellyn funktion ja palauttaa funktiossa määritellyn palautusarvon. tässä tapauksessa se palauttaa applikaation kotisivun käyttäjälle.

```
#home page
@app.route("/")
def main():
          return render_template('index.html')
```

Nuotiovahti.info/data urlille määriteltiin funktio **getData** reitin **@app.route('/data')** alle. Käytimme EnvatoTuts-sivuston ohjetta getData-funktion luomisessa (EnvatoTuts 2015).

Funktio suorittaa tietokantahaun MySQL-tietokantaan ja palauttaa summan viimeisen viiden minuutin aikana kantaan lisätyistä sensoridatan tietueista. Koska funktio käyttää mysql-connector pakettia, piti se tuoda ensin ohjelmaan:

```
import mysql.connector
from mysql.connector import errorcode
```

Reitti ja funktion nimi:

```
@app.route('/data')
def getData():
```

Funktio getData ottaa ensiksi yhteyden tietokantaan käyttäen mysql-connector pakettia:

```
try:
          cnx = mysql.connector.connect(user='nuotiovahti', pass-
word='passwd', host='                  127.0.0.1', data-
base='nuotiovahti')
          except mysql.connector.Error as err:
                    if err.errno == errorcode.ER_ACCESS_DENIED_ER-
ROR:
                              print("Something is wrong with your
user name or password")
                    elif err.errno == errorcode.ER_BAD_DB_ERROR:
                              print("Database does not exist")
                    else:
                              print(err)
```

Luomme cursor objektin, joka on osa mysql-connector pakettia. Cursor-objektilla toteutamme tietokantahaun ja tallennamme haun tulokset. Alla oleva koodi alustaa cursor-objektin ja suorittaa haun kannasta.

```
          cursor = cnx.cursor()

          cursor.execute("select (select count(f.detection_time)
from Flame as f where detection_time >now()- INTERVAL 300 SECOND)
as flame, (select count(p.detection_time) from Pir as p where de-
tection_time >now()- INTERVAL 300 SECOND) as pir;")
```

Python- ja MySQL-osaamisemme oli hieman vajaata, joten emme ole varmoja, kuinka optimoitu yllä oleva tietokantahaku on. Myös haun tulosten muuntamisessa haluttuun JSON-muotoon oli ongelmia, kun käytimme Flaskin mukana tulleita metodeja. Lopulta jouduimme muokkaamaan haun tuloksia “manuaalisesti” ennen tulosten muuntamista JSON-muotoon Flaskin jsonify-metodilla.

Haun tulokset on tallennettu edellä mainittuun cursor-objektiin, josta haetaan hausta saatujen kolumnien nimet (pir, flame) listaan nimeltä columns. Sensoridatan summien tulokset haetaan ensin listaan datarows, ja sitten listan ensimmäinen arvopari (datarows-listan ensimmäinen arvo sisältää summien tuloksen arvoparina) tallennetaan toiseen listaan nimeltä datalist.

Columns- ja datalist-listat yhdistetään dicta-nimiseen dictionary-tietotyyppiin avainarvoparein, siten että kumpikin summa liitetään yhteen sen pylvään nimen kanssa. Käytämme Flaskin mukana tullutta jsonify-metodia muuttamaan dicta-rakenteen JSON-muotoon, joka tallennetaan rows-muuttujaan. Suljemme cursorin ja tietokantayhteyden ja palautamme rows-muuttujan vastauksena sivun pyyntöön.

```
#getting colum names from the resultset
          columns = cursor.column_names
          #getting the counts fro mthe resultset
          datarows = cursor.fetchall()
          #get the values from the rows
          datalist = datarows[0]
          #combine the colum names and values into a dict

          dicta = dict(zip(columns, datalist))

          rows = jsonify(dicta)
          cursor.close()
          cnx.close()

          return rows
```

3.2.4 MySQL

Sensorien lähettämä data tallennettiin MySQL-tietokantaan, joka asennettiin paketinhallinnan kautta. Asennetut paketit olivat ”mysql-server” ja ”mysql-client”. Taitomme eivät aivan riittäneet MySQL-kannan luomiseksi sillä tavalla, että se skaalautuisi niin hyvin kuin olisimme toivoneet. Tämänhetkinen ratkaisumme toimii vain yhdelle laitteelle. Jos laitteita on useampia, tarvitsee tietokanta suunnitella uudelleen.

