Counting Coventry

Proof-of-Concept Working Paper

Si Chun Lam, Insight Development Manager, Coventry City Council
Coventry 2021 Monitoring and Evaluation Team

May 2021

Coventry City Council

ONE COVENTRY

Contents

Executive summary

This working paper sets out a proof-of-concept and demonstration of using a Raspberry Pi device to estimate footfall using Wi-Fi network tracking to approximate the number of visitors to an event, activity, or venue. It uses a few lines of Python code to, essentially, sniff for Wi-Fi devices, and record the resulting unique device addresses (known as MAC addresses) alongside a timestamp down to a resolution of milliseconds. This proof-of-concept proves that this methodology is indeed feasible; and the data collected can be combined using Power Query to estimate and count footfall and dwell time over a bespoke time period.

Background

Footfall counting, that is, counting the number of people visiting an event, activity, or venue provides a way in which operators and funders to understand the popularity of an event, activity, or venue.

Counting people helps us understand footfall traffic patterns across the city centre, enabling us to:

- put together funding bids to attract businesses and retailers to the city centre;
- improve public transport and green travel provision;
- understand the impact of our programme of investments and public arts in the city.

This is particularly important for Coventry in 2021; as the city has an ambition to be globally connected and locally committed. Over the past few years, and the past 12 months in particular, the local government has made significant investments to transform the city through public realm work and public arts as part of the UK City of Culture programme. This is set to continue during the City of Culture year in 2021/22; as a Commonwealth Games Host City in 2022; and with the City Centre South redevelopment and Friargate developments in the future years.

The long-term benefits of our investments can be measured through economic and wellbeing metrics such as an increase in gross value added (GVA), a local measure of economic output; GVA per capita, a local measure of economic productivity, and wellbeing measures. However, these are lagged measures, and none of it captures has the immediacy that is possible with footfall tracking.

Footfall is also a key measure in successful awards of business cases often supporting Pedestrian Environment Review System (PERS) calculations and driving a Value for Money measure or Benefit Cost Ratio calculation. It becomes the basis for targeted outcomes and output improvements.

Footfall tracking provide near real-time feedback to our activities and investments, by monitoring whether an activity or investment has increased the number of visits to various city venues – therefore justifying our investments, and providing justification for retailers, leisure operators, and other businesses to choose to invest in Coventry.

There are a variety of ways in which footfall can be tracked, such as:

- Clickers – using manual 'clicker' devices to count people;
- traditional footfall camera systems – often used to count people in/out of shops and malls;
- Wi-Fi tracking – estimating the number of people based on Wi-Fi signals on mobile phones;
- mobile network tracking – estimating the number of people using mobile networks; and
- survey-based sample – estimates using questionnaires and survey sampling.

The pros and cons of each method are outside the scope of this paper. Instead, the purpose of this paper is to set out a proof-of-concept of using a Raspberry Pi device to estimate footfall using Wi-Fi network tracking to approximate the number of visitors to an event, activity, or venue – how it is done; challenges encountered; and mitigations.

Introduction

Footfall counting using Wi-Fi tracking with a Raspberry Pi is not new; a paper, ***WiFiPi: Involuntary Tracking of Visitors at Mass Events*** was published at the 2013 IEEE 14th International Symposium[1].

> *The paper sets out "a method for tracking people at mass events without the need for active cooperation by the subjects. The mechanism works by scanning at multiple locations for packets sent out by the Wi-Fi interface on visitors' smartphones, and correlating the data captured at these different locations. The proposed method can be implemented at very low cost on Raspberry Pi computers. This implementation was trialled in two different contexts: a popular music festival and a university campus. The method allows for tracking thousands of people simultaneously, and achieves a higher coverage rate than similar methods for involuntary crowd tracking. Moreover, the coverage rate is expected to increase even further as more people will start using smartphones. The proposed method has many applications in different domains. It also entails privacy implications that must be considered when deploying a similar system."*

How it works

When a smartphone or tablet has Wi-Fi enabled, the device will continually search for a Wi-Fi network to connect to. When searching for a Wi-Fi network, the device sends out a probing request which contains an identifying number specific to that device known as a Media Access Control (MAC) address. This is also known as 'Wi-Fi connection data'.

When an individual with a smartphone or tablet with Wi-Fi enabled is near a counting device, the device will send a probing request to connect. This will be received by the counting device. This is no different from the probing requests received by any other Wi-Fi infrastructure devices.

When the device sends a probing request, it will contain a MAC address. Most modern devices send out a randomly generated MAC address to prevent unknown routers identifying the device.

Essentially, Wi-Fi tracking works by collecting the address sent out by devices, alongside a timestamp. This helps approximate the number of people visiting a location at different times of the day, and different days of the week.

---

[1] B. Bonné, A. Barzan, P. Quax and W. Lamotte, "WiFiPi: Involuntary tracking of visitors at mass events," *2013 IEEE 14th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, 2013, pp. 1-6, doi: 10.1109/WoWMoM.2013.6583443.

Use of Wi-Fi counting

The technology is now in widespread use, not least in Coventry itself – for instance, with the city centre Wi-Fi network run by InTechnology Wifi on behalf of the Council; and in the West Midlands 5G pilot using devices by a company called Meshh networks.

Similar techniques have been trialled in the UK by Transport for London (TfL), which began collection MAC addresses in 2019 (https://tfl.gov.uk/corporate/privacy-and-cookies/wi-fi-data-collection) with the intention of "gaining a far better understanding of how customers move through stations".

Privacy implications and mitigations

The intention of counting MAC addresses is to understand visitor patterns as a whole, not how specific individuals visit the city. In the original 2013 paper, the work involved correlating individuals as they move across different locations.

Since 2013, a number of things have taken place with differing implications on using Wi-Fi to undertake footfall tracking. Most notably is the implementation of media access control (MAC) address randomisation by device manufacturers to reduce tracking; initially, by Apple in 2016, which replaced the MAC address sequence that uniquely identifies an iPhone, iPad or Mac wireless hardware with randomly generated values (https://support.apple.com/en-gb/guide/security/secb9cb3140c/web) and similar techniques have been increasingly adopted by manufacturers of mobile devices using the alternative Google Android platform.

As most modern devices send out randomly generated MAC addresses, this means that devices now report different MAC addresses to different devices. This poses challenges for *tracking* individuals' movement (as detailed in the 2013 paper); but should not cause a problem for the purposes of footfall monitoring – and indeed, has privacy benefits as the technique cannot be used to track or identify any individual devices from the data collected.

Additionally, should visitors wish not to be tracked in this way, they can prevent their devices from sending out Wi-Fi connection requests by turning off Wi-Fi on their device; turning off the device entirely; or by putting the device into airplane mode.

