NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

WIRELESS COMMUNICATIONS FOR A MULTIPLE ROBOT SYSTEM

by

Alexander J. Bekas

March, 1997

Thesis Co-Advisors: Bert Lundy
Xiaoping Yun

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1997	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE WIRELESS COMMUNICATIONS FOR A MULTIPLE ROBOT SYSTEM	5. FUNDING NUMBERS
6. AUTHOR(S) Alexander J. Bekas	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

A multi-disciplinary research project is being undertaken at NPS to develop a semi-autonomous robotic system to detect and clear land mines and Unexploded Ordnance (UXO). The robotic system under development consists of a land vehicle, an aerial vehicle, and a ground-based control station. Reliable communication between these three stations is needed. A traditional wire-based network requires that the vehicles be tethered and severely limits the mobility of the vehicles. A wireless Local Area Network (LAN) is proposed to provide communications between the control station and the vehicles.

The objective of this thesis is to develop the physical (hardware) and logical (software) architecture of a wireless LAN that accommodates the needs of the mine/UXO project. Through an analysis of wireless modulation techniques, a market survey of wireless devices, and a field testing of wireless devices, a wireless LAN is designed to meet the technological, performance, regulation, interference, and mobility requirements of the mine/UXO project. Finally, the wireless communication protocols and the development of an error-free application protocol (specified by a FSM model and implemented in ANSI C code using Windows socket network programming) completes the wireless LAN implementation.

14. SUBJECT TERMS Wireless Local Area Networks ; Spread Spectrum modulation; Wireless MACA and MACAW protocols ; Protocol specification and verification with FSM models ; Windows sockets API	15. NUMBER OF PAGES 116
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

WIRELESS COMMUNICATIONS FOR A MULTIPLE ROBOT SYSTEM

Alexander J. Bekas
Lieutenant, Hellenic Navy
B.S., Hellenic Naval Academy, 1988

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1997

Author: [illegible]
Alexander J. Bekas

Approved by: [illegible]
Bert Lundy, Thesis Co-Advisor

[illegible]
Xiaoping Yun, Thesis Co-Advisor

[illegible]
Yutaka Kanayama, Second Reader

[illegible]
Herschel H.Loomis, Jr, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

A multi-disciplinary research project is being undertaken at NPS to develop a semi-autonomous robotic system to detect and clear land mines and Unexploded Ordnance (UXO). The robotic system under development consists of a land vehicle, an aerial vehicle, and a ground-based control station. Reliable communication between these three stations is needed. A traditional wire-based network requires that the vehicles be tethered and severely limits the mobility of the vehicles. A wireless Local Area Network (LAN) is proposed to provide communications between the control station and the vehicles.

The objective of this thesis is to develop the physical (hardware) and logical (software) architecture of a wireless LAN that accommodates the needs of the mine/UXO project. Through an analysis of wireless modulation techniques, a market survey of wireless devices, and a field testing of wireless devices, a wireless LAN is designed to meet the technological, performance, regulation, interference, and mobility requirements of the mine/UXO project. Finally, the wireless communication protocols and the development of an error-free application protocol (specified by a FSM model and implemented in ANSI C code using Windows socket network programming) completes the wireless LAN implementation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I must acknowledge the unfailing and unconditional support I have received through it all from my wonderful wife, Eleni, without which this work could never have been completed. Her positive attitude and understanding were remarkable as they were essential to my success. I am also indebted to my parents, Major-General John Bekas and Stella Bekas, for instilling in me the value of education and the desire to pursue my goals.

I also wish to express my deepest gratitude to Professors G.M. Lundy and Xiaoping Yun, whose support and guidance have been a constant inspiration to me. I would like to extend a special note of thanks for their patience and their valuable scientific guidance during the difficult phases of this work.

A special thanks goes to Professor Yutaka Kanayama for his support and his interest from the initial stages of my investigation , as well as for his insights on the reviewing of my thesis.

I. INTRODUCTION

A. BACKGROUND AND MOTIVATION

Wireless communications and mobile data applications are currently in an evolving stage. This research focuses on the design and development of a Wireless Data Network. This network will provide reliable, error free communications for the Unexploded Ordnance (UXO) /mine detection project [Ref. 9]. The UXO/mine detection and clearing project is a multi-disciplinary robotics project. The main objective of the project is to investigate and develop a semi-autonomous robot system for land mine/UXO searching and processing tasks in humanitarian operations. The proposed robot system consists of a land vehicle, an aerial vehicle, and a ground-based control station, coordinated to solve difficult tasks of mine searching and processing. The ground-based station controls and coordinates the overall operation. It also serves as a network manager for the communications among the three units. It has the ability to retrieve data, gathered by the two semi-autonomous vehicles, on demand, process the data and make decisions concerning the searching operation. The role of the land vehicle will be : detecting mines and UXOs in a small area, clearing/neutralizing mines or marking mine locations, and confirming the absence of them in an area if they do not exist. It takes remote commands concerning the search patterns (modes) and the operation in general from the ground control station and passes the search result data to the ground-based control station on demand. The aerial vehicle will perform : global surveying, assessment of terrain conditions, and guaranteeing a communication link between ground-based control station and the land vehicle for distances out of the Line Of Sight (LOS).

The objective of this thesis research is to provide transparent, reliable communications for the coordinated units. Because of the terrain topology and the nature of the UXO/mine detection tasks a wireless data link approach has been chosen. The group of the three coordinated units can be thought as a stand-alone wireless Local Area Network (LAN) composed of three stations.

B. SCOPE OF THE THESIS

To serve the thesis objectives this research is subdivided in two different, but closely related tasks.

Firstly, this research investigates the hardware organization for the wireless network that will support the UXO project. It defines the requirements that should be met and it determines the technology alternatives, products, and configurations providing a solution to this network. As with any engineering activity, the goal of the research is to find a solution that meets the desired requirements at the least cost. To accomplish this task several experiments and laboratory tests will be conducted. These tests try to imitate the true network's performance under various experimental conditions. A wireless network consisting of two wireless nodes simulates the true UXO wireless network. The AirEZY[1] wireless devices will be used as the Network's Access Points (NAP) to implement this point-to-point wireless data link. These wireless radio links use Direct Sequence Spread Spectrum (DS-SS) technology and intend to provide the wireless communications solution for the first steps of the UXO detection project. The main purpose of the experiments was to indicate which parameters are affecting network performance the most, and to verify that the existed hardware (basically the AirEZY radio links) offers proper functionality to the wireless network.

Secondly, this research develops the communication protocols that will be used by the wireless network. Especially, the application protocol will be designed, specified, and verified for error free operation. The protocol will be implemented in ANSI C code by the use of Windows sockets (Winsock) network programming interface. A **socket** is the basic building block for point-to-point communications in any network domain. Sockets are actually the endpoints for every communication path between a transmitter and a receiver. They can either be *stream - connection* oriented sockets, or *datagram - connectionless* sockets. Datagram sockets, which implement the User Datagram Protocol (UDP) as the transport entity, will provide bidirectional flow of data between the stations of the wireless

[1] Wireless data link transceivers manufactured by OTC Telecom Inc.

network that supports the UXO detection project. The UXO communications protocol consists of two Winsock applications, a client and a server, communicating via datagram (UDP) sockets.

C. THESIS ORGANIZATION

The thesis has six chapters. Chapter II gives some basic background knowledge about wireless communication networks, their limitations in achieving optimal performance and the protocols that run under this type of networks. Chapter III describes the UXO detection project requirements and indicates the proposed solutions to fulfill these requirements. Chapter IV presents the proposed wireless network protocols. The application protocol is being designed using the **Communicating Finite State Machine** (CFSM) model and verified with the **global reachability analysis** method for operation without deadlocks and unspecified receptions. Chapter V presents the code implementation of the application protocol, using the Windows Sockets network programming interface. Chapter VI concludes the thesis with a research review and suggestions for future work.

II. BACKGROUND ON WIRELESS NETWORKS

This Chapter specifies the main differences between wired and wireless LANs and briefly indicates the important properties of wireless communications. This knowledge is necessary to understand the limitations in the UXO network implementation and to provide the significance for each of the network parameters that are measured and tested in the experiments described in Chapters II and III.

A. WIRELESS LOCAL AREA NETWORKS (LANs)

In the last few years a new type of Local Area Network (LAN) has appeared, the wireless LAN. This new type of LAN provides an alternative to the traditional LANs based on twisted pair, coaxial cable, or optical fiber. Actually, the idea of wireless communications is not so new. The first attempt to merge network technologies and radio communications began in 1971 by Norm Abramson, at the University of Hawaii, as a research project called ALOHANET (funded by ARPA). The ALOHANET system enabled computer sites at seven campuses spread out over four islands to communicate with a central computer on Oahu island without using existing, unreliable, expensive phone lines. Later on the U.S. military embraced the technology and the Defense Advanced Research Projects Agency (DARPA) began testing wireless networking to support tactical communications in the battlefield. This research lead to the development of the initial Ethernet technology. However, the advent of the wired Ethernet technology steered many commercial companies away from radio-based networking components, towards the production of Ethernet related products. Recently, the need for wireless communications has reemerged and the wireless LAN market is currently in an evolving stage. Nowadays, most networking vendors develop wireless products and most computer companies scramble to develop products that support wireless connectivity methods.

Generally, the wireless LAN serves the same purpose as that of a wired or optical LAN: to convey information among the devices attached to the LAN. However, several advantages that wireless communications provide to the users make wireless network implementations more attractive to modern applications and attracts the interest of many new network investments. In general, the lack of physical cabling, to tie down the location of a node on the network, makes wireless LANs much more flexible than traditional wired LANs.

The main advantages that a wireless LAN implementation offers to the users can be summarized by the following:

1. Wireless LANs offer increased mobility to the users.
2. Wireless network's installation, is much easier than that of a traditional wired LAN, particularly in difficult-to-wire areas.
3. Installation time is also reduced. The installation of cabling is one of the most time-consuming activities in wired networks. The installation of the experimental network that models the UXO wireless network took on average about five minutes.
4. Wireless networks, under particular circumstances can also be more reliable. Cable faults, moisture metallic conductors and imperfect cable splices are some examples of problems usually occurring in wired networks. These and other problems, mainly associated with cabling, are major problems that interfere with the user's ability to utilize a wired LAN. In some cases these problems can bring a whole network down. The lack of cabling reduces these problems in wireless LANs.

On the other hand, wireless LAN implementation also faces some problems, mainly associated with the nature of radio signal propagation through the air. Only the problems that can affect the UXO's wireless network implementation are mentioned in this research. The main problems associated with a wireless LAN implementation are summarized as follows:

1. Wireless transmissions are usually very error prone. Wireless networks lose packets frequently.
2. Wireless networks have to overcome the problem of radio signal interference.
3. Wireless networks are limited by the operating distance between stations. In the wired world there is no such limitation. Usually, signals propagating through wired medium span large distances before attenuation occurs. Air waves are highly affected by phenomenon like power attenuation and fading. These phenomenon depend on the frequency content of the wireless transmission. Higher frequencies

attenuate faster, resulting smaller operating distances. Unfortunately, these frequencies carry higher bandwidth, providing a good solution when high performance is needed.

5. Wireless networks face a problem associated with the lack of standards. There exist many standards governing protocols and specifications for wired LANs, but just a few for wireless LANs. The lack of standards for wireless networks also causes a system interoperability problem. Products from one vendor will not interoperate with those from a different company. The Institute of Electrical and Electronic Engineers (IEEE) 802 working group, responsible for the development of LAN standards began operations on late eighties to develop a wireless LANs standard, the 802.11. Eventually, IEEE 802.11 working group plans to issue the final form of the 802.11 standard for wireless LANs by 1997. After that wireless LAN vendors should embrace the directions of the 802.11 standard. This research follows the principles and the rules of the latest draft of the 802.11 in the areas of frequency usage, transmission and modulation technologies and the protocols that implement the wireless protocol stack. The analysis of the UXO project needs in Chapter III is based on the 802.11 standard guidelines and on the Federal Communications Commission provisions for the wireless microwave band usage.

Wireless LANs can be implemented in two ways, depending on the user's needs and the topology of the area to be covered. They can either be connected to an existing wired LAN (i.e., connected to the backbone of an ETHERNET or FDDI LAN) as an extension, or can form the basis of a new network. Figure 1 presents the two possible configurations.

The nature of the UXO/Mine searching operations as well as the topology of the mine scattered lands suggests the later implementation. No connection with an existing network should be assumed. The network should be easy to install in the difficult-to-wire areas of the UXO project and should provide true mobility to the three stations. This implies a stand alone wireless network configuration as seen in Figure 2.

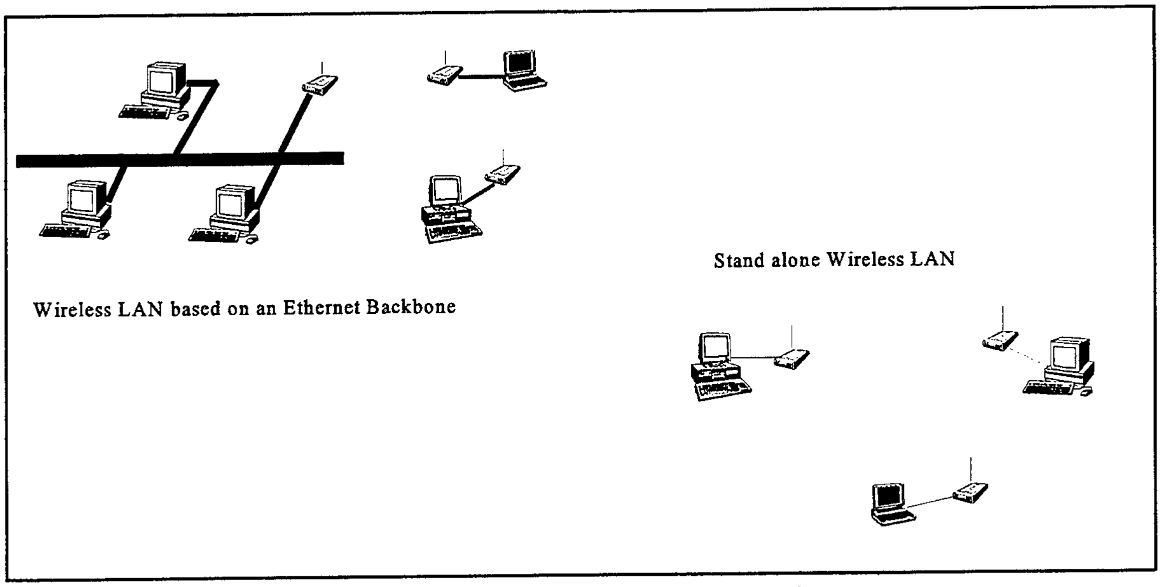


Figure 1 : Wireless LAN configurations

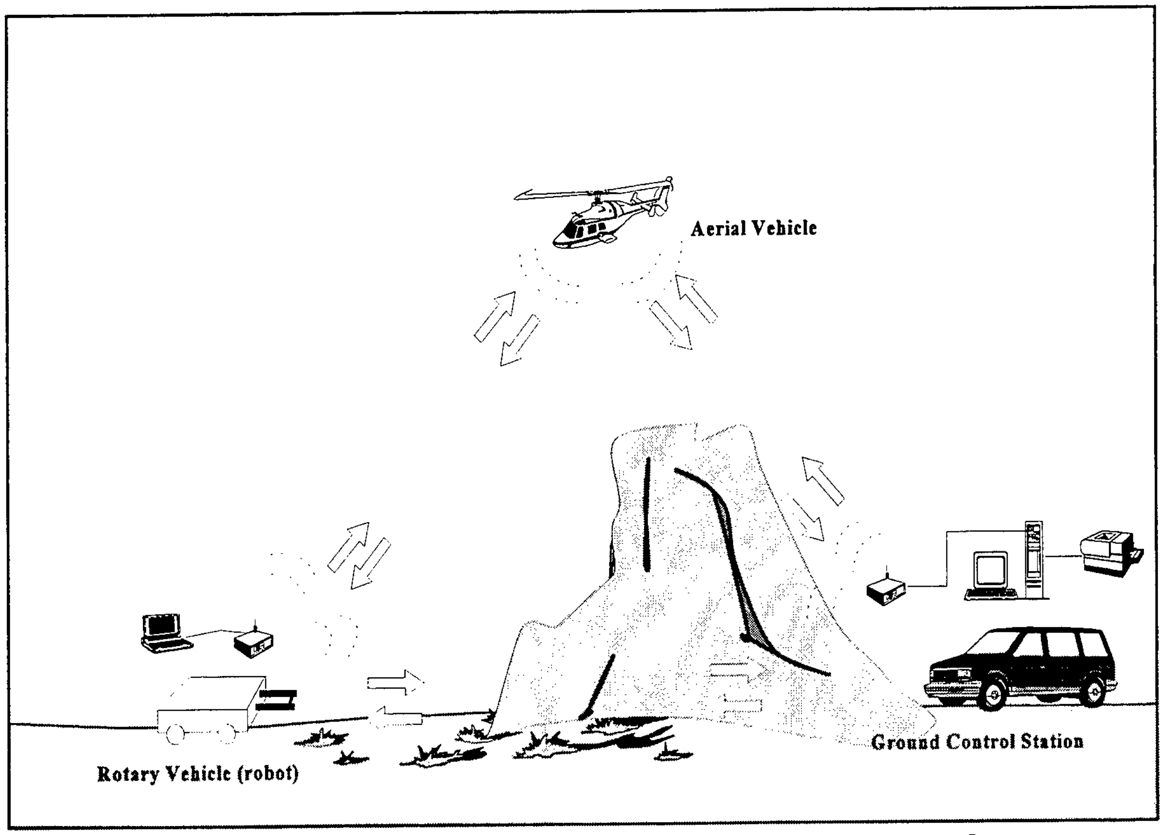


Figure 2 : The UXO detection stand alone wireless network

B. WIRELESS LANs TRANSMISSION TECHNIQUES

Wireless LANs can be implemented using one of three transmission techniques: infrared, narrowband microwave, and spread spectrum. Each technique has advantages and disadvantages depending on the particular needs of the wireless network that employs it.

Infrared LANs use infrared light signals to transmit data. There are two types of infrared light LANS: diffused and point-to-point. Diffused infrared is the technology used in products like remote controls for televisions and VCRs. The difference when this technology is implemented in networks is the usage of higher power levels and the use of communications protocols to transport digital data. Communication signals are reflected off of some types of surfaces (usually the ceiling) and by which data can travel from transmitters to receivers allowing a small chance of mobility. Typical data rates of this type of networks are 1-3 Mbps. Due to geometry, diffused infrared stations are limited in separation distance, typically 30-50 ft. This range is also limited by the reduction of the reflecting surface height. Lower ceilings result in smaller operating ranges between stations. Obviously, because this technology depends on reflective surfaces diffused infrared will not operate outdoors.