MySQL-palvelimen aikavyöhyke piti vaihtaa Suomen aikaan, jotta sensoridatan aika saatiin lisättyä oikein tauluihin.

```
SET time_zone = "+02:00";
```

Loimme uuden tietokannan nimeltä **nuotiovahti** ja annoimme oikeudet kyseiseen tietokantaan käyttäjälle nuotiovahti. Kantaan loimme taulut **Pir** ja **Flame**.

```
CREATE TABLE Pir(pir_id int, detection_time TIMESTAMP NOT NULL DEFAULT NOW());

CREATE TABLE Flame(Flame_id int, detection_time TIMESTAMP NOT NULL DEFAULT NOW());
```

```
+----------------------+
| Tables_in_nuotiovahti |
+----------------------+
| Flame                |
| Pir                  |
+----------------------+
```

Kuva 32. Tietokannan taulut.

```
mysql> describe Pir;
+----------------+-----------+------+-----+-------------------+-------+
| Field          | Type      | Null | Key | Default           | Extra |
+----------------+-----------+------+-----+-------------------+-------+
| pir_id         | int(11)   | YES  |     | NULL              |       |
| detection_time | timestamp | NO   |     | CURRENT_TIMESTAMP |       |
+----------------+-----------+------+-----+-------------------+-------+
```

```
mysql> describe Flame;
+----------------+-----------+------+-----+-------------------+-------+
| Field          | Type      | Null | Key | Default           | Extra |
+----------------+-----------+------+-----+-------------------+-------+
| Flame_id       | int(11)   | YES  |     | NULL              |       |
| detection_time | timestamp | NO   |     | CURRENT_TIMESTAMP |       |
+----------------+-----------+------+-----+-------------------+-------+
```

Kuvat 33 ja 34. Kummassakin taulussa on kaksi kenttää: id ja detection_time. Kun kantaan lisätään sensoridataa, tietokanta laskee detection_time-kenttään automaattisesti ajan, jolloin data lisättiin.

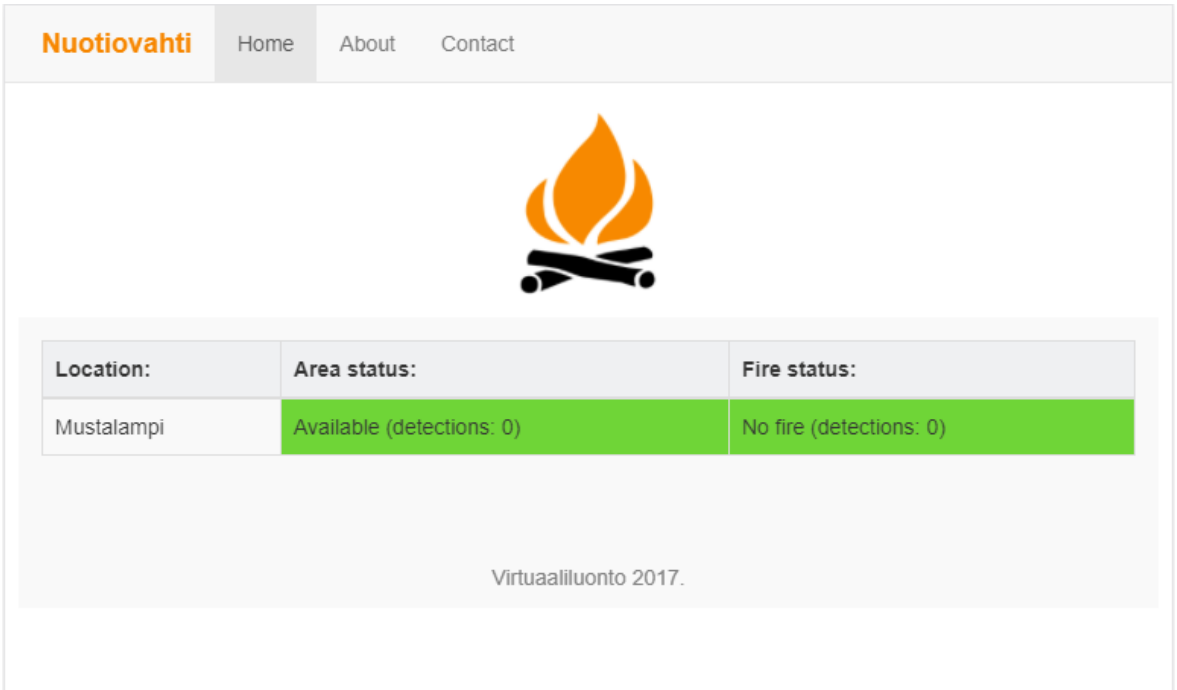
3.3 Front-End

Kehittäessämme verkkosivuja palveluprototyyppiä varten, otimme huomioon sen, että enemmistö käyttäjistä tulisi olemaan matkailijoita mobiililaitteiden kanssa. Näin ollen halusimme sivuista responsiiviset ja sopivat kaiken kokoisille laitteille, mikä olikin helppoa Bootstrap- ja JQuery-kirjastoja käyttämällä. Index.html pitää sisällään HTML-koodin ja CSS löytyy styles.css-tiedostosta. Kaikki HTML-sivustolla käytetyt kirjastot:

```
https://max-
cdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css
https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js
https://code.jquery.com/ui/1.12.1/jquery-ui.js
https://max-
cdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js
```