Implementation

Bonné et al (2013) sets out the rationale and requirements for tracking visitors – a Raspberry Pi device running Raspberry Pi OS (then known as Raspbian); and making use of a Python library called *scapy* packet manipulation tool for computer networks. However, the source code used by Bonné et al is not published or made open source; so I have produced my own implementation for Counting Coventry.

My implementation for Counting Coventry includes:

- Raspberry Pi 4 Model B single-board computer with 2GB RAM (£34)
- External USB wireless LAN radio supporting monitor mode using the RT5370 chipset (£8.99)
- Raspberry Pi USB-C 5.1V / 3A power supply (£7.49)
- USB stick / SD card with Raspberry Pi OS and associated packages (£15.19 for 128GB)

The USB stick / SD card was provisioned with Raspberry Pi OS and the following is a summary of how it is provisioned:

1. **Install and update**
    a. connect the Raspberry Pi to an external monitor, keyboard and mouse; connect to a Wi-Fi network; on Raspberry Pi Configuration, enable SSH and VNC (so that we can connect remotely from another computer), rename hostname to *countingcoventryN* (where N is a unique name/number to make it easier to identify the device); set a password; reboot the computer

    Note: connecting to an existing Wi-Fi network is important to enable timestamping each MAC address, as the Raspberry Pi device does not have an internal real-time clock. Alternatively, work can be done post-hoc to adjust incorrect date/time.

    b. update the Raspberry Pi OS to the latest version (to make sure we have, amongst other things, the latest version of Python3) by running the following in Terminal:

    ```
    sudo apt update && sudo apt upgrade && sudo apt autoremove
    ```

    c. install scapy for Python3 by running the following in Terminal:

    ```
    pip3 install scapy
    ```

    d. set up the Raspberry Pi to launch the desktop environment even without a HDMI monitor connected by running the following in Terminal:

    ```
    sudo raspi-config
    ```

...and go to *Display Options* and select *1920x1080*

2. **Detect wireless configurations**
    a. list all network adapters – there should be two wireless radios – the internal Raspberry Pi wireless radio (to connect to a Wi-Fi network to send/receive data) should typically be `wlan0` and the external USB wireless radio (to monitor other Wi-Fi devices) should typically be `wlan1`. *(If this isn't the case – reboot.)*

    ```
    ifconfig
    ```

    b. double-check to ensure that the external USB wireless radio can be set in monitor mode:

    ```
    sudo ifconfig wlan1 down && sudo iwconfig wlan1 mode monitor && sudo ifconfig wlan1 up
    ```

3. **Write Python code**
    a. Using a Python editor, Thonny IDE, I adapted code published in a blog post by the SANS Institute, a cooperative for information security, on ***Wireless Client Sniffing with Scapy*** (https://www.sans.org/blog/special-request-wireless-client-sniffing-with-scapy/). I have adapted this with a thread for 'hopping' between Wi-Fi channels taken from a set of instructions published on Shellvoide on wireless sniffing in Python (https://www.shellvoide.com/python/how-to-code-a-simple-wireless-sniffer-in-python/).

    The code, in summary, tells the Raspberry Pi to "sniff" for Wi-Fi networks, and record the MAC address along a timestamp.

    The code is saved as `countingcoventry.py` and is as follows: Lines of Python code are explained in the commented text in blue.

```
#Counting Coventry 1.2.3

#Si Chun Lam

#27 May 2021

#Adapted from https://www.sans.org/blog/special-request-wireless-client-sniffing-with-scapy/

# This tells Python to import the scapy library for packet sniffing

from scapy.all import *

# Wi-Fi networks run on multiple channels, so we have code to hop between the Wi-Fi channels below to ensure we capture all devices.

import threading

import random
```

```
# To record the hostname of the device we import the socket (this is important if you have more than one device)

import socket

# To add a timestamp to each MAC address, we import the time from the operating system

import os, time

# This sets interface "wlan1", that is, the external USB wireless radio, into monitoring mode

interface = "wlan1"

os.system("ifconfig " + interface + " down")

os.system("iwconfig " + interface + " mode monitor")

os.system("ifconfig " + interface + " up")

print ("Welcome to Counting Coventry.")

print ("Please make sure the date and time is correct. Change it with sudo date -s 'yyyy-mm-dd hh:mm:ss'")

print ("Beginning Counting Coventry on",socket.gethostname(),"at", datetime.now(),".")

# The following code essentially tells the Raspberry Pi to sniff for Wi-Fi MAC addresses, and record each MAC address detected along with the timestamp when the MAC address was detected and the device own hostname onto a csv text file called countingcoventry.csv in the same folder.

#Code edited to use Dot11ProbeReq which MAY be better at capturing only packets requesting a Wi-Fi probe rather than all packets following GitHub thread at https://gist.github.com/dropmeaword/42636d180d52e52e2d8b6275e79484a0

with open("countingcoventry.csv", "a") as f:

                f.write("Counting Coventry")

                f.write(",")

                f.write(str(datetime.now()))

                f.write(",")

                f.write(str(socket.gethostname()))

                f.write ("\n")

                f.write("mac")

                f.write(",")

                f.write("datetime")

                f.write(",")

                f.write("device")

                f.write ("\n")

observedclients = []
```

```
def counting(coventry):
    if coventry.haslayer(Dot11ProbeReq):
#        if coventry.type == 0 and coventry.subtype == 8:
        stamgmtstypes = (0, 2, 4)
        print (coventry.addr2, datetime.now())
        with open("countingcoventry.csv", "a") as f:
                f.write(str(coventry.addr2))
                f.write(",")
                f.write(str(datetime.now()))
                f.write(",")
                f.write(str(socket.gethostname()))
                f.write ("\n")
        observedclients.append(coventry.addr2)
sniff(iface=interface, prn=counting)

# The following code is implemented as a parallel thread and is intended to tell the Raspberry Pi to hop between each of the Wi-Fi channels.  This is taken from code published by Shellvoide.

#Channel Hopping Thread https://www.shellvoide.com/python/how-to-code-a-simple-wireless-sniffer-in-python/

def hopper(iface):
    n = 1
    stop_hopper = False
    while not stop_hopper:
        time.sleep(0.50)
        os.system('iwconfig %s channel %d' % (iface, n))
        dig = int(random.random() * 14)
        if dig != 0 and dig != n:
            n = dig
if __name__ == "__main__":
    thread = threading.Thread(target=hopper, args=('wlan1', ), name="hopper")
    thread.daemon = True
    thread.start()
    while True:
```

4. **Run**
    a. To run the above Python code, we open Terminal and type in:

    ```
    sudo python3 countingcoventry.py
    ```

5. **Launch automatically on startup**
    a. Finally, so that the Raspberry Pi OS device can run automatically on start-up without any user intervention, we add this to **systemd** in Raspberry Pi, and set it to **not only run on start-up, but also restart the script if, for whatever reasons, it quits**.

