Fleet Management Demonstration Setup

Abstract

This is a description of building the fleet management demo. Bill of Materials and setup procedures are included.

Application

This description is specific for fleet management demonstration only.

Content

1 Raspberry Pi 3 Setup

1.1 Preparation

The microSD class needs to be a minimum of 8 GB and Class 10.

For convenience, you may want to get a microSD to USB converter. This is optional.

A disk imager is needed. The environment was created in a Windows environment. The tool we used was “Win32DiskImager-0.9.5-install.exe”. It can be substituted by another similar tool.

An HDMI display monitor is recommended. [One can use a converter such as HDMI-to-DVI instead. However, the Raspberry Pi would not be able to detect the HDMI resolution. You would need to modify the display setting in /boot/config.txt to force the display resolution]

Gather the bill-of-materials as listed in 1.3.

1.2 Raspbian OS and tools

The reason we use the microSD card disk image for distribution is that there are a few tweaks needed in the Raspbian OS. This approach is the quickest way to get your demo going.

If you rather want to build it yourself. You would need to do these steps:

- Get a raspbian (I don’t use Noobs because of the extra disk consumption) https://www.raspberrypi.org/downloads/raspbian/
- Install PPP, add catm modem related scripts
- Enable 1-wire support and the 1-wire Python library
- Install tools to help debugging (e.g. Putty, iperf, wiringPi)
- A disk imager to allow SD card backup. Raspberry Pi likes to be shutdown via a command. Simple unplugging may corrupt the filesystem.

1.2.1 Prepare the SD card

Download the image (e.g. Fleet_Mgmt_Plano_RPi3_default.zip) and unzip it. The unzipped disk image is about 3G.

Use the disk imager to burn the image onto the microSD.

1.2.2 First Raspberry Pi boot

Connect the HDMI cable, keyboard and mouse to the Raspberry Pi.

Insert the microSD card into the Raspberry Pi.

Connect a power supply to the Raspberry Pi. It is recommended to use a power supply with a minimum of 2A rating.

You should see the bootup message on the display. Then the display may become blank with a cursor on the upper left. If it happens please follow these steps:

- Press Alt-F1, you would gain a command prompt
- Type “df” to see the percentage of disk used
- Type “sudo raspi-config” and <ENTER>. The userid is “pi” and the password is “raspberry”
- Select “1” to expand the filesystem. When the operation is completed, use the <Tab> to move the cursor to [Finish]
- The Raspberry Pi will reboot. You would see a GUI after the reboot.
- Open a terminal and type “df”. The disk should have been expanded.

1.3 Circuit Setup

1.3.1 Bill of Materials

1 x Sequans/Gemalto CAT M1

1 x Large size toy truck with doors that can open – local Toy store

1 x Raspberry Pi 3 kit with, 5Volt/2Aamp USB power, Ethernet cable, HDMI cable and Case - (Amazon or locally)

1 x Raspberry Pi heatsink set (optional)

1 x USB mouse and keyboard

1 x 8G microSD card. 16G or higher can also be used. The microSD card is class 10 or higher

1 x Breadboard - (Amazon or locally)

1 x GPIO Extension board – (i.e. SunFounder GPIO Breakout Expansion Board for Raspberry Pi B+ 40-pin GPIO Cable Plus Adapter)

Breadboard connection cables.

1 set Rainbow 40 Pin Flat Ribbon Cable + 65 Pcs Jumper Wires for Raspberry Pi 2 or 3

Sensors (Primary and backup) – order from Amazon

- 2 x Sunfounder tilt sensor or Tilt Ball sensor (SODIAL(R) 3.3V-5V DC Tilt Switch Sensor). If you use the tilt ball sensor, you may need to update the python source code
- 2 x Temperature sensor (DS18B20)
- 2 x Contact Sensor (Obtain from any hardware store (i.e Home Depot)

