RFI Monitoring for the MeerKAT Radio Telescope

Christopher Schollar

Department of Computer Science

University of Cape Town

Thesis submitted for the degree of Master of Science

February 2015

Supervisors:

Sarah Blyth

Michelle Kuttel

Anja Schroeder

Plagiarism Declaration

I know the meaning of plagiarism and declare that all of the work in this thesis, save for that which is properly acknowledged, is my own.

Christopher Schollar

Abstract

South Africa is currently building MeerKAT, a 64 dish radio telescope array, as a pre-cursor for the proposed Square Kilometre Array (SKA). Both telescopes will be located at a remote site in the Karoo with a low level of Radio Frequency Interference (RFI). It is important to maintain a low level of RFI to ensure that MeerKAT has an unobstructed view of the universe across its bandwidth. The only way to effectively manage the environment is with a record of RFI around the telescope.

The RFI management team on the MeerKAT site has multiple tools for monitoring RFI. There is a 7 dish radio telescope array called KAT7 which is used for bi-weekly RFI scans on the horizon. The team has two RFI trailers which provide a mobile spectrum and transient measurement system. They also have commercial handheld spectrum analysers. Most of these tools are only used sporadically during RFI measurement campaigns. None of the tools provided a continuous record of the environment and none of them perform automatic RFI detection.

Here we design and implement an automatic, continuous RFI monitoring solution for MeerKAT. The monitor consists of an auxiliary antenna on site which continuously captures and stores radio spectra. The statistics of the spectra describe the radio frequency environment and identify potential RFI sources. All of the stored RFI data is accessible over the web. Users can view the data using interactive visualisations or download the raw data. The monitor thus provides a continuous record of the RF environment, automatically detects RFI and makes this information easily accessible.

This RFI monitor functioned successfully for over a year with minimal human intervention. The monitor assisted RFI management on site during RFI campaigns. The data has proved to be accurate, the RFI detection algorithm shown to be effective and the web visualisations have been tested by MeerKAT engineers and astronomers and proven to be useful. The monitor represents a clear improvement over previous monitoring solutions used by MeerKAT and is an effective site management tool.

Contents

Acronyms

ADC – Analogue to Digital Converter

AGA – Astronomy Geographic Advantage act

ASIC – Application Specific Integrated Circuit

CSS – Cascading Style Sheet

CSV – Comma Separated Value

CUDA – Compute Unified Device Architecture

DBMS – Database Management System

DSP – Digital Signal Processing

FD – Frequency Domain

FPGA – Field Programmable Gate Array

GMRT – Giant Metrewave Radio Telescope

GPS – Global Positioning System

GSM – Global System for Mobile Communications

GUI – Graphical User Interface

HDF – Highly Definable Format

HTML – Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

ICASA – Independent Communication Authority of South Africa

IP – Internet Protocol

ITU – International Telecommunications Union

KAT – Karoo Array Telescope

KCAAA – Karoo Core Central Astronomy Advantage Area

LAN – Local Area Network

LOFAR – Low Frequency Array

MAD – Median Absolute Distance

MMS – Mobile Measurement System

MPI – Message Passing Interface

PAPER – Precision Array for Probing the Epoch of Reionization

PC – Personal Computer

PDF – Portable Document Format

PNG – Portable Network Graphics

RA – Regulatory Article

RAS – Radio Astronomy Service

RATTY – Real Time Transient Analyser

RF – Radio Frequency

RFI – Radio Frequency Interference

ROACH – Reconfigurable Open Architecture Computing Hardware

RQZ – Radio Quiet Zone

SARAS – South African Radio Astronomy Service

SKA – Square Kilometre Array

SKA SA – Square Kilometre Array South Africa

SNR – Signal to Noise Ration

SQL – Structured Query Language

TCP – Transmission Control Protocol
TD – Time Domain
XHTML – Extended Hypertext Markup Language

1 Introduction

In 2012 South Africa and Australia were chosen to host the Square Kilometre Array (SKA) radio telescope. The telescope, an array of many antennas, will be the largest and most sensitive radio telescope ever built. It is planned to be built over the coming decade. The National Research Foundation (NRF) has created a business unit called Square Kilometre Array South Africa (SKA SA) to construct a precursor instrument called the MeerKAT telescope on the South African site of the planned SKA telescope. Helping to ensure a clear view of the sky for the MeerKAT and SKA telescopes is the motivation for this thesis.

The telescopes observe radio waves emitted by astronomical objects. Man-made sources of radio waves can interfere with observations. We call this radio frequency interference (RFI). The main sources of RFI are modern telecommunications and satellites, although all electronics produce some RFI. In order to avoid RFI, these telescopes are situated in a remote area of the Karoo. The relative radio quietness of this site is a major reason why South Africa was awarded a large part of the SKA telescope. Maintaining this radio quietness is an important part of the SKA project. If the area around the telescope becomes too noisy, some science goals of the telescope will become more difficult or even impossible.

Over the years radio telescopes have become more sensitive while the use of radio for communication and of electronics in general have become ubiquitous. As such RFI has become increasingly problematic for radio astronomy. Many strategies to protect radio telescopes from RFI have been developed. These approaches are grouped under the term RFI mitigation. They include laws preventing the use of certain electronics around telescopes, monitoring RFI, avoiding RFI during observations, detecting RFI in observations, removing corrupted data automatically from observations. This thesis focuses on providing a RFI monitoring system for the MeerKAT site.

1.1 Motivation

RFI monitoring makes other aspects of RFI mitigation easier. It simplifies location and removal of RFI sources, making legislative protection of radio telescopes enforceable. Avoiding RFI is only possible if it is well characterised, especially in terms of direction and frequency. Finally, good knowledge of the properties of RFI on site can help in the process of RFI detection and removal from collected data.

At the beginning of this thesis, the SKA SA team had many tools which allowed measuring RFI on the MeerKAT site. These included a 7 dish radio telescope called the KAT7 telescope, commercial hand held spectrometers and special built transient RFI measurement systems. However the team did not have an automatic, continuous RFI monitoring system. This meant that some RFI was not measured as it occurred outside of the times those tools were used. It also meant that there was no standard, easy way to monitor RFI in conjunction with an observation.

Although it was technically feasible to fulfil most RFI monitoring with the tools the team already had, setting up on site RFI monitoring systems involved significant use of man hours. There was no dedicated RFI monitoring team to perform these tasks or to analyse the data. Adequately monitoring RFI with existing tools would be an inefficient use of the SKA SA's available technical skills. There was a clear need for a system which could continuously monitor the environment and automatically detect RFI. This thesis describes the development of such an RFI monitoring system.

1.2 Aims

The main aim of this thesis was to provide an automatic RFI monitoring system for the MeerKAT site. This RFI monitor had to fulfil three criteria: to provide a continuous record of the Radio Frequency (RF) environment, automatic RFI detection using this data and effective access to the collected data.

Consequently a record of the RFs on site allows site administrators and engineers to understand how new equipment on site affects the radio quietness of the site. Automatic RFI detection allows RFI management to know when unusual sources of RFI are present. Providing effective access makes it possible for this information to be quickly digested and acted upon by astronomers, engineers and site management.

1.3 Approach

The approach used in this thesis was to build iterative prototypes, each prototype adding new features to the monitor. This approach was chosen so that the features of the RFI monitor could be tested by stakeholders before the final monitor was finished. The main benefit was the continued support of stakeholders who had the opportunity to point out problems early on. Having basic functionality from early in the thesis also allowed for the most basic functionality to be well tested over long periods of time. This provided insights into issues that were not obvious initially and allowed the design to be shaped by actual use of the RFI monitor.

1.4 Contribution

The RFI monitor created in the completion of this thesis provided the SKA SA team with a continuous, automatic RFI monitoring system. This provides a new tool for managing RFI. It has been used multiple times in testing the effect of equipment on site, corroborating tests performed with other systems and in revealing RFI before other available RFI monitoring could have.

The RFI monitor was always intended to be a prototype system. Due to hardware restraints, it could not cover the bands of the MeerKAT telescope. Since the completion of the final prototype, a new, upgraded monitoring system has been developed using the software and experience developed in this thesis. The new monitoring system, which covers the whole MeerKAT band, has recently been installed on site.

In the interests of clarity, I state here the parts of the monitor which I did not develop. The Real Time Transient analYser (RATTY), described in section 2.5.3 was developed by Jason Manley. The Median Absolute Distance Thresholding algorithm (section 2.6.2) was suggested to me by Sean Passmoor, however I developed the code. Jason and Sean are both employees of SKA SA. The Javascript bar chart visualisation (section 4.3.1.1) is developed by a company called amCharts. All other work is my own.

1.5 Thesis Overview

The structure of this thesis is as follows.

Chapter 2 contains the background necessary to understand the motivation and technology in this thesis. Sections 2.1 and 2.2 cover some basic ideas in Astronomy, Radio Astronomy and Radio Telescopes. Section 2.3 covers Radio Frequency interference and 2.4 covers the different ways in which RFI can be mitigated. Section 2.5 focuses on RFI monitoring at the MeerKAT site. Section 2.6 covers RFI detection. Finally 2.7 Covers the technology we used in providing our RFI monitor.

Chapter 3 covers the Design process of the RFI monitor. This includes specifying the goals, defining the users and their requirements and discussing the limitations of our equipment. We state the design approach and the design decisions for the different prototypes that were developed, including the initial data capture, archival, RFI detection and visualisation prototypes.

Chapter 4 covers the implementation of the RFI monitor. This chapter follows the development of each of the four prototypes. Rather than describing the databases developed for each prototype, a section is dedicated to all the databases for each prototype. The final part of this chapter shows how each prototype fulfilled the goals and user requirements defined in the design chapter.

Chapter 5 contains the validation and testing. This covers the reliability of the measurements, the efficiency of the database storage and retrieval, the accuracy of the RFI detection algorithms and the user test of the visualisations and web site which provide access to data. This is followed by out Conclusions and further work in Chapter 6.

2 Background

2.1 Astronomy

Objects in the universe radiate electromagnetic waves/particles. These waves/particles, travelling at the speed of light, are the only evidence we have that the universe is populated with countless stars, galaxies, planets and other astronomical objects. Observational astronomy is the study of electromagnetic emission from objects in space using instruments called telescopes. Astronomy includes some of the most interesting problems in modern science, such as searching for earth-like planets, searching for extra-terrestrial life, and learning about the structure of planets, stars, galaxies, the early universe and how they evolve over time (Burke, 2010). Astronomy is also used as a way of investigating fundamental forces such as the theories of relativity and particle physics (Chaisson & McMillan, 2005).

Observing the skies has been something of an obsession for human beings for millennia. Before there was any kind of professional astronomer, people were observing the sky and learning about how the stars and planets moved and how this related to the Earth. Ancient people built observatories that could be used to tell important times in the year from the positions of celestial bodies. Observation was performed purely through human eyes which was the cutting edge of astronomy up until the beginning of the scientific revolution. One of the last great astronomers who performed all of his observations by the power of his eyes was Tycho Brahe. His observations of Mars were instrumental in formulating and supporting Kepler's laws of planetary motion which were the precursor for Newton's laws of gravity.

That these fundamental physical concepts could be explored using nothing more than the naked eye and what seems like rudimentary equipment is impressive. However the astronomy we perform today is far removed from charting the location of astronomical objects which was the essential practice of observatories in the past. There are two things which allowed astronomers to move past these ancient practices.

The first was the application of lenses to studying the sky. In the early 17th century when the first telescopes were pointed at the sky they revealed many objects which had never before been visible. From those times one of the main goals of astronomers has been to build ever larger telescopes in the hopes that the more sensitive telescopes will allow us to better see and understand astronomical objects. However good these telescopes were, they did not allow astronomers to delve into the nature of the objects they studied. All they could do is find the position and relative brightness of these objects to ever higher levels of accuracy.

The next revolution in astronomy was the understanding that the relative strength of the frequencies emitted by celestial could help us understand the nature of those celestial objects. By studying the frequencies emitted by an object you can deduce the chemical composition, temperature, density, mass, distance and relative motion of that object. The study of splitting up light into its constituent frequencies is called spectroscopy.

The form of electromagnetic radiation humans are most familiar with is visible light, however astronomical objects typically radiate over a range of wavelengths which are invisible to the naked eye in addition to visible light. For the rest of this thesis we shall use the term light to mean all electromagnetic radiation, regardless of whether it is visible to the human eye. Accelerating electric charges produce light which can travel through the vacuum of space for millennia. The dual nature of light means that its behaviour can be described as both a wave and as a particle (called a photon). These particles/waves transmit energy that is collected by telescopes to create an image or spectrum of astronomical objects (Chaisson & McMillan, 2005). We provide a graphical representation of the electric and magnetic fields that constitute a light wave in Figure 1.

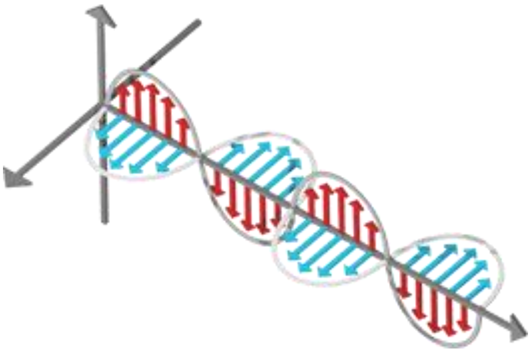


Figure 1 - Electromagnetic waves are formed by vibrations of electric and magnetic fields. The vibrations in these fields are perpendicular to each other and the direction of the wave's propagation. The figure depicts the magnetic field in blue, the electric field in red and the direction of propagation as moving towards the right.

Figure source: http://missionscience.nasa.gov/images/ems/emsAnatomy_mainContent_EMwave.png

Not all light is equal; we talk about light being a spectrum. Some light is very energetic and can be harmful to us; other light is very low powered and can be safely used for communications technologies like radio-waves. We use 3 related terms to describe where on the spectrum a particular photon of light belongs. These terms are wavelength, frequency and energy. Figure 2 is a graphic representation of the spectrum and these related ideas. The figure shows that the wavelengths of light span from the order of hundreds of metres, to incredibly small in the order of nanometres or less.

We can imagine that a photon is a packet of electromagnetic energy with $E = hf$ where h is Planck's constant f is frequency (Chaisson & McMillan, 2005); it travels through space until it hits our telescope's receiver. For radio telescopes the electromagnetic energy is converted into electrical energy which we can digitise and analyse. We describe photons using the standard unit of the electron volt (eV). Frequency is inversely proportional to wavelength. This means that a large wavelength of light will be transmitted by low energy photons with a low frequency (Chaisson & McMillan, 2005).

Observational astronomers spend their careers collecting and analysing photons which have been emitted by astronomical objects. Some photons reach Earth many billions of years after they were first emitted. The energy in light emissions at a receiver will be inversely proportional to the distance that emission has travelled from its source. This is called the inverse square law and it is for this reason that distant astronomical objects are incredibly faint and need very sensitive telescopes in order to be observed.

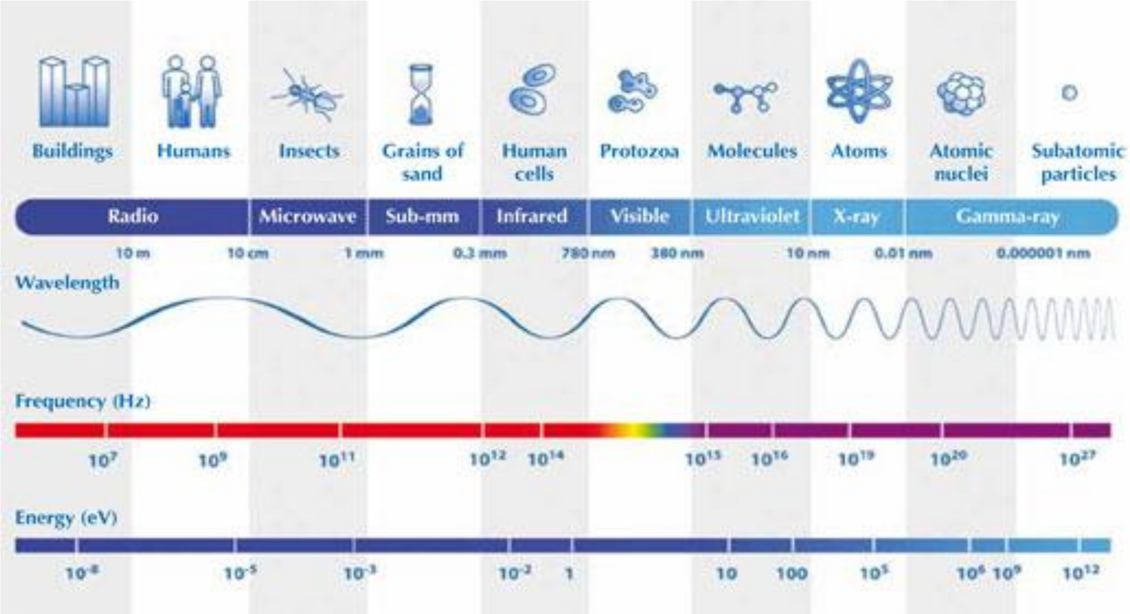


Figure 2 – The electromagnetic spectrum. The figure above shows the wavelengths, frequencies and energy electromagnetic waves across the spectrum. The wavelengths are compared to objects of a similar size. Figure source: http://www.scienceinschool.org/repository/images/issue20em2_l.jpg

Although the distance that light has travelled should not affect the frequency of that light, there is a way in which the wavelength that we observe is different from the wavelength that was emitted by the source. These changes are similar to the **Doppler** Effect we see in sound waves. The relative velocity of the source to the observer changes the frequency of the light observed. Objects emitting light at the same frequency will be observed to emit at different frequencies if their motion relative to the observer is different. Objects moving away from the observer will appear to emit at longer wavelengths and objects moving towards the observer will appear to emit at shorter wavelengths. In astronomy we call these situations **red shift** (moving away, longer wavelengths) and **blue shift** (moving towards, shorter wavelengths) (Chaisson & McMillan, 2005). The collective term for both is called the **Doppler shift**.

Studying the spectrum of emission and absorption of astronomical objects allows us to categorise them and understand them. These emissions are separated into two distinct types for our purposes. Firstly, there are **continuum emissions** which are electromagnetic emissions which vary continuously with frequency, otherwise known as **broadband emissions**. The energy from astronomical objects is spread across all frequencies.

One form of continuum emission is **blackbody** emission. Blackbodies are objects which absorb light and emit that energy on all frequencies at the same rate that they receive energy, thereby maintaining a constant temperature (Chaisson & McMillan, 2005). The amount of radiation emitted on each frequency is dependent on the temperature of the object. In Figure 3 we see example emissions for blackbodies at different temperatures. The figure shows that a blackbody will have a peak frequency on which most energy is emitted. Warmer objects have more energy and emit more energy on higher energy (shorter) wavelengths, they will also have a higher peak frequency. Stars and plasma are examples of objects which can be approximated as blackbodies (Miller, 1998).

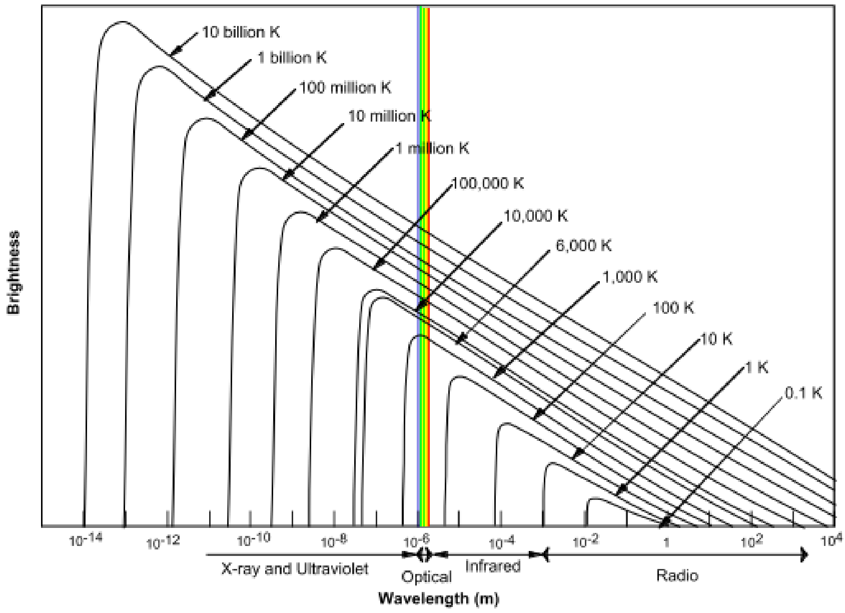


Figure 3 - Brightness of electromagnetic radiation at different wavelengths for blackbody objects at various temperatures. Astronomers can use the peak frequency of an object to deduce the temperature of that object.

Figure source: http://astronomyonline.org/Science/RadioAstronomy.asp

The second type of emission that spectral astronomy categorises is **spectral line emission**. Spectral lines are a result of discrete changes in the energy states of atoms/molecules. They can be divided into two forms, absorption and emission lines. Since atoms/molecules have discrete energy states, to change from one energy state to another it needs to either absorb or emit exactly the amount of energy that separates two states. Therefore the only photons that can be either emitted or absorbed by an atom/molecule are those photons that have the same quantum of energy as the difference between two energy states of the atom/molecule (Chaisson & McMillan, 2005).

This means that all molecules have a distinct spectral signature (Chaisson & McMillan, 2005). In the case of emission lines we see radiation which is emitted due to atoms moving from higher energy to lower energy states. For absorption lines, a gas is in between a source of continuum emission and the observer. When we observe the continuum emission we see discrete gaps where those frequencies are being absorbed by the gas.

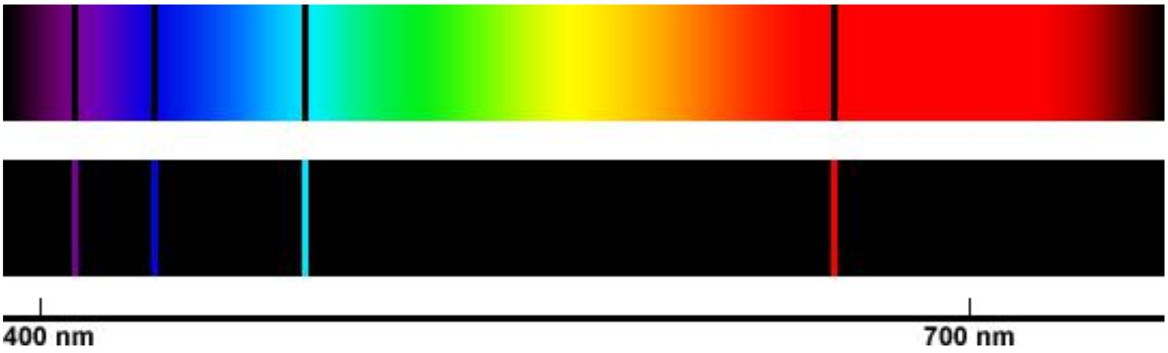


Figure 4 - Spectral lines of Hydrogen. The bar shows absorption lines and the bottom bar shows emission lines

Figure source: http://jtgnew.sjrdesign.net/images/spectra_hydrogen.jpg

For any given element or element/molecule the frequencies of absorption and emission lines are identical. The set of spectral lines of every element/molecule in the laboratory creates a unique signature similar to Figure 4. By understanding the spectral lines of molecules on Earth, we can infer the elements that make up astronomical objects and the interstellar medium based on the spectral lines in their emission. We can also learn about the temperature, pressure, relative motion and magnetic properties of objects based on the structure of their spectral lines (Tennyson, 2010), as all of these properties cause observable effects on spectral lines.

2.2 Radio Astronomy

Radio astronomy is the study of astronomical objects and phenomena which are observable in the radio part of the electromagnetic spectrum. Radio waves are low energy, low frequency and long wavelength electromagnetic waves. There are good reasons to observe in the radio spectrum. The first is that some astronomical objects emit most radiation in the radio part of the spectrum, if we did not observe radio waves we would not know about these objects. Some examples of objects which are observable in different parts of the spectrum are provided in Table 1. One can see that there are some objects, like interstellar gas, which are only commonly studied using radio.

Another reason is that wavelengths of light are absorbed by the Earth's atmosphere before reaching the ground. This means that the only way to observe those wavelengths is by placing a telescope outside of the Earth's atmosphere. However a large part of the radio spectrum can pass through the atmosphere (Miller, 1998). This "radio window" allows us to build large radio telescopes on Earth which can be orders of magnitude larger than anything that could be launched into space.

For the last 50 years astronomers have measured radio waves with instruments called radio telescopes, which are generally composed of a receiver and a dish to focus radio waves from an area onto that receiver. The larger the dish, the more radio waves that it can catch and focus on the receiver which means the telescope can detect weaker signals. The scale of these radio telescopes has been increasing since their inception in the 1950s (Burke, 2010; Chaisson & McMillan, 2005; Dewdney, Hall, & Schilizzi, 2009; National Astronomy and Ionosphere Center, n.d.). This growth in the size of telescopes culminated with the world's largest single dish telescope in Arecibo which has a diameter of 305m (National Astronomy and Ionosphere Center, n.d.). Arecibo is soon to be replaced as the largest single dish telescope by the planned Five-hundred-meter Aperture Spherical radio Telescope (FAST) being constructed in Guizhou Province in southern China (Quick, 2011).

Astronomers would like to have more sensitive telescopes than Arecibo or FAST, however these monolithic dishes are only possible in areas where there is already a suitable natural depression in mountainous regions and the cost of building them is prohibitive. Fortunately it is possible to combine the response of many small dishes into one signal with the resulting signal having the sensitivity and resolution of a single dish higher than any of the composite dishes (Thompson, 1999). All these dishes and the computer which combines their signals are collectively called an **interferometer**. These arrays of dishes have their own kind of engineering problems, mostly these issues are related to the infrastructure of the array and the computation required in combining multiple feeds. However this technology has allowed astronomers to continue building more sensitive steerable telescopes than would be possible with single dish telescopes (Dewdney et al., 2009; Thompson, 1999).

The reason that a larger dish telescope can detect weaker objects in the sky is because it collects more photons from that object. The power of the received signal is the amount of energy received per second. If a larger dish collects more photons from a source, the observed power of that source will be higher for the larger telescope. This creates a problem, how can we objectively compare the brightness of objects observed by different telescopes which have different collecting areas? For point sources the solution is to divide the power of the signal by the collecting area of the telescope. This gives us the flux of the object.

$$power = \frac{energy}{second}$$

$$flux = \frac{power}{area}$$

There is one further issue which should be considered before we can easily compare observations of from different telescopes. That is the bandwidth over which the observation is performed. A observation will typically only measure a portion of the frequencies emitted by a source. Any energy emitted by the source which is not captured will make the source seem to have a lower flux. To remove this bias, astronomers refer to the flux density of a source. This is the flux per unit frequency.

$$flux\ density = \frac{flux}{bandwidth}$$

The units of flux density are watts per unit area per unit frequency ($\frac{w}{m^2Hz}$.). Using these units most radio sources have very low values of flux density, typically on the order of 10^{-26}, for this reason radio astronomers created a unit called the Jansky (J). The Jansky is equal to $10^{-26}\ \frac{w}{m^2Hz}$. Radio sources will typically have flux densities from a few milli Janskys to a for thousand Janskys.

There are plans to build a new interferometer called the Square Kilometre Array (SKA) by 2024 (SKA SA, 2006b). SKA will consist of thousands of dishes and will be the largest and most sensitive radio telescope ever built. The telescope will contain 3 different types of antennas and be split between South Africa and Australia. South Africa will host the mid frequency arrays while Australia will host the low frequency aperture array (SKA, 2015). The location of the SKA high has yet to be decided. This telescope is so ambitious that it is not possible for it to be built with current technology and as a result there are a number of precursor projects which are being built to test its design principles and develop the technologies which will make SKA possible.

Table 1 - List of subsets of the spectrum and their common applications (Chaisson & McMillan, 2005)

Type of Electromagnetic wave	Common applications
Radio	Radar studies of planets Planetary magnetic fields Interstellar gas clouds and molecules Galactic structure Galactic nuclei and active galaxies Cosmic background radiation
Infrared	Star formation Centre of milky way galaxy Active galaxies Large-scale structure of the universe
Visible	Planets Stars and stellar evolution Normal and active galaxies Large-scale structure of the universe
Ultraviolet	Interstellar medium Hot stars
X-ray	Stellar atmospheres Neuron stars and black holes Active galactic nuclei Hot gas in galaxy clusters
Gamma-ray	Neutron stars Active galactic nuclei

One of these precursors is the MeerKAT interferometer being built in the Karoo region of South Africa which will also be the core site for the African part of the SKA (SKA, 2015). MeerKAT is planned to be operational by 2017 and will have 64 dishes (SKA SA, 2011). The volume of data created by these interferometers will be incredibly large, with MeerKAT producing 100s Gb/s and SKA producing Pb/s (Dewdney et al., 2009; SKA SA, 2011).

The South African portion of the SKA telescope is planned to be built on the same site as the MeerKAT telescope. The MeerKAT dishes will eventually be incorporated into the SKA telescope. Figure 5 shows the location that these instruments will be built in South Africa. To date there are already two operational radio telescopes on the site. There is the KAT7 instrument, which is a 7 dish precursor to the MeerKAT telescope and the Precision Array for Probing the Epoch of Reionization (PAPER) instrument, a telescope run by the University of California in Berkeley (PAPER, n.d.).

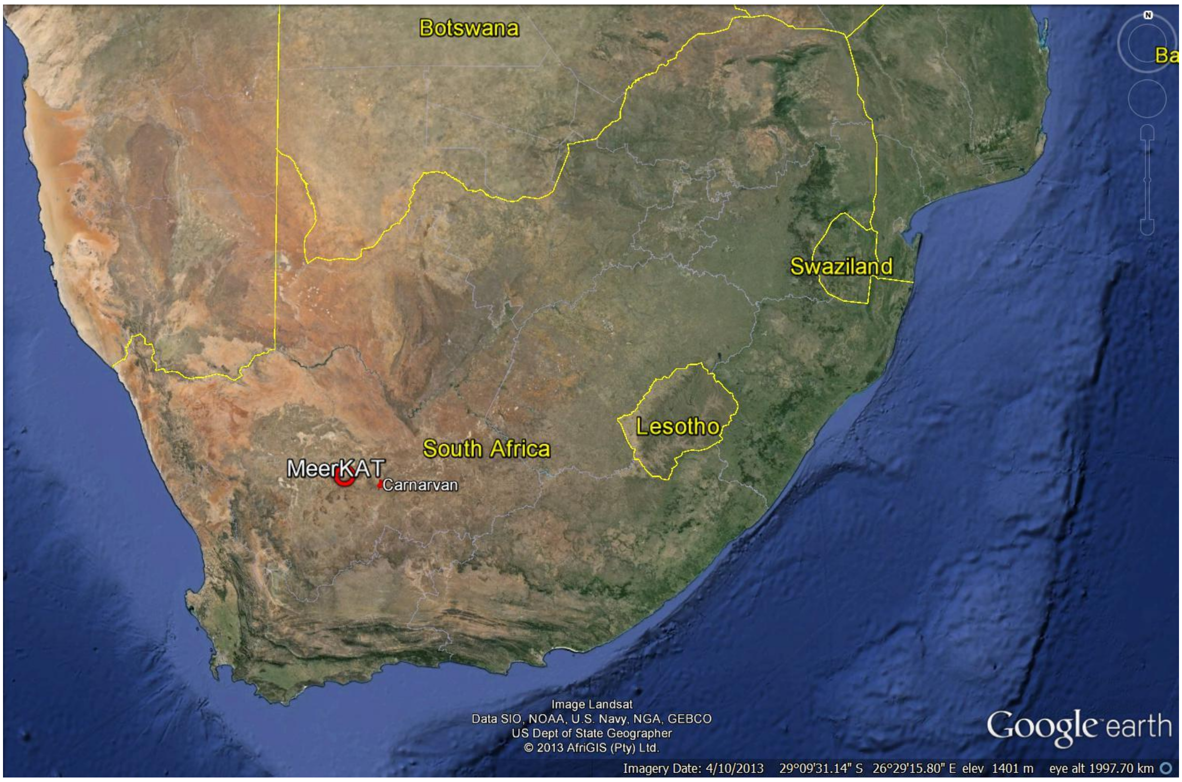

Botswana
Ba
Swaziland
Lesotho
South Africa
MeerKAT
Carnarvan
Image Landsat
Data SIO, NOAA, U.S. Navy, NGA, GEBCO
US Dept of State Geographer
© 2013 AfriGIS (Pty) Ltd.
Google earth
Imagery Date: 4/10/2013 29°09'31.14" S 26°29'15.80" E elev 1401 m eye alt 1997.70 km

Figure 5 - The red circle shows the location of the MeerKAT array. The closest town, Carnarvon is shown as a red pin. Carnarvan is approximately 73km in a straight line from the core of the MeerKAT array

Figure Source: Google Earth

2.2.1 Radio Telescopes

A radio telescope is a large dish which reflects waves onto a receiver. Figure 6 shows how the dish reflects light so that it focuses on a receiver which converts the electromagnetic signal into a radio frequency (RF) voltage signal. This voltage is sampled by an **Analogue to Digital Converter** (ADC). The ADC is a device which quantises continuous analogue signals, the ADC would be placed in between the receiver/amplifier and the computer system. The voltage of the signal is converted into a number of discrete voltage levels determined by the ADC resolution (the number of bits in the ADC output). In the case of an 8 bit ADC the power is converted into one of 256 power levels. The input values can be of any power; however the output will always have a defined range which does not cover all possible analogue values. There are therefore some signals which will be too powerful or too weak to be quantised, in this case we say the ADC over-ranges. The ADC samples the analogue signal at discrete intervals in time and the number of samples collected per second is called the ADC's sampling rate. Figure 6 is a simple diagram which shows how these different components work together to direct a radio signal from an astronomical source to a computer which can be used to analyse the signal.

In radio astronomy the goal is to measure the RF power **signal** of an astronomical radio source. The signals of astronomical sources are stochastic in nature. This means the signal is a random variable with a defined probability function. The PDF of the emitted signal from most astronomical sources are normally distributed. The challenge is to be able to tell if a signal is a bona fide astronomical signal or if the signal is just **noise**. All astronomical signals are embedded in the cosmic background noise, which is unavoidable. The receiver itself also introduces noise to the signal. Both types of noise have a normal distribution (Fridman, 2010).

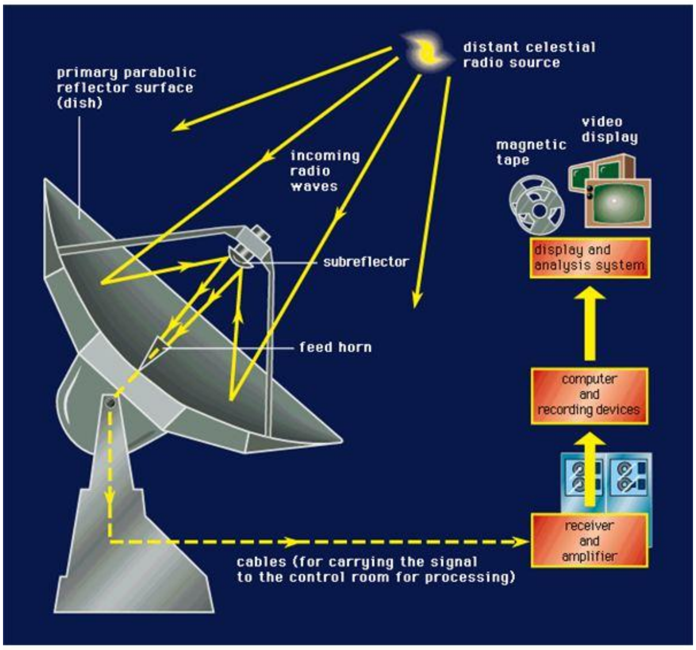

distant celestial radio source
primary parabolic reflector surface (dish)
incoming radio waves
magnetic tape
video display
display and analysis system
subreflector
feed horn
computer and recording devices
receiver and amplifier
cables (for carrying the signal to the control room for processing)

Figure 6 - Diagram of a typical radio telescope. The main reflector collects light from an astronomical source. The radio waves are reflected towards the subreflector which again reflects the radio waves to the feed horn. The feed horn converts the radio waves into an electric signal which can be amplified and digitised using an ADC. It is then fed into a computer to be processed, analysed and stored.

Figure source: http://pixgood.com/radio-telescope-diagram.html

The aim of the engineers building MeerKAT is to create an instrument which allows astronomers to distinguish weak astronomical signals from the noise of the receiver. The factors that affect the noise of the receiver can be described with the radiometer equation.

$$\sigma_T = \frac{T_{sys}}{\sqrt{Bt}}$$

In this equation σ_T is the uncertainty of the receiver detected power, T_{sys} is the effective noise temperature of the detected power, B is the bandwidth over which we receive and t is the amount of time the signal is integrated over (Kraus, 1986). A receiver with a low uncertainty will provide the clearest view of an astronomical signal.

By lowering the uncertainty of the receiver detected power we increase the **Signal to Noise Ratio** (SNR). SNR is defined as the ratio of the power of the desired astronomical signal over the power of background noise. It can be quoted in decibels (dB) using the formula and is calculated using the formula $SNR_{dB} = 10\log_{10}\left(\frac{P_{signal}}{P_{noise}}\right)$. From the radiometer equation we see we can lower uncertainty by lowering T_{sys}. One way of lowering T_{sys} is by cooling the physical receiver to very low temperatures, for instance the receiver of the MeerKAT dishes is cooled by helium to an operating temperature of around 13 degrees Kelvin.

Another way to increase the SNR of the signal is to **accumulate** the signal (Miller, 1998). Accumulation is the process of adding multiple contiguous samples of a signal, this accumulation time is represented by t in the radiometer equation. When the signal is accumulated, the random noise rises by the square of the accumulated time while a signal present in all samples will rise linearly with the accumulated time. Therefore the uncertainty of the received signal will fall as the square root of the accumulation time. Accumulation has the added benefit of reducing the amount of data that needs to be handled by the signal chain. Accumulation is called integration in the case we are dealing with a continuous analogue signal, rather than a digital signal with discrete values.

As we said earlier, another way to increase the sensitivity or SNR of a telescope is to combine the input of separate dishes. The signal from each of the dishes must be combined in a sensible way so that signals that are common to all dishes become more powerful and signals which are not common for each dish (i.e. the internal noise of the separate receivers) are diminished. We call signals which are common to two dishes coherent signals. This process of combining the signals from multiple dishes is called correlating the signals.

Correlation is performed by a computer or cluster of computers called a **correlator**. The function of a correlator is that it correlates the signals from all possible pairs of dishes in the array. We call each pair of dishes in an interferometer a **baseline**. **Correlation** means multiplying each pair of values from each baseline together, which will tend to increase the coherent signals and diminish the effect of noise signals. It is important to ensure that the signals from different dishes are delayed before being correlated. Figure 7 shows how the angle of the dishes affects the time each dish will sample the same signal. The angle a dish observes at will affect the time that dish samples a wavefront.

Dishes which sample earlier must be delayed in order to preserve the phase of the signal. As correlation is performed on each baseline, the computational complexity of correlation rises as the square of the number of dishes in the interferometer.

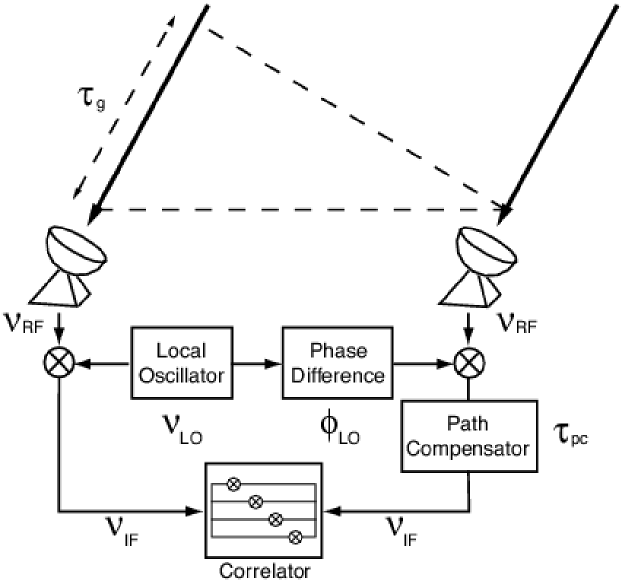


Figure 7 - A diagram showing the geometric delay τ_g between two dishes in an interferometer. We assume the incoming signal is a plane. As such, unless the incoming plane is perpendicular to the dishes in the array, the signal will arrive at each dish at different times.

Figure source: http://www.aanda.org/articles/aa/full/2008/40/aa8117-07/aa8117-07.html

All of the combined signals of the interferometer create something called a synthesised **beam**. This represents the directions which the interferometer is sensitive to. The shape of the beam can be changed by applying weights to the signals from the separate baselines and the beam can be directed by changing the direction the dishes point at. In Figure 8 you see an example of a hypothetical beam of an interferometer. It is made up of a main beam which is the direction which we observe objects with. The beam also has less sensitive areas which point away from the main beam. These areas are called the side lobes of the beam. It is not possible to differentiate which signals come from the main beam and which come from the side-lobes, although the interferometers are generally designed to suppress the side-lobes so that they are far less sensitive than the main beam. Nonetheless it is possible that powerful signal in a side lobe would obscure a weak signal in the main beam.

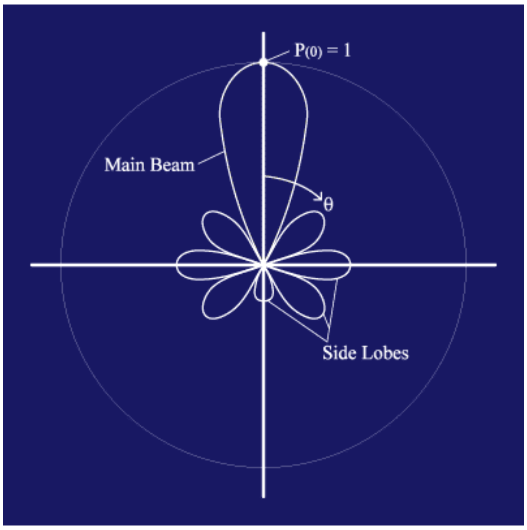


Figure 8 - A hypothetical beam of a radio telescope. The distance of the line from the centre represents the sensitivity of the telescope in that direction. You can see the main beam is by far the most sensitive beam. This image is a 2 dimensional representation or a three dimensional beam.

Figure source: http://www.ras.ucalgary.ca/radiotel/calibration.html

An interferometer can be judged on two basic metrics. The first is sensitivity; the larger the combined area of the dishes in an interferometer, the more sensitive that telescope is. The second is **the angular resolution** which is related to the size of the main beam. The angular resolution is a measure which tells us the smallest angle between two objects at which they can be resolved as separate objects. If two objects are closer together than the angular resolution, they will appear as one object with the combined power of both objects. That is, they will not be resolved.

The angular resolution of an interferometer is determined by the length of its longest baseline, i.e. the maximum distance between two dishes in the array. This means that a longer baseline provides a smaller angular resolution. This allows us to resolve objects that are close to each other on the sky. It is important to remember that for a given angular resolution objects which are further from the observer also need to be further apart from each other in order to be resolved, as the distance needed to satisfy the minimum angle grows with the distance from the observer. Providing long baselines is the main reason that the planned SKA telescope will be spread across the continent of Africa, as no single country is large enough to contain an interferometer with the angular resolution that SKA requires to perform its science goals.

2.3 Radio Frequency Interference (RFI)

Radio Frequency Interference (RFI) is any signal captured by a radio telescope which does not come from astronomical objects. RFI can be separated into 2 categories; internal and external RFI. Internal RFI is interference generated by the telescope system itself. As the telescope is made from electronic devices which emit radio waves, all radio telescopes pick up some radiation which is generated by its own internal components. External RFI comprises all interference generated by sources on the Earth and its satellites. This RFI may be generated by anything from electric fences to lightning to aeroplanes and satellites (Porko, 2011; Ekers & Bell, 1999).

