...SUPERFREQ...

**Abstract—The use of SDRs is not trivial and the learning curve along with the time overhead required to retrieve useful data is a deterrent to users regardless of their skill level. Even a basic FM radio application requires the user to boot to a live image, configure drivers, or install specialized operating systems, requiring hours of troubleshooting and configuring. Current existing SDR packages do not offer a comprehensive solution and most usable features are scattered over a variety of platforms. While modern hardware has made strides towards reducing the size and price of SDRs, current software in SDR space is underdeveloped and is not simple to deploy or use. Because of this, we developed a user-friendly package, called SUPERFREQ, mainly composed of two parts. Firstly, we use HackRF, a device used to intercept wide range of frequencies, to be the terminology, and create an easily distributable package that allows for collection and display of wireless SSID, encryption, and signal information on 2.4GHz and 5GHz frequency bands along with collection of Bluetooth information on the 2.4GHz frequency band (collection will be done from an antenna and its base converter unit). The package, which received can be easily translated onto webpage. In addition, the application will decode additional frequency bands including but not limited to Zigbee, LTE, or GSM. This stage will achieve full functionality of hardware identification through obtaining broadcasted identity information such as MAC addresses or hardware model. Secondly, the software package includes a command line interface and web GUI tools to allow for user interaction and manipulation of data received from the Antenna. All tools do not require a high level of technical knowledge, and software can be compatible with different distributions on Linux (Windows and macOS are out of scope on this objective due to high learning costs) and be able to setup and maintain its own database saved to a user's hard drive, so we can improve the usability of software. Facts have proved that SUPERFREQ has well addressed previous issues by allowing everyday users, developers, or security professionals access to a single platform which produces actionable information using a SDR.**

I. INTRODUCTION

A **Software Defined Radio (SDR)** is a dynamic piece of hardware that uses a general CPU and software to interpret and decode digital signals on a range of frequencies. SDR uses digital signal processing technology in the programmable control of the common hardware platform, the use of software to define the various functions of the radio station: including the front-end reception, IF processing and signal baseband processing and so on. It means the entire radio from high-frequency, IF, baseband until control protocol completely by software programming to complete. Its core idea is to use broadband "digital / analog" converter as close to the antenna as possible, and to digitize the signal as early as possible so that the function of the radio station is defined and implemented as much as possible in software. In short, SDR is a brand new wireless communication architecture based on digital signal processing (DSP) chip and software. For a Software Defined Radio(SDR) system to be useful as an adaptable future-proof solution, and to cover both existing and emerging standards, it is required to have elements of reconfigurability, intelligence and software programmable hardware. In addition, the emerging user requirements on reconfigurable mobile systems and networks are paving the way for the introduction of reconfigurability in future mobile systems.

The use of SDRs is a relatively new concept. The idea started in the 1980’s, but was not fully realized until the 90’s. Many see Joseph Mitola as the founding father of the field, starting with publication of his papers *Software Radios: Survey, Critical Evaluation and Future Directions (1993)* [1] and *The Software Radio Architecture (1995)* [2]. Concurrent to Mitola’s work, the Department of Defense developed the SPEAKeasy project, designed to bring all of the different radios used onto one device [3]. Later in the same decade, the concept was picked up and developed by the newly created SDR Forum, who has continued progress to this day [4]. In 2001 (when the SDR architecture was first patented [12]) Eric Blossom created the GNU Radio Companion to help make common SDRs more accessible to dedicated programmers which was followed in 2004 by the first commercially approved SDR [11], the Anywave, used by cellphone carriers. Mitola continues to drive the field forward [9] and in the past few years, many more SDRs have become available allowing anyone with an interest to acquire the necessary hardware.

Every step of the way, SDR technology has moved towards ease of use and this project continues to push that boundary. The first big step in the field towards ease of access for SDRs was the release of GNURadio [5] which moved developers from pure hardware programming to a simple user interface. In addition to project development software, many people have developed individual projects like BLE Dump [6] (packet capture using a SDR for Bluetooth), BTLE Decoder [7] (packet capture using RTL-SDR for NRF24 and BTLE), and the RFSec Tool Kit [8] (a collection of RF tools). Furthermore, progress is being made to simplify and collect the underlying protocols in a reusable library for SDR development [10]. While these other projects contribute to the utility of SDRs, they are still hosted on version control servers and require the user to configure the underlying systems to enable their use.