Projektin skaalaan ei kuitenkaan kuulunut hienostuneiden sivustojen teko tai mobiilisovellus. Jätimme sellaiset tavoitteet mahdolliselle jatkokehitykselle digitaalisia palveluita tuottaville opiskelijoille. Halusimme kuitenkin saada sensoridatan verkkosivulle asti. Se teki prototyypin demoamisesta huomattavasti helpompaa ja avusti samalla projektiryhmää testaustilanteissa.

Sivun visuaalinen puoli on projektiryhmän omaa CSS:ää. Sivun rakenteen ja käytettävyyden saavuttamisessa hyödynnettiin yllämainittuja valmiita kirjastoja.



Kuva 35. Sivun vaatimaton, mutta responsiivinen ulkoasu.

Palvelimen back-end tarjoilee tiedot sensorien havainnoista front-endille **main.js**-tiedoston pyynnöstä. Verkkosivulla tehdään GET-pyyntö AJAX:illa (Asynchronous JavaScript and XML) viiden sekunnin välein, jolla haetaan Python Flaskilta sensorihavaintojen määrä JSON-muodossa. Näin käyttäjän ei tarvitse ladata verkkosivua joka kerta uudelleen, vaan sivuston taulukot päivittävät niiden sisällön viiden sekunnin välein livenä.

Kuten aina, koodia kannattaa mieluummin selata GitHubissa. Emme käy tässä raportissa läpi koko koodia kohta kohdalta, vaan puhumme ainoastaan oleellisimmista kohdista. Alla tehdään uusi XMLHttpRequest (AJAX)-pyyntö, joka taasen tekee GET-pyynnön Python Flaskille, joka tarjoilee JavaScriptille dataa JSON-muodossa.

```
function getData () {

          // Request & receive JSON Data
          var pageRequest = new XMLHttpRequest();
          pageRequest.open('GET', '/data');
          pageRequest.onload = function() {

                    // Save JSON data to a variable
                    var mySensorData = JSON.parse(pageRequest.re-
sponseText);
                    // Call the renderTest function and pass it to
mySensorData variable
                    renderData(mySensorData);
          };

// Send the request
pageRequest.send();

} // Function ends here

          // Call function before setting an interval
          getData();
          // Set Interval. Last argument is in milliseconds NOTE:
setInterval() keeps triggering expression again and again unless
you tell it to stop
          setInterval( getData, 5000 );
```

JSON-data pitää sisällään sekä PIR-sensorin että liekkisensorin tuottamien rivien lukumäärän MySQL-tietokantaan viimeisen viiden minuutin ajalta. Koska dataa talletetaan kantaan ainoastaan, kun havainnot ovat positiivisia (liikettä tai tulta havaittu), niin pystyimme JavaScriptillä asettamaan tässä havaintorajan, joka sensorien tulee ylittää. Tarkoituksena on siis toteuttaa tietynlaista virheentarkistusta. Emme halua, että sivu näyttää nuotiopaikan varatuksi yhden tai muutaman liikehavainnon perusteella.

Testatessamme prototyyppiä paikan päällä Nuuksiossa, tulimme siihen tulokseen, että PIR-sensorin pitäisi ylittää 25 havaintoa, ennen kuin paikka merkitään verkossa varatuksi.

Liekkisensori ei ole yhtä altis virheille, joten sen havaintoraja asetettiin melko matalaksi viiden havainnon paikkeille.

```
// If the amount of rows from the PIR sensor is higher than value,
then use the HMTL class "AreaStatusYes"
if (data.pir > 25) {
          pirhtml = "<td class='AreaStatusYes'>";
}

// If the amount of rows from the flame sensor is higher than
value, then use the HMTL class "AreaStatusYes"
if (data.flame > 5) {

          flamehtml = "<td class='FlameStatusYes'>";
}
```

Muuttelemme siis taulukoiden CSS-classejä oletuksesta, jos if-lauseen mukainen ehto täyttyy. Tuolloin oletusväri vihreä vaihtuu punaiseksi taulukon sarakkeiden kohdalla, jotka vastasivat if-lauseen ehtoa.