Carbon Film Resistors

- 2 x 4.7 K Ohm
- 4 x 220 Ohm

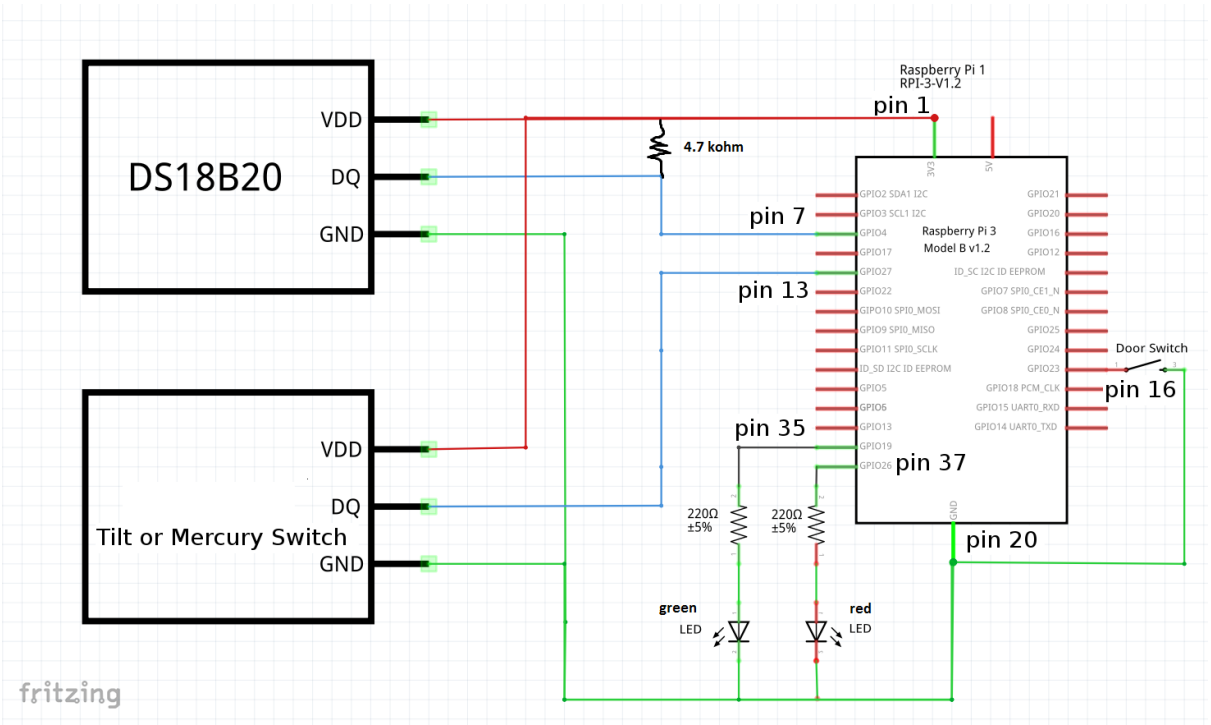
LEDs

- 2 x Red LEDs
- 2 x Green LEDs

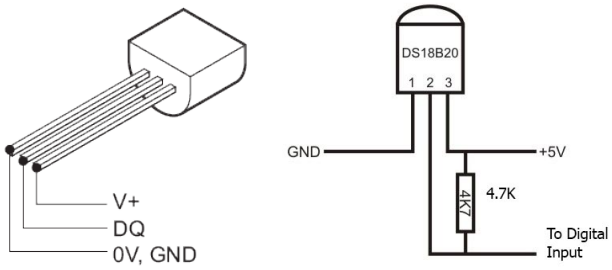
A custom built 8-wire cable out of a 5ft long CAT 6 cable.

1.3.2 Circuit

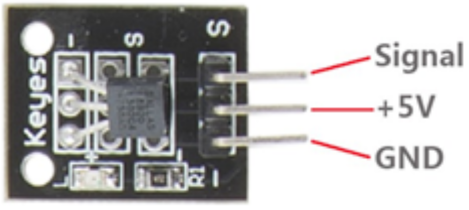
The circuit uses the 3.3V from the Raspberry Pi. Do not use the 5V as although the DS18B20 temperature sensor and the tile sensor can run on 5V, the input back to the Raspberry Pi is too high.



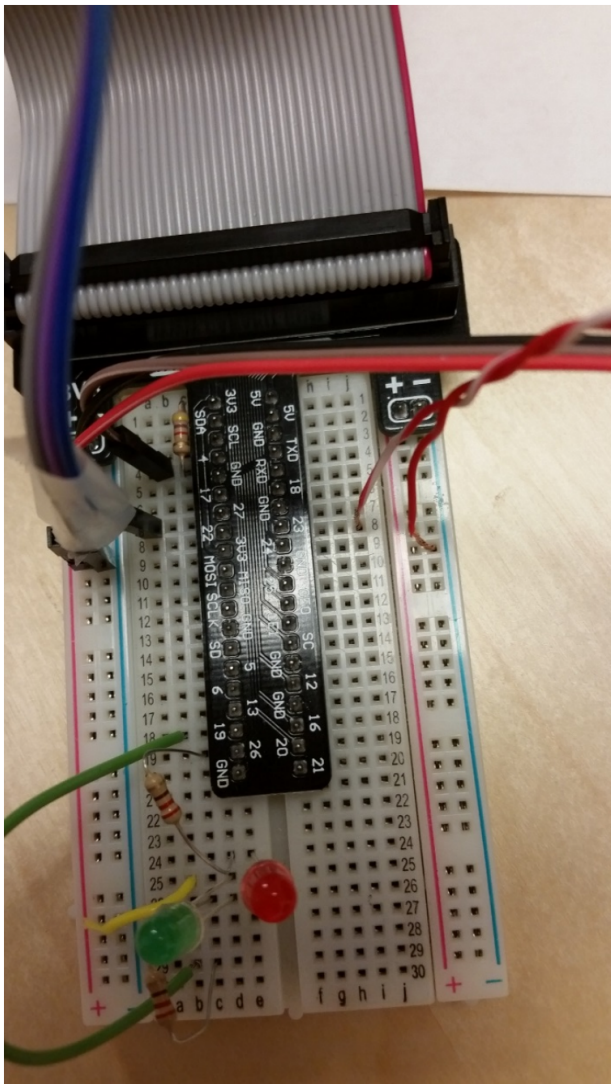
The above diagram assumes you are using the DS18B20 directly.



