Crowdsourcing GNSS Jamming Detection and Localization

Luka Strizic

Space Engineering, master's level (120 credits)
2017

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

Crowdsourcing GNSS Jamming Detection and Localization

Luka Strizic

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

12 November 2017

Abstract

Global Navigation Satellite Systems (GNSS) have found wide adoption in various applications, be they military, civilian or commercial. The susceptibility of GNSS to radio-frequency interference can, thus, be very disruptive, even for emergency services, therefore threatening people's lives. An early prototype of a system providing relatively cheap widescale GNSS jamming detection, called J911, is explored in this thesis.

J911 is smartphone-based crowdsourcing of GNSS observations, most interesting of which are carrier-to-noise-density ratio ($\frac{C}{N_0}$) and Automatic Gain Control (AGC) voltage. To implement the prototype, an Android application to provide the measurements, a backend to parse and store the measurements, and a frontend to visualize the measurements were developed. In real-world use, the thesis argues, the J911 system would best be implemented over existing Enhanced 9-1-1 (E911) infrastructure, becoming a standardized part of the Public Switched Telephone Network (PSTN).

The Android application, running on a smartphone, would periodically construct messages to be sent to the backend over an Internet connection. The messages would include: current location from all location providers available in Android OS, observed satellites from all supported constellations, the satellites' $\frac{C}{N_0}$, and a timestamp. Once a message is received on the backend, the data would be extracted and stored in a database. The frontend would query the database and produce a map with the collected datapoints overlaid on top of it, whose color indicates received signal strength at that point. When a jammer gets close enough to a few smartphones, they will all be jammed, which is easily observed on the map. On top of that, if enough samples are gathered, a Power Difference of Arrival localization algorithm can be used to localize the jammer.

The smartphones that the system was planned to be tested with did not support AGC level readings, therefore in order to obtain AGC levels over time, a few SiGe GN3S Samplers, which are radio-frequency frontends, were used. In eastern Idaho, United States, over three nights in July 2017, an exercise, named 2017 DHS JamX, was performed with the help of the US Department of Homeland Security. Sadly, the approval for the publication of the test results did not come in time to be included in this thesis.

Acknowledgement

First of all, I would like to thank the SpaceMaster Consortium, the SpaceMaster students and the staff, for providing an enjoyable and thriving environment to study in. There is not much opportunity to study space science and engineering in Croatia, thus being accepted to such a program was a blessing.

A big thanks goes to the University of Colorado Boulder, Colorado Center for Astrodynamics Research, Dr. Nagaraj Channarayapatna Shivaramaiah and, most importantly, to my supervisor, Professor Dennis M. Akos, for hosting me for six months and providing me a considerable amount of guidance and support during the work on this thesis and beyond.

Special thanks to my examiner, Professor Mathias Milz, and Paul Sabatier University for tolerating the unfortunate delays that have befallen the results of this thesis.

The University of Zagreb and all the professors and assistants, whose courses I took, and the colleagues, with whom I have worked, deserve many thanks for accepting, understanding and enduring my absence through the past two years.

Last, but not least, I am grateful to have a loving and supporting family that encouraged me through my two years of studies and work abroad.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

In November 2009, a Global Positioning System (GPS) Ground Based Augmentation System (GBAS) was installed at Newark Liberty International Airport and immediately began experiencing seemingly irregular outages. The GBAS, which is used to provide navigation and high precision services to incoming and outgoing aircraft, was rendered useless due to radio-frequency interference from unknown sources. Outage analysis indicated a localized event and cast suspicion on vehicles on a nearby freeway. Several-months-long efforts were initiated to find the offending vehicles and their users. Advanced interference detection equipment and surveillance cameras were used to catch a GPS jammer user, who turned out to be a truck driver attempting to prevent GPS-based vehicle tracking. The truck driver had in his possession a broadly available \$33 200mW GPS jammer, easily procured on the Internet. In order to avoid such incidents in the future, the Federal Aviation Administration (FAA) relocated the GBAS further away from the freeway to a better protected location. [8]

Services derived from Global Navigation Satellite Systems (GNSS) are widely used in many critical military and civil applications. Hence, the susceptibility of GNSS to interference can potentially result in inconvenient, costly and even life-threatening conditions. This is a problem as GNSS jammers, while illegal, can still be easily and cheaply procured. While there are regulations that seek to prevent GNSS interference, they are difficult to enforce. Enforcement requires detection and localization. However, detection and localization currently requires expensive signal processing equipment and manpower [8] [9]. Resources dedicated to GNSS interference detection are few and far between. Non-dedicated resources may not distinguish interference from natural or physical outages. As a result, many jamming instances are either undetected or misidentified. Hence, developing flexible and cost-effective methods to identify and locate the source of interference is necessary.

Smartphones can be extremely valuable sensors for detecting and localizing GNSS interference. They are deployed in the billions and are in high density in our most populated areas, where GNSS interference has the potential to do the most damage. Crowdsourced information from even a small fraction of these could prove very valuable, as they can detect local interference that cannot be easily detected by reference station networks. Additionally, the Android operating system (OS) will provide access to many GNSS receiver observables that will allow for robust detection of jamming, allowing us to distinguish these events from other, natural causes of signal degradation.

An Android application (app), a backend and a frontend were developed to demonstrate a basic capability of smartphone-based GNSS interference detection. The setup was tested on the JamX 2017 exercise hosted by the US Department of Homeland Security (DHS). The app gathers basic GNSS observables such as satellite information and carrier-to-noise-density ratio ($\frac{C}{N_0}$) and pushes this data to a central server for interference detection assessment.

This thesis discusses the technology involved and proposes a way to deploy such a jammer detection and localization system, called J911.

CHAPTER 2

Background

Smartphone-based crowdsourcing of GNSS observations has been previously suggested for interference detection due to its numerous advantages [10] [11]. The existing infrastructure provides built-in communication capabilities and a smartphone-service-based system allows for ease of deployment, updating and getting feedback [10]. The high adoption rate of smartphones allows for low-cost but high-density deployment. A density of as little as 100 smartphones per squared kilometer can provide useful detection and localization results, which is easily achieved in urban areas [10]. Even lower densities enable detection, hence being an efficient deterrent to an attacker, although they may not be suitable for providing precise localization due to inaccuracies in measurements, unknown variables and other errors [10] [11]. Smartphones have been successfully used to detect GPS jamming in specific, controlled scenarios, such as interference by commercial drivers to circumvent vehicle tracking systems [12] or via an Android application on a single phone with $\frac{C}{N_0}$ readings and dead reckoning [13].

The work presented in this thesis expands upon previous research with an actual implementation of a GNSS jamming detection and localization system around low-cost, consumer hardware with a bigger scale, distributed design and wider deployment in mind. This chapter briefly goes over specifications of existing GNSS constellations, mechanisms of interference detection and localization, as well as current public infrastructure suited for deployment of the crowdsourcing system.

2.1 GNSS Basics

GNSS is a system that has global coverage and uses a satellite constellation to enable trilateration, a surveillance technique based on a receiver's distances to multiple points at known locations. Usually, a receiver is interested in four unknowns: three coordinates in a 3D system (x, y, z) and time (t). With that in mind, the system of four equations to solve for x, y, z and t can essentially be written as:

$$(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 = (\widetilde{t_i} - b - s_i)^2 \cdot c^2,$$

where x_i, y_i and z_i are coordinates of the i^{th} known satellite, and s_i its time. b is the difference between satellites' clock and the receiver's clock and $\widetilde{t_i}$ is the receiver's clock at signal reception, thus the real reception time t_i is given by $t_i = \widetilde{t_i} - b$. c is the speed of electromagnetic wave propagation, or the speed of light in vacuum. The three coordinates of the receiver, x, y and z, can be translated into latitude, longitude and altitude to a precision of a few meters in ideal conditions.

Without any prior knowledge of the receiver's location, time or state, four satellites need to be acquired and tracked to solve for the four unknowns. Tracking less than that leads to an unsolvable, underdetermined system and having more to an overdetermined system, which likely does not have a unique solution, but can still be approximated with minimization methods such as the least squares method. Another way to think about the system is through pseudoranges. A pseudorange is a distance from a satellite to a receiver, calculated through the travel time of the signal. With four such distances, a location of the receiver can be determined as the intersection of four spheres where each is a set of possible solutions to one pseudorange.

There are many high-quality resources on the topic of GNSS, such as [14], [15] and [16], to name just a few, so not much new can be added to that subject in the scope of this thesis.

2.1.1 Space Segment

As of December 2016 only the United States NAVSTAR (GPS), the Russian GLONASS (*Globalnaya Navigatsionnaya Sputnikovaya Sistema*) and the European Union's Galileo are global operational GNSSs. The European Union's Galileo is scheduled to be fully operational by 2020 [17]. China is in the process of expanding its regional BeiDou Navigation Satellite System into the global BeiDou-2 by 2020 [18].

Global coverage for each system is generally achieved by a satellite constellation of 18 to 30 medium Earth orbit (MEO) satellites spread between several orbital planes. The actual systems vary, but use orbital inclinations of over 50° and orbital periods of roughly twelve hours, at an altitude of about 20000 kilometers or 12000 miles. A comparison of various orbital characteristics of the mentioned GNSS constellations can be seen in table 2.1.

System	GPS	GLONASS	Galileo	BeiDou
Orbital altitude	20180 km	19130 km	23222 km	21150 km
Period	11 h 58 min	11 h 16 min	14 h 5 min	12 h 38 min
Number of satellites	24	24	30	35 (5 in GEO)
Number of orbital planes	6	3	3	3
Inclination	55°	64°	56°	55°
Nominal public precision	15 m	4.5 m — 7.4 m	1 m	10 m

Table 2.1: Basic comparison of GPS, GLONASS, Galileo and BeiDou constellation characteristics.

2.1.2 Signal Characteristics

The main differences and similarities, when it comes to signal characteristics of the four mentioned GNSS constellations (GPS, GLONASS, Galileo, BeiDou), are shown in table 2.2.

Frequencies used by the satellites are carefully chosen to minimize attenuation due to various atmospheric conditions, contents and layers. Furthermore, using multiple different frequencies can provide additional information about the state of the ionosphere and the troposphere, because the refraction angle is partly determined by the frequency, in turn influencing signal travel time to receiver and localization accuracy. More precise corrections are thus possible, when all available frequencies' travel time is taken into account.

Every form of wireless communication uses the same medium to transmit information – the space between the transmitter and the receiver. In order to enable simultaneous use of the common medium by multiple sources, when they are close enough to interfere with each other, channel multiple-access methods must be employed, which are CDMA and FDMA in the case of GNSS, explained further in this section.

System	GPS	GLONASS	Galileo	BeiDou
Channel access	CDMA	FDMA (moving to CDMA)	CDMA	CDMA
Public frequency	1.57542 GHz 1.2276 GHz	1.602 GHz 1.246 GHz	1.1895 GHz 1.28 GHz 1.5755 GHz	1.561098 GHz 1.561098 GHz 1.20714 GHz

Table 2.2: Basic comparison of GPS, GLONASS, Galileo and BeiDou signal characteristics.

Code-Division Multiple Access (CDMA)

CDMA is a spread-spectrum technique, meaning that the bandwidth of the data is spread around a carrier signal's frequency. The spreading code is a carefully constructed pseudo-random sequence running at a significantly higher frequency than the data to be transmitted. Before transmission, the digital data is combined with the pseudo-random code by a bitwise xor (exclusive or) operation and modulated over the carrier. When modulating on top of a carrier, every rising or falling edge in the digital signal indicates a phase shift.

Construction of the spread-spectrum data signal is shown in figure 2.1, where T_c is much lower than T_b. Resulting bandwidth (BW) is inversely proportional to signal period, thus $BW = \frac{1}{T_c}$, and is much larger than the starting bandwidth, i.e. spread. The spreading factor is determined by the ratio $\frac{T_b}{T_c}$ and to a certain extent determines the maximum number of satellites the receiver can simultaneously listen to [19].

When demodulating and decoding received CDMA signals, an inverse process is performed. In short, a reference carrier signal generated on the receiver, a plain sine wave, can be used to remove the real carrier from the input signal. Then, correlation results indicate which pseudo-random code was used to spread the data signal. The same xor operation used in coding can be used to decode the spreading code out of the received signal and leave the message intact. All these operations must account for the Doppler shift due to satellite's velocity.

Each transmitting satellite, when CDMA is used, is assigned a different pseudo-random code to modulate their signal with. It is important that all the assigned pseudo-random codes are orthogonal to each other, meaning that their correlation function is close to zero. Otherwise, it would be impossible to distinguish one satellite's signal from another's. A consequence of using CDMA is that the resulting signal is more resistant to interference than it would be if not spread.