The other technique that infrared LANs can use is the point-to-point installation. Here, the devices maintain direct LOS links with one another. A simple implementation interface example is the so called "point and beam" link between a computer and a printer, or other peripherals, exchanging data using a direct infrared link. A more advantageous example of this technique is the implementation of a whole infrared LAN system of computers that uses point to point links. Advanced protocols like the *token ring* (IEEE 802.5) usually regulate a fair access on the medium in such kind of LANs. One company, the InfraLAN Technology, Inc. is currently producing devices that implement this interface. The focused infrared beams can result in throughput up to 4 or 16 Mbps with these devices. This is the only wireless LAN system on the market that can support that type of performance nowadays. The system is also extremely secure for indoor environments. But again, the main disadvantage is that as with every other infrared technology, it does not accommodate mobility. Another disadvantage, for all infrared technologies is that infrared transmissions can be very easily obstructed, since light waves cannot pass through solid objects. Their wavelength is in the range of micro meters leading to a quick attenuation problem. For all these reasons this technology is not considered as a solution to the wireless network that will support the UXO project.

Narrowband microwave is another technology proposed by some vendors to implement wireless LANs. Long distance telephone carriers were first to use this technology. They used microwave towers as transmission repeaters to overcome cabling limitations. This technology is not really advantageous for LAN implementations, but it is rather useful for interconnection between LANs. Microwave dishes are used on both ends of the communication link. One disadvantage is that the dishes must be in LOS to transmit and collect the microwave signals. Another major drawback to the use of narrowband microwave is that the frequency band used requires licensing by the FCC. Once a license is granted for a particular location, that frequency band cannot be licensed to anyone else, for any purpose, within a 17.5 mile radius. These limitations prevent the use of this technology for implementing the wireless UXO project network.

The next technology discussed, Spread Spectrum, appears to be the most advantageous technology for wireless LAN implementations. This is the most widely used transmission technique nowadays. It was initially developed by the military (during World War II) to avoid jamming and eavesdropping of the radio signals, and now is being exploited for commercial and industrial purposes. As the name implies the goal in such a system is to purposely spread the spectrum of the transmitted signal over a wider range of frequencies than is required by the bandwidth of the data alone. This operation decreases the transmitted Power Spectral Density (PSD) to an extent that it is below the thermal noise level of any unfriendly receiver. Actually, the signal might look just like noise. This is in contrast to technologies using a narrow bandwidth of frequencies. In narrowband technologies, the power of the signal is concentrated in a small portion of the spectrum, which makes it easier to detect and identify the signal and perform jamming or interference operations. In order to classify a system as a spread spectrum system, we require that the system's transmitted energy occupy a bandwidth much larger than and relatively independent of the information bit rate. There exist three major methods to spread a signals spectrum : Direct Sequence Spread Spectrum (DS-SS), Frequency Hopping Spread Spectrum (FH-SS), and a hybrid Spread Spectrum consisting of some combination of DS and FH.

Direct sequence systems spread the spectrum of a modulated signal by directly modulate that signal a second time using a wideband spreading waveform. The simplest form of DS-SS employs Binary Phase Shift Keying (BPSK) as the basic modulated signal. A general expression for the BPSK waveform is:

$$\phi(t) = A\, p(t) \cos f_c \mathrm{t}\,,$$

where $p(t)$ is a binary switching function with possible states ± 1. This signal is our message signal containing the data bits we like to transmit. The data bit rate f_b of the BPSK signal is $1/T_b$. The PSD of this signal is given by the following equation:

$$S_{BPSK}(f) = \frac{A^2 T_b}{2} [sinc^2 [(f + f_c)T_b] + sinc^2 [(f - f_c)T_b]]$$

This signal is modulated again by multiplication with a spreading waveform $c(t)$. The resulting DS-SS signal is:

$$\chi(t) = A\, c(t)\, p(t) \cos f_c \mathrm{t}\,,$$

where $c(t)$ is the spreading waveform. A common choice[1] for $c(t)$ is that of a pseudo random noise (PN) binary (two-phase) sequence having values ± 1 usually called "chips." The number of chips within a PN code between repeating-sections of the code is called the period $\mathrm{T_{ch}}$ of this code. The resulting DS-SS signal has now a data or "chip" rate of $f_{ch} = 1/\mathrm{T_{ch}}$. If we have κ chips (κ +1 and -1 distinct values) per bit, where κ is an integer greater than one,

[1] Other two-phase sequences also exist like Gold-codes and Kasami-codes.

often called the "processing gain" of the DS system, then:

$$T_b = \kappa \, T_{ch}$$

What appears as a multiplication, of the BPSK and the c(t) waveform, in the time domain is actually a convolution operation of the PSDs of the two signals in the frequency domain. As a consequence of the convolution operation the bandwidth of the resulting DS-SS signal is equal to the sum of the bandwidth of the two convoluted waveforms. The PSD of the resulting DS-SS signal is:

$$S_{DS}\,(f\,) = \frac{A^2\,T_b}{2\kappa}\,[sinc^2\,[\,(f+f_c)\,\frac{T_b}{\kappa}] + sinc^2\,[\,(f-f_c)\,\frac{T_b}{\kappa}]]$$

The DS-SS operation has basically two effects on the BPSK signal:

1. It spreads the signal's null-to-null bandwidth: $\mathbf{B}_{\mathbf{nn\,DS}} = \kappa\,\mathbf{B}_{\mathbf{nn\,BPSK}}$
2. It reduces the maximum PSD level: $\mathbf{S}_{\mathbf{DS}}\,(f_c) = 1/\,\kappa\,\mathbf{S}_{\mathbf{BPSK}}\,(f_c)$

These are the two basic properties of the DS-SS modulation. To visualize these properties Figure 3 presents the PSDs of a BPSK signal and the resulting DS-SS signal when a PN sequence with three chips is used ($\kappa = 3$) for spreading. Observation of Figure 3 shows that the effect of the direct sequence modulation is to spread the bandwidth of the transmitted signal by a factor of 3, and that this spreading operation reduces the level of the PSD by a factor of 3. In actual systems the spreading factor is typically much larger than 3. The second plot presents the same quantities in a semi-log scale, as it would appear in a power spectrum analyzer. The plots where obtained using MATLAB[2] ver. 4.2c.

[2] MATLAB™ "User's Guide for Personal Computers," The MATWORKS, Inc.

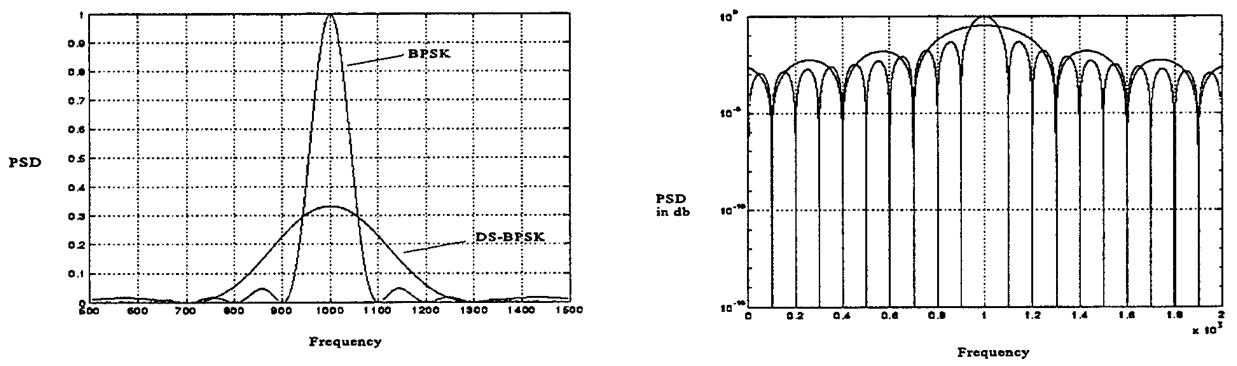


Figure 3 : The PSD spreading effect of a DS-SS system with a gain factor of 3

Phase synchronization between transmitter and receiver is assumed, not only for the BPSK waveform but also for the spreading waveform. At the receiver end, proper synchronization and multiplication of the spreading waveform, with the received signal, is called **despreading**, and is a critical function in spread spectrum systems. Interference and noise rejection in the receivers antenna is accomplished by this desreading operation. The multiplication of the received signal with the spreading code (despreading of data signal) also performs a spreading operation to the noise present to our signal. The noise and interference level is thus reduced significantly. Since the noise and interference energy is spreaded over a bandwidth much larger than the data signal's bandwidth, most of this unwanted energy can be rejected by a selective filter. [Ref. 19, 21]

In computer networks visualization of bitwise signal operation is more important. What actually happened by multiplying the BPSK modulated signal with the PN code is that each data bit of the original signal is mapped into a pattern of "chips" by the transmitter. At the receiver end the chips are mapped back into a bit, recreating the original data. This is achieved by multiplying again the incoming signal with the same spreading PN code (c(t) waveform) and with the carrier $\cos\omega_c t$. Figure 4 presents a this bitwise operation when two bits are transmitted from a station that uses a PN code with **κ =7.**

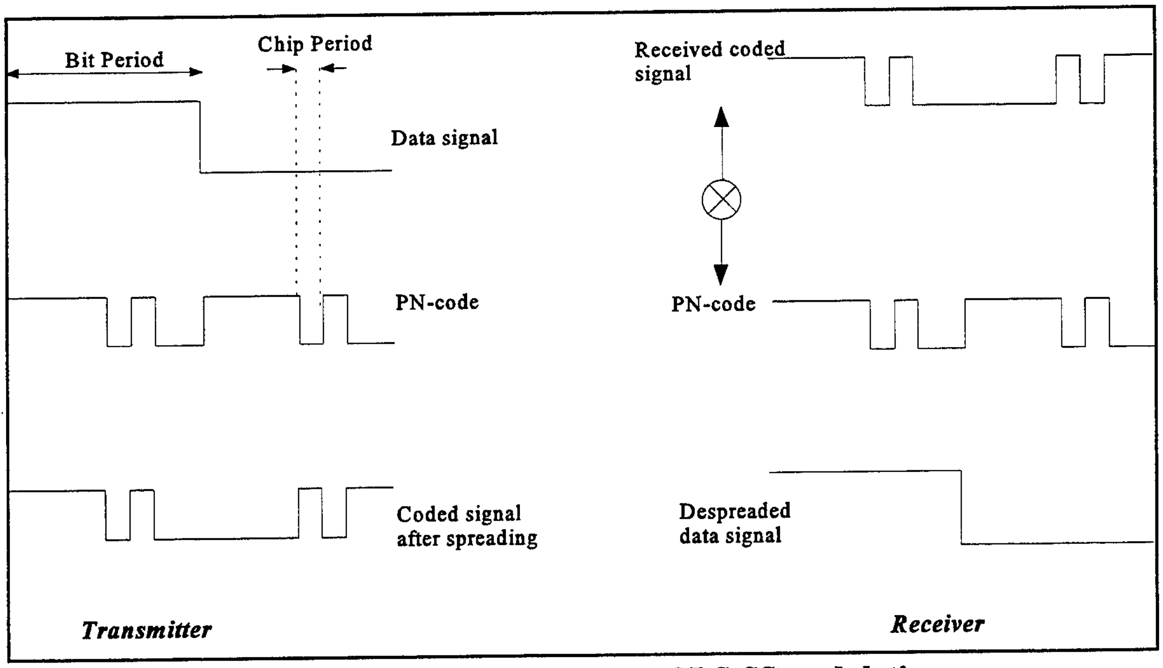


Figure 4 : Bitwise visualization of DS-SS modulation

In the most simple case a complete PN sequence is multiplied with every single data bit of the signal to be transmitted. Using a bipolar notation a binary 0 is represented as -1 and a binary 1 as a +1. Thus the PN sequence of Figure 4, represented as a sequence of chips is: +1+1+1+1-1+1-1-1. After cross correlation (multiplication) with the first information bit, which is a 1 bit, the same sequence is transmitted, and after cross correlation with a 0 bit (-1 in polar) the opposite sequence i.e: -1-1-1-1+1-1+1+1 is transmitted. In the receiver end cross correlation of the coded signal with the same PN code regenerates the bits of the original data signal. A resulting +1 means a 1 bit was transmitted, a -1 means a 0 bit was transmitted and all the irrelevant or interfering bits that give a 0 bit value as a result are just ignored by the receiver.

The PN code signal referred to as *m-sequence* in communications literature, is a noise like signal, called pseudo random because it is not actually random. Theoretically, at each equally spaced interval, a decision is made as to whether this signal should be +1 or -1. If a coin were tossed to make such a decision about 1/2 the chips will be +1 and 1/2 will be -1. However, in such a case, the receiver would not know the sequence a priori and could not properly receive the transmission. Practically, both transmitter and receiver must know the sequence. This sequence is generated electronically by a shift register sequence generator and it has certain properties to allow identification of transmissions in the receiver side. Basically, as mentioned previously, the cross correlation (normalized inner product) of any two chip sequences gives a bitwise zero and the auto correlation of a sequence with itself gives a bitwise 1. These properties suggesting a new multiplexing technique for a number of stations who want to share the same medium. With the use of different PN codes for each station, multiple channel access can be dealt with very easily. Spread spectrum systems allocate the wireless channel using the Code Division Multiple Access (CDMA) technique. CDMA is a multiplexing or medium access technique completely different from Frequency Division Multiple Access and Time Division Multiple Access. Frequency division multiplexing (FDMA) divides the channel into frequency bands and assigns it statically, or on demand, allowing indefinite use of this band to the owner. In the wireless domain, the traditional analog cellular systems, such as those based on the Advanced Mobile Phone Service (AMPS) and Total Access Communications System (TACS) standards, use the FDMA technique. In these systems only one subscriber at a time is assigned to a band of the wireless channel. Theoretically, it can hold this allocated band forever, but no other conversations can access this band until the subscriber's call is finished, or until that original call is handed off to a different channel by the system. Another common multiple access method, employed in new digital cellular systems, is TDMA. Digital standards employing this multiplexing technique are the North American Digital Cellular (IS-54), the Global System for Mobile Communications (GSM) and the Personal Digital Cellular (PDC). In these systems the channel is allocated in burst, so that each station has the entire channel dedicated for a fixed time slot. Time slots can be assigned statically or dynamically. Again, only one subscriber at a time is assigned to each time slot, or channel. No other conversations can access this channel. CDMA allows a large number of subscribers to share the entire frequency spectrum all the time. Multiple simultaneous transmissions are separated using coding theory, based on the important principles of spread spectrum communication. In a CDMA system, each user is given a distinct code sequence. This sequence identifies the user. When a receiver desires to listen to a particular's user's transmission it actually receives at its antenna not just the users transmission, but also the energy sent by all the other users that operate under the same CDMA system at that moment. However, after despreading the users signal, it will see all the energy sent by that particular user, but only a small fraction of the energies sent by other users. CDMA multiplexing initially employed for military satellite communications. Nowadays, most new wireless and cellular system implementations strive to employ CDMA technology and benefit from the advantages it provides. For the cellular telephony, CDMA technique is specified by the Telecommunications Industry Association (TIA) as "IS-95."

The other spread spectrum modulation technique is Frequency Hopping Spread Spectrum (FH-SS). The same principle, of spreading a signal's spectrum, applies for FH-SS, but it is accomplished differently. With FH-SS the spectrum of a data modulated carrier is widened by changing the frequency of the carrier periodically. As the name implies, the signal "hops" from frequency to frequency over a wide band. The duration of each hop is usually called "chip," for consistency with DS-SS. The specific order in which frequencies are occupied is a function of a code sequence (as in DS systems) of length κ. The rate of change of the carrier frequency is called the "hopping rate" f_h. Typically, each carrier frequency is chosen from a set of κ frequencies which are spaced approximately the width of the data modulation spectrum apart. The length of the spreading code is again the "processing gain factor." However, in FH-SS the spreading code does not directly modulate the data-modulated carrier but is instead used to control the sequence of carrier frequencies. The resulting bandwidth of the FH-SS signal, is κ times the bandwidth needed for the data modulation without spread spectrum. [Ref. 21]

In FH-SS systems the hopping rate is chosen independently from the bandwidth consideration. This advantage of FH systems, is not found in DS systems, and it allows separate control of the hopping (chip) rate and the bandwidth. Generally, two types of data modulation may be used by FH spread spectrum systems: M-ary frequency-shift keying

(MFSK) and binary frequency shift keying (BFSK). When binary FSK is used, the FH signal is:

$$\chi(t) = A \sin\left[\int_0^t [f_c + (\Delta f)\, p(t)]\, dt\right]$$

where Δf is the frequency shift from the carrier and $p(t)$ is the binary switching function with possible states ± 1. The carrier frequency f_c, in the above formula, is constant for an interval T_h (hopping period) and then changes to another preselected carrier frequency for the next time interval.

The transmitted PSD of a frequency hopping signal is quite different from that of a direct sequence system. The instantaneous power of the data to be transmitted (original BFSK signal) and that of the FH modulated signal (FH/BFSK) are the same (not as in DS-SS systems). However, as the signal hops around the spectrum, if we assume that it is equally likely that any hop among κ is occupied, the average PSD that an unfriendly receiver experiences in the antenna is $1/\kappa$ times the PSD of the original signal[3]. The over whole transmitted PSD does not have a $sinc^2$ ( ) shape, as in DS-SS, but is rather flat over the band of frequencies used. There are two possible FH techniques, depending on the selection of the frequency hopping rate:

1. Slow frequency hopping (SFG) is one in which $f_h \leq f_b$, where f_b is the modulated-data symbol rate. In this technique one or more data bits are transmitted within one frequency hop. An advantage of this method is that coherent data detection is possible. A disadvantage is that if one frequency hop channel is jammed or distorted, one or more data bits will be lost. So, we are forced to use error correcting codes to limit the probability of error in our transmissions.
2. Fast frequency hopping (FFH) is one in which $f_h \geq f_b$. In this technique one data bit is divided over more frequency hops. In FFH for every frequency hop a

[3] The probability that a hop is occupied is $1/\kappa$.

decision is made whether a -1 or a +1 is transmitted. At the end of each entire data bit a majority decision is made. In this case the need for error correcting codes is limited. If only a small portion of a data bit is destroyed, the entire bit can be recovered. The probability that one or more bits will be jammed is very small. Another advantage of this method is that diversity can be applied to overcome a possible system's performance degradation due to fading. A disadvantage is that coherent data detection is not possible because of phase discontinuities when fast frequency hopping is applied.