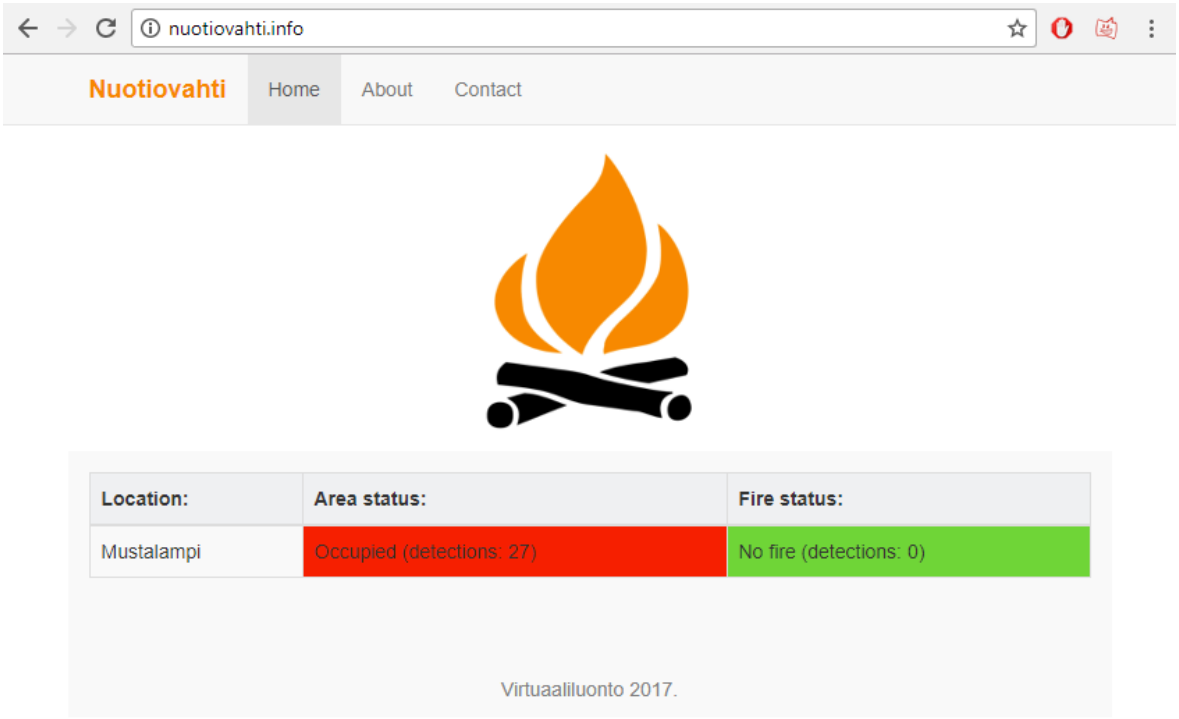
Elementit lisättiin htmlStringiin,

```
// Add elements to htmlString

htmlString += "<tr><td>" + "Mustalampi" + "</td>"+ pirhtml +
data.pir + "</td>" + flamehtml + data.flame + "</td></tr>";
```

jonka jälkeen htmlString lisättiin sisältönä HTML-sivulle. Sarakkeiden tekstit vaihdettiin jQueryllä asettamalla classeille tekstisisältö .text-metodilla:

```
// Add htmlString as content to HTML
$("#datatable tbody").html(htmlString);

          // Text values for the classes
          $( ".AreaStatusNo" ).text("Available (detections: " +
data.pir + ")");

          $( ".AreaStatusYes" ).text("Occupied (detections: " +
data.pir + ")");

          $( ".FlameStatusNo" ).text("No fire (detections: " +
data.flame + ")");

          $( ".FlameStatusYes" ).text("Burning (detections: " +
data.flame + ")");
```

Esimerkiksi, jos PIR-sensori on havainnut liikettä yli 25 kertaa viimeisen viiden minuutin sisään, mutta liekkisensori ei ole havainnut yhtään tulta:



Kuva 36. Sivusto ilmoittaa Mustalammen nuotiopaikan varatuksi. Tulta ei kuitenkaan havaita.

Punainen tarkoittaa varattua tilaa tai havaittua tulta. Vihreä tarkoittaa vapaata tai nuotiotonta nuotiopaikkaa. Havaintojen määrä viimeisen viiden minuutin sisään (detections) ei oikeastaan tarjoa loppukäyttäjälle mitään, vaan se lisättiin sivulle pikemminkin projektiryhmän testaustyötä helpottamaan.