    This is adapted from https://learn.sparkfun.com/tutorials/how-to-run-a-raspberry-pi-program-on-startup#method-3-systemd

    b. Firstly, we create a new .service file in the systemd directory:

    ```
    sudo mousepad /lib/systemd/system/countingcoventry.service
    ```

    c. In the countingcoventry.service file, we tell it to launch terminal and run countingcoventry.py – and most importantly, restart itself if for whatever reasons the script ends up in an error and quits itself.

    ```
    [Unit]
    Description=Counting Coventry
    [Service]
    Environment=DISPLAY=:0
    Environment=XAUTHORITY=/home/pi/.Xauthority
    ExecStart=lxterminal -e "/usr/bin/sudo /usr/bin/python3
    /home/pi/countingcoventry.py"
    Restart=always
    RestartSec=10s
    KillMode=process
    TimeoutSec=infinity
    [Install]
    WantedBy=graphical.target
    ```

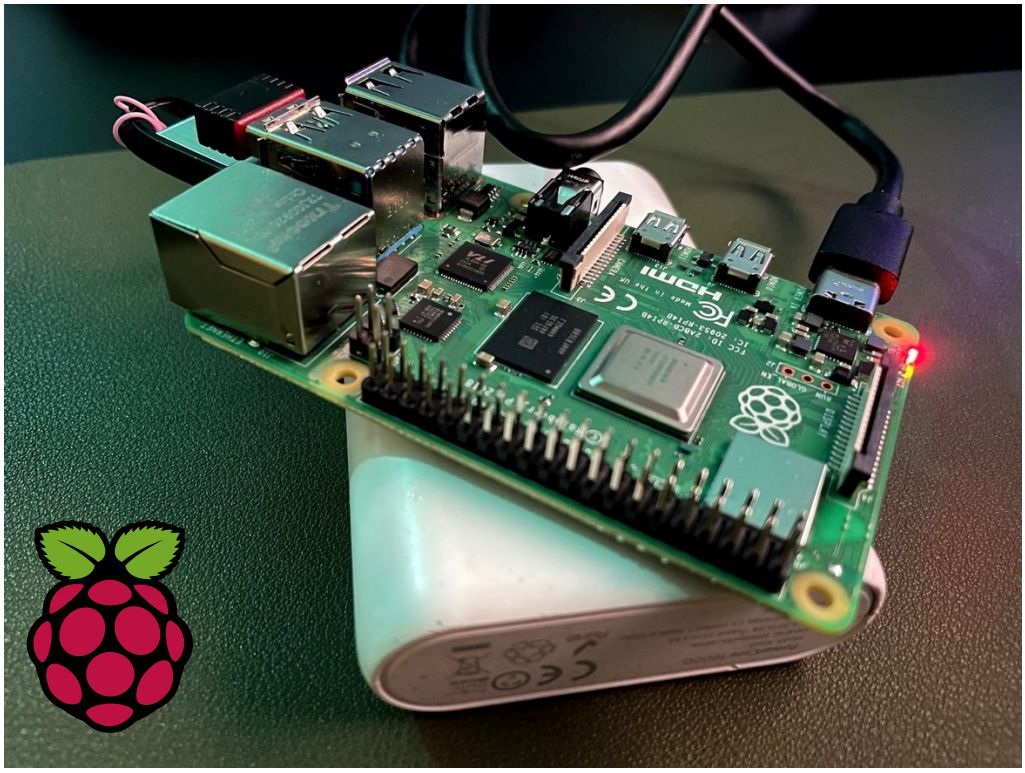
    d. Now, we tell Raspberry Pi's systemd to recognise our service and start it on boot by typing in:

    ```
    sudo systemctl daemon-reload

    sudo systemctl enable countingcoventry.service

    sudo reboot
    ```

Demonstration

As a proof-of-concept to demonstrate that this was a feasible way to count footfall ‘on the move’, I went for a drive and walk around Coventry with the Raspberry Pi device set up to ‘count’ footfall using the implementation set up above. The device was connected to a USB battery pack, which lasted for around two hours before it was no longer able to reliably power the device and counting ceased. For reliability, a Raspberry Pi should be connected to an approved power supply rather than a USB device.

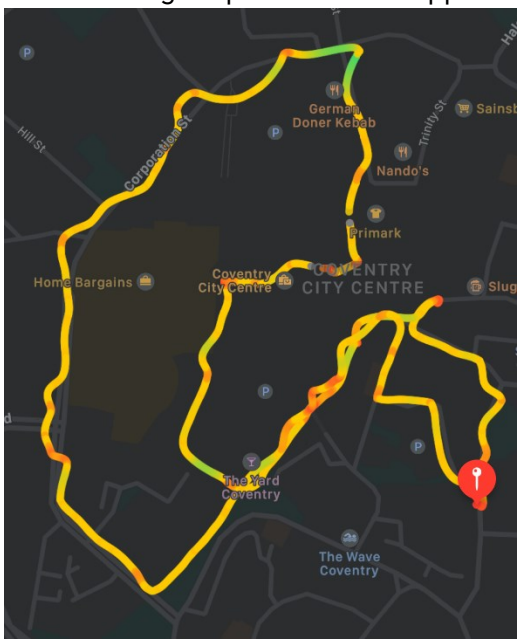
The Raspberry Pi device used; with a second external USB wireless radio, USB stick powered on connected to a USB battery pack is shown in the photograph below:



Experiment 1: Proof of Concept, 16 May 2021

Between 09:25 and 11:05 on 16 May 2021, I collected a total of 750 MAC addresses (including duplicates and unknowns) on the countingcoventry.csv file. This was – (roughly) driving at 09:25; Raspberry Pi left in my car at a COVID-19 vaccination centre (Coventry and North Warwickshire Sports Club on Binley Road) from 09:30-10:00; a drive to the city centre arriving at around 10:10; and a walk around the city centre until 11:05.

