POCSAG pager decoding on the Raspberry Pi using the RTL-SDR

Jack Riley – Modified report for jrprnet.com

Project Objectives

There are many hobby groups dedicated to appreciating or exploring old, forgotten technology. It is always interesting to see progress of technical abilities over the decades. In many cases, these old technologies are the foundation of the products and services we take for granted today. For example, two of the biggest communities are the retro HiFi and computing groups, who collect, repair and preserve old computers and media formats.

The same types of people are also present in the radio community, who explore protocols and equipment that has been long superseded by newer and better technology. One such technology is the pager. In the 80's, it was a pioneer in wireless, private, instant messaging, being the first technology of its type. Today, services like SMS and the internet, of which the pager laid the foundations for, have made pagers practically irrelevant to consumers, however to some specific industries, it still plays an important part in their communication.

To attempt to gain more awareness and appreciation of the history of communication technology, the idea of a project to connect pager technology to its successors was planned. The goal of this project is to build a pager using modern radio and computing technology, and to link it to the internet using email. Through the research and development of this project, a greater understanding of the history of technology should be achieved.

Review and Discussion of Technologies Used

Pagers were communication devices primarily used during the 80's and 90's which became the predecessor of SMS messaging. They were small, portable devices capable of one-way, instantaneous messaging. First generation pagers were developed in the 1930's, which could only play tones, which correspond to words or instructions, however most peoples first idea of a pager are of the last generation models, which were capable of displaying all alphanumeric characters, as seen in Figure 1 (Bellis, 2018). At the peak of popularity in the early 90's, there were 61 million pagers in use worldwide.

Figure 1 – Typical Pager available for purchase since the 90's

When someone wishes to send a message to a pager, they would usually dial an operator using a landline telephone and dictate a message which is added into the pager queue to be transmitted over radio waves. The internet eased this process by allowing pages to be sent over email (Bellis, 2018). Each pager is programmed with a specific radio frequency and capcode, which is similar to different mobile networks (e.g. Telstra or Optus) and a phone number (SIGIDWIKI, 2018). The pager monitors the radio signal sent from the pager tower and listens for messages. When a message

address matches the capcode, the message is displayed on screen and the pager emits a loud beep to alert the user, which led to pagers also being known as “beepers”.

Although the rise of the internet, mobile networks and SMS have largely replaced pagers for consumers, pagers still have an important role today. Due to their independence from other communication networks, they are still used for emergency and backup communication. This includes dispatch notifications for emergency services including the ambulance and fire services, as well as for back to base monitoring of critical systems. One notable example of this is in the monitoring of internet nodes, as a node failure will remove the ability for alerts and messages to be sent over the internet. Pagers also see significant use in hospitals, as the technology has been tested to not interfere with sensitive medical equipment which would malfunction in presence of a mobile phone.

In Australia, apart from the local systems in place for hospitals and nursing homes, there is only one national pager system. This is the national Hutchinson (trading as Vodafone) Paging Network (HUTPDA), which has a wide coverage area that includes Brisbane. According to the product summary, customers using this network “include emergency services, hospitals, government departments, trades people and other essential services” (Vodafone Hutchison Australia Pty Ltd, 2018). Unlike SMS, which is two way communication, allowing messages to be only sent to their intended recipient, pager signals in Australia are one way, meaning all messages must be unencrypted for the pager to determine which messages are for it. As a result, it is possible to read all messages sent from the pager tower. Combined with the wide coverage of the Hutchinson pager network, it provided the perfect inspiration for this project.

In this project, pager signals will be received from the HUTPDA network. A program will act like a receiving pager, which is programmed to monitor specific capcodes. Instead of producing a beep, the program will email the received message, to modernise the pager technology. In order to complete the project, the background of the technologies used must be investigated.