4 Ulkoilmatestaus

Prototyyppiä oli läpi projektin testailtu aina tietyllä osa-alueella tekemisen lomassa. Tarvitsimme kuitenkin kattavammat testit siitä, miten sensorilaite suoriutuu kylmässä ja sateisessa säässä ulkona. Raspberry Pi:tä ei ole suunniteltu ulko-olosuhteisiin, ja olimmekin aluksi skeptisiä siitä, kuinka hyvin se tulee selviämään Suomen talvisäästä.

Ratkaistaviksi ongelmiksi muodostui mm. vedenkestävyys sekä kosteuden ja lämmön eristys. Elektroniikkalaitteet operoivat aina tietyssä lämpötilassa, ja kylmyys vaikuttaa erityisesti akkujen toimintaan. Sensorilaitteen laatikon pitäisi olla melko vedenkestävä, mutta kosteusvaurioita vastaan hankimme kolme silikageelipussia, jotka sitovat kosteutta ilmasta. Pussit voidaan uudelleen käyttää lämmittämällä niitä kaksi tuntia 120 °C:ssa. Lämpöä pyrittiin eristämään täyttämällä laatikko styroksisella pakkaustäytteellä. Vaikka lämpöeristyksemme ei olekaan kestävä ja käytännöllinen tuotantoratkaisussa, niin se soveltui kuitenkin tähän prototyyppivaiheeseen hyvin.

Testauspaikaksi valittiin Nuuksion kansallispuisto ja tarkemmin toinen Mustalammen nuotiopaikoista. Nuuksio valittiin sen läheisyyden ja aktiivisen käytön vuoksi, mikä helpotti projektin logistiikkaa sekä antoi meille oikeaa dataa kohteen käytöstä. Mustalammen nuotiopaikka taasen oli kaikista perinteisin vaihtoehto, ja etäisyys nuotiosta sensorilaitteen oletettavaan asennuspaikkaan oli suotuisin. Jouduimme hakemaan virallisen luvan Metsähallitukselta testauksen suorittamiseksi, joka sisälsi mm. sensorilaitteen poraamisen nuotiopaikan kattopaalun päälle. Lupa saatiin marraskuun alusta joulukuun loppuun.

4.1 Parveketesti

Ensimmäinen ulkoilmassa toteutettava testi tehtiin yhden projektiryhmäläisen kodin parvekkeella. Tavoitteena oli lähinnä nähdä paljonko kylmempi sää vaikuttaa litium-ioni akun akunkestoon. Emme myöskään vielä yhtään tienneet, miten Raspberry itse selviää ulkoisemmassa säässä.

Akkuna käytimme pienempää 2600 mAh varavirtalähdettä. Alustavasti testasimme myös liekkisensorin kykyä havaita kynttilä, mikä onnistuikin hyvin. Sensorilaite jätettiin parvekkeelle muiden oppituntien ajaksi, jolloin sensoridataa seurattiin MyMQTT-nimisellä mobiiliclientilla. Lopulta sensorilaite kesti 6h 11min. Tuolloin ainoastaan Raspberry Pi oli 2600 mAh akussa kiinni. Huawein LTE-modeemi käytti omaa 3000 mAh akkuaan. Testin perusteella päättelimme, että laite tulee kestämään parhaimmillaan noin 40 tuntia sillä suuremmalla 30000 mAh akulla. Arvio oli kuitenkin optimistinen.

4.2 Nuuksiotesti 1

Kuva 37. Sensoriskriptejä muokattiin paikan päällä ja MQTT-dataa seurattiin palvelimella.

Ensimmäinen Nuuksion ulkoilmatesti 24.11. oli lähinnä tarkoitettu selvittämään, miten hyvin prototyyppimme toimii sen suunnitellussa ympäristössä. Samalla testattiin kotelon ulkoilmakelpoisuutta ja sitä, että havaitseeko liekkisensorimme tulta suunnitellulta etäisyydeltä (n. 2,5 metriä). Muutenkin halusimme tietää, kuinka pahasti nuotiopaikan hormi estää sensorien näkyvyyttä.

Käytimme ensimmäisessä testissä lopullista pienempää 2600 mAh akkua. Tavoitteena ei vielä ollut nähdä, kauanko laite kestää kylmässä säässä kyseisellä, vaan ylipäätään testata, miten hyvin ulkoilmatestaus suunnitellulla paikalla luonnistuu. Monitoroimme nuotiovahti.info-sivustoamme ja sensorien lähettämää dataa kannettavalla tietokoneella ja MQTT-mobiiliclienteilla. Testi alkoi kello 10.00 ja päättyi kello 14.00.