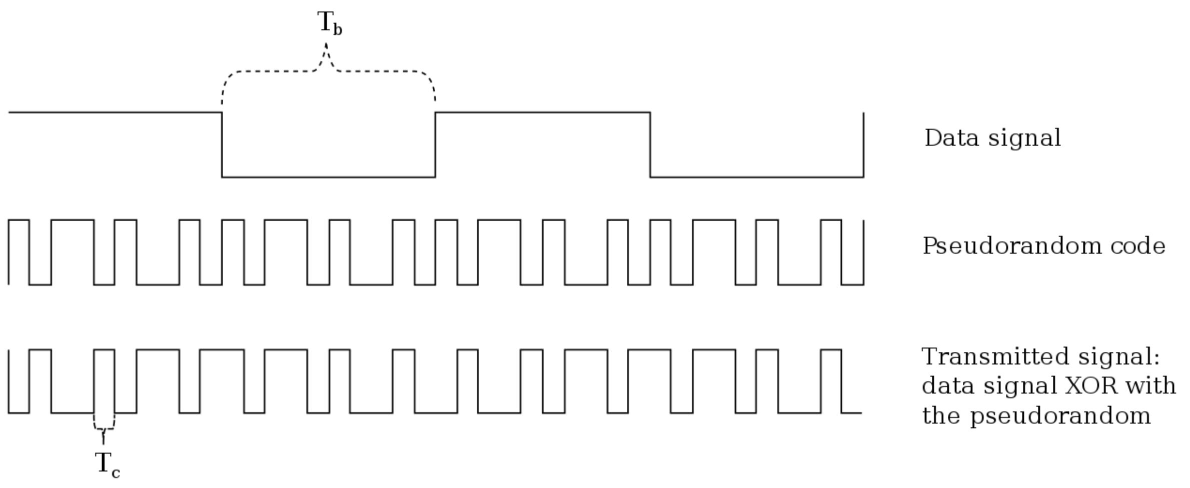


Figure 2.1: Construction of the spread-spectrum data signal from the original data signal and a pseudo-random code. [1]

Frequency-Division Multiple Access (FDMA)

FDMA is more intuitive and easier to understand than CDMA, as each transmitter is assigned one frequency, or one frequency band, of the channel and is its sole user. In FDMA, different users are then identified by the frequency they are transmitting on. However, there is a benefit in CDMA which does not apply to plain FDMA, namely interference resistance. FDMA in GLONASS uses a spreading technique, similar to CDMA, to achieve greater resistance, thus each satellites needs to be assigned a frequency band to account for the spreading and Doppler shift to a lesser extent, instead of a single frequency. All satellites in GLONASS' case use the same pseudo-random code. A visual representation of FDMA and CDMA can be seen in figure 2.2.

2.1.3 Transmitted Information

To be able to solve the system of at least four equations given in section 2.1, the satellite needs to know precise coordinates of each acquired satellite, as well as their clocks. The receiver does know the nominal orbit of each satellite, but perturbations happen, so each satellite sends accurate orbital information, called ephemeris. It also contains the correction needed because of atmospheric delays through the ionosphere and the troposphere, due to refraction.

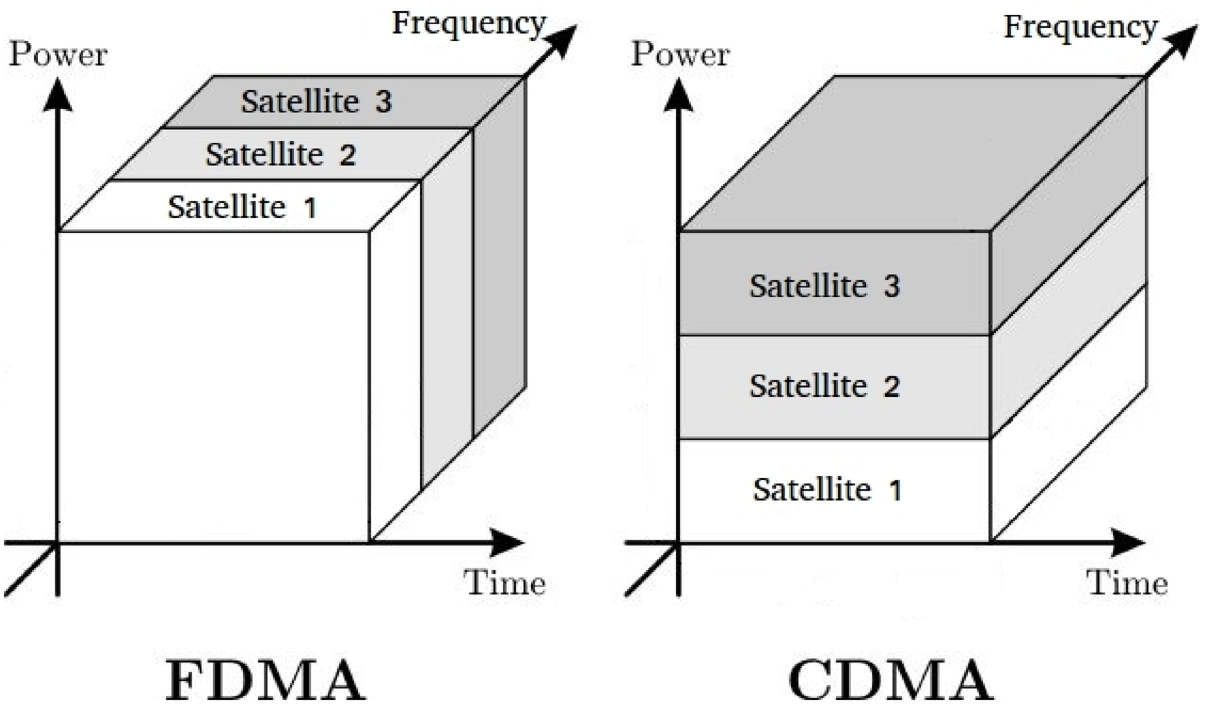


Figure 2.2: Visual representation of FDMA and CDMA. [2]

The clocks on different satellites are not actively synchronized, but are atomic, therefore very stable and precise, with multiple backup clocks. Each message includes the time of its own transmission.

Part of the information about other satellites in constellation, called almanac, is also included in every message, to ease acquisition as mentioned in section 2.1.4.

2.1.4 Receiver

A usual receiver design has an analog part and a digital part. In such a setup, the analog, also called the radio-frequency frontend, deals with downconversion, filtering, demodulation and digitization. The digital part controls the acquisition, tracking and decoding processes.

Analog Part

Processing the input signal at the carrier frequency digitally can be challenging with current computational capabilities, especially in an embedded system, so the first step on the receiving end is downconverting the signal to baseband, i.e. closer to the code's originating frequency. Mixing, also called multiplying or heterodyning, two signals centered around f_1 and f_2 will produce two new signals at $|f_1 - f_2|$ and $f_1 + f_2$ containing the same information, therefore a specific reference signal is used on the receiver to downconvert the input to a frequency of choice that is easier to process.

The next step is filtering out the high-frequency $f_1 + f_2$ component and a lot of noise, leaving only the $|f_1 - f_2|$ part of the spectrum present, with the relevant information and a relatively small amount of noise. Since in FDMA each satellite has its own, much narrower bandwidth, compared to CDMA, high-performing filters must be used in the frontend to separate them.

Next step usually brings the processing into the digital domain – sampling. A variable-gain amplifier (VGA) is used in receivers to amplify the incoming signal, after down-conversion and filtering, to a specific level. After amplification, it is sampled, then digitized/quantized and passed on to the digital part of the receiver, all via the analog-to-digital converter (ADC). The specific amplification level needed is determined by the receiver design and, more specifically, its digital section. To be able to accommodate input signals of varying power, automatic gain control (AGC) mechanism is responsible for altering the gain level of the VGA by varying the voltage input to one of its pins, as illustrated in figure 2.3. The required AGC voltage is controlled by the number of samples in each of the quantization bins, as illustrated by the histogram in figure 2.4. For example, if the incoming signal is quantized with two bits into four possible values, a goal could be to have 32% of the samples in minimum and maximum bins, which correspond to -3 and 3 in figure 2.4, respectively. If the percentage of samples in the maximum and minimum bins starts increasing, the AGC voltage is lowered to reduce gain, thus increasing the number of samples which fall into bins closer to 0. Inversely, if the number of samples in maximum and minimum bins starts decreasing beyond 32%, the AGC voltage is increased.

Digital Part

When a receiver is turned on, it does not know which GNSS satellites are visible, so it needs to try to find a satellite in the seemingly random signal received, in a process called acquisition. A satellite can be found by correlating its pseudo-random code with

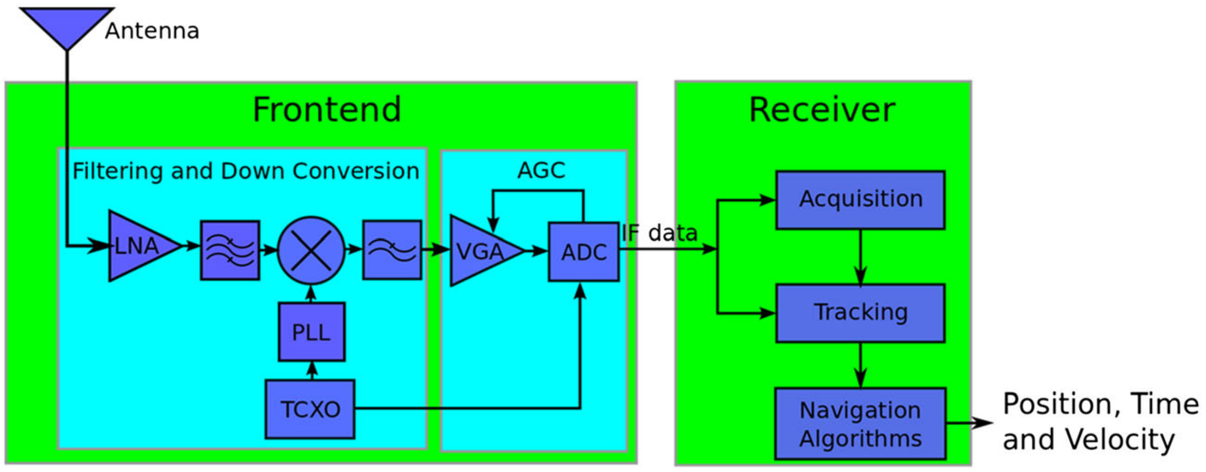


Figure 2.3: Block diagram of a GNSS receiver with Low Noise Amplifier (LNA), Phase Lock Loop (PLL), Temperature Compensated Crystal Oscillator (TCXO) and AGC. The AGC utilizes a Variable Gain Amplifier (VGA). [3]

the input signal, optionally after removing the carrier wave. If that satellite's signal is being clearly received, the correlation result will show a noticeable spike at the matching position. The ratio of the biggest spike to the second biggest spike can be used as an acquisition metric. Furthermore, the velocity of satellites in orbit causes a significant Doppler shift which needs to be accounted for when acquiring and tracking satellites. Therefore, to acquire a satellite, the search space needs to contain all satellites' codes and all possible Doppler shift amounts, usually divided into bins, as well as phase shifts. In the case of FDMA, the receiver knows which satellite to look for, as they all have the same pseudo-random code, but there are multiple central frequencies to look at, not counting the Doppler bins, so the total acquisition time is comparable to CDMA systems.

Depending on the exact system specifications, the whole acquisition process with no prior knowledge of the constellation state can take several minutes or even an hour on low-powered devices. Parsing almanac data received from one satellite can significantly reduce that time by eliminating large portions of the search space, thus ignoring the search for satellites which are on the other side of the planet.