An RTL-SDR was the hardware used to receive the radio signals from the pager tower. The RTL-SDR is a USB radio dongle which is a software tuner able to receive signals raging from 30-1700MHz by default and 0.5-2000MHz with modification. These receivers were initially mass produced from China as digital TV tuners (as can be seen on the shell of the device in Figure 2), until it was discovered in 2015 that the RTL2832U chipset used in the receiver could be directly interfaced with by using a custom driver to send raw I/Q (radio data) to the computer, allowing any radio application to use the dongle (Laufer, 2018).

Figure 2 – The RTL-SDR receiver. The plastic shell of the dongle has been removed so that the components can be seen

RTL-SDR stands for Realtek (the name of the company which produces the RTL2832U chip) Software Defined Radio. Unlike traditional hardware receivers, which use physical components to filter and decode signals, software defined radios like the RTL-SDR use software and digital signal processors to create more flexible decoders. This limits the hardware components required to decode radio signals. Due to the few components and the fact that the dongles are mass produced, it made the

RTL-SDR incredibly cheap (<20$) and was a major contributor to the increase in the number of people doing radio analysis as a hobby.

With the RTL-SDR and free DSP applications like SDR#, one can instantly begin to explore the airwaves. The UI of SDR# can be seen in Figure 3. Examples of transmissions that can be received include AM and FM commercial radio, digital TV and radio, air traffic control, amateur radio communication, satellite beacons, space communication (e.g. from the ISS), emergency radio (police, ambulance, fire) as well as many other digital voice and data protocols, including pager protocols.

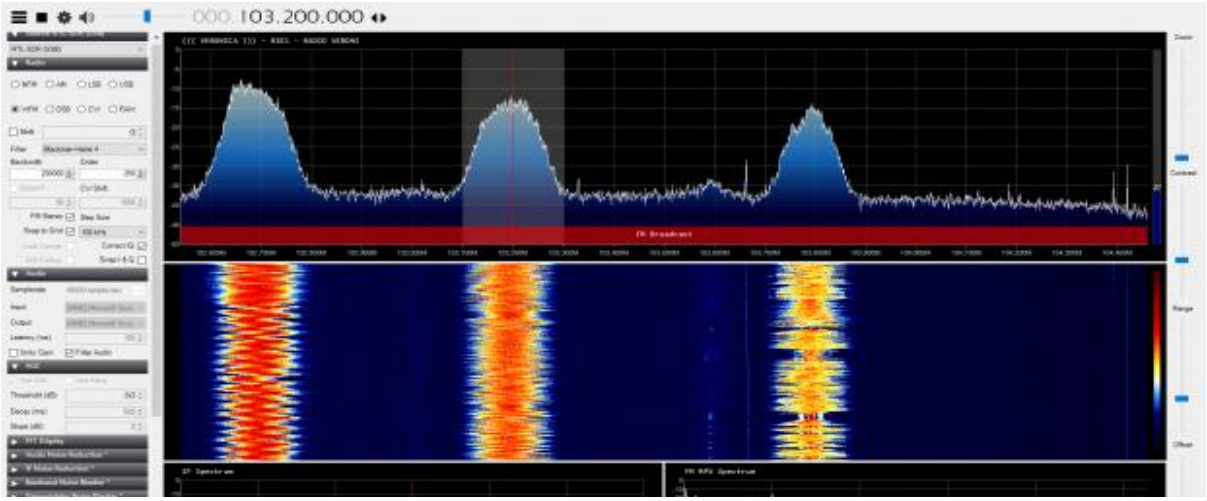

Figure 3 – The SDR# user interface receiving a commercial FM signal. This program was useful for testing and developing the project.

Although there are many more sophisticated and higher quality SDR receivers on the market like the SDRPlay RSP (which was used in the developing of this project), the HackRF or the Airspy, the RTL-SDR is by far the most cost effective and has the most support in radio applications. Even including the flaws of the receiver, it is still more than adequate for this project.

There are many different paging protocols in use worldwide including FLEX, ACARS, MOBITES and ERMES but the pager protocol in use by almost all systems in Australia (including HUTPDA) is POCSAG (SIGIDWIKI, 2018). This stands for Post Office Code Standard Advisory Group and was developed by the British Post Office in 1976. The latest version of the protocol called Super-POCSAG allows for 512,1200 and 2400bps transmissions and is backwards compatible with older versions of the protocol.