Testauksen aikana palvelimemme kaatui pariin otteeseen. Tällöin ilmeni käänteisen SSH-tunnelin automaatioskriptimme ongelma: palvelimen muisti täyttyi turhista sshd-prosesseista. Väliaikainen ratkaisu oli vähentää SSH-tunnelin yhteydenottoja, sekä hankkia palvelimellemme enemmän muistia. Etähallintaa sittemmin kehitettiin, kuten raportin kohdasta 2.6.1. ilmenee. Sensorit toimivat kuitenkin luotettavasti, tosin PIR-sensori havaitsi myös infrapunasensorina nuotion liikkeen. Emme kuitenkaan kokeneet sitä ongelmaksi, sillä jos paikalla on tulta, voitaneen olettaa siellä olevan myös ihmisiä. Kaiken kaikkiaan, ensimmäinen testi oli onnistunut.

4.3 Nuuksiotesti 2

Toinen ulkoilmatesti 27.11. oli pääosin samanlainen kuin ensimmäinen, asetettiin sensorilaite paikalleen ja valvottiin nettisivulle tulevaa dataa. Tavoitteena oli tuolloin kuitenkin enemmän optimoida se raja nettisivulla, joka sensorihavaintojen tulee ylittää. Mittasimme siis, paljonko liikettä saimme PIR-sensorin havaitsemaan viiden minuutin sisään. Siinä tuli ottaa huomioon, että ihmisten käyttäytyminen vaihtelee huomattavasti. Esimerkiksi nuotiontekovaiheessa liikettä on huomattavasti enemmän verrattuna makkaroiden paistamiseen. Näin ollen, raja tuli asettaa sopivan alas, jotta se kattaa eri tilanteita monipuolisesti, mutta samalla ei ilmoita paikkaa väärin perustein varatuksi, jos paikalle sattuu hetkellisesti jokin luontoeläin.

Kuva 38. Kipinät ja hiillos eivät enää saaneet liekkisensoria havaitsemaan tulta.

Ensimmäisen testin ongelmat oli korjattu tai vähintäänkin minimoitu toisen testin ajaksi. Käytimme edelleen pienempää akkua, mutta tällä kertaa annoimme sen kulua loppuun. Säädimme samalla nettisivun datan raja-arvoja sensorihavaintojen perusteella. PIR-sensorin rajaksi asetimme 25 havaintoa viiden minuutin sisään, jonka jälkeen nettisivu ilmoittaa paikan varatuksi. Vastaavaksi arvoksi liekkisensorille annettiin 5. Liekkisensori ei ole yhtä altis virheille kuin PIR-sensori, ja sen vuoksi raja laitettiin melko alas. Liekkisensorin kyky havaita tulta rajoittuu toki nuotion kokoon. Esimerkiksi yllä olevan kuvan kokoista hiillosta ja kipinää sensori ei pysty havaitsemaan.

Testi aloitettiin kello 9.20, ja päätettiin akun loputtua 15.37.

4.4 Nuuksiotesti 3

Kolmas ja viimeinen testi Nuuksiossa aloitettiin perjantaina kello 10.20. Tavoitteena oli nähdä lopullinen virrankulutus ja akun kesto koko kokoonpanolla. Projektiryhmä valvoi laitteen toimintaa vähän yli neljän tunnin ajan, jonka jälkeen laite jätettiin Mustalammen nuotiopaikalle. Jopa projektin ohjaaja kävi tuolloin paikan päällä katsastamassa prototyyppimme toiminnassa.

Kuva 39. Viimeisin ja pisin testi Mustalammella. Sää kävi jo talviseksi.

Hypoteettisena tavoitteena laitteen kestolle esitimme noin 40 tuntia toiminnassa. Tämä siis perustuen aiempiin testeihin pienemmällä akulla. Valitettavasti kuitenkin sensorilaitteeseen menetettiin kaikki yhteydet kello 20.00-20.15 välisenä aikana. MQTT-dataa ei sensoreilta tullut, emmekä päässeet etäyhteydellä laitteeseen kiinni. Laite kesti siis noin 10 tuntia.

Projektipäällikkö palasi paikanpäälle lauantaina kello 13.40, jolloin selvisi, että Huawei LTE-modeemista oli akku loppu. Näin ei pitäisi olla käynyt, sillä molemmat laitteet saivat virtaa siitä isosta 30000 mAh akusta. Kävikin ilmi, että modeemin mukana tulleen USB-johdon liittimen kontakti oli heikko. Tämä aiheutti sen, että modeemi on menettänyt virransaannin isosta akusta joko sensorilaitteen asennuksen yhteydessä tai myöhemmin, kun liikuttelimme sitä taataksemme paremman näkyvyyden liekkisensorille. Pieni korjaus johtoon sai modeemin taas latautumaan.