The following map sets out the approximate route taken around the city centre:



The data contained the MAC address alongside a timestamp down to milliseconds; and included a large number of duplicates and “none’s”. This is illustrated in the screenshot below, with the latter part of the MAC address obfuscated:

ba:d7:a[redacted]	2021-05-16 10:43:17.061043
ba:d7:a[redacted]	2021-05-16 10:43:17.163469
ba:d7:a[redacted]	2021-05-16 10:43:17.264220
None	2021-05-16 10:43:17.523345
None	2021-05-16 10:43:17.526960
None	2021-05-16 10:43:17.530316
None	2021-05-16 10:43:17.533863
None	2021-05-16 10:43:17.537778
None	2021-05-16 10:43:17.541233
c2:0f:90[redacted]	2021-05-16 10:43:17.547763
None	2021-05-16 10:43:17.551317
None	2021-05-16 10:43:17.554685
None	2021-05-16 10:43:17.557947
None	2021-05-16 10:43:17.561279
None	2021-05-16 10:43:17.564594
c2:0f:90[redacted]	2021-05-16 10:43:17.571734
None	2021-05-16 10:43:17.575499
None	2021-05-16 10:43:17.579341
None	2021-05-16 10:43:17.582875
c2:0f:90[redacted]	2021-05-16 10:43:17.590174
c6:5e:3[redacted]	2021-05-16 10:43:18.074478
None	2021-05-16 10:43:18.082670
None	2021-05-16 10:43:18.091065
ba:d7:a[redacted]	2021-05-16 10:43:18.110240
ba:d7:a[redacted]	2021-05-16 10:43:18.127572
ba:d7:a[redacted]	2021-05-16 10:43:18.145088
ba:d7:a[redacted]	2021-05-16 10:43:18.161355
None	2021-05-16 10:43:18.167183

With individual MAC addresses and timestamps, we are able to combine this data in different ways to work out footfall at very specific intervals. In my demonstration, as I was walking about / driving, I decided to count unique devices on a five, fifteen, 30 and 60-minute interval. To do so, I wrote a series of Microsoft Power Query script to do so.

This code is set out below:

```
// countingcoventry-src
let
    Source = SharePoint.Files("https://address.sharepoint.com/ ", [ApiVersion = 15]),
    #"csv" = Source{[Name="countingcoventry.csv",#"Folder Path"="
https://address.sharepoint.com/path/to/file/"]}[Content],
    #"Imported CSV" = Csv.Document(#"csv",[Delimiter=",", Columns=3,
Encoding=1252, QuoteStyle=QuoteStyle.None]),
    #"Removed Top Rows" = Table.Skip(#"Imported CSV",1),
    #"Promoted Headers" = Table.PromoteHeaders(#"Removed Top Rows",
[PromoteAllScalars=true]),
    #"Filtered Rows1" = Table.SelectRows(#"Promoted Headers", each [datetime] <>
null and [datetime] <> ""),
    #"Changed Type" = Table.TransformColumnTypes(#"Filtered Rows1",{{"mac", type
text}, {"datetime", type datetime}, {"device", type text}}),
    #"Removed Errors" = Table.RemoveRowsWithErrors(#"Changed Type",
{"datetime"}),
    #"Filtered Rows" = Table.SelectRows(#"Removed Errors", each [mac] <> "Counting
Coventry" or [mac] <> "mac"),
    #"Inserted First Characters" = Table.AddColumn(#"Filtered Rows", "manuf", each
Text.Start([mac], 8), type text),
    #"Merged Queries" = Table.NestedJoin(#"Inserted First Characters", {"manuf"},
manuf, {"mac"}, "manuf.1", JoinKind.LeftOuter),
    #"Uppercased Text" = Table.TransformColumns(#"Merged Queries",{{"manuf",
Text.Upper, type text}}),
    #"Expanded manuf.1" = Table.ExpandTableColumn(#"Uppercased Text",
"manuf.1", {"manufacturer"}, {"manufacturer"}),
    #"Removed Columns" = Table.RemoveColumns(#"Expanded manuf.1",{"manuf"}),
    #"Inserted Date" = Table.AddColumn(#"Removed Columns", "Date", each
DateTime.Date([datetime]), type date),
    #"Inserted Time" = Table.AddColumn(#"Inserted Date", "Time", each
DateTime.Time([datetime]), type time)
in
    #"Inserted Time"
// countingcoventry-5m
let
    Source = #"countingcoventry-src",
    #"Trimmed Text" = Table.TransformColumns(Table.TransformColumnTypes(Source,
{{"Time", type text}}, "en-GB"),{{"Time", Text.Trim, type text}}),
    #"Changed Type1" = Table.TransformColumnTypes(#"Trimmed Text",{{"Time",
type time}}),
    #"Added Custom" = Table.AddColumn(#"Changed Type1", "Time Interval", each
[Time] + #duration(0,0,5- Number.Mod(Time.Minute([Time]),5),0)),
    #"Merged Date and Time" = Table.CombineColumns(#"Added Custom", {"Date",
"Time Interval"}, (columns) => List.First(columns) & List.Last(columns), "Date-Time"),
    #"Changed Type2" = Table.TransformColumnTypes(#"Merged Date and
Time",{{"Date-Time", type datetime}}),
```