Finally, when a satellite is acquired, the bits it is sending can be decoded with certainty, but its Doppler shift needs to be tracked since the satellite is still orbiting. The way the tracking is usually done is by generating a total of three pseudo-random code replicas, shown in figure 2.5. One replica is prompt, being shifted by, ideally, the signal's Doppler shift. The other two replicas are early and late – one slightly shifted positive and the other negative of the prompt replica. All three replicas are correlated with the incoming signal; the prompt one to demodulate the signal and the other two to tell in which

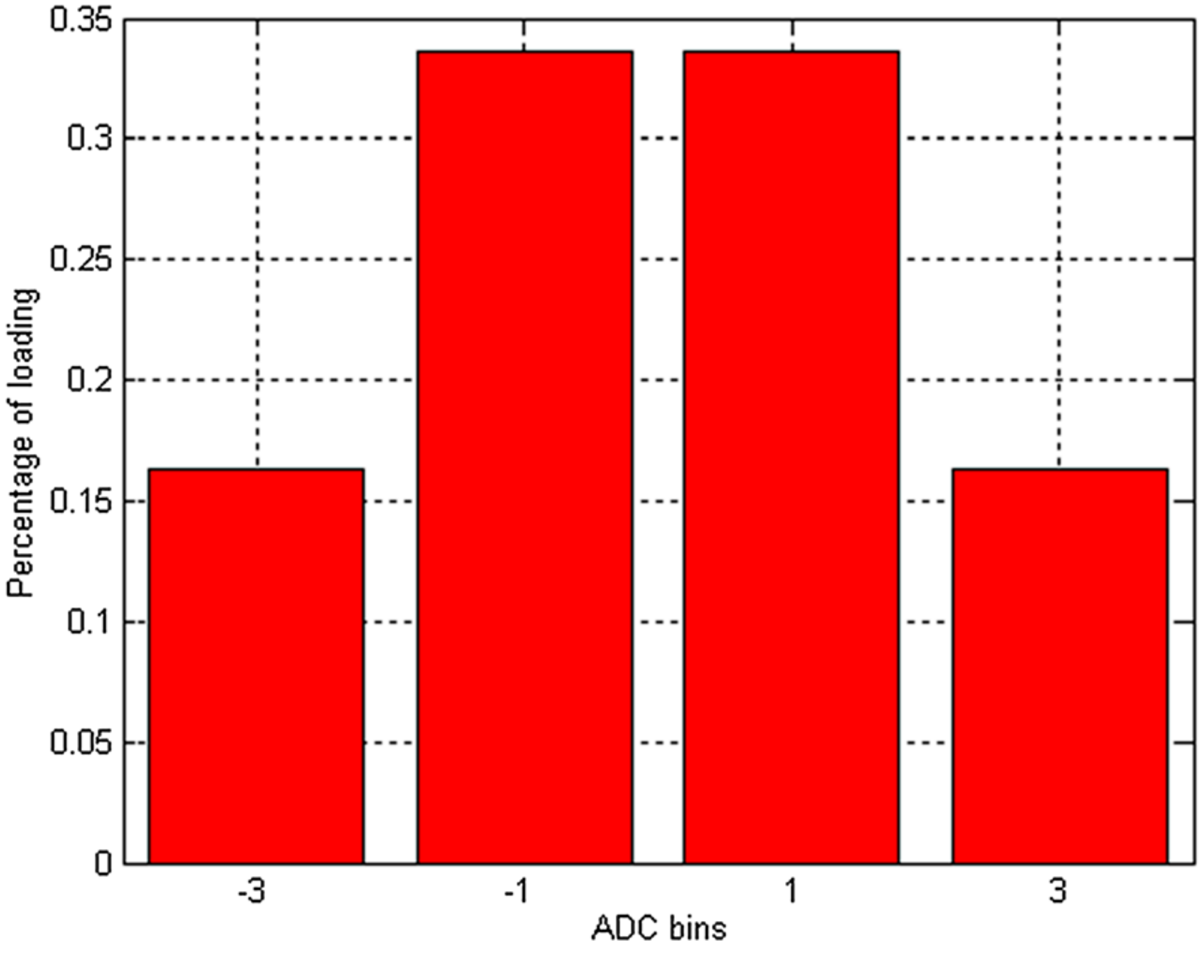


Figure 2.4: Bins for a 2 bit ADC. [3]

direction is the signal moving. Results of that correlation tell the receiver the signal's shift amount and direction, so that the prompt pseudo-random code can be adjusted as needed.

A high-level architecture of the digital part of a GNSS receiver, as explained in the previous paragraphs, is shown in figure 2.6.

2.2 Interference Detection

GNSS interference detection can be accomplished using observables, such as $\frac{C}{N_0}$ and AGC levels, available to most GNSS receivers. $\frac{C}{N_0}$ is a standard metric used to quantify the

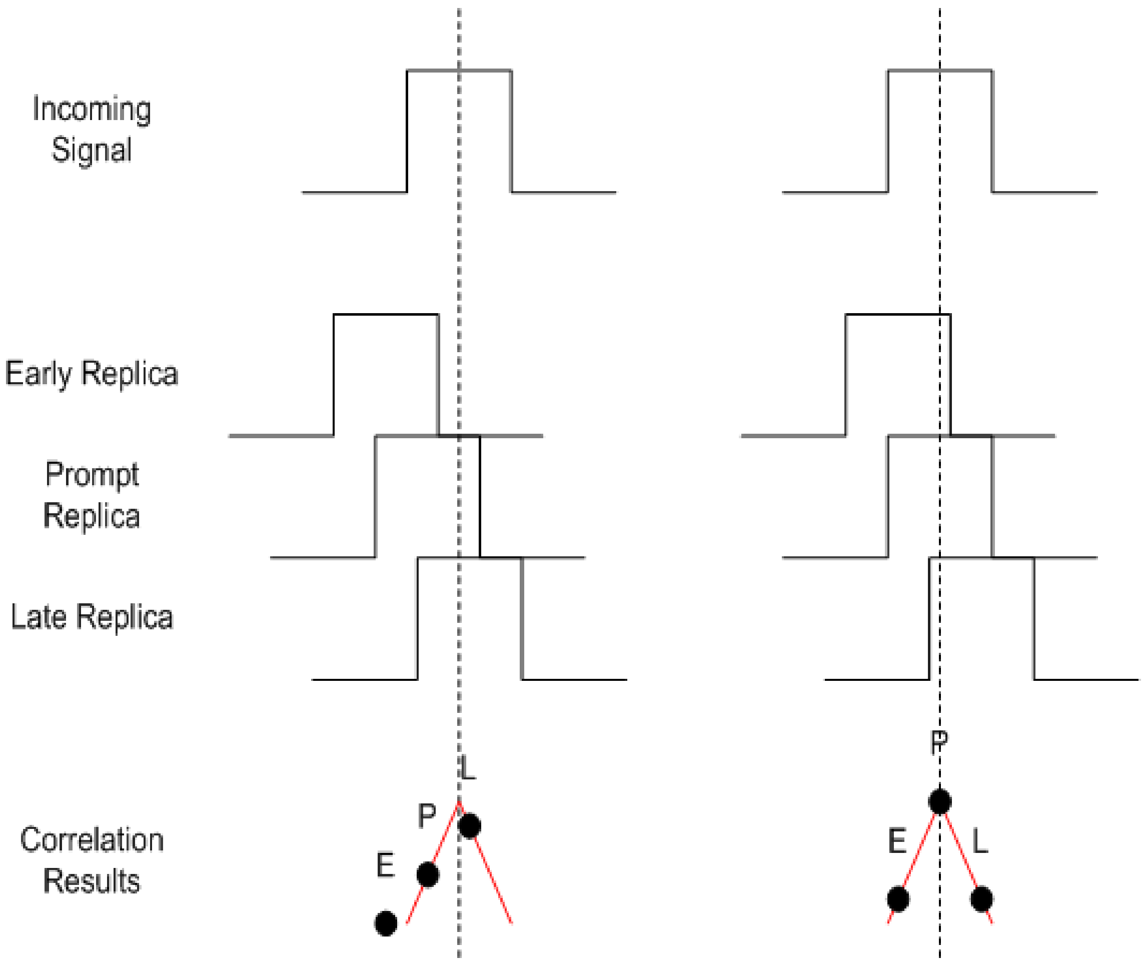


Figure 2.5: Illustration of correlation of an input with three replicas, late, prompt and early, when the late replica is closer to the incoming signal (Left) and when the prompt replica is spot on (Right). [4]

power of a tracked GNSS satellite signal relative to noise and is similar to signal to noise ratio $(\frac{S}{N})$ [1]. It is calculated for each individual tracked signal. AGC level essentially indicates the amount of energy entering the antenna, and as such does not change by the tracked signal. $\frac{C}{N_0}$ and AGC can provide indications of anomalous energy that may be the result of interference. More powerfully, these two complementary measures can be used together to differentiate degradation due to interference from spoofing and natural causes. These are convenient measures for a crowdsourced detection system. Android OS provides access to $\frac{C}{N_0}$ measurements. Android 8 or Oreo, introduced in August 2017, also provides the ability to access AGC information. Future smartphones should have AGC measurements accessible, provided that the smartphone original equipment manufacturer (OEM) and the GNSS chipset support it.

[1] $\frac{C}{N_0}[dBHz] = \frac{S}{N}[dB] + 10\log(BW[Hz])$, where BW is the signal bandwidth.

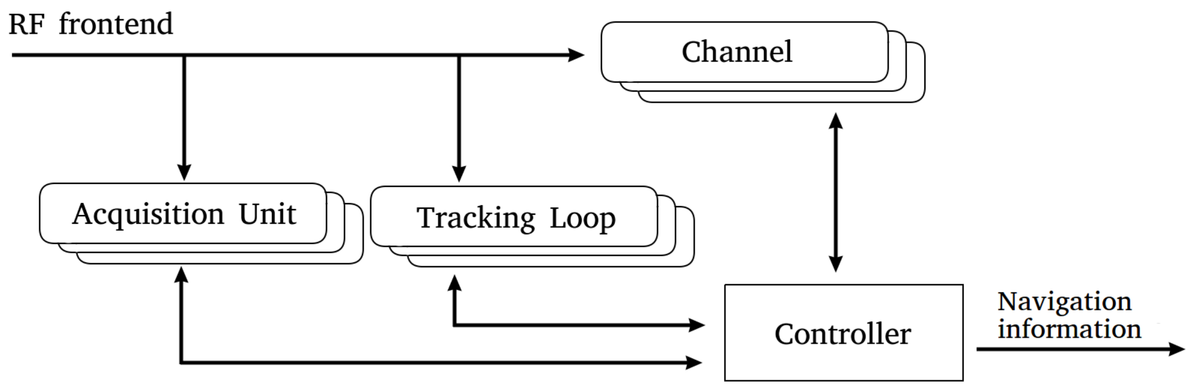


Figure 2.6: High-level architecture of the digital part of a GNSS receiver.

When a jammer approaches a GNSS receiver, such as a smartphone, the interfering noise level increases. This increases noise on all satellite signals and hence decreases $\frac{C}{N_0}$ on all satellites. As the jammer gets closer, the received interference may become powerful enough to cause the GNSS receiver to lose track of the satellites. However, reduction of $\frac{C}{N_0}$ and loss of tracking can occur due to natural causes such as going under foliage or going indoors. Hence, $\frac{C}{N_0}$ measurements from individual receivers are generally not adequate for robust jamming detection. Additional information from AGC, $\frac{C}{N_0}$ time history, different frequency bands and other receivers can be used to make $\frac{C}{N_0}$-based jamming detection more robust.

Using $\frac{C}{N_0}$ for rapid interference detection typically requires a comparison to expectations. For the comparison, the nominal $\frac{C}{N_0}$ for the receiver must be known. This is typically around 30 to 40 dB-Hz. Also, it should be known which satellites are reachable from the location of the receiver. This allows determining if the $\frac{C}{N_0}$ value for a specific satellite should be received, indicating if something has caused the receiver not to track an available satellite.

A jammer emits a relatively powerful and noisy signal into the GNSS spectrum. The high power signal causes the AGC voltage to drop in nearby receivers, reducing amplification of the input signal, including the useful signal from the GNSS satellite. In that case, the digitized samples correspond more closely to the jammer's noise and do not contain enough information to resolve and extract the weak signal from the GNSS satellites – the receiver is jammed.

To use AGC for interference detection, the nominal AGC voltage needs to be known. Each device has its own nominal AGC voltage, as different receivers use different discrete components and different antennas. This is important to ascertain as many factors (i.e. imperfections in manufacturing processes, differences in operating temperature, etc.) can result in different nominal AGC levels even in the same model. The nominal AGC voltage/level can be determined by collecting samples over time and, for example, averaging them in ideal conditions. Statistical deviations of AGC from its nominal average or standard deviation provides useful information. A voltage increase means that satellites are obstructed, thus less power is received, and a voltage decrease indicates interference or jamming that deposits more power into the spectrum. A scenario is illustrated in figure 2.7, where around hour 13 (on the x axis), the AGC level increases to around 2750, which is significantly above the average of 2510 plus its standard deviation of 99. Such an increase might suggests obstruction. On the other hand, around hour 54, the AGC level decreases significantly below the standard deviation, to around 2300, suggesting interference in the GNSS spectrum.