Code Division Multiple Access (CDMA) is also the multiplexing technique used by stations that employ frequency hopping spread spectrum. The main principles and the benefits gained by CDMA multiplexing are the same as those described for the direct sequence modulation.

A third method of spectrum spreading is to employ both direct sequence and frequency hopping techniques in a hybrid system. Usually fast frequency hopping is combined with direct sequence to produce extremely wide spectrum spreading results (Figure 5).

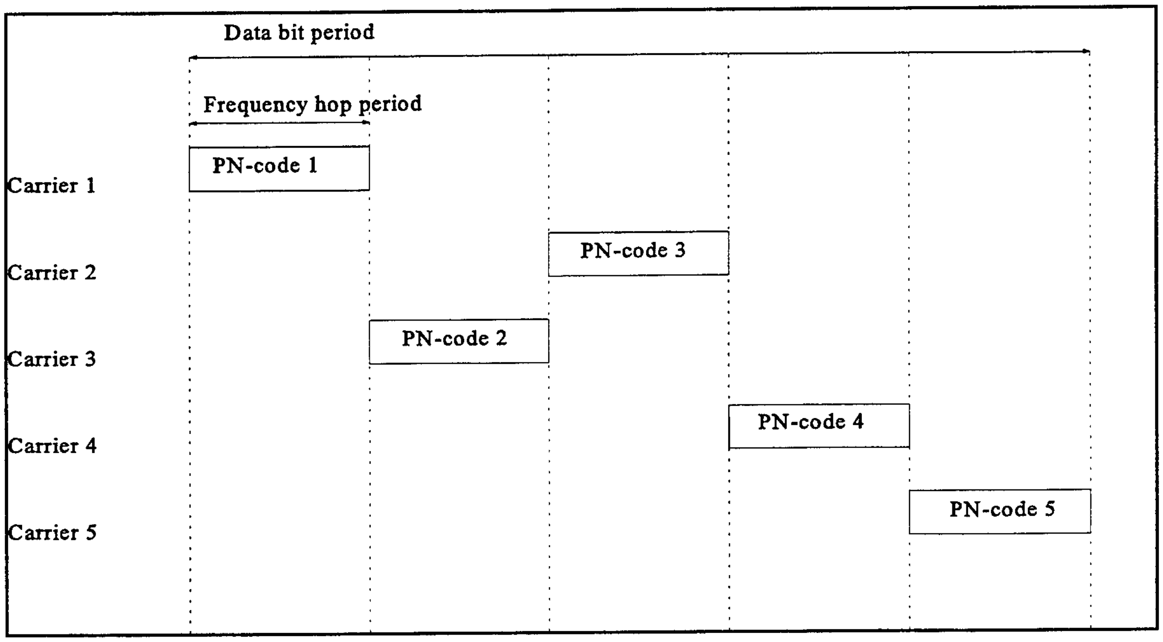


Figure 5 : A hybrid FH and DS Spread Spectrum system

Each data bit is divided over κ_1 frequency-hop channels (carrier frequencies). In each frequency-hop channel one complete PN code of length κ_2 is added to the data signal. An example of a 5-hop DS/FH system is shown in Figure 5.

As the FH sequence and the PN codes are coupled, a station's address is a combination of an one FH sequence (one carrier) and κ_1 PN codes (Figure 5). This technique combines the advantages of both direct sequence and frequency hopping techniques.

Spread spectrum, in either form, is the technology proposed by this research for the UXO detection network implementation. Summarizing the properties of the spread spectrum modulation technique, the following constitute the benefits gained by using this technique in a communications system implementation:

1. As the signal is spread over a large frequency band, the power spectral density is getting very small, so other communications systems do not suffer from systems employing spread spectrum.
2. Random access to the air-medium can be dealt with (CDMA). As a large number of spreading codes can be generated a large number of users can be permitted. However, the maximal number of users is interference limited. There is a limit to how many users one can overlay on top of one another. Each overlay decreases the Signal to Noise Ratio (SNR) slightly and thereby increases the probability of error. The phenomenon is known as "graceful degradation," and can be very critical to high data rate implementations, like ISDN. A solution to this problem is given by the FCC and other governmental agencies, that regulate the number of spread spectrum CDMA users and also provide certain restrictions in power usage. The upcoming 802.11 wireless standard includes similar provisions. Another limitation of spread spectrum technology is that the number of proper code sequences (that perform the spreading operation) is somehow limited.

   The Federal Communications Commission (FCC) and the upcoming wireless standard 802.11, have rules and provisions regulating the processing gain and other critical parameters affecting performance in spread spectrum systems. These concepts are studied in Chapter III.

3. Spread spectrum systems provide enhanced security. Without knowing the spreading code, it is nearly impossible to recover the transmitted data. Employing other modulation techniques suggests the use of special hardware or software components to provide security for the wireless network.
4. Spread spectrum systems provide fading rejection. Fading is a major problem for wireless transmissions. Spread spectrum systems are less susceptible to such distortions, as a large part of the spectrum is utilized.

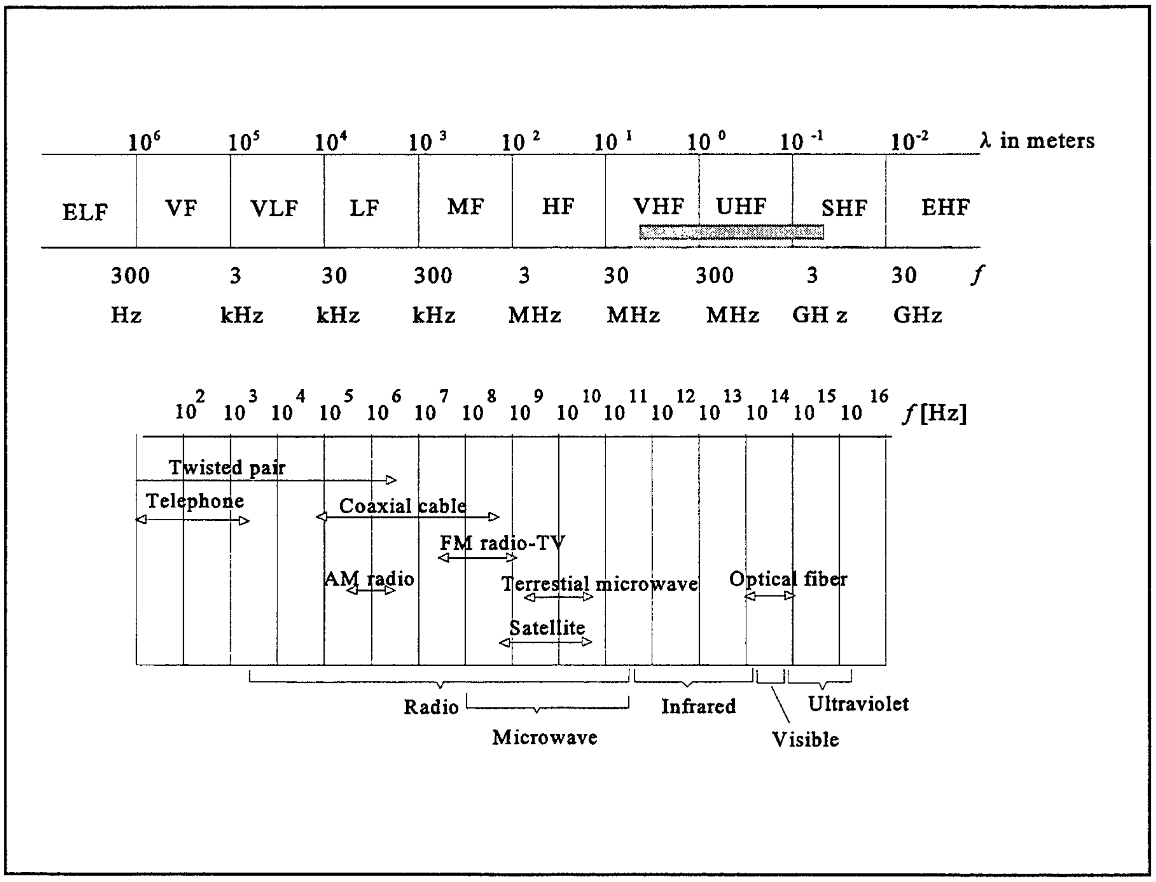
C. PROPERTIES OF WIRELESS TRANSMISSIONS

Traditional LANs, based on wired medium, deal with very low probabilities of error (below 10^{-8}) in their signal transmission. New cable fabrication techniques, especially in fiber optic lines, as well as the very well tuned protocols, that run under wired LANs, provide their users with very high quality of services. These LANs can detect and recover from bit errors very fast. Unfortunately, the same thing does not apply for the wireless medium as well. Usually, wireless transmissions are very error prone, restricting wireless LANs from providing high quality of services to the users. The errors in wireless transmissions are mainly due to the characteristics and the properties of the electromagnetic wave propagation through the air. These properties are mostly dependent on the frequency content of the air-waves.

Generally, communication literature refers to air-waves between 10^3 -10^{12} Hz as radio waves, without making a distinction in the microwave portion, which is approximately between 10^8 -10^{12} Hz. This distinction is necessary when implementing a wireless network. Certain properties of the medium used, can have a great effect on the performance of the wireless network. These properties are frequency dependent. Radio waves are usually easy to generate (simple circuitry), can travel long distances and generally propagate through walls, buildings and other obstructions with fairly little attenuation. Radio waves also have the property of omnidirectional transmission. Omnidirectional antennas (yagi-type) can enhance this property. A disadvantage resulting from the omnidirectional transmission of radio waves is that they have low transmission gain (omni- antennas have a unity gain).

Another disadvantage of radio, especially in the LF and MF band, is that because of their frequency content they can not carry enough bandwidth, for wireless LAN implementations.

The technology proposed by this research to implement the wireless UXO detection network is spread spectrum. Spread spectrum modulation, in either form, uses microwaves as the transmission medium.



10^6 10^5 10^4 10^3 10^2 10^1 10^0 10^{-1} 10^{-2} λ in meters
ELF VF VLF LF MF HF VHF UHF SHF EHF
300 Hz 3 kHz 30 kHz 300 kHz 3 MHz 30 MHz 300 MHz 3 GHz 30 GHz f
10^2 10^3 10^4 10^5 10^6 10^7 10^8 10^9 10^{10} 10^{11} 10^{12} 10^{13} 10^{14} 10^{15} 10^{16} f[Hz]
Twisted pair
Telephone
Coaxial cable
FM radio-TV
AM radio
Terrestial microwave
Optical fiber
Satellite
Radio
Microwave
Infrared
Visible
Ultraviolet

Figure 6 :The electromagnetic spectrum

Generally, as seen in Figure 6, microwaves (terrestrial and satellite) include some portions of the VHF, UHF, and SHF frequency bands [Ref. 20]. Practically, radio waves above 100 MHZ belong to the microwave portion of the spectrum. At these frequencies waves travel in straight lines and are usually narrowly focused. Unlike radio waves at lower frequencies, microwaves do not pass through obstacles so well. Microwaves in the GHz range bounce off obstacles. These waves are a few centimeters long and attenuate very fast. The signal power falls sharply with the distance from the source, and signal attenuation follows the following formula [Ref. 17]:

$$L = 10 \log \left(\frac{4\pi d}{\lambda} \right)^2 \qquad \text{dB}$$

where L is the loss (attenuation) expressed in dB, d is the distance from the source, and λ is the wavelength, in the same units as d. This formula implies that microwave loss varies as the square ($1 / d^2$ analogy) of the distance. Microwave attenuation, is also dependent on the environmental and weather conditions, covering the transmission area. Moisture environments and rainfalls increase attenuation of microwave transmissions. A general practical rule under all conditions would be roughly a $1/d^3$ dependence on the distance [Ref. 17, 20].

Transmission impairments for microwave signals, operating under constant environmental conditions, can be summarized in the following factors:

1. The impairment of **multipath fading**. Fading is a major problem in microwave communication links. Although microwave transmission is narrowly focused there is still some divergence in space. Some waves follow the direct LOS path, from the transmitter to the receiver, without any scattering. Others are scattered by a random medium. This medium, when operating outdoors, is usually the lower tropospheric inversion layers. Mobile communications suffer from these kind of fading channels. For indoors operation, several obstructions (walls and other obstacles) in the direct path can cause multipath fading.

   The phenomenon occurs when some indirect waves take slightly longer to arrive to the receivers antenna, than the direct LOS waves. These delayed waves usually arrive out of phase with the direct waves and thus cancel, or cause significant attenuation, of the signal. The effect is frequency and weather (formation of low tropospheric inversion layers) dependent. Multipath fading can be time-selective or frequency-selective. Time selective fading occurs when the scattering medium varies with time causing a variance in the fading phenomenon. Frequency selective fading assumes a fixed (nonmoving) scattering medium, but different frequencies affected differently by the scattering medium.

2. The impairment of **shadowing**. The presence of obstacles in the direct path, from a transmitter to the receiver, causing signal attenuation at the receiver's antenna. For mobile communications, shadowing results in the form of time-varying received signals, depending each time on the mobile's station and base station relative positions. The phenomenon can also be viewed as a time selective fading. The nature of the terrain surrounding the base and the mobile antennas as well as the respective antenna heights with respect to the terrain determines the extent of shadowing.
3. Another signal impairment is the **variation of signal strength**, depending on the distance between the transmitter and the receiver. For mobile communications, where relative movement is very frequent, this is a very important factor. The formula provided above, for microwave attenuation, measures the loss in dBs as the distance increases.
4. Signal impairments caused by **interference** from other electronic devices. These devices either operate in the same frequency band (microwave), or produce harmonics in the frequency band of interest. The spread spectrum modulation technique deals perfectly with this problem. However, the FCC has issued some rules and provisions concerning the usage of the microwave spectrum. These issues are studied in Chapter III.

To investigate the effect of these transmission impairments to an existing wireless system, the AirEZY wireless data link transceivers were tested. These devices provide an access point (wireless nodes) solution for wireless LAN implementations. They can either be connected directly to an Ethernet bus, to provide connectivity with an Ethernet backbone, or to a computer's (PC, MAC, laptop or workstation) Network Interface Card (NIC) through BNC or RJ-45 connectors. The main advantage, that these wireless nodes provide, is that they are platform independent. Their drivers support all major Network Operating Systems (NOS), without special software installation needed (plug and play). The AirEZY nodes utilize direct sequence spread spectrum BPSK technology, advertised to provide 1.0 Mbps throughput for distances up to 500 ft indoors and 800 ft outdoors. They utilize the I band (902-928 MHZ) of the ISM (Instrumental Scientific and Medical) bands provided by the FCC for unlicensed usage of wireless LANs. The total PF power transmission is limited to 100 mW. To investigate the impairments of microwave transmission, two AirEZY nodes were connected to the NIC of two PC's. To capture the microwave transmission a Hewlett Packard 3585 B spectrum analyzer, with an omnidirectional antenna installed, was used. Large file transfers between the two PC's allowed a continuous transmission, with fairly constant output power from the transmitting node's antenna. By increasing the distance between the spectrum analyzer's antenna and the transmitting node's antenna, the plots of the microwave transmission Power Spectral Density (in milli Watts per unit Hertz) were obtained. Figures 7, 8 and 9 are showing these plots. To capture these transmissions the PSD level of the spectrum analyzer was tuned (lowered) to a reference level of -37.2 dBm (top of plot). This means that the power spectral density at this level is 0.00019054 mw ($10^{-3.72}$). The PSD scale is 5 dB/dev. and the frequency scale, which is centered at 914.76 MHZ (carrier frequency), is 2.646 MHZ/dev. The plots are analogous to Figure 2 of current Chapter. The processing gain factor for the AirEZY wireless nodes is $\kappa = 11$, resulting in a more spreaded spectrum.

Figure 7 presents the spectrum (power spectral density versus frequency) of the AirEZY transmitter 1m away from the transmission antenna. The spectrum has a $sinc^2$ ( ) shape. The area under the curve's envelope represents the total transmitted power (PSD) that a receiver senses at this distance. This power was found, using partial integration methods, to be 82 mW.

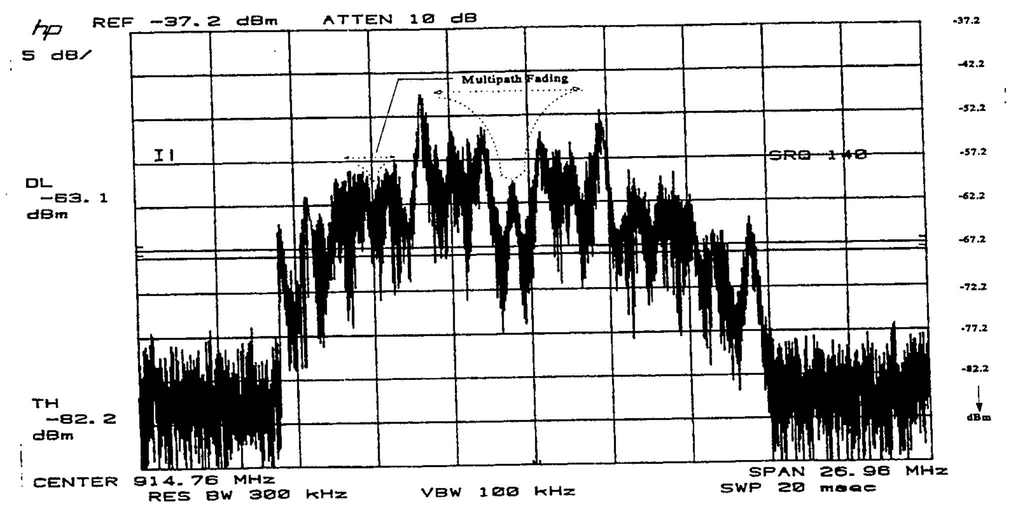


Figure 7 : The PSD of an AirEZY transmitter 1m away from the antenna

This implies an attenuation of 20 mw if we assume that the device transmitted at full power (100mW) at the antenna. Figure 7 also shows signal attenuation due to multipath fading. The phenomenon is stronger in the area around the carrier frequency (915 MHZ) implying a frequency-selective fading. This area of frequencies should give the highest power spectral density values (as in Figure 2), but instead power degradation occurs. The fade is approximately 10 dBm deep and 2 MHZ wide. A narrowband signal having bandwidth of less than 2 MHZ would be greatly attenuated due to such fading. However, spread spectrum technology makes this system relatively insensitive to fading since the power is not concentrated in this particular portion of the spectrum, but is instead spreaded over a 26 MHZ band (902-928 MHZ). Some fading can also be observed in the side lobes, but it is relatively smaller.