Testissä tavoittelemamme tulokset olivat jo tämän epäonnistumisen myötä menetetty, minkä vuoksi projektipäällikkö otti laitteen kotiinsa, latasi akut ja aloitti testin uudelleen takapihallaan.

4.5 Takapihatesti

Epäonnistuneen viimeisen Nuuksio-testin myötä jäi vielä epäselväksi se, kauanko projektin lopullinen kokoonpano tulee kestämään kylmissä ulko-olosuhteissa. Sensorilaite laitettiin ulos projektipäällikön takapihalle, kunnes akku tyhjeni.

Testi aloitettiin maanantaina kello 15.00 ja modeemin latautumista sekä sensorilaitteen yleistä toimivuutta seurattiin säännöllisin väliajoin. 24 tunnin kohdalla sensorilaitteen akku näytti akkuvarauksen jäljellä olevaksi määräksi 56 %. 30 tunnin kohdalla akkua oli jäljellä 32%. Laitteesta loppui virta tiistaina kello 22.00-22.15 välillä.

Prototyyppimme kesti siis ulko-olosuhteissa noin 31 tuntia. Tämä siis ilman projektisuunnitelmassa kaavailtua virransäästöominaisuutta. Huomattavaa oli, että virrankulutus kasvoi selvästi testin loppupuolella, tai sitten akku ei osaa tarkasti ilmoittaa virtatasoaan.

Kuva 40. Kiinnitystapa tehtiin nollabudjetilla, mutta laite pysyi hyvin paikallaan.

Lähteet

Karvinen, K., Karvinen, T., & Valtokari, V. (2014) Make: Sensors: A Hands-on Primer for Monitoring the Real World with Arduino and Raspberry Pi. Luettu 22.9.2017.

Raspberry Pi 2017. Raspberry Pi 3 Model B. URL: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. Luettu 15.9.2017.

Raspberry Pi 2017. Raspbian download. URL: https://www.raspberrypi.org/downloads/raspbian/. Luettu 15.9.2017.

Etcher 2017. Burn images to SD cards & USB drives, safely and easily. URL: https://etcher.io/. Luettu 15.9.2017.

ModMyPi 2016. How to give your Raspberry Pi a Static IP Address – Update. URL: https://www.modmypi.com/blog/how-to-give-your-raspberry-pi-a-static-ip-address-update. Luettu 15.9.2017.

Raspberry Pi 2017. Setting WiFi Up Via The Command Line. URL: https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md. Luettu 16.11.2017.

Botbook 2017. Make: Sensors. URL: http://makesensors.botbook.com/code.html. Luettu 22.9.2017.

Python 2017. Paho-MQTT Installation. URL: https://pypi.python.org/pypi/paho-mqtt/1.1. Luettu 22.9.2017.

Element14 2017. Raspberry Pi 3 Model B GPIO 40 Pin Block Pinout. URL: https://www.element14.com/community/servlet/JiveServlet/previewBody/73950-102-11-339300/pi3_gpio.png. Luettu 22.9.2017.

Henry's Bench 2017. Arduino HC-SR501 Motion Sensor Tutorial. URL: http://henrysbench.capnfatz.com/henrys-bench/arduino-sensors-and-input/arduino-hc-sr501-motion-sensor-tutorial/. Luettu 10.11.2017.

AliExpress 2017. IR Flame Sensor Module Detector Smartsense for Temperature Detecting Suitable. URL: https://goo.gl/X2LmA7. Luettu 17.9.2017.

Itead Wiki 2016. RPI SIM800 GSM/GPRS ADD-ON V2.0. URL: https://www.itead.cc/wiki/RPI_SIM800_GSM/GPRS_ADD-ON_V2.0. Luettu 6.10.2017.

ModMyPi 2016. How to connect your Raspberry Pi to a 3G network. URL: https://www.modmypi.com/blog/how-to-connect-your-raspberry-pi-to-a-3g-network. Luettu 6.10.2017.

Ubuntu 2015. Minicom Installation. URL: https://help.ubuntu.com/community/Minicom. Luettu 17.10.2017.

Cube-Controls 2015. Disable Serial Port Terminal Shell Output on the Raspbian/Raspberry Pi. URL: https://www.cube-controls.com/2015/11/02/disable-serial-port-terminal-output-on-raspbian/. Luettu 17.10.2017.

SIMCom 2015. SIM800 Series_AT Command Manual_V1.09. URL: https://www.elecrow.com/download/SIM800%20Series_AT%20Command%20Manual_V1.09.pdf. Luettu 17.10.2017.

Telia 2017. Huawei E5577T Asennus- ja käyttöohjeet sekä vianmääritys. URL: https://www.telia.fi/asiakastuki/laitteet/nettitikut-ja-4g-reitittimet/huawei-E5577T. Luettu 1711.2017.