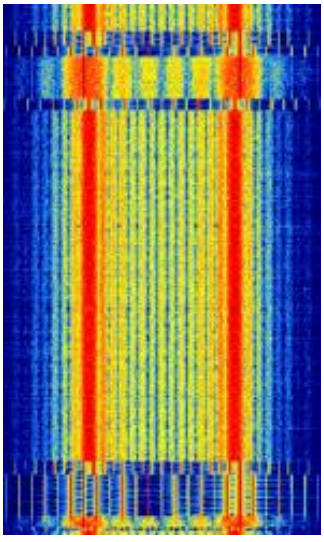

Figure 4 – Waveform of a POCSAG signal, showing the FSK encoding.

The information is sent over the radio waves with an encoding method called Frequency Shift Keying (FSK). This means that digital data is represented by the frequency shift from the centre carrier of a radio channel. In Figure 4, the digital data can be seen by the vertical red bars, which are 4.5KHz left and right of the centre. A positive shift represents a binary 0, and a negative shift represents a binary 1 (SIGIDWIKI, 2018). In Australia, POCSAG is most commonly seen in the VHF high band (138-

175MHz), which is just above commercial FM radio and air traffic control frequencies.

To decode the signals on the Raspberry Pi, two programs were used. The first is rtl_fm, which demodulates the raw radio waves into audio. It was designed for efficiency, so that it could work on low powered CPUs, unlike similar programs like GNURadio and gqrx. The audio from rtl_fm is then piped into another application called Multimon_ng. This application supports decoding of many data protocols, with POCSAG and Super-POCSAG being included (Oenal, 2012). The two applications combined use no more that 50% of the Raspberry Pi's CPU at any time.

To send emails, a protocol called Simple Mail Transfer Protocol (SMTP) is used. It is a TCP/IP protocol which is the standard for sending email messages (Mitchell, 2018). Although it was originally developed in the 80s, it is still in use today.

SMTP traditionally operated on port 25, but revisions enabled a second port 587 which supports TLS encryption (Brain & Crosby, 2018). The SMTP commands are very simple and can be readable by humans. The key commands are:

- HELO/EHLO -> Greeting. First command sent on connection and requests server features.
- MAIL -> Initiates email message
- RCPT -> Specifies destination of message
- DATA -> Email message data
- QUIT -> Ends session and disconnects

SMTP is both a client-server and server-server protocol. When a client sends an email, it is sent to the SMTP server for their domain, where it is queued for delivery. The server then looks for the SMTP server of the recipients account, establishes a connection with it and relays the information from the client (Mitchell, 2018). The recipient can then receive the email by using another protocol, such as POP3 or IMAP.

Although in this project, the messages are sent over email, it is possible to use a wide range of communication methods. This could be hardware, such as a light and buzzer or another messaging protocol, like SMS or Facebook Messenger.

Python will be the language used to process the received pager messages. It includes an SMTP library which encapsulates the TCP/IP connection and interfacing with the SMTP server. All that is required to send an email are variables for sender, recipient, message data and server details (Python, 2018). As such, a connection to the server, sending the email and disconnecting requires only three lines of code.

The topics incorporated from the IFB102 Lectures are as follows:

- Hardware: RTL-SDR radio dongle and Raspberry Pi
- Networking: SMTP protocol (TCP/IP). POCSAG pager protocol could be considered a form of primitive networking as well.
- Libraries and Languages: Python and the SMTP library. Bash was also used to a small extent.

Design and Implementation

The first step in developing the pager is to find a paging network to connect to. In Australia, there are a number of local pager systems used by hospitals, aged care facilities and restaurants, which all have limited range. Unless the receiver is close to these locations, it will be almost impossible to decode these low powered signals. However, Australia also has a national pager network still in operation which covers all capital cities and most of the east coast. This is the Vodafone Hutchison network.