Figure 8 shows the signal's spectrum 2m away from the transmitter's antenna. The over whole power spectral density has now become 15 mW. This implies a power attenuation of 85 mW.

Figure 9 shows the signal's spectrum 3m away from the transmitter's antenna. The power spectral density is approximately 8 mW, implying a signal's power attenuation of almost 90 mW.

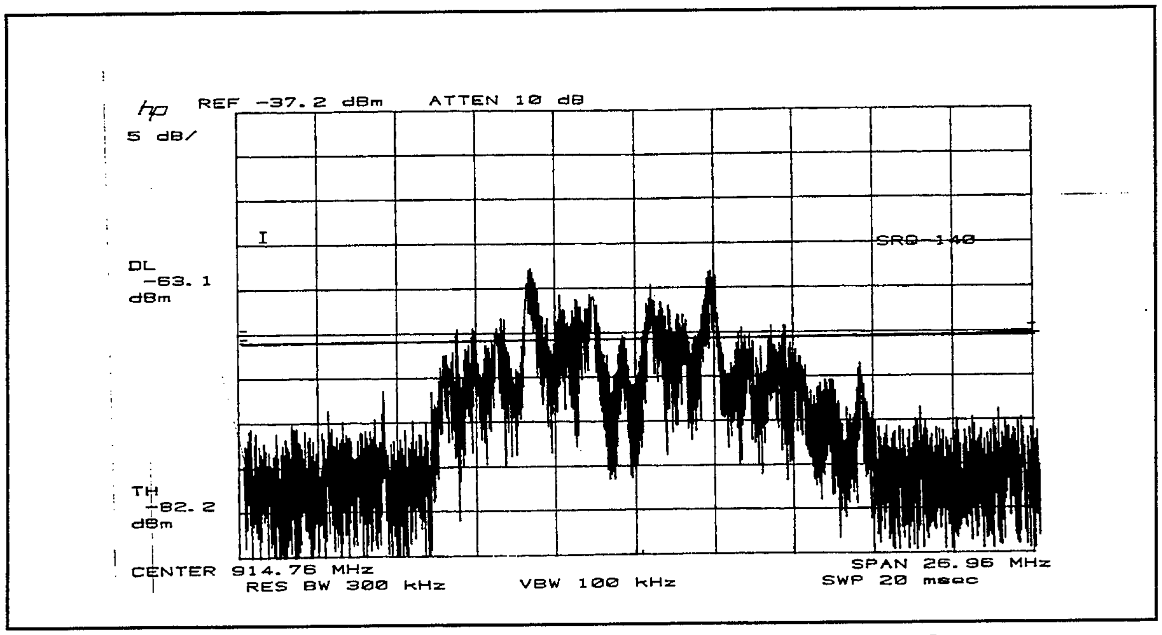


Figure 8 : The PSD of an AirEZY transmitter 2m away from the antenna

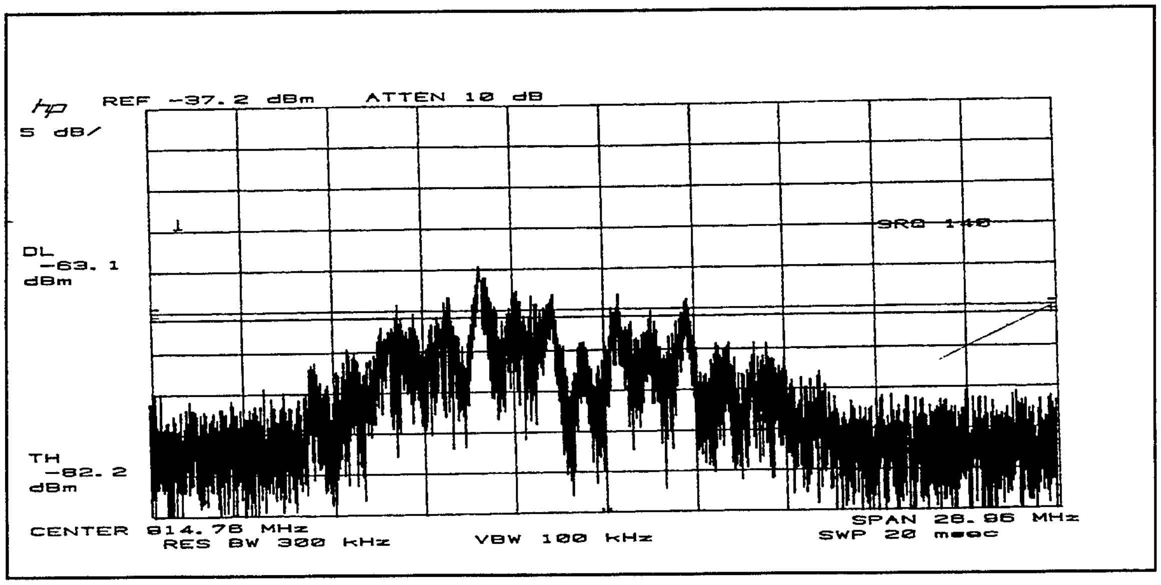


Figure 9 : The PSD of an AirEZY transmitter 3m away from the antenna

For any particular frequency selected in the 26 MHZ spectrum, the signal strength should attenuate following the loss formula of the microwaves. Selecting the carrier (915 MHZ) to be the frequency of interest, the loss formula provides:

TABLE 1 : Attenuation of the AirEZY microwave transmission with distance

Distance	Attenuation for the 915 MHZ frequency
From d_1 =1 m to d_2 =2 m	$L1-L2 = 10\log(\frac{4\pi d_1}{0.3278}) - 10\log(\frac{4\pi d_2}{0.3278}) = \mathbf{7.69}\ db_m$
From d_2 =2 m to d_3 =3 m	$L2-L3 = 10\log(\frac{4\pi d_2}{0.3278}) - 10\log(\frac{4\pi d_3}{0.3278}) = \mathbf{4}\ db_m$

The numbers provided in Table 1 are calculated by application of the theoretical formula that calculates the microwave loss. These numbers match with the practical results seen in the spectrum plots of Figures 7, 8 and 9. The same calculation for other frequencies, also gives comparable results.

III. REQUIREMENTS FOR THE UXO DETECTION WIRELESS NETWORK

Requirements are crucial in all development projects. They provide the basis for design, implementation, and support of the developed system. Requirements are usually defined based on the needs of potential users of a system. The UXO/mine detection project needs to be supported by a simple and reliable communication link, that has some similarities, but also a lot of differences from other common wireless network implementation, based on the available information about the functionality and operation of the UXO detection semi-autonomous robotic system. The following requirements are studied:

1. Hardware-technological requirements
2. Performance requirements
3. Regulation requirements
4. System interference requirements
5. Mobility requirements

A. HARDWARE-TECHNOLOGICAL REQUIREMENTS

Hardware implementation of most wireless networks is fairly simple. The basic physical and logical architecture of wireless LANs is shown in Figure 10.

The physical components of the wireless LAN implement the Physical, Data Link and Network Layer functions of the protocol stack (logical architecture). The Network Operating System (NOS) [1] supports the shared use of network applications, printers and disk space among the wireless LAN hosts. The NOS communicates with the wireless Network Interface Card (NIC) via driver software, enabling applications to utilize the wireless network for data transport. After that, the NIC prepares the data signals for transmission via the wireless node. It interfaces between the real network (wireless node and physical medium) and the user. The wireless node utilizes a specific wireless modulation technique (described in Chapter II) to

[1] Like TCP/IP, Novell NetWare, AppleTalk, Windows NT, etc.

transmit digital data through the air medium, via the antenna. The destination host is comprised of the same set of components (Figure 10).

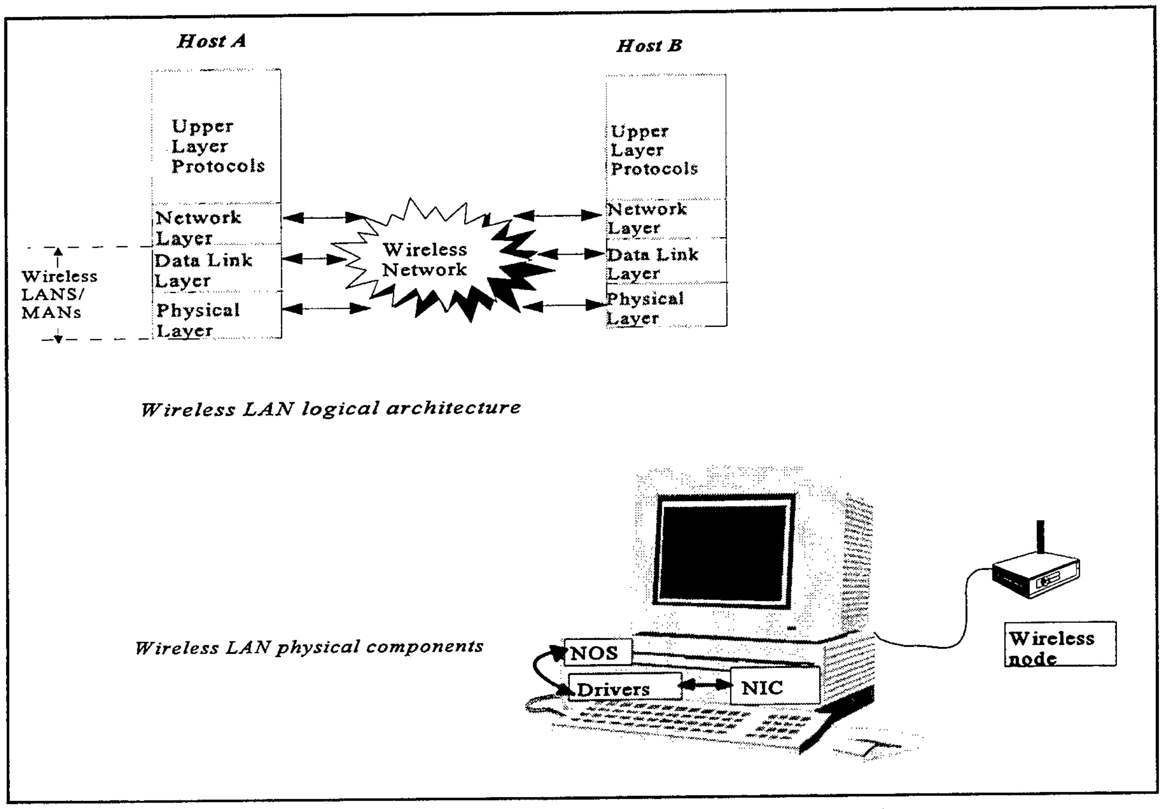


Figure 10 : Wireless LAN logical and physical architecture

For the UXO/mine detection project implementation, the ground control station uses a desktop workstation or a personal computer (PC) as the network user appliance. The protocol stack, implementing the functionality of a NOS, of choice is TCP/IP. The TCP/IP was chosen because it is the protocol stack most widely used as a NOS and it is supported by all major NICs providers. The choice of the NIC is not important, as long as it can support the desired NOS. The semi-autonomous mobile robot[2] uses a Motorola 68040 CPU configured on a TAURUS board [Ref. 14] to control robot's motion and process data relevant to the vehicle's tasks. This robot is specifically designed for UXO/mine searching

[2] Mobile robot "*Shepherd*."

tasks and it is four-wheel steerable and four-wheel drivable. It is able to traverse rough terrains, and has an independent rotational degree of freedom. Controlling robot's movement and processing the robot's sensor data are very time critical operations. Because of that, the TAURUS board's kernel is composed of ANSI C procedures written from the robot's system developers. It is not based on already existing operating systems like UNIX or MS-Windows. This minimal kernel is able to provide a 10 ms computation cycle needed to achieve smooth and satisfactory motion control and movement to the robot. Implementing the NOS in such a board would require the development of a new Network Operating System to interface with the minimal robot's kernel, since traditional NOSs are designed and specifically tuned to interface and operate on standard operating systems (OS) like MS-Windows and UNIX. Moreover, adding an NOS to the kernel's code would possibly cause more overhead in the robot's computation cycle. To overcome this problem this thesis proposes the implementation of the NOS on a laptop computer, siting on the robot's frame, interfacing with the TAURUS board via parallel (RS232) board-to-board connection (Figure 11). This laptop runs Windows'95 operating system, which implements TCP/IP, and communicates with the ground control station's terminal with out any interoperabillity problems.

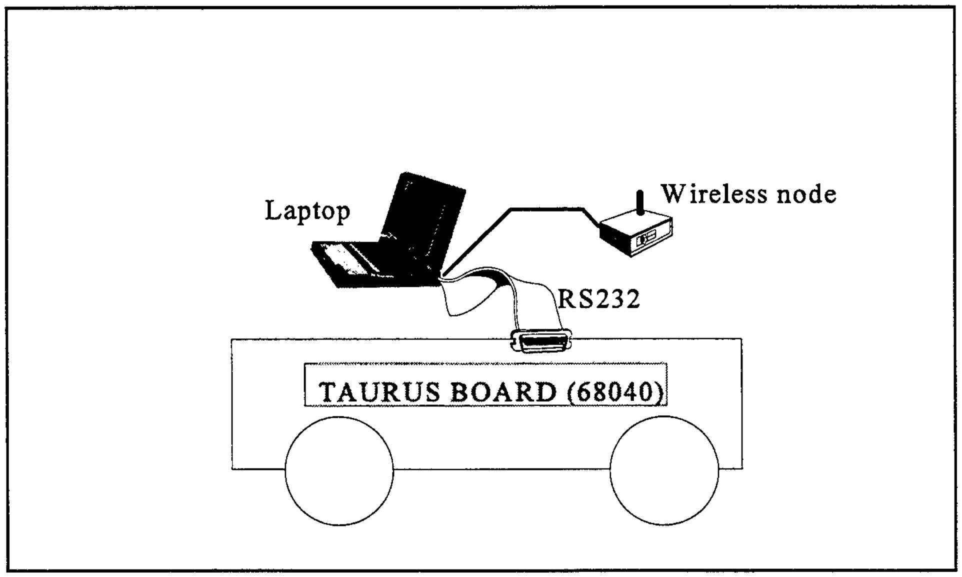


Figure 11 : Hardware configuration of the land vehicle (robot) station

The technology of choice is spread spectrum modulation (Chapter II). The challenge of choosing between direct sequence or frequency hopping spread spectrum is not critical for this project. Generally, both technologies deliver the same advantages to wireless network implementors. Only some technological specific and frequency or performance demanding applications can discriminate and make a choice of spread spectrum modulation, depending on the needs. Direct sequence systems are considered to be Low Probability of Detection (LPD) systems since the power spectral density (PSD) of such systems is very low in comparison with the original unmodullated signal to be transmitted. On the other hand, in frequency hopping systems the instantaneous PSD is the same as for conventional BFSK signals, thus these systems are not considered as LPD systems. Both, DS and FH systems are considered as Low Probability of Intercept (LPI) systems, since the spreading code is required, in each case, in order to recover their signals. The UXO project definitely needs the LPI property.

Recently many network vendors have produced wireless nodes implementing both spread spectrum technologies. The choice of the wireless products (nodes) that can support the UXO detection project is made after analysis of all the project's requirements (performance, regulations, mobility etc.) and it is not based on the modulation technique that these products implement.

B. PERFORMANCE REQUIREMENTS

Performance requirements indicate how well the wireless network provides the UXO detection project application programs[3] and services. The basic information exchange between the two major UXO search project entities, the ground control station and the mobile robot, is shown in Figure 12. Video image transfer is not included in the digital data exchanged between the two entities. Video image will be transmitted through a video camera system.

[3] Robot's motion control, search-data processing programs, robot's status information etc.

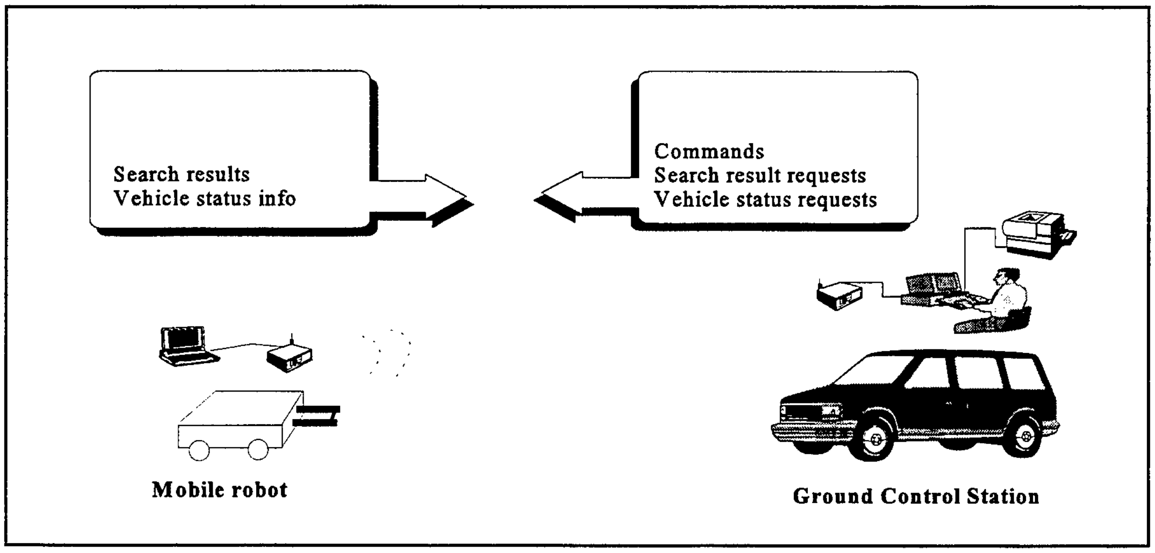


Figure 12 : Information exchange between the UXO detection stations

To measure the performance requirements for the UXO detection project, network **delay**, **reliability** and **availability** are identified [Ref. 3].

Network **delay** is the length of time the UXO detection system (mobile robot and ground control station) and its users have to wait for the delivery of the wireless network services. This is actually the network's throughput (effective data rate) measurement in Kilo bits per second (Kbps) combined with the length of the data segments transferred from one network entity to the other. Commands and requests from the ground control station are small segments just a few bytes long. Search results and robot's status information, called result and status vectors returned from the mobile robot, contained in data segments no more than 1500 bytes long (maximum Ethernet segment). TCP allows a lot larger data segments, but for the particular wireless implementation 1500 bytes segments are considered adequate. This size also results in smaller vector transfer delays enabling the application programs to process the robot's data more frequently.