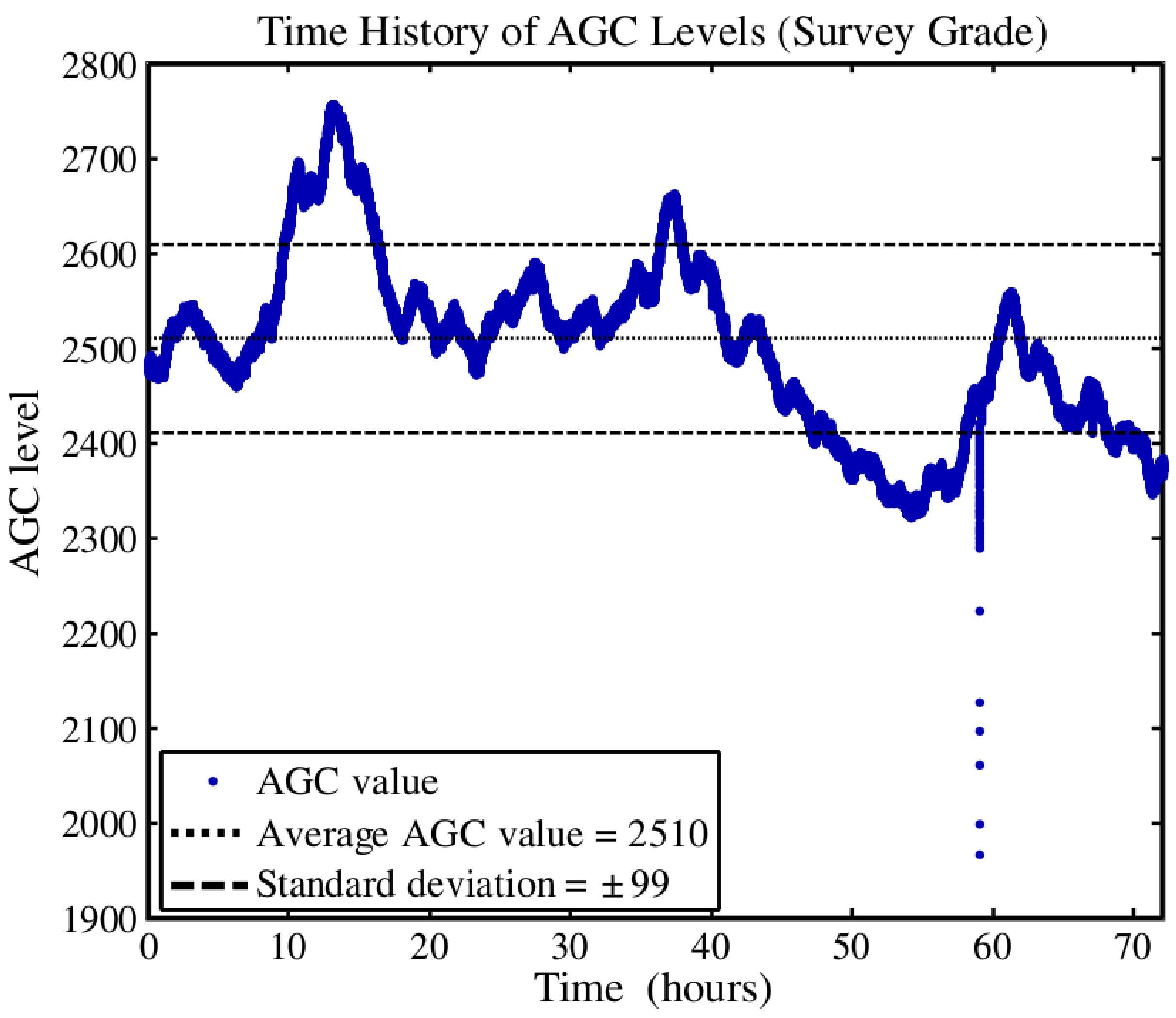


Figure 2.7: AGC samples with the average and the standard deviation marked. [3]

While $\frac{C}{N_0}$ and AGC can be used individually for interference and potentially spoof detection, when used together, they can distinguish different causes of GNSS degradation [5] [20]. They can differentiate cases of environmental changes (i.e. going indoors) from man-made interference. Figure 2.8 shows AGC and $\frac{C}{N_0}$ values from different scenarios. It shows the different relationship between $\frac{C}{N_0}$ and AGC for spoofing, radio-frequency interference (RFI) and normal/nominal conditions. Generally speaking, jamming and RFI increase incoming energy (i.e. lowers AGC levels) and lowers $\frac{C}{N_0}$, whereas spoofing increases incoming energy while having similar or higher $\frac{C}{N_0}$ levels. In summary, the combined use of $\frac{C}{N_0}$ and AGC increase jamming detection robustness by reducing false positives.

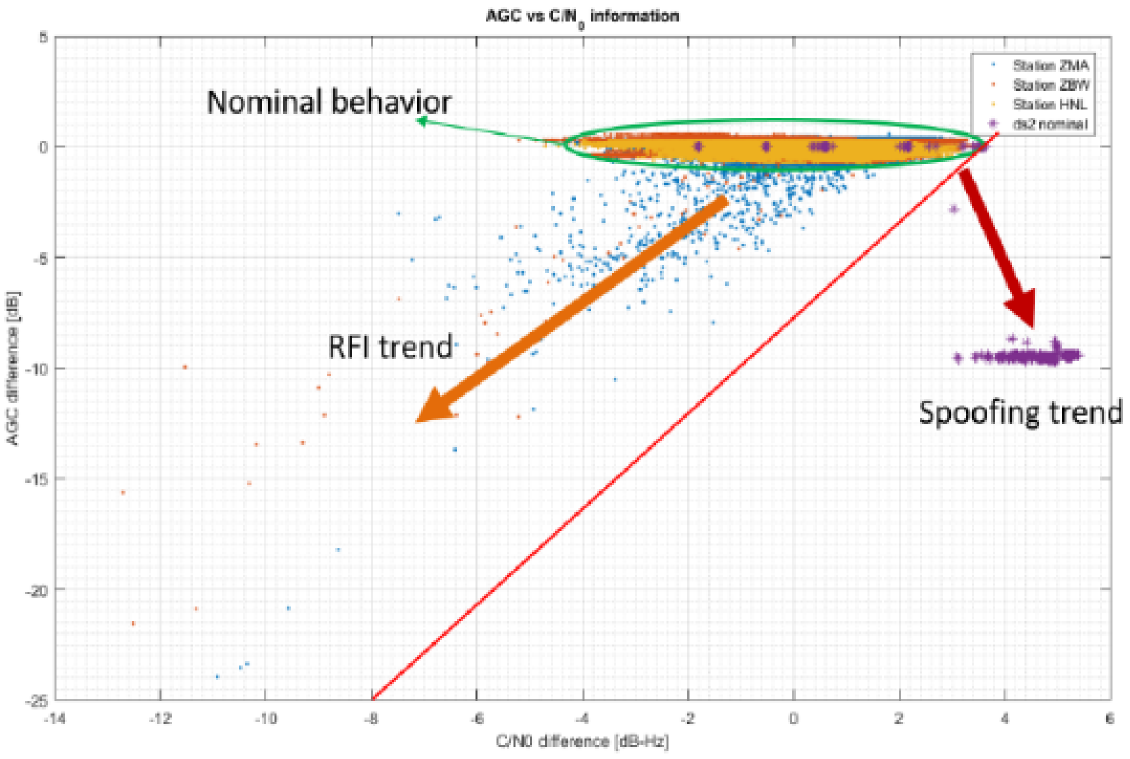


Figure 2.8: Effects of RFI, Spoofing and Nominal Conditions on AGC and $\frac{C}{N_0}$. [5]

2.3 Localization

A powerful capability of a crowdsourced GNSS measurements is the ability to localize jamming. $\frac{C}{N_0}$ and AGC are coarse measures that are not meant to provide ranging, bearing or positioning information. Hence, single measurements cannot indicate jammer location. However, measurements from multiple smartphones can allow for geolocation of a jamming source due to the geometric diversity of these measurements. This capability requires that the smartphone measurements be shared or sent to a central server for processing. Time difference of arrival (TDOA) processing and power difference of arrival (PDOA), both multilateration techniques, are possible ways to provide refined localization.

With both TDOA and PDOA, if more than three measurements, i.e. four devices, want to be taken into account when solving for jammer's location, the system becomes overdetermined and likely will not have a unique solution. Even with three measurements a solution often does not exist in practice due to various unknown variables in the environment. In such scenarios, a good approach is using a least squares method to find the solution which best fits all measurements in the least squares sense.

2.3.1 Time Difference of Arrival (TDOA)

TDOA, like the name implies, works by measuring the difference in times of the signal's arrival to two or more receivers with known locations (x_i, y_i and z_i, for the i^{th} receiver). Assuming that there are three unknown variables of interest – jammer's coordinates in a three-dimensional Cartesian system (x, y and z) – a starting system of equations is:

$$d_i = \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2}, \tag{2.1}$$

where d_i is the distance from the jammer to the i^{th} receiver. The distance is, of course, unknown, but information about it can be gained from the difference of arrival times, when compared to another receiver:

$$\begin{aligned} t_i - t_{i+1} &= \Delta t_i, \\ d_i - d_{i+1} &= \Delta t_i \cdot c, \end{aligned} \tag{2.2}$$

where t_i is the time of arrival of the i^{th} signal. If equation (2.1) and equation (2.2) are combined to relate coordinates to time difference, it goes:

$$\sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} - \sqrt{(x_{i+1} - x)^2 + (y_{i+1} - y)^2 + (z_{i+1} - z)^2} = \Delta t_i \cdot c. \tag{2.3}$$

With a single time difference measurement from a pair of receivers, the system in equation (2.3) can be solved for a hyperboloid which makes up all the possible positions of the jammer [21].

When another, third receiver is present, an additional measurement can be procured, reducing the possible solutions to the curve that is the intersection of the two hyperboloids. Note that with a set of three receivers, three pairs of receivers can be made, however, one pair's information is contained in the other two pairs. Therefore, to solve system in equation (2.3) for a single point, a total of at least four receivers is needed [21]. It is necessary to have the receivers' time synchronized [21], which will be the case if they acquired GNSS satellites "recently".

2.3.2 Power Difference of Arrival (PDOA)

Similar to TDOA, PDOA, also known as Received Signal Strength (RSS), measures the difference in the received power at two or more receivers with known locations (x_i, y_i and z_i, for the i^{th} receiver). Again, the unknowns are same as in the TDOA case, jammer's coordinates (x, y and z), as shown in equation (2.1). To properly relate the power difference to distances, an electromagnetic wave propagation model is needed. It is impossible to take into account all the obstacles and antenna properties that can influence input power measurements, so a valid approximation ranges from the inverse square law to the inverse biquadratic law [22]:

$$from \quad P_d \propto \frac{1}{d^2} \quad to \quad P_d \propto \frac{1}{d^4},$$

where P_d is the power received at distance d. For the sake of simplicity, the inverse square law is used as an example in the following equations, which as a consequence has:

$$\frac{P_{d_i}}{P_{d_{i+1}}} = \frac{d_{i+1}^2}{d_i^2},$$

for the measurements from the i^{th} and the $(i+1)^{th}$ receiver. In the field of radiocommunication, the power is often expressed using the logarithmic scale:

$$P_{d_{i+1}}[dB] - P_{d_i}[dB] = \Delta P[dB] = 10 \cdot 2 \cdot \log\left(\frac{d_{i+1}}{d_i}\right). \tag{2.4}$$

Finally, when equation (2.1) and equation (2.4) are put together, the system is:

$$\Delta P[dB] = 10 \cdot \log \left(\frac{(x_{i+1} - x)^2 + (y_{i+1} - y)^2 + (z_{i+1} - z)^2}{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} \right) \tag{2.5}$$

Unlike with TDOA, a single measurement of PDOA with a pair of receivers solves the system in equation (2.5) for a sphere of possible jammer locations [21] [22]. With additional power difference measurements, the jammer's location solution becomes the intersection of multiple spheres. Like with TDOA, four receiving devices are needed to solve the system of equation (2.5) for a single point [21] [22].

2.4 Enhanced 911 (E911)

In the United States of America, E911 is an upgrade to the 911 system that enables the automatic reporting of telephone number and location of every 911 caller, wired or wireless, to public safety entities. Such a system allows for prompt reaction even when communicating a location is difficult or impossible.

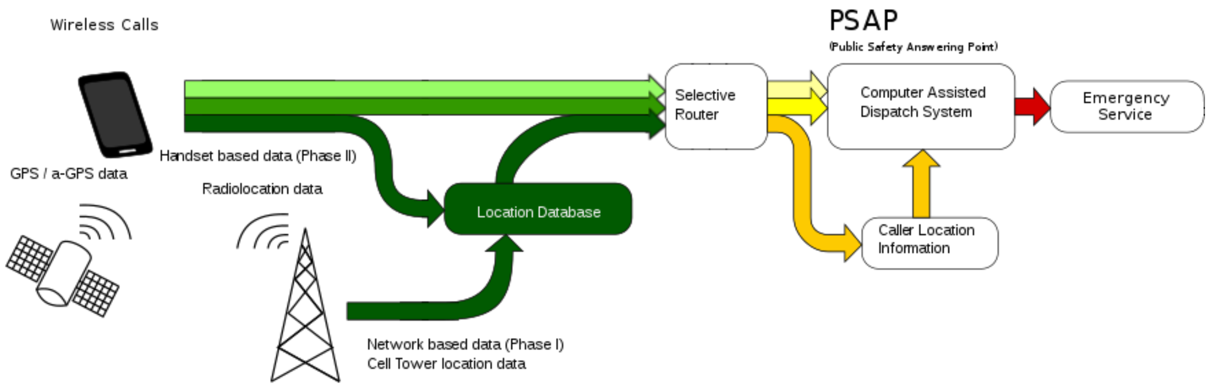

Wireless Calls
PSAP
(Public Safety Answering Point)
Selective Router
Computer Assisted Dispatch System
Emergency Service
Handset based data (Phase II)
GPS / a-GPS data
Radiolocation data
Location Database
Caller Location Information
Network based data (Phase I)
Cell Tower location data

Figure 2.9: Illustration of the process behind E911 localization. [6]

Location is determined by the information in the Automatic Location Information (ALI) database, which is also used to update the Master Street Address Guide (MSAG) database. Emergency calls are routed to the appropriate Public Safety Answering Point (PSAP) based on the data in the MSAG database, after being routed through the Public Switched Telephone Network (PSTN) and through a special facility called a Class 4 telephone switch or a Tandem office. Then, the ALI database is queried for the caller's location from the PSAP.