However, if you are using an integrated DS18B20 temperature sensor module as depicted below, the pull up resistor is already built-in and you can eliminate the 4.7 kohm resistor. Watch out that the pinout is different.



The custom built cable is used between the sensors and the Raspberry Pi. In other words, it serves as an extension cord only. There is no change to the circuit diagram.



2 AppIoT Setup

2.1 Preparation

Please request an AppIoT account from the Ericsson AppIoT administrator.

2.2 Creating your unique sensor IDs

The following is more or less the recipe that we used to configure Ericsson APPIoT. This example is taken from a previous email and you can substitute “Plano” with your own location. The Gateway is based on unique ID therefore the text less important.

1 From the drop-down menu "Settings -> Location" create a "Location" with a "Child Location". I created "FleetDemo" top-level location and a "Truck1" child location each with a sudo-random "External Id" a 4 digit integer value. (I started with 9988 for the External Id and as I added components ... I updated the last two digits 9977, 9966, 9955 ...)

2 From the drop-down menu "Settings -> Resolutions" I created 1 min, 5 min and 1 hour resolutions.

3 Next you would create a sensor if it does not already exist under "Settings -> Hardware Types -> Sensor Types". The Temperature sensor already exists, so I added the "Is_Open" and "Is_Tilted" sensors with a "Unit" setting of "1=True,0=False".

4 Next you need to create a "Device Type" which creates and association for the three sensors sensor types. This is done under "Settings -> Hardware Types -> Device Types". As part of the create process you need to provide a "Name" and a "Type Id" (an integer) and add the "Sensor Types" that this device will manage. In this case I created the device type "FleetDemo" and added the sensors, Temperature, Is_Tilted and Is_Open. These should be found in the pick list of defined "Sensor Types".

5 Next create a Gateway ... "Settings -> Hardware Types -> Gateway Types". As usual you need to give it a name and an integer based type ID. The "Internal Device Type" drop down should list the device that you created in the previous step. I called the GW => "FleetDemo_GW_Plano".

6 Expand your locations on the left hand side and select your Child Location. On the "Gateways" panel select "Register Gateway" and give it an integer "Serial Number" and select the GW you just created from the "Gateway Type" ... then select continue. In the next panel give the GW a name (TruckDemo_GW_SE1) and select the "Register" button.

With the above set up we would configure the Python simulation scripts to "post" sensor data and "get" sensor data.

2.3 Unique IDs needed by Raspberry Pi Python script and WebGUI

The "Http SAS" string is found by selecting your Gateways Name (FleetDemo_GW_Plano) link on the "Truck1" location page. In the new page that shows up ... you should see an "Actions" button drop-down menu. From this you would select the "Show Tickets". From the displayed information you want the "Http SAS" code from the "Outbox Access Ticket". Note that it is located in the section below the "Inbox Access Ticket".

The "Gateway ID" is found in the "Information" section on the page of the link you opened above before you selected the "Actions" button. It is called “Internal Id” in that page.

Next you need the "Internal Id" for each of the sensors. In the "Truck1" location page in the "Devices" section (middle section) ... select the "TruckDemo_GW_SE1" link. This should open a page with the Sensors listed on the right hand side of the page. Select each sensor link and locate the "Information" section to obtain the sensors associated "Internal Id".

3 WebGUI Setup

The web GUI consists of an HTML file referencing CSS, JavaScript, and JSON files, and a Perl script which operates in both offline and online modes. All required files are contained in the catMWebGui folder/archive.

3.1 Web Browser

The HTML et al portion of the web GUI requires no special software other than an up-to-date web browser, such as FireFox v45 or higher, or Internet Explorer 11 or higher. **Note:** Currently, the GUI does **not** function on Google Chrome.

To access the web GUI, simply open the ~/catMWebGui/nidex.html file in FireFox or IE.

3.2 Perl

The Perl script is called fleetsim.pl and uses several Perl modules that are not included in standard Perl distributions. These non-standard modules must be installed. If running the script from a Windows desktop or laptop PC, **ActivePerl** is recommended. We experienced some issues when trying to install the non-standard modules in Cygwin Perl. We have not been able to resolve the Cygwin Perl issues so far, thus **ActivePerl** is recommended. Also, the simulation script has not been tested on any Unix/Linux Perl distribution; presumably it would work if the correct modules are installed, but this has not been confirmed.

For more info on installing Perl on a Windows PC, please refer to Ericsson internal web page:

http://access.mo.ca.am.ericsson.se/cgi-bin/prod/radar.pl?DBASE=Tips&ACTION=VIEW&DOC=TIP20070925103647

3.2.1 Required Non-Standard Perl Modules

The following Perl modules are not part of a typical Perl distribution (e.g. ActivePerl), and so must be installed.

- REST::Client
- URI::Encode

If using ActivePerl, these can be easily installed using the Perl Package Manager, which is installed with ActivePerl.