Reliability is the length of time the wireless network or component will operate without malfunctions or disruption. In the network market this is referred to as Mean Time

Before Failure (MTBF). This factor is important for mine and UXO searching operations because UXO/mine detection has random occurrence and the network must be available at this particular instance.

Availability defines the period of time the wireless network must be operational. For the particular wireless network this depends on the duration and the needs of UXO detection operations. The mobile robot is powered by four 12V DC batteries giving it a 1.5 hours operational endurance (autonomy). The laptop that is connected with the vehicle's TAURUS board, sits on top of its frame and provides the NOS (TCP/IP). This laptop (Figure 11) works, if not plugged in an AC outlet, with 12 V DC batteries for 1 hour of operation. The wireless nodes used as the network access points usually need 5 V DC power. This power can be provided by the mobile robot's batteries via a DC-to-DC converter.

Identification of the significance of these performance measurement factors (delay, reliability and availability) is performed by usage and testing of the AirEzy (OTC Inc.) wireless nodes under several network and environmental conditions. Firstly the wireless devices were tested in an indoors environment to verify performance versus distance between the wireless nodes. The File Transfer Protocol WS_FTP95 (32-bit version for Windows 95 operating system) was used to transfer files from a SUN workstation to a PC laptop, and measurement of the effective data rate (throughput) in Kbps was obtained. The tests were held in Spanagel hall along the second floor's corridor. The two wireless nodes (AirEzy) were always in Line Of Sight (LOS), with their antennas pointing each other with out any intermediate obstruction. Files of different size were transferred in each distance measurement. For every distinct distance ten file transfer (WS_FTP95) operations, for each different sized file performed, for a total of 270 file transfers. The mean value of these measurements was used to construct Table 2.

TABLE 2 : Throughput versus distance for different sized files

Dist./File size	15K	50K	100K	250K	500K	1.5M	2M	4.3M	Mean
25m	921	700	660	658	650	646	636	634	**688**
55m	733	705	420	505	491	574	561	558	**569**
100m	435	446	405	410	400	391	358	383	**404**
Mean Throughput in Kbps									**553**

Figure 13 and 14 present graphically (from different view points) the relationship between effective data rate in Kbps and the distance between the wireless nodes, with the file size in bytes as a parameter.

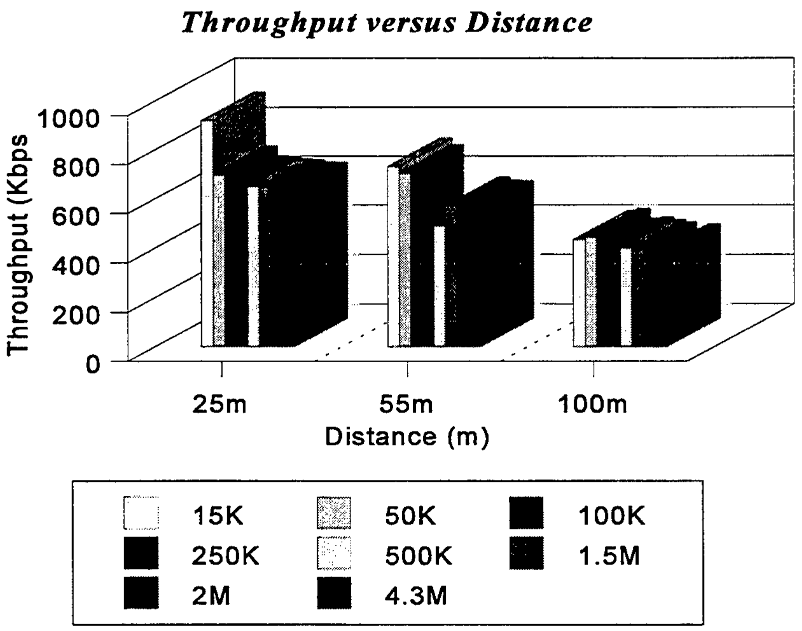


Figure 13 : Indoors throughput versus distance - I

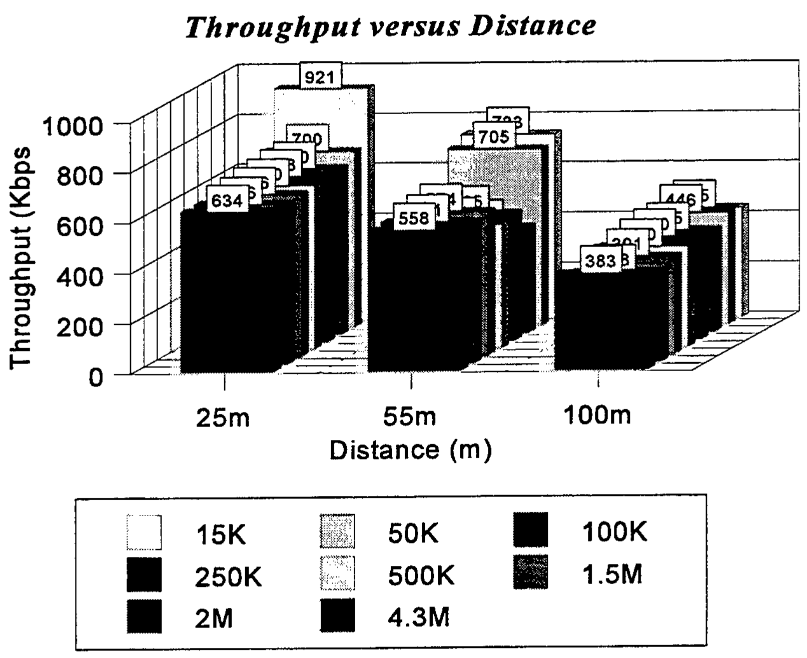


Figure 14 : Indoors throughput versus distance - II

Figures 15 and 16 present graphically (from different view points) the relationship between effective data rate in Kbps and the file size in bytes, with the distance between the nodes in meters as a parameter.

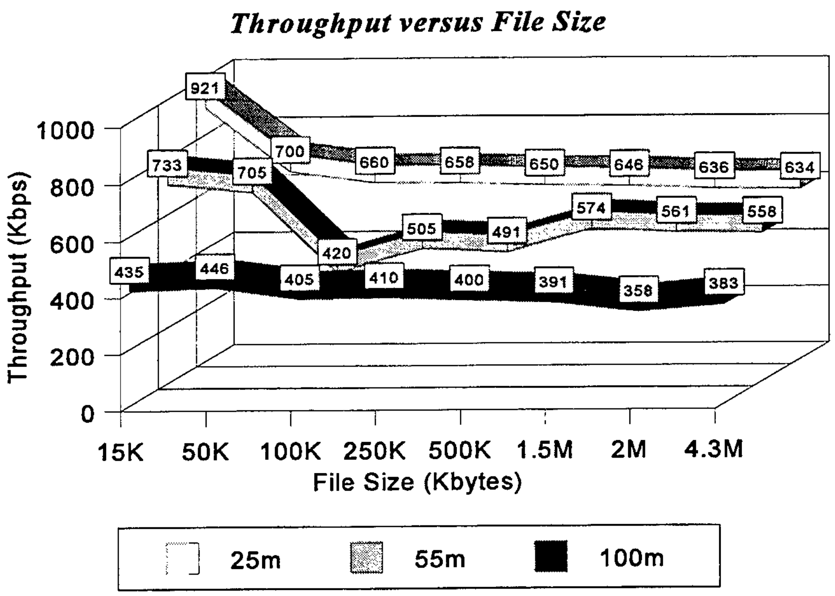


Figure 15 : Indoors throughput versus file size - I

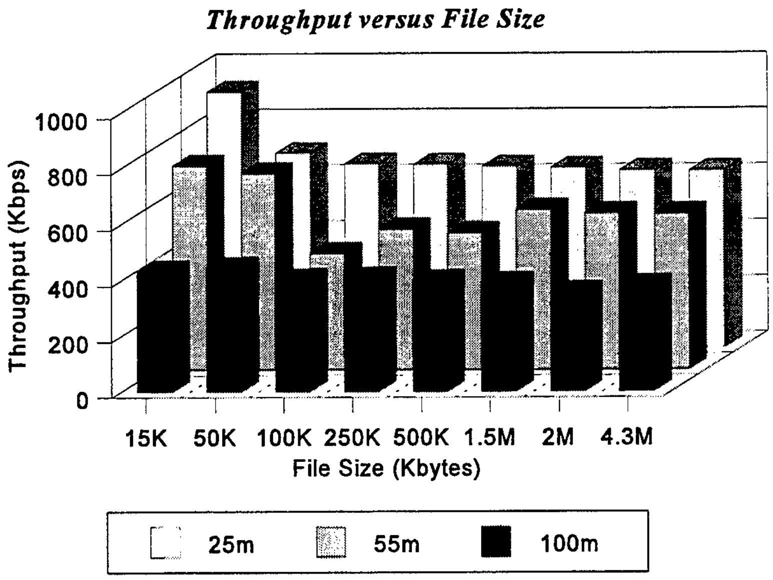


Figure 16 : Indoors throughput versus file size - II

Theoretically, the AirEzy wireless nodes can achieve a data rate of 1 Mbps, if the channel is fully utilized [Ref. 15]. Practically, the measurements are showing a 50% utilization for distances between 25 and 100 m. For distances above 100 m the effective data rate dropped under 400 Kbps, performing indoors. Generally, because of the increased overhead appended by the TCP/IP protocol suite, larger files are transferred slower. However, beyond performance degradation, due to increased distance between the two nodes, file size doesn't seem to have a significant effect on the network's throughput.

Indoors wireless transmissions usually suffer from microwave bouncing off metallic obstacles and heavy concrete walls. In this experiment these performance degrading factors were reduced to the minimum. The same experiments were held in an outdoors environment to indicate how these wireless nodes would work in their actual operating field. Usually, wireless devices achieve higher effective data rates and perform better in open space environments. Performance degradation factors here are : the natural vegetation (trees, buses etc.), the weather and the atmospheric conditions. The outdoors experiments were held inside the NPS campus in different day-time periods. A total of four experiments, two during early afternoon hours and two during late afternoon hours, were held. Generally, during the late afternoon measurements temperatures were lower and atmospheric humidity was higher (as expected). Tables 3 and 4 are showing the mean throughput values obtained in these measurements.

TABLE 3 :Throughput versus distance for different sized files - early afternnon

Day-time : 13:40 -------- Temperature : 62.9 ^{0}F								
Dist./File size	**100 K**	**200 K**	**400 K**	**600 K**	**800 K**	**1.2 M**	**2.5 M**	**Mean**
25 m	925	900	707	710	700	638	610	**741**
55 m	600	545	543	606	570	459	508	**547**
120 m	390	428	400	410	394	398	398	**403**
250 m	265	360	245	352	311	320	303	**308**
Mean Throughput in Kbps								**500**

TABLE 4 :Throughput versus distance for different sized files - late afternoon

Day-time : 20:00 -------- Temperature : 48.2 ^{0}F								
Dist./File size	**100 K**	**200 K**	**400 K**	**600 K**	**800 K**	**1.2 M**	**2.5 M**	**Mean**
25 m	904	828	654	690	705	608	610	**714**
55 m	400	423	388	406	370	359	348	**385**
120 m	205	328	364	308	364	349	332	**321**
250 m	265	158	300	289	270	270	203	**251**
Mean Throughput in Kbps								**418**

Figures 17 to 20 represent graphically the throughput measurement tabulated results, operating the AiEzy wireless nodes outdoors. In Figures 16 and 17 the relationship between throughput in Kbps and the distance between the wireless nodes, with the file size in bytes as a parameter, is shown.

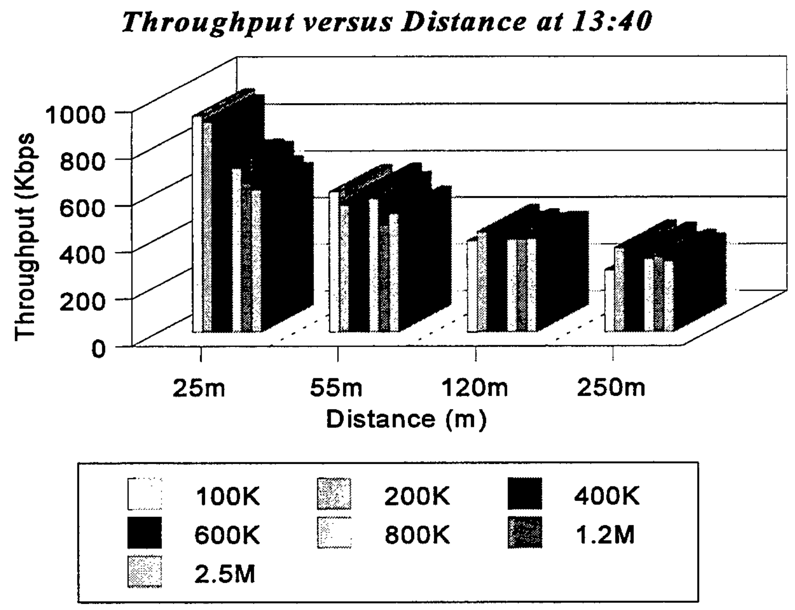


Figure 17 : Outdoors throughput versus distance at 13:40

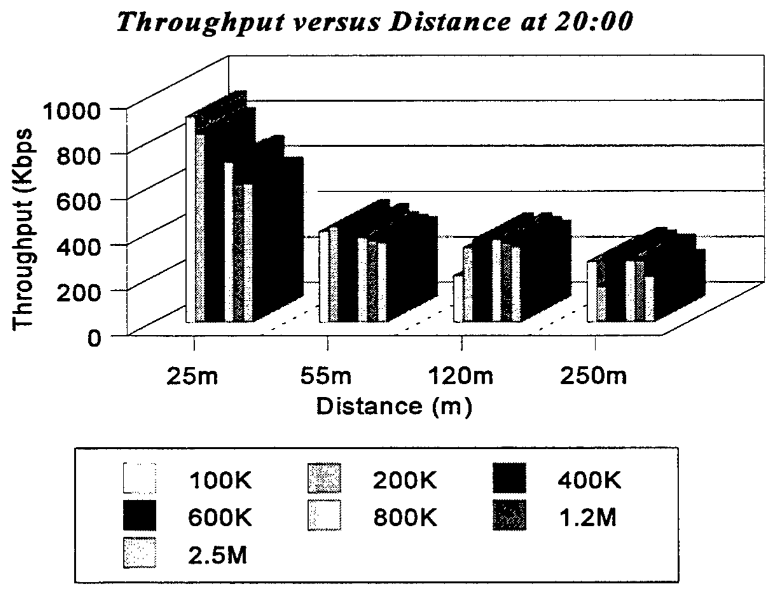


Figure 18 : Outdoors throughput versus distance at 20:00

In Figures 19 and 20 the relationship between throughput in Kbps and the file size in bytes, with the distance between the nodes as a parameter, is shown.

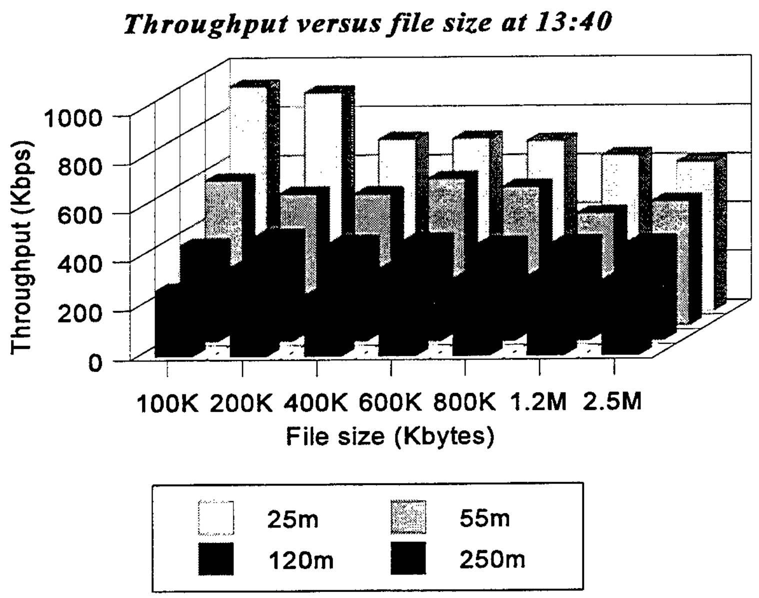


Figure 19 : Outdoors throughput versus file size at 13:40

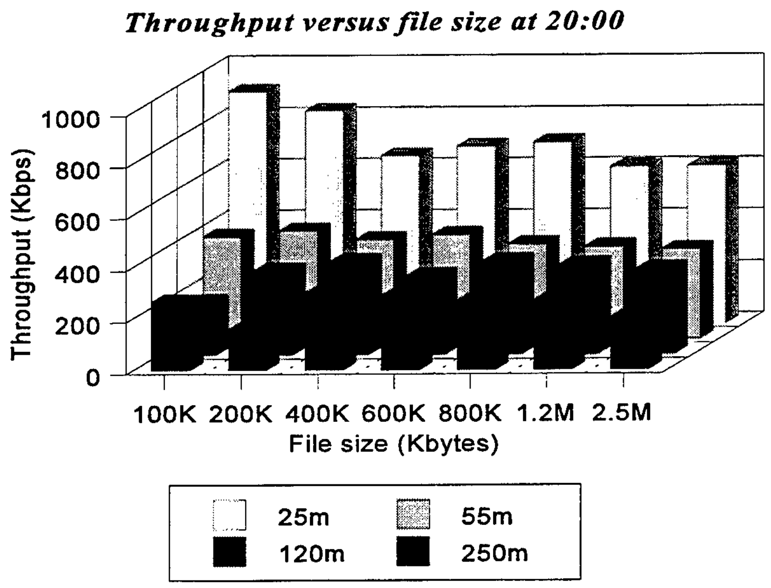


Figure 20 : Outdoors throughput versus file size at 20:00

Observation of Tables 3 and 4 and Figures 17 through 20 shows a throughput degradation of 100 Kbps during the late afternoon hours. The same measurements performed during a rainy day, with temperatures between 46.4 ^{0}F and 51.8^0 F, resulted in a mean throughput for all ranges (up to 250 m) of 300 Kbps. These results indicate the significance of the wireless transmission properties, described in Chapter II, for this particular band of frequencies (902-928 MHZ). This band was the dominant wireless band a few years ago. Many wireless vendors still produce devices utilizing the same frequency band. The low transmission power, regulated by the FCC, combined with the properties of microwave transmissions results in an average of 400 Kbps and a maximum of almost 1Mbps for most wireless devices utilizing this band. Nowadays, most vendors in the wireless market use even higher frequencies in the GHz range (2.4 or 5.7 GHz). Higher frequencies can carry higher bandwidth, resulting in increased channel throughput, but this advantage is counterbalanced by the great performance degradation effect when distance increases (Chapter II). At distances over 200 m throughput decreases rapidly. The only solution, when distance is of great importance, is the use of directional antennas (dishes) combined with higher transmission power.