However, current existing SDR packages do not offer a comprehensive solution and most usable features are scattered over a variety of platforms. While modern hardware has made strides towards reducing the size and price of SDRs, current software in SDR space is underdeveloped and is not simple to deploy or use.In this paper, our main goal is to make SDR software user friendly, users can easily capture network data on wide range of wireless protocols and frequencies, allow for various implementations of capturing/storing data on different systems

II. BACKGROUND

To achieve above aims, we need import some background. In this section, we lay out some parts of background information of our research, including hardware, software, database, and so on. In these knowledge backgrounds, we can continue to carry out our work.

1. GNU Radio

The radio industry is gradually being monopolized by giants such as Agilent Technologies and Rohde & Schwarz, who create pieces of instruments one after another, stack up yet another pile of incomprehensible communications standards and put good functions split into one option after another to sell for money. As early as the programmers can have access to some command lines to understand some of the internal principles of the computer, but now programmers has long been 360 and other companies surrounded by the taciturn, so you have no time nor power to understand any principle of the computer and the Internet.

GNU Radio is a representative project for software radios in the open source world. Its emergence has enabled the open source world to break the monopoly of traditional communications giants and allowed people to freely understand any details of the entire communications system.

GNU Radio implements most of the modules required for software radios, completes the buffering and scheduling of sampled data streams, and is collectively maintained by the open source community. It is worth mentioning that, GNU Radio is different from MATLAB and other tools designed to simulate, it is born to be ready to play true, GNU Radio software is very comprehensive for RF front-end hardware support, such as USRP, HackRF, BladeRF and so on.

GNU Radio performs all the signal processing. You can use it to write applications to receive data out of digital streams or to push data into digital streams, which is then transmitted using hardware. GNU Radio has filters, channel codes, synchronization elements, equalizers, demodulators, coders, decoders, and many other elements (in the GNU Radio jargon, we call these elements blocks) which are typically found in radio systems. More importantly, it includes a method of connecting these blocks and then manages how data is passed from one block to another. Extending GNU Radio is also quite easy; if you find a specific block that is missing, you can quickly create and add it.

Since GNU Radio is software, it can only handle digital data. Usually, complex baseband samples are the input data type for receivers and the output data type for transmitters. Analog hardware is then used to shift the signal to the desired center frequency. That requirement aside, any data type can be passed from one block to another - be it bits, bytes, vectors, bursts or more complex data types.

GNU Radio applications are primarily written using the Python programming language, while the supplied, performance-critical signal processing path is implemented in C++ using processor floating point extensions, where available. Thus, the developer is able to implement real-time, high-throughput radio systems in a simple-to-use, rapid-application-development environment.

1. HackRF

As shown above, GNU Radio software is very comprehensive for RF front-end hardware support, such as USRP, HackRF, BladeRF and so on. USRP B2X0 series is the most expensive and the best in performance; the BladeRF support band is not so wide, but the advantage is that it can be run offline; the HackRF is cheap, open, particularly worth mentioning is that , HackRF is a fully open source software RF front-end. It from the schematic to the PCB map, from the driver to the microcontroller firmware, and even the processing board required process requirements, all released without reservation GPL agreement, which is undoubtedly a very fortunate thing for our study of software radio. So in our paper, we use HackRF One, shown in the figure.

HackRF is an open source hardware project to build a Software Defined Radio (SDR) peripheral. It has a few advantages:

1. Wide Operating Frequency Range

HackRF operates from 30 MHz to 6 GHz, a wider range than any SDR peripheral available today.  This range includes the frequencies used by most of the digital radio systems on Earth.  It can operate at even lower frequencies in the MF and HF bands when paired with the Ham It Up RF upconverter.

1. Transceiver

HackRF can be used to transmit or receive radio signals.  It operates in half-duplex mode: it can transmit or receive but can't do both at the same time.  However, full-duplex operation is possible if you use two HackRF devices.

1. Portable

You don't have to carry an external power supply with you when taking HackRF on the road because it is powered by USB. It is small enough to fit easily into a typical laptop bag. Your HackRF will be slightly smaller than the beta unit pictured above and will protected by a full enclosure.

HackRF is designed primarily for use with a USB-attached host computer, but it can also be used for stand-alone applications with Jared's HackRF PortaPack, an add-on that gives HackRF an LCD screen, directional buttons, and audio ports.