Two phases are involved in obtaining the caller's location. Phase I sends the cell tower's location to the PSAP, which is easier to obtain, while Phase II sends the cellphone's location, as is illustrated in figure 2.9

Enhanced 911 infrastructure is an excellent candidate to be reused for J911, since the changes needed for J911 implementation would need to be made only in software, across the whole stack, as further detailed in chapter 3.

Chapter 3

J911 Prototype

The J911 system could be seen as having three independent, but connected, parts. On one end is the measurement-collecting hardware and software. In the prototype, a smartphone with the developed Android app took that role, however, any device capable of GNSS reception would fit it. On the other end is the visualization frontend. It is a piece of software that can show the taken measurements to a user or simply display/dispatch a notification when a jammer is detected. Between the two ends sits the backend, responsible for receiving and parsing the smartphones' messages and storing the data in a database for easy querying.

This chapter explains the architecture and implementation of the J911 prototype system in greater detail, covering the whole stack.

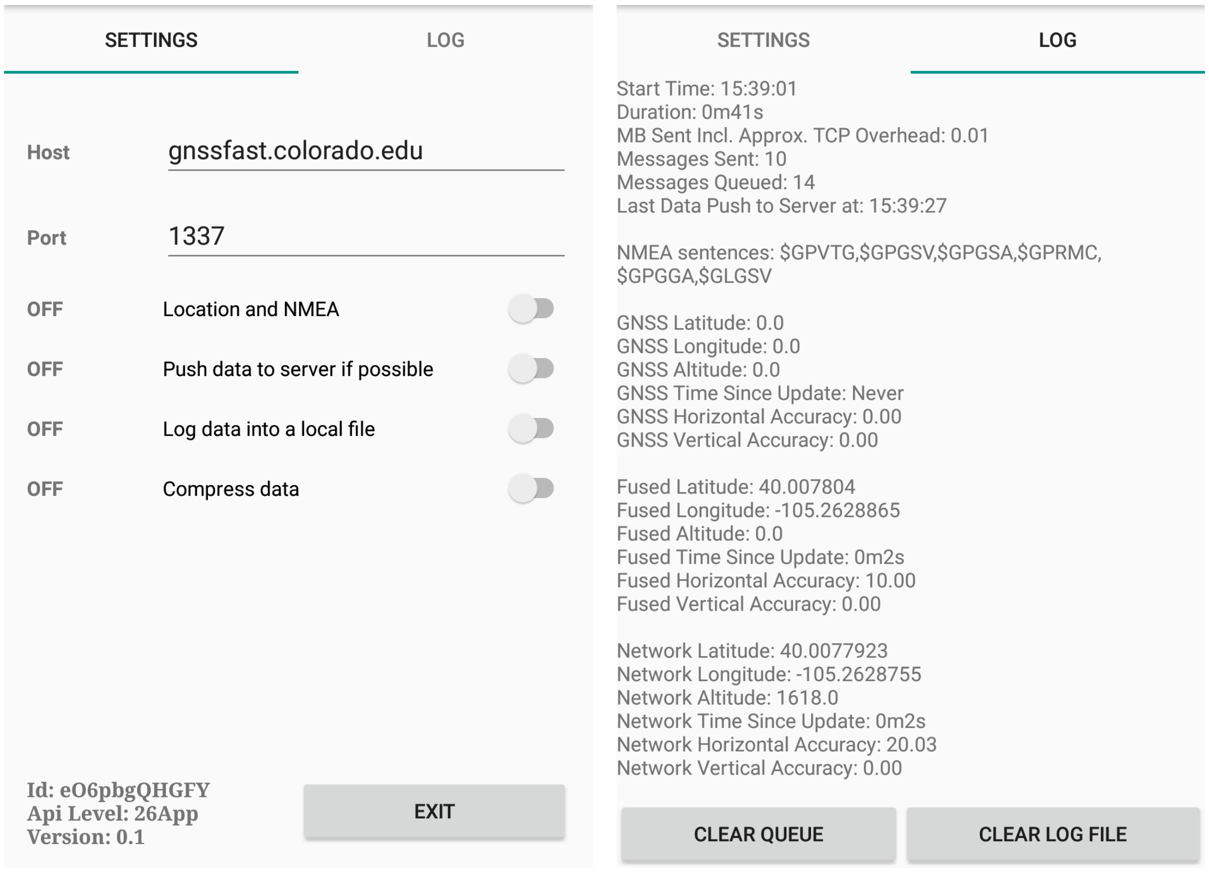
3.1 Smartphone App

The bespoke app for capturing, storing and sending GNSS measurement information was developed and designed to get that information, if available, from the OS. It looks for any of the following: position, accuracy, satellite, $\frac{C}{N_0}$, pseudo range and AGC values. The app supports all receivable constellations, namely GPS, GLONASS, Galileo and BeiDou. It is built without using the features introduced in Android OS 8.0, but assumes to support it at some point and is prepared for it. Hence, it can operate on earlier version of Android as well, having an eye towards the future and newer capabilities. These observables are captured at a software-configurable rate, defaulting to 1 Hz, and can be uploaded to a server, should connectivity be available.

The application was developed in the programming language Java, which is well supported in the Android ecosystem, through the official Android software development kit

(SDK) and the official integrated development environment (IDE) Android Studio. It is based on the open-source GNSSLogger application developed by Mohammed Khider of Google to demonstrate GNSS capabilities of the Android OS [23].

The settings screen is shown on the left side of figure 3.1, with options explained later in this section. The logging screen, shown on the right of figure 3.1, includes all the information that is sent to a remote server, for user's preview.



Host gnssfast.colorado.edu

Port 1337

OFF Location and NMEA

OFF Push data to server if possible

OFF Log data into a local file

OFF Compress data

Id: eO6pbgQHGFY
Api Level: 26App
Version: 0.1

```
Start Time: 15:39:01
Duration: 0m41s
MB Sent Incl. Approx. TCP Overhead: 0.01
Messages Sent: 10
Messages Queued: 14
Last Data Push to Server at: 15:39:27

NMEA sentences: $GPVTG,$GPGSV,$GPGSA,$GPRMC,
$GPGGA,$GLGSV

GNSS Latitude: 0.0
GNSS Longitude: 0.0
GNSS Altitude: 0.0
GNSS Time Since Update: Never
GNSS Horizontal Accuracy: 0.00
GNSS Vertical Accuracy: 0.00

Fused Latitude: 40.007804
Fused Longitude: -105.2628865
Fused Altitude: 0.0
Fused Time Since Update: 0m2s
Fused Horizontal Accuracy: 10.00
Fused Vertical Accuracy: 0.00

Network Latitude: 40.0077923
Network Longitude: -105.2628755
Network Altitude: 1618.0
Network Time Since Update: 0m2s
Network Horizontal Accuracy: 20.03
Network Vertical Accuracy: 0.00
```

Figure 3.1: GNSS interference detection app. Data collection settings left and log right.

3.1.1 Raw GNSS Measurements, NMEA Sentences and Location

The application can gather the following location- and GNSS-related information offered through the Android application programming interface (API):

- location updates sourced by GNSS,
- location updates sourced by the cellular network,
- location updates sourced by the fused provider,
- standardized National Marine Electronics Association (NMEA) sentences.

Location updates provided by the cellular network are based on the known cellular tower locations and are fairly imprecise in comparison to GNSS-sourced location. The fused provider is available through the Google Play Service. While not built into Android, Google Play Services are widely available on smartphones, therefore they were used too. The fused provider takes into account GNSS, cellular network, visible WiFi and Bluetooth stations when calculating current location. Ephemeris and information about acquired satellites is sent to the application using NMEA sentences, which are parsed to extract the desired information. A sample of NMEA sentences acquired on a smartphone is shown in listing 3.1.

```
$GLGSV,2,1,07,69,70,061,35,79,66,069,37,80,37,187,26,70,35,318,33*60
$GLGSV,2,2,07,68,25,111,32,86,11,330,32,85,11,279,30*56
$GPGSV,3,1,11,22,77,258,42,31,60,059,42,03,55,305,41,14,38,072,36*72
$GPGSV,3,2,11,01,33,235,38,26,28,136,30,23,26,278,38,32,19,082,32*7E
$GPGSV,3,3,11,25,12,036,26,11,08,225,29,16,09,159,*49
$GPGGA,080741.00,4332.360641,N,11249.798235,W,1,17,0.4,1527.6,M,-17.4,M,,*63
```

Listing 3.1: Sample of NMEA sentences captured on a smartphone.

Later Android versions (7.0 and up) offer an interface to programmatically extract the same information that is contained in the NMEA sentences, however it was not used in the application to avoid diverging in source code between older and newer Android versions. Version 8.0 introduced API to get AGC level, which will be useful in future revisions of the application.

```
OnTrackingTurnedOn() {
    locationManager.requestLocationUpdates(NETWORK_PROVIDER,
                                           LOCATION_UPDATE_RATE,
                                           onLocationUpdate);
    locationManager.requestLocationUpdates(GNSS_PROVIDER,
                                           LOCATION_UPDATE_RATE,
                                           onLocationUpdate);
    googlePlayServices.requestLocationUpdates(FUSED_PROVIDER,
                                              LOCATION_UPDATE_RATE,
                                              onLocationUpdate);
    locationManager.requestNMEAUpdates(onNMEAUpdate);
    locatoinManager.requestRawGNSSUpdates(onRawGNSSUpdate);
}

OnTrackingTurnedOff() {
  locationManager.removeLocationUpdates(NETWORK_PROVIDER);
  locationManager.removeLocationUpdates(GNSS_PROVIDER);
  googlePlayServices.removeLocationUpdates(FUSED_PROVIDER);
  locationManager.removeNMEAUpdates();
  locatoinManager.removeRawGNSSUpdates();
}
```

Listing 3.2: Pseudocode of starting and stopping raw GNSS measurements, NMEA sentences and location updates.

Listing 3.2 shows pseudocode of the start and stop of location updates, along with GNSS measurements and NMEA sentences. Functions `OnTrackingTurnedOn` and `OnTrackingTurnedOff` are tied to "Location and NMEA" switch shown in figure 3.1. Each of the `requestLocationUpdates` function calls take a callback as one of its arguments, to deliver its information to. NMEA sentences and raw GNSS measurements call their callback whenever a measurement is ready, with the update rate provided to `requestLocationUpdates` with `GNSS_PROVIDER` as the first argument. A notable difference with `FUSED_PROVIDER` is that it is provided by Google Play Services.

```
onLocationUpdate(provider, location){
  switch(provider) {
    case NETWORK_PROVIDER:
      network_lat = location.lat;
      network_lon = location.lon;
      network_alt = location.alt;
      network_hacc = location.hacc;
      network_vacc = location.vacc;
      network_timestamp = location.timestamp;
      break;
    case GNSS_PROVIDER:
      gnss_lat = location.lat;
      gnss_lon = location.lon;
      gnss_alt = location.alt;
      gnss_hacc = location.hacc;
      gnss_vacc = location.vacc;
      gnss_timestamp = location.timestamp;
      break;
    case FUSED_PROVIDER:
      fused_lat = location.lat;
      fused_lon = location.lon;
      fused_alt = location.alt;
      fused_hacc = location.hacc;
      fused_vacc = location.vacc;
      fused_timestamp = location.timestamp;
      break;
}

// In future version, for AGC support.
onRawGNSSUpdate(measurement) {
  switch(measurement.constellation) {
    case BEIDOU:
      beidou_agc = measurement.agc;
    break;
    case GALILEO:
      galileo_agc = measurement.agc;
      break;
    case GLONASS:
      glonass_agc = measurement.agc;
      break;
    case GPS:
      gps_agc = measurement.agc;
      break;
}

onNMEAUpdate(sentence) {
  storeSentence(sencence);
  parsed = parseNMEA(sentence);

  switch(parsed.type) {
    case GPGSV:  // GPS
      visibleGPSSats = parsed.visibleSats;
      visibleGPSCN0s = parsed.CN0s;
      break;
    case GLGSV:  // GLONASS
      visibleGLONASSSats = parsed.visibleSats;
      visibleGLONASSCN0s = parsed.CN0s;
      break;
    case GPGGA:  // End of NMEA cycle.
      message.all_nmea = popStoredSentences();
      message.timestamp = currenTime();
      message.id = getId();
      message.gnss = {gnss_lat, gnss_lon, gnss_alt,
                      gnss_hacc, gnss_vacc, gnss_timestamp};
      message.network = {network_lat, network_lon, network_alt,
                         network_hacc, network_vacc, network_timestamp};
      message.fused = {fused_lat, fused_lon, fused_alt,
                        fused_hacc, fused_vacc, fused_timestamp};
      message.agcs = {beidou_agc, galileo_agc, glonass_agc, gps_agc};
      message.gps_sats = visibleGPSSats;
      message.glonass_sats = visibleGLONASSSats;
      message.gps_cn0s = visibleGPSCN0s;
      message.glonass_cn0s = visibleGLONASSSats;

      storeAndPutIntoSendQueue(message);
      message.clear();

      break;
    case
  }
}
```

Listing 3.3: Pseudocode of raw GNSS measurements, NMEA sentences and location updates callbacks.