For more information on how to install non-standard Perl modules, see internal Ericsson web page:

http://access.mo.ca.am.ericsson.se/cgi-bin/prod/radar.pl?DBASE=Tips&ACTION=VIEW&DOC=TIP20130829091108

3.3 fleetsim.pl

The fleet management simulation script is called fleetsim.pl. It can be run in online or offline modes.

3.3.1 Route & Warehouse .dat Files

In either (offline or online) mode, four virtual trucks follow pre-defined routes. These routes are contained in the ~/catMWebGui/routes folder, and have the filename format of 'route<n>.dat' where <n> is 0..3 and represents the truck ID number. Only route0.dat is mandatory. At their simplest level, the route<n>.dat files are a list of geographic (latitude,longitude) coordinate pairs:

```
<lat>,<lng>
```

Where <lat> and <lng> are the latitude and longitude of coordinates along the truck's virtual route. E.g.

```
45.35250,-75.91568
```

It is recommended that each consecutive pairs of coordinates be ~100m apart, as this script displays a set of coordinates every 5 seconds, and so with ~100m between coordinates, the virtual trucks move at ~72 km/h. These coordinates can be collected or generated in various ways, but the Ottawa IoT Garage team has scripts to help generate truck routes.

The route files can optionally include additional information like truck name, driver name, route name, and the names of destinations along the route.

If the other 3 trucks do not have route files provided, they will remain in the virtual warehouse.

Optionally, a warehouse.dat file can be defined in the same folder as the route files. The warehouse.dat file contains a single line of CSV data:

```
<lat>,<lng>,<name_string>
```

Where <lat> and <lng> are the latitude and longitude of the virtual warehouse, expressed in decimal degrees (not degrees-minutes-seconds). West and south coordinates are negative numbers. E.g.

```
47.58136,-52.67638,My Virtual Warehouse
```

If no warehouse.dat file is provided, it is assumed that the starting coordinates in the route0.dat file is the warehouse location and it will be simply called “Warehouse”.

3.3.2 Online Mode

In online mode, it queries the sensor readings from the AppIoT cloud, using more-or-less the same credentials and sensor IDs as described in AppIoT Setup, above.

The AppIoT data used by the script is contained in two files, found in the ~/catMWebGui/json folder.

1. The first is adal.config.json, which is also required by the various Python scripts for posting and querying sensor data to the AppIoT cloud.

2. The second file is called sensor.json, and is also located in ~/catMWebGui/json. It contains the Temperature, Door, and Accident sensor IDs to be queried.

It is recommended to edit the files provided using information for your specific AppIoT account and sensor information.

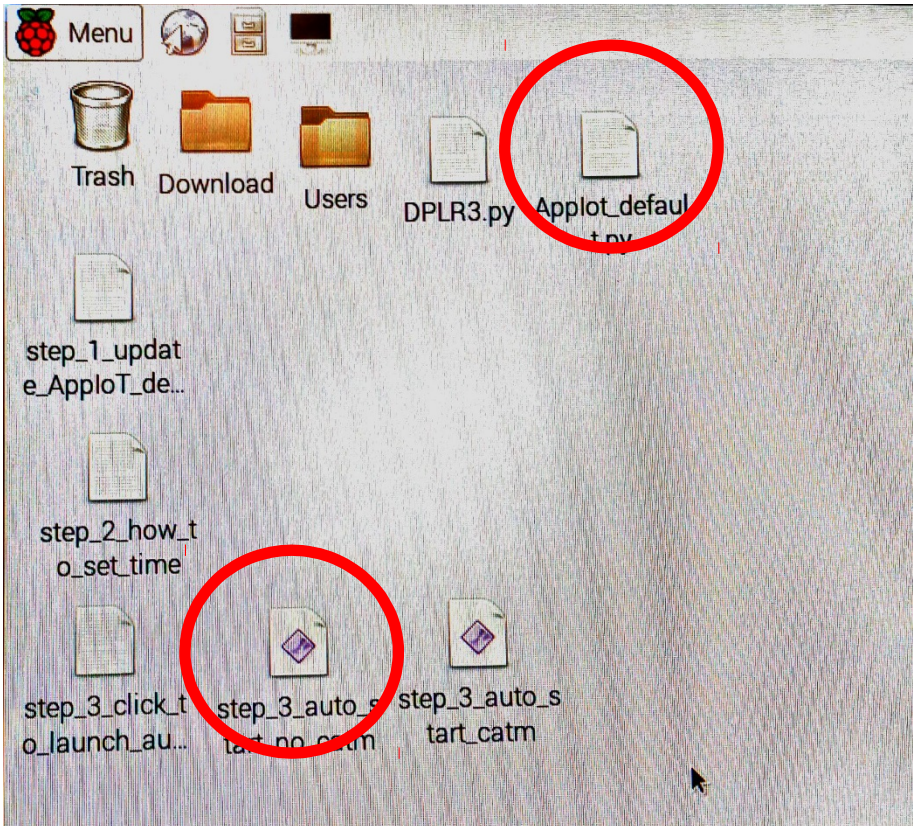
3.3.3 Offline Mode

In offline mode, sensor data is **not** queried from the AppIoT cloud, but is instead randomly generated. By default, there is a 1% chance at each new coordinate pair for truck 0 to have an accident or a door open. Temperature also changes randomly. Also by default, trucks 1 – 3 do not have events.