To find the radio frequency that the pager network operates on, a quick search on the Australian Communications and Media Authority's (ACMA) *Register of Radiocommunications Licences* database was performed. Here it was possible to search for frequencies licenced to Hutchison. Figure 5 shows the result of the search, displaying a frequency of 148.6375MHz.

Figure 5 - Result of the ACMA search for the frequency of the Hutchison pager network

It was confirmed that the service was active and in range by attempting to "see" the signal. For any development or testing not on the Raspberry Pi, the SDRPlay RSP1 was used over the RTL-SDR. Using SDR# to visualise the airwaves, a strong signal was seen, which should be powerful enough to receive on the smaller RTL-SDR. The signal was confirmed to be POCSAG by looking at the FFT plot, as seen in Figure 6.

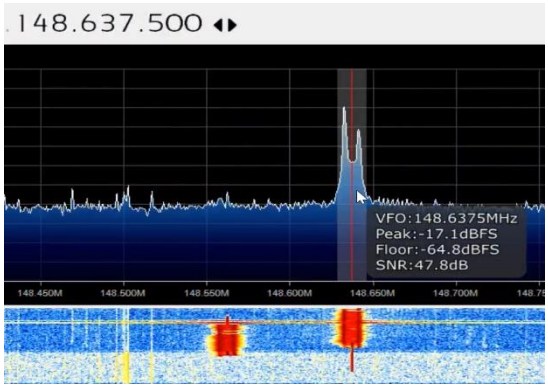

Figure 6 - Screenshot of the SDR# window tuned to the frequency of the pager network, showing a strong pager signal which will be perfect for decoding

Originally, the plan was to use a piece of software called GNURadio to decode the pager messages. GNURadio is a free, professional signal processing program. By dragging and dropping individual blocks (each with their own function) into a flowchart, any signal can be decoded. It was intended to perform demodulation of the FSK using GNURadio to output binary into a script that would convert the binary into text.

Before installing anything, it is best practice to update the repository's, so that the latest version of software is installed:

sudo apt-get update

Installing GNURadio was simple. A single command was all that was required:

sudo apt-get install gnuradio gnuradio-dev

This took a significant amount of time as a number of dependencies also had to be downloaded and installed.

To use the RTL-SDR with the Raspberry Pi, the drivers had to be installed. Again, this was a simple process, requiring only one command:

sudo apt-get install rtl-sdr gr-osmosdr

GNURadio could be run by typing gnuradio-companion in a terminal.

The first test with GNURadio was a failure. Although flowcharts could be edited, any attempt to execute the flowchart resulted in the program crashing. The error message was ***"Error in '/usr/bin/python': corrupted double-linked list"***. Numerous troubleshooting steps were taken to diagnose the issue, with no results. The issue seems to be relatively common as there are multiple forum posts about the issue, however none have suggestions on how to solve the problem. As a last resort, the Raspbian OS was replaced with Ubuntu, which ended up working.

Figure 7 shows the flowchart that was designed. When monitoring the processed signal before the Binary Slicer, which is responsible for converting the signal into binary data, it could be seen (Figure 8) that there was a square wave that represented binary data. The binary depicted in the waveform corresponded with the expected structure specified in the protocol, however the binary slicer spat out garbage on the other side. It was noticed that when the binary slicer is running, the console of GNURadio was printing "O" repetitively. Researching the symptoms identified that GNURadio was running out of CPU resources and was unable to properly decode the signals. This is because GNURadio is a resource heavy program and was not designed to run on lightweight hardware like the Raspberry Pi. This meant an alternative program had to be used to receive the pager messages.