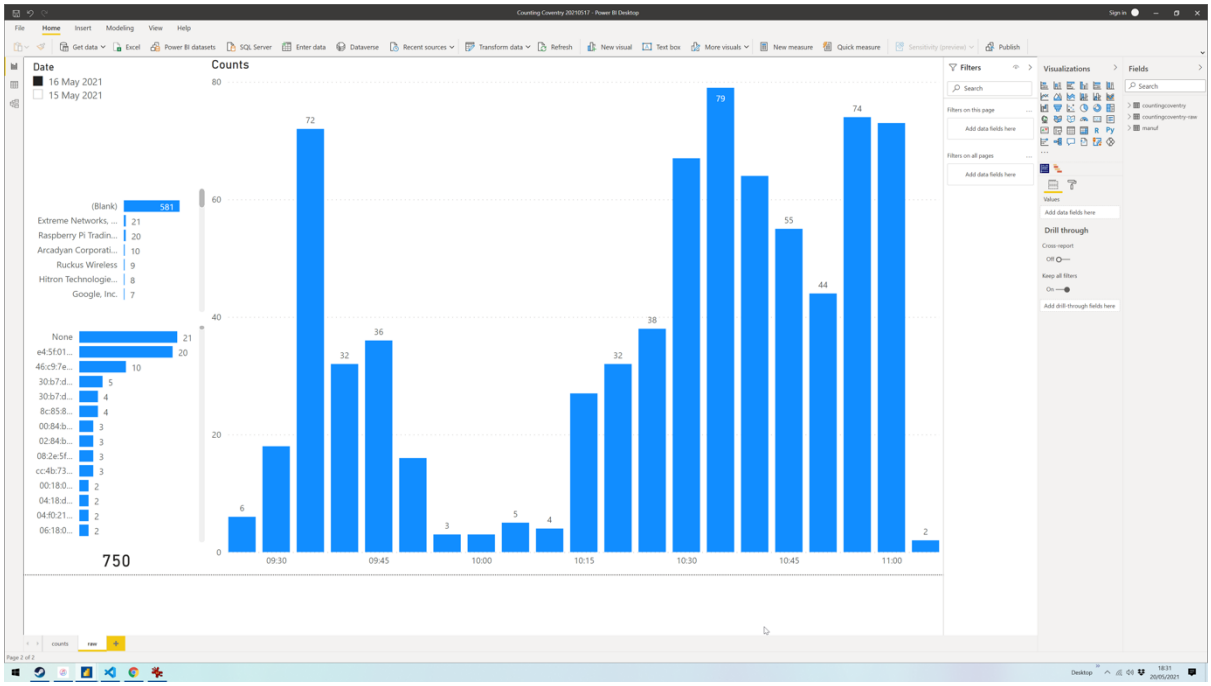
```
    #"Removed Columns" = Table.RemoveColumns(#"Changed Type2",{"datetime",
"Time"}),
    #"Removed Duplicates" = Table.Distinct(#"Removed Columns"),
    #"Inserted Date1" = Table.AddColumn(#"Removed Duplicates", "Date", each
DateTime.Date([#"Date-Time"]), type date)
in
    #"Inserted Date1"
// countingcoventry-15m
let
    Source = #"countingcoventry-src",
    #"Trimmed Text" = Table.TransformColumns(Table.TransformColumnTypes(Source,
{{"Time", type text}}, "en-GB"),{{"Time", Text.Trim, type text}}),
    #"Changed Type1" = Table.TransformColumnTypes(#"Trimmed Text",{{"Time",
type time}}),
    #"Added Custom" = Table.AddColumn(#"Changed Type1", "Time Interval", each
[Time] + #duration(0,0,0- Number.Mod(Time.Minute([Time]),15),0)),
    #"Filtered Rows" = Table.SelectRows(#"Added Custom", each true),
    #"Merged Date and Time" = Table.CombineColumns(#"Filtered Rows", {"Date",
"Time Interval"}, (columns) => List.First(columns) & List.Last(columns), "Date-Time"),
    #"Changed Type2" = Table.TransformColumnTypes(#"Merged Date and
Time",{{"Date-Time", type datetime}}),
    #"Removed Columns" = Table.RemoveColumns(#"Changed Type2",{"datetime",
"Time"}),
    #"Removed Duplicates" = Table.Distinct(#"Removed Columns"),
    #"Inserted Date1" = Table.AddColumn(#"Removed Duplicates", "Date", each
DateTime.Date([#"Date-Time"]), type date)
in
    #"Inserted Date1"
// countingcoventry-30m
let
    Source = #"countingcoventry-src",
    #"Trimmed Text" = Table.TransformColumns(Table.TransformColumnTypes(Source,
{{"Time", type text}}, "en-GB"),{{"Time", Text.Trim, type text}}),
    #"Changed Type1" = Table.TransformColumnTypes(#"Trimmed Text",{{"Time",
type time}}),
    #"Added Custom" = Table.AddColumn(#"Changed Type1", "Time Interval", each
[Time] + #duration(0,0,0- Number.Mod(Time.Minute([Time]),30),0)),
    #"Filtered Rows" = Table.SelectRows(#"Added Custom", each true),
    #"Merged Date and Time" = Table.CombineColumns(#"Filtered Rows", {"Date",
"Time Interval"}, (columns) => List.First(columns) & List.Last(columns), "Date-Time"),
    #"Changed Type2" = Table.TransformColumnTypes(#"Merged Date and
Time",{{"Date-Time", type datetime}}),
    #"Removed Columns" = Table.RemoveColumns(#"Changed Type2",{"datetime",
"Time"}),
    #"Removed Duplicates" = Table.Distinct(#"Removed Columns"),
    #"Inserted Date1" = Table.AddColumn(#"Removed Duplicates", "Date", each
DateTime.Date([#"Date-Time"]), type date)
in
    #"Inserted Date1"
// countingcoventry-60m
let
    Source = #"countingcoventry-src",
```

```
    #"Trimmed Text" = Table.TransformColumns(Table.TransformColumnTypes(Source,
{{"Time", type text}}, "en-GB"),{{"Time", Text.Trim, type text}}),
    #"Changed Type1" = Table.TransformColumnTypes(#"Trimmed Text",{{"Time",
type time}}),
    #"Inserted Start of Hour" = Table.AddColumn(#"Changed Type1", "Time Interval",
each Time.StartOfHour([Time]), type time),
    #"Merged Date and Time" = Table.CombineColumns(#"Inserted Start of Hour",
{"Date", "Time Interval"}, (columns) => List.First(columns) & List.Last(columns),
"Date-Time"),
    #"Changed Type2" = Table.TransformColumnTypes(#"Merged Date and
Time",{{"Date-Time", type datetime}}),
    #"Removed Columns" = Table.RemoveColumns(#"Changed Type2",{"datetime",
"Time"}),
    #"Removed Duplicates" = Table.Distinct(#"Removed Columns"),
    #"Inserted Date1" = Table.AddColumn(#"Removed Duplicates", "Date", each
DateTime.Date([#"Date-Time"]), type date)
in
    #"Inserted Date1"
```