4 Running the demo

4.1 Raspberry Pi and AppIoT

On the Raspberry Pi desktop you will see this. There are three steps to follow and they are explained in the subsections.



4.1.1 Step 1: Update AppIoT default setting (Important !)

Edit the AppIoT_default.py with your unique ID. They are:

- gateway_id
- url

- sensors
- httpSAS

Otherwise you will be posting your data to someone else.

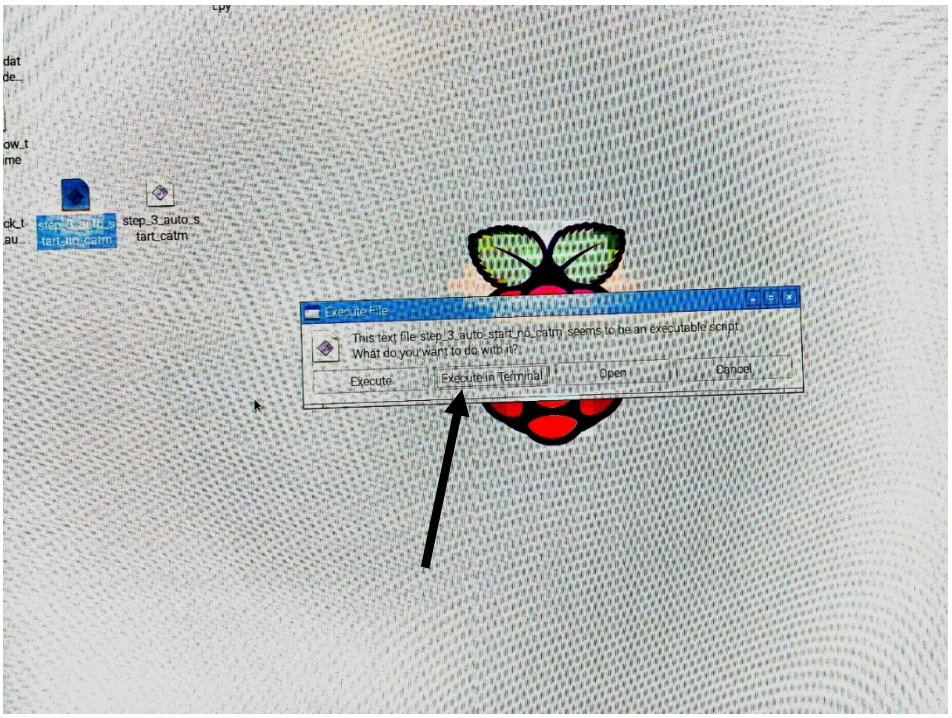
4.1.2 Step 2: Update the local time

If the Raspberry Pi was started without an Internet connection, the date/time will be wrong. The timestamp of your posted data will be wrong.

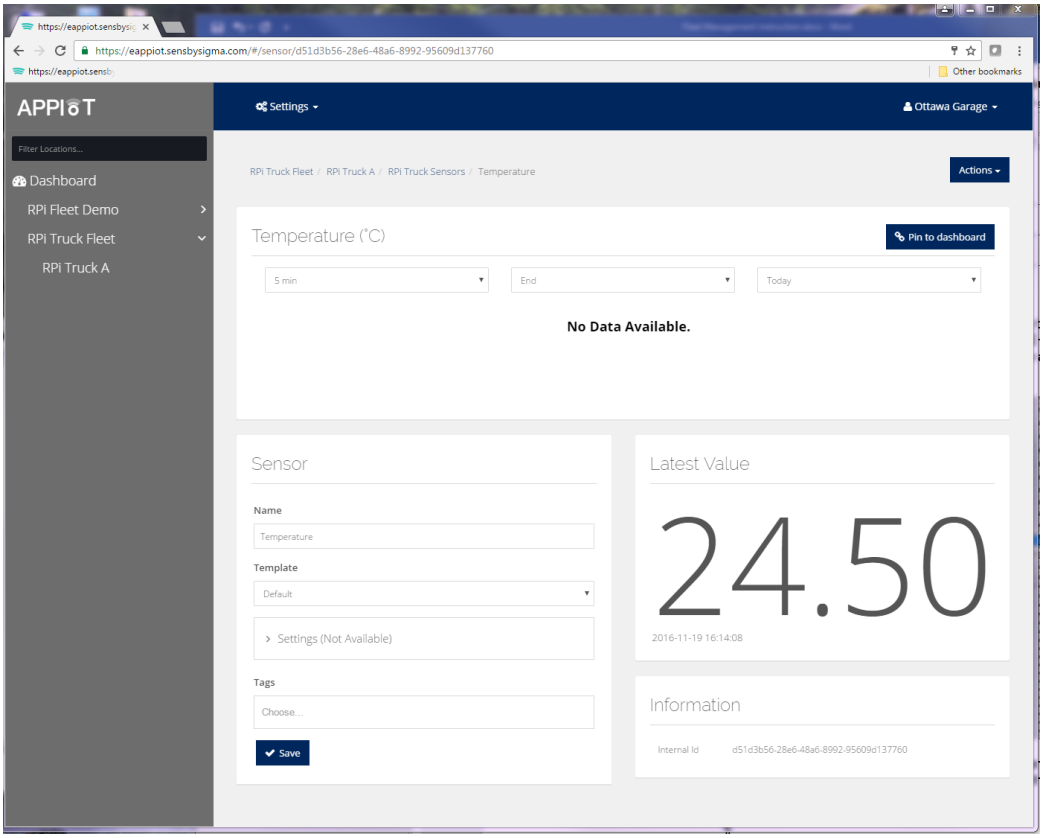
There are Linux command to do that. Open the step_2_how_to_set_time to see the example.