1. Wideband

The maximum bandwidth of HackRF is 20 MHz, about 10 times the bandwidth of TV tuner dongles popular for SDR. That means that HackRF could be used for high speed digital radio applications such as LTE or 802.11g.

1. Open Source

The most important goal of the HackRF project is to produce an open source design for a widely useful SDR peripheral. All hardware designs and software source code are available under an open source license. The hardware designs are produced in KiCad, an open source electronic design automation tool.



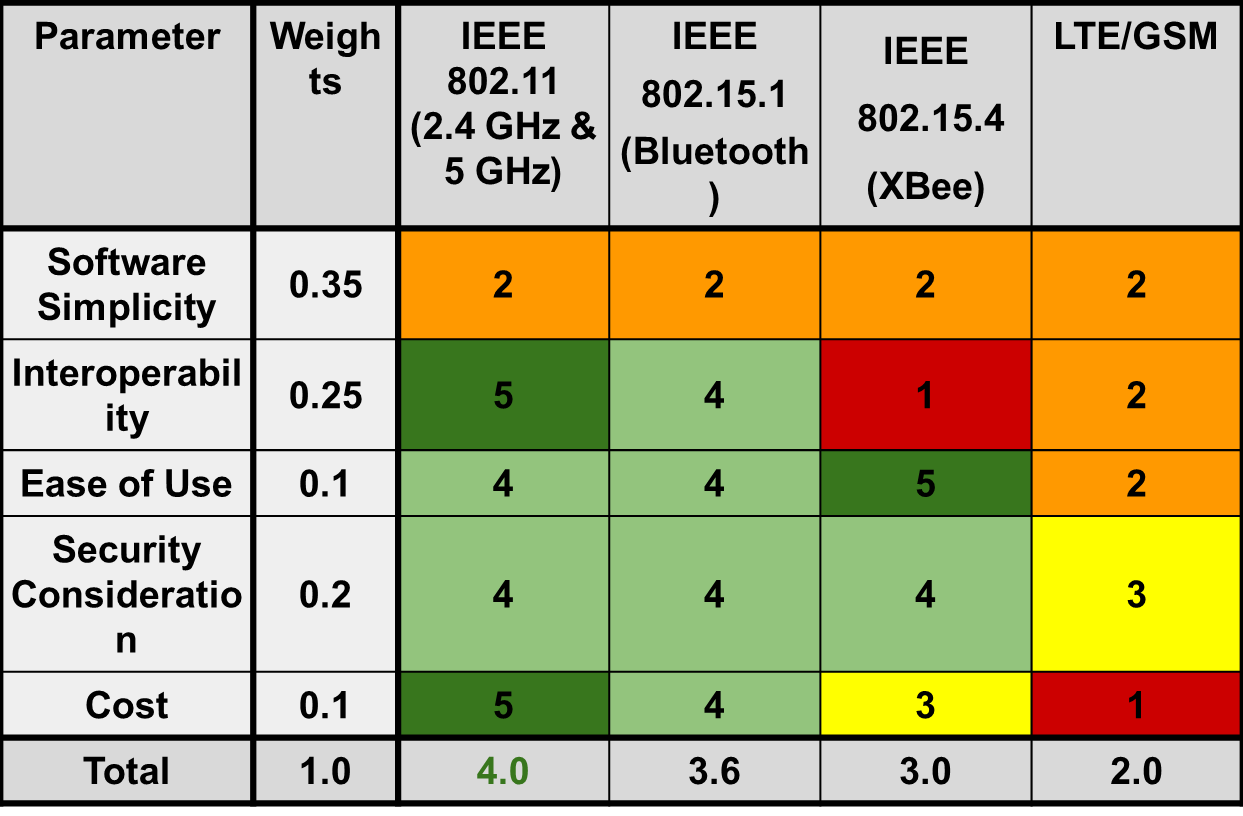
1. SQLite

In order to meet the needs of users, we pick up a special database ---- SQLite, a lightweight database, is a relational database management system that adheres to ACID and is contained in a relatively small C library. It is a public domain project created by D. Richard Hipp. Its design goal is embedded, and it has been used in many embedded products so far. It occupies very low resources. In embedded devices, it may only need a few hundred K of memory. It can support mainstream operating systems such as Windows / Linux / Unix, and can be combined with many programming languages, such as Tcl, C #, PHP, Java, and ODBC interfaces. Compared with MySQL and PostgreSQL, the world's leading database management system, it's faster than they are.