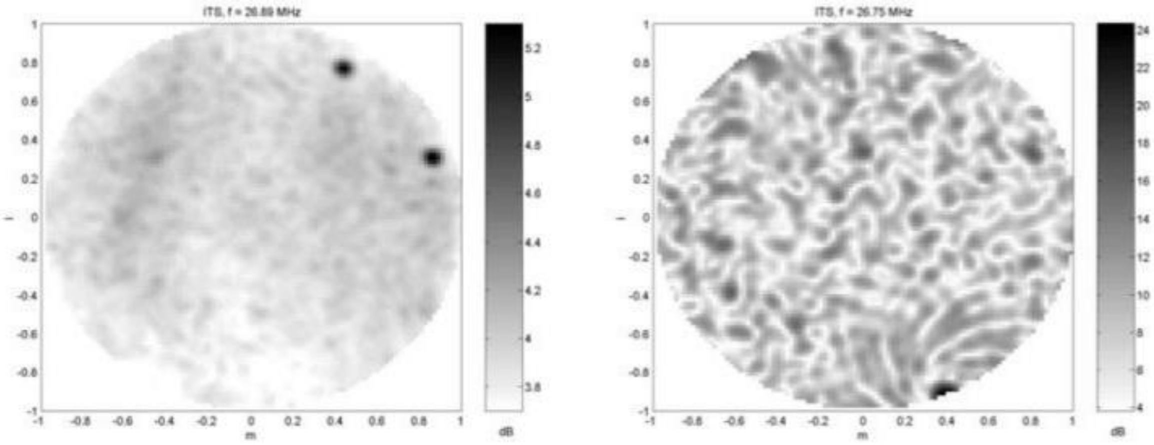


Figure 9 : Example of the effect of RFI. Both images were obtained from the LOFAR test station of exactly the same part of the sky at the same time, on separate frequencies. The image on the right is on a frequency which has a strong transmitter, the objects which you can see in the RFI free case (left) are totally obscured.

Figure source : http://www.ece.vt.edu/swe/RFI2004/18p.PDF

Radio telescopes suffer from RFI in much the same way that optical telescopes suffer from light pollution. As astronomical signals are generally 60 dB below the receiver noise level (Gillani, 2010), even relatively weak man-made sources can completely obscure astronomical signals, placing an effective limit on sensitivity (Gillani, 2010). Figure 9 shows just how badly a local source can affect an observation. The RFI contaminated observation on the right contains artefacts from an RFI source which completely obscures the astronomical sources visible in the image on the left. As radio telescopes become more sensitive, RFI becomes a bigger problem. For example, a telescope as sensitive as SKA could pick up the signal made by a cell phone tower from hundreds of kilometres away (SKA SA, 2006b).

In recent times the radio spectrum has been widely used as a communication medium and we expect that both the spectrum use and power of wireless technologies will continue to increase. Some forms of interference can be avoided; however the use of telecommunications and GPS satellites creates interference all over the globe. As a result it has become increasingly difficult to avoid interference by building telescopes in radio quiet areas. As such it is inevitable that some observations will be corrupted by RFI and it is now essential that radio telescopes have some form of automated RFI monitoring and flagging or excision (Fridman & Baan, 2001; Offringa, Bruyn, & Biehl, 2010; Ekers & Bell, 1999).

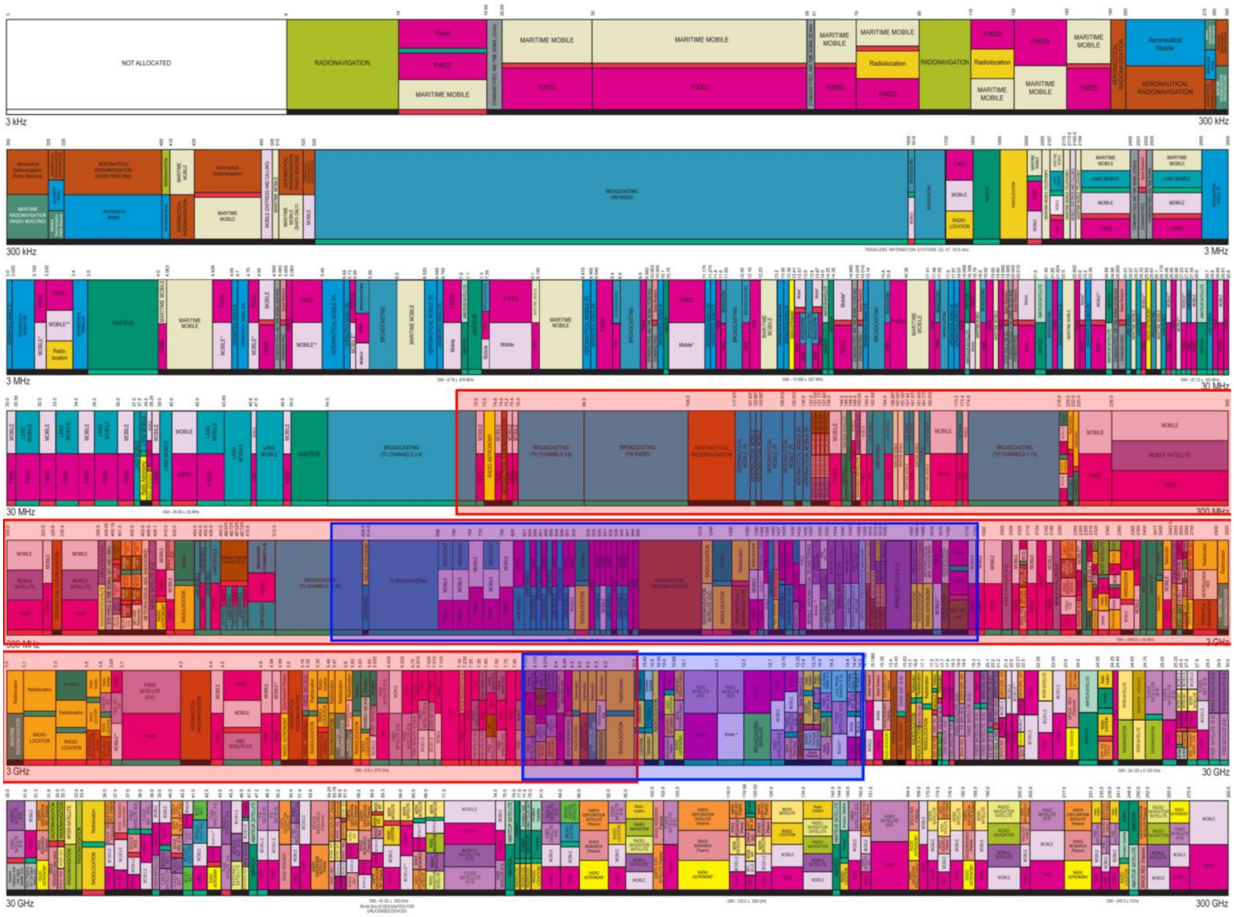


Figure 10 : The allocated uses of the radio spectrum for communication. The spectrum which SKA will observe on is shown by the transparent red boxes and the spectrum which MeerKAT will observe on a surrounded by the transparent blue boxes. As you can see most parts of the radio spectrum are allocated to some form of communication.

Figure source: http://www.ntia.doc.gov/files/ntia/publications/2003-allochrt.PDF

2.3.1 RFI Characteristics

All RFI sources have a combination of the characteristics defined below.

Broadband – The source emits on a range of frequency channels

Narrowband – The RFI emits on discrete frequencies, for our purposes a RFI source is narrowband if it is received on discrete frequency channels

Persistent – The RFI source is always emitting

Intermittent – The RFI source emits intermittently for periods of minutes to hours

Burst-like – The RFI source emits in short bursts of the order of milliseconds to seconds

High-powered – The RFI is more powerful than the instrument noise

Low-powered – The RFI has a similar power to the instrument noise

Stationary – The RFI source is stationary

Mobile – The RFI source is moving

Polarised – The RFI source emits mostly polarised emissions

Table 2 - List of RFI sources and their characteristics

RFI Source	**Characteristics**
Two way radio	Narrowband, Intermittent, High-powered, Mobile, Polarised
GPS satellites	Narrowband, Persistent, High-powered, Stationary, Polarised
Electric fences	Broadband, Burst-like, power dependent on distance, Polarised
Ethernet cables	Broadband, Intermittent, High-powered, not Polarised
Lightning	Broadband, Burst-like, High-powered, not Polarised
TV	Narrowband, Persistent, High-powered, Stationary, Polarised

Table 2 contains a non-exhaustive list of RFI sources; it illustrates the large range of possible RFI that can affect observations. As there are so many different types of sources there is no single way to eliminate RFI from observations (Ekers & Bell, 1999). There are also some astronomical sources and RFI sources which produce signals with similar characteristics, it is important to ensure we do not ignore astronomical sources because of this similarity. For this reason minimising the effect of RFI on observations is a process which requires multiple and complementary solutions.

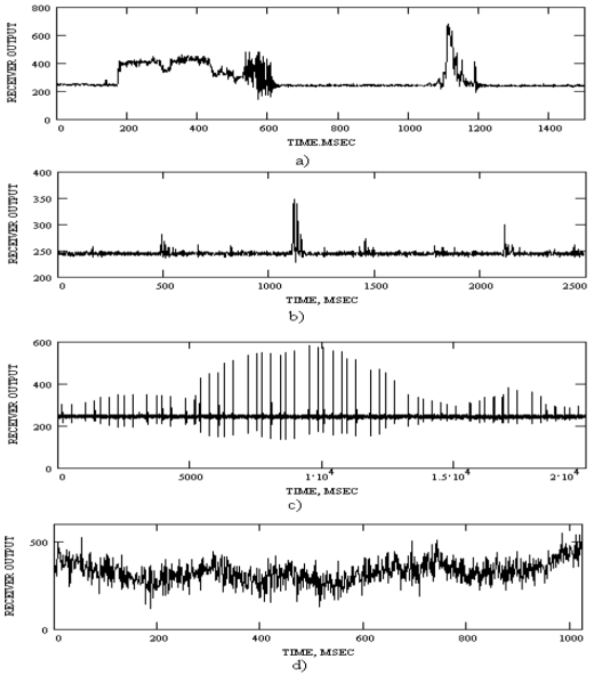


Figure 11 - Examples of RFI waveforms in receiver output (power) vs. time; a) b) and c) are impulse-like and intermittent; d) persistant, narrow-band. In these plots we are looking at power over the whole bandwidth of the reciever

Figure source: (Friedman & Baan, 2001)

Figure 11 shows the waveforms of some representative RFI sources. The first 3 sources are examples of intermittent burst like RFI, you can clearly see the RFI bursts in a time series. The third source is has a obvious period, in this case the RFI is generated by radar pulses. The 4th source is a continuous narrow band signal. One can see that different characteristics of the RFI emission result in different features in the captured data.

2.3.2 Effects of RFI on Science

RFI can limit the science that is possible with a radio telescope as some parts of the spectrum are unusable because of constant interference. For example, in the study of the evolution of galaxies over cosmic time; interference over a small band of frequencies can prevent astronomers from learning what forms of galaxies existed for certain ranges of time. In order to learn about galaxies astronomers look for the tell-tale spectral line of neutral hydrogen which emits at the 21cm wavelength or 1420 MHz (Burke, 2010). This spectral line is called the HI line. Astronomers can use the HI line to understand how galaxies evolve over cosmic time. If the HI line is obscured by RFI, they cannot use it to learn about these ancient galaxies.

Studying the evolution of galaxies at different periods of cosmic time is possible because of the constant speed of light, red-shifting and the expansion of the universe. The universe has been expanding since the Big Bang and, as a result all objects in the universe other than our local cluster of galaxies are moving away from us. Those objects which are further away are moving away at a higher velocity. This means that those objects which are further from us will have a higher red shift (their HI line will be shifted to a lower frequency). We can approximate the distance to a galaxy based on the red shift of its HI line using Hubble's Law (Burke, 2010). Also because light travels at a constant speed, the light from these distant galaxies must have been travelling for a long time. So HI emission from high red shift galaxies left those galaxies when the universe was younger than it is today. The higher the red shift of a HI spectral line of a galaxy, the faster that galaxy is moving away from us and the younger that galaxy was when it emitted the HI line.

If the frequencies from 1000MHz to 1010MHz were unusable because of RFI, astronomers would not be able to see what kinds of galaxies existed from 9.14 to 9.26 billion years after the big bang, a gap of over 100 million years (Bremer, 1995; Kempner, n.d.).This problem is made worse by the fact that the HI line is a weak signal and for distant galaxies the signals are even weaker. This means that for studies of the most distant galaxies where observations must be accumulated over long periods of time, even very low powered and intermittent RFI can corrupt time intensive studies.

RFI has varying effects on different science goals depending on the type of emission. If the science requires observing spectral line emissions then RFI can completely obscure the emission. If a science goal relies on observing continuum emission then it is possible to ignore frequencies of the emission which are affected by RFI and use the remaining clean frequencies to achieve your science goal (P A Fridman & Baan, 2001). Even if the RFI does not make all required frequencies unusable, it still negatively affects the SNR of any observation made over those frequencies. In order to allow our new generation of radio telescopes to fulfil the science goals they are being built for we must mitigate the effects of RFI.

Another consideration when dealing with RFI is the type of telescope being used. Different dishes have differently shaped beams; the width of the main beam determines how close an RFI source can be to the astronomical source without interfering. Other than the main beam, side-lobes can pick up signals which come from unexpected directions. Finally, in an interferometer there is a natural spatial filter against ground based RFI. As the signal from an interferer will reach dishes at different locations at different times, it is possible that the RFI will not be correlated. In this case the process of correlation naturally removes RFI. However, for short baselines and persistent RFI it is far more likely that RFI will be correlated (Baan, 2011). It is also possible to de-correlate satellite RFI using fringe washing as satellites do not have sidereal motion.

2.4 RFI Mitigation

There are multiple approaches to preventing RFI from affecting observations. These techniques are referred to as RFI mitigation. Each radio telescope facility has a RFI mitigation plan which uses those techniques which are both effective and possible for their environment. These mitigation techniques range from detection algorithms to physical filter systems to legislative actions (Baan, 2011). As each telescope exists in a different physical, legislative and radio frequency environment; there is no unique set of mitigation techniques for all radio telescopes (P A Fridman & Baan, 2001). There are common techniques which are used in some form by most radio telescopes.

As a rule it is generally better to perform mitigation as early as possible. The best option is to prevent RFI sources from emitting. If this is not possible we would like to prevent RFI from affecting observations. The final option is to remove RFI from observations. It is always better to prevent RFI as removing RFI from observations necessarily lowers the sensitivity of that observation(Baan, 2011).

2.4.1 Legislation

The International Telecommunications Union (ITU) created the ITU regulation RA 769 in 1992 to provide guidance on the parts of the spectrum which are useful for radio astronomy (ITU-RA, 2005). They provide guidelines on minimum flux densities of RFI sources that will impact radio telescopes of a minimum sensitivity. The ITU has allocated some parts of the spectrum exclusively for Radio

Astronomy. These parts of the spectrum are called the Radio Astronomy Service (RAS). The RAS bands can only be used passively, which means no one is allowed to emit signals on those bands. There are only 21 frequency bands which are allocated to passive use. Some of these bands are shown in Figure 10, they are shown in bright yellow. They generally protect bands on which important science relies on, however there is no provision for the effects of red shifting of astronomical objects (Diepenbeek, 2010). From Figure 10 it is clear that most of the spectrum is not protected. As spectrum allocation is becoming increasingly valuable economically, it is important that the radio astronomy community is involved in regulations drafted both internationally and in their own countries.

Other than general spectrum allocation over the world or country, it is also possible to make special Radio Quiet Zones (RQZ) in which large parts of the spectrum are legally protected for passive use. In preparation for the SKA telescope, both the Australian and South African governments have designated RQZs around the sites of their portions of the SKA (van Driel, Gergely, Liszt, & Ohishi, 2012). These RQZs introduce legislation which guarantees that emissions of electronics and telecommunications must remain low enough that they do not interfere with radio astronomy.

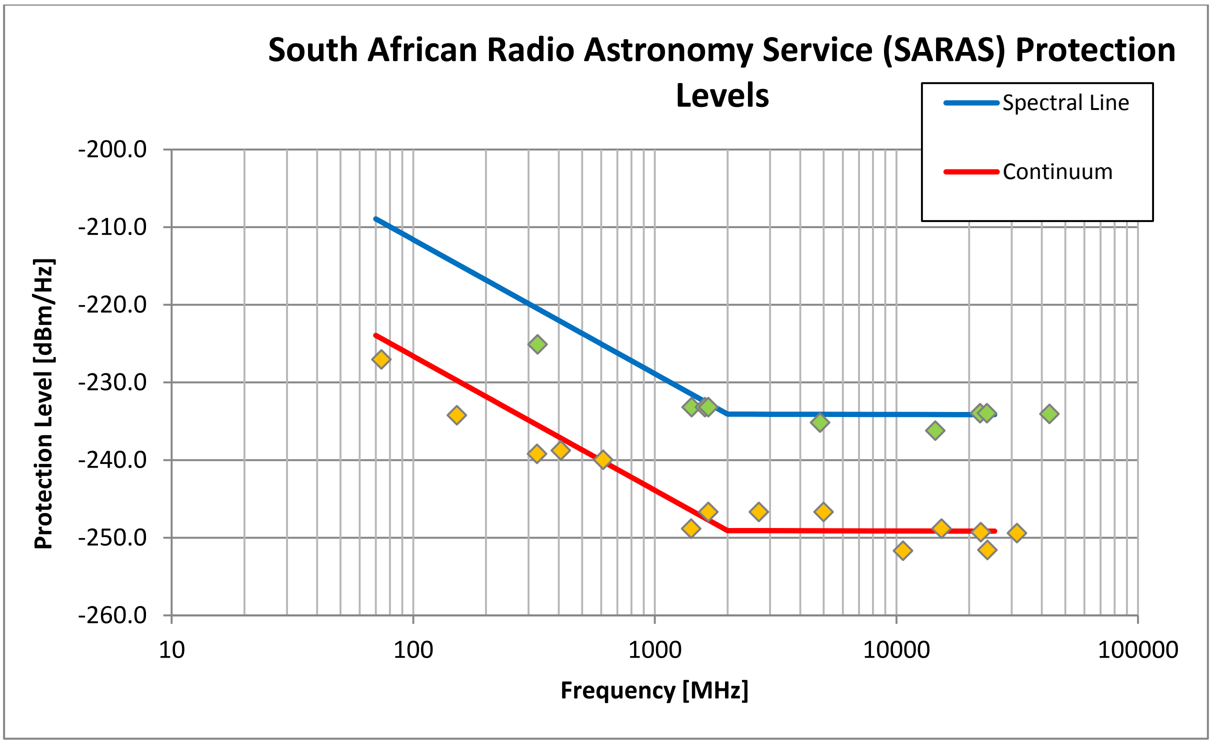


Figure 12- SARAS protection levels for spectral line and continuum observations. If a RFI source is measured at a dish to have a higher power than the threshold it violates the AGA

In South Africa the RQZ called the Karoo Core Central Astronomy Advantage Area (KCAAA) (Driel et al., 2012) is under the purview of the Department of Science and Technology. The protected status of this area was signed into law with the Astronomy Geographic Advantage Act (AGA) of 2007 (Department of Science and Technology, 2007). The area which makes up the KCAAA is shown in Figure 12. According to the AGA any transmitter going into this area must first get a license from the minister. These transmitters will only get permission if they do not exceed the levels set by the South African Radio Astronomy Service (SARAS) shown in figure 13 (van Driel et al., 2012). All transmitters which already exist in the KCAAA must conform to SARAS levels. Failure to comply with the regulations can result in a warning, suspension of license, a fine or even jail time (van Driel et al., 2012).

These legislative measures ensure that radio astronomers will have some guarantee that the area around new and expensive radio telescopes will be protected from the advance of ever more powerful telecommunications systems. Unfortunately no national regulation can prevent the existence of RFI generated by satellites, however satellite emissions are still governed by the ITU and cannot emit in the RAS part of the spectrum(Zoller, 2011).

2.4.2 Monitoring

For obvious reasons we build radio telescopes in locations which already have as low a level of RFI in the environment as possible. The site of the MeerKAT telescope for example is in one of the least densely populated parts of South Africa. It is surrounded by mountains which help block the signals of high powered radio communications, such as cell phone towers, TV and radio stations and air traffic control towers, whose signals can be picked up hundreds of kilometres from their source (SKA SA, 2006b). The site was extensively surveyed to understand the RF environment; however, it is hard to tell what will happen to the radio frequency environment as the years to come (Bolli, Gaudiomonte, & Messina, 2010).

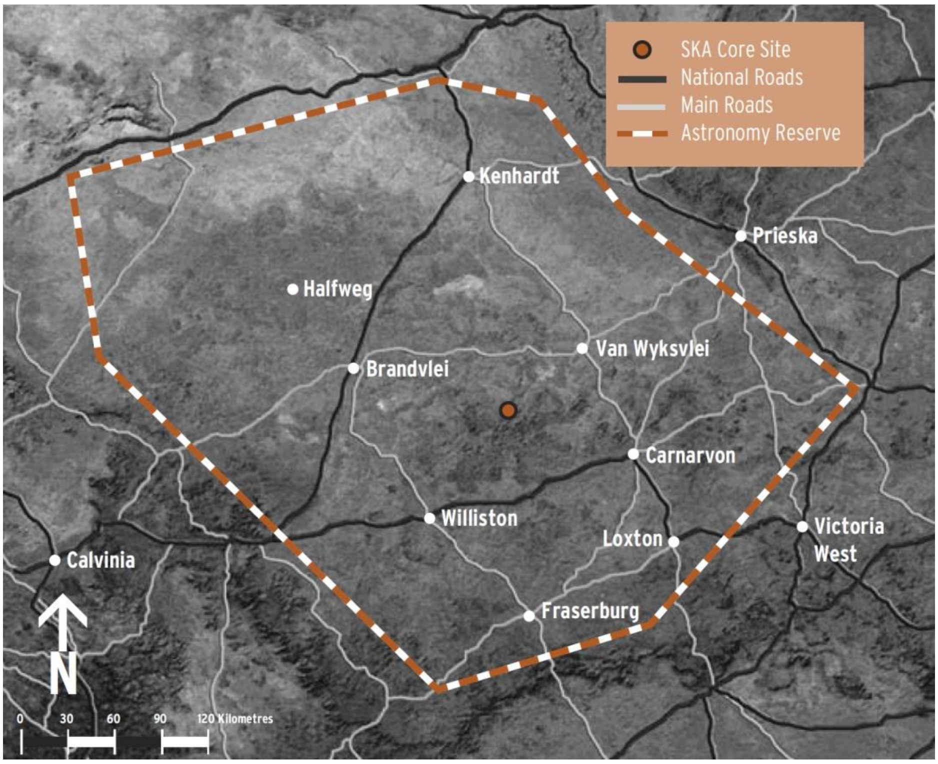


Figure 13 - Map of the South African Radio Quiet Zone around the future SKA sit

Figure source: http://www.ee.co.za/article/hans-05-how-will-the-ska-affect-people-in-the-astronomy-advantage-area.html

For example, the site of the Northern Cross radio telescope in Medicina, Italy which had a quiet radio environment when the Northern Cross was built in the 1960s. However, over the years the growth of the nearby city Bologna, the increasing use of the radio spectrum for telecommunication and the proliferation of personal electronic devices have caused the environment to seriously deteriorate. With even the internationally recognised RAS part of the spectrum deteriorating, the observatory decided to create a team of technicians whose primary responsibility was to monitor RFI and report official complaints on emitters interfering with the RAS (Bolli et al., 2010). Their dedication has made it possible to preserve the bands utilised by Medicina's instruments.

The best way to prevent RFI from corrupting observations is to understand the RFI environment around the interferometer. If we know the environment well enough it is easier to find and remove sources which fail to meet legislated levels (Bolli et al., 2010). It is easier to avoid RFI sources during observations and also to accurately remove that RFI which will inevitably corrupt observations. It also allows engineers to discover whether RFI is internally or externally generated by having a record of the external RFI (Boonstra, 2005; Gillani, 2010).

2.4.2.1 RFI Monitoring at Arecibo

The Arecibo radio telescope is the largest single dish radio telescope in the world (National Astronomy and Ionosphere Center, n.d.). The RFI environment around the dish is monitored by an omni-directional antenna which receives on the 1.7 GHz to 10 GHz range. The system is run 24 hours a day under computer control. The monitoring system steps through 19 frequency bands, taking 1 minute to observe each band (Perillat, n.d.).

All of the raw data is saved to disk and can only be accessed on site. In Figure 14 one can see the type of plots generated automatically from data captured each day. These plots can be accessed through the internet. They include mean power for each frequency channel, image plots of the time-frequency power, the Root Mean Squared (RMS)/mean and PDF documents containing the dynamic spectra for all frequencies over the last 8 days. Along with the daily plots there is also a plot of the power spectrum for the Industrial Scientific and Medical (ISM) band for every 20 minutes for the last month (Perillat, n.d.).

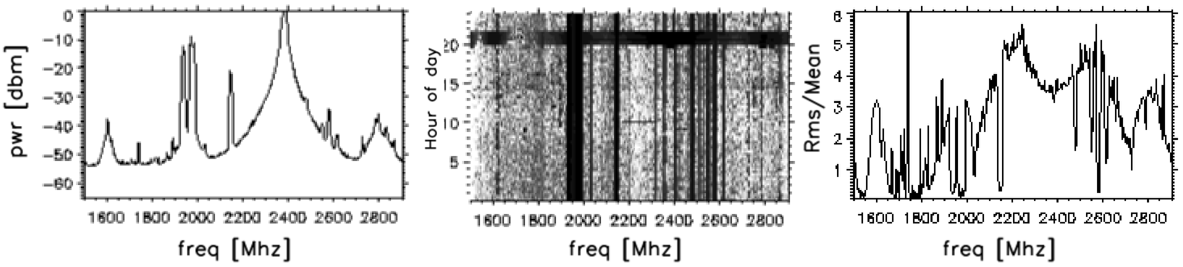


Figure 14 : Examples of the automatically generated plots from the Arecibo RFI monitoring system. The images from left to right are mean power, time-frequency power and the rms/mean

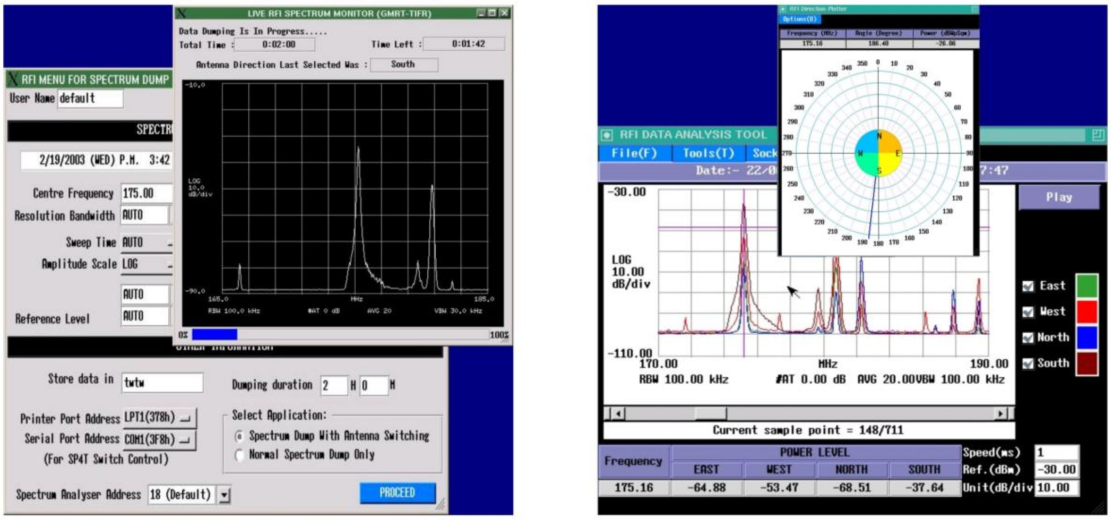
Figure source: http://www.naic.edu/~phil/rfi/hilltop/hilltop.html#History_

These plots are the only way to access the data collected by the monitoring system without going to the site. It is not possible to generate custom plots of selected frequencies or times and there is no way to search the plots except by eye, which means one has to look through a 30 page document in the case of some plots (Perillat, n.d.). The monitoring system does not run any kind of RFI detection algorithm so all analysis must be performed manually.

2.4.2.2 RFI Monitoring at the Giant Metrewave Radio Telescope

The Giant Metrewave Radio Telescope (GMRT) is a radio telescope array consisting of 30 steerable dishes each with a diameter of 45 metres. It was built 80km outside a town called Pune because of the low levels of RFI in the area (GMRT, 2008). The observatory also has a RFI monitoring system, consisting of four antennas each pointing to one of the cardinal points. The system cycles through each antenna to gather spectral data and is capable of finding the direction of a RFI source (Joardar, 2005).

The system does not operate 24/7 like the monitoring system at Arecibo, it must be initiated using a terminal connected to the LAN at the facility. The operator can specify how long to observe for, what frequencies to observe on and where to save the spectral data which are collected. There is software which can visualise the collected data in real time or from a completed data file. Screenshots of the monitoring software are shown in Figure 15. These screen shots show the time averaged spectral power plot, the spectrum of a time sample and the direction of a the most powerful source for a selected band/range (Joardar, 2005). This software can be used by operators to determine in real time if unusual levels of RFI are present and from which direction that RFI is being emitted.


LIVE RFI SPECTRUM MONITOR (GMRT-TIFR)

Data Dumping Is In Progress.....

Total Time : 0:02:00 Time Left : 0:01:42

Antenna Direction Last Selected Was : South

RFI MENU FOR SPECTRUM DUMP

User Name default

2/19/2003 (WED) P.M. 3:42

Centre Frequency 175.00

Resolution Bandwidth AUTO

Sweep Time AUTO

Amplitude Scale LOG

AUTO

Reference Level AUTO

Store data in twtw

Dumping duration 2 H 0 M

Printer Port Address LPT1(378h)

Serial Port Address COM1(3F8h)

(For SP4T Switch Control)

Select Application:

● Spectrum Dump With Antenna Switching

○ Normal Spectrum Dump Only

Spectrum Analyser Address 18 (Default)

RFI Direction Plotter

Frequency (MHz)	Angle (Degree)	Power (dBm)
175.16	186.40	-26.06

RFI DATA ANALYSIS TOOL

Date:- 22/0 … 7:47

Current sample point = 148/711

Frequency	POWER LEVEL			
	EAST	WEST	NORTH	SOUTH
175.16	-64.88	-53.47	-68.51	-37.64

Speed(ms) 1

Ref.(dBm) -30.00

Unit(dB/div 10.00

Figure 15 : Screen shots of the RFI monitoring visualisation system of the GMRT. The image on the left shows the GUI used to set up and monitor an ongoing observation. The image on the right shows the online data visualiser and the direction finder, the direction finder shows the direction of a source at the frequency selected by the cursor on the spectrum visualiser.

Figure source : (Joardar, S. 2005)

The monitoring system also provides an offline RFI analysis program which can flag narrowband and broadband interference and visualise where this RFI is in the spectrum. Figure 16 shows how the offline flagging software can show observers on which frequencies and times RFI was emitted. Figure 17 shows how this software can provide spectral occupancy plots from saved data. This allows observers to find the percentage of time a frequency was contaminated with RFI during an observation.

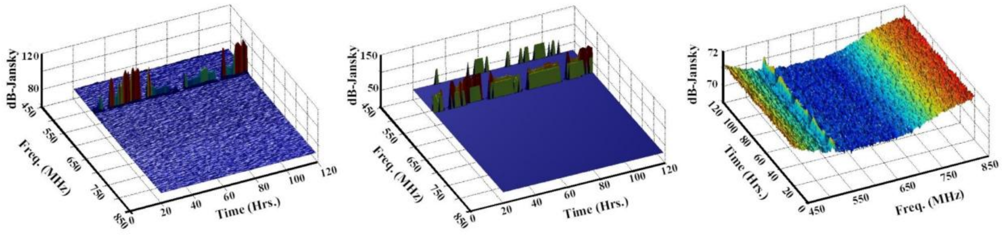


Figure 16 : Screenshots from the RFI analysis software of the GMRT. From left to right, the first image shows the whole spectrum as observed by the RFI monitoring system. The second image shows the narrowband RFI and the third image shows the broadband interference flagged by the analysis software.

Figure source : (Joardar, S. 2005)

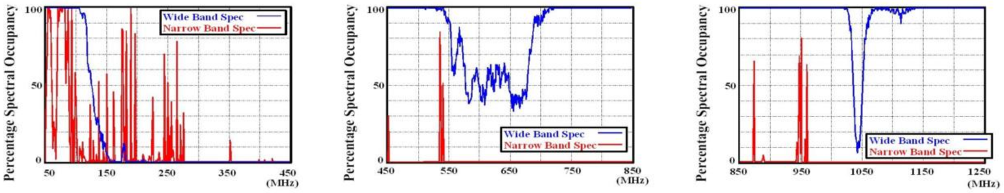


Figure 17 : Spectral Occupancy plots for narrowband and broadband interference collected by the RFI monitoring system of the GMRT and flagged by the analysis software.

Figure source : (Joardar, S. 2005)

2.4.3 RFI Flagging

As even the best RFI mitigation methods cannot totally prevent all RFI (Baan, 2011), there must be methods which reduce the effect of RFI after observation. One method is to flag those parts of the spectrum which have RFI and not use those frequencies/times when RFI was present while analysing observations. As this technique involves removing data from analysis it is imperative that the flagging process does not erroneously flag real astronomical signals as RFI, otherwise our mitigation would become counterproductive (Offringa, 2012).

The effectiveness of RFI flagging depends on the accuracy of the RFI detection process. In the past RFI detection was performed by visual inspection, identification and flagging; however, due to the huge increase in the amount of data to be flagged, this is no longer a feasible solution for most telescopes (P A Fridman & Baan, 2001). As a result algorithms which can automatically detect RFI have been developed. These algorithms detect RFI by looking for characteristics of RFI, but not astronomical sources, such as high power levels, non normal signals and non-sidereal motion. As with most of the mitigation techniques mentioned, there is no one detection algorithm which is good at detecting all RFI (P A Fridman & Baan, 2001). As such it is important to consider the properties of the RFI that must be detected before deciding which detection algorithm to use.

It is important to realise that when telescope data is flagged as RFI, the information of astronomical signals on that channel is lost. There are situations where flagging RFI will completely remove the desired astronomical signal from data. In cases where RFI is constant and covers all frequencies on which a signal can be observed, we need to use either RFI excision or spatial nulling (Baan, 2011).

2.4.4 Spatial Nulling and RFI Excision

Spatial nulling is used in multi-element systems, such as interferometers or dishes with multiple feeds; it is possible to form a beam such that the null of that beam are in the direction of a known interferer. In this case the RFI from any source in the null will not affect the observation. This can only be performed if the location of the RFI source is known and either stationary or follows a known path.

RFI excision is a technique which allows us to remove the effect of RFI from observations and reveal the underlying signal (P A Fridman & Baan, 2001). Unlike flagging, this approach allows us to perform observations on frequencies with interference. This is done is by determining which part of the observed signal is due to RFI and to then subtract that RFI signal from the observations.

To get an estimate of the RFI signal, we need to have an observing dish which points at the astronomical source and a reference dish which observes at the same time but points off the source but also contains the RFI. From both of these data streams it is possible to estimate the signal which is due to RFI and to subtract that RFI from the observation, revealing the underlying signal. In the case of a multi-feed dish, where each feed can point to a different part of the sky, this method can be applied using only one dish(P A Fridman & Baan, 2001).

2.5 RFI Monitoring at MeerKAT

2.5.1 SKA Site Bid – RFI Measurement Campaign

As MeerKAT is being built on the proposed site for SKA, the RFI environment was extensively tested as part of the SKA site bid. The point of this testing was to find if the site was radio quiet enough to host the SKA (Manners, 2007). This testing involved 12 months of RFI monitoring at the core site in the Karoo and at sites in other host countries in Southern Africa. To facilitate this process three mobile measurement systems were developed which allowed for accurate measurements of the environment.

The Mobile Measurement Systems (MMS) were designed according to SKA specifications for RFI measurements. They were used to perform high sensitivity, high frequency resolution scans of small parts of the spectrum with particular astronomy interest, these were called the Mode 2 requirements. Lower sensitivity, lower frequency (> 1Hz) resolution scans of larger parts of the spectrum were required in Mode 1 requirements, shown in Table 3, which were meant to measure the RFI environment over most of SKA's proposed bandwidth from 0.07GHz to 26.5GHz (Manners, 2007).

Table 3 - Mode 1 RFI measurements. The table is divided into the requirements for measurements as required by SKA and the settings that the MMS had to be set to meet the requirements. The total time shows how long it took to measure all required frequencies (Manners, 2007)**.**

SKA Requirements		Operational Settings	
Frequency Band GHz	RBW kHz	ΔT_{RBW} ms	Integration Time s
0.07 – 0.15	3	10	1334
0.15 – 0.30	3	10	500
0.30 – 0.80	30	10	167
0.80 – 0.96	30	10	1000
0.96 – 1.40	1000	0.002	900
1.40 – 3.00	30	10	534
3.00 – 22.00	1000	10	190
Total			1.3 hours

Table 3 shows the resolution bandwidth (RBW), i.e. the bandwidth of the frequency channels in the measured spectrum required by the SKA. The values of $\boldsymbol{\Delta T_{RBW}}$ are the integration times each band must be measured in order to get the required sensitivity for the measurements. The mode 1 measurements were far less sensitive and time consuming than those required in the Mode 2 measurements. To see mode2 measurements refer to appendix.

In order to characterise the RFI around the core, Mode 1 measurements were performed at all of the sites shown in Figure 18. At each site, Mode 1 measurements were made for 3 to 5 antenna pointings on 2 polarisations each; this process took 27 hours on average and was performed once for each site. The idea of these measurements was to characterise strong interferers in the parts of the spectrum which are not critical for main SKA science and therefore do not need more sensitive measurements.

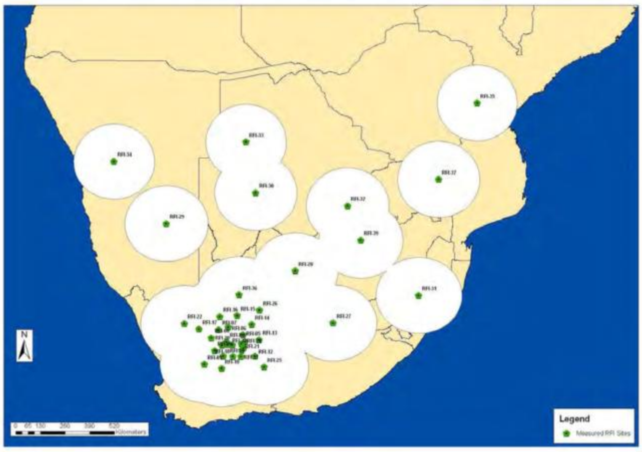


Figure 18 - Locations of the Mode 1 RFI measurements

Figure source : (SKA SA, 2006b)

The result of RFI measurements which are shown in Figure 19 show there was virtually no RFI above 2GHz and although there was significant RFI below 2 GHz, it was contained in the frequency bands allocated to terrestrial and satellite services (Manners, 2007). This is good news for the SKA and MeerKAT, however we need to remember that these measurements were not continuous, so it is unlikely that it characterised intermittent or burst-like interference. Also these measurements only tell us about the RFI in 2005. The list of strong emitters found is shown in table 4. The site has changed significantly since then with the introduction of the infrastructure required for MeerKAT and the on-going construction of MeerKAT, and SKA which will continue until 2024 (SKA SA, 2011).

Some of the radio, TV and cellular transmitters were within the RQZ at the time of the measurement campaign as the RQZ had not yet been signed into law. As those transmitters are above the levels required by the RQZ they are going to be replaced by low powered transmitters which will cause less interference. As of December 2012 TV transmitters are in the process of being replaced. The mobile communication providers are in consultation with the SKA SA. Different solutions which will not interfere with MeerKAT, but will allow for mobile communication in the area are being discussed. Unfortunately, the satellite emitters are not under the purview of South Africa and the avionics systems are critical for the safety of aircraft. Currently there is no alternative and these emitters will remain for the foreseeable future.

Table 4 - Table of strong emitters and their frequency bands found during the RFI measurement campaign

Frequencies MHz	Source
90-108	FM Radio
137-138	Satellite
240 - 270	Satellite
250 - 450	Mobile Communications
170-320	VHF TV
480 - 830	UHF TV
900 - 960	GSM
960 – 1150	Avionics
1164 - 1600	GNSS satellites

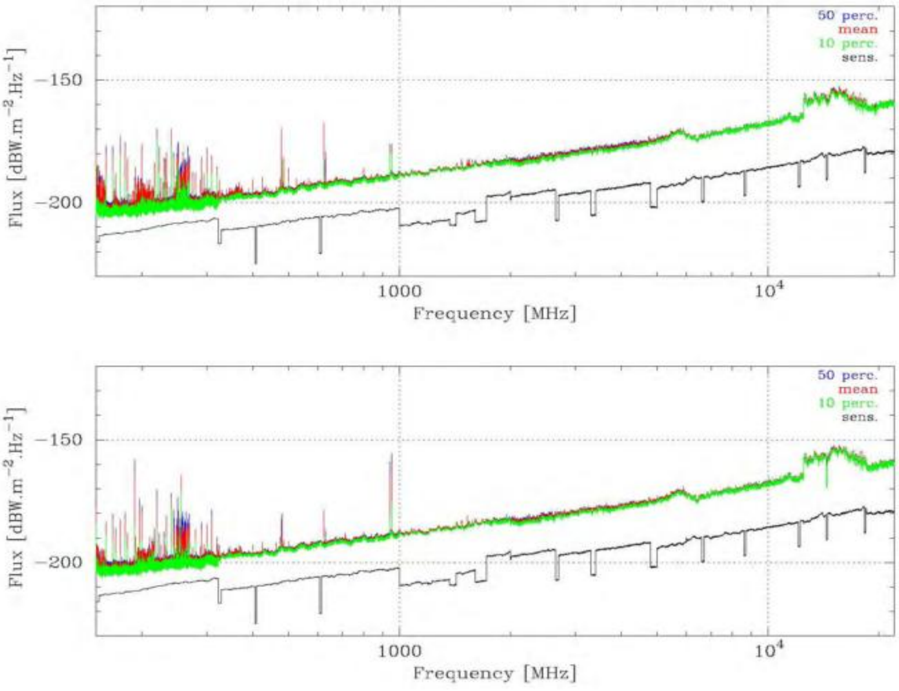


Figure 19 - Summary of Mode 2 measurements over the frequency range 150MHz - 22GHz. The top plot shows the results for vertical polarisation and the bottom plot shows results for horizontal polarisation.

Figure source : (Manners, 2007)

In conjunction with the site measurement campaign a database of transmitters in South Africa was also built with the help of communications regulator ICASA and the main telecommunications operators. This database was used to estimate the effect of those transmitters using standard propagation models. Two independent commercial organisations performed these propagation studies for transmitters up to 200 km from the site and even further for broadcast transmitters(SKA SA, 2006a).

These propagation studies revealed some issues with the RFI environment. Broadcast and other radio communication services exceed the thresholds of ITU-R RA 769 even in remote areas (SKA SA, 2006a). The distribution of radio communication transmitters is not tightly correlated to population density as rural transmitters are often high-powered to improve coverage. Local mountains (especially to the south) are an effective shield of local transmitters and tropospheric conditions significantly affect the propagation of distant transmitters.