4.1.3 Step 3a: Start the demo without CatM

To verify the AppIoT is working before integrating the catm, I would connect the Raspberry Pi to the internet either through an LTE dongle or WiFt. Double click the icon step_3_auto_start_no_catm, you will see a dialog box. Pick “Execute in Terminal”.



You will see a terminal showing the sensor data and the return status of posting to AppIot. If you see a return status of 201, then the posting is successful. You can also check from the AppIoT page.



4.1.4 Step 3b: Start the demo with CatM

Now that step 3a confirmed the AppIoT is working, you can disconnect the LTE dongle or disable the WiFi connection. Connect the CatM modem. This is using the USB-to-serial connection cable.

Review the CatM modem user guide. Connect the serial connection to UART1. [Note: There are two serial connectors on the USB cable. The script assumes USB0 is used. Try the script and if the raspberry Pi cannot communicate with the modem, switch to the other serial connector of the cable. Most likely it was connected to USB1]

Double click the icon step_3_auto_start_catm, you will see a dialog box. Pick “Execute in Terminal”. This will launch a terminal on the screen.

You should see the download progress from the terminal. If it does not download the firmware the modem. Use a pin to press the reset button on the CatM modem. Try the icon step_3_auto_start_catm again.

There were many variables in the demo environment. Four scripts were created, for example setting the HPP proxy or not. You can find four scripts in /home/pi. You can experiment to see which one works for your setup. Once decided, you can edit the file “step_3_auto_start_catm” on the desktop.

4.2 Web GUI

To view the web GUI, simply load the ~/catMWebGui/index.html file in FireFox or Internet Explorer.

Sometimes, the GUI may stop responding. In this case, simply reload the page.

To begin the simulation, open a Windows command line window, navigate to the ~/catMWebGui folder, and run fleetsim.pl. Once the script is running, it will "move" the trucks approximately once every 5 seconds. The simulation will continue until all trucks have gone through their route at least once, or one hour (or as specified by the -t option), or until the user types Ctrl-C.

The fleetsim.pl script has some basic help, which can be viewed by executing:

```
fleetsim.pl -h
```