The file size also does not seem to have a great effect in these throughput performance measurements either. However, reliability was highly dependent on the size of the transferred file. File sizes up to 600 K were transferred with constant data rate. For larger files data rate sometimes dropped up to 150 Kbps during the transfer session, and most of the times never got back to the initial transfer rates. Moreover, in some cases the file transfer stopped completely. Large files could not be transmitted/received continuously without any intermediate time interval between the FTP operations for distances beyond 120 m. The phenomenon occurred experimenting consecutive 1.2 Mbytes and 2.5 Mbytes FTP operations in the 120-250 m distance range. In some cases there was no communication between the two nodes, although the devices continuously transmitted, trying to reestablish communication and continue the file transfer protocol. Besides the fact that FTP stopped operating, reliability problem occurrence was easy to detect and confirmed because the "transmitting" indicator LEDs of the two wireless nodes were blinking (variable red flashes), without any indication of signal reception (same LEDs constant green flash). The MTBF (Mean Time Before Failure Factor) rated for 1.2 Mbyte and 2.5 Mbyte files was approximately 5 minutes after continuous operation, and had an occurrence rate of 20 % (one out of every five FTP experimental measurements).

There are two possible factors contributing to the reliability problems occurrence. Firstly, the disturbance of the direct LOS radio path between the transmitter's and the receiver's antennas (like obstacle or human presence). In this category even a slight change of the two antennas orientation, during an FTP measurement (it happened very often), should be included. Secondly, and more important, the way the protocol stack, and TCP particularly, handles communication errors. In wireless links, as the distance between transmitter and receiver increases, and because of the properties of wireless transmissions, error occurrence in the data packets is a very common phenomenon. If the errors occur only in a small portion of the packet (a few bits in error), recovery mechanisms implemented in software, mainly in the data link and transport layers, allow error detection or even correction. However, completely damaged packets occur very often in wireless links. The only way to handle this packets is retransmission. The transport entity (TCP) is responsible for this decision. Theoretically, transport protocols should be independent of the technology of the underlying network layer. The TCP should not care whether IP runs over fiber or over wireless medium. Practically, it does matter because TCP and most implementations based on this transport protocol have been carefully optimized, several years ago (TCP invented in 1977), for wired networks. In particular, if a TCP entity is waiting for a packet which doesn't arrive during a predefined period, TCP assumes this was due to network congestion. The transport protocol (TCP) is then notified by a **timeout** triggered by the so called "congestion control algorithm" (implemented in the same protocol). For wired networks this assumption holds, and time out occurrence notifies the sending TCP entity to slow down[4] and send packets less vigorously [Ref. 18]. Nowadays, in this type of networks packet loss is a very rare phenomenon (probability of bit error is less than 10^{-12}). But, for wireless networks packet

[4] Jacobson's slow start algorithm.

loss is the main reason triggering time outs, not network congestion. The proper approach to dealing with lost packets is to send them again, and as quickly as possible. The TCP protocol is doing exactly the opposite. It slows down (assuming congestion occurrence), making matters even worse. During the AirEzy performance measurements, when throughput degradation occurred (probably due to packet losses), instead of recovering and increasing the data rate, most of the times TCP dropped throughput leading to network reliability problems. A solution to this problem is being dealt with in Chapter IV.

Finally, availability has not been tested in these experiments. The AirEzy wireless nodes and the laptop PC took their power from AC outlets. Availability testing will be performed in later stages of the UXO detection project, with the mobile robot *shepherd* providing DC power, for the wireless communications nodes and the laptop PC, from his built in batteries.

C. REGULATION REQUIREMENTS

The usage of radio transmissions is regulated by the FCC (Federal Communications Commission) and by the upcoming IEEE 802.11 standard for wireless LANs. The UXO wireless network should be consistent with the provisions of both regulatory committees. These committees regulate the usage of frequency bands worldwide. Since the UXO detection project will not operate in a limited geographic spot, but will rather have a world wide application spectrum, adaption of the FCC and IEEE regulations and provisions is needed.

The lack of standards and regulations was the main reason for the limited widespread use of wireless LAN products up to last decade. In 1985, the FCC made the commercial development of radio-based LAN components possible by authorizing the public use of the Industrial, Scientific, and Medical (ISM) bands. This band of frequencies resides between 902 MHZ and 5.85 GHz, just above the cellular phone operating frequencies. The ISM band is very attractive to wireless network vendors because it provides a part of the spectrum upon which they can base their products, allowing the users to operate these products without obtaining an FCC license. Moreover, the deregulation of the frequency spectrum eliminates the need to perform costly and time-consuming frequency planning to coordinate wireless installations that will avoid interference with existing radio systems. This appears even more advantageous for installations requiring frequent movement of the communications equipment, like the UXO detection project, because paperwork involving relicensing of the equipment at a new location is also avoided. The allocation of the ISM band has had a dramatic effect on the wireless industry, prompting the development of wireless LAN components. Unfortunately, the ISM band frequencies are not available in all parts of the world, limiting the capability to operate wireless products (like the OTC's AirEzy wireless links) sold in the United States. The ISM frequency bands appear in Figure 21. The same Figure identifies which of the ISM bands are available for unlicensed usage around the world. The S band (2.4 GHz) is the only unlicensed band available worldwide. This band was approved in North and South America in the mid-1980s and was accepted in Europe and Asia in 1995 [Ref. 6]. Companies first began developing products in the I band because manufacturing cost in this band was cheaper. However, the lack of availability of this band in some countries and the need for greater bandwidth drove most companies to migrate their products to the S band. Nowadays, many vendors striving for higher data rates produce

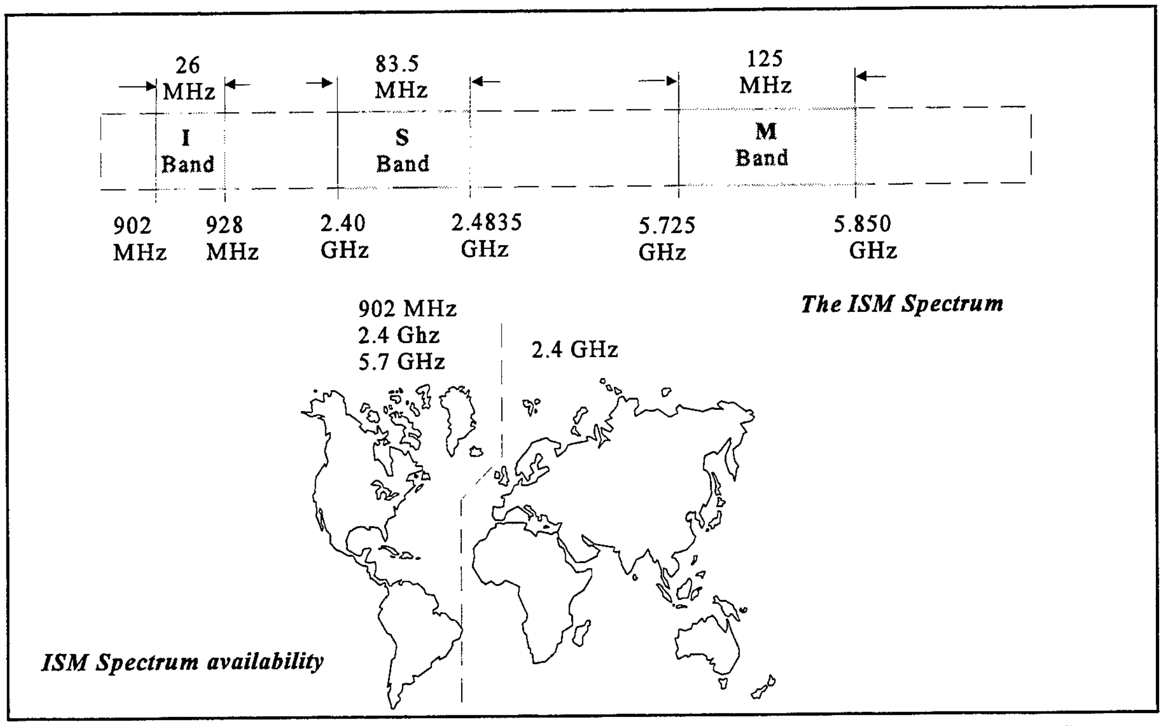


Figure 21 : The Industrial, Scientific, and Medical (ISM) frequency bands

wireless products in the M band. The only drawback for the M band is that many medical electronic equipment found in hospitals and clinics operate in this band (cause of interference). To some extend, the same applies for the I band because some industrial components utilize the same radio frequencies as wireless LANs, which could cause interference. That's the reason that made FCC regulate these frequency bands. Products that operate according to Part 15.247 of the FCC Rules and Regulations (wireless LANs) must utilize spread spectrum modulation to avoid interference.

With consideration for the wireless LANs and the radio spectrum usage, the IEEE 802 group responsible to harmonize regulations and standards throughout the world, has drafted the 802.11 standard for wireless LANs. The IEEE 802.11 standard regulates the technology (spread spectrum and infrared) used in wireless network implementations, and specifically develops a Medium Access Control (MAC) and Physical Layer (PHY) specification for wireless connectivity for fixed, portable and moving stations within a local area. The countries that can be accommodated, so far, by this standard and the frequency bands identified by the IEEE 802.11 group for world wide coverage appear in Table 5 [Ref. 7].

TABLE 5 : IEEE frequency bands for worldwide coverage

COUNTRY	***Frequency Band***
USA, Canada and most European countries	2.4 - 2.4835 GHz
Japan	2.471 - 2.497 GHz
France	2.446 - 2.4835 GHz
Spain	2.445 - 2.475 GHz

The FCC and IEEE 802.11 committees, have also made rules for the efficient spread spectrum modulation usage of wireless products. The FCC dictates that transmitters, utilizing frequency hopping spread spectrum, must not spend more than 0.4 seconds on any one channel every 20 seconds in the I band and every 30 seconds in the S band. Also, the transmitters must hop through at least 50 channels in the I band and 75 channels in the S band (a channel consists of a frequency width determined by the FCC). The IEEE 802.11 committee limits frequency hopping spread spectrum transmitters to the S (2.4 GHz) band (Table 5). For the direct sequence spread spectrum products the FCC dictates that ten or more chips per bit (spreading code) should be used. This rule limits the raw data throughput of direct sequence transmitters to 2 Mbps in the I band and 8 Mbps in the S band. Unfortunately, the number of chips is directly related to a signal's immunity to interference (Chapter II). The IEEE 802.11 standard dictates the use of eleven chips per bit for direct sequence products. Finally, the transmission output power for both technologies is limited by the FCC to under 1 watt. [Ref. 6]

The wireless products supporting the UXO detection project should meet all the regulation requirements mentioned.

D. SYSTEM INTERFERENCE REQUIREMENTS

Interference requirements are basically met by choosing the microwave spread spectrum technology. Spread spectrum systems experience very little interference, as described in Chapter II. When systems incorporating spread spectrum co-exist in the same area theoretically their is no interference problem. The FCC and IEEE 802.11 power management and frequency usage provisions set the basis for wireless systems cooperating under interference immune conditions. Generally, interference is uncommon with ISM band products because they operate on such little power. Testing two pairs of AirEzy wireless nodes operating in the same room did not show any indication of signal interference.

Narrowband interference from electronic devices that don't utilize spread spectrum is not expected to cause any significant problem to the UXO detection wireless network. The spread spectrum frequencies are far beyond the ones used from the remaining systems (mobile robot etc.) of the project. Even if a narrowband interference is present (frequency harmonics of lower basic frequencies) this type of interference only affects a small part of the information signal, resulting in few or no errors, since spread spectrum type products cover a wide amount of the bandwidth. Testing the AirEzy wireless devices on board of YAMABICO robot's platform did not show any interference problems. Narrowband interference with signal-to-interference ratios less than 10 dB does not usually affect spread spectrum transmissions.

Wideband interference, however, can have damaging effects on any type of radio transmission. Some typical sources of wideband interference are : domestic microwave ovens, elevator motors, duplicating machines, cordless phones, theft protection equipment etc. the primary source is microwave ovens operating in the 2.4 GHz band. Typical microwave ovens operate at 50 pulses per second and sweep through frequencies between 2.4 and 2.45 GHz, corrupting the wireless data signal if within 50 feet of the interfering source. The only way to handle wideband interference is to avoid it.

E. MOBILITY REQUIREMENTS

The maximum operating distance between the cooperating units of the UXO detection project has not been specified yet. However, mobility requirements have to be met for a flexible wireless network implementation. Most vendors producing spread spectrum communication systems ensure mobility capabilities. The maximum distance covered is product and technology dependent. Usually, wireless spread spectrum products can cover distances up to 800 or 1000 ft outdoors, operating at a maximum data rate of 1-2 Mbps.

Taking in to account all the requirements of the UXO detection project, Table 6 lists the wireless products that currently fit the UXO project needs in the wireless network market. This market is currently growing very fast, as wireless LANs becomes a communication necessity of our times. This Table may become obsolete after a few months, but the requirements for the wireless network that serves the UXO detection project do not change. Table 6 lists products covering distances more than 200 m, having a claimed data rate (the effective will be much lower) of at least 1 Mbps.

TABLE 6 : Wireless products that can accommodate the UXO project needs

Product Name	*Company*	*Interface Protocol*	*Wireless Technology*	*Data Rate*	*Open space Range*	*Comments*
AIRLAN	Soletek	ISA, PCMCIA II or parallel	DS-Spread spectrum (I band)	2 Mbps	≥330 m	Requires clear radio LOS. Power: 4W ERP
AiEZY 900	OTC Telecom	standalone Ethernet:BNC or Rj-45	DS-Spread spectrum I band	1 Mbps	240 m	Plug and Play Power: 100 mW
AirPort II	Windata	standalone (Ethernet)	DS-Spread spectrum S,M bands	5.7 Mbps	2.9 Km	Power: 1W ERP
ARLAN 630-631 APs	Aironet	ISA, PCMCIA II	DS- Spread spectrum I band or → S band →	0.4 Mbps 2 Mbps	up to 600m 300 m	Power: 1W ERP
BreezeNET	BreezeCOM	ISA, PCMCIA II	FH-Spread spectrum S band	1-3 Mbps	500 m	Provides multiple cell configuration. Power: 100mW
CruiseLAN	Zenith Data Systems	ISA, PCMCIA II	FH-Spread spectrum S band	1.6 Mbps	300 m	Features software encryption. Provides multiple cell configuration
CreditCard	Netwave	PCMCIA II	FH-Spread spectrum S band	1 Mbps	200 m	Power : 25 mW

Product Name	***Company***	***Interface Protocol***	***Wireless Technology***	***Data Rate***	***Open space Range***	***Comments***
FreePort	Windata,Inc.	Standalone (Ethernet)	DS-Spread spectrum S,M bands	5.7 Mbps	240 m	Uses a centralized wireless hub. Power : 1W
GoPrint	AeroComm	Parallel port	FH-Spread spectrum	1 Mbps	240 m	
Netwave	Xircom	PCMCIA II	FH-Spread spectrum S band	1 Mbps	220 m	Plug-and Play
PortLAN	RDC	PCMCIA II	FH-Spread spectrum S band	1 Mbps	830 m	Plug and Play Power: 100m
RangeLAN	Proxim	ISA, PCMCIA II	FH-Spread spectrum S band	1.6 Mbps	300 m	Plug-and Play Provides multiple cell configuration (via Ethernet)
Roamabout 2400	DEC	ISA, PCMCIA II	FH-Spread spectrum S band	1.6 Mbps	300 m	Plug-and Play

Product Name	***Company***	***Interface Protocol***	***Wireless Technology***	***Data Rate***	***Open space Range***	***Comments***
WaveLAN	Lucent (AT&T) and C-SPEC	ISA, PCMCIA II	DS-Spread spectrum I,S bands	2 Mbps	240 m	Plug-and Play Provides multiple cell configuration (via Ethernet) Power: 88mW
WaveLAN PC wireless adapter	KarlNet	ISA, PCMCIA II	DS-Spread spectrum S band	2 Mbps	240 m	Plug-and Play Power: 88mW
Wireless LAN	IBM	ISA, Micro Channel card, PCMCIA II	FH-Spread spectrum	0.5-1.2 Mbps	250 m	Plug-and Play Provides multiple cell configuration (via Ethernet and token ring). It provides continuous data encryption

IV. PROTOCOLS FOR THE UXO DETECTION WIRELESS NETWORK

A. NETWORK SYSTEM PROTOCOLS

Like wired LAN implementations, wireless networks also follow the layered protocol model. The same protocol hierarchy used for wired LANs, is also applicable for wireless LAN implementations. However, wireless networks have fundamental characteristics which make them significantly different from traditional wired LANs. The differences, accounting for the different medium (wireless medium (WM) versus cable), impact the wireless LAN design. These differences are found in the following layers of the protocol stack :

1. The physical layer (PHY) and,
2. The Medium Access sub-layer (MAC) of the Data Link layer.

One critical difference, addressed by the 802.11 standard, is that destination addresses do not equal destination locations for wireless LANs. In wired LANs an address (like an Ethernet address) is equivalent to a physical location. This is implicitly assumed in the design of wired LANs. In 802.11 standard, the addressable unit is a wireless station. This station is a message destination, but not (in general) a fixed physical location. The wireless PHY and MAC protocols have to take this into account. Generally, the IEEE 802.11 standard defines the major fundamental characteristics that wireless LAN implementors must take into account in their design. These characteristics, indicating special PHY and MAC protocol design, are described in the standard as follows [Ref. 6] :

1. Wireless LANs use a medium that has neither absolute nor readily observable boundaries outside of which stations with comfortant PHY transceivers are known to be unable to receive network frames.
2. They are unprotected from outside signals.
3. They communicate over a significantly less reliable media than wired LANs.
4. They have dynamic topologies. For wireless PHYs, well defined coverage areas simply do not exist. Propagation characteristics (as described in Chapter II) are dynamic and unpredictable. Small changes in position and direction (measurements

of Chapter II and III) may result in drastic differences in signal strength. Similar effects occur whether a station is stationary or mobile.

5. They lack full connectivity and therefore the assumption normally made that every station can hear every other station is invalid (the "hidden" and "exposed" station problems).