In addition to studying the propagation of known transmitters, a reciprocal propagation study was conducted (SKA SA, 2006a). For these studies a hypothetical transmitter is placed at one of the sites and run through standard propagation models, the result of this modelling is shown in Figure 20. These studies can be used to find an appropriate size for a RQZ. From these studies it was determined that even weak, distant transmitters can emit signals that exceed the ITU-R RA 769 thresholds (SKA SA, 2006a). The implication being that it is impossible to limit broadcast and other radio communication signals to ITU-R levels at any site in the world

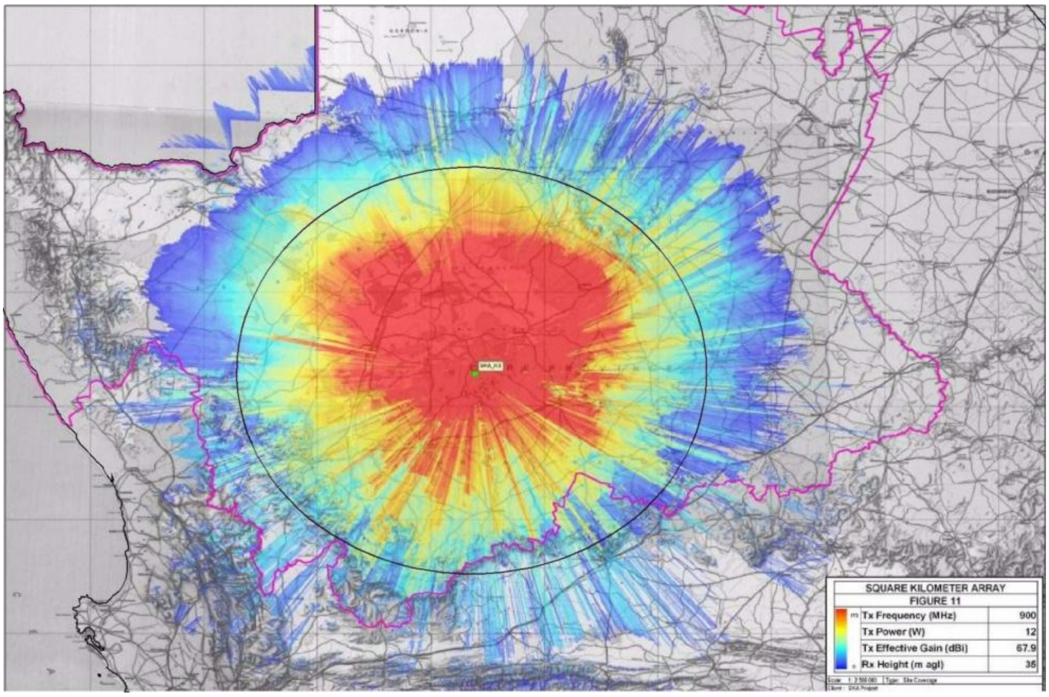


Figure 20 - Example reciprocal propagation study for a hypothetical 12W GSM transmitter at the core site. The low (blue) end of the scale represents a signal flux level of ITU-R RA 769-2 continuum threshold. The circle has a radius of 200km. As can be seen this type of transmitter will be above the ITU threshold for over 200km in every direction. Also obvious from the plot is how the mountains to the south of the core provide good protection from RFI.

Figure source : (SKA SA, 2006a)

2.5.2 Current RFI monitoring

The 7 dish interferometer (KAT-7) currently on the MeerKAT site can be used to perform high sensitivity RFI surveys when it is not being used for science purposes. The KAT-7 telescope is used for 2 regular RFI monitoring activities. The first is a 360° scan at 3°, 9° and 15° above the horizon across the frequency coverage of KAT-7 which is 1200MHz – 1950MHz. Figure 21 shows an example of the plots generated by this scan. The data are captured in 1 second accumulations and with a channel resolution of 400 kHz. The data are run through a RFI detection algorithm which flags data which are 11 sigma above the noise. The scan is run twice a week and is meant to monitor the changes in RFI due to the construction of MeerKAT infrastructure.

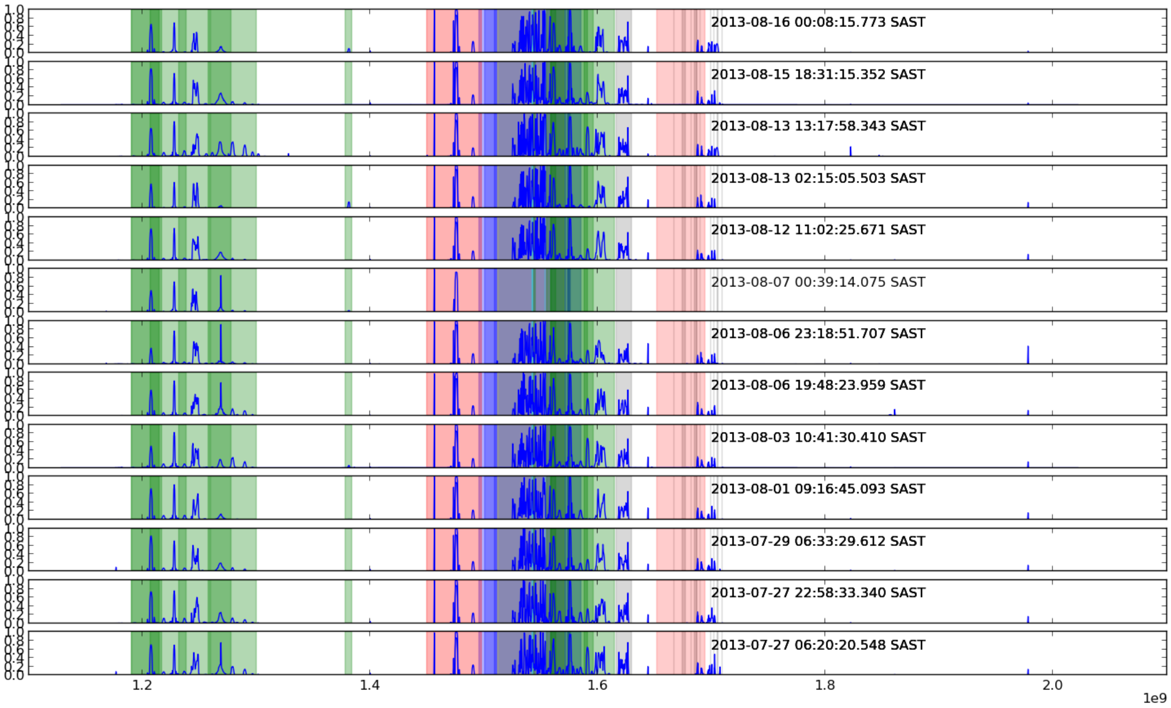


Figure 21 - Example plot of the bi-weekly RFI survey using the KAT7 dishes. There have been 472 of these RFI scans run as of 20/01/2013.

Figure source : (Passmoor, 2013)

The second RFI monitoring activity performs a high elevation scan with a higher frequency resolution of 6.3 kHz wide channels over the same bandwidth of 1200 kHz-1950 kHz. The purpose of this scan is to find narrow band RFI which are hidden in the lower resolution scan. This scan was run because RFI was discovered in narrowband science observations. It was found the RFI was caused by the receiver component of one of the dishes. Although the internally generated RFI does not show up in correlated data, it does raise the noise floor of observations. This scan is run once a month to ensure the problem does not reoccur. RFI can also be detected during science observations by examining the science data. The same software which is used in the monthly scans can be run on science observations to detect RFI during normal observations.

The results of these scans are discussed at a monthly RFI working group headed by the RFI manager for MeerKAT. The RFI manager co-ordinates many RFI mitigation activities such as using spectrometers on site to track down RFI culprits, measuring equipment for potential RFI before it is installed on site, measuring the shielding efficiency of different buildings, modelling the path loss of signals generated on site and measuring representative signals to see whether they fit propagation models.

As of December 2012 there were many RFI mitigation strategies in action at all times with the goal of maintaining the low RFI levels on site, however there was no continuous monitoring system. Although the KAT-7 dishes and other RFI campaigns provide useful, high quality information on the environment, they provided snapshots of RFI. This means that it is likely that there was some RFI which was not monitored as it occurs outside of these snapshot.

2.5.3 ReAl Time Transient analYser (RATTY)

The RATTY instrument was originally developed in order to study the emissions of electric fences around the MeerKAT site. An example of the RATTY in use is shown in figure 22. The system was designed to be portable and consists of an antenna (not shown in the figure), a Reconfigurable Open Architecture Computing Hardware (ROACH) board and a PC. The antenna collects the electromagnetic radiation, the ROACH board performs the analogue to digital conversion and signal processing and the PC is used for control and data storage. The ROACH board is the basic computing unit used in the KAT7 correlator; we will discuss the ROACH board in more detail in the technology section of this chapter.

The RATTY can be configured and calibrated to run using many different types of antenna and ADCs. It can be set to any number of channels, bandwidth or integration time. The states of physical LEDs on the RATTY can be configured to display the status of different control bits in the ROACH registers and the IP address of the RATTY device can be updated.

The system can record either time or frequency domain signals. In the case of time domain signals the power of the signal can be saved at the highest time resolution possible with the connected ADC which samples at 1.5 GHz. It is possible to capture nanosecond pulses this way, which is exactly the kind of pulses one gets from electric fences. In the frequency domain, the RATTY system can perform a 32768 bin Fourier Transform on the time domain data, resulting in a power spectrum from 0 MHz to 850 MHz with a channel resolution of 42 kHz. These spectra can be accumulated for a variable time to increase the sensitivity of measurements.

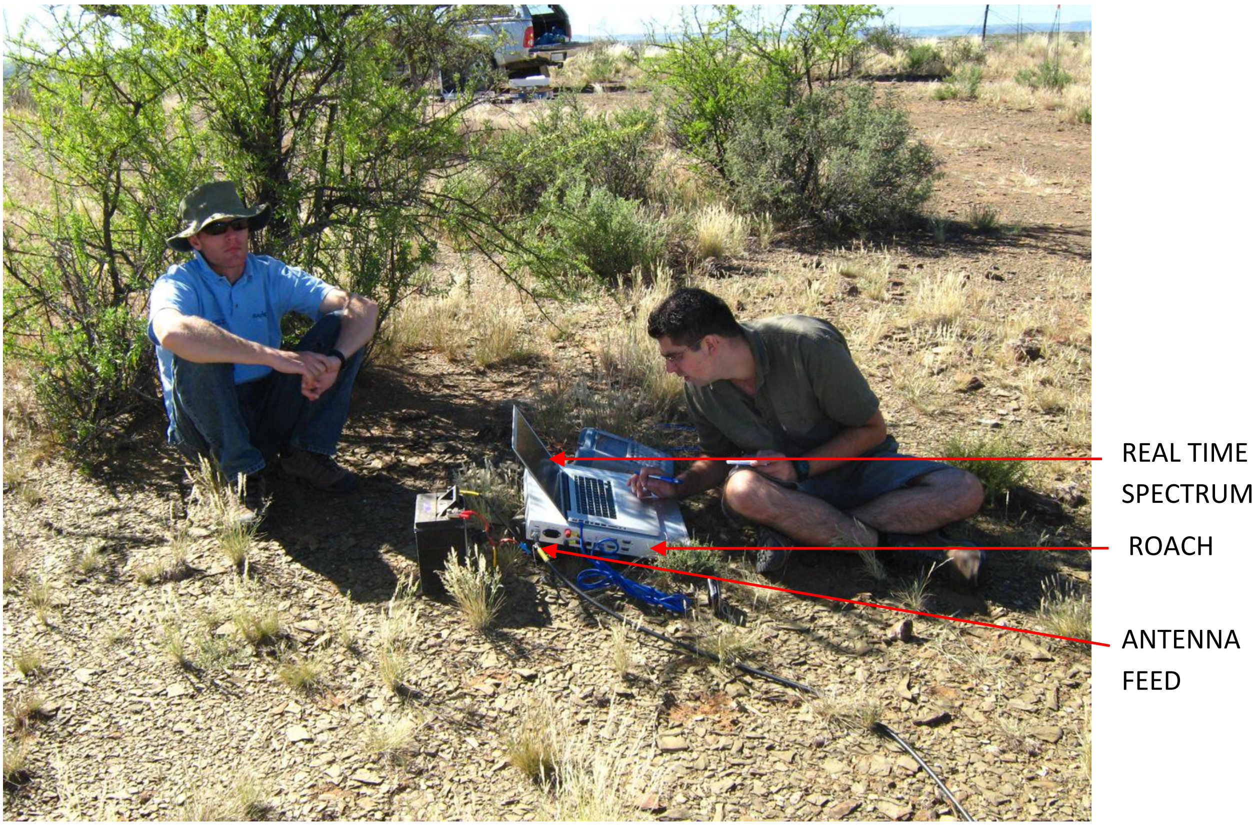


Figure 22 - Example of the RATTY system being used in the field

The RATTY system is currently being upgraded to be used as a more general tool for RFI measurements. As the RFI requirements imposed on MeerKAT subsystems and infrastructure are extremely stringent the RATTYv2 is being designed to provide measurements which can verify these requirements. The RATTYv2 will have bandwidth of 50 – 2600 MHz with a channel resolution of 1 kHz to 20 kHz.

2.6 RFI Detection Algorithms

Measuring the spectrum is the first step in RFI monitoring. After collecting data we must detect RFI in that data. It is possible for an expert to visually inspect the data and find RFI, however this is a time consuming process. Using signal processing it is possible to perform automatic RFI detection. It is important to ensure that our RFI detection algorithms are effective; we do not want to falsely claim a spectrum is free of RFI because our detection algorithm is not sensitive enough. We also do not want to spuriously classify clean spectra as contaminated with RFI, as this will result in resources being wasted trying to track down RFI culprits which do not exist. This section discusses what types of RFI detection algorithms are appropriate for the type of data that the RATTY system collects.

2.6.1 Data

Before describing the different RFI detection algorithms, it is important to have an understanding of the data. Firstly the data is a quantisation of a continuous signal. In Figure 23, we see that a continuous signal can be converted into 1 of 4 digital values using two bits. The figure shows that the digital values that we perform analysis on are not exactly the signal which was received by the antenna. An Analogue to Digital Converter (ADC) can only sample at a certain rate and we cannot see features which occur over a shorter time frame than the sampling rate of the ADC. The ADC quantises continuous voltages into discrete voltage levels at discrete intervals. As a result the digital value will always be some distance from the actual value; we call this the quantisation error. It is possible for the real value to be outside of the range of possible values that can be quantised, in this case we do not know how powerful the real signal was at all, simply that it was higher/lower than the maximum/minimum value of the ADC.

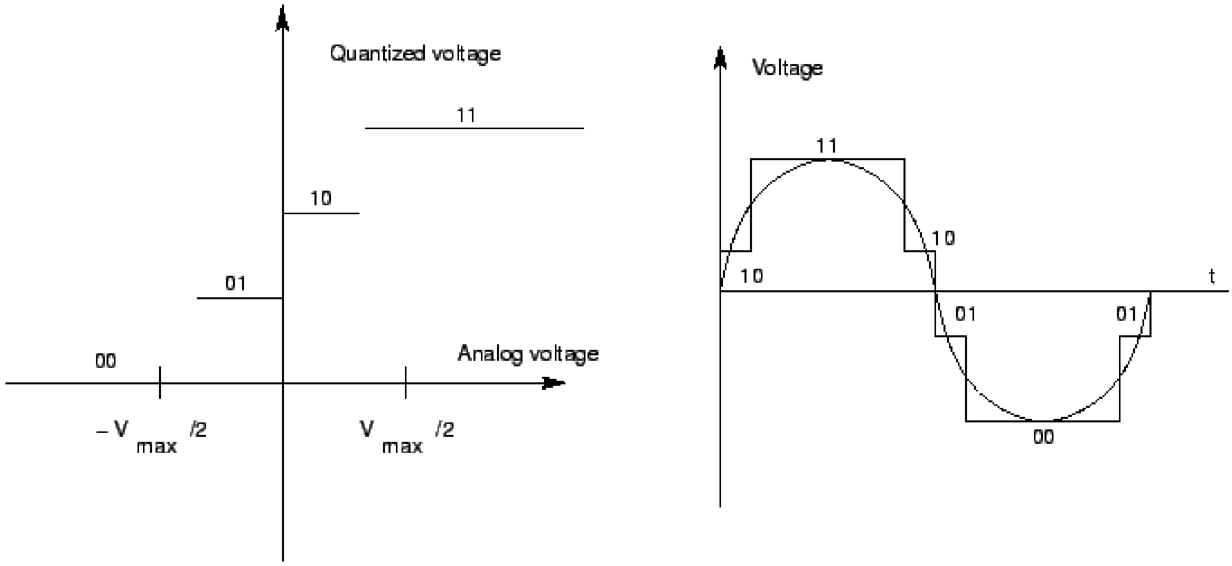


Figure 23 - The figure on the left shows how the analogue values map to digital values for a 2bit ADC. The figure on the right shows an example quantisation of an analogue signal

Figure source: http://gmrt.ncra.tifr.res.in/gmrt_hpage/Users/doc/WEBLF/LFRA/node66.html

Although it is possible to perform RFI detection on the voltage data, this only tells us at what time RFI was present, not on which frequencies it was present. It is possible to calculate the contribution each frequency makes to total power by using a technique called the Fourier transform. As we perform our Fourier transform on digital data, the output is a spectrum showing what contribution was made by frequency channels across the bandwidth of our receiver. Each channel's power is the combined power of all of the frequencies within that channel's band of frequencies.

Figure 24 shows what the data coming out of a Fourier transform would look like. At each time step we can see that some frequencies contribute more power than others. This allows us to see which frequencies have the highest power emissions. As many RFI sources emit on specific frequencies, this information allows us to narrow down the possible sources of RFI in our captured data.

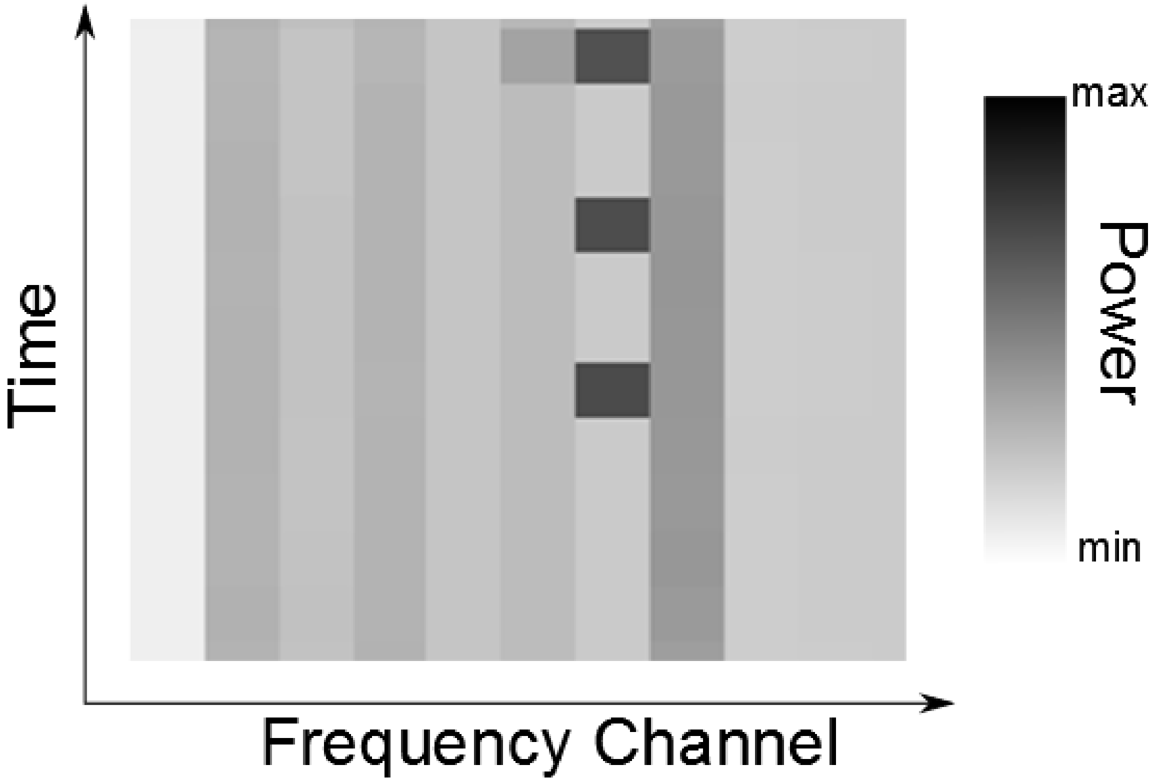


Figure 24 - Example of data after it is Fourier transformed. For each time step we get the contribution of each frequency channel to the total power

Finally the data is a combination of signals from different sources with different power and probability distributions. The received power signal can be represented by the formula

$$x(t) = x_{sig}(t) + x_{sys}(t) + x_{RFI}(t) \qquad \text{(Fridman \& Baan, 2001)}$$

$x_{sig}(t)$ is the signal of interest, or in the case of RFI monitoring, the background noise. $x_{sys}(t)$ is the noise introduced by the receiver. Both $x_{sig}(t)$ and. $x_{sys}(t)$ have a normal distribution with a mean of zero and will generally be low powered. $x_{RFI}(t)$ is the signal/s of RFI, they can have many kinds of probability distributions and power levels as there are many different types of RFI.

2.6.2 Frequency and Temporal Thresholding

One characteristic which separates RFI and astronomical signals is that RFI can be orders of magnitude more powerful (A. Offringa, 2012). This characteristic makes thresholding techniques useful, although other techniques are needed to detect low powered RFI. It is possible to set a constant threshold for all channels based on the knowledge of the RF environment; however, there are more sophisticated techniques which estimate thresholds based on the neighbourhood around a data point. Thresholding techniques are divided into two categories, those which flag in the temporal domain and those which flag in the frequency domain.

Temporal thresholding can be performed in the time domain, before or after the signal is Fourier transformed into a power spectrum. In the case that the signal is flagged before being transformed, flagging the data means that all frequency data is lost, whereas performing the flagging after a Fourier transform allows for only those frequency channels with RFI to be flagged. The choice is a trade-off between the extra time and computation required to perform a Fourier transform and to perform RFI detection on each resulting channel versus the extra sensitivity you get by not throwing away channels with no RFI. Frequency thresholding can only be performed after a Fourier transform.

The algorithms for thresholding all work the same way in the time or frequency domain. For both cases the higher the resolution of the data, the more chance there is that RFI will be detected (Guner, Johnson, & Niamsuwan, 2007). Other than a simple global threshold, it is possible to define a threshold based on the varience of spectrum or channel. If the distance of a data point is far from the median or mean of the data in terms of its standard deviation, that data point is likely not normally distributed, in that case the signal $x_{RFI}(t)$ must dominate and the data point can be flagged (Guner et al., 2007).

We will now describe the steps needed in the algorithm. For our purposes we will detect RFI in a recorded power signal $x(y) = x_{sig}(y) + x_{sys}(y) + x_{RFI}(y)$ where y varies either in time or frequency. We are interested in determining whether the power at $x(y_i)$ is contaminated by RFI.

1. Calculate an estimate of the varience in a window of length $2k$ centred at the point $x(i)$, we will call this estimate $VAR[x_{i-k} \dots x_{i+k}]$. From this we can estimate the standard deviation $\sigma_i = \sqrt{VAR[x_{i-k} \dots x_{i+k}]}$
2. Calculate an estimate of the mean or median of the window around the point $x(i)$, we will call this estimate $AVE[x_{i-k} \dots x_{i+k}]$ = a_i

3. With T a threshold determined by the properties of the environment

   a) If $x(i) - a_i > T\sigma_i$ then flag $x(i)$ as RFI

   b) Else if $x(i) - a_i <$ - $T\sigma_i$ then flag $x(i)$ as Unknown

   c) Else $x(i)$ is RFI free

The result of the algorithm will be either (a) $x(i)$ is most likely RFI as it is significantly more powerful than the window, (b) $x(i)$ is significantly weaker than the window or (c) $x(i)$is within the range we accept as unlikely to be RFI (P. A. Fridman, 2008). The outcome (b) deserves some more discussion; this result is interesting as it means that the power dips significantly from the immediate neighbours. In the temporal case this could mean that a long lasting RFI pulse (i.e. at least half the length of the window) has stopped emitting. In the frequency direction, this is less likely to be a result of RFI and may indicate an absorption line in a spectrum.

It is important to ensure you have a good estimate of the standard deviation, which is denoted by σ. In radio data with interference the combined signal is not strictly normal as RFI introduces strong outliers. The outliers significantly affect both the mean and standard deviation calculated using

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \; and \; \mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

It is important to find robust ways of calculating statistical properties of the data to make thresholding as effective as possible(P. A. Fridman, 2008).

There are two ways we will use evaluate a robust estimates of statistics. The first is the breakdown point which describes percentage of the data that can contain outliers before the estimated value differs from the actual value. For a mean or varience calculated in the standard way this value is 0 as one outlier can alter the estimation. The breakdown point of the median is 0.5 as 50% of the data must be outliers before it affects the estimation of the median. The stability of an estimate is achieved at the expense of effectiveness, in the absence of outliers the robustly estimated varience is larger than that of a standard estimate. This ratio of the true varience over the estimated varience is called the LOSS of the robust estimate; the LOSS of an estimator can be in the range $(0, 1]$ with a value of 1 meaning the estimate is in total agreement with the true varience. In deciding a robust estimate of the varience it is important to find a robust estimate where the breakdown point and LOSS are appropriate (P. A. Fridman, 2008).

The **median absolute distance (MAD)** estimator is a robust estimator. The median of distances from the median is calculated from a sample $X = \{x_1, \dots, x_n\}$. The median is found by ordering the data such that $x_{(1)} \le x_{(2)} \le \cdots \le x_{(n)}$. The median is the value $x_{({}^{n}/_{2})}$, or if n is odd we can interpolate the median as $\left(x_{({}^{n-1}/_{2})} + x_{({}^{n+1}/_{2})}\right)/2$. The varience can then be calculated using

$$\sigma_{MAD}{}^2 = 1.4383 \ \times med_{1\le i\le n}\{|x_i - med(X)|\}$$

The MAD estimation lowers the varience of the data, the coefficient 1.4383 corrects for this underestimation. The breakdown point of the MAD estimator is 0.5 and the LOSS is 0.6 (P. A. Fridman, 2008). This means that the MAD estimator will be robust up until more than half of the data points are outliers.

2.7 Technology

In this section we will discuss the different technologies available to provide RFI monitoring at MeerKAT.

2.7.1 Databases

Databases are essentially collections of information which are stored over a period of time. As this is a problem which must be solved by many different systems, there exist different database management systems (DBMS) to provide reliable data storage and recovery. As any RFI monitoring system is going to have to store spectral information over long periods of time, it makes sense to use a DBMS to store and access this data(Garcia-Molina, Ullman, & Widom, n.d.).

A DBMS allows users to create a database with a specified logical data structure (schema); it should be able to retrieve (query) the data using an appropriate language; allow the storage of large amounts of data; allow for the recovery of data in the case of errors or misuse and control who gets access to the data (Garcia-Molina et al., n.d.). As DBMSs have been used for 50 years, there are many mature and reliable DBMSs to choose from.

Most DBMSs are relational databases where all data are logically stored in a table. This allows the user to work with a high level language to perform logical operations and create relationships on the data without knowing the data storage structure. Separating the data structure allows for more complicated and more efficient structures to be used while allowing the database to be conceptually simple. Relational databases are the de facto standard for databases which must store large amounts of data.

In order to create, update and access a relational database we use a programming language called a structured query language (SQL). SQL is a language which allows for data definition for declaring database schemas and for data manipulation. It allows the user to query and modify a database and allows users to create relationships between different data fields. It also allows users to perform queries based not only on the values of fields, but the relations between them.

There are a few different options of DBMSs. Some of the most popular options are Oracle, MYSQL, PostGreSQL, Microsoft SQL server and IBMs DB2. Of these MYSQL and PostGreSQL are open source packages. This means they are free but lack some of the features of the other offerings.

2.7.2 Highly Definable File Format

The KAT7 telescope uses the Highly Definable File v5 (HDF5) format to store most of its data and the MeerKAT telescope is planned to use the same format. HDF5 is a technology suite which includes a portable file format, a software library to efficiently access and manipulate those files with interfaces for most common programming languages. It also includes tools for viewing, manipulating and analysing that data. The HDF format was developed by the National Centre for Supercomputing (HDF Group, 2011). The development has since been taken up by the HDF Group which is responsible for the latest version of HDF and its accompanying technologies called HDF5. The HDF5 data model, file format, API, library, and tools are all open source and can be used free of charge.

The HDF format allows users to define their own hierarchical data format which can store multiple data groups each of which can contain multi-dimensional arrays with associated metadata (HDF Group, n.d.-a). The format allows for the creation of HDF5 groups and datasets. A group is a structure containing HDF5 objects which could be other groups or datasets; each group also contains a set of metadata which describes that group. The contents of a group can be defined by the user. A dataset is a multi-dimensional array of data along with metadata.

HDF5 files have some useful features; they can be compressed with a range of algorithms. This compression can be performed on subsets of the file or chunks of a dataset. This allows access to parts of the file without having to decompress the entire file. There is also a parallel version of HDF5 which allows for parallel access to files using standard libraries such as OpenMP and MPI (HDF Group, n.d.-b).

2.7.3 Web

For RFI monitoring systems it is important that information about RFI is easy to access and that observers can be alerted to potential RFI as quickly as possible. We have seen that at some telescopes the RFI monitoring systems use static PDF files or offline data analysis and special software packages to provide data to users. These systems either do not provide immediate access to RFI data or required access to the RFI monitoring servers. Although these methods have been successful, it is possible to provide data quickly to users on client machines far faster. Widely disseminating information to computers without using any specialised software other than a web browser is exactly what the web was designed to do. As such the web should always be considered when information must be shared easily and quickly.

The World Wide Web is a set of technologies which allows for the sharing of information over the internet. It is based on 3 fundamental technologies developed in the 90s (World Wide Web Foundation, 2012); these are the HyperText Markup Language (HTML) a language for formatting documents; the Uniform Resource Identifier (URI), a unique address for each web page on the web and Hypertext Transfer Protocol (HTTP), which allows for retrieving resources across the internet using a link on a webpage. Web uses the internet to make access to data around the world simple and universal, that is anyone with an internet connection and a web browser has access to all of the information on the web.

In order to provide interoperability of services on the web, an organisation called the World Wide Web Consortium (W3C) develops standards for web services, servers and browsers. These standards are implemented in many open source and proprietary solutions. The standards ensure that web pages and services are interoperable regardless of who implemented them. This set of standards is what allows for virtually any kind of computer with a compliant browser to access most web sites and services. The easy access to information provided by these technologies is what makes the web arguably the most powerful communication medium on Earth.

Two important standards developed by W3C make designing attractive and functional websites simpler are Extended HTML (XHTML) and Cascading Style Sheets (CSS). XHTML provides a language for defining the structure of the web site. XHTML includes the ability to publish documents, text and images, provide hypertext links which point to other online information and to embed web applications directly into a webpage. CSS is a language for describing the presentation of the information in HTML; CSS defines the colours, layout and fonts of the page. CSS also allows for the content of the web page to be displayed differently according to the device the page is viewed on (World Wide Web Consortium, n.d.-a).

One of the most powerful technologies available for web development is scripting, mainly performed using a language called JavaScript. JavaScript is a programming language which does not need to be compiled to be run. The browser will run the code straight from the source file. JavaScript allows for the web page to be dynamic; content on the page can be altered and content can be sent from the page to a server without resending the HTML or CSS pages. Data can be sent which simply update elements already present. The scripts also allow web pages to incorporate data from the user's environment such as location, local time, etc. This added dynamism allows for web pages to act as fully fledged applications, much like traditional programs. The caveat being that the web browser used must implement all of the required functionality JavaScript used in the application. JavaScript is generally slower than native code such as c++ and different browsers will have different performance (Webber, 2011; World Wide Web Consortium, n.d.-b).

HTTP is the protocol used to send information between applications on the web (Kristol, n.d.). The HTTP protocol is a request/response protocol, where one application must request information in order to get a response containing the information it requires. HTTP is used on top of the normal internet protocols, normally the Transmission Control Protocol (TCP).

Most of the time people deal with the client side of the web, using their PCs or other devices to access information which is being served. However to create a website a server of some kind is needed. A server can be divided into 2 sections, the hardware and the software. The hardware is generally a computer which is connected to the internet. The server software responds to HTTP messages sent over the internet by clients who are interested in accessing information on that server.

Servers contain the files required to serve a webpage to clients. These include the HTML, CSS, JavaScript files and content, such as images, sounds, videos and documents. These files are stored using the basic file system, a database or a content management system. The Server software decodes a HTTP request and returns the files needed to render a webpage. There is a large and growing number of possible server software available. The most popular of these software in order of popularity are Apache, Microsoft and nginx which are used by 42%, 29% and 14% of websites respectively (Netcraft, 2014).

2.7.4 Reconfigurable Open Architecture Computing Hardware (ROACH)

The ROACH board is a primary building block of the signal processing chain for many modern radio telescopes, including the KAT7 and PAPER telescopes. The ROACH board was developed largely in South Africa by MeerKAT engineers as part of the Collaboration for Astronomy Signal Processing and Electronics Research (CASPER) (CASPER, 2013; Parsons, Werthimer, & Backer, 2009). The idea behind the ROACH board was to facilitate the development of signal processing back ends. Rather than developing Application Specific Integrated Circuits (ASIC) and customised networking protocols for radio telescopes, the ROACH philosophy is to use commodity products where ever it is advantageous. The ROACH board is built using commodity processors and networking.

Previously the signal processing systems of radio telescopes needed to be custom built due to the costs of Digital Signal Processing (DSP) technology. However this approach requires years of development and integration in order to create a functioning telescope. As the commercial use of DSP technology has increased, advances in commercial DSP technology have made the older approach counter-productive (Parsons et al., 2009). By using commodity hardware, all advances in the commercial DSP field can be included in telescopes. This allows engineers to focus on the construction of scalable, general purpose solutions to radio astronomy DSP problems. New telescopes can then use the solutions of older systems scaled appropriately, rather than designing new systems (CASPER, 2013).

The ROACH v1 board is a DSP computing platform designed with these ideas in mind. The main processing is performed by a commodity Virtex5 (CASPER, 2013) based Field Programmable Gate Array (FPGA). FPGAs are chips which can be programmed after manufacture. Unlike an ASIC which has a pre-determined function an FPGA can be re-programmed to adapt to new standards and applications.

It is possible to develop libraries for FPGAs which allow the functionality of algorithm to be parameterised and installed on different FPGAs (Parsons et al., 2009). This allows for a DSP algorithm to be taken from an old FPGA and redeployed on a newer, faster FPGA, or to scale the algorithm up or down depending on the particular application. Upgrading or altering ASICs on the other hand requires designing and manufacturing a completely new chip.

The other major advantage of ROACH boards is that they use commodity networking hardware and protocols. This means that it is possible to connect the ROACH to any other commodity hardware which has implemented the same network interface/protocol. This allows the ROACH board to be used in heterogeneous computing systems where data is processed by FPGAs, commodity CPUs and Graphical Processing Units (GPU).

2.7.5 Python

Python is one of the most popular, high level programming languages in use today(Redmonk, 2014). It is an interpreted language which, although it generally runs slower than compiled languages like c++, it also has a lower development time. It is interoperable with many other languages and runs on the major operating systems, with many wrappers for popular languages such as MATLAB, Java, c++ and CUDA (Python Software Foundation, 2012), this makes it very useful for interfacing with diverse systems.

Python is slow for large numerical calculations. There are libraries which can mitigate this problem for scientific applications. The library used by MeerKAT engineers is Scipy. The most fundamental part of Scipy in terms of efficiency is Numpy which implements N-dimensional array manipulation in C (Numpy Developers, 2013). When performing Numpy functions on Numpy data types and arrays, the processing is generally 3 orders of magnitude faster than normal python code.

For these reasons Python is the scripting language of choice used by MeerKAT engineers. The ROACH board has a Python interface and all of the scripts for the RATTY device are written in Python. Python has libraries for manipulating HDF5 files and popular databases, there are also web frameworks developed for Python. It can fulfil most of the scripting needs of the engineers at SKA SA. Also as the development timeline for the MeerKAT telescope is short and the manpower limited, the quick development cycle of Python is useful.

3 Design

In order to provide RFI monitoring for the MeerKAT telescope we designed and implemented a system which samples the Radio Frequency (RF) environment, detects Radio Frequency Interference (RFI) events and makes this data available for the end users. We call our monitoring solution the "RFI monitor" or simply "monitor" for the rest of this thesis.

In this chapter we state the goals of our RFI monitor in section 2.1. The limitations of the available resources are discussed in section 2.2. We describe the users of the monitor and their requirements in section 2.3. We provide the reasoning behind our design approach in section 2.4. Finally we state our design decisions and show how these decisions were informed by the goals, requirements and limitations in section 2.5.

3.1 High Level Requirements

The first requirement of the monitor was to **provide a continuous high time resolution record of the RF environment**. The resolution of the data need to be high enough in the time dimension to describe short spikes in RF that are caused by transient events, such as lightning strikes and sparks which last in the order of 10s of microseconds. As we cannot predict when these events will occur, it is necessary to monitor continuously to ensure that no RFI is missed.

The second requirement of the station was to **provide automatic RFI detection**. This automatic detection allows the system to be an active part of the site management. Rather than simply providing continuous data from which the user can find RFI themselves, this allows the user to be informed of new RFI as and when it is detected. This also allows users to know when and where the most RFI is generated, allowing for RFI to be avoided when using valuable observation time.

The third requirement of the station was to **provide access to the data**. This was an obvious but very important requirement for the station, since the data are not useful if they are impossible to obtain or understand. The requirement was not only to make the data available, but also to provide useful visualisations of the data which allows the user to easily work with and understand the RF environment. To ensure users cannot corrupt the data they are not allowed access to the original raw data files as they could mistakenly alter values or corrupt the formatting.

3.2 Resources

Our RFI monitor was implemented using hardware made available by SKA-SA. As our resources were finite the hardware placed limitations on our monitor. We had to understand these limitations in order to ensure that our goals could be met. We also had access to existing software which provided some basic functionality required by the system. In this section all of the hardware available for the system will be described and the limitations the hardware placed on our system will be discussed. We will also see what software was available and whether this software could be used as part of the monitor.

Table 5 - Table of the resources provided by SKA SA and a description of each resource

Resource	Description
ROACH board	Standalone FPGA processing board based on Virtex5 FPGA
KatADC	8 bit ADC 1.5 GHz max sampling frequency Sampled bandwidth 50MHz to 850MHz
DELL PowerEdge R410 server	2 × Quad-core Intel E5640 @ 2.67GHz 1TB HDD 8GB DDR3 RAM
Rohde & Schwarz HL033 Antenna	Frequency range: 80MHz to 2 GHz Directional Typical radiation pattern shown in figure 26
Spectrum analyser code	FPGA code to calculate a spectrum from time domain power values collected by the ADC
Data collection scripts	Python scripts used to control the RATTY system • Configure ROACH • Start data capture • Display spectra • Save spectra to hdf5 file • Display saved spectra

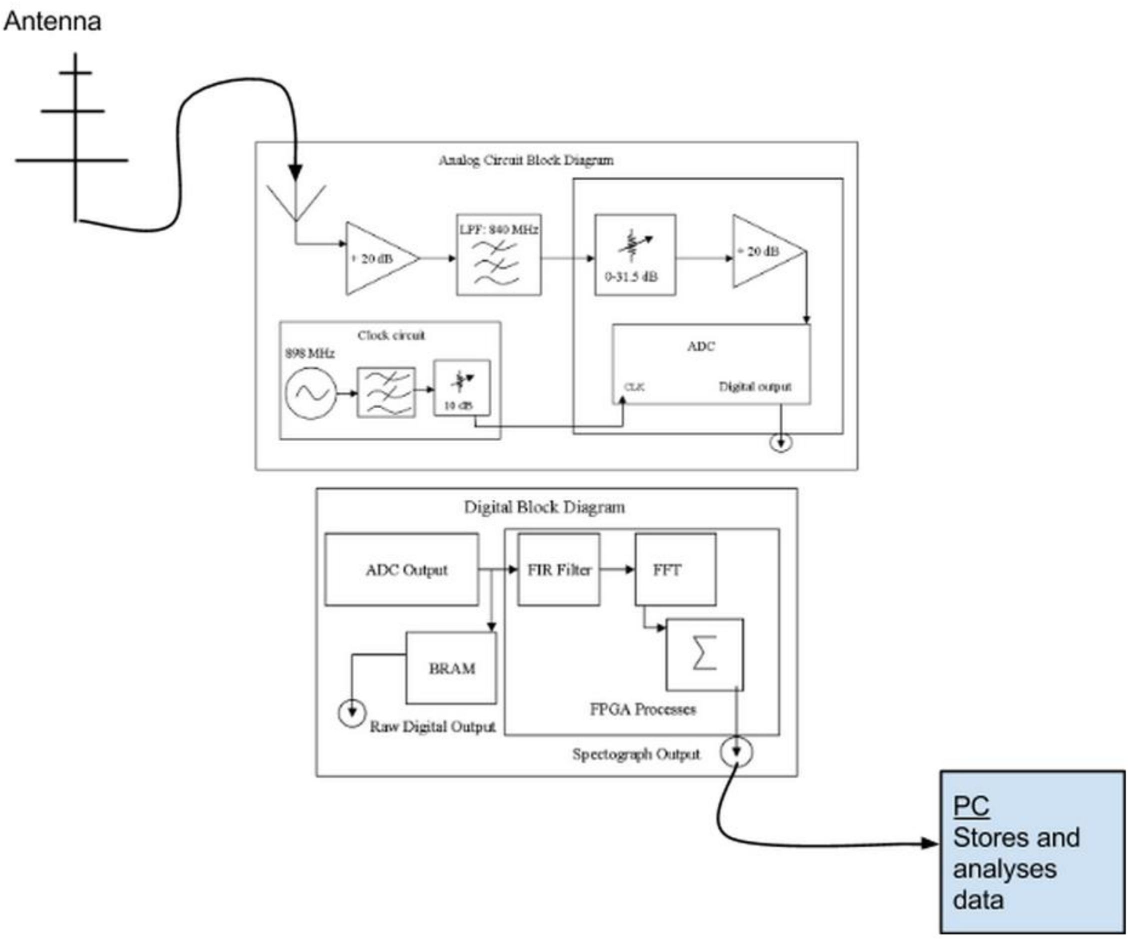


Figure 25 - Diagram of the RATTY system. The antenna is connected to the RATTY via a coaxial cable. The signal goes through the analogue chain and is digitised by the ADC. Then the signal is run through a polyphase filterbank, accumulated and sent to a connected PC which can store and analyse the data.

The resources available were essentially all of the components of a RATTY system along with some extra hardware which was donated by the SKA-SA group. A more full description of the RATTY system can be found in the background chapter. In Table 5 we show each resource along with a short description of each resource.

A diagram for the RATTY system is shown in Figure 25. It shows how the voltage signal from the antenna is sent into the analogue chain of the RATTY system, which amplifies the signal, passes it through a low pass filter, and a low noise amplifier before it is digitised with the ADC. The ADC is connected directly to the ROACH board, the digital signal is processed on the ROACH boards FPGA.

The FPGA is programmed to run the signal through a 4 tap Finite Impulse Response Filter and a 32768 sample FFT. It is then accumulated for a configurable amount of time. At this stage we have an accumulated spectrum. The spectra are sent to a connected PC over Ethernet where the spectra are converted into power spectra and finally saved to an HDF file.

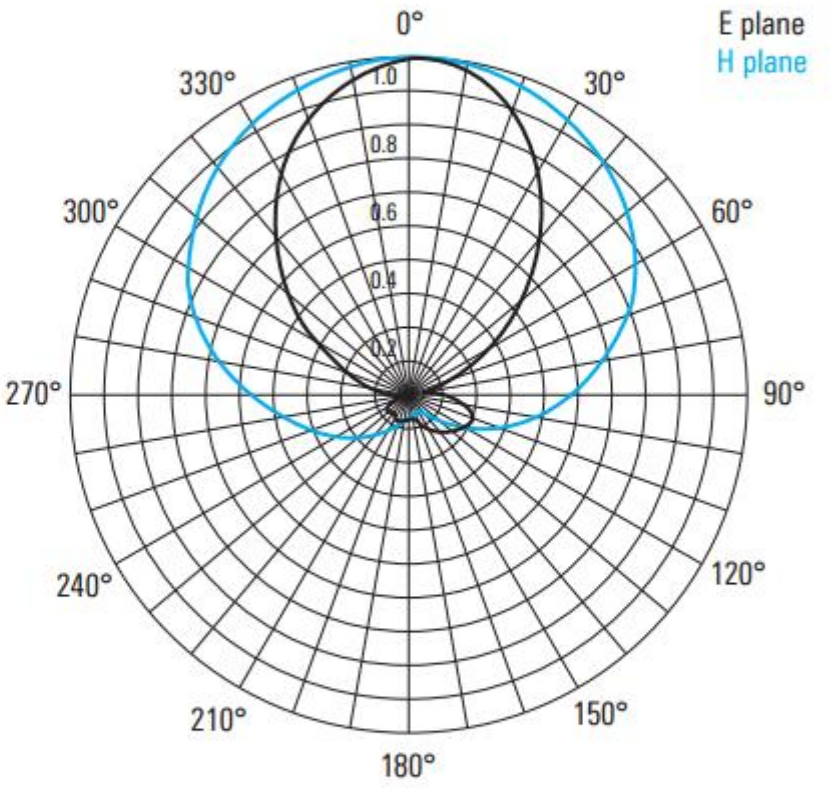


Figure 26 - Typical radiation patterns for antenna
Image src : http://cdn.rohde-schwarz.com/pws/dl_downloads/dl_common_library/dl_brochures_and_datasheets/pdf_1/HL033_cat_2015_78-79.pdf

3.2.1 Limitations

These resources imposed limitations on our system. The limitations are listed below in order of how serious the limitation was to our system.