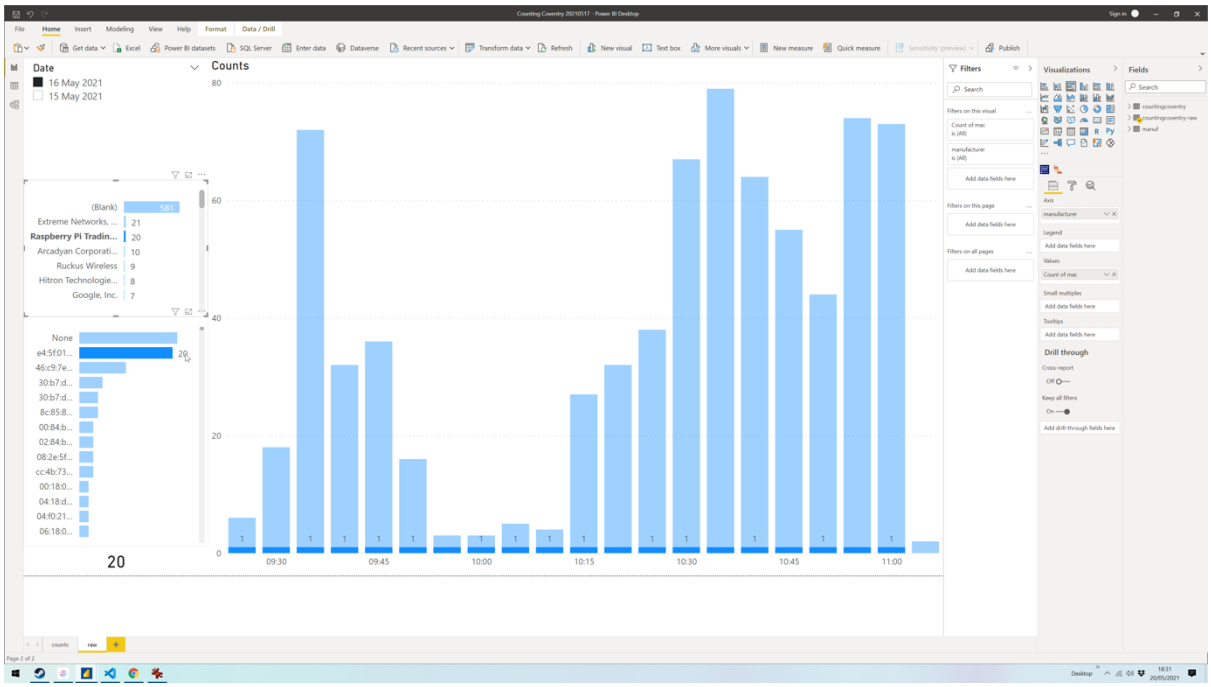
The first part of a MAC address identifies the manufacturer, and Wireshark has published a list of manufacturers on Gitlab. So, my code above is merged with code to try and identify those manufacturers.

```
// manuf
let
    Source =
Csv.Document(Web.Contents("https://gitlab.com/wireshark/wireshark/-
/raw/master/manuf"),[Delimiter="#(tab)", Columns=3, Encoding=65001,
QuoteStyle=QuoteStyle.None]),
    #"Filtered Rows" = Table.SelectRows(Source, each [Column3] <>
null and [Column3] <> ""),
    #"Renamed Columns" = Table.RenameColumns(#"Filtered
Rows",{{"Column1", "mac"}}),
    #"Removed Columns" = Table.RemoveColumns(#"Renamed
Columns",{"Column2"}),
    #"Renamed Columns1" = Table.RenameColumns(#"Removed
Columns",{{"Column3", "manufacturer"}})
in
    #"Renamed Columns1"
```

With this above code, we can now visualise the 750 MAC addresses collected using Microsoft Power BI. The following screenshot shows the distribution of each of the 750 Wi-Fi devices ‘counted’ at five-minute intervals:

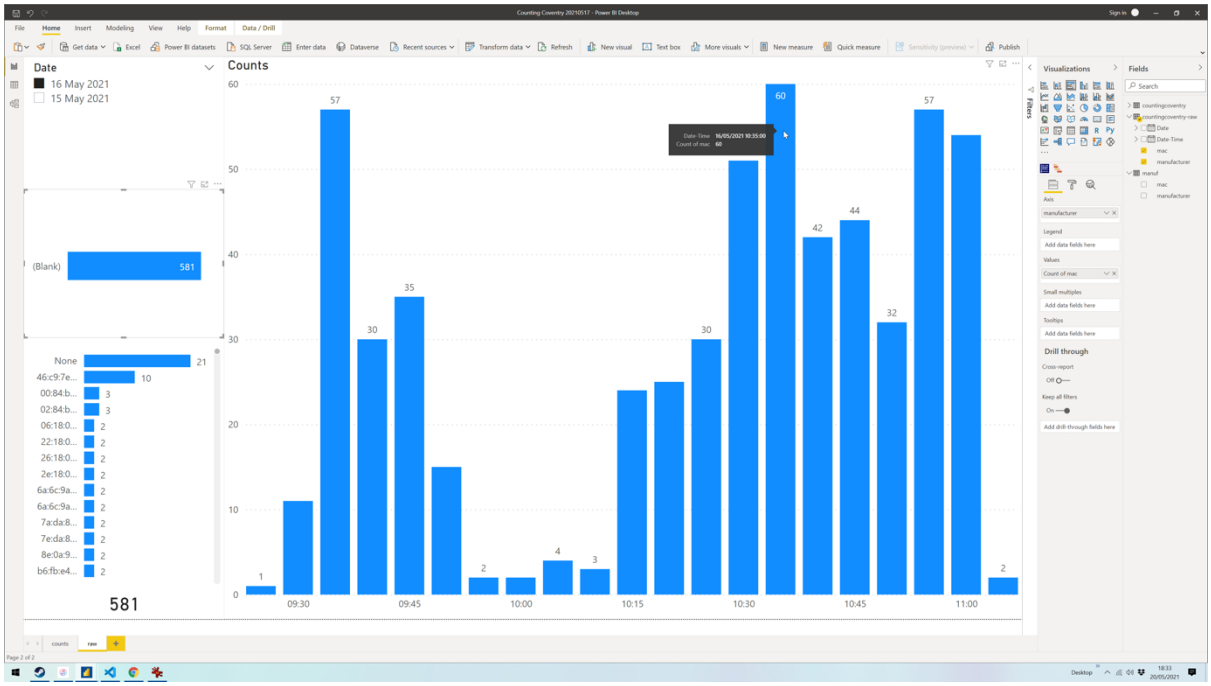


However, by filtering by devices, we are able to detect duplicates. For example, we are finding a Raspberry Pi device every five minutes from 09:25 until 11:00. This is because the Wi-Fi monitoring radio (`wlan1`) is "detecting" the internal Wi-Fi radio (`wlan0`) present throughout this period:

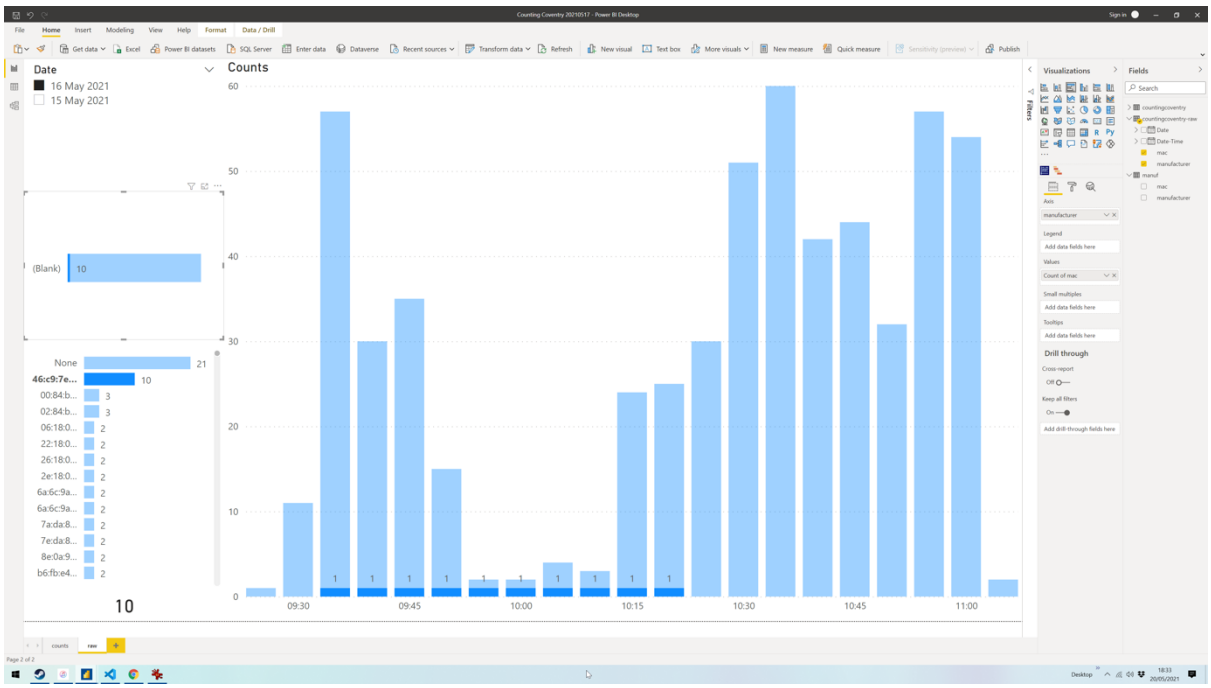


There are other commonly-detected networks that are linked to Wi-Fi routers and other infrastructure devices (e.g. Extreme Networks, Arcadyan Corporation, Ruckus Wireless, Hitron Technologies) which can be filtered out too.

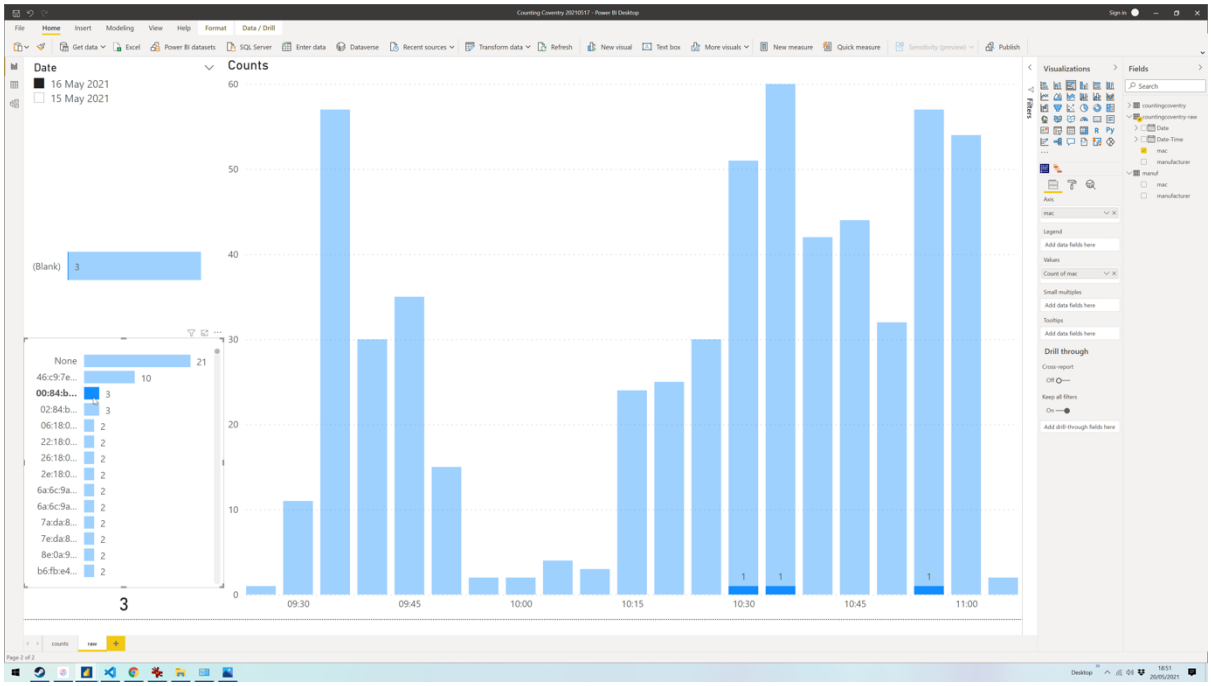
Indeed – one way is to focus on the "private" MAC addresses (i.e. likely to be modern iPhone, Android and other modern, non-infrastructure mobile devices) that was detected. This leaves a count of 581, with a peak of 60 unique devices counted at 10:35; and a minimum of 1 unique device at 09:25.



It is also possible to calculate ‘dwell time’ – for instance, the highlighted device below is likely to be my ‘work’ phone which was in my car next to my Raspberry Pi device:



And the following device may be an individual who may have been in close contact at multiple points during my walk around the city centre:


Counting Coventry 20210517 - Power BI Desktop
Date
16 May 2021
15 May 2021
Counts
(Blank) 3
None 21
46:c9:7e... 10
00:84:b... 3
02:84:b... 3
06:18:0... 2
22:18:0... 2
26:18:0... 2
2e:18:0... 2
6a:6c:9a... 2
6a:6c:9a... 2
7a:da:8... 2
7e:da:8... 2
8e:0a:9... 2
b6:fb:e4... 2
3
60
50
40
30
20
10
0
09:30
09:45
10:00
10:15
10:30
10:45
11:00
Visualizations
Fields
Drill through
counts
raw

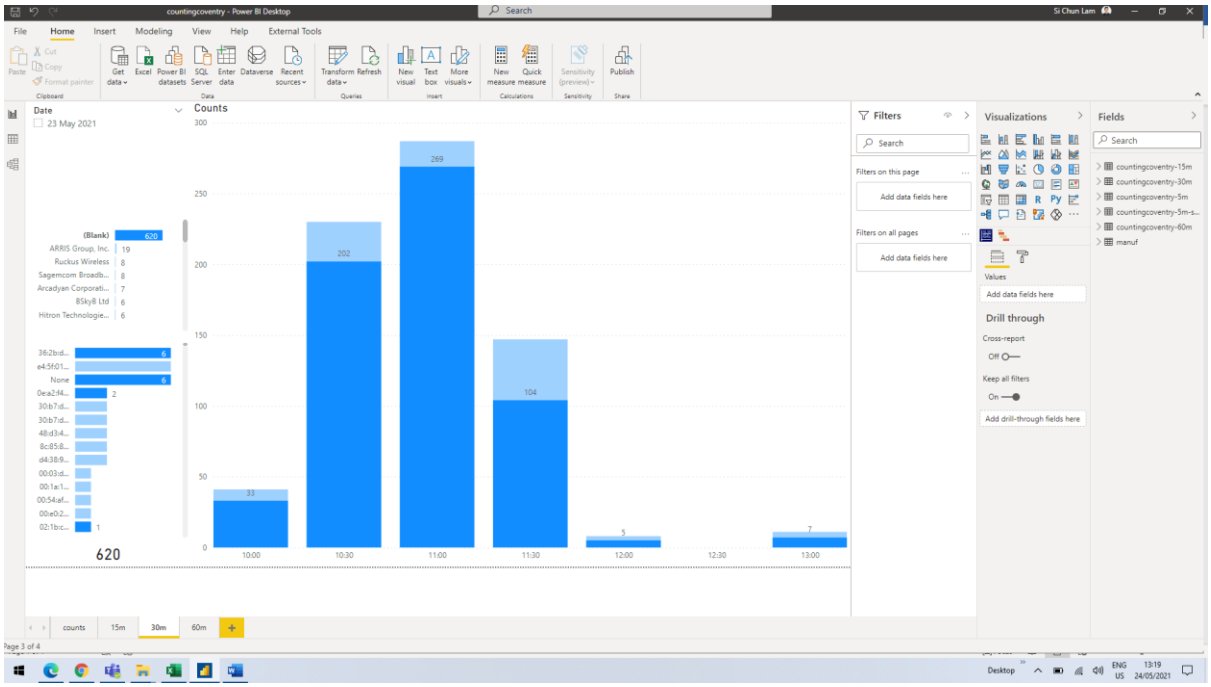
Experiment 2: Calibrating Accuracy, 23 May 2021