or

```
perl fleetsim.pl -h
```

4.2.1 Online Simulation

Run:

```
fleetsim.pl
```

or

```
perl fleetsim.pl
```

4.2.2 Offline Simulation

Run:

```
fleetsim.pl -offline
```

or

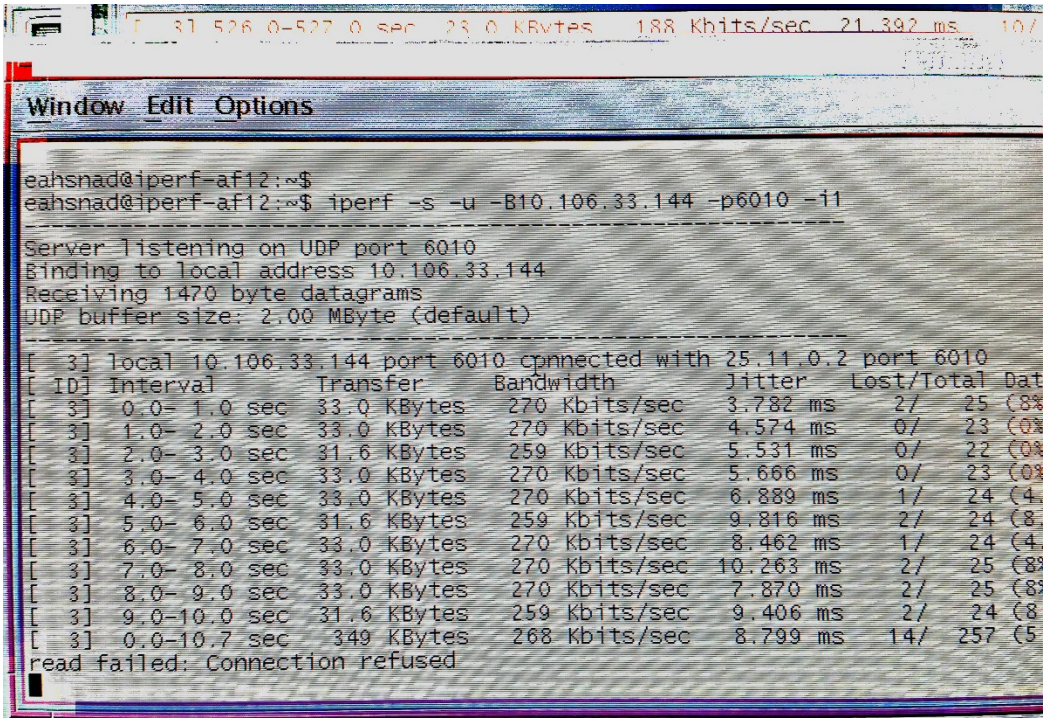
```
perl fleetsim.pl -offline
```

4.3 Iperf testing

To keep the cat-M link alive, we ping the DNS server from the UE to prevent the UE from sleeping. You should kill the ping process while running the iperf test, however the result does not differ a lot.

4.3.1 Uplink testing

Start the “server” side first from the iperf server



Where 10.106.33.144 is the lab iperf server address and the port chosen is 6010.

From the Raspberry Pi (i.e UE) issue this command as client.

```
pi@raspberrypi:~ $ iperf -c10.106.33.144 -i1 -b300k -B25.11.0.2 -p6010
WARNING: option -b implies udp testing
------------------------------------------------------------
Client connecting to 10.106.33.144, UDP port 6010
Binding to local address 25.11.0.2
Sending 1470 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[  3] local 25.11.0.2 port 6010 connected with 10.106.33.144 port 6010
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec  37.3 KBytes   306 Kbits/sec
[  3]  1.0- 2.0 sec  37.3 KBytes   306 Kbits/sec
[  3]  2.0- 3.0 sec  35.9 KBytes   294 Kbits/sec
[  3]  3.0- 4.0 sec  37.3 KBytes   306 Kbits/sec
[  3]  4.0- 5.0 sec  35.9 KBytes   294 Kbits/sec
[  3]  5.0- 6.0 sec  37.3 KBytes   306 Kbits/sec
[  3]  6.0- 7.0 sec  35.9 KBytes   294 Kbits/sec
[  3]  7.0- 8.0 sec  37.3 KBytes   306 Kbits/sec
[  3]  8.0- 9.0 sec  35.9 KBytes   294 Kbits/sec
[  3]  9.0-10.0 sec  37.3 KBytes   306 Kbits/sec
[  3]  0.0-10.1 sec   369 KBytes   300 Kbits/sec
[  3] Sent 257 datagrams
[  3] Server Report:
[  3]  0.0-10.7 sec   349 KBytes   268 Kbits/sec   8.799 ms   14/  257 (5.4%)
```

Where 10.106.33.144 was the server side IP and port is 6010. The UE IP address assigned was 25.11.0.2 (which you can find out from ifconfig)