Limited bandwidth – The ADC could only capture spectra below 850MHz. This meant that our system would only be able to monitor a portion of MeerKAT's observation bands (SKA SA, 2011). As such our monitor was a proof of concept and could not be used for the final monitoring solution until an adequate ADC is used.

Limited storage – An inevitable consequence of continuous monitoring was that the data would eventually grow larger than the hard drive.

Limited memory (RAM) – The data was large. We would not be able to hold all of the data in memory, processing should be performed on pieces of data which can fit in memory.

Limited Processing – We had 8 cores to perform processing; it was unlikely that we would need more processing power.

3.3 Users

The RFI monitor has 3 types of users; astronomers, engineers and site management. Each of these users gains valuable information from the RFI monitor. This section describes how each of these users could use the RFI monitor if it were to accomplish all of the goals stated earlier.

3.3.1 Astronomers

Astronomers are the main end users of the telescope. They are generally skilled in mathematics and physics. They are typically comfortable working with spectra as raw data as well as standard visualisations of spectra. Astronomers are experienced in computers and have some programming experience, most are comfortable working with standard programs such as spread sheets, web browsers and other common computer applications (Asa, 2013).

There are a few possible ways that astronomers would use the RFI monitor:

RFI detection/removal – RFI is likely to contaminate at least some of the bands that an astronomer is observing on. The astronomer must identify RFI present during their observations or risk inaccurate results (Ekers & Bell, 1999). The astronomer can use the RFI monitor to find when and on which channels RFI was detected and whether the power of the signal was high enough to corrupt their observations. This information can be used to find and remove RFI from their observations.

RFI avoidance – When planning their observations astronomers will have access to occupancy charts for each hour of the day and each channel captured by the RFI monitor. They can use this information to decide whether they should use certain channels or times of day for their observations. This is arguably more valuable than RFI detection as the ability to intelligently avoid RFI contamination will lead to less corrupted data in observations (Boonstra, 2001).

3.3.2 Engineers

The MeerKAT telescope is being built by a group consisting largely of electrical engineers. They generally have a very good knowledge of computer systems and programming and most have significant knowledge of the spectrum. They should also be comfortable working with raw data as well as standard visualisations of spectra. Most are comfortable working with standard computing applications and are likely comfortable working with technical applications, using the command line and other more advanced uses of computers.

RFI detection – One of the main goals of the engineers is to ensure that the RFI generated by the instrument itself and the surrounding infrastructure is minimal. They will be able to use the RFI monitor to check if there was a significant increase in RFI after they installed a particular piece of equipment (Boonstra, 2001).

RFI record – Engineers need to know the likelihood of strong narrow-band RFI so that they can design the signal chain with enough headroom. Otherwise strong RFI could damage the system. If they find that there is some RFI in observations and engineers are unsure whether this is internally or externally generated, they can check whether the RFI was captured by the monitor. If both MeerKAT and the monitor capture the same signal, then that signal cannot be introduced from some internal component of the signal pipeline. This will allow them to not waste time looking for possible internal RFI which in fact does not exist (Boonstra, 2001).

3.3.3 Site Management

Site managers are likely to be technically skilled; a dedicated RFI manager will have intimate knowledge of the radio spectrum. However other site managers may not be comfortable working with the raw data and would benefit significantly from clear visualisations of spectral data and automatically generated lists of RFI events (R. Lord, personal communication, 7 June 2013).

RF record - The most valuable part of the monitor for site management is likely to be the continuous record of the spectrum and RFI events. This will allow the manager to find or be notified when there is a significant increase in the amount of RFI being captured. This will allow them to act as soon as a new RFI culprit starts emitting signals. It is important to stop RFI culprits as early as possible before the use of the device becomes habitual, at which stage it will be significantly harder to stop people from using it.

RFI detection and RFI record - Another valuable use of the monitor is that it can be used to prove to a RFI culprit that they are indeed causing an unacceptable amount of RFI. The radio frequencies around the site are protected in a radio quiet zone, which makes the use of certain electronics illegal. However, without any proof, it will be hard to convince locals that a particular device is in fact causing a problem. If it can be demonstrated that a certain type of RFI captured by the monitor only occurs when the device is in operation, this will give the RFI manager far more leverage in confrontations with RFI culprits. If there were no monitor, it is likely that RFI culprits would be noticed later; be harder to find and it would be harder to prove culpability.

3.3.4 User Requirements

From the description of the users above we distilled some requirements of our RFI monitor. These were the data products that the RFI monitor had to make available to satisfy the needs of all users. The requirements and the users of each requirement are shown in table 6.

Table 6 - Table showing user requirements of the RFI monitor

Requirement	**High-level requirement**	**Description**	**Users**
Automatic RFI detection	Automatic RFI Detection	The monitor must automatically detect RFI events	Astronomers Engineers Site Management
RFI record	Continuous high time resolution record of the RF environment	The monitor must maintain a record of every detected RFI event	Astronomers Engineers Site Management
RF record	Continuous high time resolution record of the RF environment	The monitor must maintain a long-term continuous record of the RF environment	Site Management
Access to raw data	Access to the data	Allow experienced users to perform their own analysis	Astronomers Engineers
Data Visualisations	Access to the data	Make large data set comprehensible Allow users with less experience with spectra to use data	Astronomers Engineers Site Management
Descriptive statistics	Continuous high time resolution record of the RF environment	Statistics which add understanding to the underlying signal provided along with spectral data.	Astronomers Engineers Site Management

3.4 Design Approach

The design approach which we used was **evolutionary prototyping**. Evolutionary prototyping is an iterative process which allows for user input throughout the development process. The basic idea with this approach is to provide a basic prototype which can be incorporated into the final system. The prototype is presented to the users who provide feedback. The developer then uses this feedback to design the next prototype which ideally is built on the previous prototype. This process continues until the final prototype fulfils all of the user requirements. The final prototype is the final system, unlike “throw-away” prototypes, where small prototypes are used to test out ideas but ultimately are not used as part of the final system (Vennapoosa, 2012).

The motivation for using evolutionary prototyping for this project was that our monitor should start collecting data before the whole system was complete. This was valuable as the longer the record we have of the RF environment, the easier it is to see long term changes in RFI. Also, as the whole system was supposed to work automatically for very long periods, we needed to test each component for a number of months to ensure it could function over those periods. If we used throw-away prototypes, we would have had to stop testing the functionality of the last prototype each time we implemented a new prototype. This was mainly because we only had 1 ROACH board and antenna, so we could not simultaneously run two instances of the monitor.

Table 7 - The 4 prototypes, the functionality they add and which requirements they satisfy

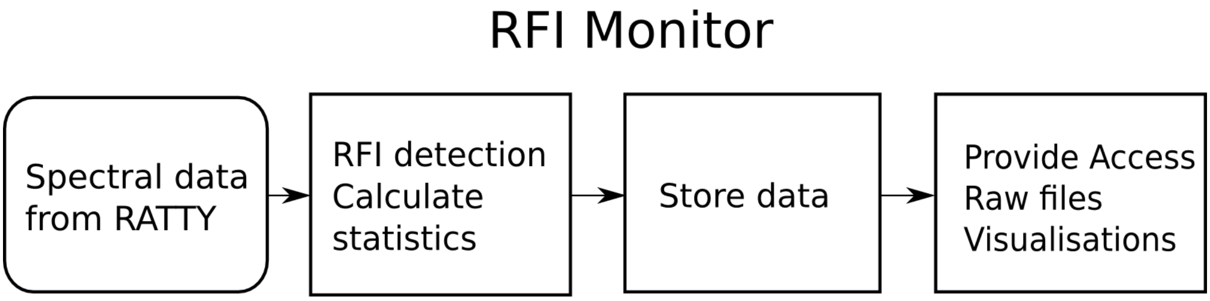
Prototype	Functionality	Requirements satisfied
1 - Data capture	Capture spectra from RATTY Save spectra in searchable format	RF record
2 - Archival	Calculate hourly descriptive statistics Store statistics in searchable format Delete “old” raw data	Descriptive statistics
3 - RFI detection	Perform RFI detection on raw data Extract RFI events Calculate descriptive statistics from RFI events Store in searchable format	Automatic RFI detection RFI record
4 - Access	Create web site with access to: Raw data Visualisations	Access to raw data Data visualisations

For our monitor we designed and implemented 4 prototypes to provide and test the functionality of data collection, storage and access. Table 7 shows the 4 different prototypes and the requirements which they satisfied. The first prototype provided the ability to capture and store raw spectral data. The second prototype provided the calculation of descriptive statistics and long term archiving of these statistics. The third prototype provided automatic RFI detection and storage. The final prototype added the ability to access the data which the first 3 prototypes collect, calculate and store.

3.5 Design Decisions

After gathering user requirements it was possible for us to get a high-level idea of what the RFI monitor should do. First it had to automatically and continuously gather data from the existing RATTY system. After collecting data the monitor should then run an automatic RFI detection on collected spectra and calculate a number of descriptive statistics that astronomers and engineers use to better understand the spectra. The raw spectral data, RFI detections and descriptive statistics are then stored in a searchable format. Finally the monitor had to provide access to these data. Figure 27 presents a high-level diagram showing data flow and processing for the RFI monitor.

Figure 27- High level data flow of RFI monitor


RFI Monitor
Spectral data from RATTY
RFI detection Calculate statistics
Store data
Provide Access Raw files Visualisations

3.5.1 RATTY

In order to use the RATTY system to measure spectra for our monitor, we first needed to alter the existing software to make data collection automatic and continuous. Originally human guidance was needed throughout the data capture process. We needed to alter the RATTY so that it could be started and run indefinitely with no human intervention. The control scripts for RATTY were written in Python. Converting these scripts to perform automatic spectra capture was the first task when developing the initial raw data capture prototype.

3.5.2 RFI Detection

There are many algorithms which exist to detect RFI in a signal; ranging from very simple thresholding to complicated statistical techniques which require intimate knowledge of the RF environment. The method chosen to implement was called thresholding based on the varience of the signal. The reason we chose this method is that it is simple and effective. The algorithm was already being used for the bi-weekly RFI scans and had proven its reliability. We allowed for other RFI detection algorithm to be used instead of thresholding if a better algorithm was found by defining interfaces on both sides of the RFI detection algorithm. Any algorithm implemented with the correct interface would fit easily in the RFI monitor.

Once RFI has been detected we store the detections in the form of RFI ‘signatures’ which could be used to find the cause of RFI. The information that would make up a RFI signature is described below.

Detection time – The time that the RFI signal was detected
Event length – The duration of the RFI event
Average power – The average power for each channel that RFI affected
Max power – The maximum power level for each channel
Min power – The minimum power level for each channel
Median power – The median power for each channel
Low channel – The lowest channel affected by the event
High channel – The highest channel affected by the event

We also store the raw data for the time period and channels which are affected by RFI. This will allow for users to better understand what the source of the RFI might have been.

3.5.3 Storage

Monitoring RFI creates a large amount of data. It is important to store this data in a way which maintains the data integrity and can handle both the continuous input from the RATTY system and intermittent access from users. The best way to achieve this was to use a database. This is because databases are mature technology which has been designed for maintaining large collections of data. All of the databases were implemented in MYSQL because it is a mature and reliable product which is free and many people have experience using it. It is also the format that the current bi weekly RFI scans (See section 2.5.2) are stored. It made sense for all databases to have the same format.

From our requirements we saw that there are **3 different types of data** that users would be interested in. The **raw spectral data** which is provided by RATTY, the **descriptive statistics** which are calculated each hour and the **RFI detection data**. As most of the incoming data is raw data, it makes sense to separate this data into a high-input database which can handle a high amount of continuous input. It was important to make this data easy to search and we should store it in a database which focuses on efficient access.

As the process of this design was iterative there were 3 prototypes implemented to test the design of data capture and storage. Figure 28 shows what processing is performed by each prototype. Each of the three data types are stored in a separate databases; each prototype focuses on providing the data that belongs in each database. In order to effectively use our multi-core machine, each prototype’s processing is performed by a separate thread.

3.5.3.1 Basic prototype

The idea for the first prototype was to provide a database which could store all the data which is captured and calculated each second. The DB had to store at least the raw data and most basic data products to ensure that we could store a complete description of the RF environment with a high enough resolution to be used for the intended RFI detection.

This design was meant to hold just the data which describes the RF environment. It did not contain any extra information on RFI as there was no RFI detection for the first prototype. The goal with this initial implementation was to have a system which could store all of the data at the rate that the RATTY system captures. We also wanted to ensure that the database could respond to requests. This prototype was meant to be a skeleton on top of which the rest of the monitor could be built.

We decided to use the Frequency Domain capture of the RATTY. It was the most appropriate for our continuous monitoring system. The data rate of the time domain capture is simply too high. The trade-off is that more time samples in the particular frequency than is strictly necessary may be flagged. Also, in the event that there is RFI for which we do need to get a high resolution TD scan, we can stop monitoring with the FD mode and use the original RATTY software to perform a TD scan. The functionality will still exist, it will just not be the standard mode of operation

3.5.3.2 Archival prototype

After implementing the above database and running the monitor for 3 months, it became apparent that the data rate of the monitor was too high to store high resolution data permanently. In fact it took only 3 months to fill the entire 1 TB hard drive on which the database was stored. Seeing as this RFI monitor is meant to store many years’ worth of data this was a situation which we had to rectify.

There were two options to increase the lifetime of the monitor storage, the simple solution was to simply increase the amount of storage space. The other option was to reduce the amount of stored data. If we chose the second approach we had to ensure that the data stored could describe the general RF environment and how it changed over long periods of time and also be able to describe RFI events in a high resolution. In both cases we could consider archiving the data on a separate storage medium like DVDs for long term storage, although that data would no longer be easily accessible.

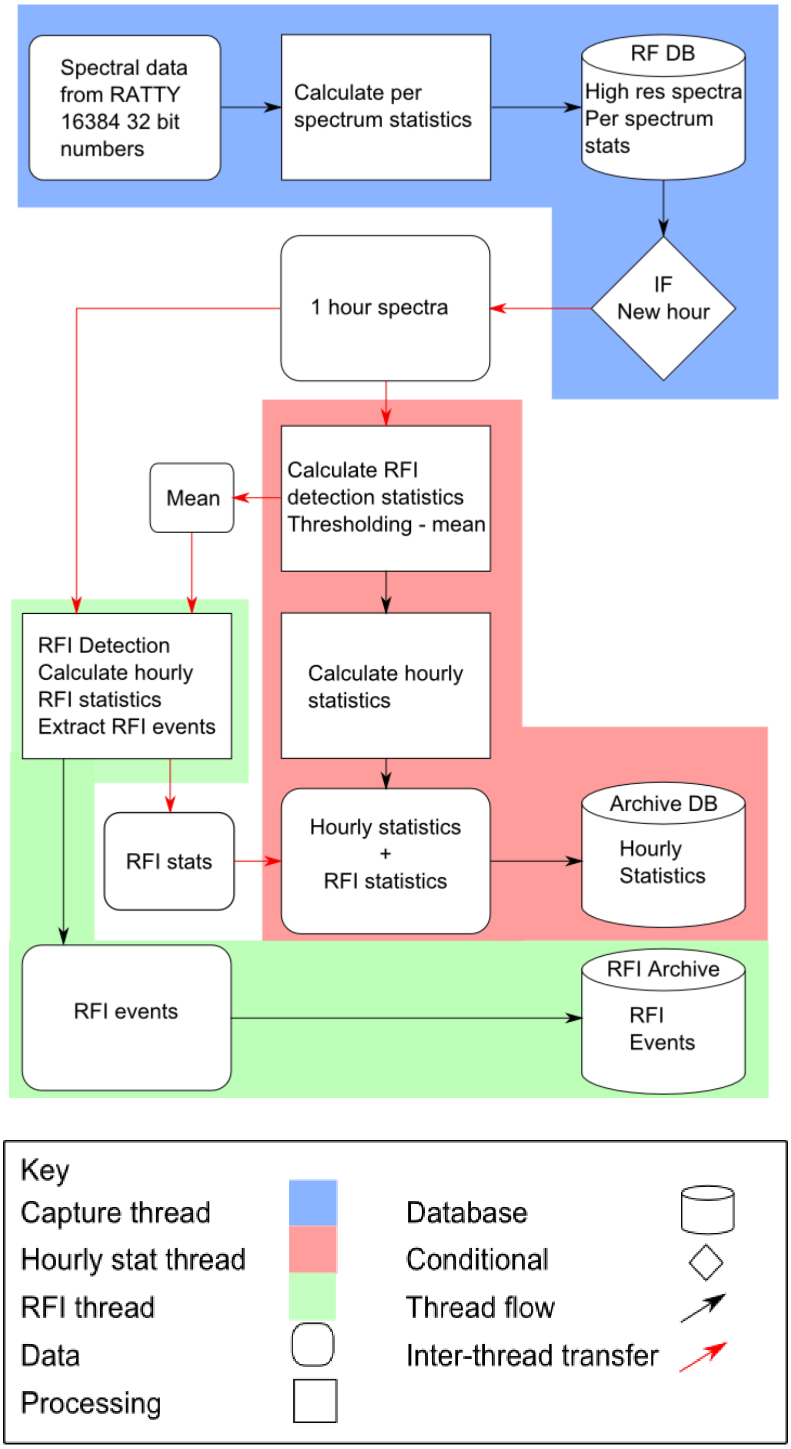

Spectral data from RATTY 16384 32 bit numbers
Calculate per spectrum statistics
RF DB
High res spectra
Per spectrum stats
IF New hour
1 hour spectra
Mean
Calculate RFI detection statistics Thresholding - mean
RFI Detection Calculate hourly RFI statistics Extract RFI events
Calculate hourly statistics
RFI stats
Hourly statistics + RFI statistics
Archive DB
Hourly Statistics
RFI events
RFI Archive
RFI Events
Key
Capture thread
Hourly stat thread
RFI thread
Data
Processing
Database
Conditional
Thread flow
Inter-thread transfer

Figure 28 - Diagram of first 3 prototypes and how they communicate.

We preferred option two, the main reason was that high resolution data was not required to describe the average RF environment or how it changes over years. Hourly statistics are more than adequate for a long term description of the RF environment. We did still need high resolution data to detect and characterise RFI. However, RFI affects only a small fraction of the spectral range captured. It was not cost effective to store data where only a small percentage of that data would actually be required. Since we planned on implementing an RFI detection algorithm, we could use detection information to intelligently ‘compress’ the data.

Table 8 - Table showing the expected data rate for each form of data

Data	Data rate
Raw spectral data	450.00 MB per hour
Hourly statistics	1.06 MB per hour
RFI data	11.25 MB per hour
Total	462.31 MB per hour

We were planning on calculating hourly statistics for each channel which described the RF environment and how it changed over long periods of time. This data had to be stored indefinitely to fulfil our long term record requirement. If the only high res data we stored indefinitely was those parts of the spectrum which were affected by RFI, we would save space while still having a long term record of the RF environment and high resolution data on RFI events. If we assumed RFI to be present in 5% of high resolution data and we knew that the hourly statistics are 150th of the space of the raw data over that hour, then storing only the hourly stats and high resolution RFI spectra we could increase the storage time of the RFI monitor from 3 months to 55 months. If we always stored at least the last 2 months of high resolution data, we can still increase the lifetime of the RFI monitor to 20 months, without adding any extra hard drives. Table 8 shows the expected data rate for each of the data types.

Eventually it would be necessary to either add more hard drives to the monitor or move some of the data on the monitor to another location. This was an inevitable outcome of providing a continuous record of the RF environment. However extending the lifetime of the monitor in this way made it a far more cost-effective solution.

3.5.3.3 RFI archival prototype

We needed to keep a permanent record of all detected RFI, however because of space concerns we chose to delete all raw data after two months of capturing it. In order to keep high resolution data on all detected RFI, we had to extract the high resolution data from our initial database and store it permanently in an RFI archive before deleting the raw data. This database meant that even if a user wanted to access high resolution data on RFI many months after the original data had been removed from our raw database, they would still be able to get high resolution data on the detected RFI events.

3.5.4 Access

We needed to provide an interface through which users could access and interact with the data. The interface had to make it easy to quickly access relevant information. The intention was that the interface would provide visualisations that add value to the data products.

As we wanted to make data access as simple as possible, we decided to provide access using a web site. This meant that anyone with a web browser would be able to access our RFI monitor. Another reason to use the web was that there are many products available to make the web design and implementation process easier.

3.5.4.1 Web Framework

As creating web sites has become such a common task there are many frameworks to choose from which provide a skeleton of a website as well as the ability to generate web pages based on templates and variable data. These frameworks make the process of building and maintaining a website easier and generally allow for a standard and clear separation between content, style and templates.

The web framework we used for the web interface was the Pyramid framework. This is a python based web framework. The reasons for using Pyramid were that all of the software written for the original RATTY system was in python and a lot of the MeerKAT tools were in Python. It seemed prudent to use the same programming language for the whole system. Other than being a python framework, pyramid was powerful enough to provide all of the functionality we required for our interface and had a strong online community with good tutorials.

3.5.4.2 Visualisations

As the amount of data produced by the RATTY device was large, it was not possible to quickly understand the data by simply looking at the raw values. While many users would have the skills to analyse the data, the process of downloading and analysing the data would reduce the ability for the information to be acted upon timeously. It is important that the data is provided in a format which can be quickly digested. For this reason we decided that the monitor had to include some data visualisations to make the information truly accessible.

We decided to provide two visualisations which are standard tools of radio astronomy and RF engineering and also have wide usage in most fields that deal with data analysis. The visualisations chosen were a simple line chart and a waterfall plot (you may be more familiar with the term heat map).We decided it should be possible to save the visualisation as an image file and that graphs/plots should be interactive. The users should be able to zoom in and pan around the data. We also wanted to display the value of any data point which can be selected with the cursor.

Line Charts

Figure 29 shows an example of a 1 second spectrum captured using the RATTY software. The channels shown span from 0 – 900MHz. This plot was produced using the standard visualisation available using HDFView, a standard program to view HDF5 files. We decided to create a javascript linechart library which could plot all frequency channels in one spectrum in the web browser.

For the initial prototype we decided to provide a line chart which updates as new data is captured, allowing the user to see the latest spectrum captured. After this we it would be trivial to use the same visualisation to plot all hourly stats and also allow individual channels to be plotted over time.

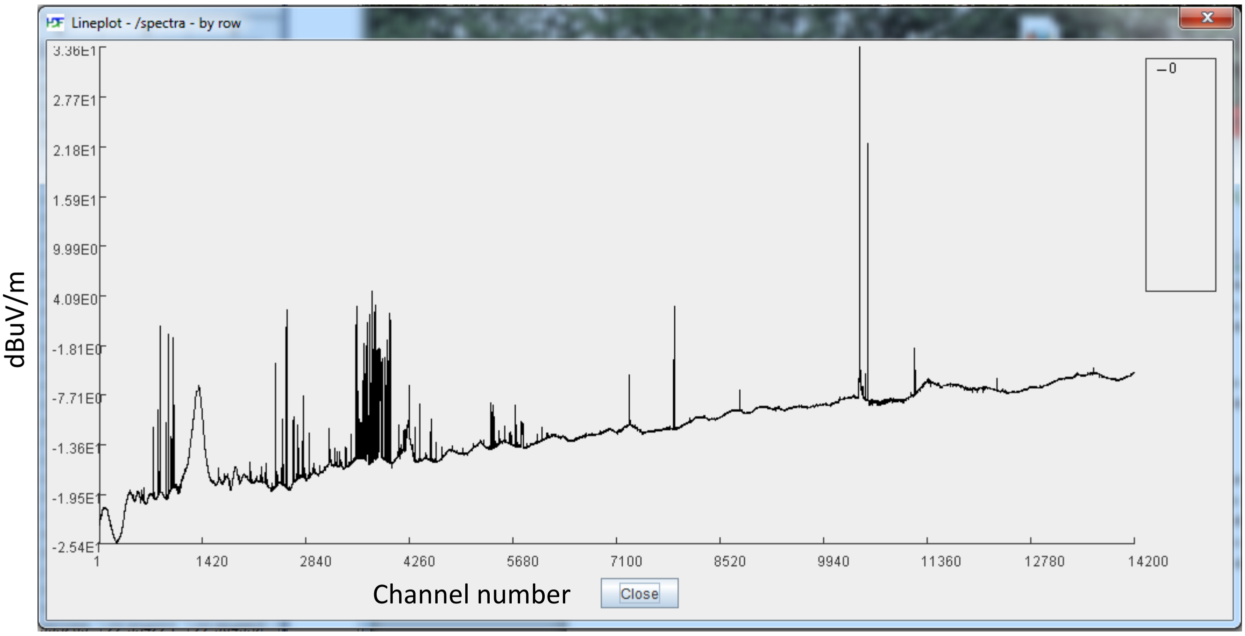


Figure 29 - Example line chart of RATTY spectral data created by HDFView

Waterfall plots

Waterfall plots are a classic way to graph frequency, time and signal amplitude. Figure 30 shows an example with time and frequency on the axes and amplitude plotted as colour intensity. These plots use the same idea as a heat map. These plots would allow our users to view the changes on multiple channels and times at once. We decided the user should also have a overlay which shows when and on which channels RFI was detected by the system. The users would then be able to look at an entire hours worth of data and have a quick visual cue to how much RFI was present across that spectrum over the hour and which channels were worst affected.

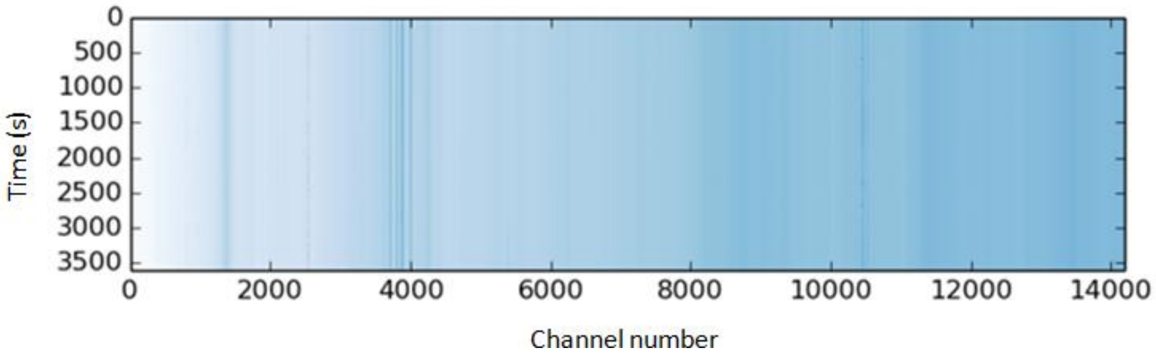


Figure 30 - Example waterfall plot generated with using an hour of data from RATTY. The intensity of the colour maps to the power of the RF at that time and channel.

3.5.4.3 Files

We decided to provide users with the ability to specify a selection of data to download. The data would be available in Comma Separated Value (CSV) format. This would allow astronomers to download all high resolution data from the database after their observations, so long as they got the high resolution data within 2 months of their observation. If data was retrieved after the 2 month deadline there would still be high resolution data on all RFI detected by the RFI monitor. Hence anyone could use the data to perform their own analysis in the program they are most comfortable.

4 Implementation

In this chapter we discuss the implementation of the RFI monitor. In this process, the RFI monitor went through four prototypes. The initial horizontal prototype constructed a basic RFI monitoring system. Subsequent prototypes developed the long term archival of data, RFI detection and visualisation via the web. Here we describe the implementation of the prototypes which culminated in a working RFI monitor on the site of the MeerKAT telescope. In the interests of keeping each section focussed on the main goals of each prototype, the final database structures implemented for each prototypes 2, 3 and 4 have been described in a separate section 4.5. All code can be found in the appendix.

4.1 Prototype 1: Basic Operations

The initial prototype comprised a data collection class based on the original RATTY code, a database to store the data and a web page running on a web framework which provides access to the data. The database and web framework combined make a classic Model, View, Controller (MVC) pattern. The model is all of the data which is contained in the DB, the controller is the web framework combined with the RATTY code and the view is the web pages in the framework and the structural data contained in the HTML. Figure 31 shows the components of the RFI monitor in the initial prototype.

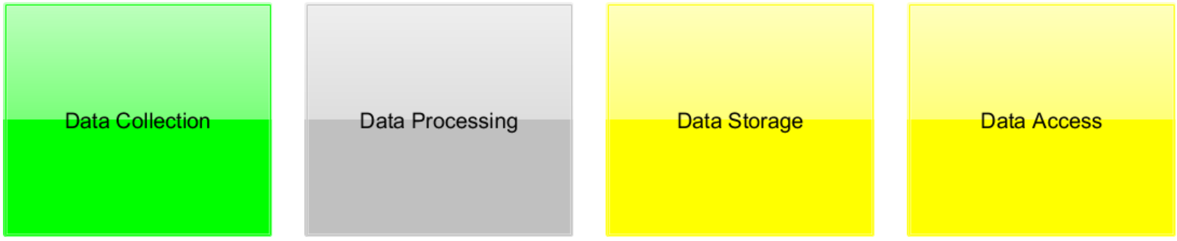


Figure 31 – Components of the initial prototype. Green blocks were complete. Yellow blocks had basic functionality and grey blocks had no functionality in the initial prototype.

4.1.1 Data Collection - RATTY Software Consolidation

The software is reliant on the RATTY board (discussed in section 2.5.3). The original RATTY software contains a script *rfi_spectrum.py* which initialises the RATTY in FD, sets the calibration for the system, sets the integration time of data capture and saves that data to an HDF file. This code was encapsulated in a class which allows integration with other applications. The class *roach_handle.py* contains all of this functionality. The only change to the original code is that the initialisation of the RATTY is moved into two methods. Communication with the ROACH board was moved from the *_init_* script into the *roach_handle* object initialisation. The rest of the initialisation contained in the *rfi_spectrum* script was copied into a method called *rfi_init* which loads the FPGA code onto a connected RATTY device. All of the data access methods were simply copied as they were in the *rfi_spectrum.py* script. This gives us one class, *roach_handle,* which can initialise the RATTY system, calibrate the system, set the integration time, start data collection and pass that data to another application.

Table 9 - Hardcoded values which were moved into the system_parameters file

Hardcoded value name	Description
bitstream	Location of the firmware to load onto the ROACH FPGA
n_chans	Number of channels in each spectrum
n_par_streams	Number of data streams in the FPGA firmware
desired_rf_level	The desired power level of signal
adc_type	Type of ADC connected to ROACH
Spectrum_bits	Number of bits for value of each channel in spectrum
fft_shift	Number of channels to shift fft by
gain_map	Array of gain mappings. This was moved to the gain_map csv. The system_parameters file simply has the location of the gain_map file
bandpass	Location of bandpass csv

The initial code contains two classes, one for control and monitoring (cam.py) and one for calibration (cal.py). The cam.py class was not changed, however the cal class contained hard coded values. These hard-coded values are shown in table 9. We removed the hard-coded values from within the *cal* class and the *rfi_spectrum* script into csv files and a human readable configuration file called *system_parameters.* The antenna calibration and gains are either already in csv files or are hard coded into Numpy arrays. We copied the hard coded values into separate csv files and added the location of those files in our *system_parameters* configuration file. All of the values in the config file can be accessed using python's standard option parser library. Now, to change the configuration of the RATTY system we edit one, human readable configuration file. To calibrate the system for a new antenna or ADC for example, we calibrate that new part of the system and save the calibration data in a CSV file.

This means RATTY can be altered to accommodate new hardware as and when it became available without the need to alter any code. We can simply run the standard python setup.py file provided and the RATTY will be accessible in any python code or the python interpreter by importing the library. After re-structuring the RATTY system had the structure shown in Figure 32.

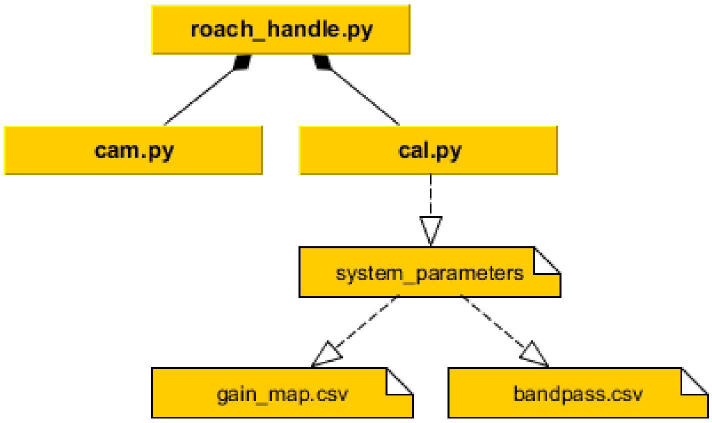


Figure 32 - Structure of roach_handle class. It contains instances of cam and cal to gather spectra with the RATTY. cam.py controls the RATTY and data access. cal.py calibrates the RATTY using data from the system parameters. The diamonds indicate that the an instance of the class is used. The arrows are simply links which point to data in another file.

By this point all initialization was handled by the *roach_handle* class and using a *roach_handle* object we can start the RATTY system collecting spectra. All data could be accessed using the method *get_unpacked_data*. The data is returned as a Numpy array, it is then possible to pass it to code which analyzes the data, displays the data or simply saves the data to file.

4.1.2 Data Storage - Database Prototype

The data capture prototype software has a simple database which holds the spectra captured by the RATTY system and a class to handle databases requests. MYSQL was chosen as the DBMS we would use for our RFI monitoring system.

There are two tables in the database. The *system* table contains configuration information of the RFI monitor; this allows us to recreate the system used to capture a spectrum. The contents of the system table are shown in Table 10. In case we discover that the data is unreliable and need to recalibrate it; we will know the configuration of the system which captured it. Table 11 shows the structure of the spectrum table. This contains the data returned by the *get_unpacked_data* method, this includes the actual spectrum and a timestamp of the spectrum.

Table 10 - The System table. It contains all data on the configuration of the RFI monitor

Field name	Data type	Description
n_chans	integer	Number of channels in each spectra
n_accs	integer	Amount of accumulations for each spectra (determines the rate of spectral dumps)
bitstream	String	The location of the FPGA code
bandwidth	Double	the size of the range of frequencies that can be observed
adc_type	String	the name of the ADC used
spectrum_bits	integer	the number of bits for each value in the spectrum
rf_gain	double	The gain of the antenna

Table 11 - The spectrum table. It contains all the data captured with each spectrum

Field name	Data type	Description
spectra	Variable length string	Spectrum
acc_cnt	integer	The number of spectra captured since capture started
timestamp	Unsigned integer	The time the spectrum was captured as a Unix timestamp
adc_overrange	int	1 if the voltage level was too high to be captured by the ADC for any sample in the sample period, 0 otherwise
fft_overrange	int	1 if the power of a channel of the spectrum was too high to fit in 32 bits
adc_level	double	The lowest power level the ADC could capture
ambient_temp	double	The temperature at the time of capture
adc_temp	double	The temperature of the ADC
system_id	Unsigned integer	The id of the system configuration

The database is accessible by other parts of the RFI monitor code. We coded a python interface between the database and other software. This class is called *dBControl.py*. It contains methods which allow data to be added to the database and to retrieve the latest spectrum in the database, insertDump and getCurrentSpectrum respectively.

For the initial prototype we store all of the spectra directly in the database as a variable length string. The spectra can only be identified by the timestamp values associated with each spectrum. This is also the primary way of querying the database. This means that a spectrum can be found quickly if searched a timestamp is used to identify it.

We created a program that accesses the RATTY and saves data to the database using the dbControl class. We call this the rfi_monitor.py class. The roach_handle and dbControl functions are run on separate processes spawned from the rfi_monitor class. In this way data can be collected from the RATTY using one process dedicated to roach_handle. The rfi_monitor class then co-ordinates passing that data to the dbControl thread; dbControl inserts data into our database.

4.1.3 Data Access - Visualisation Prototype

The visualisation prototype comprised a simple one page website. This one page website displays the latest captured spectrum using a Javascript line chart, Figure 33 shows an example spectrum displayed with this line chart. The Python based web framework Pyramid was used to serve the web page. This makes interfacing with the python based dbControl class simpler.

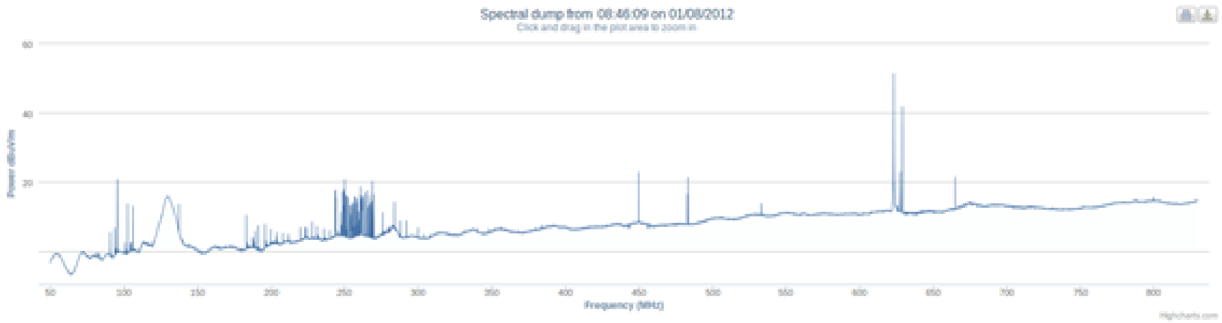


Figure 33 - Example plot of the Visualisation prototype

The pyramid web framework separates the web page's structural and styling content (HTML, CSS) from the server side processing logic. The processing logic is defined as views, which are python methods which return a dictionary containing data required for each web page. The data in the dictionary can by placed symbolically in the HTML files and the pyramid framework will replace those symbols with content created by a view each time the associated page is requested.

The web framework runs from a class called rfiWeb which connects to the database through a new instance of dbControl. The MYSQL database ensures the data is easily accessible by the web server which handles requests from machines across a network. The server provides the latest spectra captured to each client using the pyramid framework. The data is displayed on the client side using the JavaScript line chart. At this stage the RFI monitor had the structure shown in Figure 34.

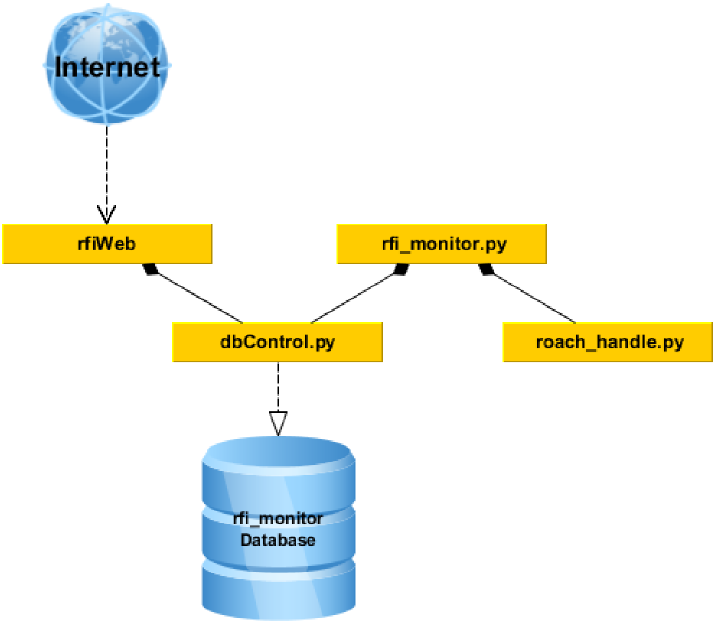


Figure 34 - The structure of the initial RFI monitor prototype. The diamonds indicate a class is called in another class. The arrows indicate a link over which data can flow.

4.1.4 Calibration and Installation

To provide good quality information on the Radio Frequency Environment (RFE) the signal chain must be calibrated. The signal passes from the antenna, through an amplifier, along a coaxial cable and into an ADC. At each stage of this process, the signal is analogue and can be altered by the characteristics of the path it travels along. To calibrate the signal chain, we must determine the gain for the complete signal chain so that it can be de-embedded from the measured signal to regain the incident signal.

The antenna comes with an antenna factor and a calibration file which records the calibration values of the antenna. The signal chain from the antenna output to the spectra output from the RATTY was calibrated using a simple python script to find the highest response of each frequency channel of the RATTY to a constant tone which swept through the band 0-900MHz. The difference between the constant known tone and the output was used to calibrate the response of the RFI monitoring signal chain. This calibration file is called bandpass.csv and can be found in the cal_files folder of the RATTY source.

The final stage of the initial RFI monitoring prototype was to actually install the equipment at the MeerKAT site on Wednesday 18 July 2012. The MeerKAT engineer who designed and built the original RATTY system assisted. The antenna, a ROACH board, a signal amplifier and the DELL server to control the monitoring system were in place.

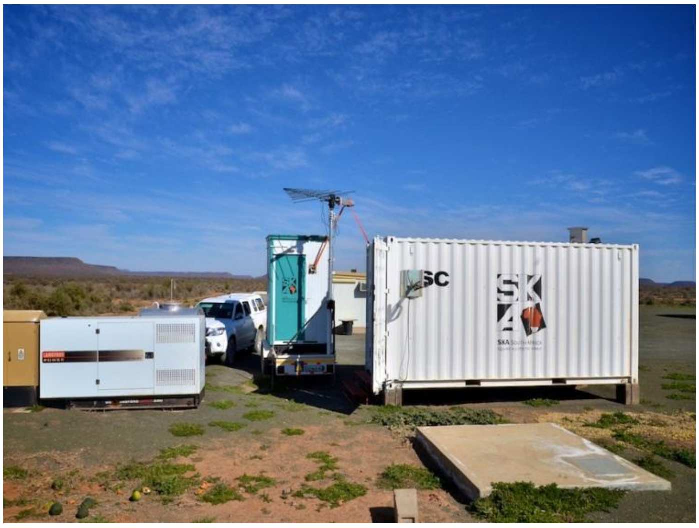


Figure 35 - RFI monitor antenna and ASC container at the KAT7 site

We mounted the antenna on the mast of a disused RFI trailer which had been used to perform site measurements during the 2005 RFI measuring campaign. Figure 35 shows the antenna situated at the current KAT7 site, next to a shielded container which contains the ROACH board and our server.

The ROACH board and DELL server (Figure 36) were installed into a rack in the ASC container. We connected the antenna to the RATTY, attached the RATTY to the server and then connected the server to the KAT network (the internal network between the site and the SKA SA offices in Johannesburg and Cape Town). The network is connected to the internet and can be accessed from any internet connected PC provided we have access to the proxy. This allowed us to continue with the rest of our RFI monitor development from Cape Town, so long as we did not need to physically alter the monitor's signal chain.

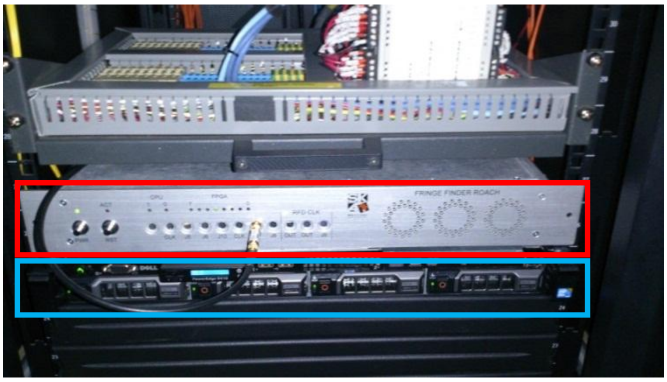


Figure 36- The ROACH board and DELL server that perform the processing for our RFI monitor. The ROACH is marked by the red box and the server by the blue box.

After we had physically set up the RATTY system, we performed the calibration process again. This time however we performed a faster sweep of the channels in order to check our previous calibration process. The results showed our previous calibration made in Cape Town was accurate enough and we could trust the spectra gathered by this system.