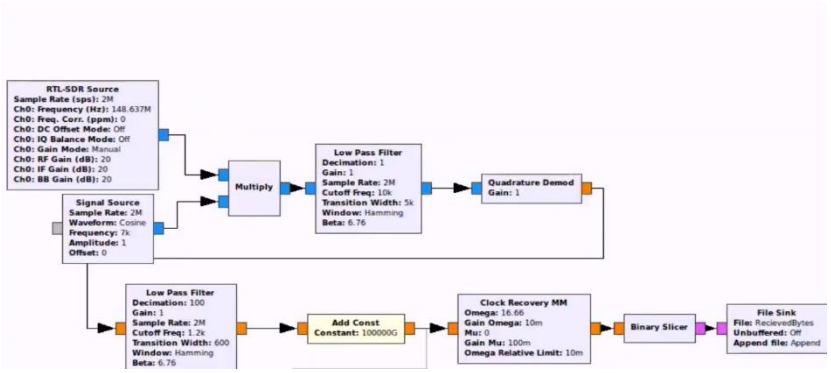

Figure 7 - The GNURadio flowchart that was the first attempt at decoding the POCSAG protocol

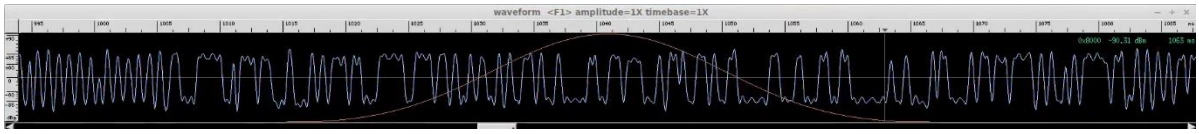

Figure 8 - The waveform of the isolated signal before being converted to binary. The preamble of the message is seen at the left, followed by the synchronisation codeword.

Fortunately, there are two programs that were developed for this reason. They are rtl_fm and multimon_ng, both of which are Linux applications designed to operate with few resources, making them perfect for use with the Raspberry Pi. Rtl_fm was already installed as part of the driver packages, meaning only multimon_ng was left to install. This required compiling from the source code, which is more difficult than using apt, but still isn't too hard.

First, the source code had to be downloaded from github:

git clone http://github.com/EliasOenal/multimonNG

Next, the source code was compiled and installed

mkdir build
cd build
qmake ../multimon-ng.pro

make

sudo make install

Rtl_fm and multimon_ng work by piping the decoded audio between the two applications. This is similar to redirecting the output from a program to a text file, which will be used on the output from multimon_ng.

The command to demodulate audio from rtl_fm is:

/usr/bin/rtl_fm -f [Frequency] -s [Sampling Rate]

The command for decoding the pager messages with multimon_ng is:

multimon-ng -t raw -a [Protocol 1] -a [Protocol 2]… -e -f alpha /dev/stdin > [Output File]

Where the protocols are POCSAG512, POCSAG1200 and POCSAG2400, representing the three data speeds possible.

Combining the two commands to redirect the output of rtl_fm into multimon_ng and the output of multimon_ng into a text file gives the command to run.

/usr/bin/rtl_fm -f 148.631M -s 22050 | multimon-ng -t raw -a POCSAG512 -a POCSAG1200 -a POCSAG2400 -e -f alpha /dev/stdin >pager.txt

The first time this was run, nothing happened. It was identified that no pager signal was received. Using SDR# and comparing the RTL-SDR to the higher quality RSP1 showed that the aging RTL-SDR had a significant drift in frequency and therefore the pager signal was not where it was tuned. Figure 9 shows how the frequency drift was cutting off half the pager signal. This was counteracted by down tuning the RTL-SDR until the centre carrier of the pager signal was in line. This was determined to be 148.631MHz.

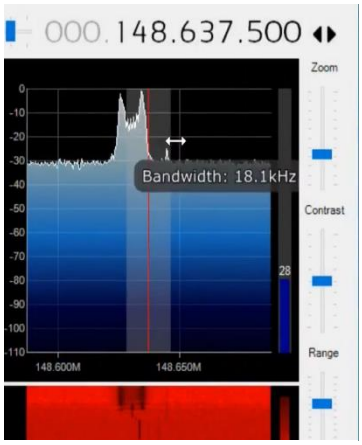

Figure 9 - The frequency drift of the RTL-SDR which caused no messaged to be decoded on the first trial.