The other reason why we use SQLite, is Python has built-in SQLite3, so using SQLite in Python does not require anything to be installed for immediate use. Python defines a set of APIs that manipulate the database. Any database that connects to Python needs only to be provided with Python-compliant database drivers. Since the SQLite driver is built into the Python standard library, we can manipulate the SQLite database directly.

1. Protocol Capture

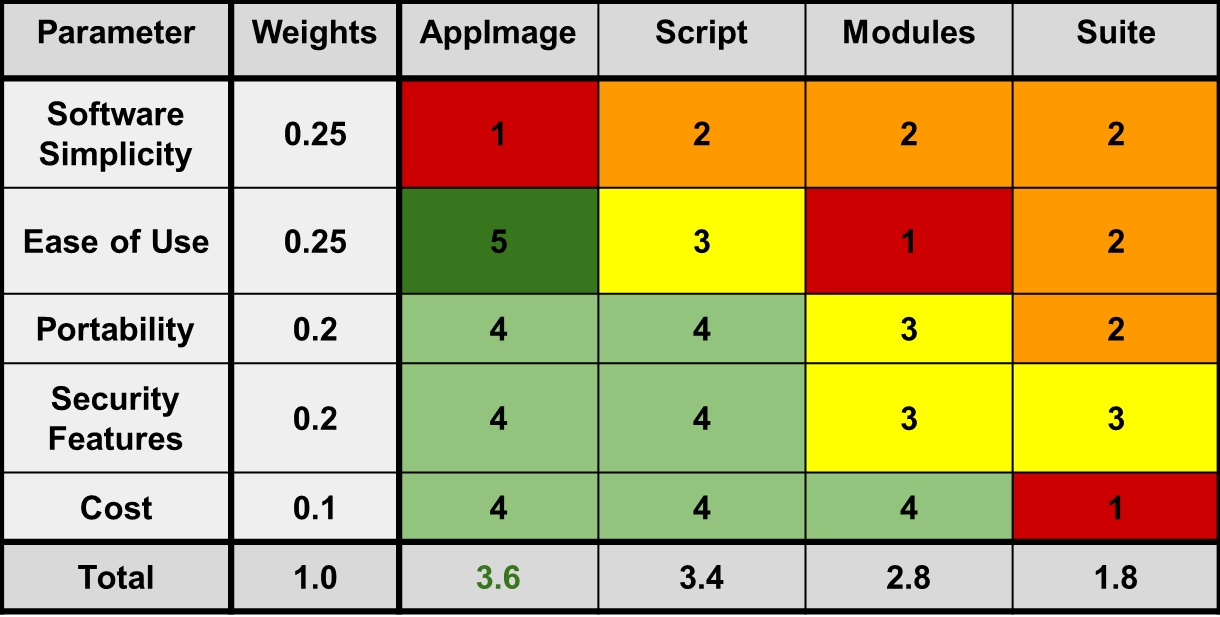
We do some research about protocol, we select the largest power among IEEE 802.11, IEEE802.15.1, IEEE 802.15.4 and LTE/GSM. We set five parameters to evaluate performance, the result is shown in the figure.



In the result, IEEE 802.11 had the best performance. But in this paper, all of these protocols will be accessible.

1. App Packaging

Because our application is to be used by our customers, we need an easy and cost-effective application packaging. We compare several application packaging tools on the market in five ways, as shown in the following figure, from software simplicity, ease of use, portability, security features and cost, each with a different weight.

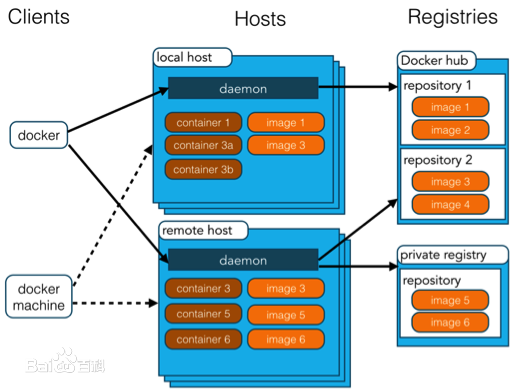


As can be seen from the figure, AppImage has the highest scores and performs better in almost all aspects. The AppImage format is a format for packaging applications in a way that allows them to run on a variety of different target systems (base operating systems, distributions) without further modification. It has many advantages: no unpacking or installation necessary, applications packaged as an AppImage can run on many distributions (including Ubuntu, Fedora, openSUSE, CentOS, Linux Mint, and others), and works out of the box, no installation of runtimes needed.