4.1.5 Discussion

Once installed, the initial prototype was a functioning data capture system which could store all captured spectra and allow users to view the latest spectrum captured with a web page. We set up the monitor to capture a spectrum once a second. This translates to 0.1246 MB/s or 315.4 GB per month of data. The databases could easily handle this data rate while serving the data to a web page. Unfortunately, with only 1TB of hard drive space we would only be able to store about 3 months of data, not including the RFI detection data and useful statistics which we planned on storing. This represented a serious risk for achieving our goal of storing a long term record of the RF environment. A solution which allows long term storage of data became the focus of the next prototype.

The other aspects of the first prototype performed well, the RATTY software was complete and did not require any changes. The web server and database handled multiple simultaneous queries while simultaneously inputting new data. However the data visualisation library from Highcharts cannot be used for free by SKA SA as it is not a non-profit organisation. Although it is legal to use for development purposes we had to come up with our own visualisation solution for the final RFI monitor.

4.2 Prototype 2: Archival Functionality

For the archival prototype we moved data from the text form saved in the databases into an HDF5 file. This allowed us to save the overhead of representing a binary python object as text. It also allows us to use HDF5's compression algorithms. However HDF5 does not support simultaneous reading and writing to the same file so there was no way to watch how the current spectrum changed as it was captured by RATTY. We created a new database called current_spectra to duplicate the last hour's worth of data. In this way we could access the latest data via the current_spectra database and we could access older data using the HDF5 files.

Each file contains an hour's worth of data, starting on the hour. We altered the spectra table so that instead of containing the individual spectrum the table points to the file which contains the spectrum. Each spectrum is placed in chronological order. This allows us to find a time within the file based on the location of the data in that file. For more information on the structure of the database see section 4.5.

The best option to increase the length of time that we could store data was to average the data over 1 hour periods and delete the original data to maintain enough space. The archive class takes in an hour's worth of data and calculates statistics which describe the data for each channel over that hour. Table 12 describes the statistics that we calculated for each channel and each hour.

A class called archive_rfi_spectra.py, contains methods to take an HDF5 file and calculate these statistics for all channels in that file each hour. We also added methods to the dbControl class which take the statistics and stores them in an archive database. We added a process to the rfi_monitor class which would create a new archive_rfi_spectra object after each hour to archive new data. We can see all of the processes controlled by the rfi_monitor class in figure 37.

To store more than 3 months of data, we always maintain at least 10% free space on the HDD. Whenever the HDD is more than 90% filled the data storage process will deletes hours of data stored in the HDF5 files and remove records on those files from the rfimonitor database. We still keep a long term record of the RF record in the archival database which has a resolution of 1 hour and a short term record of the RF at a resolution of 1 second. This extends the lifetime of the rfi_monitor to approximately 3 years assuming we maintain a minimum of two weeks of high resolution data.

Table 12 - Table of values in archival database

Statistic	Data type	Description
Mean	double	Mean power in channel over an hour
Min	double	Minimum value in channel over an hour
Max	double	Maximum value in channel over an hour
Standard Deviation	double	Standard deviation of channel over an hour
Median	double	The median value of channel over an hour
Percentiles (99,95, 90,80,70,60,40,30,20,10)	double	The value which is higher than n% of the values of channel over an hour, with n = 99,95,90, etc
Time Occupancy (3 sigma, 6 sigma)	double	The percentage of the time that values where 3 or 6 sigma above the standard deviation

Prototype 2 consists of a long term monitoring system which records the RF environmental changes over long periods of time and a short term record of the last couple of months/weeks in high resolution. We were still missing a vital component, that of automatic RFI detection. This was the focus of the 3rd prototype.

4.3 Prototype 3 : RFI Detection

For the third prototype we introduced the RFI detection. To provide RFI detection we needed to decide on a RFI detection algorithm. The algorithm had to be able to perform RFI detection on data at least as quickly as data was captured and to do so reliably. Many of the concepts discussed here have already been described in the RFI detection part of the background chapter.

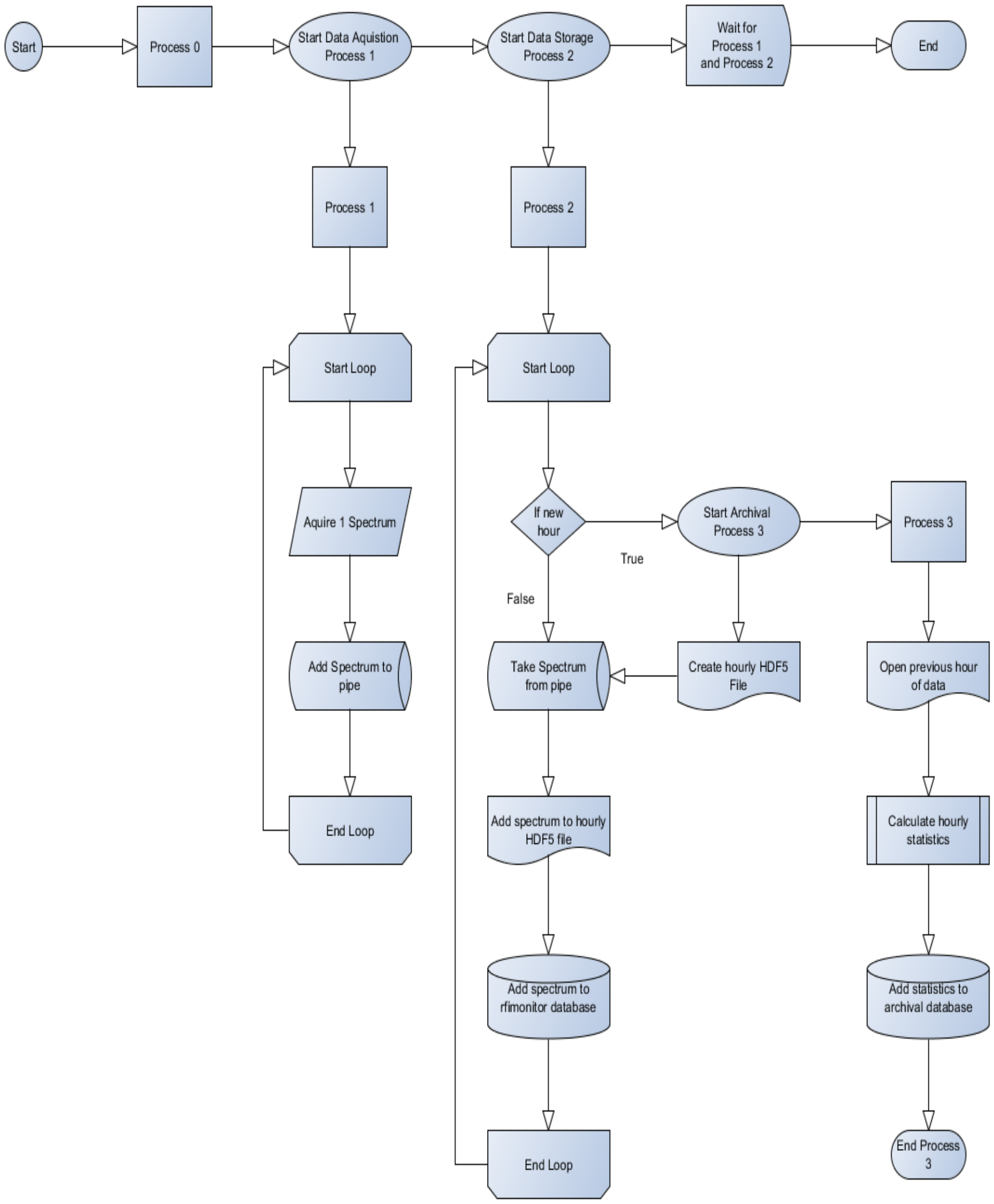

Start
Process 0
Start Data Aquistion Process 1
Start Data Storage Process 2
Wait for Process 1 and Process 2
End
Process 1
Process 2
Start Loop
Start Loop
Aquire 1 Spectrum
If new hour
True
False
Start Archival Process 3
Process 3
Add Spectrum to pipe
Take Spectrum from pipe
Create hourly HDF5 File
Open previous hour of data
End Loop
Add spectrum to hourly HDF5 file
Calculate hourly statistics
Add spectrum to rfimonitor database
Add statistics to archival database
End Loop
End Process 3

Figure 37 - Flowchart of the rfi_monitor class. The rfi_monitor main process, called process 0 starts 2 processes. Process 1 which acquires data from the RATTY and process 2 which adds that data to databases and creates hourly data files. Process 1 and 2 run continuously. Once an hour Process 2 creates a process 3 which calculates the hourly statistics of the last hour's data and stores the result in the archival database.

We decided to perform RFI detection using temporal thresholding. Since the thresholding algorithm is easy to implement, can easily handle the throughput of the RATTY system and has a proven efficiency. The algorithm has already been discussed in the background section 2.6. We decided to implement thresholding using the Medium Absolute Distance estimator suggested by the MeerKAT engineers.

We implemented this algorithm in Python, using Numpy to implement the algorithm. The implementation could perform RFI detection on an hour's worth of data in a couple of minutes. The RFI detection algorithm we implemented detect RFI on a 2 dimensional array of data and returns a 2 dimensional boolean array of the same size. The output array contains a false value at each point where no RFI was detected and a True value at each point where RFI was detected, this output array is called the RFI mask. This process is shown in the figure 38.

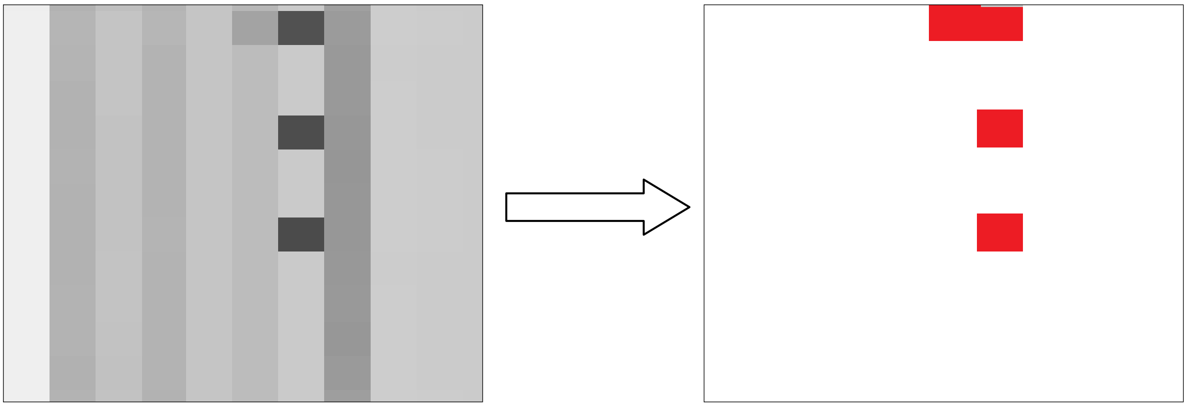


Figure 38 - Example of the RFI detection process. On the left is the raw spectral data for a few seconds. This data is fed through the RFI detection algorithm, represented by the arrow. The output is a mask. The red blocks indicate where the algorithm produces a True value in the RFI mask.

4.3.2 RFI event extraction

As described in the Archival section 2 of this chapter, we delete all of the high resolution data after a few months. This means we only keep data on short RFI bursts until the high resolution data containing that event is deleted. Any RFI which lasted for a short period, on the order of a few seconds, is averaged out and lost in the long term record. This means that we are not providing an adequate record of RFI events. To solve this problem we decided to extract and save all events detected by the algorithm in a third RFI_archive database. As RFI affects only a small (<5%) part of each spectrum, this allows us to store high resolution data on individual RFI events as well as a long term record of the RF environment while still allowing us to store around 2 years of data on 1TB.

The process of extracting RFI events is simple. First we group contiguous parts of the RFI mask to find RFI events which are most likely caused by 1 source. This was achieved by using a python image processing library, mahotas, which is implemented in c++ to avoid the efficiency issues with Python. The mahotas library contains a method which takes in a boolean matrix and labels each contiguous group as an integer. We select and extract a rectangle around each group, which represents an RFI event in the original data. These extracted events can then be saved in a RFI database. This process is shown in the Figure 39.

We created another database called the rfi_archive database to store raw data on all detected events as well as some metadata to facilitate querying the database. This metadata includes the start and end times of the event, the low and high channels of the detected events and the average power of the event. We also calculate hourly RFI related statistics to add to the archival database. For each channel and each hour we count the number of RFI events on that channel and the period of time that channel was affected by RFI. The flowchart in Figure 37 shows how the RFI detection thread interacts with the archival thread in the rfi_monitor process.

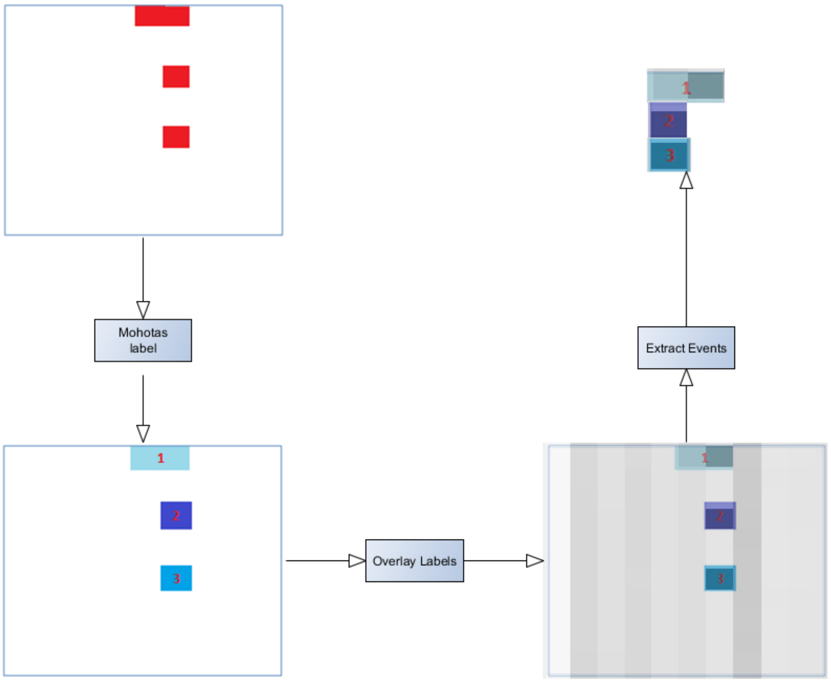


Figure 39 - Event Extraction process. We take the RFI mask created by the RFI detection algorithm and process it with the mohotas label function. This gives us a mask with all of the connected points in the RFI mask labeled as unique integers. We then use this information to extract RFI events from the original data. To see how this fits in with the flowchart in figure 37, see figure 40.

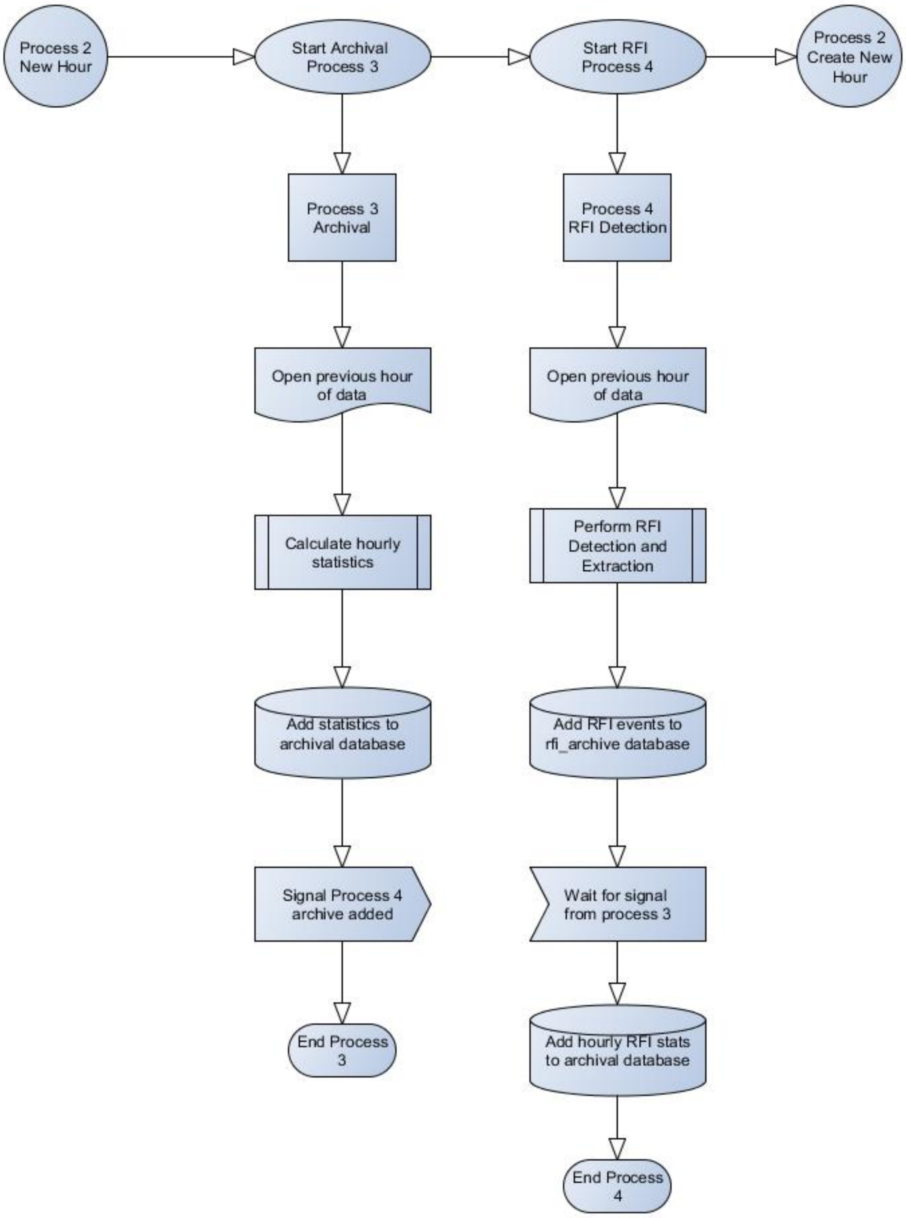

Process 2 New Hour
Start Archival Process 3
Start RFI Process 4
Process 2 Create New Hour
Process 3 Archival
Process 4 RFI Detection
Open previous hour of data
Open previous hour of data
Calculate hourly statistics
Perform RFI Detection and Extraction
Add statistics to archival database
Add RFI events to rfi_archive database
Signal Process 4 archive added
Wait for signal from process 3
End Process 3
Add hourly RFI stats to archival database
End Process 4

Figure 40 - How the RFI detection thread interacts with the archival thread. The chart starts at the point in Figure 37 where process 2 branches into a new hour. Process 2 then creates the archival and RFI processes and then continues as it did in Figure 37 by creating the next hours file. There is only 1 extra step in the archival process. Once the hourly statistics have been added to the archive, the RFI thread is signalled that there is an archive entry into which it can add the hourly RFI statistics such as number of events. The process in figure 39 is encapsulated in the block "Perform RFI Detection and Excision".

4.4 Prototype 4 : Web Interface

Prototype 3 can access data from the RATTY system, scan that data for potential RFI and archive that data. In prototype 4, we allow users to access this data. In this section we describe how we made a functional RFI monitor website from the initial prototype, to enable researchers to access and interact with RFI data generated at the MeerKAT site.

4.3.1 Visualizations

Visualization of the data is a key function of the RFI web interface. The data comprises spectral data, which is essentially continuous data in time and frequency, hourly statistics and RFI event data which consists of spectra on the events, RFI event counts for channels and occupancy statistics for each channel.

Users need to be able to view the data in a variety of ways, including 1 second spectrums or 1 channel over multiples timestamps, a range of times and channels at once and which channels have the most powerful RFI or the least amount of time corrupted by RFI. These are queries that could easily be satisfied by graphs and charts. In the final prototype, we implemented three visualizations to assist researchers in exploring the data: a line chart, a waterfall plot and a bar chart. The implementation and function of each of these views is described below.

4.3.1.1 Line Chart

We required a line chart which could display up to 14200 points for the visualisation as this is the number of points in each RATTY spectrum. As there are no free libraries to do this, we implemented our own system using the processing.js visualisation library. The Processing.js library is a JavaScript library which can convert a Processing visualization written in Java into a JavaScript canvas.

An example of the line chart is shown in Figure 41. The chart can handle all of the 16384 points needed to display one spectrum. The point closest to the cursor has a pop-up, which displays the value of the point. The chart can be zoomed by using the cursor wheel and scrolled by clicking and dragging; allowing the user to explore the data while zoomed in. The user can also save their view as a PNG image at any time. The line chart can display a single spectrum or a single channel over many hours.

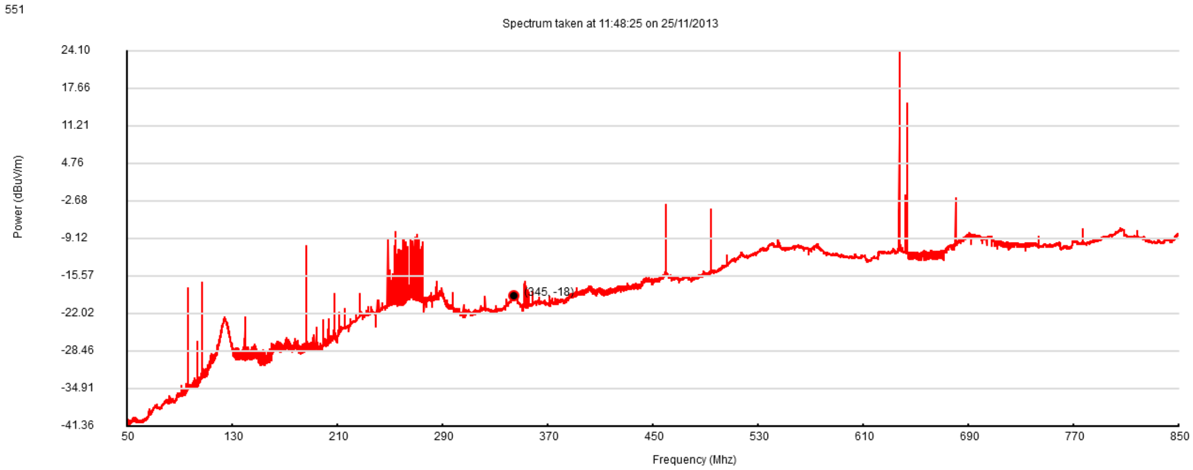


Figure 41 - Example of line chart. One spectrum is displayed here.

The line charts can plot the latest spectrum as it is collected, giving users a real-time view of the RF environment. The user can pause the view of the spectrum and also move back and forward in time through the data while maintaining their zoom. They can see how the whole spectrum or a subset of the spectrum is changing over time, either in real-time or after the data has been captured. An example of zooming is provided in Figure 42 a,b and c. The line chart can also show the data on a single channel over a specified period as shown in Figure 42 d. The user can also choose to display the hourly statistics using the line chart.

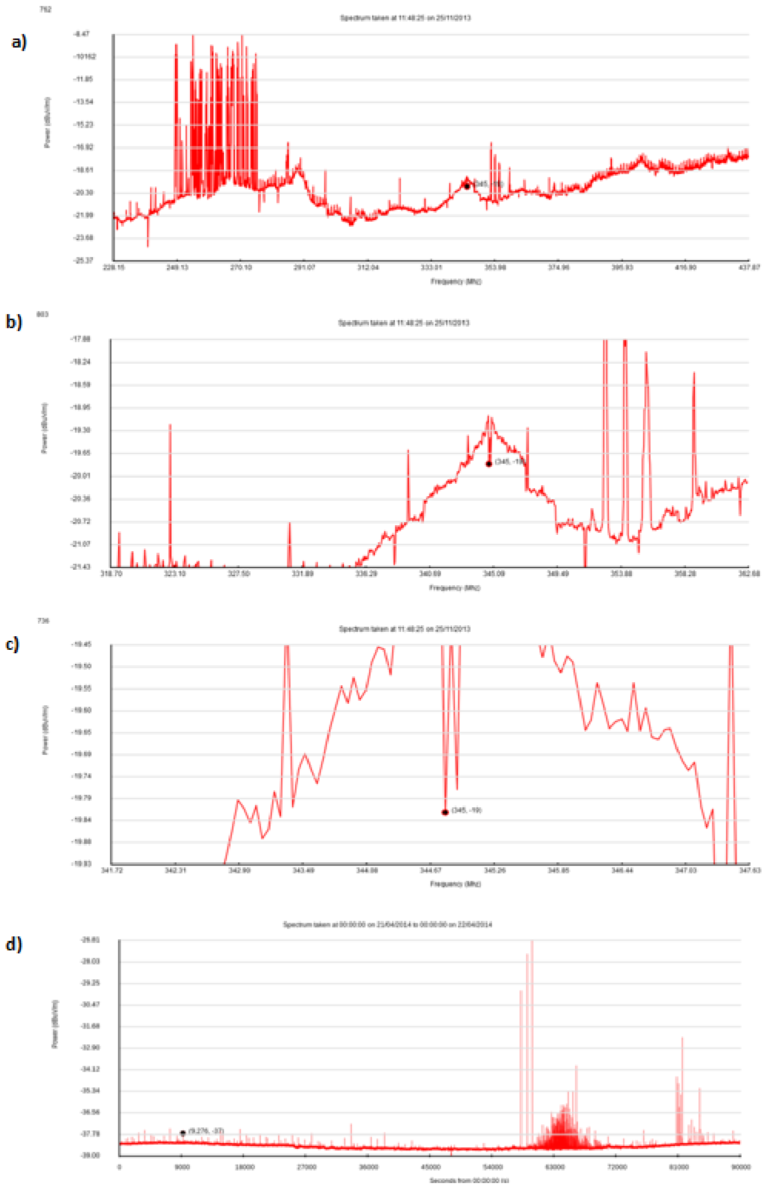


Figure 42 – a,b and c demonstrate the zoom function. Here we show progressive levels of zoom onto the black circle shown in a. As you can see it is simple to switch between a high level view and a very close view to see small scale details. d) Example plot of the channel 70.4MHz over 24 hours (86400 points). 70.4MHz is the frequency used by 2 way radio's on site.

In order to provide a continuous flow of data we used the JavaScript/Python library socketIO which allows real-time communication between a web application and a server. The web browser uses a socketIO.js file to provide client side connections. In our client side we create a socket which we can use to access data sent from the server. The socketIO library determines the fastest connection available to perform data transfer. The server uses the Python socketIO library for real time data access. The server creates a new greenlet to serve the data to each client. A greenlet is a lightweight thread in Python to allow multiple clients to be served simultaneously from one python server, without the overhead of the processing library.

We also created two databases to allow for the current hour of data to be accessible by our visualizations. The HDF5 files cannot be simultaneously written to and read from, so we keep a copy of the latest data which can be both written to and read from; a database called current_spectra. This database is very simple; the structure is shown in Table 13. It contains a timestamp and a serialized Numpy array containing the spectrum at that timestamp. Each hour, the last hour of data stored in the database is deleted. In this way, the database will only ever contain data from the beginning of the current hour.

Table 13 - Data contained in the current_spectra database

Value	Data type	Description
Timestamp	uint(10)	Unix timestamp is seconds
Spectrum	binary	Serialised Numpy array containing the spectrum

Unfortunately deleting an hour’s worth of data can take up to 10 minutes when the server is running all of the components of the RFI monitor. While this data is being deleted, no new data can be added to the current_spectra table. Therefore we created two current_spectra databases so that when we need to delete data from the last hour in the first database we simply start adding data to the second database. To co-ordinate which database is currently in use we had to add a new table containing a bit value current. When the value of current is set to 1 then we know that current_spectra database is the one which contains the latest data and the other current_spectra database is being deleted from. This allows us to seamlessly transition between hours.

4.3.1.2 Waterfall plot

The web interface also provides a waterfall visualization of the data. Waterfall plots are standard visualizations for representing three dimensional data in two dimensions. The plot shows time on the y axis, frequency on the x axis and the power of the particular channel at a point in time is displayed by the intensity of the colour of that point. A waterfall plot can show multiple channels and times in one plot. This grid was also created using the Processing and Processing.js libraries.

Just ten minutes of data for 14200 channels is over eight million 64 bit numbers. Unfortunately, it is physically impossible to display these points on a typical screen of resolution 1024 by 768 or approximately 1 million pixels. The average computer and web browser can only practically display around forty thousand points. Therefore an interactive waterfall plot using JavaScript we would need to reduce the number of points used to display data to a manageable amount.

The most straight forward way to reduce the number of points is to simply divide all of the points into a grid of the amount of points we would like to display and combine the points in each grid cell. In Figure 43 you can see a small scale example of what we needed to achieve. In this case we take a 200 by 200 grid of points, aggregate those points into a grid of 8 by 8 points and then attempt to represent the data in those 40 000 points using just 16 points.

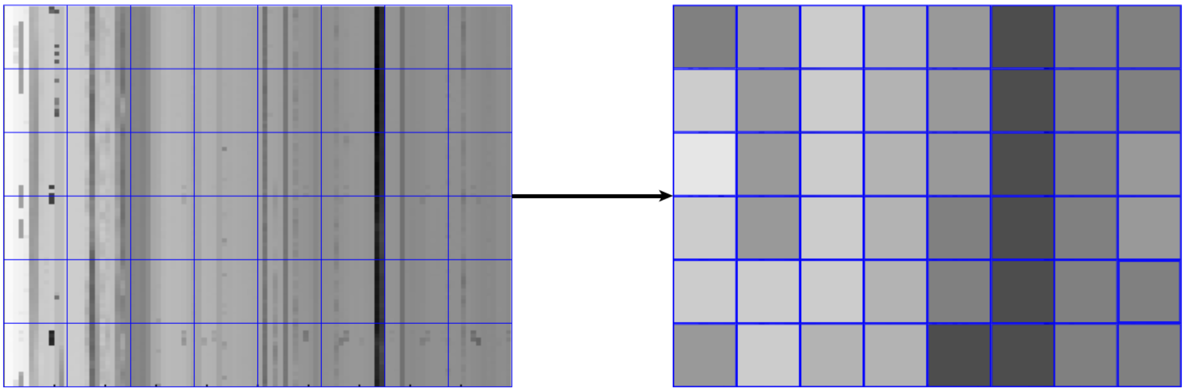


Figure 43 - Example of the gridding and data aggregation process. In this example we are reducing 40 000 points to 16 points. In the actual web interface we would be performing data reductions on the order of 10s of millions of points to 10s of thousands of points.

To ensure that the important features of the data are preserved, we decided to aggregate data in the plot by choosing the maximum value from each grid block. Looking at these plots it is easy to spot the most powerful signals, which are indicative of a RFI event. The aggregation reduces the accuracy to which the RFI event can be located. It was important to allow zooming in to the data so that RFI events can be shown with the highest accuracy.

To solve this problem we decided to implement a Google Maps style zoom function. This allows the user to zoom into aggregate data and sends higher resolution data when the zoom goes past a certain limit, we then send this higher resolution data for the part of the data which the user is zoomed into. The server sends new data whenever a user zooms in to a level where they are displaying a quarter of the points that were previously sent. This means that the data which is represented by for example a block of 100 by 100 points will be resent however they will be represented by 200 by 200 points. This way we only send higher resolution data which the user actually zooms into and each time we only send 0.3KB of data. This process is shown in the Figure 44.

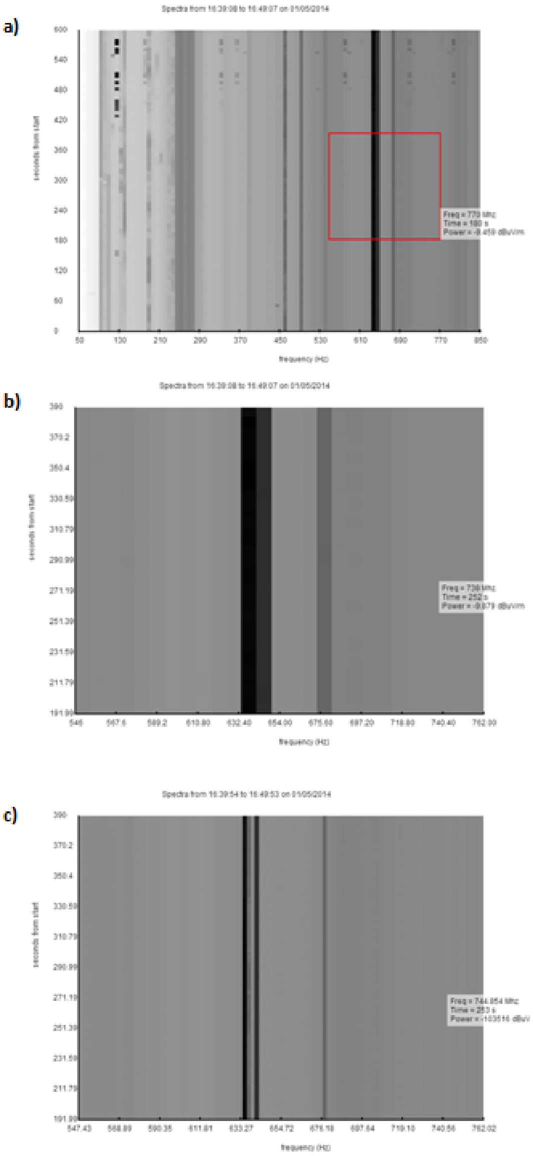


Figure 44 - Example of the zooming process: a) is a fully zoomed out view, the red box shows the area being zoomed to, b) shows the zoomed view before data is re-aggregated and c) shows the zoomed view after data is re-aggregated.

Figures 44 b and c show how the level of detail changes as the server re-aggregates and transmits data on zooming. The user can see a high level overview of the data. They can then zoom in on points until they reach the level of individual samples, allowing them to pinpoint RFI to the second and channel which it affected. The process is reversed as the user zooms out. Due to the fact that only the spectra are stored in HDF5 files, this waterfall plot can only display the spectra and none of the other calculated statistics. Although it is not impossible to provide access to the other data, it is not being implemented for this thesis.

The waterfall chart can also be displayed with a red RFI mask overlay. From Figure 45 we see the mask shows the results of RFI detection at a glance. The RFI mask can be toggled on and off, so the user can see the underlying signal if they choose to. The RFI mask is aggregated in the same way as the underlying data. If any data in the aggregated block contains RFI, that block will be red in the RFI mask.

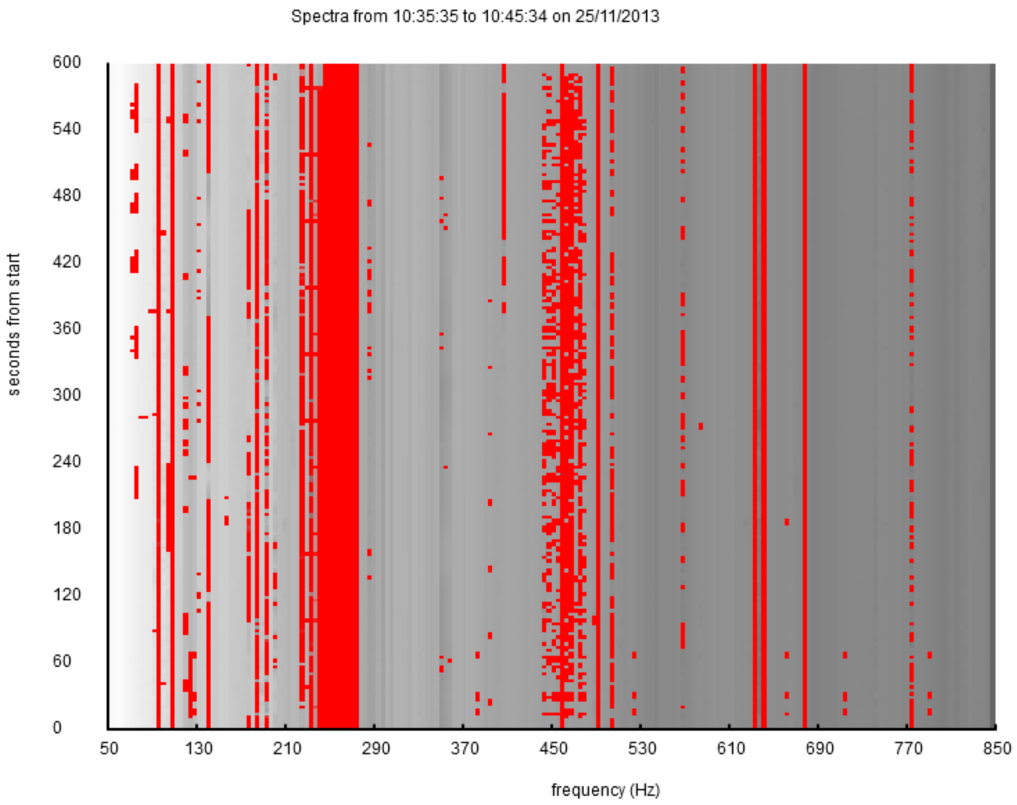


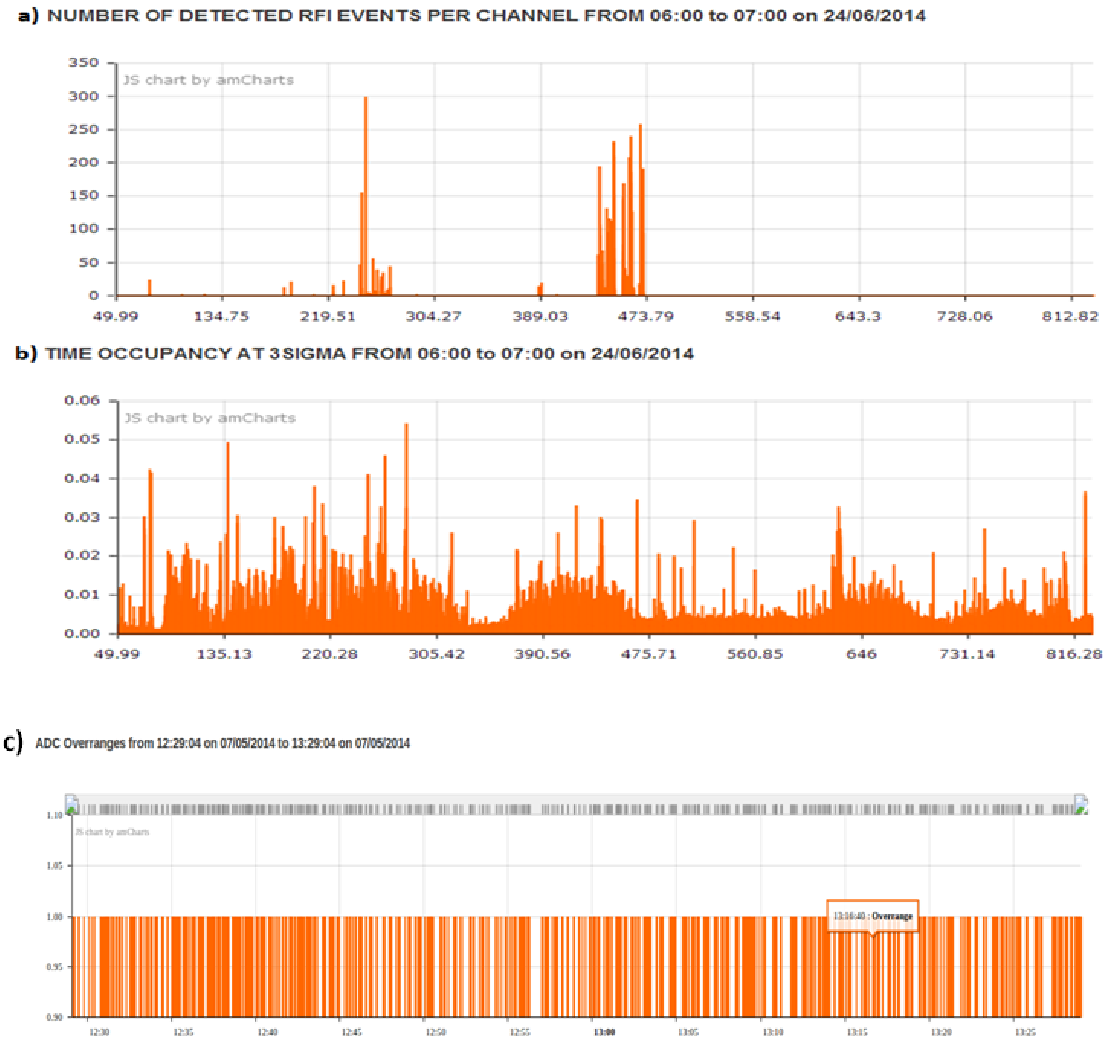
Figure 45 - Example of the RFI mask overlay for the waterfall plot.

4.3.1.1 Bar Chart

The third visualization that we provide is a bar chart created with the JavaScript library amCharts[1]. Figure 46 shows how the bar chart can displays hourly RFI event counts, channel occupancy and ADC over ranges. Figures 46 a) and b) show which channels have been worst affected by RFI according to our RFI detection algorithm. Figure 46 c) shows the times in an hour when the ADC over-ranged. If there was an ADC over range while a spectrum was collected the spectrum is untrustworthy. ADC over-ranges occur when there was a powerful signal sampling for the spectrum. The figure shows a

[1] www.amcharts.com

particularly bad example of ADC over ranging: there were many strong signals generated at the time of capture.



a) NUMBER OF DETECTED RFI EVENTS PER CHANNEL FROM 06:00 to 07:00 on 24/06/2014

b) TIME OCCUPANCY AT 3 SIGMA FROM 06:00 to 07:00 on 24/06/2014

c) ADC Overranges from 12:29:04 on 07/05/2014 to 13:29:04 on 07/05/2014

Figure 46 - Bar chart showing a) the number of RFI event s detected in each channel for each hour and b) Time occupancy at 3 sigma. c) An hour's worth of over range data. Each orange bar represents an ADC over range for a 1 second spectrum

4.5 Database Structures

The database structure is for the initial prototype is shown in Figure 47. The current_spectra and rfi_monitor databases respectively are essentially one table databases with a corresponding system table containing the information about the state of the RFI monitor at initialisation.

For the final rfi_monitor database the spectral data is stored in hourly HDF5 files. Each entry in the spectra table contains a path to the file which contains the spectrum. Although the spectral data is not stored in the database, we can still query the database to find extrema we calculate for each channel every hour. This allows us to quickly find the file which contains the maximum value for a particular channel as each extrema value also links to a hourly spectra file containing that spectrum. This can be found by using the MYSQL database.

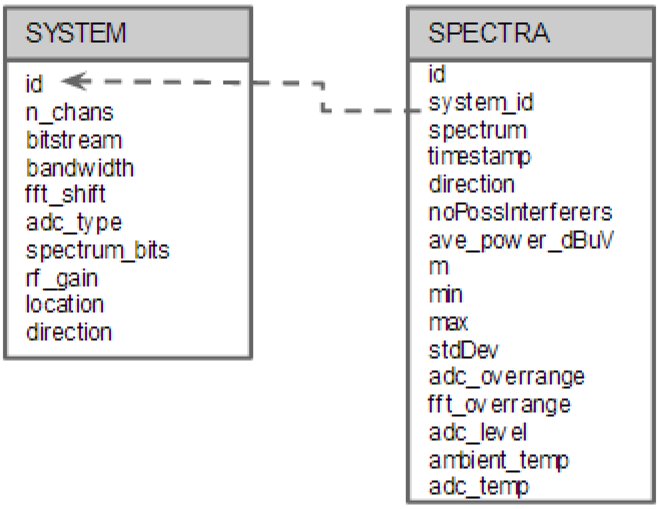


Figure 47 - Design of the rfi_monitor and current_spectra databases. System contains data which is constant for each instance of the monitor and spectra contains data which is collected each second

The current_spectra database has the same structure as the rfi_monitor, however spectra are stored as binary strings in the database, not in files. There are two identical tables to store spectra, one of which holds current data while the other is emptied. Each hour the roles of the two tables are swapped. In this case we store the actual spectra in the database as a serialized Numpy array stored in a string format. We can access the latest data here as hdf5 files cannot be simultaneously written and read to.

The archive database is broken into five tables, the structure is shown in Figure 48. It has a system table which is identical to those in the rfi_monitor database. The data is then broken into four tables. We have an element table which corresponds to a frequency and channel number and contains an id for each channel. A spectra table containing the first timestamp of an hours worth of data and a confidence value, the percentage of that hour during which the monitor was collecting valid data. If a spectrum is not valid because it was not captured or there was an overrange we want to quantify how this affects the confidence of our hourly statistics. This confidence value ranges from 0 to 1. Each spectrum also contains a unique id.