Listing 3.3 shows pseudocode where all the information requested in listing 3.2 gets received and processed. As previously mentioned, AGC measurements will be supported in the future. Whenever any location provider gives an update, the information is stored, overwriting previous values and waiting to be sent out. GNSS location provider is synchronized with NMEA sentence update cycle, which usually starts with GPGSV and ends with GPGGA sentences. The cycle may be different between different phones, but on the same phone it was observed to always be a consistent cycle, thus it is possible to rely on a single sentence type to signal the end of a cycle. Once the cycle is finished, a message is constructed from all available data and is stored on the phone and sent over the network if desired. Only GPGSV and GLGSV sentences are used to extract information from, as they hold detailed satellite information about GPS and GLONASS, respectively. Other *GSV sentences will be researched and implemented once observed on phones which support BeiDou and Galileo. GPGGA is exclusively used for signaling the end of the cycle as seen in all phones tested on.

3.1.2 Internet Connectivity

Messages are created periodically on the phone and sent to the server as soon as a connection can be established, if the user wishes. The user can select the remote destination – host address and port – where the messages get sent to. A message contains all the information that is received from the OS and the times of last updates. A pretty-printed example of such a message is shown in listing 3.4.

```
{
   "other_nmea":"$GPGGA,080741.00,4332.360641,N,
                 11249.798235,W,1,17,0.4,1527.6,M,-17.4,M,,*63",
   "GLGSVs":[
      "$GLGSV,2,1,07,69,70,061,35,79,66,069,37,
      80,37,187,26,70,35,318,33*60",
      "$GLGSV,2,2,07,68,25,111,32,86,11,330,32,
      85,11,279,30*56"
   ],
   "GPGSVs":[
      "$GPGSV,3,1,11,22,77,258,42,31,60,059,42,
      03,55,305,41,14,38,072,36*72",
      "$GPGSV,3,2,11,01,33,235,38,26,28,136,30,
      23,26,278,38,32,19,082,32*7E",
      "$GPGSV,3,3,11,25,12,036,26,11,08,225,29,
      16,09,159,*49"
   ],
   "id":"cjACrijA1QA",
   "timestamp":1500624462232,
   "lat_gnss":43.53934467869471,
   "lon_gnss":-112.82997114268005,
   "alt_gnss":1510.1347219543397,
   "hacc_gnss":4,
   "vacc_gnss":0,
   "timestamp_gnss":1500624459999,
   "lat_network":0,
   "lon_network":0,
   "alt_network":0,
   "hacc_network":0,
   "vacc_network":0,
   "timestamp_network":0,
   "lat_fused":43.5393447,
   "lon_fused":-112.8299711,
   "alt_fused":1510.1347219543397,
   "vacc_fused":0,
   "hacc_fused":4,
   "timestamp_fused":1500624459999,
   "agc_bei":0,
   "agc_gal":0,
   "agc_glo":0,
   "agc_gps":0
}
```

Listing 3.4: Sample of JSON message that gets sent from a smartphone to a server.

The message is constructed in JavaScript Object Notation (JSON) format and is ASCII encoded. Because of its encoding, there is a lot of redundancy. The message, as given in the example, is 929 bytes long. If, instead, binary encoding was used, with just sending the extracted data, where every numerical value is presented with 8 bytes as a double-precision floating-point number and timestamps as 8 byte integers, the message would be 647 bytes long, which is a significant saving even with the approximate transmission control protocol (TCP) overhead of 40 bytes with each message. Furthermore, presenting numerical values with 4 bytes as single-precision floating-point numbers and timestamps as 4 byte integers can still give enough precision while further reducing the message size.

Message size and frequency are important considerations if one wishes to reduce the load on cellular towers in densely populated areas, if E911 infrastructure were to be used for jamming detection [10].

Listing 3.5 shows pseudocode of transmission and (re)connection logic. `sendOneMessage` function is repeatedly called in a dedicated background thread in the application. At first, the `doTransmit` variable is checked, as it says whether any messages should be sent at all. The value of the variable is controlled by the "Push data to server if possible" switch from figure 3.1. If transmission is turned off, busy-waiting on the condition would waste a lot of processing power, thus the app sleeps for a second, before exiting the invocation, which will promptly be repeated. `currentMessage` is initialized to a null value and is filled up with an actual message from the sending queue at the beginning of the `while` loop. Popping from the sending queue is a blocking operation in this case, thus it waits for a message indefinitely, if the queue is empty. However, if at the beginning of the `while` loop a message already exists, it means that it was not successfully sent in the previous loop, so the application sleeps for a second before retrying. Both before and after transmission, the socket is checked for connection status, as data loss is unacceptable. At the end, only if the current message is successfully sent, the queue is tested for other messages. When there is no other message waiting in the sending queue, the invocation will return.

```
sendOneMessage() {
  if (!doTransmit) {
    sleep(1000);
    return;
  }

  currentMessage = null;
  do {
    // If the current message doesn't exist, grab a new one.
    if (currentNmeaMessage == null) {
      currentMessage = popFromSendQueue();
      // If the current message already exists, it means that last
      // transmission failed, so sleep and repeat.
    } else {
      sleep(1000);
    }

    // If below check passes, it means the socket isn't connected.
    if (socket == null || !socket.isConnected() || socket.isClosed() ||
          !socket.isBound() || socket.isOutputShutdown()) {
        socket.connect(ipAddress, port);
        output = socket.getStream();
    }

    output.write(currentMessage);
    if(output.writeFailed()) {
      socket.close();
      continue;
    }
    // Loop until we send curren message or until the queue is emptied.
  } while (!isSendQueueEmpty());
}
```

Listing 3.5: Pseudocode of message transmission and connection logic.

3.1.3 Storage

To prevent data loss in unexpected scenarios, where the developed application crashes, every message that is created is stored into a file on the phone in the JSON format, shown in listing 3.4. If recovered from the phone, the file can be "replayed" to the remote server.

When Internet connectivity is unavailable, the messages to be sent are stored in working memory of the application. To remember which messages need to be sent from the previous app run, if a restart happened, all unsent messages are saved from the working memory into an SQLite database that is a standard part of the Android OS and independent of other applications. On app initialization, all unsent messages are recovered from the SQLite database, and when a message is sent, it is removed from the database, shown in listing 3.6. The layout of the SQL `unsentMessagesDatabase` is very simple – one column for the mandatory unique ID and one column for the message content.

```
restoreFromUnsentDatabaseOnAppInit() {
  cursor = unsentMessagesDatabase.query(TABLE_NAME, COLUMN_NAME_TEXT);
  while (cursor.moveToNext()) {
    message = cursor.message;
    putIntoSendQueue(message);
  }
  cursor.close();
}

storeIntoUnsentDatabasOnShutdown(message) {
  valuesToStore = {COLUMN_NAME_MESSAGE, mesage};
  unsentMesageDatabase.insert(TABLE_NAME, valuesToStore);
}

removeFromUnsentDatabaseWhenSent(message) {
  selection = COLUMN_NAME_MESSAGE + " LIKE ?";
  selectionArguments = {message};
  unsentMessagesDatabase.delete(TABLE_NAME, selection, selectionArguments);
}
```

Listing 3.6: Pseudocode of unsent message receovery, storing and removal.

3.2 Server Software

A publicly reachable machine was used to host the backend, which consisted of a web server, implemented in Python using the Twisted networking library, and the PostgreSQL database. When a message on the phone is ready, it is sent through the opened TCP connection to the server. After receiving a message, the server interprets it as a JSON object and extracts all data from it. If no problems were encountered in the message formatting, the data is stored into the PostgreSQL database, laid out in figure 3.2.

```
onLineReceived(line) {
  message = parseAsJSON(line);
  cursor = database.startCommit():
  messageID = database.insertMessage(cursor, message.id, message.timestamp)
  database.insertNetworkLocation(cursor, messageID, message.network_lat,
                                 message.network_lon, message.network_alt,
                                 message.network_hacc, message.network_vacc,
                                 message.network_timestamp);
  database.insertGNSSLocation(cursor, messageID, message.gnss_lat,
                              message.gnss_lon, message.gnss_alt,
                              message.gnss_hacc, message.gnss_vacc,
                              message.gnss_timestamp);
  database.insertFusedLocation(cursor, messageID, message.fused_lat,
                              message.fused_lon, message.fused_alt,
                              message.fused_hacc, message.fused_vacc,
                                  message.fused_timestamp);
  database.insertGPSSatsAndCN0s(cursor, messageID,
                                  message.gps_sats, message.gps_cn0s);
  database.insertGLONASSSatsAndCN0s(cursor, messageID,
                                  message.glonass_sats, message.glonass_cn0s
  database.commit(cursor);
}
```

Listing 3.7: Pseudocode of the backend.

The crux of the server software, also called backend, is illustrated in listing 3.7. The Twisted framework takes care of the low-level details of multiplexing connections and invokes the `onLineReceived` callback every time a message, or a line of text, is received. The callback will try interpreting the received bytes as a JSON object and will extract data from it to be stored in the database.

3.2.1 Processing Algorithm and Visualization Software

With the data in the central server, processing was developed to identify jamming. For the initial development, a simple algorithm was created for detection and identification. Each receiver reported, in a message to the server, the $\frac{C}{N_0}$ of all satellites that it sees and each message was treated individually without consideration of other receivers or prior messages. Because at least four satellites need to be acquired for a position solution to be available, the best four measurements reported in a message were averaged for each constellation. If less than four satellites' $\frac{C}{N_0}$ was measured at a time, the missing ones were assumed to be zero. The single calculated number for each constellation for an instant in time is compared to a fixed threshold to determine if that given receiver indicates jamming at that given time. A more sophisticated version may have examined the time history of the receiver to set thresholds. Additionally, satellites may be weighted based on their likelihood to be attenuated by jamming or other sources.

Frontend software was developed to visualize all the measurements taken. Since all captured information gets stored into the database, the frontend is independent of other parts of the stack, as long as the database layout does not change. MATLAB was used to query the database for specific time periods and specific phones and then used to plot locations of phones and the measured $\frac{C}{N_0}$, overlaid on a map of the area, over the duration of the test. An example of a processed batch is shown in figure 3.3, where on the left hand side in the image, a lot of low $\frac{C}{N_0}$ data points may be seen, which is an indicator of GNSS jamming, according to the theory.