After substituting the corrected frequency into the Linux command, pager messages started to appear in the text file. A sample of the output is seen in Figure 10. Each message is formatted in the text file in the order of protocol, address (capcode) and message. Since each line is consistently formatted, the capcode and message can be easily extracted from the text file using string manipulation.

```
POCSAG1200: Address:  209549  Function: 2  Alpha:   0905-20:46 #:10093802,1B,06D02 ,DIFF SPEAK BETW BREATHS 6D02 , , , , ,[redacted] ,EVERTON PARK ,B118
POCSAG1200: Address:  604504  Function: 2  Alpha:   20:46 P406740.HUTMSG.MON.F2<NUL>
POCSAG512: Address:  444389  Function: 0  Alpha:   Your time at Hospital has exceeded 30mins, please contact SOS (or OCS if after hrs) to advise of delays, T
POCSAG1200: Address:  544681  Function: 0  Alpha:   20:46 P90394.BDA2
POCSAG1200: Address:  446162  Function: 0  Alpha:   1108<NUL>
POCSAG512: Address:  857521  Function: 0  Alpha:   20:46 P4828436..HUTPDA2<NUL><NUL>
POCSAG512: Address:  540575  Function: 0  Alpha:   #10093463 , R:18:24, DI:20:23, RES: , STAGED: , SCENE:20:23, AT PT: , DEP:20:37, DEST:20:39
POCSAG512: Address:  913233  Function: 2  Alpha:   #10093654, R:19:39, DI:19:40, RES:19:52, STAGED: , SCENE:19:57, AT PT: , DEP:20:31, DEST:20:46
POCSAG512: Address:  578520  Function: 2  Alpha:   FIRE CALL<NUL><NUL>
POCSAG512: Address:  910310  Function: 2  Alpha:   SEQ 274031<NUL>
POCSAG1200: Address: 1008516  Function: 2  Alpha:   #:10093210,4B, ,MATB6 ,OFAKI AFEAKI ,PAH (ARTS DIALYSIS 2731/2996) ,5537 , ,[redacted] ,WOOLLOONGABBA ,B1
POCSAG1200: Address:    1710  Function: 2  Alpha:   20:47 P175666.MDA
POCSAG1200: Address:  604504  Function: 2  Alpha:   20:47 P406740.HUTMSG.MON.F2<NUL>
POCSAG1200: Address:  544681  Function: 0  Alpha:   20:47 P90394.BDA2
POCSAG512: Address:  857521  Function: 0  Alpha:   20:47 P4828436..HUTPDA2<NUL><NUL>
POCSAG1200: Address:  540482  Function: 0  Alpha:   Subj: Alert: Failed to Connect to Computer Resolution state: New, Msg: Alert: Failed to Connect to Comput
POCSAG1200: Address:  540482  Function: 0  Alpha:   Subj: Alert: Failed to Connect to Computer Resolution state: New, Msg: Alert: Failed to Connect to Comput
POCSAG1200: Address:  867168  Function: 0  Alpha:   X 7141 09/05/18 20:48:18<NUL>
POCSAG1200: Address:  408324  Function: 2  Alpha:   Name: SHARNIE, Ph: DECLINED, Message: SORRY MISSED YOUR CALL-FROM YOUR NIECE<NUL>
POCSAG1200: Address:  867168  Function: 0  Alpha:   X 7141 09/05/18 20:48:24<NUL>
POCSAG512: Address:  910310  Function: 2  Alpha:   SEQ 274032<NUL>
POCSAG1200: Address:  451370  Function: 2  Alpha:   #:10093818,2A,32B02 ,UNK PROB COMM ALARM NO PT INFO, , , , ,[redacted] ,ROCKVILLE ,TWB57A20, , ,<NUL>
POCSAG1200: Address:  490827  Function: 2  Alpha:   Mt Helena BFB TO Inspect a controlled burn [redacted] MOUNT HELENA X EALY 237F-8 389760 Please Ack 1800
POCSAG1200: Address:  604504  Function: 2  Alpha:   20:48 P406740.HUTMSG.MON.F2<NUL>
POCSAG1200: Address:    1710  Function: 2  Alpha:   20:48 P175666.MDA
POCSAG1200: Address:  544681  Function: 0  Alpha:   20:48 P90394.BDA2
POCSAG512: Address:  857521  Function: 0  Alpha:   20:48 P4828436..HUTPDA2<NUL><NUL>
POCSAG1200: Address:  425260  Function: 2  Alpha:   #:10093820,1C,06D04 ,BREATH PROB CLAMMY , ,QPS LOGAN CENTRAL POLICE BEAT , , ,[redacted] ,LOGAN CENTRAL
```