Based on the above characteristics, the IEEE 802.11 standard defines several physical layer signaling techniques and interface functions that shall be controlled by the 802.11 MAC sub-layer. The physical layer is the actual interface with the real network, and is implemented by the Network Interface Card (NIC), the wireless network access point (AP) and the transmitting antenna (instead of cable) of the wireless AP. Directional (yagi type) or omni-directional antennas can be used depending on the particular implementation.

The MAC sub-layer in wireless LANs is not one of the standard multiple access protocols like CSMA/CD, token bus or token ring, that IEEE 802 produced for wired LANs. Wireless LAN implementations use the Medium Access Control with Collision Avoidance (MACA) protocol or its improved successor MACAW. Currently most wireless network vendors (Table 6) implement the MACA protocol in their devices. The MACA protocol was used as the basis for the IEEE 802.11 wireless LAN standard. The protocol was proposed by P. Karn in 1990. The basic advantage of this protocol over the standard multiple access protocols, like CSMA/CD, is that it solves the so called "hidden and exposed stations" problems. These problems occur in wireless networks based only on carrier sensing to resolve multiple access problems. The nature of these problems is shown in Figure 22.

As shown in Figure 22, when A is transmitting to B (left hand side) it becomes a "hidden" station for station C, which is out of the radio range of A. Thus, C could sense an idle medium and falsely conclude that it can transmit to B, without any interference problem. This conclusion is obviously wrong. If C starts transmitting it will interfere with station's A frame at B. The problem is that station C is not able to detect a potential competitor (station A) because is too far away from it. The "exposed" station problem appears on the right hand side of Figure 22. Here C senses the medium and detects an ongoing transmission, from

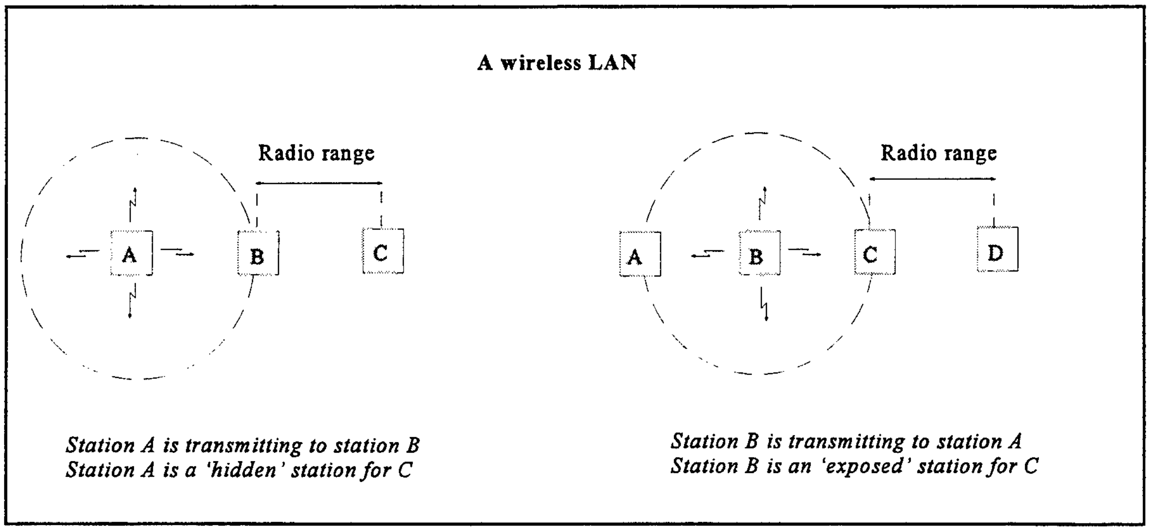

A wireless LAN
Radio range
A
B
C
Station A is transmitting to station B
Station A is a 'hidden' station for C
Radio range
A
B
C
D
Station B is transmitting to station A
Station B is an 'exposed' station for C

Figure 22 : The "hidden" and "exposed" station problems

station B to station A. Station C falsely concludes that it may not transmit to station D and backs off to avoid interference. In both situations ("hidden" or "exposed") the problem is the detection of radio signal presence at the receiver, not at the sender side. Carrier sensing, used in CSMA, does not provide the stations with the proper transmission status information for the area around the receiver. The MACA protocol solves this problem by giving the ability to each sender to stimulate the receiver before actual data frame transmission takes place. The sender sends a Request To Send (RTS) packet to the receiver indicating his intentions to transmit data frames. The RTS frame (30 bytes long) contains the length of the upcoming data frame. A cooperating receiver replies with a Clear To Send (CTS) packet containing the copied frame length from the RTS packet. After these transmissions the stations in the radio coverage area of the sender heard the RTS packet, those closer to the receiver the CTS packet, and the remaining stations heard both transmissions or neither of them. Each station behaves according to his position relative to the receiving station. Clearly, a station that hears the CTS packet must defer from sending anything [Ref. 10]. Despite these precautions, collisions can still occur. Collisions are resolved by usage of the binary exponential back off algorithm [Ref. 18].

Simulation studies and network measurements showed that MACA could be improved to perform better [Ref. 1]. First of all the basic utility of the CSMA, carrier sensing, had to be added to avoid synchronous RTS transmissions. Another basic improvement was the addition of acknowledgments (ACK packets) for successful data frame transmissions. Finally, a congestion control mechanism and a more sophisticated back off (after collision) mechanism was added to the protocol, which was renamed to MACAW by its designers.

The MACA and MACAW protocols are implemented in software drivers, and directly communicate with the NIC in the top down layered view, and the protocol suite (TCP/IP) in the bottom up view.

The various wired and wireless standards differ at the physical and MAC sub-layer but are compatible in the data link layer. The Logical Link Control (LLC) sub-layer is also present in wireless LAN implementations. Moreover, the 802.11 standard (PHY layer, MAC sub- layer) is required to appear to the LLC sub-layer as a current style 802 LAN [Ref. 6]. Figure 23 shows the wireless LAN protocol hierarchy in comparison with IEEE 802 standard for wired LANs.

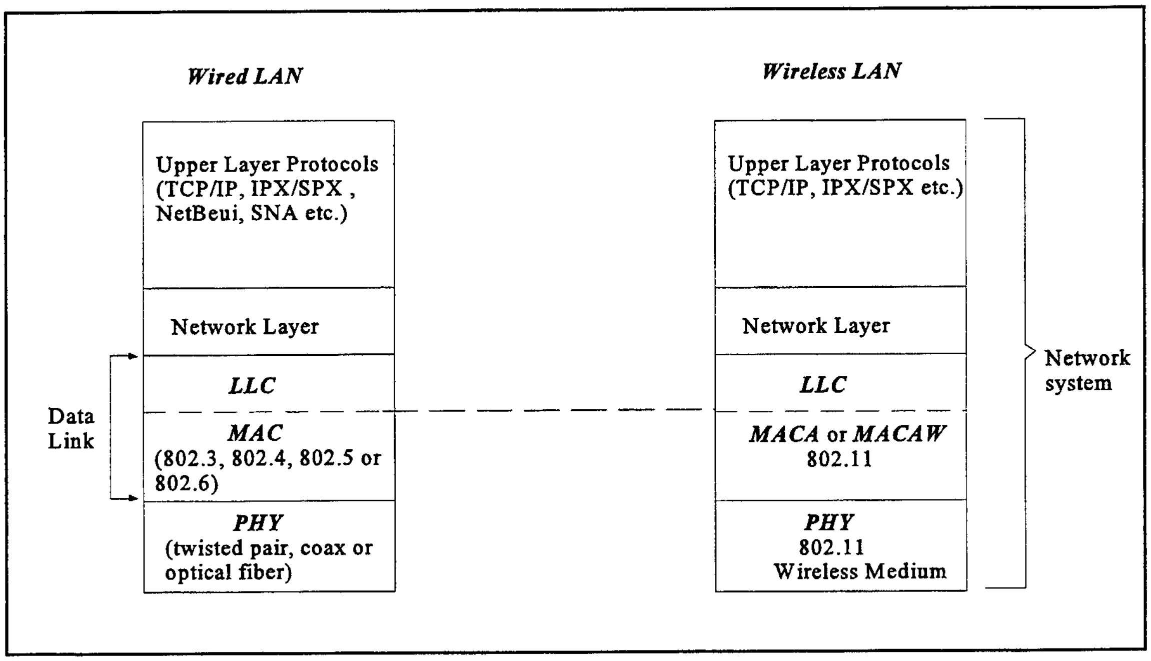

Wired LAN
Wireless LAN
Upper Layer Protocols (TCP/IP, IPX/SPX , NetBeui, SNA etc.)
Upper Layer Protocols (TCP/IP, IPX/SPX etc.)
Network Layer
Network Layer
LLC
LLC
Data Link
MAC (802.3, 802.4, 802.5 or 802.6)
MACA or MACAW 802.11
PHY (twisted pair, coax or optical fiber)
PHY 802.11 Wireless Medium
Network system

Figure 23 : Wireless LAN protocol hierarchy

B. APPLICATION PROTOCOL FOR THE UXO DETECTION PROJECT

1. Protocol Specification

The protocols shown in Figure 23 constitute the actual "network system", representing the first four layers of the Open Systems Interconnect (OSI) network model (physical, data link, network, transport). The network system modular architecture is generally provided, in hardware or software form, for any network implementation. Lower layers implemented in hardware and software (network interface and drivers), and the upper layers (TCP/IP protocol stack) constitute the NOS, implemented in software.

The UXO detection project needs a communication protocol to govern synchronized message exchange between the ground control station and the mobile robot. This protocol has to provide error free transparent communications between the two stations. The communications protocol that serves the UXO detection project belongs to the upper layer protocols of the OSI model. Specifically, this protocol implements the functionality of the session and application layers of the OSI model or just the application layer for other network models (like the DoD model). The design, specification and verification of this protocol follows the Communicating Finite State Machines (CFSM) model. The CFSM model is a Formal Description Technique (FDT) used to specify a procedure or protocol used for communication between two or more processes connected by a communication network [Ref. 13]. Most official network standards use FDT models as a descriptive tool of their protocols. Formal modeling of network protocols, with models like the CFSM, has two basic advantages :

1. Network protocol description is unambiguous. Protocol implementors and users can understand the exact protocols's operation.
2. It provides a formal framework for a rigorous analysis of the protocol.

The communication protocol for the UXO detection network is specified by a CFSM model using two communicating machines (application processes). One simulates the communication behavior of the ground control station and the other the mobile robot's vehicle's behavior. Transitions in the CFSM specification of the two machines (ground control station and mobile robot) characterize external message events like sending (+ sign) or receiving (- sign) or internal events (a timer goes off) happening locally in each machine (no sign). As in every CFSM specification, the states of the two machines are chosen to be those instants that each protocol machine is waiting for the next *event* (internal or external) to happen. The ground control station is defined as "machine A" and the mobile robot as "machine B" for protocol description simplification.

Figures 24 and 25 show the CFSM specification for these machines.

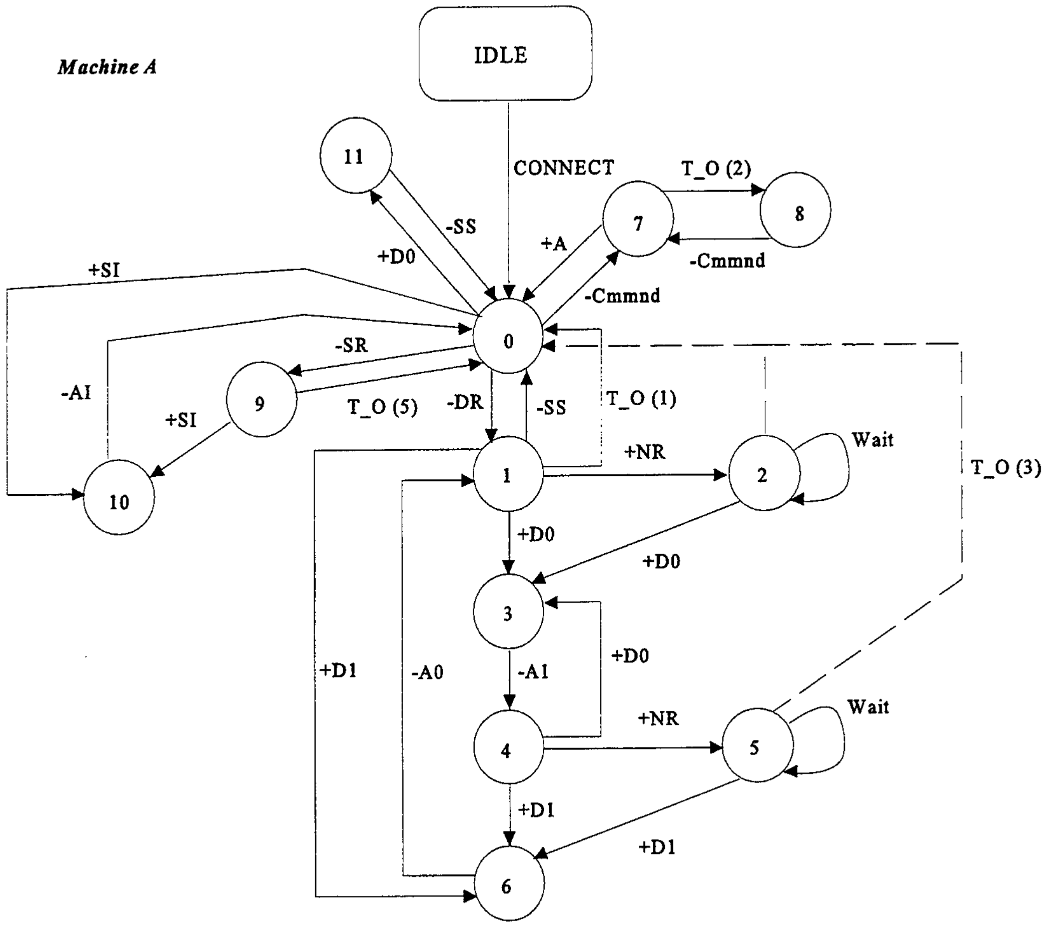


Figure 24 : CFSM specification for machine A (ground control station)

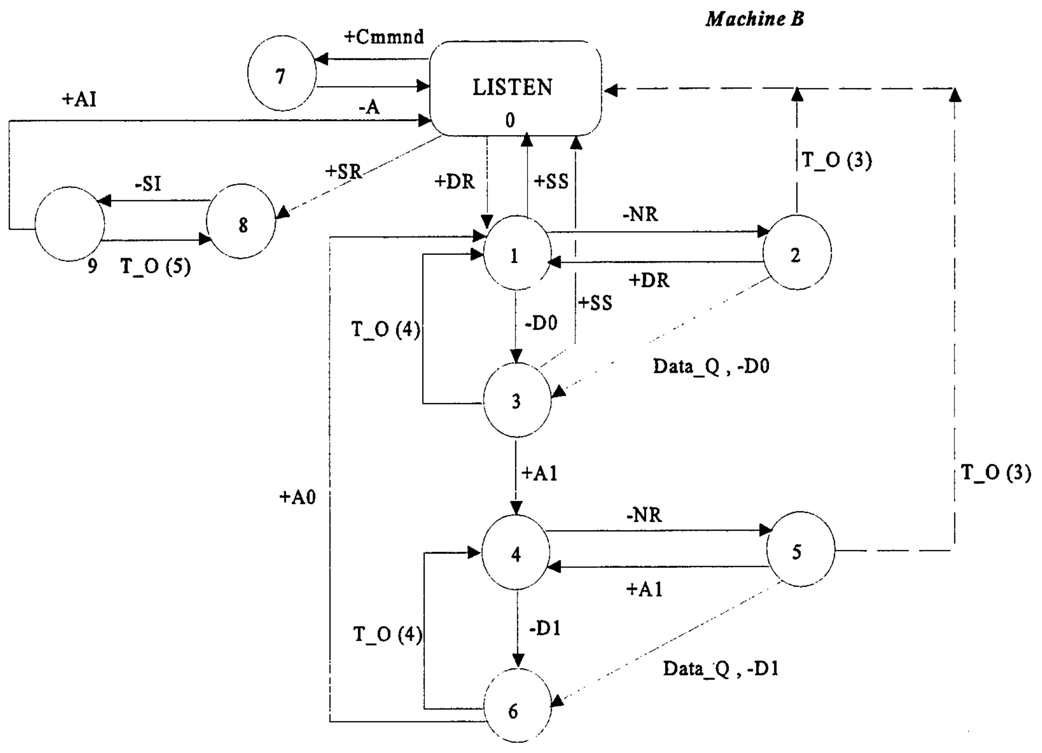


Figure 25 : CFSM specification for machine B (mobile robot)

Table 7 contains transition symbol explanation, for the CFSM specification of the two machines.

TABLE 7 : Transition symbol explanation

Transition Event	*Explanation*
DR, SR	Data Request, Status information Request (polling messages)
SS	Stop Sending command
NR	Not Ready to send data
D0, D1	Data block 0, Data block 1
A0, A1	Acknowledgments of D1 and D0 respectively
Cmmnd	Command from machine A to machine B (select message)
A, NAK	Positive (A) or negative (NAK) acknowledgment on A's Cmmnd

SI	Status Information message from machine B to machine A
AI	Acknowledgment of status Information message
T_O (I) **I=1,2,3,4,5**	Time out events. Timer expiration is an internal event triggered from the application software that implements each machine's specification
Data_Q	Data block Queued in machine's B (software) buffers. This is an internal event implemented in machine's B specification

2. Protocol Verification

The communication protocol specification for the two machines follows the **Poll/Select** discipline control scheme. Machine A can poll machine B for data delivery, or select machine B, to send a command.