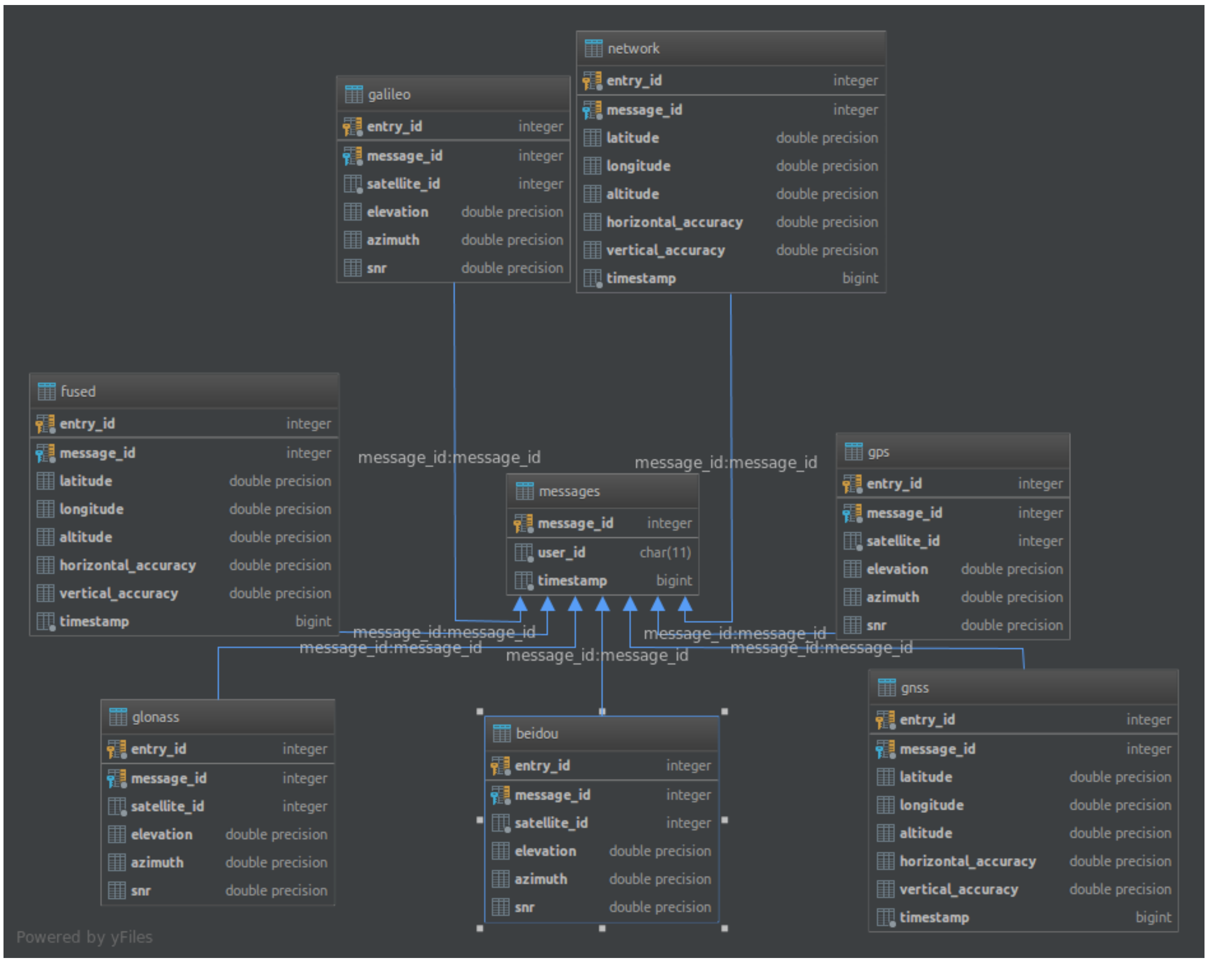


Figure 3.2: Layout of the PostgreSQL database used in the backend.

3.2.2 Privacy

Whenever, yet another, third party wants to collect, store and process user-identifiable and sensitive data, such as near-realtime location information, privacy concerns are raised, and rightly so.

Google, the owner of the Android platform, already collects location information about its users [24] and wireless cellular service providers do the same [25]. However, the developed application does not collect identifiable data, but only the unique user ID which resets on every app (re)installation. The unique id is important in determining device's nominal values for AGC and $\frac{C}{N_0}$, and their standard deviation, but is not critical for the app's basic functionality. Nonetheless, the collected information is easily correlated to Google's or wireless service providers' data, in the unlikely case of law enforcement requiring it.

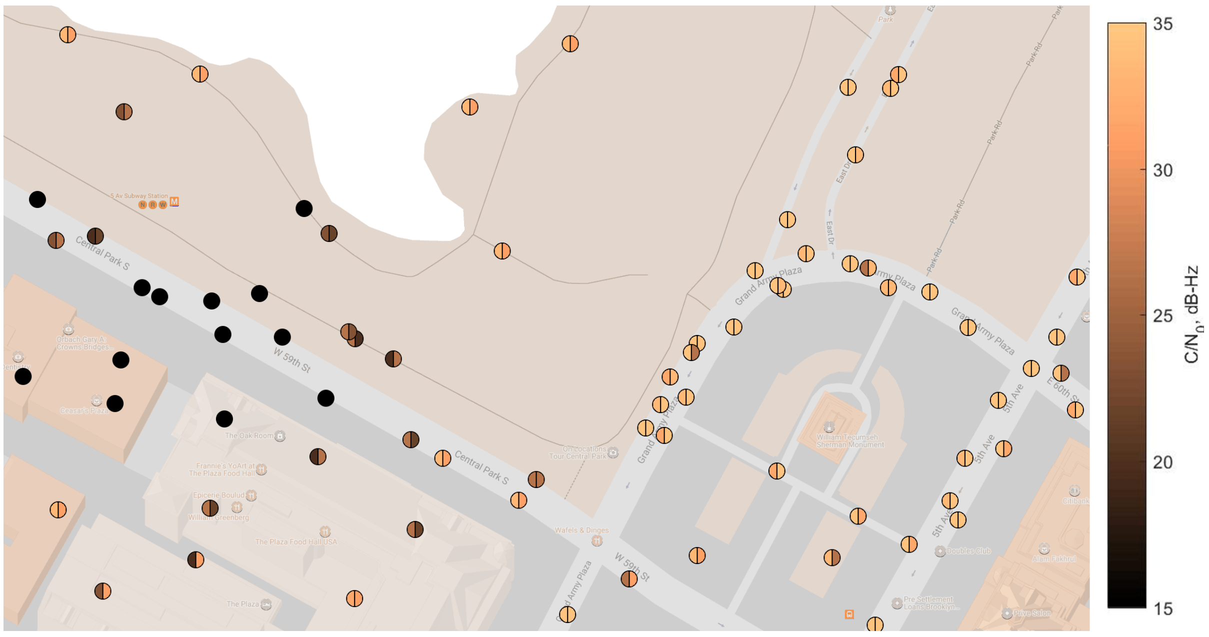


Figure 3.3: An example of a processed batch of GNSS measurements for GPS (Left in each sphere) and GLONASS (Right in each sphere) $\frac{C}{N_0}$.

After all, it should be noted that, in the cases of law enforcement requiring otherwise sensitive information, little value is added with the developed app, since the data that law enforcement finds useful exists elsewhere [24] [25].

3.3 J911 Architecture

Architecture of the J911, a system for realtime GNSS jamming monitoring on the national level, can be similar to the E911 system, relying on existing standards and infrastructure [10]. All essential hardware is already in place, except the smartphones with GNSS chips supporting required features. However, with the current relatively short smartphone life-cycle, it is a matter of a few years when a significant portion of the Android market will have a minimum or higher Android OS version needed to run the developed prototype application, if historical trends continue [26]. Furthermore, current phones with supporting hardware can be upgraded by over-the-air updates. Overall, the current upgrading practices, albeit not good for consumers, are a big benefit for fast deployment. To get the system up and running, cell stations and some nodes in the PSTN would also have to update their software. Biggest addition to the system would be the servers commissioned to run the backend.

It is hard to discuss the best placement of backend servers and its interface to the rest of the E911 system, without knowing the proprietary details of the PSTN implementations. Nonetheless, an educated guess could be made that a placement close to the ALI database could be useful in updating it from the same incoming datastream. Depending on which services would answer a jamming incident, the path to the other endpoint, a PSAP, may or may not be shared with the J911 system. On one hand, if local emergency services should be immediately notified, it would make sense to route J911 warning to the PSAP. On the other hand, notification from the J911 servers could be sent elsewhere for further analysis. The proposed placement of the backend servers is illustrated in figure 3.4.

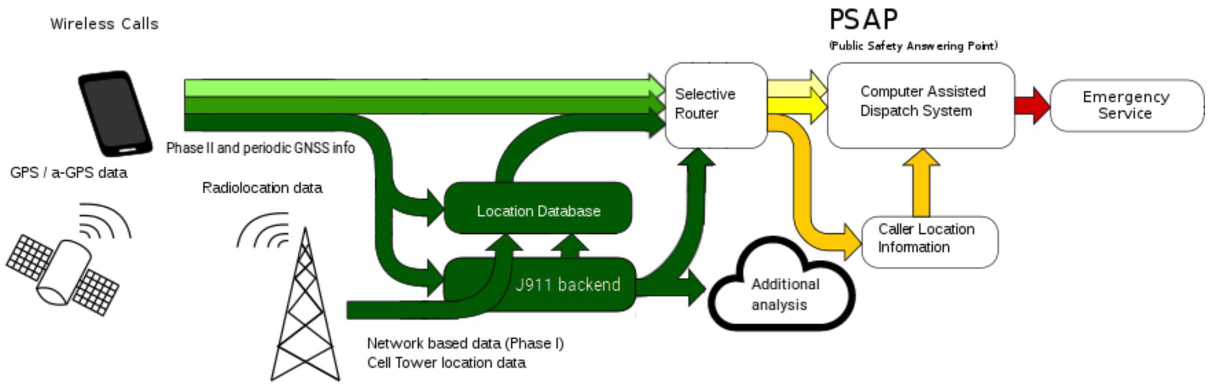


Figure 3.4: Illustration of the process behind E911 localization and J911 servers placement. [6]

3.4 SiGe GN3S Sampler Module

Not strictly the topic of this thesis, but definitely worth a mention, is the SiGe GN3S Sampler, a GPS RF frontend developed by the Colorado Center for Astrodynamics Research. A significant amount of time allocated for this thesis went into updating the host software for this software-defined radio (SDR), to use the latest, stable version of the libusb library for USB communication, and to make it robust for use in tests. During testing of the J911 prototype, it was important to have the GN3S Sampler present, to be able to corroborate AGC measurements with $\frac{C}{N_0}$ measurements, since the AGC was inaccessible on the Android smartphones. Despite its use as the go-to SDR at CCAR, and the fact that the in-house software was built around that SDR, the raw data being recorded is SDR- and processing-software-agnostic [27].

In summary, the SiGe GN3S Sampler is a low cost platform based on the SiGe SE4120 or the SiGe SE4110 GPS frontend chip and Cypress FX2 USB controller. Having such a platform, controlled by custom software, allows far more control than Android OS and OEMs provide at the moment. With that control, of course, comes raw data collection of EM samples at the intermediate frequency (IF) and AGC value samples. Since everything after the frontend is based in software, adding high-level features, such as distributed data collection via networking, is possible. The GN3S Sampler performs the same process that is shown in figure 2.3, as well as illustrated in figure 3.5.

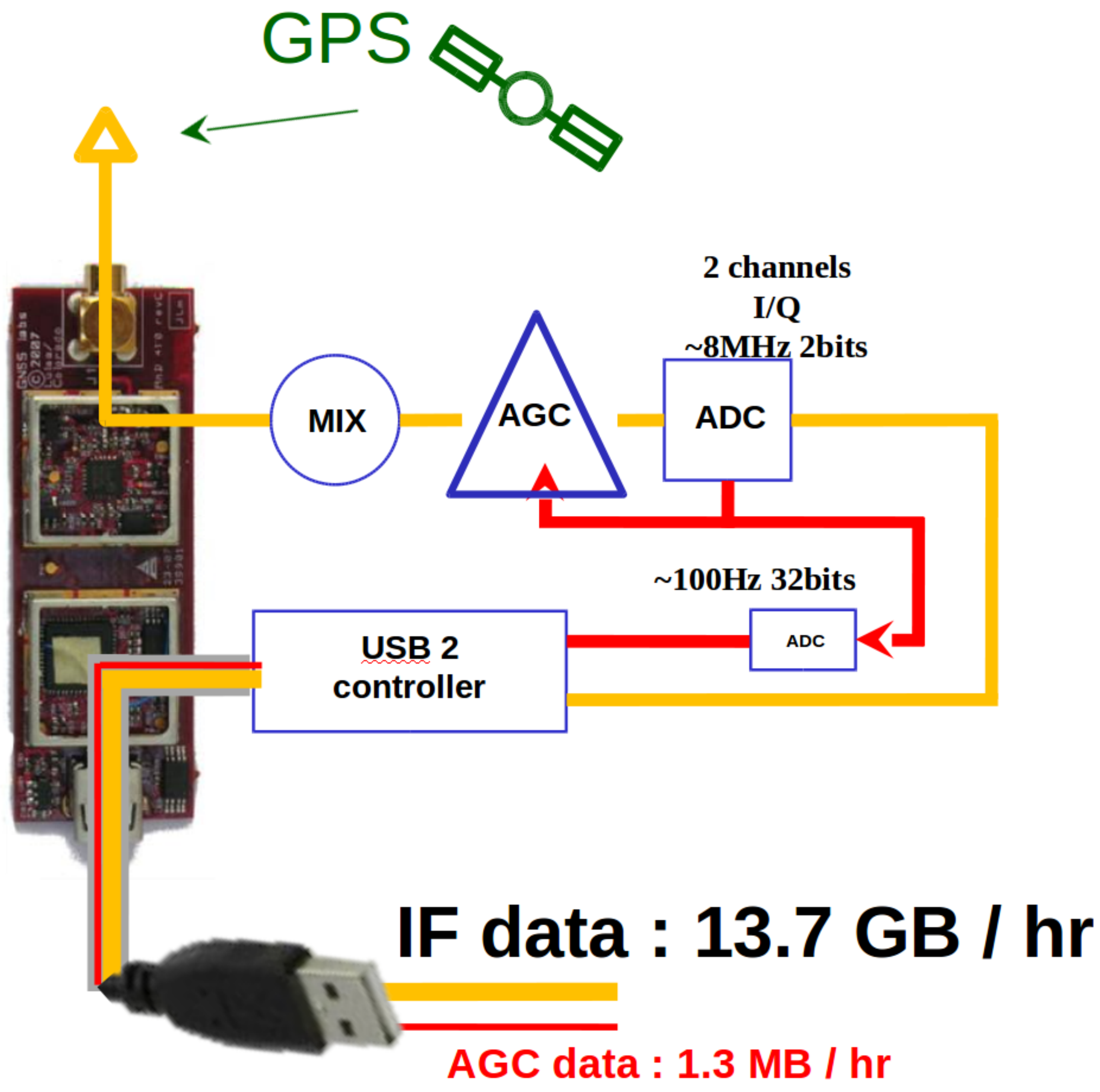


Figure 3.5: Illustration of data flow in a SiGe GN3S Sampler, overlaid on an image of the module. [7]

The module is connected via a USB 2.0 connection to a PC, which has to be running the host software, a portable program written in C++ and utilizing the libusb library. The host software accepts command line parameters to configure the module with. After the USB connection has been established and the module is configured, it immediately begins streaming IF and AGC samples to the host at a maximum rate of roughly 4 MB and 200 B per second, respectively. Apart from simply recording data to a file, the host software also supports dumping a user-specified period of samples to a file when triggered by AGC voltage going above a specific threshold. Another feature is controlling many other GN3S Samplers through their respective hosts with the low level Asio networking library, on top of a standard TCP/IP network stack. In that case, the captured data can be sent to the central host if the connection is fast enough, or it can just receive data dumps whenever any of the remote modules' AGC threshold is triggered.