Figure 10 - Several decoded pager messages. Personal information such as names, addresses and phone numbers have been blurred out.

The language used to process the received messages and send emails was python. Developing the code was a simple process. The structure of the text file containing messages was studied to identify the text patterns. Every 2 seconds, the script checks for new lines in the file. When a new message is

received, the capcode and message are extracted and the capcode is compared to a comma separated list of capcodes imputed as variables by the user. If a match is found, an email containing the message is sent to the email address specified by the user. For a more in-depth explanation of the python script, refer to the comments in the code.

The final step in developing the pager was to combine the message decoding and python script together. The best way to do this was to use a bash script. To suppress the diagnostic output of rtl_fm and multimon_ng, it was executed in a null terminal, which means that the decoding of the pager messages runs in the background. The user only sees the output of the python script, which is a user-friendly command line interface.

This is the bash script that was created:

```
#!/bin/bash
{ /usr/bin/rtl_fm -f 148.631M -s 22050 | multimon-ng -t raw -a POCSAG512 -a
POCSAG1200 -a POCSAG2400 -e -f alpha /dev/stdin >pager.txt; } &>/dev/null &
python3 pager.py
```

The following is a screenshot of the completed product in action. A video of the program working can be found at https://youtu.be/8tqZXFEvxJU?t=5m50s. It is also recommended that that the whole video is watched, as it complements this report.