The value table contains all of the hourly statistics collected. Each value has a three element id made from the channel id, the timestamp of the hour it was collected in and the type of data value. We can pinpoint the hour and channel/frequency which any particular value comes from. Having a datum table allows the user to break the spectra table into separate values before searching for the times they are interested in. New types of hourly statistic can be added by simply adding an extra row to the datum table.

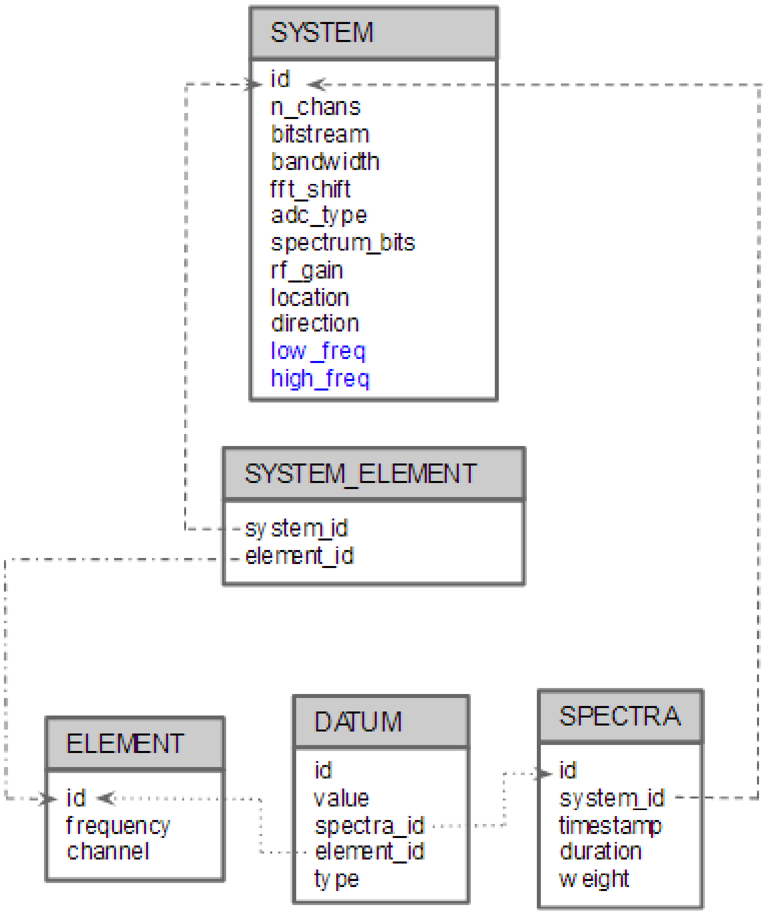


Figure 48 - Design of the Archive database

The final database is the rfi_event_archive. This database has an almost identical structure to the rfi_monitor. Except we have an rfi_event table rather than a spectra table. The rfi_event table contains the raw data which we extract in the RFI detection process and some metadata describing time and frequency that the particular RFI event was detected, this is shown in Table 14. The metadata places each RFI event in the context of the archived data. As RFI is detected on a 6 sigma threshold, this table contains all points where the power on a channel differed by more than 6 sigma from the median.

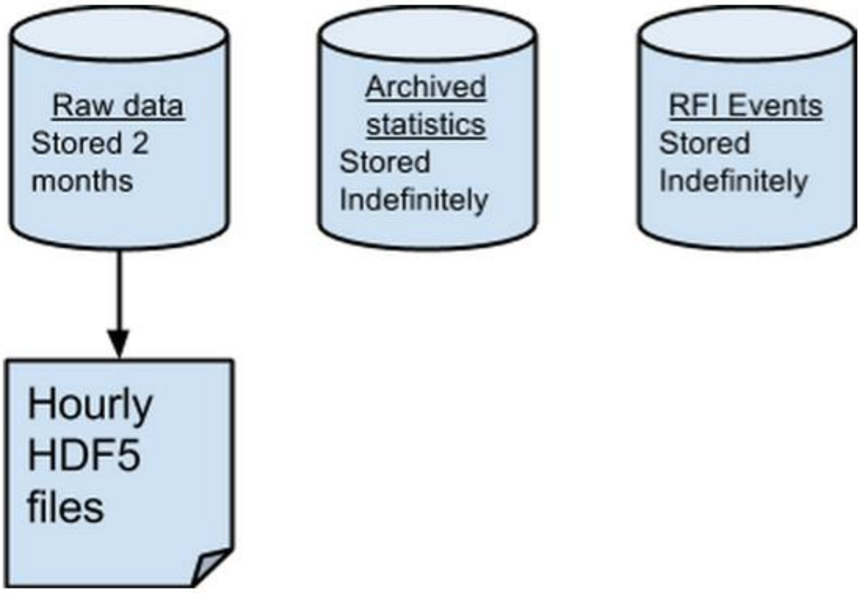


Figure 49 - Diagram showing the 3 databases, the data they contain and the amount of time that data is stored. The link from the raw data to the hourly HDF5 files represents the fact that the database simply stores the location of the HDF5 files which contain the raw data.

Together these databases provide the functionality to store a record of the RF environment at a resolution of one second for a minimum of two months and the ability to serve the latest collected data as it is captured. They allow long term storage of hourly aggregated data and raw data on detected RFI events for periods of up to two years. Together they allow us to easily access good data on the RFI around the MeerKAT antenna for long periods using a 1TB harddrive.

Table 14 - rfi_event table

Datum	Description
id	Id of the event
system_id	Id of the system used to capture data (corresponds to system table)
startTime	The timestamp at the first detection of this event
endTime	The timestamp at the last detection of this event
low_chan	The lowest channel this event was detected on
high_chan	The highest channel this event was detected on
spectra	The raw data that was captured between the times and channels above
mask	A mask which shows which off the values in the spectra were flagged as RFI

4.6 Conclusions

Each of the prototypes helps fulfil at least one of the goals of the RFI monitor: providing a continuous record of the Radio Frequency (RF) environment, automatic RFI detection and effective access. Table 15 shows how each prototype progressively improved upon previous prototypes until all of the requirements were fulfilled. The requirements were designed to achieve our goals. By prototype 4 all of our goals had been achieved.

Table 15 – Table goals from section 3.1 and the requirements from section 3.3.4 that each prototype fulfils. Requirements with an asterisk were completely fulfilled at the by that prototype.

Prototype	Requirements fulfilled	Goals
Prototype 1 : Basic Operations	RF record Access to raw data Data visualisations	Continuous RF Record Effective Access Effective Access
Prototype 2 : Archival Functionality	RF record Descriptive statistics*	Continuous RF Record Continuous RF Record
Prototype 3 : RFI detection	Automatic RFI detection* RF record* RFI record*	Automatic RFI detection* Continuous RF Record Continuous RF Record*
Prototype 4 : Web Interface	Data visualisations* Access to raw data*	Effective Access* Effective Access

5 Validation and Testing

In this Chapter we will discuss the effectiveness of the components of the RFI monitor. Proving the monitor is reliable is vital if it is to become a useful tool for RFI mitigation. After implementing the last prototype, the system was allowed to run for six months without any human interaction except over the web page. This shows that the system is stable over long periods. The rest of this chapter shows the RFI monitor is also accurate, performs RFI detection efficiently, provides useful visualisations and is useful for real RFI campaigns on site

5.1 Data Calibration

The first and most important step of calibration was to ensure that the data collected by the RATTY system are accurate. This was achieved by calibrating the signal chain during RFI monitor installation on site. Calibration involved sweeping a known signal with a known power across the bandwidth of the receiver. We compared the measured result with the known input signal and created a gain calibration for each channel in dBs. This is a floating point number which we add to the value of each channel to ensure that the measured signal is equal to the expected signal. We also ensured the archived data is accurate by independently calculating the statistics for twenty hourly data files in excel and checked that the archived statistics were equal.

5.2 Database Access

There are two ways we tested the database: we showed data could be processed and inserted in a timely manner and data could be accessed quickly. The first metric shows that our RFI monitor can handle the incoming data rate. The second metric shows that our database is structured well enough to make data accessible without long delays.

5.2.1 Inserting

We ensured each spectrum can be captured and stored in less than one second and that hourly statistics can be calculated in less than 1 hour. We ran the monitor for an hour and found it took an average of 0.3s to insert one spectrum. This leaves time for a higher data rate if needed for later upgrades. We tested hourly statistics and RFI detection archiving by running the monitor for one week and averaging the time of each step of data collection, calculation and insertion. The results showed our monitor can perform all this data capture and archival in under an hour as necessary.

Table 16 shows the times necessary for each of the archiving processes. Some of the processes are performed simultaneously. In the end we can expect all of our hourly data to have been calculated and stored by an average of 33 minutes after a capture period ends. The time taken to store data is dominated by the hourly percentile calculation as it involves sorting each channel.

Table 16 - The timing results for data retrieval and storage.

Data process	Time used
Single spectrum archival	0.3 s
Hourly statistic calculation	28 min
Hourly statistic archival	5 min
Hourly RFI detection	2 min
Hourly RFI event extraction and archival	5 min

5.2.2 Retrieval

Data access is important for usability. Although data exists, they are not useful if the retrieval times are long. The data are presented to the user via the web interface with multiple views. In the graphs below we show how long data collection takes for each of these views.

Table 17 shows results for views which should take a constant time to retrieve. These views always show the same amount of data. These results show the time from accessing a view on the web browser to when the data are visualised. The views were opened 20 times and the mean retrieval time is shown in the table below. The retrieval times for these views were fast enough that most users will not notice them.

Table 17 - The Retrieval time for spectra and archival data.

View	Retrieval time
Current Spectra	0.007s
Hour Archive	0.171s

The next measurements were tests which show how views with scalable data ranges performed. These include the graph channel view, the over-range view, the waterfall plot and the hourly statistic plot. We used the same time ranges for each tests. The first test was for an hour's worth of which is an example of a small but meaningful amount of data, the next test is 10 hours worth of data and finally we tested retrieval of 100 hours worth of data. Unfortunately the waterfall plot cannot display 100 hours worth of data as it is too large. For the hourly statistics we use retrieval for 10, 100 and 1000 hours as each hour represents 1 value.

Table 18 – The scaling of data access for different data. This data is plotted in Figure 49.

Num Hours	Over-ranges	Channel	Waterfall	Stats
1	0.008 s	2.47 s	11.48 s	
10	0.094 s	1 min 21 s	6 min 52 s	8.45 s
100	3.129 s	17 min 51 s		1 min 32 s
1000				20 min 48s

Figure 49 shows that the retrieval times for the data scales approximately linearly with the number of hours of data requested. There is a limit to which the data can be retrieved without causing significant delay. This limits the ranges of data that a user can choose to view at a time. However the retrieval times are still usable in the data ranges for which the monitor is generally used (1 hour to a day).

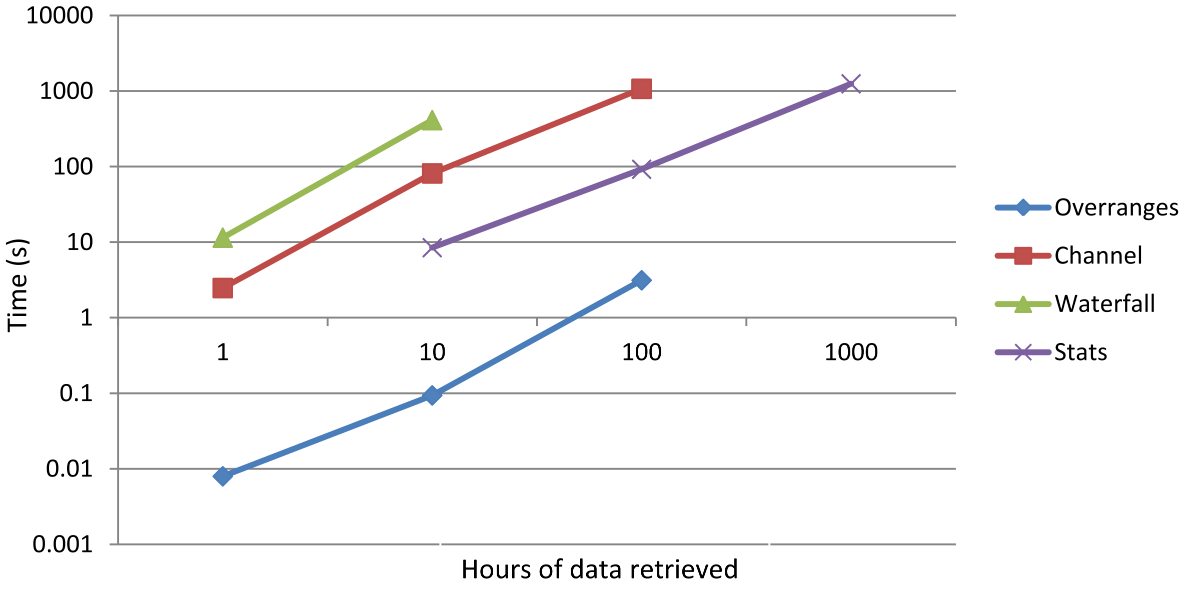


Figure 50 - Chart showing the scalability of different views on the data, both axes are logarithmic.

One issue which is worrying is the amount of time it takes to retrieve the hourly statistics. In this case the number of values is far lower than the other 3 cases and the retrieval times are disappointing. In the 10 hour case retrieval takes approximately 8 seconds. We should be able to retrieve this data far faster. The main reason for this disappointing speed is that the hourly statistics are stored in a large database with approximately 2 years worth of data. As the database grows, the seek time dominates the retrieval time.

This problem could be solved by storing the hourly statistics in data files in the same way that the raw data is stored. This should increase the retrieval time for the hourly statistics to the same speed as retrieval for spectra. This is the recommended way to store hourly statistics in the next iteration of the monitor.

5.3 RFI Detection

One of the goals of the RFI monitor is storing a long term record of RFI on site. As discussed previously, we perform this task without storing all of the raw data. We only store raw data which contains suspected RFI in the long term. The goal of a RFI record can only be achieved if the RFI detection algorithm is accurate.

We tested the RFI algorithm by creating simulated data with simulated RFI. The simulated data has a normal distribution as one expects for a RFI clean environment. We added broadband and narrowband noise of different power levels. RFI in a power level have a total power equal to the level number times the standard deviation of the simulated data. We ran the detection algorithm on this data and recorded the proportion of RFI that was correctly identified. We ran this test for different window sizes and threshold factors. The window length is the length of the window on either side of a data point used to estimate sigma using the MAD algorithm. The threshold factor is the multiple of the estimated sigma that is used to threshold a data point. This allows us to choose the best values for use on our monitor.

False Positives VS Window Length and Threshold Factor

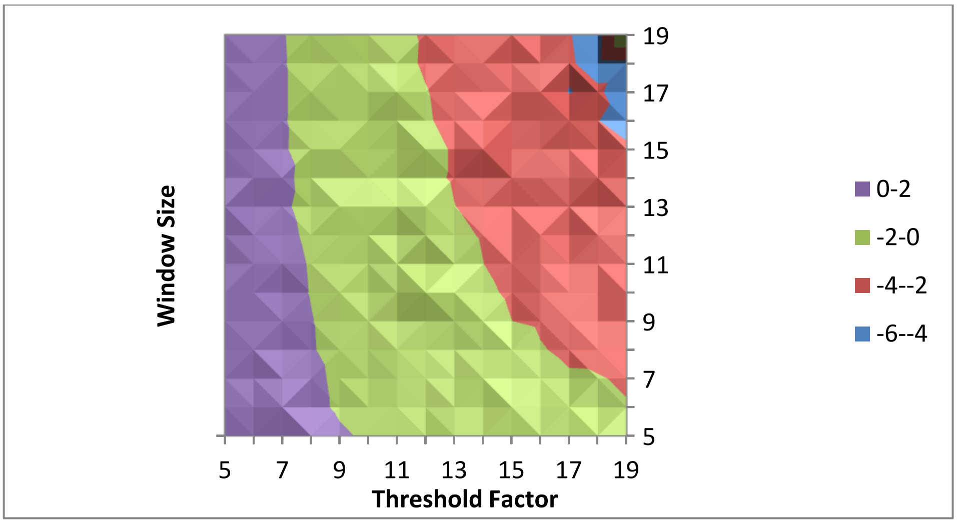


Figure 51 - Heat map of false positives divided by total number of true RFI values VS window length and thresholds factor. Values are on a log scale. Lower values are better. A value of 0 means that there were as many false negatives as there were values corrupted by RFI.

Figures 50 and 51 show that, in general, the better the algorithm does at detecting RFI, the more likely it is to falsely classify clean data as RFI. Figure 48 shows the main factor in accurately classifying true RFI is the threshold value. Using a threshold of up to 13 times sigma will mean you are capturing most of the RFI. However from Figure 50, any threshold below 8 with most window lengths will create at least as many false positives and actual RFI contaminated values. These charts show the best way to maximise the true positive to false positive ratio is to choose a low threshold which is between 9 and 13 with a window of at least 11.

Percentage of True Positives VS Window Length and Threshold Factor

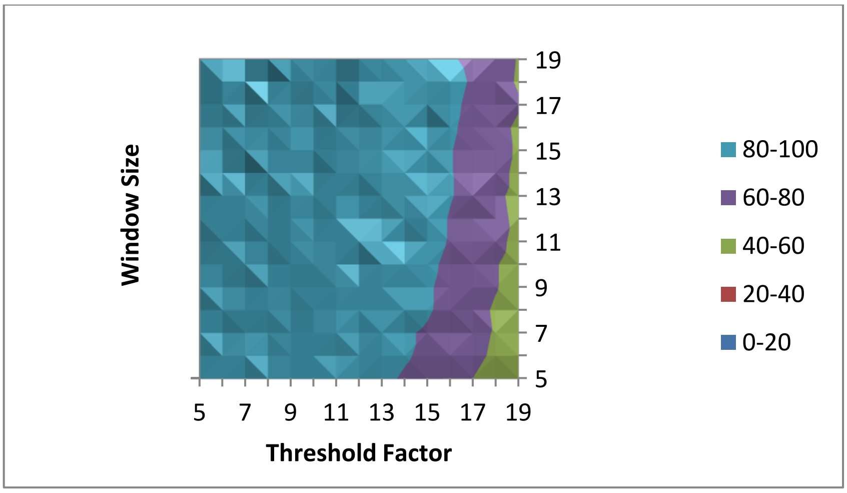


Figure 52 - Percentage of true positives VS window sizes and thresholds. Higher values are better.

Percentage of True Negatives VS Window Length and Threshold Factor

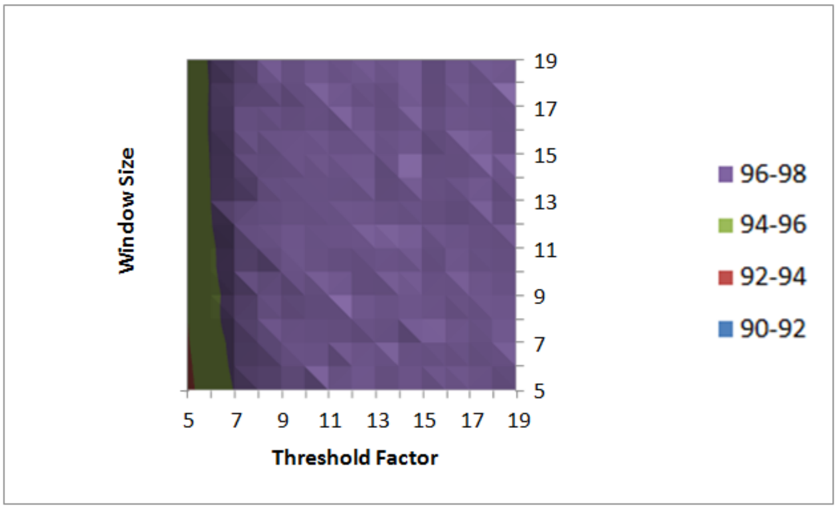


Figure 53 - Percentage of RFI clean values which are classified as clear of RFI. Higher values are better.

Figure 52 shows that the amount of false positives is low for most configurations above a threshold of seven sigma. Any threshold larger than nine sigma is an adequate choice to prevent too many false positives. This doesn't affect the earlier finding that the best range of values is a threshold between 9 and 13 and a window of at least 11.

As we only archive raw power spectrum data which is contaminated with RFI, it is better to have false positives than lose data because of false negatives. We decided to use a threshold of 6 sigma to ensure that we can accurately recreate the environment with archived data. To test how well our archival system could recreate the RF environment after raw data was deleted we compared the one week of raw data with its corresponding archived data. We calculated the mean and standard deviation of each channel in the hour from the raw data. We then used a 6 sigma threshold to create a RFI mask of the raw data. We then calculated the percentage of the raw data that differed by more than 6 standard deviations from the archived data. This test showed that only 0.04% of the archived data was not within 6 standard deviations. This proves our archival process can effectively reproduce the RF environment while discarding most of the raw data by storing only data that are likely RFI.

5.4 Visualisation and Web Page

The goal of the web page and visualisations was to provide easy access to the data. It is difficult to test ease of use quantitatively so we tested our visualisations with a user survey. The survey was completed by 15 users who represent astronomers, engineers and RFI management. All of the users tested are employees at SKA-SA or academics with SKA funding. As the users are experts, their opinion is valuable.

Our survey tested the ease of data access as rated by the user and timed how long users took to perform common tasks. Users were asked to rate their experience on a 5 point scale. We asked users to explore the web page and provide comments how useful they found the RFI monitor. We tested the usability of the line chart and waterfall plot as these were the visualisations created as part of this thesis. We also tested the usability of the web page itself.

The first 6 Questions established who the users were. We asked users to provide information on their position, their level of education, internet connection, etc. These questions were voluntary so users could remain anonymous, however all respondents chose to provide these details.

Of the survey respondents twelve were employees of SKA-SA and three were post-doctoral or masters students on SKA grants. Of the respondents four were astronomers, four were post-doctoral researchers/students and seven were engineers. Two of the engineers were involved in RFI management. Of the users two thirds used Chrome and 1 third used a Firefox. All users were on the Internal SKA-SA network, so their network connection was 1 Gbps.

5.4.1 Line Chart

This section includes the results for the Current Spectrum and Graph Channel views. Both views used the Line Chart visualisation. The Current Spectrum displays a constantly updating spectrum and the Graph Channel view was displays a data from one channel over a chosen time range.

5.4.1.1 Current Spectrum

Users were asked to open the Current Spectrum page from the home page. After checking the visualisation was updating, users paused the updates and used the line chart to discover the value of a two channels captured at that time. The first channel contains an obvious spike and the second channel is in the noise and would require zooming. They saved a picture of their view which were used to independently verify the results they got. Table 19 shows the time taken for each of these tasks and the amount of users who successfully completed it.

Table 19 - Times it took for users to complete the example tasks as well as the percentage of users who correctly completed the task.

Task	Mean Time	Accuracy
Find power level of channel with obvious spike	12s	100%
Find power level of channel in noise	27s	100%

Table 19 shows that users found it easier to find the values of channels with a spike than those buried in the noise. This is a positive result as channels with spikes are most likely to contain RFI.. The users managed the more complicated task of finding a data point buried in the noise in the reasonable time of 27. This seems to show that the visualisation and zoom function is easy to use. This is backed up by the user responses in table 20.

Table 20 - User responses to the usability questions. The answers to each question were ranked on a 5 point scale. Strongly disagree was given the value 1, no feelings is a 3 and strongly agree given the value 5.

Question	Mean
I found the link the Current Spectrum easily on the home page.	4.3
The instructions on the page were helpful.	4.3
The visualisation was easy to use.	4.1
The visualisation was responsive.	4.2
The visualisation was clear and easy to understand.	4.3

The users were also asked to provide comments on their experience. Below are some responses to these open ended comments.

Please comment on any part of the visualisation you found hard to use.

There was one comment in this section which came up in most of the responses. Below is a typical comment.

“Zooming in and out was a little strange in that it would zoom to the centre of the displayed spectrum, rather than to the location of the cursor on the figure”

This explains why it took almost 3 times longer for users to find the value hidden in the noise. The zoom zooms into the centre of the view, requiring panning to find data points. Allowing zooming onto the cursor cursor should be implemented for the next iteration of the line chart.

Please comment on how you would improve this visualisation.

Some of the improvements that were common in many comments:

1. Zoom into location of cursor.
2. Provide a reset zoom button.
3. Provide a save image button (in addition to the keyboard shortcut).
4. Larger text in general.

From the results above we can see that although the visualisation provides the basic functionality for users to perform their tasks and they felt positive about the usability of the visualisation, however there are areas of possible improvement.

5.4.1.2 Graph Channel

Users were asked to open the Graph Channel view. They were then told to select 2 hours of data to display between 8am and 10am on the 11th of September on the channel with frequency 150MHz. They were asked to find the values at specific times in this interval in much the same way as the previous Current Spectra test above. As the visualisation is the same in both cases, we wanted to test the usability of the forms we use to select data and also the ability to download data. This also allowed us if users got faster after their first attempt to use the visualization.

Table 21 - Average time for users to complete tasks along with accuracy

Task	Mean Time	Accuracy
Select a date range and frequency	9s	100%
Find power at time with obvious spike	8s	100%
Find power at time buried in noise	20s	100%

Table 21 shows users found it easier to find spikes than data buried in the noise. It seems that the users have gotten faster in this second set of tasks than they were at the first task. The usability responses in Table 22 back up the results from the previous test.

Table 22 - The user ratings for different tasks that they had to perform. They show that in general users found the task easy.

Q17. I found the link the Graph Channel easily.	4.3
Q18. The instructions on the page were helpful.	4.1
Q19. The visualisation was clear and easy to understand.	4.1
Q20. Choosing a time range for data was simple.	4.5
Q21. Choosing a Frequency for data was simple.	4.3
Q22. Downloading data was simple.	4.3

5.4.2 Waterfall Plot

The waterfall plot allows the user to see a range of times and channels along with the RFI which was detected on each of those channels. For this task users were asked to view 10 minutes of data for all channels. Users were asked to identify the frequencies affected worst by. Users then had to zoom into a portion of the data and to analyse the amount of RFI events and which channel was worst affected by RFI over that portion. Table 23 shows the results of the survey.

Table 23 - The times it took for users to complete their tasks and the accuracy of their results

Task	Mean time	Accuracy
Which Frequency range is worst affected (over all data)	8s	100%
How many RFI events occurred in this time (includes zooming)	43s	93%
Which frequency had the most RFI events	4s	93%

Table 23 shows users could identify which channels were badly affected by RFI quickly and accurately. One user who could not perform the tasks after zooming as the visualisation failed to load the zoomed data. The times show it is easy to visually identify RFI with the waterfall plot, however the zooming takes more time than the visual analysis. This shows that the representation of the data is appropriate, but more work is required in making the zooming simpler. The usability questions seem to show that this visualisation is less successful than the line chart.

Table 24 - Responses to the usability questions for the waterfall plot.

Q27. The instructions on the page were helpful.	4.2
Q28. The visualisation was easy to use.	3.3
Q29. The visualisation was responsive.	3.5
Q30. The visualisation was clear and easy to understand.	3.5

Table 24 shows we need to improve the usability of the waterfall plot. Although the users did not struggle to get the results they required, the user experience was not as positive as the line chart. It is worrying that the zoom failed to work for one of the users. Suggestions from users on improvements follow.

Q31 Please comment on any part of the visualisation you found hard to use

The two features users had the most problem with are zooming and the RFI mask. Below are some representative comments.

"Zooming in and out was very hard and completely non-intuitive."

There were other users who agreed. A more intuitive way to zoom needs to be implemented. Selecting a data range using the red box is either not implemented well or is not an intuitive way to zoom.

Q32 Please comment on how you would improve this visualisation

The main common improvement here was to provide a home zoom button to allow the users to reset the zoom.

5.4.3 General Comments

After completing these tasks users were asked to explore the other functionality on the RFI monitor website at their own discretion and provide comments. These comments are analysed below.

Q33 If you work as an engineer, manager or scientist involved with the KAT7/MeerKAT projects. Would you use this RFI monitor as part of your operations? If so can you please explain how you would use this monitor?

The responses here were generally positive; they show that there is a need for a RFI monitoring system on site. The responses show that the RFI monitor we provided is useful for employees of SKA-SA. Some example comments are shown below.

"It would be useful for telescope operators to have a browser tab showing the waterfall plot to compare to the waterfall plot for KAT7"

"Yes. As I use data from the KAT7 [beamformer] regularly, I would find this monitor useful for RFI characterisation of such observations for reference"

"I could use the RFI monitor to help me monitor RFI on site. It would be especially useful to allow me to corroborate my RFI measurements which I gather independently on site."

These comments show that although the RFI monitor could use improvement, it does achieve the goal of providing a RFI monitor that is useful to scientists, engineers and RFI management.

<u>Q34 If you could add extra functionality to the RFI monitor what would you add?</u>

The most common functions requested were:

A description of the RFI monitor, including information such as how the system was calibrated and where the antenna is situated.

Statistics over weeks and months as well as the hourly statistics.

Integration with telescope system so data can be accessed and collated with telescope data.

These suggestions and some other ideas will be discussed in the future work section 6.1 of this thesis.

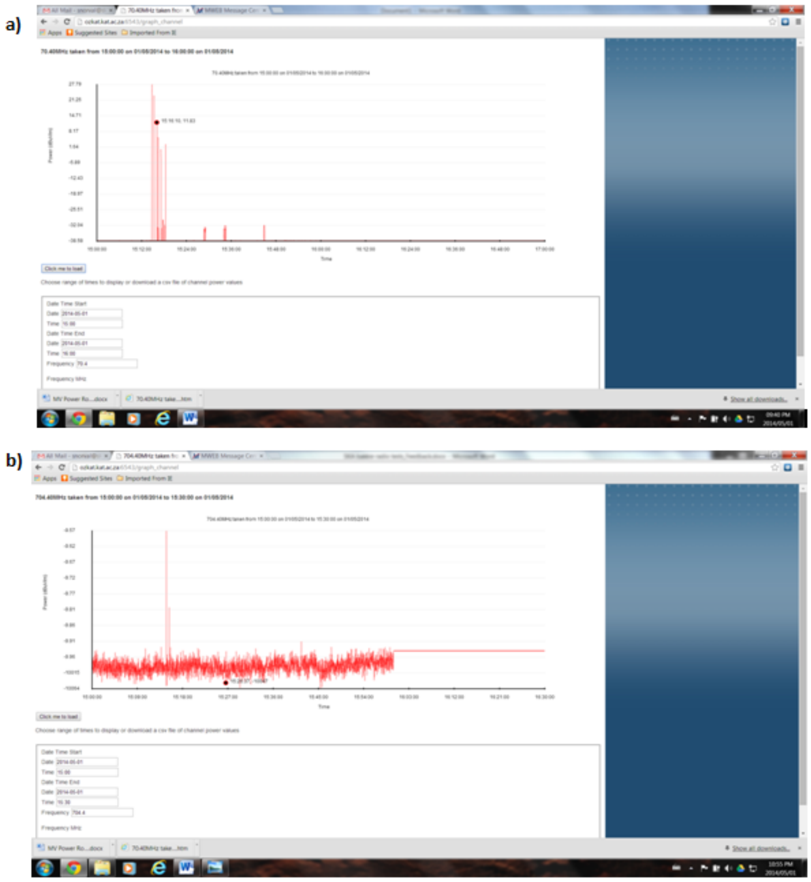
5.5 Case study

In this section we discuss how the RFI monitor was used in an RFI measurement campaign on the MeerKAT site. The purpose of the campaign was to discover whether the two way radios in the bakkies on site cause harmonics which interfere with the KAT7 or MeerKAT telescopes. The radios themselves transmit on a fundamental frequency of 70.4 MHz. There could be a repeated signal on multiples of 70.4MHz, we call these repeated signals harmonics. The harmonics are less powerful than the 70.4MHz signal and harmonics further from the fundamental are weaker. If harmonics are detectable by the RFI monitor, they could interfere with KAT7 and MeerKAT.

The test was conducted by the Head of the RFI mitigation team at SKA-SA. He went on site and keyed the radio multiple times in the beam of the RFI monitor antenna. He used the RFI monitor's graph channel view to look for harmonics at the time of the tests. Harmonics were detectable up to the 11th harmonic, at which stage the harmonics were out of the monitor's band. You can see some example plots captured by Simon in Figure 53.

This proved that the RFI monitor was useful for on-site measurements; it also showed that the monitor improves the efficiency of these types of tests. Without the RFI monitor Simon would have had to set up his own antenna and spectrum analyser, co-ordinate keying the radio and capturing data, upload the data from the spectrum analyser and then finally search the data. He would have had to use a CSV file to created his own graphs with excel. With the RFI monitor in place, Simon just had to key the radio and remember the time he did so. He could then use the online visualiser to search for the harmonics.

The result of these tests was that further controlled tests were needed in a reverberation chamber, so that all confounding factors could be removed. It was also decided on the strength of these results to purchase and install a filter for the radio on one of the bakkies to test if it would remove the harmonics of the radio.



70.40MHz taken from 15:00:00 on 01/05/2014 to 16:00:00 on 01/05/2014

Choose range of times to display or download a csv file of channel power values

Date Time Start
Date 2014-05-01
Time 15:00
Date Time End
Date 2014-05-01
Time 16:00
Frequency 70.4
Frequency MHz

704.40MHz taken from 15:00:00 on 01/05/2014 to 15:30:00 on 01/05/2014

Choose range of times to display or download a csv file of channel power values

Date Time Start
Date 2014-05-01
Time 15:00
Date Time End
Date 2014-05-01
Time 15:30
Frequency 704.4
Frequency MHz

Figure 54 – Example plots captured in search of the two way radio harmonics : a) The fundamental signal at 70.4 MHz and b) the 10th harmonic clearly visible at 704.4 MHz

6 Conclusions

In this thesis the effects of RFI on observations made with radio telescopes have been presented. Different methods for mitigating RFI were described, ranging from legislation to post observation RFI excision. We examined how RFI is handled at the MeerKAT: the site had been protected by legislation and RFI on the site has been well characterised. We found that RFI management is a continuous process which needs to be informed by accurate and current information about the RF environment. RFI monitoring is performed at many radio telescopes and has proven to be an effective way of limiting RFI. These systems have been used to find RFI culprits and also to provide astronomers with an accurate picture of how badly their observations are affected by RFI. This led us to conclude that a continuous RFI monitoring system was necessary to maintain the clean RFI environment that the MeerKAT site was chosen for.

This thesis focussed on the process of providing a RFI monitoring system for the SKA-SA organisation on the MeerKAT site. The RFI monitor was created with three goals in mind:

- Provide automatic RFI detection
- Provide a continuous record of the RF environment
- Provide access to this data

The monitor was designed with 3 distinct types of users; astronomers who use the telescope; engineers who build the telescope and RFI managers who must limit RFI on site.

The main limitation we had to deal with was the bandwidth across which we needed to monitor. Unfortunately it was not possible at the beginning of this project to monitor the bands over which the KAT7 and MeerKAT telescopes observe. As such our RFI monitor could only be a prototype system, showing that the techniques used work in principle. In any case the prototype still proved valuable for some RFI related tasks. The techniques presented in this thesis can be extended to a system which covers the bandwidth of the instruments themselves. In fact such a system is already in development and is using much of the work done in this thesis.

The project was successful in that the outlined goals were met, within the limitations. The RATTY system was successfully upgraded into an autonomous RFI monitoring and detection system. The data was proven to be accurate. All of the data captured is run through a RFI detection which can reliably and automatically detect RFI in the environment. These detected RFI events are stored indefinitely, giving a long term record of RFI on site. This allows the RFI management team to see how different campaigns affect the RFI levels on site over time periods of months to years.

The second goal of providing a continuous high definition record of the RF environment was achieved by combining the high resolution RFI event archive with the hourly averages. Due to storage constraints, it was not possible to store all data at a high resolution. After a period of 1-2 months we delete the raw data and only keep the RFI events and hourly statistics. Although this does not allow storing a continuous long term record in high resolution, we showed that it does allow us to recreate the RF environment as if we had stored all of the data. Although there is some difference between the high resolution and recreated data, the difference is small enough that it can be attributed mainly to natural noise inherent in capturing radio waves with receivers such as the RATTY and is therefore not a substantial loss in terms of a record of RFI.

The third goal of providing access to the data collected by the RFI monitor was achieved by creating a website through which users could access the data, allowing them to download any data they were interested in over the web. JavaScript visualisations which allowed the user to explore the data on the web page were developed. These visualisations needed to be developed as there were no existing JavaScript libraries that could handle the amount of data we needed to display. This website and its visualisations have already been used to facilitate with RFI campaigns on site.

The greatest proof of the success of the RFI monitor is that it has been used by RFI managers already. It has been useful in exploring known RFI such as two way radios. It has also been used to alert RFI managers of high levels of intermittent RFI around the KAT7 dishes. This RFI may only have been discovered months later if it were not for the RFI monitor. This is strong evidence that the RFI monitor provides a valuable tool for RFI management.

6.1 Further work

Although the RFI monitor has proven to be useful in RFI management already, there are aspects of the monitor that could be improved upon. The next iteration of the RFI monitor will use a new receiver called the RATTY2 based on the old RATTY system. This new system however will cover the frequency from 100MHz to 2.6 GHz. This means it will be able to monitor over the frequencies that KAT7, MeerKAT and eventually the SKA observe on. This system is currently in development and we will be using the software and knowledge gained from this prototype to ensure it is a successful monitoring system.

As discussed in the validation portion of this thesis, the retrieval speed of hourly statistics was far too long. For the new RFI monitor we will be saving our hourly data in hdf5 files, rather than in the MYSQL database as in the case of the prototype. This should significantly reduce the retrieval time of archived data.

Some other features we are planning on adding are an omni-directional antenna which will allow the monitor to capture RFI from all over the site, rather than the approximately 60 degrees possible with the current antenna. We will also be upgrading the server to hold at least double the disk space so that we can store far more raw data. Eventually we would like to add the RFI monitor to the MeerKAT observations and archive. This way each observation could have an accompanying RFI report from the monitor at the time, and over the frequencies of the observation.

There are plans to provide a daily, weekly and monthly RFI report which can be mailed to RFI managers and other interested users. We would also like to include some RFI classification, so that the report can say not only how much RFI there was, but what some of the likely culprits may be. This will hopefully help RFI managers narrow down the potential emitters they will need to measure to find RFI sources in future.