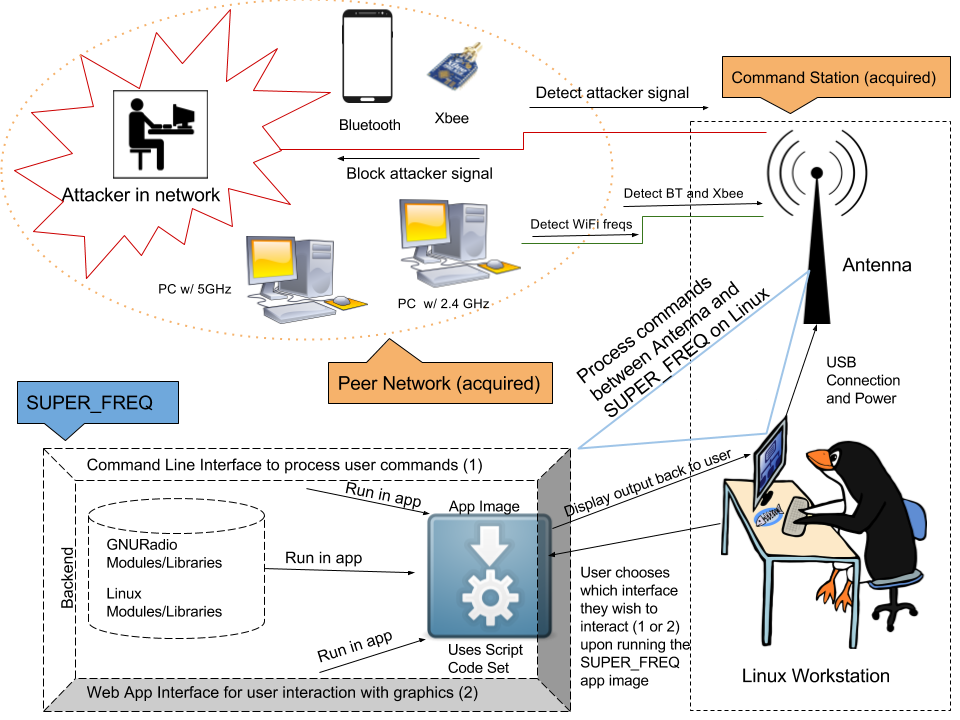
1. Docker

In addition to the project requires hardware and software support, but also need a container. Here we use docker as an application container engine. Docker is an open source application container engine that enables developers to package their applications and dependencies into a portable container and then publish to any popular Linux machine for virtualization. Containers are completely sandboxed and do not have any interface to each other.

A complete Docker has the following components: DockerClient, Docker Daemon, Docker Image, DockerContainer. The working principle as shown. Docker uses the client-server (C / S) architectural paradigm, and manages and creates Docker containers using remote APIs. Docker containers are created from Docker images. The relationship between containers and mirroring is similar to objects and classes in object-oriented programming. Docker takes a C / S architecture Docker daemon as a server to accept requests from clients and process those requests (create, run, dispatch containers). Clients and servers can run on either a single machine or through a socket or RESTful API.