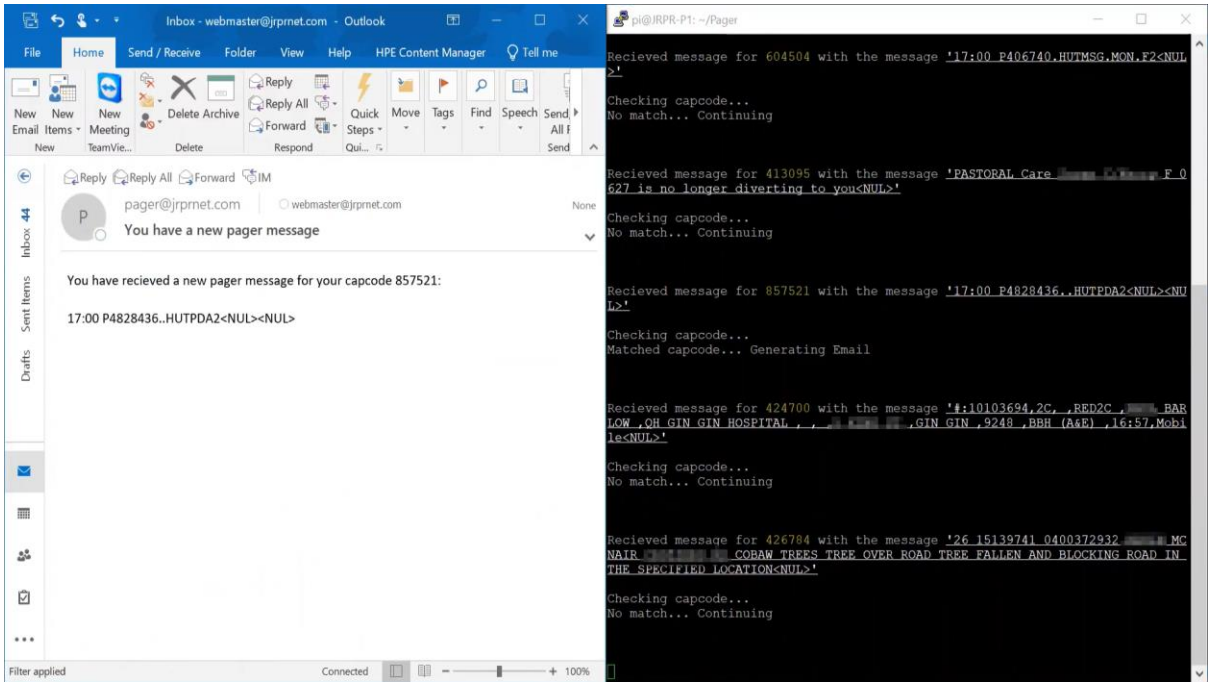

Figure 11 - Screenshot from the demonstration video showing the pager generating emails. On the right hand side of the screen is the user interface showing the pages received and the left hand side shows an email generated and sent by the program.

There are several improvements or suggested extensions to this project. First, alerts are not limited to email. It is possible to send alerts over any platform, including Facebook Messenger or Twitter for example. One could also have a physical alerting device such as a light and speaker and use text to speech technology to speak the message out.

Although the python script works and has some error handling inbuilt, it still isn't perfect. This is most notable when network issues occur. If the connection times out, either the program will freeze and indefinitely attempt to send an email, or it will crash due to a DNS lookup failure. This could be solved by adding timers to interrupt the attempt after a certain period of time. Furthermore, in cases where emails are not sent, there is currently no record of failed attempts. A cache could be implemented to store undelivered messages and attempt to send them when network issues are resolved. This would mean no messages would be lost.

Finally, an interesting extension to this project would be to convert the email portion of this program to a website. The decoded pager messages would be uploaded to a database, which would allow users to search for previous messages, or login with a username and "follow" certain capcodes to create a personalised message "feed". To implement this, a HTTP server would be required along with a SQL database and a scripting language like PHP or Javascript.

References

Bellis, M. (2018, April 3). *History of Pagers and Beepers*. Retrieved from ThoughtCo: https://www.thoughtco.com/history-of-pagers-and-beepers-1992315

Brain, M., & Crosby, T. (2018). *How E-mail Works*. Retrieved from HowStuffWorks: https://computer.howstuffworks.com/e-mail-messaging/email3.htm

Keen, K. (2018). *Rtl_fm Guide*. Retrieved from kmkeen: http://kmkeen.com/rtl-demod-guide/

Laufer, C. (2018). *About RTL-SDR*. Retrieved from RTL-SDR.com: https://www.rtl-sdr.com/about-rtl-sdr/

Mitchell, B. (2018, February 25). *Guide to Simple Mail Transfer Protocol (SMTP)*. Retrieved from Lifewire: https://www.lifewire.com/definition-of-smtp-817975

Oenal, E. (2012, May 24). *multimonNG*. Retrieved from Elias' Blog: https://eliasoenal.com/2012/05/24/multimonng/

Python. (2018). *smtplib - SMTP protocol client*. Retrieved from Python Docs: https://docs.python.org/2/library/smtplib.html

SIGIDWIKI. (2018, May 4). *POCSAG*. Retrieved from Signal Identification Wiki: https://www.sigidwiki.com/wiki/POCSAG

Vodafone Hutchison Australia Pty Ltd. (2018). *Vodafone Paging Services*. Retrieved from Vodafone: https://www.vodafone.com.au/messaging/paging

Programs Used

Development:

SDR# - Windows RF Analyzer Software https://airspy.com/download/
PuTTY - Serial and SSH client https://www.putty.org/
Baudline - Signal Analyzer http://www.baudline.com/

Final Product:

Python 3 https://www.python.org/
rtl_fm - Linux NFM demodulator http://kmkeen.com/rtl-demod-guide/
multimon_ng - Data decoder https://github.com/EliasOenal/multimon-ng