7 References

ASA. 2013, "How to become an astronomer," *http://asa.astronomy.org.au/become.html* [7 June 2013]

Astrobaki. 2013, "Radiometer Equation Applied to Telescopes," *wiki, https://casper.berkeley.edu/astrobaki/index.php/Radiometer_Equation_Applied_to_Telescope*s [14 June 2013]

Baan, W. 2011, "RFI mitigation in radio astronomy," In *General Assembly and Scientific Symposium, 2011 XXXth URSI*, pp. 1, 2, 13-20

Bolli, P., Gaudiomonte, F., & Messina, F. 2010 "The RFI monitoring systems for the Medicina and the Sardinia Radio Telescopes," In *Proceedings of Science; RFI Mitigation Workshop*, pp. 1–6, March 29-31, Groningen

Boonstra, A. J. 2001, "Radio frequency monitoring for radio astronomy," In *IUCAF RFI-Mitigation Workshop*, Bonn, March 28 - 30

Boonstra, A. J. 2005, "Radio frequency interference mitigation in radio astronomy," *Ph.D thesis, Delft University of Technology*

Bremer, M. 1995, "Conversion of source velocity/redshift to sky frequency," *web page, http://iram.fr/IRAMFR/ARN/may95/node4.html* [7 June 2013]

Burke, B. 2010, *An introduction to radio astronomy*, 3rd Ed, Cambridge Iniversity Press, Cambridge

CASPER. 2013, "ROACH" *https://casper.berkeley.edu/wiki/ROACH*, [4 February 2014]

Chaisson, E., & McMillan, S. 2005, *Astronomy today*. 5th Ed, Addison-Wesley

Department of Science & Technology. 2007, “Astronomy Geographic Advantage Act” *National Gazettes, No 37397*

Dewdney, P., Hall, P., & Schilizzi, R.. 2009, "The square kilometre array," In *Proceedings of the IEEE*, vol. 97, no. 8, pp. 1482-1496,

Diepenbeek, C. Van. 2010, "Passive Spectrum Use and Upcoming Changes in the Spectrum Environment," In *Proceedings of Science; RFI Mitigation Workshop*, pp. 1–7, March 29-31, Groningen

Driel, W. van, Gergely, T., Liszt, H., & Ohishi, M. 2011, "Expert Panel on Radio Quiet Zone and RFI Regulation," In *Expert and SDPO Reports and Supporting Material - RFI*, SKA, Manchester

Ekers, R. D., & Bell, J. F. 1999, "Radio frequency interference," In *Proceedings of IAU Symposium 199; The Universe at Low Radio Frequencies*, pp. 498-505, 30 Nov - 4 Dec, Pune

Fridman, P. A. 2010, "Statistically Stable Estimates of Varience in Radioastronomical Observations as Tools for RFI Mitigation," In *The Astronomical Journal*, vol. 135, no. 5, pp. 1810–1824

Fridman, P. A., & Baan, W. A. 2001, "RFI mitigation methods in radio astronomy," In *Astronomy & Astrophysics*, Vol. 378, no. 1, pp. 327–344

Garcia-Molina, H., Ullman, J. D., & Widom, J. 2008, *Database Systems: The Complete Book* 2nd Ed,. Prentice Hall, New Jersey

Gillani, S. 2010, "RF Interference Monitoring for the Onsala Space Observatory," *M.Sc. thesis, Chalmers University of Technology*

GMRT. 2008, "Introducing GMRT," *http://gmrt.ncra.tifr.res.in/gmrt_hpage/GMRT/intro_gmrt.html* [10 May 2012]

Guner, B., Johnson, J. T., & Niamsuwan, N. 2007, "Time and Frequency Blanking for Radio-Frequency Interference Mitigation in Microwave Radiometry," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 45 no. 11, pp. 3672–3679

HDF Group. n.d -a , "HDF5 Tutorial: Learning the Basics HDF5 File Organization," *http://www.hdfgroup.org/HDF5/Tutor/fileorg.html* [13 March 2013]

HDF Group. n.d. -b, "Overview of Parallel HDF5 Design," http://www.hdfgroup.org/HDF5/Tutor/poverview.html [13 March 2013]

HDF Group. 2011, "What is HDF5?" *http://www.hdfgroup.org/HDF5/whatishdf5.html* [13 March 2013]

ITU-RA. 2005, "Protection criteria used for radio astronomical measurements," *recommendation, 769-2, Geneva*

Joardar, S. 2005, "RFI monitoring system of GMRT and radio interference analysis on various radio-astronomy bands," In *Proceedings of the XXVIIIth URSI General Assembly*, New Dehli

Kraus, D. K. 1986, "Radio Astronomy," 2nd Ed, *Cygnus-Quasar Books*

Kempner, J. n.d, "Cosmology calculator," *http://www.kempner.net/cosmic.php* [17 November 2014]

Kristol, D. M. n.d, "HTTP What is HTTP" *http://www.silicon-press.com/briefs/brief.http/brief.PDF,* [18 March 2013]

Manners, P. 2007, "Measuring the RFI environment of the South African SKA site," *M.Sc. thesis, Rhodes University*

Miller, D. F. 1998, "Basics of Radio Astronomy for the Goldstone-Apple Valley Radio Telescope," *http://www2.jpl.nasa.gov/radioastronomy/radioastronomy_all.PDF* [19 November 2014]

National Astronomy and Ionosphere Center. n.d, "The Observatory," *http://www.naic.edu/general/index.php?option=com_content&view=article&id=149&Itemid=631* [10 May 2012]

Netcraft. 2014, "January 2014 Web Server Survey," *http://news.netcraft.com/archives/2014/01/03/january-2014-web-server-survey.html* [30 January 2014]

Numpy Developers. 2013, "Numpy," *http://www.Numpy.org/#* [5 February 2014]

Offringa, A. 2012, "Algorithms for radio interference detection and removal," *Ph.D thesis, University of Groningen*

Offringa, A. R., Bruyn, A. G. De, & Biehl, M. 2010, "Post correlation radio frequency interference classification methods," *Monthly Notices of the Royal Astronomical Society,* vol. *167*, no. 1, pp. 155–167

PAPER. n.d, "PAPER," *http://eor.berkeley.edu/* [17 Jan 2015]

Parsons, A., Werthimer, D., & Backer, D. 2009, "Digital Instrumentation for the Radio Astronomy Community," *Astro2010: The Astronomy and Astrophysics Decadal Survey, Technology Development Papers, no. 21*

Perillat, P. n.d , "Hilltop RFI monitorying system," *http://www.naic.edu/~phil/rfi/hilltop/hilltop.html#History_* [10 May 2012]

Porko, J.-P. G. 2011, "Radio frequency interference in radio astronomy," *M.Sc thesis, Aalto University*

Python Software Foundation. 2012, "PythonSpeed," *https://wiki.python.org/moin/PythonSpeed* [5 February 2014]

Quick, D. 2011, "China building world’s biggest radio telescope," *http://www.gizmag.com/five-hundred-meter-aperture-spherical-radio-telescope/18930/* [17 July 2012]

Redmonk. 2014 , "The RedMonk Programming Language Rankings: January 2014," *http://redmonk.com/sogrady/2014/01/22/language-rankings-1-14/* [04 February 2014]

SKA. 2015, "SKA Project," *https://www.skatelescope.org/project/* [19 Novermber 2015]

SKA SA. 2006a , "Analysis of the Radio Frequency Environment Transmitter Database and Propagation Studies," In *Proposal to site the Square Kilometre Array,* SKA SA, Cape Town

SKA SA. 2006b, "Analysis of the Radio Frequency Report," In *Proposal to site the Square Kilometre Array*, SKA SA, Cape Town

SKA SA. 2011, "South africa’s meerkat array," *http://www.ska.ac.za/download/fact_sheet_meerkat_2011.PDF* [17 July 2012]

Tennyson, J. 2010, *Astronomical Spectroscopy: An Introduction to the Atomic and Molecular Physics of Astronomical Spectra,* 2nd Ed, World Scientific Publishing Company, Singapore

Thompson, A. 1999, "Fundamentals of radio interferometry," In *Synthesis Imaging in Radio Astronomy II*, vol. 180, pp. 11-36

Vennapoosa, C. 2012, "The Evolutionary Prototyping Model," *http://www.exforsys.com/career-center/project-management-life-cycle/the-evolutionary-prototyping-model.html* [10 June 2013]

Webber, J. 2011, "Box2D as a Measure of Runtime Performance," *http://blog.j15r.com/blog/2011/12/15/Box2D_as_a_Measure_of_Runtime_Performance* [29 January 2014]

World Wide Web Consortium. n.d -a , "HTML & CSS," *http://www.w3.org/standards/webdesign/htmlcss* [29 January 2014]

World Wide Web Consortium. n.d. -b, "JavaScript Web APIs," *web page, http://www.w3.org/standards/webdesign/script* [29 January 2014]

World Wide Web Foundation. 2012, "History of the Web," *https://www.webfoundation.org/vision/history-of-the-web/,* [29 January 2014]

Zoller, J. N. 2011, "Satellite Regulations," *http://www.itu.int/net/newsroom/wrc/2012/features/satellite_regulations.aspx* [18 November 2014]

Appendix A : Code

A.1 Roach_handle.py

```
#!/usr/bin/python
import pylab,h5py,time, corr, numpy, struct, sys, logging, os, cam, cal

bram_out_prefix = 'store'

class roach_handle:
    def __init__(self, init = False):
        try:
            print 'Connecting to ROACH...'
            self.r = cam.spec()

            if (init):
                self.rfi_init() #initiate rfi prog on fpga
                self.last_cnt = self.getUnpackedData(self.r.fpga.read_uint('acc_cnt'))[2]

            if self.r.spectrum_bits != 64:
                print 'ERR: Sorry, this is only for 64 bit systems.'
                exit()

            #Access configuration of RATTY
            self.acc_time, self.n_accs = self.r.acc_time_get()      #Get time for each accumulation
and number of accumulations
            self.freqs = self.r.freqs
            self.fft_shift = self.r.fft_shift_get()
            self.fft_scale = self.r.fft_scale
            self.rf_gain = self.r.rf_status_get()[1]
            self.bandwidth = self.r.bandwidth
            self.n_chans = self.r.n_chans
            self.bandshape = cal.bandshape(self.freqs)

            print 'Scaling back by %i accumulations.'%self.n_accs

            self.last_cnt = self.r.fpga.read_uint('acc_cnt')

            self.af=None
            self.units='dBm'

        except Exception as e:
            print 'Runtime error: ',e
            raise e
            exit()


    def getSpectrum(self,n_acc):

            spectrum = []
            acc_cnt = []
            adc_bad = []
            timestamp = []
            adc_overrange = []
            fft_overrange = []
            adc_shutdown = []
            adc_level = []
            input_level = []
            adc_temp = []
            ambient_temp = []

        while n_acc:                #While we still need to grab next spectra
            spectra, time, self.last_cnt, stat = self.getUnpackedData(self.last_cnt)
            spectrum.append(spectra)
            acc_cnt.append(self.last_cnt)
            timestamp.append(time)
            adc_bad.append(stat['adc_bad'])
            adc_overrange.append(stat['adc_overrange'])
            fft_overrange.append(stat['fft_overrange'])
            adc_shutdown.append(stat['adc_bad'])
            adc_level.append(stat['adc_level'])
            input_level.append(stat['input_level'])
            adc_temp.append(stat['adc_temp'])
            ambient_temp.append(stat['ambient_temp'])
            n_acc = n_acc - 1

        freqs = numpy.arange(self.n_chans)*float(self.bandwidth)/self.n_chans #channel center
freqs in Hz
        bandshape = cal.bandshape(freqs)

        ret = {'spectrum':spectrum, 'acc_cnt':acc_cnt, 'timestamp':timestamp,
'adc_overrange':adc_overrange, 'fft_overrange':fft_overrange, 'adc_shutdown':adc_shutdown,
'adc_level':adc_level, 'input_level':input_level, 'adc_temp':adc_temp,
'ambient_temp':ambient_temp, 'bandshape':bandshape, 'adc_bad': adc_bad}

        if (self.r.antenna_bandpass != 'none'):
            af=cal.af_from_gain(freqs,cal.ant_gains(self.r.antenna_bandpass,freqs)) #antenna
factor
            ret['antenna_factor'] = af

            return ret

    def getAttributes (self):

        ret = dict()
        ret['n_chans'] = self.r.n_chans
        ret['n_accs'] = self.r.acc_time_get()[1]
        ret['bitstream'] = self.r.bitstream
        ret['bandwidth'] = self.r.bandwidth
        ret['adc_type'] = self.r.adc_type
        ret['spectrum_bits'] = self.r.spectrum_bits
        ret['fft_shift'] = self.r.fft_shift_get()
        ret['rf_gain'] = self.r.rf_status_get()[1]
        ret['antenna_calfile'] = self.r.antenna_bandpass

        return ret


    def getUnpackedData(self,last_cnt):
        """Gets data from ROACH board and returns the spectra, the state of the roach at the last timestamp"""
        while self.r.fpga.read_uint('acc_cnt') == last_cnt:  #Wait untill the next accumulation has been performed
            time.sleep(0.1)
            #print "cnt = " + str(self.r.fpga.read_uint('acc_cnt'))
        spectrum = numpy.zeros(self.r.n_chans)                                      #Get spectra
        for i in range(self.r.n_par_streams):
            spectrum[i::self.r.n_par_streams] = numpy.fromstring(self.r.fpga.read('%s%i'%(bram_out_prefix,i),self.r.n_chans/self.r.n_par_streams*8),dtype=numpy.uint64).byteswap()
        stat = self.r.status_get()
        ampls = self.r.adc_amplitudes_get()
        stat['adc_level'] = ampls['adc_dbm']
        stat['input_level'] = ampls['input_dbm']
        stat['adc_temp'] = self.r.adc_temp_get()
        stat['ambient_temp'] = self.r.ambient_temp_get()
        last_cnt = self.r.fpga.read_uint('acc_cnt')
        timestamp = time.time()

        #print '[%i] %s: input level: %5.2f dBm (ADC %5.2f dBm).'%(last_cnt,time.ctime(timestamp),stat['input_level'],stat['adc_level']),
        if stat['adc_bad']: print 'ADC selfprotect due to overrange!',
        elif stat['adc_overrange']: print 'ADC is clipping!',
        elif stat['fft_overrange']: print 'FFT is overflowing!',
        #else: print str(last_cnt) + 'all ok.',
        #print ''
        return spectrum, timestamp, last_cnt, stat

    def rfi_init (self):
        try:
            #r = rfi_sys.rfi_sys(mode=args[0]).
            print 'Connecting to ROACH...',
            self.r.logger.setLevel(logging.DEBUG)
            print 'done.'

            fpga_prog = True #Set to default value in rfi_init I got from Jason
            fft_shift = -1 #Set to default value in rfi_init I got from Jason
            acc_period = 1 #Set to default value in rfi_init I got from Jason (1 second)


            print '-----------------------'
            print 'Programming FPGA...',
            sys.stdout.flush()
            if fpga_prog:
                self.r.fpga.progdev(self.r.bitstream)
                print 'done'
            else:
                print 'Skipped.'

            print 'Checking clocks...',
            sys.stdout.flush()
            if fpga_prog:
                est_clk_rate=self.r.clk_check()
                print 'ok, %i MHz.'%est_clk_rate
            else:
                print 'Skipped.'

            print 'Auto-calibrating ADC...',
            sys.stdout.flush()
            self.r.adc_selfcal()
            print 'done'

            print 'Attempting automatic RF gain adjustment...'
            max_n_tries=10
            n_tries=0
            tolerance=1
            rf_gain=self.r.rf_gain_range[0]
            self.r.rf_gain_set(rf_gain)
            time.sleep(0.1)
            self.r.ctrl_set(mrst='pulse',cnt_rst='pulse',clr_status='pulse',flasher_en=True)
            rf_level=self.r.adc_amplitudes_get()['adc_dbm']
            if self.r.status_get()['adc_bad'] or self.r.status_get()['adc_overrange']:
                raise RuntimeError('Your input levels are too high!')

            while (rf_level < self.r.desired_rf_level-tolerance or rf_level>self.r.desired_rf_level+tolerance) and n_tries < max_n_tries:
                rf_level=self.r.adc_amplitudes_get()['adc_dbm']
                difference = self.r.desired_rf_level - rf_level
                rf_gain=self.r.rf_status_get()[1] + difference
                print '\t Gain was %3.1fdB, resulting in an ADC input level of %5.2fdB. Trying gain of %4.2fdB...'%(self.r.rf_status_get()[1],rf_level,rf_gain)
                if rf_gain < self.r.rf_gain_range[0]:
                    print '\tWARNING: Gain at minimum, %4.2fdB.'%self.r.rf_gain_range[0],
                    self.r.logger.warn('Gain at minimum, %4.2fdB.'%self.r.rf_gain_range[0])
                    self.r.rf_gain_set(self.r.rf_gain_range[0])
                    break
                elif rf_gain > self.r.rf_gain_range[1]:
                    print '\t WARNING: Gain at maximum, %4.2fdB.'%self.r.rf_gain_range[1],
                    self.r.logger.warn('Gain at maximum, %4.2fdB.'%self.r.rf_gain_range[1])
                    self.r.rf_gain_set(self.r.rf_gain_range[1])
                    break
                self.r.rf_gain_set(rf_gain)
                time.sleep(0.1)
                n_tries += 1
            if n_tries >= max_n_tries: print 'Failed.'
            else: print 'done!'

            print 'Setting FFT shift... ',
            sys.stdout.flush()
            self.r.fft_shift_set(fft_shift)
            print 'set to 0x%x.'%self.r.fft_shift_get()

            print 'Configuring accumulation period to %2.2f seconds...'%acc_period,
            sys.stdout.flush()
            self.r.acc_time_set(acc_period)
            print 'done'

            print 'Resetting counters...',
            sys.stdout.flush()
            self.r.ctrl_set(mrst='pulse',cnt_rst='pulse',clr_status='pulse',flasher_en=False)
            print 'done'

            print 'Current status:',
            sys.stdout.flush()
            stat=self.r.status_get()
            if stat['adc_bad']: print 'ADC selfprotect due to overrange!',
            elif stat['adc_overrange']: print 'ADC is clipping!',
            elif stat['fft_overrange']: print 'FFT is overflowing!',
            else: print 'all ok',
            print ''

        except KeyboardInterrupt:
            exit_clean()
        except Exception as e:
            print e
            exit_fail()

    def exit_clean(self):
        try:
            self.r.fpga.stop()
        except:
            pass
        exit()

    def exit_fail(self):
        print 'FAILURE DETECTED. Log entries:\n',
        try:
            self.r.lh.printMessages()
            self.r.fpga.stop()
        except Exception as e:
            print e
            pass
            raise e
        exit()

if __name__ == '__main__':
    test = roach_handle()
    data = test.getSpectrum(5)
    for i in range(len(data['acc_cnt'])):
        print '\n----------------%i-------------------'%i
        print data['spectrum'][i]
        print data['acc_cnt'][i]
        print data['timestamp'][i]
        print data['adc_overrange'][i]
        print data['adc_level'][i]
        print data['input_level'][i]
        print data['adc_temp'][i]
        print data['ambient_temp'][i]
    test.exit_clean()
```

A. 2 cam.py

```
#!/usr/bin/env python
'''
You need to have KATCP and CORR installed. Get them from
http://pypi.python.org/pypi/katcp and
http://casper.berkeley.edu/svn/trunk/projects/packetized_correlator/corr-0.4.0/

Hard-coded for 32bit unsigned numbers.
\nAuthor: Jason Manley, Feb 2011.
'''

import corr,time,numpy,struct,sys,logging,cal,iniparse, os

front_led_layout=['adc_clip','adc_shutdown','fft_overflow','quantiser_overflow','new_accumulation','sync','NA','NA']

#roach='192.168.64.112' Edit by Chris
#mode_params={'hr': {'bitstream':'r_spec_1ghz_16k_r106_2011_Feb_24_1810.bof',
# mode_params={
#           #'hr': {'bitstream':'r_spec_1ghz_16k_iadc_r106_2011_Mar_10_1724.bof',
#           'hr': {'bitstream':'r_spec_1ghz_16k_iadc_r107_2011_Mar_14_0850.bof',
#                   'n_chans':16384,
#                   'n_par_streams':4,
#                   'bandwidth':898000000,
#                   'desired_rf_level':-25,
#                   'adc_type':'iadc',
#                   'spectrum_bits':64,
#                   'fft_shift':0b0011111111111100},
#           'hr_900': {'bitstream':'r_spec_1ghz_16k_iadc_r107_2011_Mar_14_0850.bof',
#                   'n_chans':16384,
#                   'n_par_streams':4,
#                   'bandwidth':900000000,
#                   'adc_type':'iadc',
#                   'desired_rf_level':-25,
#                   'spectrum_bits':64,
#                   'fft_shift':16383},
#           #'hr_kadc': {'bitstream':'r_spec_1ghz_16k_kadc_r108_2011_Jul_26_1810.bof',
#           'hr_kadc': {'bitstream':'r_spec_1ghz_16k_kadc_r108_2011_Nov_09_1541.bof',
#                   'n_chans':16384,
#                   'n_par_streams':4,
#                   'bandwidth':800000000,
#                   'adc_type':'katadc',
#                   'desired_rf_level':-25,
#                   'spectrum_bits':64,
#                   'fft_shift':16383},
#            #'lr': {'bitstream':'r_spec_1ghz_1k_r108lr_2011_Feb_28_1051.bof',
#            'lr': {'bitstream':'r_spec_1ghz_1k_iadc_r108lr_2011_Feb_28_1655.bof',
#                   'n_chans':1024,
#                   'desired_rf_level':-25,
#                   'n_par_streams':4,
#                   'adc_type':'iadc',
#                   'spectrum_bits':32,
#                   'bandwidth':900000000,
#                   'fft_shift':1023},
#           }
#katcp_port=7147 Edit by Chris

class spec:
    def __init__(self, log_handler=None, log_level=logging.INFO):
        #-------------------------------------Code By Chris------------------------------------------
        self.config_file = cal.cal_files("system_parameters")
        try:
            self.sys_config = iniparse.INIConfig(open(self.config_file, 'rb'))     #load config file
        except IOError as e:
            print "Error opening the config file : ",
            print e
            exit()
        roach = self.sys_config['connection']['roach_ip'].strip()              #load Roach IP
        katcp_port = int(self.sys_config['connection']['katcp_port'])              #load Roach port
        #----------------------------------End Code By Chris------------------------------------------
        if log_handler == None: log_handler=corr.log_handlers.DebugLogHandler(100)
        self.lh = log_handler
        self.logger = logging.getLogger('RFIsys')

self.fpga=corr.katcp_wrapper.FpgaClient(roach,katcp_port,timeout=10,logger=self.logger)
        self.logger.setLevel(log_level)
        self.logger.addHandler(self.lh)
        time.sleep(1)
        try:
            self.fpga.ping()
            self.logger.info('KATCP connection ok.')
        except Exception as e:
            self.logger.error('KATCP connection failure. Connection to ROACH failed.')
            print e
            print('KATCP connection failure.')
            raise RuntimeError("Connection to FPGA board failed.")
        #--------------------------------Edited by Chris-------------------------------------------------
        #self.mode = self.sys_config['digital_system_parameters']['mode'].strip()
        self.n_chans = int(self.sys_config['digital_system_parameters']['n_chans'])
        self.bandwidth = int(self.sys_config['digital_system_parameters']['bandwidth'])
        self.n_par_streams = int(self.sys_config['digital_system_parameters']['n_par_streams'])
        self.bitstream = self.sys_config['digital_system_parameters']['bitstream']
        self.fft_shift = int(self.sys_config['digital_system_parameters']['fft_shift'])
        self.adc_type = self.sys_config['digital_system_parameters']['adc_type']
        self.desired_rf_level =
int(self.sys_config['digital_system_parameters']['desired_rf_level'])
        self.spectrum_bits = int(self.sys_config['digital_system_parameters']['spectrum_bits'])
        self.antenna_bandpass = self.sys_config['analogue_frontend']['antenna_bandpass']
        if self.adc_type== 'katadc':
            self.fpga_clk=self.bandwidth/4
            self.sample_clk=self.bandwidth*2
            self.rf_gain_range=(-11.5,20)
        elif self.adc_type== 'iadc':
            self.fpga_clk=self.bandwidth/4
            self.sample_clk=self.bandwidth*2
            self.rf_gain_range=(-31.5,0)
        self.chan_width=numpy.float(self.bandwidth)/self.n_chans
        self.freqs=numpy.arange(self.n_chans)*float(self.bandwidth)/self.n_chans #channel
center freqs in Hz


        #-----------------------------End Edited By Chris ---------------------------------------------

    def initialise(self,rf_gain=-10,acc_time=1,fft_shift=0xffffffff):
        """Initialises the system to defaults."""
        self.fpga.progdev(self.bitstream)
        self.fft_shift_set(fft_shift)
        self.rf_gain_set(rf_gain)
        self.acc_time_set(acc_time)
        self.ctrl_set(flasher_en=False,cnt_rst='pulse',clr_status='pulse')
        #self.ctrl_set(flasher_en=True,cnt_rst='pulse',clr_status='pulse')

    def clk_check(self):
        """Performs a clock check and returns an estimate of the FPGA's clock frequency."""
        est_rate=round(self.fpga.est_brd_clk())
        if est_rate>(self.fpga_clk/1e6 +1) or est_rate<(self.fpga_clk/1e6 -1):
            self.logger.error('FPGA clock rate is %i MHz where we expect it to be %i
MHz.'%(est_rate,self.fpga_clk/1e6))
            raise RuntimeError('FPGA clock rate is %i MHz where we expect it to be %i
MHz.'%(est_rate,self.fpga_clk/1e6))
        return est_rate

    def adc_selfcal(self):
        if self.adc_type=='iadc':

corr.iadc.configure(self.fpga,0,mode='inter_I',cal='new',clk_speed=self.bandwidth/1000000)
        elif self.adc_type=='katadc':
            corr.katadc.set_interleaved(self.fpga,0,'I')
            time.sleep(0.1)
            corr.katadc.cal_now(self.fpga,0)

    def fft_shift_set(self,fft_shift_schedule=-1):
        """Sets the FFT shift schedule (divide-by-two) on each FFT stage.
            Input is an integer representing a binary bitmask for shifting.
            If not specified as a parameter to this function (or a negative value is supplied),
program the default level."""
        import cal
        if fft_shift_schedule<0: fft_shift_schedule=self.fft_shift
        self.fpga.write_int('fft_shift',fft_shift_schedule)
        self.fft_shift=fft_shift_schedule
        self.fft_scale=2**(cal.bitcnt(fft_shift_schedule))
        self.logger.info("Set FFT shift to %8x (scaling down by
%i)."%(fft_shift_schedule,self.fft_scale))

    def fft_shift_get(self):
        """Fetches the current FFT shifting schedule from the hardware."""
        self.fft_shift=self.fpga.read_uint('fft_shift')
        self.fft_scale=2**(cal.bitcnt(self.fft_shift))
        return self.fft_shift
#         return self.fft_scale

    def ctrl_get(self):
        """Reads and decodes the values from the control register."""
        value = self.fpga.read_uint('control')
        return {'mrst':bool(value&(1<<0)),
                'cnt_rst':bool(value&(1<<1)),
                'clr_status':bool(value&(1<<3)),
                'adc_protect_disable':bool(value&(1<<13)),
                'flasher_en':bool(value&(1<<12)),
                'raw':value,
                }

    def ctrl_set(self,**kwargs):
         """Sets bits of all the Fengine control registers. Keeps any previous state.
             \nPossible boolean kwargs:
             \n\t adc_protect_disable
             \n\t flasher_en
             \n\t clr_status
             \n\t mrst
             \n\t cnt_rst"""

         key_bit_lookup={
             'adc_protect_disable':   13,
             'flasher_en':   12,
             'clr_status':   3,
             'cnt_rst':      1,
             'mrst':        0,
             }
         value = self.ctrl_get()['raw']
         run_cnt=0
         run_cnt_target=1
         while run_cnt < run_cnt_target:
             for key in kwargs:
                 if (kwargs[key] == 'toggle') and (run_cnt==0):
                     value = value ^ (1<<(key_bit_lookup[key]))
                 elif (kwargs[key] == 'pulse'):
                     run_cnt_target = 3
                     if run_cnt == 0: value = value & ~(1<<(key_bit_lookup[key]))
```

```
                    elif run_cnt == 1: value = value | (1<<(key_bit_lookup[key]))
                    elif run_cnt == 2: value = value & ~(1<<(key_bit_lookup[key]))
                elif kwargs[key] == True:
                    value = value | (1<<(key_bit_lookup[key]))
                elif kwargs[key] == False:
                    value = value & ~(1<<(key_bit_lookup[key]))
                else:
                    raise RuntimeError("Sorry, you must specify True, False, 'toggle' or 'pulse' for
%s."%key)
            self.fpga.write_int('control', value)
            run_cnt = run_cnt +1

    def rf_gain_set(self,gain=None):
        """Enables the RF switch and configures the RF attenuators on KATADC boards. \n
        \t KATADC's valid range is -11.5 to 20dB. \n"""
        self.rf_gain=gain
        if self.adc_type == 'katadc':
            #RF switch is in MSb.
            if gain > 20 or gain < -11.5:
                raise RuntimeError("Invalid gain setting of %i. Valid range for KATADC is -
11.5 to +20dB.")
            self.fpga.write_int('adc_ctrl0',(1<<31)+int((20-gain)*2))
        elif self.adc_type == 'iadc':
            if gain > 0 or gain < -31.5:
                raise RuntimeError("Invalid gain setting of %i. Valid range for RFI frontend is -
31.5 to 0dB.")
            self.fpga.write_int('adc_ctrl0',(1<<31)+int((0-gain)*2))
            #print 'Set RF gain register to %x'%int((0-gain)*2)
        else: raise RuntimeError("Sorry, your ADC type is not supported.")

    def rf_status_get(self):
        """Grabs the current value of the RF attenuators and RF switch state. returns
(enabled,gain in dB)."""
        if self.adc_type == 'katadc':
            value = self.fpga.read_uint('adc_ctrl0')
            self.rf_gain=20.0-(value&0x3f)*0.5
            return (bool(value&(1<<31)),self.rf_gain)
        elif self.adc_type == 'iadc':
            value = self.fpga.read_uint('adc_ctrl0')
            self.rf_gain=0.0-(value&0x3f)*0.5
            return (bool(value&(1<<31)),self.rf_gain)
        else: raise RuntimeError("Sorry, your ADC type is not supported.")

    def adc_amplitudes_get(self):
        """Gets the ADC RMS amplitudes."""
        adc_levels_acc_len=65536
        if self.adc_type == 'katadc':
            adc_bits=8
        elif self.adc_type == 'iadc':
            adc_bits=8
        rv = {}
        rv['adc_raw']=self.fpga.read_uint('adc_sum_sq0')
        rv['adc_rms_raw']=numpy.sqrt(rv['adc_raw']/float(adc_levels_acc_len))
        rv['adc_rms_mv']=rv['adc_rms_raw']*cal.get_adc_cnt_mv_scale_factor()
        rv['adc_dbm']=cal.v_to_dbm(rv['adc_rms_mv']/1000.)

rv['input_rms_mv']=rv['adc_rms_raw']*cal.get_adc_cnt_mv_scale_factor(self.rf_status_get()[
1])
        rv['input_dbm']=cal.v_to_dbm(rv['input_rms_mv']/1000.)
        return rv

    def status_get(self):
        """Reads and decodes the status register. Resets any error flags after reading."""
        rv={}
        value = self.fpga.read_uint('status')
        self.ctrl_set(clr_status='pulse')
        return {
                'adc_bad':bool(value&(1<<4)),
                'adc_overrange':bool(value&(1<<2)),
                'fft_overrange':bool(value&(1<<1))
                }

    def acc_time_set(self,acc_time=1):
        """Set the accumulation length in seconds"""
        self.n_accs = int(acc_time * float(self.bandwidth)/self.n_chans)
        self.logger.info("Setting accumulation time to %2.2f seconds (%i
accumulations)."%(acc_time,self.n_accs))
        self.fpga.write_int('acc_len',self.n_accs)
        self.ctrl_set(mrst='pulse')

    def acc_time_get(self):
        """Set the accumulation length in seconds"""
        self.n_accs = self.fpga.read_uint('acc_len')
        self.acc_time=self.n_accs*self.n_chans/float(self.bandwidth)
        self.logger.info("Accumulation time is %2.2f seconds (%i
accumulations)."%(self.acc_time,self.n_accs))
        return self.acc_time,self.n_accs

    def get_adc_snapshot(self,trig_level=-1):
        if trig_level>0:
            self.fpga.write_int('trig_level',trig_level)
            circ_capture=True
        else:
            self.fpga.write_int('trig_level',0)
            circ_capture=False

        return
numpy.fromstring(self.fpga.snapshot_get('snap_adc',man_valid=True,man_trig=True,circular
_capture=circ_capture,wait_period=-1)['data'],dtype=numpy.int8)
```

```
    def adc_temp_get(self):
        if self.adc_type== 'katadc':
            return corr.katadc.get_adc_temp(self.fpga,0)
        else:
            return -1

    def ambient_temp_get(self):
        if self.adc_type== 'katadc':
            return corr.katadc.get_ambient_temp(self.fpga,0)
        else:
            return -1



def ByteToHex( byteStr ):
    """
    Convert a byte string to it's hex string representation e.g. for output.
    """

    # Uses list comprehension which is a fractionally faster implementation than
    # the alternative, more readable, implementation below
    #
    #    hex = []
    #    for aChar in byteStr:
    #        hex.append( "%02X " % ord( aChar ) )
    #
    #    return ''.join( hex ).strip()

    return ''.join( [ "%02X " % ord( x ) for x in byteStr ] ).strip()
```

A.3 Cal.py

```
# -*- coding: utf-8 -*-
import numpy,scipy,scipy.interpolate, iniparse


#smoothing functions from http://www.swharden.com/blog/2008-11-17-linear-data-
smoothing-in-python/

c=299792458. #speed of light in m/s
#cal_file_path = "/etc/rfi_sys/cal_files/"; #For when you have everything working and ready
to install with distutils
cal_file_path = "ratty/config_files/cal_files/"; #For development

def smoothList(list,strippedXs=False,degree=10):
    if strippedXs==True: return Xs[0:-(len(list)-(len(list)-degree+1))]
    smoothed=[0]*(len(list)-degree+1)
    for i in range(len(smoothed)):
        smoothed[i]=sum(list[i:i+degree])/float(degree)
    return smoothed

def smoothListTriangle(list,strippedXs=False,degree=5):
    weight=[]
    window=degree*2-1
    smoothed=[0.0]*(len(list)-window)
    for x in range(1,2*degree):weight.append(degree-abs(degree-x))
    w=numpy.array(weight)
    for i in range(len(smoothed)):
        smoothed[i]=sum(numpy.array(list[i:i+window])*w)/float(sum(w))
    return smoothed

def smoothListGaussian(list,strippedXs=False,degree=5):
    window=degree*2-1
    weight=numpy.array([1.0]*window)
    weightGauss=[]
    for i in range(window):
        i=i-degree+1
        frac=i/float(window)
        gauss=1/(numpy.exp((4*(frac))**2))
        weightGauss.append(gauss)
    weight=numpy.array(weightGauss)*weight
    smoothed=[0.0]*(len(list)-window)
    for i in range(len(smoothed)):
        smoothed[i]=sum(numpy.array(list[i:i+window])*weight)/sum(weight)
    return smoothed

def dmw_per_sq_m_to_dbuv(dbmw):
    # from http://www.ahsystems.com/notes/RFconversions.php: dBmW/m2 = dBmV/m - 115.8
    return dbmw + 115.8

def dbuv_to_dmw_per_sq_m(dbuv):
    # from http://www.ahsystems.com/notes/RFconversions.php: dBmW/m2 = dBmV/m - 115.8
    return dbuv - 115.8

def dbm_to_dbuv(dbm):
    return dbm+106.98

def dbuv_to_dbm(dbuv):
    return dbm-106.98

def v_to_dbuv(v):
    return 20*numpy.log10(v*1e6)

def dbuv_to_v(dbuv):
    return (10**(dbuv/20.))/1e6

def dbm_to_v(dbm):
    return numpy.sqrt(10**(dbm/10.)/1000*50)

def v_to_dbm(v):
    return 10*numpy.log10(v*v/50.*1000)

def bitcnt(val):
    '''Counts the number of set bits in the binary value.'''
    ret_val=0
    shift_val=val
    while shift_val>=1:
        if shift_val&1: ret_val +=1
        shift_val = shift_val>>1
    return ret_val

def polyfit(freqs,gains,degree=9):
    """Just calls numpy.polyfit. Mostly here as a reminder."""
    return numpy.polyfit(freqs,gain,deg=degree)

def af_from_gain(freqs,gains):
        """Calculate the antenna factor (in dB/m) from a list of frequencies (in Hz) and gains
(in dBi).
            There are a number of assumptions made for this to be valid:
             1) Far-field only (plane wave excitation).
             2) 50ohm system.
             3) antenna is polarisation matched.
             4) effects of impedance mismatch are included."""
        #From Howard's email:
        #The antenna factors are derived from gain by taking 9.73/(lambda(sqrt(Gain))  - note
the gain here is the non-dB gain. It is worth noting that in dB's the conversion is 19.8 –
20log10(lambda) – 10 log(Gain)
        return 19.8 - 20*numpy.log10(c/freqs) - gains

def gain_from_af(freqs,afs):
        """Calculate the gain (in dBi) from the Antenna Factor (in dB/m)."""
        return 19.8 - 20*numpy.log10(c/freqs) - afs

def getDictFromCSV(filename):
        import csv
        f = open(filename)
        f.readline()
        reader = csv.reader(f, delimiter=',')
        mydict = dict()
        for row in reader:
            mydict[float(row[0])] = float(row[1])
        return mydict

class cal:

    def __init__(self, config_file = 'default'):

        self.config = conf.rattyconf(config_file)


    def __init__(self, n_chans, bandwidth, n_par_streams, bitstream, fft_shfit, adc_type,
desired_rf_level, spectrum_bits, antenna_bandpass, atten_gain, system_bandpass, fe_gain,
acc_period):
        self.n_chans = n_chans
        self.bandwidth = bandwidth
        self.n_par_streams = n_par_streams
        self.bitstream = bitstream
        self.fft_shift = fft_shift
        self.adc_type = adc_type
        self.desired_rf_level = desired_rf_level
        self.spectrum_bits = spectrum_bits
        self.antenna_bandpass = antenna_bandpass
        self.atten_gain = atten_gain
        self.freqs = numpy.arange(self.n_chans)*float(self.bandwidth)/self.n_chans
        self.system_bandpass = system_bandpass
        self.fe_gain = fe_gain
        self.acc_period = acc_period
        self.n_accs = int(self.acc_period * float(self.bandwidth)/self.n_chans)

    def inter_adc_details(self, data):
        print 'DC offset 0: %f'%numpy.mean(data[0::2])
        print 'DC offset 1: %f'%numpy.mean(data[1::2])
        print 'Max 0: %f'%numpy.max(data[0::2])
        print 'Max 1: %f'%numpy.max(data[1::2])
        #print 'Phase difference estimate:
%f'%numpy.acos(2*numpy.mean(data[0::2]*data[1::2]))

    def plot_bandshape(freqs):
        import pylab
        pylab.plot(bandshape(freqs))
        pylab.title('Bandpass calibration profile')
        pylab.xlabel('Frequency (Hz)')
        pylab.ylabel('Relative response (dB)')

    def plot_atten_gain_map():
        import pylab
        inputs=atten_gain_map.keys()
        inputs.sort()
        pylab.plot(inputs,[atten_gain_map[k] for k in inputs])
        pylab.title('RF attenuator mapping')
        pylab.xlabel('Requested value (dB)')
        pylab.ylabel('Actual value (dB)')

    def get_gains_from_csv(filename):
        freqs=[]
        gains=[]
        more=True
        fp=open(filename,'r')
        import csv
        fc=csv.DictReader(fp)
        while(more):
            try:
                raw_line=fc.next()
                freqs.append(numpy.float(raw_line['freq_hz']))
                gains.append(numpy.float(raw_line['gain_db']))
            except:
                more=False
                break
        return freqs,gains

    def get_interpolated_gains(fileName):
        """Retrieves antenna gain mapping from /etc/rfi_sys/cal_files/ant.csv file and
interpolates data to return values at 'freqs'."""
        cal_freqs,cal_gains=get_gains_from_csv(cal_files(fileName + '.csv'))
        inter_freqs=scipy.interpolate.interp1d(cal_freqs,cal_gains,kind='linear')
        return inter_freqs(self.config['freqs'])

    def plot_ant_gain():
        """Plots the antenna gain as read from a CSV file specified as "ant"."""
        import pylab
        pylab.plot(self.config['freqs']/1e6,self.ant_gains())
        pylab.title('Antenna gain %s'%self.config['antenna_bandpass_file'])
        pylab.xlabel('Frequency (MHz)')
        pylab.ylabel('Relative response (dBi)')

    def plot_ant_factor():
        """Plots the antenna factor over the given frequencies as calculated from the specified
antenna CSV file."""
        import pylab
        pylab.plot(self.freqs/1e6,af_from_gain(self.freqs,ant_gains()))
        pylab.title('Antenna factor as a function of frequency (%s)'%self.config['antenna_bandpass_file'])
        pylab.xlabel('Frequency (MHz)')
        pylab.ylabel('Antenna factor (dBuV/m)')

    def get_adc_cnt_mv_scale_factor(atten_gain=None): #TODO FIX THIS!!
        """Calculate and return a scale factor for calibrating a raw ADC count to millivolts. Optional atten_gain in dB to map to input levels."""
        if atten_gain==None:
            return 3.93
        else:
            return 3.93/(10**((atten_gain_map[atten_gain]+self.config['fe_gain'])/20.))

    def freq_to_chan(frequency):
        """Returns the channel number where a given frequency is to be found. Frequency is in Hz."""
        if frequency<0:
            frequency=self.config['bandwidth']+frequency
            #print 'you want',frequency
        if frequency>self.config['bandwidth']: raise RuntimeError("that frequency is too high.")
        return round(float(frequency)/self.config['bandwidth']*self.config['n_chans'])%self.config['n_chans']

    def get_calibrated_spectrum_from_raw_snapshot(adcdata,atten,bandwidth,ant_factor=None,bandshape=None,n_chans=512):
        """Will correct for RF frontend attenuator gains, bandshape and optionally antenna response. Returns dBm unless antenna is specified, in which case returns dBuV/m."""
    #TODO: TEST THIS
        n_accs=len(adcdata)/self.config['n_chans']/2
        freqs=numpy.arange(self.config['n_chans'])*float(self.config['bandwidth'])/self.config['n_chans'] #channel center freqs in Hz. #linspace(0,float(bandwidth),n_chans) returns incorrect numbers
        window=numpy.hamming(self.config['n_chans']*2)
        spectrum=numpy.zeros(self.config['n_chans'])
        adc_data_v=get_adc_cnt_mv_scale_factor(atten_gain=atten)*adcdata/1000. #factors-in atten_gain_map and fe_gain
        for acc in range(n_accs):
            spectrum += numpy.abs((numpy.fft.rfft(adc_data_v[self.config['n_chans']*2*acc:self.config['n_chans']*2*(acc+1)]*window)[0:self.config['n_chans']])**2)
            #print (numpy.fft.rfft(adc_data_dbm[n_chans*2*acc:n_chans*2*(acc+1)]*window)[0:n_chans])
        spectrum  = 10*numpy.log10(spectrum/n_accs/self.config['n_chans']) #now in dBV
        spectrum -= 13.034
        if bandshape != None:
            spectrum -= bandshape
        if ant_factor != None:
            spectrum = dbm_to_dbuv(spectrum)
            spectrum += ant_factor
        return freqs,spectrum

    def get_calibrated_spectrum(data, desired_rf_gain):
        '''Returns a calibrated spectrum from a raw hardware spectral dump.
            Units are dBm unless an antenna is specified in which case units are dBuV/m.\n
            Performs bandpass correction, fft_scaling adjustment, overall gain compensation,
backs out number of accumulations, RF frontend gain etc.\n
        '''

        ant_factor = af_from_gain(self.config['freqs'], self.atten_gain_map)
        #SQRT?
        data_return=numpy.array(data)
        data_return /= float(self.config['n_accs'])
        data_return *= bitcnt(self.config['fft_shift'])
        #data_return /= self.chan_width
        data_return  = 10*numpy.log10(data_return)
        data_return -= self.atten_gain_map[desired_rf_gain]
        data_return -= self.config['fe_gain']
        data_return -= 120. #overall system/algorithm gain
        if bandshape != None:
            data_return -= self.bandshape
        if ant_factor !=None:
            data_return = dbm_to_dbuv(data_return)
            data_return += ant_factor
        return data_return

    def cal_files(filename):
        import os
        return "%s%s"%(cal_file_path, filename)
```

A. 4 rfi_monitor.py

```
import ratty1
import rfDB
from multiprocessing import Process, Pipe, Queue, Lock, Event
from multiprocessing import Pool
import sys
import archive_rfi_spectra as arch
import rfi_event as rfie
import time
import os
import h5py
import numpy as np
import current_spectra as cs

numLoops = 10

def threadInsert (queue):
    db = rfDB.dbControl.dbControl("localhost", "root", "meerkat", "rfimonitor")
    current = cs.current_spectra()
    count = numLoops
    last_cnt = 0
    last_day = -1;
    data = queue.get()
    localtime = time.localtime(data[1])
    fileName = "%02i.h5"%(localtime[3]) #Filename is the hour of the observation
    path = os.path.join('/home/chris/rfi_data', str(localtime[0]), "%02i"%localtime[1],
"%02i"%localtime[2],'') #Filepath year/month/day of the observation
    location = "%s%s"%(path,fileName)
    if not os.path.exists(path):
        os.makedirs(path)
    f = h5py.File(location, 'w')
    f.create_dataset('spectra', (3600, 14200),chunks = (4, 14200), dtype = type(data[0][0]),
compression='gzip', compression_opts=4)
    last_hour = localtime[3]
    last_timestamp = data[1]
    hourstart, hourend = rfDB.dbControl.get_hour(data[1])
    count = 0
    while True:
        if (last_hour != localtime[3]): #If new hour
            #Save file
            f.flush()
            f.close()

            #Event to synchronise archive and rfi detection
            archive_added_event = Event()

            #Archive last hour
            p3 = Process(target=threadArchive, args = (last_timestamp,archive_added_event))
            p3.start()
            #Detect RFI and archive
            p4 = Process(target=threadRFIDetection, args =
(last_timestamp,archive_added_event))
            p4.start()

            #Make sure at least 25% of harddrive is free
            db.maintain_space()

            #Create new file for next hour
            fileName = "%02i.h5"%(localtime[3]) #Filename is the hour of the observation
            path = os.path.join('/home/chris/rfi_data', str(localtime[0]), "%02i"%localtime[1],
"%02i"%localtime[2],'') #Filepath year/month/day of the observation
            location = "%s%s"%(path,fileName)
            if not os.path.exists(path):
                os.makedirs(path)
            f = h5py.File(location, mode='w')
            f.create_dataset('spectra', (3600, 14200),chunks = (4, 14200), dtype =
type(data[0][0]), compression='gzip', compression_opts=4)

            #reset time keeping variables to current hour
            last_hour = localtime[3]
            last_timestamp = data[1]
            current.deleteOld(last_timestamp)
        # print "inserting to current"
        current.insertSpectrum(data[0],data[1])
        # print "inserting to rfimonitor"
        db.insertDump (data[0], data[1], data[3], 1, f)
        # print "Done Inserting'"
        data = queue.get()
        #print "Timestamp = %i"%data[1]
        localtime = time.localtime(data[1])
    db.closeDB()
    current.close()
    #f.close()

def threadArchive (timestamp, archive_added_event):
    print "In archive thread"
    timeT = time.time()
    rf_archive = arch.archive_rfi_spectra()
    #rf_archive.connect()
    print "Connected to DB, Start processing"
    rf_archive.archive_last_hour(timestamp, False)
    rf_archive.exit()
    archive_added_event.set()
    print "exiting rf archival, took %i seconds"%(time.time()-timeT)

def threadRFIDetection(timestamp, archive_added_event):
    print "In RFIDetection"
    timeT = time.time()
    num_events, len_events = rfie.rfi_detection(timestamp)
    print "Completed RFIDetection, took %i seconds"%(time.time() - timeT)
    archive_added_event.wait()
    time.sleep(6)
    print "entering insert to archive"
    rfie.insert_to_archive(timestamp, num_events, len_events)
    print "Added RFI data to archive"



def threadGetSpectrum (queue):
    rat = ratty1.cam.spec()

    rat.connect()
    rat.initialise()
    count = numLoops
    while True:
        spectrum, timestamp, last_cnt, stat = rat.getUnpackedData()
        queue.put((rat.cal.get_calibrated_spectrum(spectrum,
rat.rf_status_get()[1])[rat.cal.freq_to_chan(rat.cal.config['ignore_low_freq']):rat.cal.freq_to_c
han(rat.cal.config['ignore_high_freq'])], timestamp, last_cnt, stat))
        count -= 1

def run():

    print "start"
    queue = Queue()

    try:
        p = Process(target=threadGetSpectrum, args=(queue,))
        p2 = Process(target=threadInsert, args=(queue,))
        p.start()
        p2.start()
        print "p1 and p2 started"
        p.join()
        p2.join()
    except SystemExit, e:
        print "exiting"
        sys.exit(0)

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

dBControl.py