Chapter 4

Discussion

During the implementation of the J911 prototype system, laid out in this thesis, the plan was to have the main premise tested and have questions answered, such as:

- Can $\frac{C}{N_0}$ and AGC be acquired from a smartphone?
- What is the quality of such measurements on a smartphone?
- What conclusions can be drawn based on the measurements from a single smartphone?
- What conclusions can be drawn based on the measurements from a crowd of smartphones?
- Can a GNSS jammer be detected with confidence based on crowdsourced data?
- Can a GNSS jammer be localized with confidence based on crowdsourced data?
- Is such a J911 system feasible on a large scale, as determined by battery life, bandwidth use and computational capabilities on a phone and a server?

A test, named 2017 DHS JamX exercise, was performed, with the help and approval from the US Department of Homeland Security. This exercise supported tests of different technologies to address interference issues, such as GNSS interference. It was held at a remote location in eastern Idaho, United States, over three nights in July 2017. On two of those nights, termed Night 1 (July 19 — 20) and Night 2 (July 20 — 21), 15 smartphones, 5 GN3S Samplers and various commercial jammers were fielded. The jammers had varying specified levels of radiated power from milliwatt to watt levels. They also potentially had different jamming waveforms. Each jammer was operated statically in the test area before being driven in a roughly 2-mile loop at low speeds.

Sadly, the DHS did not approve the publication and sharing of the promising results of the exercise before this thesis was due, thus the results are left out. The team that performed the test is hoping to have the results published in the Institute of Navigation International Technical Meeting in February of 2018.

After fielding the system and reviewing the results, a few ideas for improvement came up, as outlined in the continuation of this chapter.

Increased Receiver Density and Localization

A test with a higher density of smartphones may make locating the jammer visually easier and would provide more data to be analyzed. With enough samples gathered, and a high enough receiver density, possibilities of localization with TDOA or PDOA increase.

Improved Robustness and Spoofing Detection with AGC

Android OS version 8.0 introduces an API to get AGC level of the smartphone GNSS chipset. While AGC has in general been shown to be useful for both GNSS interference and spoofing detection [3], the utility of AGC within a smartphone needs to be validated. The electromagnetic environment in smartphones is volatile due to the various radios that are located inside the limited space, but the extent of its effects on AGC measurements and spoofing detection should be researched once devices supporting Android 8.0 are in the stores and more phones with supporting hardware appear on the market.

Better Data Processing Algorithms

Current data processing methods are quite simple, but more advanced data processing or averaging algorithms may give better results, in terms of how easy it is to subsequently detect jamming or differentiate going indoors from true jamming. Furthermore, to save bandwidth and energy, an adaptation algorithm could be used on smartphones, that provides updates less frequently when there is no suspicion of jamming. When the measurements become suspicious, the update frequency could temporary be increased to provide more information to the backend to process, just when it is needed the most.

Leverage Other Smartphone Information

The received GNSS power in a smartphone can be influenced by many factors. Due to the smartphone design constraint, its orientation could significantly influence the power of received signals. Since design and hardware between different phones differs, the ideal phone orientation for one phone might not be ideal for another. Quantization of orientation's effects on measurements would be beneficial in future test. Afterwards, the use of internal smartphone measures of orientation may help calibrate out the effects of orientation. Similarly being indoors, in a pocket or under foliage can also reduce signal power and $\frac{C}{N_0}$. Smartphone sensors such as light sensor, gyroscope, other radio-frequency measures, etc. may be useful for detecting these scenarios and reducing false positives.

Data Compression

Assuming a 70% smartphone penetration in the United States [28] [29], with a population of roughly 324 million [30], the total required throughput on a national level, at 1 Hz, 1 kB updates, would be approximately 1.814 terabit per second, which can be very taxing on the systems in question. There is a lot of spatial redundancy when ASCII encoding and JSON format are used, as well as temporal redundancy. Using binary encoding reduces spatial redundancy, but temporal still remains, thus bandwidth savings could be achieved and load on cellular towers reduced if messages were batched together, compressed and then sent to the server.

Authentication

The backend that was developed as the prototype accepts incoming connections from any actor, allowing for easy abuse of the experimental system. Authentication between a GNSS receiver and the backend should have great importance in a production system and implemented for a more robust solution, to prevent spoofing of messages.

Chapter 5

Conclusion

The work presented in this thesis has laid a practical foundation towards using smartphones for crowdsourcing GNSS jamming detection, by building an Android application and supporting server software. A specific architecture for the J911 system has been proposed, albeit without going into the details of PSTN. The thesis argues that smartphones are indeed a suitable platform for collection of GNSS observables, especially when newer hardware and Android OS versions become available. While the idea is still in its infancy, it is definitely worth exploring. GNSS receiver hardware is becoming cheaper, smaller and more energy efficient, warranting further adoption. However, the same can be said about GNSS jammers, thus successful standardization and deployment of the J911 system may be an effective way to combat GNSS jamming. The test, that was performed, was not performed on a big scale and there are points that deserve further addressing, as mentioned in chapter 4. The work is still in its preliminary phase, but nonetheless, based on previous research, the results are likely to improve with a bigger scale and the points that should be addressed have been solved in other industries and applications. In the author's opinion, J911's future is bright.

Bibliography

[1] Marcos Vicente, "Generation of cdma." https://web.archive.org/web/20170920020639/https://en.wikipedia.org/wiki/File:Generation_of_CDMA.svg, 2010.

[2] Karoling, "Fdma, tdma ja cdma graafiline võrdlus." https://web.archive.org/web/20170920021340/https://commons.wikimedia.org/wiki/File:FDMA,_TDMA_ja_CDMA_graafiline_v%C3%B5rdlus.jpg, 2013.

[3] D. M. Akos, "Who's afraid of the spoofer? gps/gnss spoofing detection via automatic gain control (agc)," *Navigation*, vol. 59, no. 4, pp. 281–290, 2012.

[4] GMV Innovating Solutions, "Mc replicas." https://web.archive.org/web/20170920022200/http://www.navipedia.net/index.php/File:Mc_replicas.png, 2011.

[5] E. G. Manfredini, D. M. Akos, Y. H. Chen, S. Lo, T. Walter, and P. Enge, "Effective gps spoofing detection utilizing commercial receivers technology." submitted to Navigation: The Journal of the Institute of Navigation, 2017.

[6] Evan Mason, "9-1-1 system." https://web.archive.org/web/20170920022734/https://en.wikipedia.org/wiki/File:9-1-1_System.svg, 2012.

[7] J.-P. Poncelet and D. M. Akos, "A low-cost monitoring station for detection & localization of interference in gps l1 band," in *Satellite Navigation Technologies and European Workshop on GNSS Signals and Signal Processing,(NAVITEC), 2012 6th ESA Workshop on*, pp. 1–6, IEEE, 2012.

[8] J. C. Grabowski, "Personal privacy jammers: Locating jersey ppds jamming gbas safety-of-life signals." https://web.archive.org/web/20170910001057/http://gpsworld.com/personal-privacy-jammers-12837/, 2012.

[9] J. Coffed, "The threat of gps jamming: The risk to an information utility," *Report of EXELIS, Jan*, 2014.

[10] L. Scott, "J911: The case for fast jammer detection and location using crowdsourcing approaches," in *Proceedings of the 24th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS 2011)*, pp. 1931–1940, 2001.

[11] B. R. Secrest, D. Richie, and B. Rapp, "Mathematical underpinnings of crowdsource gps jammer geolocation using signal strength readings from various locations." 7th International Conference on Localization and GNSS, 2017.

[12] G. Kar, H. Mustafa, Y. Wang, Y. Chen, W. Xu, M. Gruteser, and T. Vu, "Detection of on-road vehicles emanating gps interference," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 621–632, ACM, 2014.

[13] I. Kraemer, P. Dykta, R. Bauernfeind, and B. Eissfeller, "Android gps jammer localizer application based on c/n0 measurements and pedestrian dead reckoning," in *Proceedings of the 25th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION/GNSS 2012)*, pp. 3154–3162, 2012.

[14] E. Kaplan and C. Hegarty, *Understanding GPS: principles and applications.* Artech house, 2005.

[15] B. Hofmann-Wellenhof, H. Lichtenegger, and E. Wasle, *GNSS–global navigation satellite systems: GPS, GLONASS, Galileo, and more.* Springer Science & Business Media, 2007.

[16] P. D. Groves, *Principles of GNSS, inertial, and multisensor integrated navigation systems.* Artech house, 2013.

[17] European Commission, "Galileo goes live!." `https://web.archive.org/web/20170905235145/http://europa.eu/rapid/press-release_IP-16-4366_en.htm`, 2016.

[18] J. Shen, "Update on beidou navigation satellite system (bds)." `https://web.archive.org/web/20170513173123/www.gps.gov/cgsic/meetings/2016/shen.pdf`, 2016.

[19] V. A. Dubendorf, *Wireless data technologies.* John Wiley & Sons, 2003.

[20] J. Gross and T. E. Humphreys, "Gnss spoofing, jamming, and multipath interference classification using a maximum-likelihood multi-tap multipath estimator," *ION ITM, Monterey Jan*, 2017.

[21] A. Roxin, J. Gaber, M. Wack, and A. Nait-Sidi-Moh, "Survey of wireless geolocation techniques," in *Globecom Workshops, 2007 IEEE*, pp. 1–9, IEEE, 2007.

[22] B. Jackson, S. Wang, and R. Inkol, "Emitter geolocation estimation using power difference of arrival," *Defence R&D Canada Technical Report DRDC Ottawa TR*, vol. 40, 2011.

[23] M. Khider and Google, “GNSSLogger.” https://web.archive.org/web/20170910000954/https://github.com/google/gps-measurement-tools, 2017.

[24] Google, “Google privacy policy.” https://web.archive.org/web/20170909001548/https://www.google.com/policies/privacy/, 2017.

[25] Computer Crime and Intellectual Property Section, U.S. Department of Justice and FBI’s Cellular Analysis Survey Team, “Retention periods of major cellular service providers.” https://web.archive.org/web/20170909235410/http://www.irisinvestigations.com/wordpress/wp-content/uploads/2016/03/Telephone-Carrier-Information-Retention-Schedule-Mar-2011.pdf, 2011.

[26] Bidouille, “Android version distribution history.” https://web.archive.org/web/20170910233737/https://www.bidouille.org/misc/androidcharts, 2017.

[27] Colorado Center for Astrodynamics Research, “Sige gn3s sampler.”

[28] Jacob Poushter, Pew Research Center, “Smartphone ownership and internet usage continues to climb in emerging economies.” https://web.archive.org/web/20170911010238/http://www.pewglobal.org/2016/02/22/smartphone-ownership-and-internet-usage-continues-to-climb-in-emerging-economi 2017.

[29] Newzoo, “Top 50 countries by smartphone users and penetration.” https://web.archive.org/web/20170911010242/https://newzoo.com/insights/rankings/top-50-countries-by-smartphone-penetration-and-users/, 2017.

[30] United States Census Bureau, “U. s. census bureau quick facts: United states.” https://web.archive.org/web/20170911011304/https://www.census.gov/quickfacts/fact/table/US/PST045216, 2016.