The first experiment on 16 May proved that the technique was feasible – however, how does the numbers compare to a manual count of people? So, for the second experiment, I sought to compare the accuracy of counting Wi-Fi signals compared to a visual estimate of the size of a crowd.

On Sunday, 23 May 2021, I took the Counting Coventry Raspberry Pi proof of concept device to estimate the size of the congregation at Sunday Mass at Sacred Heart Coventry in Harefield Road, Coventry, from around 10:35 to 11:30.

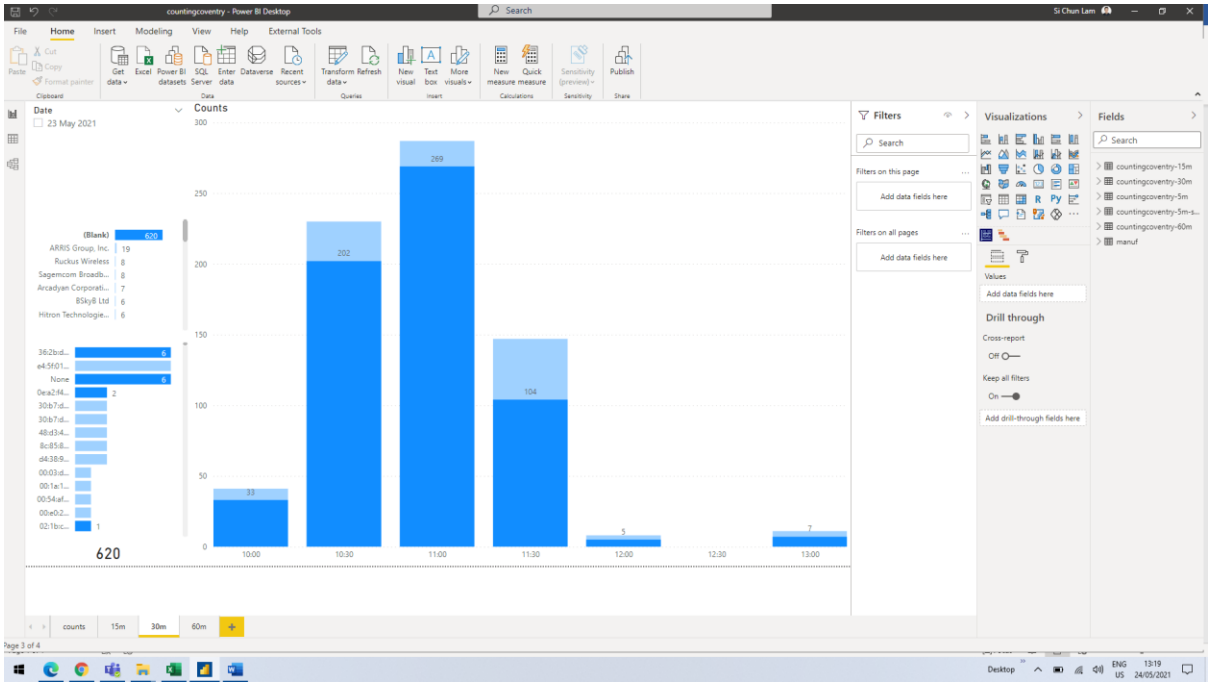
A visual inspection suggested around **150-200 parishioners** attended Sunday Mass[2].

Between 11:00-11:30, the Wi-Fi device picked up a total of **269 devices** (excluding Wi-Fi infrastructure). I was sat in the middle of the church hall (of around 30m in diameter) and Wi-Fi typically has a range of between 50m-90m – which may suggest that the device is picking up some unrelated Wi-Fi signals from outside the church hall – or multiple devices from each individual (for instance, smartwatches or tablets).



When looking at the data at 15-minute intervals, a peak of **147 devices** were picked up between 11:15-11:30 – which is closer to the estimate of 150-200 participants.

[2] There is a live-stream of the mass at https://www.youtube.com/watch?v=SgTLcOuPY20 – however the camera angle of the livestream meant it was not possible to easily compare this.



Further enhancements to the Dashboard enable us to look at multiple devices:

Conclusion

It is indeed feasible and cost-effective to use a Raspberry Pi device to estimate footfall using Wi-Fi network tracking to approximate the number of visitors to an event, activity, or venue. The Raspberry Pi device can be used to record the unique device address, the MAC address, alongside a timestamp; which can then be interpreted and analysed in different ways, for example using Power Query. By combining it with other information (e.g. known MAC address manufacturer details published by Wireshark; or reviewing dwell time) it is possible to filter out infrastructure devices such as home broadband connections.

Privacy concerns that were prominent in the early 2010s, such as associating MAC addresses with individuals or individual devices, are significantly reduced now in 2021 – as mobile device manufacturers move towards using private and randomised MAC addresses instead; without reducing the functionality of using Wi-Fi for counting purposes.

The current implementation is functional – but basic. There are potentially many avenues to develop and enhance this work with the appropriate skills and resources. This may include automating the:

- implementation, provisioning and updating process;
- automatic web-based transfer (e.g. with rclone) of the countingcoventry.csv file from a web-connected device;
- addressing remaining bugs and issues (e.g. automatic restore after power cuts/shortages; detection of the correct wlan device for monitoring, etc.)

Si Chun Lam

SiChun.Lam@coventry.gov.uk @SiChunLam

Last updated 29 May 2021