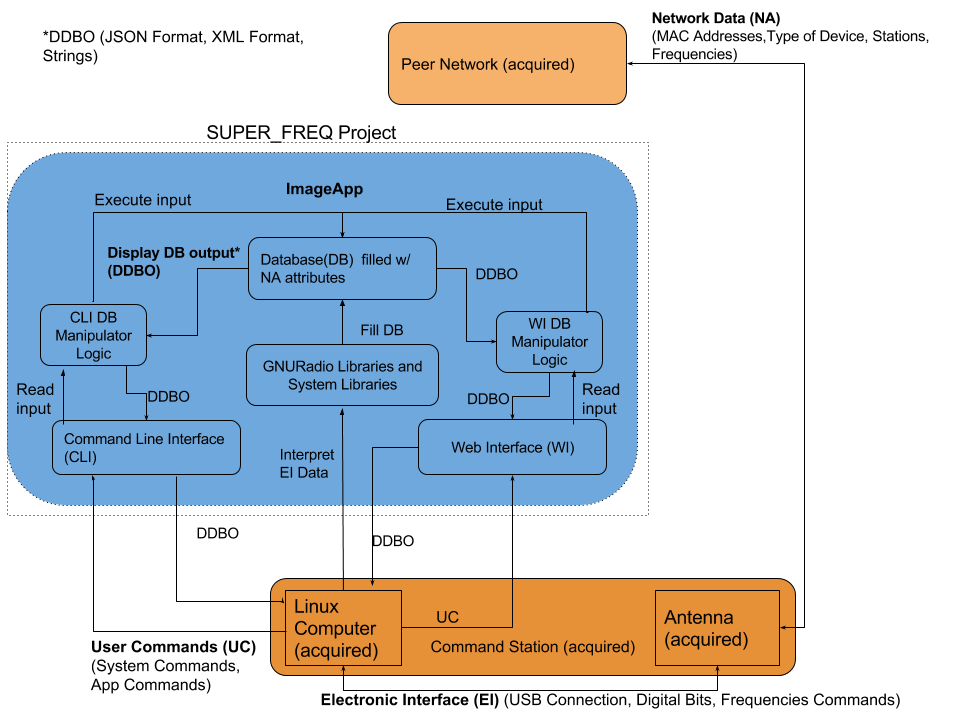


III. SUPERFREQ: DESIGN



*SUPERFREQ is a package which drives the decoding and manipulation of data packets on a Linux machine obtained from networks it resides in or near.*

**Figure 1: Concept of Operations (CONOPS)**



**Figure 2: Functional Block Diagram**

IV. RESULTS & DISCUSSION

* Results for methods
* Goals achieved?
* Contribution to exsiting research
* Significance for results

V. CONCLUSION

* Summary for objectives
* Major findings
* Implications of findings
* Scope for future

REFERENCES

1. Lackey, Raymond J., and Donald W. Upmal. "Speakeasy: The Military Software Radio." *IEEE Communications Magazine* (1995): Print.
2. Mitola, Joseph, III. "The Software Radio Architecture." *IEEE Communications Magazine* (1995): Print.
3. Mitola, Joseph, III. "Software Radios: Survey, Critical Evaluation and Future Directions." *IEEE AES Systems Magazine* (1993): Print.
4. Nutaq.com. “A Short History of Software Defined Radio Technology.” *Nutaq*, [www.nutaq.com/blog/short-history-software-defined-radio-sdr-technology](http://www.nutaq.com/blog/short-history-software-defined-radio-sdr-technology).
5. “GNU Radio.” *GNU Radio*, gnuradio.org.
6. *SDR Bluetooth LE Dumper*, drtyhlpr, github.com/drtyhlpr/ble\_dump.
7. *NRF24-BTLE-Decoder*, Omri Iluz & Tobias Schramm, [www.github.com/omriiluz/NRF24-BTLE-Decoder](http://www.github.com/omriiluz/NRF24-BTLE-Decoder).
8. *RFSEC-ToolKit*, cn0xroot, github.com/cn0xroot/RFSec-ToolKit
9. Mitola, Joseph, III. "Software Defined Radio: Challenges and Opportunities." IEEE Communications Surveys & Tutorials (Volume: 12, Issue: 4, Fourth Quarter 2010): Online.
10. Tsoeunyane, Lekhobola, et al. “Software-Defined Radio FPGA Cores: Building towards a Domain-Specific Language.” *Hindawi.com*, International Journal of Reconfigurable Computing, 17 July 2017, [www.hindawi.com/journals/ijrc/2017/3925961/](http://www.hindawi.com/journals/ijrc/2017/3925961/).
11. “Issues of Software Defined Radio Implementations for Public Safety.” FCC.gov, [www.fcc.gov/general/issues-software-defined-radio-implementations-public-safety](http://www.fcc.gov/general/issues-software-defined-radio-implementations-public-safety).
12. Ting, Pangan, et al. *Software Defined Radio (SDR) Architecture for Wireless Digital Communication Systems*. 19 Dec. 2006.