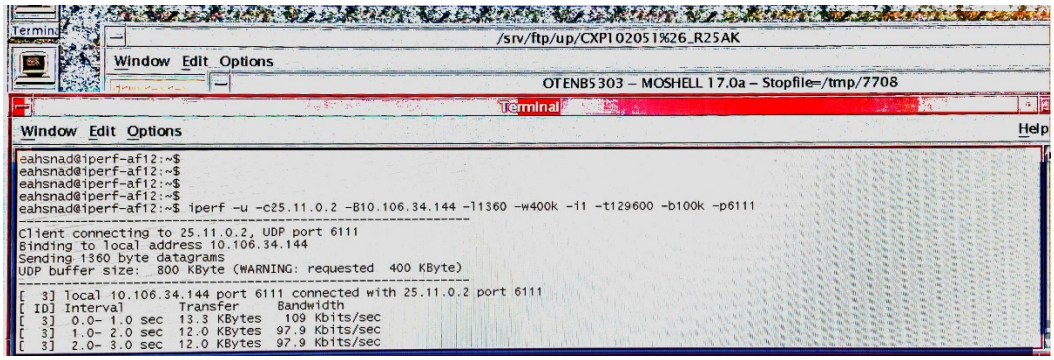
4.3.2 Downlink testing

Start the “server” side first from the UE.

```
pi@raspberrypi:~ $ iperf -s -u -i1 -w800k -p6111 -B25.11.0.2
------------------------------------------------------------
Server listening on UDP port 6111
Binding to local address 25.11.0.2
Receiving 1470 byte datagrams
UDP buffer size:  320 KByte (WARNING: requested  800 KByte)
------------------------------------------------------------
[  3] local 25.11.0.2 port 6111 connected with 10.106.34.144 port 6111
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagram
s
[  3]  0.0- 1.0 sec  10.6 KBytes  87.0 Kbits/sec   4.045 ms    0/    8 (0%)
[  3]  1.0- 2.0 sec  12.0 KBytes  97.9 Kbits/sec   6.612 ms    0/    9 (0%)
[  3]  2.0- 3.0 sec  12.0 KBytes  97.9 Kbits/sec  10.594 ms    0/    9 (0%)
[  3]  3.0- 4.0 sec  10.6 KBytes  87.0 Kbits/sec  10.162 ms    0/    8 (0%)
[  3]  4.0- 5.0 sec  12.0 KBytes  97.9 Kbits/sec  13.061 ms    0/    9 (0%)
[  3]  5.0- 6.0 sec  12.0 KBytes  97.9 Kbits/sec  11.854 ms    0/    9 (0%)
[  3]  6.0- 7.0 sec  10.6 KBytes  87.0 Kbits/sec  11.677 ms    0/    8 (0%)
[  3]  7.0- 8.0 sec  12.0 KBytes  97.9 Kbits/sec  13.151 ms    0/    9 (0%)
[  3]  8.0- 9.0 sec  10.6 KBytes  87.0 Kbits/sec  12.609 ms    0/    8 (0%)
[  3]  9.0-10.0 sec  12.0 KBytes  97.9 Kbits/sec  16.283 ms    1/   10 (10%)
[  3] 10.0-11.0 sec  12.0 KBytes  97.9 Kbits/sec  16.216 ms    1/   10 (10%)
[  3] 11.0-12.0 sec  10.6 KBytes  87.0 Kbits/sec  20.674 ms    1/    9 (11%)
[  3] 12.0-13.0 sec  12.0 KBytes  97.9 Kbits/sec  18.509 ms    0/    9 (0%)
[  3] 13.0-14.0 sec  12.0 KBytes  97.9 Kbits/sec  14.955 ms    0/    9 (0%)
[  3] 14.0-15.0 sec  10.6 KBytes  87.0 Kbits/sec  18.192 ms    1/    9 (11%)
[  3] 15.0-16.0 sec  12.0 KBytes  97.9 Kbits/sec  16.683 ms    0/    9 (0%)
[  3] 16.0-17.0 sec  10.6 KBytes  87.0 Kbits/sec  18.852 ms    1/    9 (11%)
[  3] 17.0-18.0 sec  12.0 KBytes  97.9 Kbits/sec  14.865 ms    0/    9 (0%)
[  3] 18.0-19.0 sec  12.0 KBytes  97.9 Kbits/sec  18.759 ms    1/   10 (10%)
[  3] 19.0-20.0 sec  10.6 KBytes  87.0 Kbits/sec  20.590 ms    1/    9 (11%)
[  3] 20.0-21.0 sec  12.0 KBytes  97.9 Kbits/sec  15.740 ms    0/    9 (0%)
^CWaiting for server threads to complete. Interrupt again to force quit.
[  3] 21.0-22.0 sec  12.0 KBytes  97.9 Kbits/sec  19.839 ms    1/   10 (10%)
[  3] 22.0-23.0 sec  10.6 KBytes  87.0 Kbits/sec  16.470 ms    0/    8 (0%)
[  3]  0.0-23.7 sec   272 KBytes  94.0 Kbits/sec  19.056 ms    9/  214 (4.2%)
read failed: Connection refused
pi@raspberrypi:~ $ ^C
```

25.11.0.2 is the assigned IP address of the UE (you can find out with ifconfig). We pick the port 6111.

Then move to the iperf server. Note that we select a different iperf server from the UL test because we want to test simultaneous UL and DL.



Here the UE address is 25.11.0.2 and the Iperf server that we are using is 10.106.34.144. we picked the port 6111.

4.3.3 Uplink and Downlink testing

Just start the “server” side from both uplink and downlink setup. Then launch the client side simultaneously.

4.3.4 Email instruction

Hi Scott,

Just quickly completed the throughput testing using Sequans UE. Below are the results.

- UL ONLY = 33 Kbytes.
- DL ONLY = 12~13 Kbytes.
- UL &DL together = 23Kbytes / 12~13 Kbytes.

Below some tips to run throughput using Raspberry Pi.

1-Make sure you able to ping the DNS server 10.122.34.251 from Raspberry pi. If not then factory restart the Sequan UE and load the firmware again.

2-On Raspberry Pi, ifconfig will give the UE IP address (ppp) / 25.11.0.3 in our case.

3-Ottawa lab iperf server 10.122.18.112. After login to this iperf server do ifconfig to get the ip address to be used in iperf commands. Here we picked up ip address **10.106.34.144 and 10.106.33.144.**

Below are the commands used:

UL Throughput:

- Server side (iperf)
  - iperf -s -u -B10.106.33.144 -p6010 -i1
- UE Side (Raspberry pi) / Client side
  - iperf -c10.106.33.144 -i1 -b300k -B25.11.0.2 -p6010

DL Throughput

- Server side (UE SIDE)
  - iperf -s -u -i1 -w800k -p50004 -B<ip of UE - match -C from server side command>

Client Side / iperf side

- iperf -u -c25.11.0.2 -B10.106.34.144 -l1360 -w400k -i1 -t129600 -b100k -p6111

Thanks.
Ahsan