TunnelsUp 2013. Raspberry Pi: Phoning Home Using a Reverse Remote SSH Tunnel. URL: https://www.tunnelsup.com/raspberry-pi-phoning-home-using-a-reverse-remote-ssh-tunnel/. Luettu 11.10.2017.

TunnelsUp 2015. SSH Without Password. URL: https://www.tunnelsup.com/ssh-without-password/. Luettu 11.10.2017.

Devolutions 2017. What is Reverse SSH Port Forwarding. URL: https://blog.devolutions.net/2017/03/what-is-reverse-ssh-port-forwarding.html. Luettu 11.10.2017.

Ask Ubuntu 2016. How to write a shscript to kill -9 a pid which is found via lsof -i. URL: https://askubuntu.com/questions/346394/how-to-write-a-shscript-to-kill-9-a-pid-which-is-found-via-lsof-i. Luettu 6.12.2017.

Superuser 2013. Kill all processes of a user except a few in linux. URL: https://superuser.com/questions/49114/kill-all-processes-of-a-user-except-a-few-in-linux. Luettu 6.12.2017.

Stackoverflow 2012. How to kill all processes with a given partial name. URL: https://stackoverflow.com/questions/8987037/how-to-kill-all-processes-with-a-given-partial-name. Luettu 6.12.2017.

DigitalOcean 2015. SaltStack Infrastructure: Installing the Salt Master. URL: https://www.digitalocean.com/community/tutorials/saltstack-infrastructure-installing-the-salt-master. Luettu 29.11.2017.

SaltStack 2017. SaltStack Package Repo. URL: https://repo.saltstack.com/#raspbian. Luettu 29.11.2017.

Raspberry Pi 2017. Scheduling tasks with cron. URL: https://www.raspberrypi.org/documentation/linux/usage/cron.md. Luettu 11.10.2017.

Raspberry Pi 2017. RC.Local. URL: https://www.raspberrypi.org/documentation/linux/usage/rc-local.md. Luettu 13.10.2017.

Ask Ubuntu 2015. How to command “Ping” display time and date of ping. URL: https://askubuntu.com/questions/137233/how-to-command-ping-display-time-and-date-of-ping. Luettu 6.12.2017.

UUGear 2017. Witty Pi 2: Realtime Clock and Power Management for Raspberry Pi. URL: http://www.uugear.com/product/wittypi2/. Luettu 13.10.2017.

UUGear 2017. Witty Pi 2: Realtime Clock and Power Management for Raspberry Pi – User Manual. URL: http://www.uugear.com/doc/WittyPi2_UserManual.pdf. Luettu 13.10.2017.

How to Geek 2012. How to Make your Linux PC Wake From Sleep Automatically: https://www.howtogeek.com/121241/how-to-make-your-linux-pc-wake-from-sleep-automatically/. Luettu 8.10.2017.

ArchLinux 2017. Powertop. URL: https://wiki.archlinux.org/index.php/powertop. Luettu 13.10.2017.

ArchLinux 2017. TLP. URL: http://www.uugear.com/product/wittypi2/. Luettu 13.10.2017.

DigitalOcean 2017. Cloud Computing, designed for developers. URL: https://www.digital-ocean.com/. Luettu 22.9.2017.

NameCheap 2017. Create your pro web presence in no time. URL: https://www.name-cheap.com/. Luettu 22.9.2017.

DigitalOcean 2013. How to Deploy a Flask Application on an Ubuntu VPS. URL: https://www.digitalocean.com/community/tutorials/how-to-deploy-a-flask-application-on-an-ubuntu-vps. Luettu 13.10.2017

EnvatoTuts 2015. Creating a Web App From Scratch Using Python Flask and MySQL. URL: https://code.tutsplus.com/tutorials/creating-a-web-app-from-scratch-using-python-flask-and-mysql--cms-22972. Luettu 13.10.2017

EV3Dev 2016. Sending and Receiving Messages with MQTT. URL: http://www.ev3dev.org/docs/tutorials/sending-and-receiving-messages-with-mqtt/. Luettu 27.10.2017.

W3schools 2017. Bootstrap Navigation Bar. URL: https://www.w3schools.com/boot-strap/bootstrap_navbar.asp. Luettu 22.9.2017.

Youtube – LearnWebCode 2016. JSON and AJAX Tutorial: With Real Examples. URL: https://www.youtube.com/watch?v=rJesac0_Ftw. Luettu 28.10.2017.

W3schools 2017. jQuery – Set Content and Attributes. URL: https://www.w3schools.com/jquery/jquery_dom_set.asp