```
import MySQLdb as mdb
import sys
import h5py
import os
import time
import numpy
import math
import pickle
import datetime

def get_hour(timestamp):
   new_hour = [0,0,0,0,0,0,0,0,0]
   new_hour[0:4] = datetime.datetime.fromtimestamp(timestamp).timetuple()[0:4]
   start_timestamp = time.mktime(new_hour)
   end_timestamp = time.mktime((datetime.datetime.fromtimestamp(start_timestamp) +
datetime.timedelta(hours=1)).timetuple())
   end_timestamp = start_timestamp+3600 # 1 hour in the future
   return start_timestamp, end_timestamp

def array_to_csv_file (data,headings, file_name):
   import csv
   print headings
   print data[0:5]

   out = numpy.vstack ((headings, data))

   f = open(file_name, 'w+')

   writer = csv.writer(f, delimiter = ',')
   for row in out:
      writer.writerow(row)
      writer

   f.close()
```

A.5 dbControl.py

```
class dbControl:

    def test (self):
        print "In test"

    def __init__(self, host, userName, password, database):
        self.con = mdb.connect(host, userName, password, database)
        self.cur = self.con.cursor()
        self.cur.execute ("SET bulk_insert_buffer_size= 1024 * 1024 * 256")

    def closeDB (self):
        self.con.close()

    def get_hour(self, timestamp):
        new_hour = [0,0,0,0,0,0,0,0,0]
        new_hour[0:4] = datetime.datetime.fromtimestamp(timestamp).timetuple()[0:4]
        start_timestamp = time.mktime(new_hour)
        end_timestamp = time.mktime((datetime.datetime.fromtimestamp(start_timestamp) +
datetime.timedelta(hours=1)).timetuple())
        end_timestamp = start_timestamp+3600 # 1 hour in the future
        return start_timestamp, end_timestamp


    def getDumpAve (self, startTime = 0, endTime = time.time(), lowFrequency = 0,
highFrequency = 16384):
        """Returns the average power for each 1 hour dump"""
        data = self.cur.execute("SELECT spectrum.id, AVG(ave) FROM spectrum_averages
JOIN spectrum ON spectrum.id = dump_id WHERE startTime > %s AND endTime < %s
AND frequency_bin > %s AND frequency_bin < %s GROUP BY dump_id ORDER BY
dump_id",(startTime, endTime, lowFrequency, highFrequency))
        print self.cur.fetchall()

    def getFreqAve (self, startTime = 0, endTime = time.time(), lowFrequency = 0,
highFrequency = 16384):
        """Returns the average power for each frequency over all dumps in the db"""
        data = self.cur.execute("SELECT frequency_bin, AVG(ave) FROM spectrum_averages
JOIN spectrum ON spectrum.id = dump_id WHERE startTime > %s AND endTime < %s
AND frequency_bin > %s AND frequency_bin < %s GROUP BY frequency_bin ORDER
BY frequency_bin",(startTime, endTime, lowFrequency, highFrequency))
        print self.cur.fetchmany(20)

    def getFreqDumpAve (self, startTime = 0, endTime = time.time(), lowFrequency = 0,
highFrequency = 16384):
        data = self.cur.execute("SELECT spectrum.id, frequency_bin, AVG(ave) FROM
spectrum_averages JOIN spectrum ON spectrum.id = dump_id WHERE startTime > %s
AND endTime < %s AND frequency_bin > %s AND frequency_bin < %s GROUP BY
spectrum.id, frequency_bin ORDER BY spectrum.id, frequency_bin",(startTime, endTime,
lowFrequency, highFrequency))
```

```
        print self.cur.fetchmany(20)

    def insertDumpFromFile (self, fileName):
        rootFilePath = "/home/chris/dumps"
        f = h5py.File(fileName)
        data = {}
        for key in list(f):
            data[key] = list(f[key])
        attr = {}
        for key in list(f['/'].attrs):
            attr[key] = f['/'].attrs[key]

        self.insertDump(attr, data)

    def insertDumpSaveFile (self, attr, data):
        meta = self.calcMetaData(data)

        startTime = data['timestamp'][0]
        endTime = data['timestamp'][meta['n_accs'] - 1]
        localstart = time.localtime(startTime)
        path = os.path.join(self.rootFilePath, str(localstart[0]), "%02i"%localstart[1], '')
        fileName = "%02i_%02i_%02i.h5"%(localstart[2], localstart[3], localstart[4])

        self.cur.execute("INSERT INTO spectra (fileLocation, starttime, endtime, location,
direction, n_accs, n_adc_overrange, n_fft_overrange) VALUES (%s, %s, %s,
GeomFromText(\'POINT(%s %s)\'), GeomFromText(\'POINT(%s %s)\'), %s, %s,
%s)",("%s%s"%(path, fileName), startTime, endTime, 0, 0, 0, 0, meta['n_accs'],
meta['n_adc_overrange'], meta['n_fft_overrange']))

        dump_id = self.cur.lastrowid

        values = [(dump_id, f, meta['fAves'][f], meta['fMins'][f], meta['fMaxs'][f],
meta['fStdDevs'][f]) for f in range(attr['n_chans'])]
        self.cur.executemany("INSERT INTO frequency_metadata (dump_id, frequency_bin,
ave, min, max, stdDev) VALUES (%s, %s, %s, %s, %s, %s)",(values))

        values = [(dump_id, t, meta['tAves'][t], meta['tMins'][t], meta['tMaxs'][t],
meta['tStdDevs'][t]) for t in range(meta['n_accs'])]
        self.cur.executemany("INSERT INTO timestep_metadata (dump_id, time_step, ave,
min, max, stdDev) VALUES (%s, %s, %s, %s, %s, %s)",(values))

        if not os.path.exists(path):
            os.makedirs(path)

        #Create new file and save data to file
        f = h5py.File("%s%s"%(path, fileName), mode="w")
        ds = f.create_dataset('spectra', (len(data['spectra']), len(data['spectra'][0])), chunks = (4,
16384), dtype = type(data['spectra'][0][0]), compression ='gzip', compression_opts = 4)
```

```
        for key in data:           #Put data in file
            if (key == 'spectra'):
                ds = f.create_dataset('spectra', (len(data['spectra']), len(data['spectra'][0])), chunks =
(4, 16384), dtype = type(data['spectra'][0][0]), compression ='gzip', compression_opts = 4)
                ds[:] = data['spectra']
            else:
                f[key] = data[key]

        for key in attr:           #put attributes in file
            f['/'].attrs[key] = attr[key]

        f.flush()
        f.close()

        #Commit changes to database
        self.con.commit()

    def insertSystem(self, attr):
        values = (attr['n_chans'], attr['bitstream'], attr['bandwidth'], attr['fft_shift'],
attr['adc_type'],\
         attr['spectrum_bits'], attr['rf_gain'], 0, 0, 0.0)

        self.cur.execute("INSERT INTO system (n_chans, bitstream, bandwidth, fft_shift,
adc_type, spectrum_bits,\
         rf_gain, location, direction) VALUES (%s, %s, %s, %s, %s, %s, %s,
GeomFromText(\'POINT(%s %s)\'), %s)", values)

        system_id = self.cur.lastrowid
        self.con.commit()
        return system_id

    def insertDump (self, spectrum, timestamp, data, system_id, f):
        meta = self.calcMetaData(spectrum)
        #serialSpectrum = pickle.dumps(spectrum)

        localtime = time.localtime(timestamp)
        fileName = "%02i.h5"%(localtime[3]) #Filename is the hour of the observation in the
month
        path = os.path.join('/home/chris/rfi_data', str(localtime[0]), "%02i"%localtime[1],
"%02i"%localtime[2],'') #Filepath is the year followed by the month year/month/
        location = "%s%s"%(path,fileName)
        # print "adc_overrange = %s for timestamp %s"%(str(data['adc_overrange']),
str(timestamp))

        values = (system_id, timestamp, location, meta['ave'], meta['min'], meta['max'], \
            meta['stdDev'], data['adc_overrange'], data['fft_overrange'], data['adc_level'],
data['ambient_temp'], data['adc_temp'])
```

```
        self.cur.execute("INSERT INTO spectra (system_id, timestamp, fileLocation, ave_power_dBuVm, min, max, stdDev \
            , adc_overrange, fft_overrange, adc_level, ambient_temp, adc_temp) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)", values)

        index = localtime[4] * 60 + localtime[5]
        #print 'index = %i'%index

        f['spectra'][index] = spectrum

        ave_id = self.cur.lastrowid
        f.flush()
        self.con.commit()
        return ave_id

    def insertAves (self, system_id, startTime, aveOver, num_freqs):
        self.cur.execute ("SELECT AVG(ave_power_dBuVm), MIN(min), MAX(max), SUM(adc_overrange), SUM(fft_overrange),\
         MIN(timestamp), MAX(timestamp) FROM spectra WHERE timestamp >= %s AND timestamp < %s", (startTime, startTime + aveOver))

        result = self.cur.fetchone()
        self.cur.execute ("SELECT stdDev FROM spectra WHERE timestamp >= %s AND timestamp < %s", (startTime, startTime + aveOver))
        stdDevs = self.cur.fetchall()
        stdDev = 0.0

        for val in stdDevs:
            stdDev = stdDev + val[0] ** 2 * num_freqs
        stdDev /= (aveOver * num_freqs)
        values = (system_id, result[5], result[6], result[0], result[1], result[2], stdDev, result[3], result[4])

        self.cur.execute("INSERT INTO test_ave_metadata (system_id, startTime, endTime, ave_power_dBuVm, min, \
            max, stdDev, n_adc_overrange, n_fft_overrange) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)", values)

        spec_ave_id = self.cur.lastrowid

        self.cur.execute("SELECT spectrum FROM spectra WHERE timestamp >= %s AND timestamp < %s", (startTime, startTime + aveOver))
        spec = self.cur.fetchall()
        self.cur.execute("SELECT adc_overrange, fft_overrange FROM spectra WHERE timestamp >= %s AND timestamp < %s", (startTime, startTime + aveOver))
        overrange = self.cur.fetchall()
        data = []
        for bin in spec:
            data.append (pickle.loads(bin[0]))
        values = (self.calcFreqMeta(data, overrange, system_id, spec_ave_id))

        self.cur.executemany("INSERT INTO test_frequency_ave_metadata (system_id, ave_id,
frequency, ave, min, \
            max, stdDev, spectral_occupency_3sigma, spectral_occupency_6sigma) VALUES
(%s, %s, %s, %s, %s, %s, %s, %s, %s)", values)

        self.con.commit()

    def calcFreqMeta(self, data, overrange, system_id, spec_ave_id):
        ret=[]

        for j in range(len(data[0])): #for each frequency channel
            freq = []
            ave = 0.0
            mini = data[0][j]
            maxi = data[0][j]
            stdDev = 0.0
            overrangecount = 0
            for i in range (len(data)):     #Calculate median, max and min for each frequency
channel
                if(not overrange[i][0] and not overrange[i][1]):    #only if the overrange bits are
not true
                    ave = ave + data[i][j]
                    if mini > data[i][j]:
                        mini = data[i][j]
                    if maxi < data[i][j]:
                        maxi = data[i][j]
                else:
                    overrangecount = overrangecount + 1     #If the overrange bit was set, record this
            ave /= len(data) - overrangecount

            for i in range (len(data)):     #calculate the standard deviation
                if(not overrange[i][0] and not overrange[i][1]):
                    stdDev += (ave - data[i][j])**2

            stdDev = (stdDev / (len(data)-overrangecount))**0.5

            #calculate the spectral occupency
            sigma3 = ave + 3 * stdDev
            sigma6 = ave + 6 * stdDev
            sigma3Count = 0
            sigma6Count = 0
            for i in range (len(data)): #for each time step
                if(not overrange[i][0] and not overrange[i][1]):
                    if (data[i][j] > sigma6):
                        sigma3Count+=1
                        sigma6Count+=1
                    elif (data[i][j] > sigma3):
                        sigma3Count+=1
            sigma3Occ = sigma3Count / (len(data) - overrangecount)
            sigma6Occ = sigma6Count / (len(data) - overrangecount)

            freq.append(system_id)
            freq.append(spec_ave_id)
            freq.append(j * 0.054931640625)
            freq.append(ave)
            freq.append(mini)
            freq.append(maxi)
            freq.append(stdDev)
            freq.append(sigma3Occ)
            freq.append(sigma6Occ)
            ret.append(freq)
        return ret


    def calcMetaData(self, data):
        ret = {}

        ave = 0.0
        mini = data[0]
        maxi = data[0]
        stdDev = 0.0
        length = len(data)

        for val in data:
            ave += val
            if mini > val:
                mini = val
            if maxi < val:
                maxi = val
        ave /= length

        for val in data:
            stdDev += (ave - val)**2
        stdDev = (stdDev / length)**0.5

        ret['ave'] = ave
        ret['min'] = mini
        ret['max'] = maxi
        ret['stdDev'] = stdDev

        return ret



    def getCurrentSpectrum(self):
        self.cur.execute("SELECT MAX(id) FROM spectra")
        result = self.cur.fetchone()[0]
        return self.getSpectrum(result)
    def getAllSpectraToCSV(self, fileName):
        import csv
        self.cur.execute("SELECT spectrum, timestamp from spectra")
        result = self.cur.fetchone()
        headings = [str(50 + i * 0.054931640625) + "Mhz" for i in range (16384)]
        headings.insert(0,"Timestamp")

        writer = csv.writer(open(fileName, 'wb'))
        writer.writerow(headings)

        while result is not None:
            t = time.localtime(result[1])
            spec = pickle.loads(result[0])
            data = ['%02i:%02i:%02i on %02i/%02i/%04i'%(t[3], t[4], t[5], t[2], t[1], t[0])]
            data.extend(spec)
            writer.writerow(data)
            result = self.cur.fetchone()



    def calcAveMetaData(self, data):
        """Calculate the means, mins, maxs and standard deviations of each row and column of
data."""
        ret = {}
        aves = []
        mins = []
        maxs = []
        stdDevs = []
        n_adc_overrange = 0;
        n_fft_overrange = 0;
        ignoreLast = True
        xLen = len(data['spectra'])
        yLen = len(data['spectra'][0])


        #Check if last row contains actual data
        for i in range (yLen):
            if (data['spectra'][xLen - 1][i] != 0.0):
                ignoreLast = False
        if (ignoreLast):
            xLen -= 1

        #calculate along x axis
        for t in range (xLen):
            if (not data['adc_shutdown'][t]):
                mini = data['spectra'][t][0]
                maxi = mini
                ave = 0.0
                for f in range (yLen):
```

```
            ave += data['spectra'][t][f]
            if (data['spectra'][t][f] < mini):
                mini = data['spectra'][t][f]
            if (data['spectra'][t][f] > maxi):
                maxi = data['spectra'][t][f]
        ave /= yLen
        aves.append(ave)
        mins.append(mini)
        maxs.append(maxi)
    else:
        if (data['adc_overrange'][t]):
            n_adc_overrange += 1
        if (data['fft_overrange'][t]):
            n_fft_overrange += 1
for t in range (xLen):
    if (not data['adc_shutdown'][t]):
        stdDev = 0.0
        for f in range (yLen):
            stdDev += (aves[t] - data['spectra'][t][f]) ** 2
        stdDev = math.sqrt(stdDev / yLen)
        stdDevs.append(stdDev)

ret['tAves'] = aves
ret['tMins'] = mins
ret['tMaxs'] = maxs
ret['tStdDevs'] = stdDevs

aves = []
mins = []
maxs = []
stdDevs = []

#calculate along y axis
for f in range (yLen):
    mini = data['spectra'][0][f]
    maxi = mini
    ave = 0.0
    for t in range (xLen):
        if (not data['adc_shutdown'][t]):
            ave += data['spectra'][t][f]
            if (data['spectra'][t][f] < mini):
                mini = data['spectra'][t][f]
            if (data['spectra'][t][f] > maxi):
                maxi = data['spectra'][t][f]
    ave /= (yLen - n_adc_overrange - n_fft_overrange)
    aves.append(ave)
    mins.append(mini)
    maxs.append(maxi)
for f in range (yLen):
    stdDev = 0.0
```

```
            for t in range (xLen):
                if (not data['adc_shutdown'][t]):
                    stdDev += (aves[t] - data['spectra'][t][f]) ** 2
            stdDev = math.sqrt(stdDev / yLen)
            stdDevs.append(stdDev)

        ret['fAves'] = aves
        ret['fMins'] = mins
        ret['fMaxs'] = maxs
        ret['fStdDevs'] = stdDevs
        ret['n_adc_overrange'] = n_adc_overrange
        ret['n_fft_overrange'] = n_fft_overrange
        ret['n_accs'] = xLen

        return ret

    def get_std_dev(self, startTime, channel):
        self.cur.execute("SELECT datum.value from datum inner join spectra on
datum.spectra_id = spectra.id where timestamp = %s and type = %s and element_id = %s",
(startTime, 'std', channel))
        result = self.cur.fetchone()[0]
        return result

    def get_archive (self, startTime, endTime, type):
        self.cur.execute("SELECT datum.value from datum inner join spectra on
datum.spectra_id = spectra.id where timestamp >= %s and timestamp < %s and type = %s",
(startTime, endTime, type))
        result = numpy.reshape(numpy.array(self.cur.fetchall()), (14200))
        return result

    def rfi_archive_get_period (self, startTime, endTime, type, frequency):
        import ratty1
        rat = ratty1.cam.spec()

        channel = rat.cal.freq_to_chan(frequency)

        self.cur.execute ("SELECT id FROM element WHERE channel = %s", (channel))
        elId = self.cur.fetchone()[0]
        print "elID"
        print elId
        self.cur.execute ("SELECT id FROM spectra WHERE timestamp >= %s and timestamp
<= %s", (startTime, endTime))
        spectraIDs = numpy.array(self.cur.fetchall())[:,0]
        maxID = numpy.max(spectraIDs)
        minID = numpy.min(spectraIDs)
        print minID
        print maxID
        self.cur.execute ("SELECT datum.value FROM datum FORCE INDEX (spectra_id)
WHERE datum.element_id = %s AND spectra_id >= %s AND spectra_id <= %s AND type
= %s",
                          (elId, minID, maxID, type))
        result = numpy.array(self.cur.fetchall())
        return result

    def frequency_to_channel(self, frequency):
        import ratty1
        rat = ratty1.cam.spec()
        print "converting %fMHz and %f MHz to channel %i and %i"%(frequency *
1000000, rat.cal.config['ignore_low_freq'], rat.cal.freq_to_chan(frequency),
rat.cal.freq_to_chan(rat.cal.config['ignore_low_freq']))
        return rat.cal.freq_to_chan(frequency * 1000000) -
rat.cal.freq_to_chan(rat.cal.config['ignore_low_freq'])

    #enter unix timestamps for startTime and endTime, enter a particular channel number if
you would like only 1 channel, enter a tuple channelRange = (lowchannel, highchannel)
    #if you would like a range of channels
    def rfi_monitor_get_range(self, startTime, endTime, channel = -1, channelRange =
(0,14200)):

        self.cur.execute ("SELECT DISTINCT fileLocation FROM spectra WHERE timestamp
>= %s AND timestamp < %s", (startTime, endTime))
        res = self.cur.fetchall()

        startTuple = datetime.datetime.fromtimestamp(startTime)
        endTuple = datetime.datetime.fromtimestamp(endTime)

        print startTuple
        print endTuple

        now = datetime.datetime.now()
        lastHour = datetime.datetime(now.year,now.month,now.day,now.hour)
        nHours = None

        nHours = (endTuple.day - startTuple.day) * 24  + endTuple.hour - startTuple.hour

        print "startTuple"
        print startTuple

        files = [False for i in range (nHours)]

        fileName = "%02i.h5"%(startTuple.hour) #Filename is the hour of the observation
        path = os.path.join('/home/chris/rfi_data', str(startTuple.year),
"%02i"%startTuple.month, "%02i"%startTuple.day,'') #Filepath year/month/day of the
observation
        location = "%s%s"%(path,fileName)

        ret = None

        nSecs = startTuple.minute*60 + startTuple.second
    nSpectra = (nHours) * 3600 - nSecs + endTuple.minute*60 + endTuple.second

    if (channel != -1): #only want one channel
      ret = numpy.zeros((nSpectra,), dtype=numpy.float64)
    else:
      ret = numpy.zeros((nSpectra,channelRange[1] - channelRange[0]),
dtype=numpy.float64)

    if os.path.isfile(location):
      files[0] = True
      try:
        f = h5py.File(location, 'r')
        print f['spectra'].shape
        if (channel != -1):
          ret[0:3600 - nSecs] = f['spectra'][nSecs:3600,channel]
        else:
          ret[0:3600 - nSecs] = f['spectra'][nSecs:3600,channelRange[0]:channelRange[1]]
        nSecs = 3600 - nSecs
      except IOError as e:
        print e

    currentTime = startTuple + datetime.timedelta(hours=1)

    current = 0

    print "endtuple = %s, lastHour = %s"%(str(endTuple),str(lastHour))
    print "endTuple > lastHour?"
    print (endTuple > lastHour)

    if endTuple > lastHour:
      current = 1

    for i in range(nHours - 1):
      fileName = "%02i.h5"%(currentTime.hour) #Filename is the hour of the observation
      path = os.path.join('/home/chris/rfi_data', str(currentTime.year),
"%02i"%currentTime.month, "%02i"%currentTime.day,'') #Filepath year/month/day of the
observation
      location = "%s%s"%(path,fileName)
      print location
      print nSecs
      try:
        f = h5py.File(location, 'r')
        if (channel != -1):
          print "f['spectra'].shape"
          print f['spectra'].shape
          ret[nSecs:nSecs + 3600] = f['spectra'][:,channel]
        else:
          ret[nSecs:nSecs + 3600] = f['spectra'][:,channelRange[0]:channelRange[1]]
        nSecs += 3600
      except IOError as e:
```

```
                print e
                print location
                nSecs += 3600

            currentTime = currentTime + datetime.timedelta(hours=1)

        print "nSPectra = %s"%nSpectra
        print "nSecs = %s"%nSecs

        if current == 1:
            print "IN current"
            import rfDB.current_spectra as current_spectra
            curr = current_spectra.current_spectra();
            print ("Getting %i mintues and %i seconds = %i seconds"%(endTuple.minute,
endTuple.second, endTuple.minute * endTuple.second))
            spectra, times = curr.getRange(endTuple.minute * 60 + endTuple.second)
            print spectra.shape
            if (channel != -1):
                ret[nSecs - 1:] = spectra[:,channel]
            else:
                ret[nSecs:] = spectra[:]

            nSecs += spectra.size
            curr.close()

        #Interpolate zeroes
        z = numpy.where(ret==0.0)[0]
        nz = numpy.where (ret!=0.0)[0]
        ret[ret==0.0]=numpy.interp(z,nz,ret[nz])

        z = numpy.where(ret > (100))[0]
        nz = numpy.where(ret < (100))[0]
        print z[0:5]
        print nz[0:5]
        ret[ret > 10 ** 20]=numpy.interp(z,nz,ret[nz])

        print "check"
        print "nSPectra = %s"%nSpectra
        print "nSecs = %s"%nSecs
        print ret.shape
        print "max = %f"%numpy.max(ret)

        return ret


    def replace_overrange_with_int (self, data, over_pos, rep_int):
        ret = numpy.array(data)
        ret[over_pos] = rep_int
        return ret
    def get_last_ave_timestamp(self):
        self.cur.execute("SELECT MAX(timestamp) from spectra")
        result = self.cur.fetchone()
        return result[0]

    def get_ave_archive (self, type):
        self.cur.execute("SELECT element.frequency, AVG(datum.value) FROM datum INNER JOIN element ON datum.element_id = element.id WHERE type = %s GROUP BY element.frequency ", (type))
        result = numpy.array(self.cur.fetchall()[:])[:,1]
        print result.shape
        return result

    def archive_get_val_at_time(self, time, typ):
        print "IN THE METHOD"
        starttime = self.get_last_ave_timestamp()
        print starttime
        self.cur.execute("SELECT id FROM spectra WHERE timestamp = %s",(starttime))
        result = self.cur.fetchone()
        self.cur.execute("SELECT datum.value, element.frequency FROM datum, element WHERE spectra_id = %s AND type = %s AND element.id = datum.element_id GROUP BY element.channel", (result[0], typ))
        result = numpy.array(self.cur.fetchall())
        result[:,1] = result[:,1]/1000000
        result[:,1] = numpy.around(result[:,1], decimals = 2)
        return result

    def archive_get_frequency_list(self):
        import ratty1
        rat = ratty1.cam.spec()
        low_frequency = rat.cal.config['ignore_low_freq']
        high_frequency = rat.cal.config['ignore_high_freq']
        print low_frequency
        print high_frequency
        self.cur.execute ("SELECT element.frequency FROM element WHERE element.frequency > %s AND element.frequency < %s",(low_frequency, high_frequency))
        result = numpy.array(self.cur.fetchall(), dtype=numpy.float32)
        result[:,0] = result[:,0]/1000000
        result[:,0] = numpy.around(result[:,0], decimals = 2)
        return result [:,0]

    def rfimonitor_get_adc_overrange(self, starttime, endtime):
        self.cur.execute("SELECT timestamp FROM spectra WHERE timestamp >= %s AND timestamp < %s AND adc_overrange = 1",(starttime, endtime))
        result = self.cur.fetchall()
        return result

    def rfimonitor_get_fft_overrange(self, starttime, endtime):
        self.cur.execute("SELECT timestamp FROM spectra WHERE timestamp >= %s AND timestamp < %s AND fft_overrange = 1",(starttime, endtime))
        result = self.cur.fetchall()
        print "FFT OVERRANGES"
        print len(result)
        return result

    def rfi_monitor_get_adc_overrange_pos (self, starttime, endtime):
        self.cur.execute("SELECT timestamp FROM spectra WHERE timestamp >= %s AND
timestamp < %s AND adc_overrange = 1",(starttime, endtime))
        result = self.cur.fetchall()
        ret = [int(t[0] - starttime) for t in result]
        return ret

    def rfi_monitor_get_oldest_timestamp(self):
        self.cur.execute("SELECT min(timestamp) FROM spectra")
        result=self.cur.fetchone()
        return result[0]

    #----------------------------------------------------------------------------------------------------
------------------------------------------------------------------

    def toCompressed (self):
        self.cur.execute ("SELECT MIN(id) from spectra")
        result = self.cur.fetchone()
        mini = result[0];
        print "min = %i"%mini

        last = mini + 800000;

        for i in range(mini + 4000, last, 4000):
            self.cur.execute("INSERT spectraCompressed SELECT * FROM spectra WHERE id
< %s", i)
            self.cur.execute("DELETE FROM spectra WHERE id < %s", i)
            self.con.commit()
            print "deleted up to %i"%i

    def testAves(self, startTime, aveOver):
        self.cur.execute ("SELECT spectrum from spectra where timestamp >= %s and
timestamp < %s", (startTime, startTime + aveOver));
        result = self.cur.fetchall()
        ave = pickle.loads(result[0][0])
        spectrum = pickle.loads(result[0][0])

        print ave[67]

        for i in range(1,len(result)):
            spectrum = pickle.loads(result[i][0])
            for j in range(len(spectrum)):
                if (j == 67):
                    print "ave%i = %f + %f = %f"%(i, ave[j], spectrum[j], ave[j] + spectrum[j])
                ave[j] = ave[j] + spectrum[j]
        for j in range(len(spectrum)):
            if j == 67:
                print "ave%i = %f / %i"%(j, ave[j], len(result))
            ave[j] = ave[j] / len(result)

        print "equals %f"%ave[67]


        spectrum = ave
        self.cur.execute("SELECT max(ave_id) from test_frequency_ave_metadata")
        lastave = self.cur.fetchone()[0]
        print "lastave is %i"%lastave
        self.cur.execute("SELECT ave from test_frequency_ave_metadata where ave_id = %s ORDER BY frequency", lastave)
        result = self.cur.fetchall();
        good = True;

        print "not equal %f"%result[67][0]

        print (len(spectrum))
        print (len(result))
        print (len(result[0]))

        for i in range (len(spectrum)):
            if float(ave[i]) != float(result[i][0]):
                good = False;
                #print "spectrum%i is %f, ave%i is %f"%(i,float(ave[i]),i, float(result[i][0]))

        print good

    """Delete all spectra in rfimonitor db with timestamp <= timestamp"""
    def delete_spectra (self, timestamp):
        self.cur.execute("SELECT timestamp FROM rfimonitor.spectra WHERE timestamp <= %s ORDER BY timestamp",timestamp)
        res = self.cur.fetchall()
        location = "nothing"
        # print res[0:20]
        times = self.remove_duplicates(res)
        # print times
        for t in times:
            try:
                localtime = time.localtime(t)
                fileName = "%02i.h5"%(localtime[3]) #Filename is the hour of the observation in the month
                path = os.path.join('/home/chris/rfi_data', str(localtime[0]), "%02i"%localtime[1], "%02i"%localtime[2],'') #Filepath is the year followed by the month year/month/
                location = "%s%s"%(path,fileName)
                #d = datetime.datetime()
                # print "timestamp = %i"%t
                remove =
time.mktime((localtime[0],localtime[1],localtime[2],localtime[3]+1,0,0,0,0,0))
                # print "remove = %i"%remove
                # print "diff = %i"%(remove - t)
                self.cur.execute("DELETE FROM rfimonitor.spectra where timestamp < %s",
remove)
                os.remove(location)
                print "deleted %s"%location
                if localtime[3] == 0:
                    os.rmdir(path)
                if localtime[2] == 0:
                    os.rmdir(os.path.join('/home/chris/rfi_data', str(localtime[0]),
"%02i"%localtime[1],""))

            except OSError:
                print "couldn't delete file %s"%location
                pass
        self.con.commit()

    def getSpectrum(self, timestamp):
        spectra = numpy.zeros(shape=(14200))
        try:
            localtime = time.localtime(timestamp)
            fileName = "%02i.h5"%(localtime[3]) #Filename is the hour of the observation in the
month
            path = os.path.join('/home/chris/rfi_data', str(localtime[0]), "%02i"%localtime[1],
"%02i"%localtime[2],") #Filepath is the year followed by the month year/month/
            location = "%s%s"%(path,fileName)
            hour_start =
time.mktime((localtime[0],localtime[1],localtime[2],localtime[3],0,0,0,0,0))
            f = h5py.File(location, mode='r')
            spectra = f['spectra'][timestamp - hour_start]
        except OSError:
                print "couldn't open file %s"%location
                pass
        return spectra

    def remove_duplicates(self, timestamps):
        ret = [timestamps[0][0]]
        for t in timestamps:
            if t[0] - ret[-1] >= 3600:
                ret.append(t[0])
        return ret

    def maintain_space(self):
        while self.check_space() < 0.05:
            print "Freeing space"
            self.cur.execute("select min(timestamp) from spectra")
            res = self.cur.fetchone()
            self.delete_spectra(res[0])



    def check_space (self):
        """Return percent of the harddrive is free"""
        s = os.statvfs("/")
        space = float(s.f_bavail)/s.f_blocks
        print "%f of harddrive free"
        return float(s.f_bavail)/s.f_blocks
```

A.6 rfi_event.py

```
import numpy as np
import time
from PIL import Image
import mahotas
import pickle
import MySQLdb
import archive_conf as cnf
import h5py

def threshold (data, devs, means, startTime):
    sT = time.time()
    lx = len(data)
    ly = len(data[0])
    rfi_mask = np.array([[False for i in range(ly)] for j in range(lx)])

    rfi_events = [{} for i in range(ly)]
    rfi_stats = [{'n_rfi_detections':0,'startTimes':[], 'endTimes':[]} for i in range(ly)]
    rfi_start = []
    rfi_end = []
    rfi_chan = []
    temp = 0;
    temp2 = 0;

    print "data"
    print data

    for t in range(1,lx):

        tS = time.time()
        #rfi_stats[i]['n_rfi_detections'] = 0
        #rfi_stats[i]['startTimes'] = [0]
        #rfi_stats[i]['duration'] = [0]
        for c in range (ly):
            if (data[t][c] > (means[c][0] + 3 * devs[c][0])):
                rfi_stats[c]['n_rfi_detections'] += 1
                if (not rfi_mask[t-1][c]):
                    rfi_stats[c]['startTimes'].append(startTime + t)

                    rfi_chan.append(c)
                rfi_mask[t][c] = True;
            elif (rfi_mask[t-1][c]):
                rfi_stats[c]['endTimes'].append(startTime + t)

            rfi_start.append(rfi_stats[c]['startTimes'])
            rfi_end.append(rfi_stats[c]['endTimes'])

    im = Image.new("1", (len(rfi_mask),len(rfi_mask[0])), 1)
    print ("img height = %i\nimg width = %i"%(len(rfi_mask), len(rfi_mask[0])))
    print ("length of bitmask = %i"%len(rfi_mask.flatten()))
    im.putdata(rfi_mask.flatten())
    im.save("/home/chris/rfi_monitor/database/src/RFI_mask.gif")

    tS = time.time()
    events = extractEvents(rfi_mask, data, startTime, "threshold_")
    print "event extraction took %i seconds"%(time.time() - tS)

    for key in events[0].keys():
        print "%s = "%key + str(events[0][key])

    print "%i events extracted"%len(events)

    tS = time.time()
    stats = []
    for i in range(len(events)):
        stats.append (calc_statistics(events[i]))

    print "event stats took %i seconds"%(time.time() - tS)
    print "timestamp = %i"%startTime

    print "stats[0] :"
    print stats[0]

    eT = time.time()
    print "Took %i seconds"%(eT - sT)
    return rfi_stats


def extractEvents (mask, data, startTime, file):
    structuring = np.array([[1,1,1],[1,1,1],[1,1,1]]) #Connectivity structure
    timet = time.time()
    labeled, nr = mahotas.label(mask, Bc = structuring)
    print "labeling took %i"%(time.time() - timet)

mahotas.imsave("/home/chris/Dropbox/rfi_monitor/database/src/testIm/%smask.tif"%file,ma
sk)
```

```
mahotas.imsave("/home/chris/Dropbox/rfi_monitor/database/src/testIm/%slabeled.tif"%file,l
abeled)
   events = [{} for i in range (nr)]
   for i in range (nr):
      indices = np.where(labeled == (i+1))
      minT = indices[0].min()
      maxT = indices[0].max() + 1
      minC = indices[1].min()
      maxC = indices[1].max() + 1
      events[i]['startTime'] = startTime + minT
      events[i]['endTime'] = startTime + maxT
      events[i]['lowChannel'] = minC
      events[i]['highChannel'] = maxC
      events[i]['data'] = data[minT:maxT,minC:maxC]
      events[i]['mask'] = mask[minT:maxT,minC:maxC]

   print "number of events = %i"%nr

   return events

def insert_event (rfi_event):
   db_connect = MySQLdb.connect(host=cnf.host, port=cnf.port, user=cnf.user,
passwd=cnf.passwd, db="rfi_event_archive")
   c_rfi = db_connect.cursor()

   c_rfi.execute("INSERT INTO rfi_event (system_id, startTime, endTime, low_chan,
high_chan, spectra, mask) \
         VALUES (%s,%s,%s,%s,%s,%s,%s)", (1, rfi_event['startTime'],
rfi_event['endTime'], rfi_event["lowChannel"], rfi_event["highChannel"], \
         pickle.dumps(rfi_event["data"]), pickle.dumps(rfi_event["mask"])))
   db_connect.commit()

   c_rfi.close()

def insert_to_archive(timestamp, num_events, len_events):
   import datetime

   db_connect = MySQLdb.connect(host=cnf.host, port=cnf.port, user=cnf.user,
passwd=cnf.passwd, db="rfi_archive")
   c_rfi = db_connect.cursor()

   print "timestamp = "
   print timestamp

   new_hour = [0,0,0,0,0,0,0,0,0]
   new_hour[0:4] = datetime.datetime.fromtimestamp(timestamp).timetuple()[0:4]
   timestamp = time.mktime(new_hour)
```

```
    c_rfi.execute('SELECT id FROM rfi_archive.spectra WHERE system_id=%d AND
timestamp=%d' % (1, timestamp))
    spectra_id = c_rfi.fetchone()[0]
    print "result = "
    print spectra_id

    c_rfi.execute('SELECT n_chans,bandwidth FROM rfi_archive.system WHERE id=%d;'
% (1))
    [n_chans, bandwidth_hz] = c_rfi.fetchone()
    # Only process the calibrated data between the lower and upper frequency limits
    low_channel = cnf.freq2chan(cnf.IGNORE_LOW_FREQ, bandwidth_hz, n_chans)
    high_channel = cnf.freq2chan(cnf.IGNORE_HIGH_FREQ, bandwidth_hz, n_chans)

    print "low_channel = %i high_channel = %i"%(low_channel, high_channel)

    c_rfi.execute('SELECT id FROM rfi_archive.element WHERE channel >= %s AND
channel < %s ORDER BY channel',(low_channel,high_channel))
    element_ids = c_rfi.fetchall()
    print len(element_ids)
    print len(num_events)
    print len (len_events)

    insert_num_events = [(num_events[i], spectra_id, element_ids[i][0],'num_events') for i in
range (len(element_ids))]
    insert_len_events = [(len_events[i], spectra_id, element_ids[i][0],'num_events') for i in
range (len(element_ids))]

    c_rfi.executemany('INSERT INTO rfi_archive.datum(value,spectra_id,element_id,type)
VALUES (%s,%s,%s,%s)',insert_num_events)
    c_rfi.executemany('INSERT INTO rfi_archive.datum(value,spectra_id,element_id,type)
VALUES (%s,%s,%s,%s)',insert_len_events)

    db_connect.commit()

    c_rfi.close()

def getEvents(data, mask, startTime):

    num_events, len_events = mahotas_check(mask)
    structuring = np.array([[1,1,1],[1,1,1],[1,1,1]]) #Connectivity structure
    label, n_events = mahotas.label(mask, Bc = structuring)
    print "n_events = %s"%n_events
    indices = np.nonzero(label)
    rfi_groups = label[indices]
    index = np.argsort(rfi_groups)
    sorted_groups = rfi_groups[index]
    sorted_index = np.searchsorted(sorted_groups, np.arange(1,n_events+1))
    rfi_event_indices = []
    rfi_event_indices.append(np.split(indices[0][index], sorted_index))
    rfi_event_indices.append(np.split(indices[1][index], sorted_index))
    events = np.empty(n_events,
dtype={'names':['startTime','endTime','lowChannel','highChannel','data','mask'],\
                                        'formats':['i4','i4','i4','i4', 'object', 'object']})
    # print"rfi_event_indices"
    # print rfi_event_indices

    for i in range(n_events):
        # print "i = %i"%i
        # print rfi_event_indices[0][i]
        events[i]['startTime'] = startTime + np.min(rfi_event_indices[0][i+1])
        events[i]['endTime'] = startTime + np.max(rfi_event_indices[0][i+1]) + 1
        events[i]['lowChannel'] = np.min(rfi_event_indices[1][i+1])
        events[i]['highChannel'] = np.max(rfi_event_indices[1][i+1]) + 1
        # print "data[%i:%i,%i:%i]"%(events[i]['startTime']-startTime,events[i]['endTime']-
startTime,events[i]['lowChannel'],events[i]['highChannel'])
        # print "type(numpy.asscalar(events[i]['startTime'])) = "
        # print numpy.asscalar(events[i]['startTime'])
        sT = int(np.asscalar(events[i]['startTime']-startTime))
        eT = int(np.asscalar(events[i]['endTime']-startTime))
        lC = int(np.asscalar(events[i]['lowChannel']))
        hC = int(np.asscalar(events[i]['highChannel']))
        # print data[events[i]['startTime']-startTime:events[i]['endTime']-
startTime,events[i]['lowChannel']:events[i]['highChannel']]
        events[i]['data'] = data[sT:eT,lC:hC]
        events[i]['mask'] = mask[sT:eT,lC:hC]

    return events, num_events, len_events


    index_array = [np.arange(sorted_index[i], sorted_index[i+1]) for i in
np.arange(len(sorted_index)-1)]
    event_indices = np.array([(indices[0][index][index_array[i]],
indices[1][index][index_array[i]]) for i in np.arange(len(index_array))])
    events =
[data[np.min(event_indices[i][0]):np.max(event_indices[i][0])+1,np.min(event_indices[i][1]):
np.max(event_indices[i][1])+1] for i in np.arange(len(event_indices))]




def get_rfi(data,sigma=4):
    """ Get the rfi for the middle of the window
    Data is assumed to be a 2D array
    and sigma is the standard deviation that is used"""
    mid = data.shape[0]//2+1   # mid point of window
    med = np.median(data[:,:])
    # print "med = "
    # print med
    mad = np.median(np.abs(data[:] - med),axis=0)
    # print "mad = "
    # print mad
    mad_limit = sigma/1.4826 # see relation to standard deviation
    return (data[mid]  > mad_limit*mad + med ) + (data[mid]  < -mad_limit*mad + med )

def median_filter (data, window=10, sigma = 4):
    rfi = np.zeros(data.shape)
    for t in range(window,data.shape[0]-window) :  rfi[t,:] = get_rfi(data[t-
window:t+window+1])
    return rfi

def count_rfi_events (mask):
    diff = numpy.diff(mask, axis = 0)
    count = numpy.sum (diff, axis = 0)
    n_events = numpy.floor((count+1)/2)
    print "len(n_events) = %i"%len(n_events)
    check = numpy.where(n_events==0)
    n_events[check] = numpy.logical_or(n_events[check],mask[:,check][0])
    return n_events

def mahotas_check(mask):
    structure = np.array([1,1,1])
    num_events = np.zeros(len(mask[0]))
    len_events = np.zeros(len(mask[0]))
    for i in range(len(mask[0])):
        label, num_events[i] = mahotas.label(mask[:,i], Bc=structure)
        len_events[i] = np.count_nonzero(label)



    return num_events, len_events

def equalarr (arr1, arr2):
    for i in range(len(arr1)):
        if (arr1[i] != arr2[i]):
            print "arr1[%i] = %f != %f = arr2[%i]"%(i,arr1[i],arr2[i],i)


def calc_statistics (rfi_event, stats):
    func_list = ['min', 'max', 'median', 'mean', 'std']
    masked = np.ma.array(rfi_event['data'], mask = np.logical_not(rfi_event['mask']))
    for func in func_list:
        stats[func] = getattr(np, func)(masked, axis=0)

    #one_perc = int(numpy.round(len(masked)/100))

def rfi_detection (timestamp, window = 10):
    t = time.localtime(timestamp)
    filestr1 = "/home/chris/rfi_data/%i/%02i/%02i/%02i.h5"%(t[0],t[1],t[2],t[3] - 1)
    filestr2 = "/home/chris/rfi_data/%i/%02i/%02i/%02i.h5"%(t[0],t[1],t[2],t[3])
    print "filestr1 = %s"%filestr1
    print "filestr2 = %s"%filestr2
    file1 = h5py.File(filestr1, 'r')
    file2 = h5py.File(filestr2, 'r')
    print file1
    print file1.items()
    data1 = file1['spectra']
    data2 = file2['spectra']
    data = np.concatenate((data1[(-window*2): -1],data2), axis=0)
    print "len(data1) = %i, len(data2) = %i, len(data) = %i"%(len(data1), len(data2),
len(data))
    mask = median_filter(data, window = window)
    events, num_events, len_events = getEvents(data, mask, timestamp - 2*window)
    # stats = np.empty(len(events), dtype={'names':['min', 'max', 'median', 'mean', 'std'],\
    #                                     'formats':['f8','f8','f8','f8','f8']})
    for i in range (len(events)):
        # calc_statistics(events[i], stats[i])
        insert_event(events[i])


    file1.close()
    file2.close()

    return num_events, len_events
```

Appendix B: RFI measurements

Table of frequencies and settings for mode 2 measurements required by SKA

Frequency Band MHz	RBW kHz	ΔT_{RBW} ms	Time s	Radio Astronomy Usage
150-153	1	100	300	Continuum
153-322	3	10	564	
322-329	3	1000	2334	Deuterium (DI)
329-406	30	10	26	
406-410	30	10000	1334	Continuum
410-608	30	10	66	
608-614	30	10000	2000	Continuum
614-1000	30	10	129	
1000-1370	30	300	3700	Continuum
1370-1427	30	1000	1900	Hydrogen (HI), SETI
1427-1606	30	100	597	SETI
1606-1723	30	1000	3900	Hydroxyl (OH), SETI
1723-2655	30	10	311	
2655-2700	100	1000	450	Continuum
2700-3300	100	10	60	
3300-3400	100	1000	1000	Methyladyne (CH)
3400-4800	100	10	140	
4800-5000	100	1000	2000	Formaldehyde (H2CO)
5000-6600	300	10	54	
6600-6700	300	1000	334	Methanol (CH3OH)
6700-8600	300	10	64	
8600-8700	300	1000	334	Helium (3He+)
8700-12100	300	10	114	
12100-12200	300	1000	334	Methanol (CH3OH)
12200-14400	300	10	127	
14400-14500	300	1000	334	Formaldehyde (H2CO)
14500-18300	300	10	127	
18300-18400	300	1000	334	Cyclopropenylidene (C3H2)
18400-22000	300	10	120	
Total			6.4 Hours	