Whenever machine A needs search results it polls machine B with a -DR message. This polling action initiates sequential data block transmissions by machine B to machine A. Data blocks are specially formatted data packets, stored in machine's A software buffers. These packets contain search results, obtained by machine's A searching actions, formatted in a predetermined *information vector* [1] fashion. The maximum size of each data block is 1500 bytes. To prevent synchronization problems, data block exchange implements the alternating bit (AB) protocol. Data blocks are numbered with sequential 0 and 1 values, and their subsequent acknowledgments with 1 and 0 values respectively. Data block 0 (D0) is acknowledged by machine A with an A1 and D1 with an A0, thus preventing duplicate or asynchronized packet and acknowledgment exchange. If an acknowledgment does not arrive within a specified amount of time a time-out is triggered causing a retransmission of the unacknowledged data block. The time out value (T_O (4)) is calculated based on the time needed for the acknowledgment of the corresponding data packet to arrive back to machine B. Data block processing time (for both machines) and acknowledgment transmission time,

[1] The vector's fields contain the search results obtained in a particular searching area.

are mostly hardware and network system dependent and can not yet been predicted accurately. However, for the small distances of operation (up to 800 ft) between the two machines (low propagation delay) these delay-time contributing factors seem to dominate the T-O (4) calculation over the propagation delay time. This time (T_{PROP}) is calculated based on the maximum open space distance of the average wireless device on table 7. A correction factor of 0.05 ms is added (by estimation) to account for the absence of data processing and acknowledgment transmission times.

$$T_{PROP} = \frac{Open\ space\ distance}{Speed\ of\ light} = \frac{243.84\ m}{3\ x\ 10^{8}\ m/s} = 8.128\ x\ 10^{-7}\ sec$$

$$\text{T\_O (4)} = T_{PROP} + 5 \text{ x } 10^{-5} = 5 \text{ x } 10^{-5}\ sec$$

The time-out T_O (1) is triggered to prevent a deadlock caused by a communications error or a complete loss of the -DR message. If machine B does not respond, initiating data blocks sequential transmission, within T_O (1) seconds, a time-out is triggered causing the -DR message retransmission. This time out is calculated based on the time needed for machine B to transmit a complete data block (1500 bytes long) and the propagation delay until this block reaches machine A. Again, a correction factor of 0.05 msec is added.

$$T_{Data\ Block} = \frac{Data\ Block\ size}{Average\ data\ rate} = \frac{1500\ x\ 8\ bits}{500\ Kbps} = 0.024\ sec$$

$$\text{T\_O (1)} = T_{Data\ Block} + T_{PROP} + 5 \text{ x } 10^{-5} = 0.02405 \approx 0.024\ sec$$

The protocol allows machine B responding with a NR (not ready) message, while in state 1 or state 4. This provides for situations where machine's B data buffers are empty. These situations can occur quite often in UXO searching operations. Some typical examples are : machine's B (robot vehicle) searching sensors delayed data delivery for some reason (instrument malfunction), or the application, processing the sensor's data, delayed data delivery to the communications application etc. In situations like that, both machines go to a waiting state (state 2 or state 5). In this state machine B waits for the internal event Data_Q to happen, for protocol continuation. When a data block is queued in machine's B data buffers (Data_Q internal event), this block is transmitted and both machines continue message exchange from where they had left, according to the AB protocol. To prevent a possible deadlock situation that might occur, if the internal event Data_Q never happens or delays for a significant time period, a T_O (3) time out is triggered. When this time-out is triggered both machines return to the initial state (state 0), by that way resolving any synchronization problems that might occur otherwise. From this state the protocol starts all over again.

The T_O (3) timer value is calculated based on the time needed for machine A to collect a complete data block (containing search data results) and transmit it to machine A. Since data collection time is not yet defined by the UXO detection project group, a large T_O (3) value will compensate for almost all problematic situations associated with robot's incapability to obtain and transmit searching data. It is estimated, that a T_O (3) = 5 sec value should give enough time for this operation to complete.

Another polling action, from machine A to machine B, is the status information request. This request is made by the SR message transmission to machine B. Status information returns within SI message from machine B. If status information does not arrive within a specified amount of time, a time-out event is triggered again. The time-out value for the T_O (5) timer is calculated based on the time needed for machine B to transmit a status information packet (SI) to machine A. Status information packets have the same size (1500 bytes) with the data block packets. Taking in to account the same propagation delay, as in the previous timer values calculation, this value is :

$$\text{T\_O (5)} = T_{status\ Info} + T_{PROP} = T_{data\ Block} + T_{PROP} = \text{T\_O (1)} = 0.024\ sec$$

Finally, machine A may select to send a command to machine B. Basically, this command satisfies the need of controlling the robot's motion, and changing the robot's searching pattern and operation mode. An emergency shut down or other urgent commands, like "stop searching immediately" (dangerous situations), can be implemented in the format of this message. Machine B (mobile robot) acknowledges the command and follows, if it is able, what is ordered. An -A message indicates that machine B has received the command and it is willing to follow it, and -NAK indicates that the received command cannot be followed. The T_O (2) timer prevents from deadlocks if the -Cmmnd message is lost. The timer has the same time-out value as the T_O (4) timer.

Figures 26, 27, 28 and 29 show the reachability analysis (verification) for the communication protocol between machines A and B. Figure 26 shows the reachability analysis for the DR message branch, Figure 27 for the Data Block exchange message branch, Figure 28 for the Status Information (SI) exchange and Figure 29 for the Cmmnd message branch.

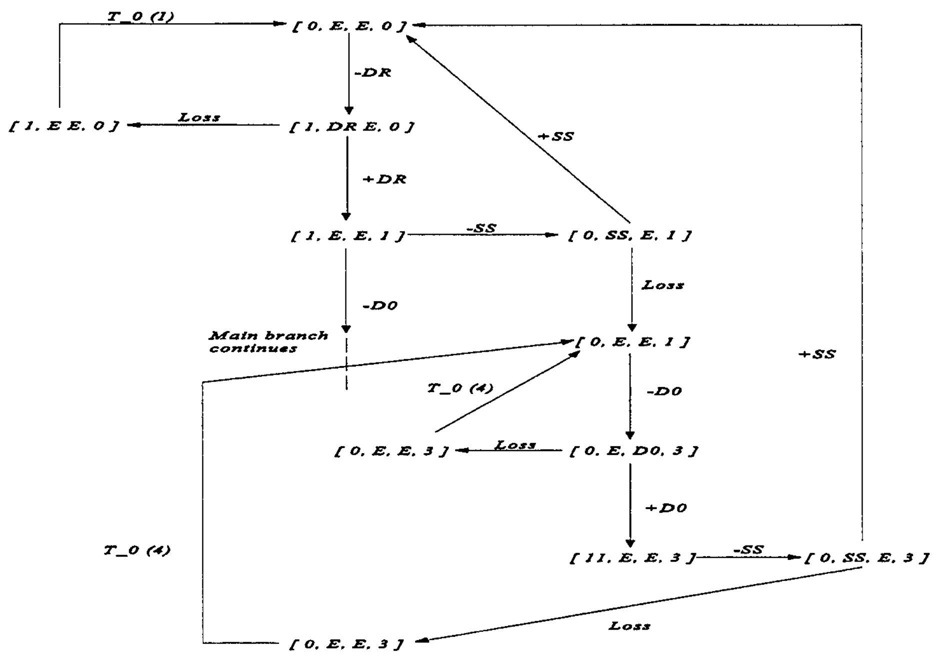


Figure 26 : Reachability analysis for the DR message branch

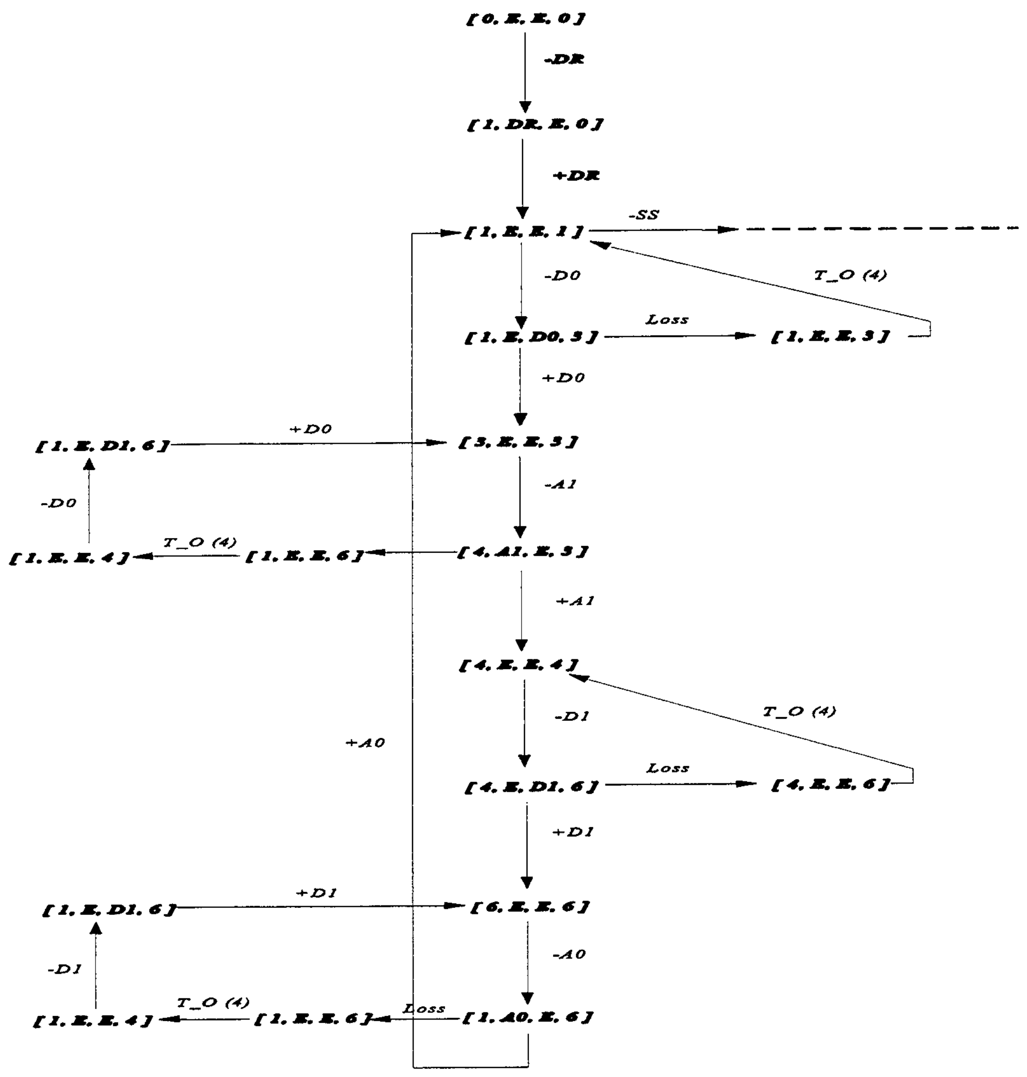


Figure 27 : Reachability analysis for the Data Block exchange branch

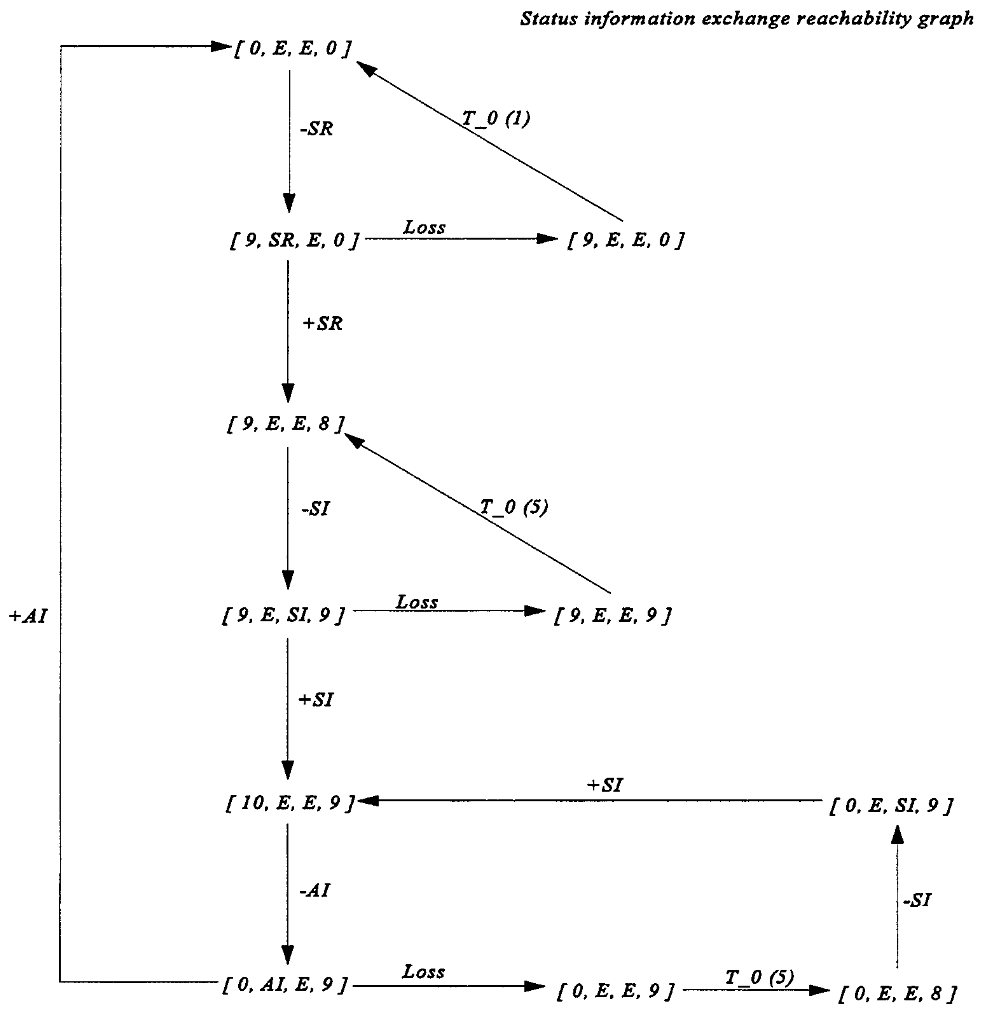


Figure 28 : Reachability analysis for the Status Information exchange branch

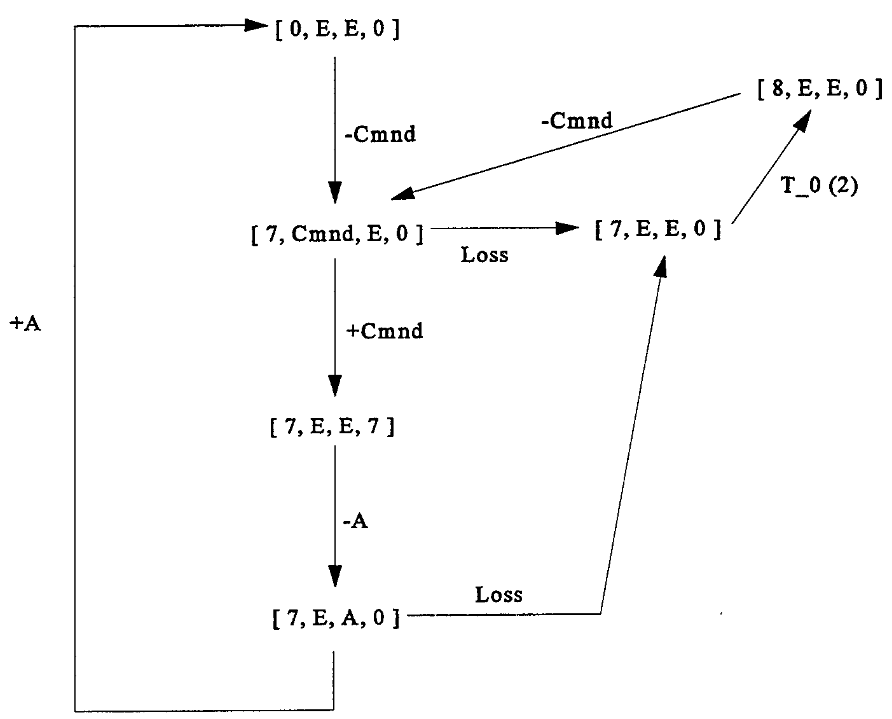


Figure 29 : Reachability analysis for the Cmmnd message branch

V. PROTOCOL IMPLEMENTATION WITH WINDOWS SOCKETS

A. WINSOCK API

Network protocols are usually implemented by a programming (software) interface tool that directly interacts, in the top-down view, with one (or more) of the underlying functional layers of the OSI network model. The usual approach, widely acknowledged as the standard programming interface with TCP/IP protocol suite, was Berkeley Sockets Application Programming Interface (API), as implemented in version 4.3 of Berkeley Software Distribution (BSD 4.3). The last five years another standard API, Windows Sockets (Winsock), is the tool of preference for many users that create network programs and especially network applications that run over the Internet.

Winsock Application Programming Interface (API) is an open interface for network programming under Microsoft Windows. The Winsock specification is "open" in the same sense as other open systems. It was created and continuously improved and tuned, in the spirit of cooperation. Different network vendors and network programmers participated and continue to participate in the development of this programming standard. The standard is freely available (the easiest access is over the Internet) allowing anyone to create, or modify already existing, Winsock applications. Winsock API (WSA) consists of a collection of function calls, data structures, and conventions. The basic header files that provide the Winsock API specification are: **winsock.h** (Appendix B) and **winsockx.h**.

Figure 30 shows the Winsock network model in comparison with the standard OSI hierarchical network structure. Winsock directly interacts with the Transport protocol of the OSI model, or the TCP/IP protocol suite of DoD network model. As mentioned in Chapter V, the network interface and the drivers constitute the Network system, which in turn interacts with the TCP/IP suite (a different proprietary API exists there) to provide reliable services to the Winsock applications.

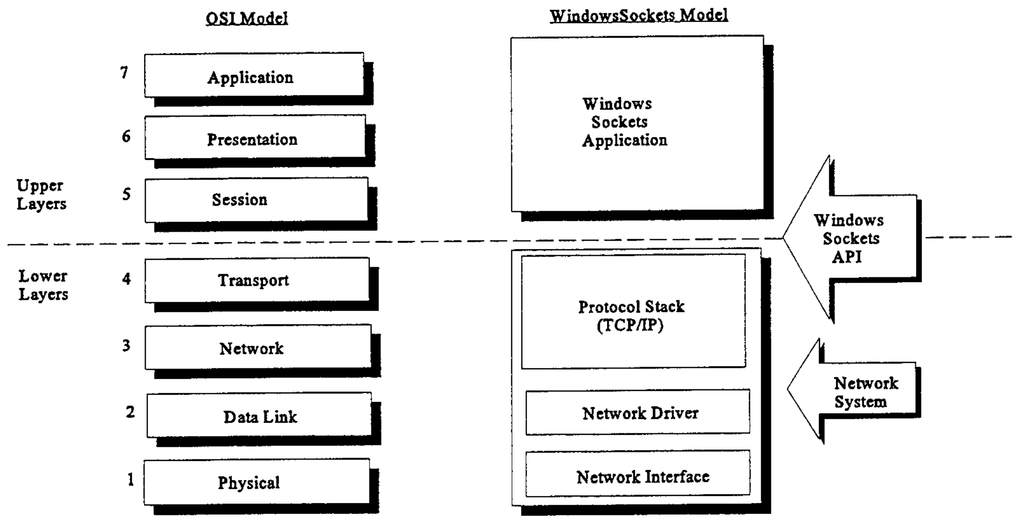
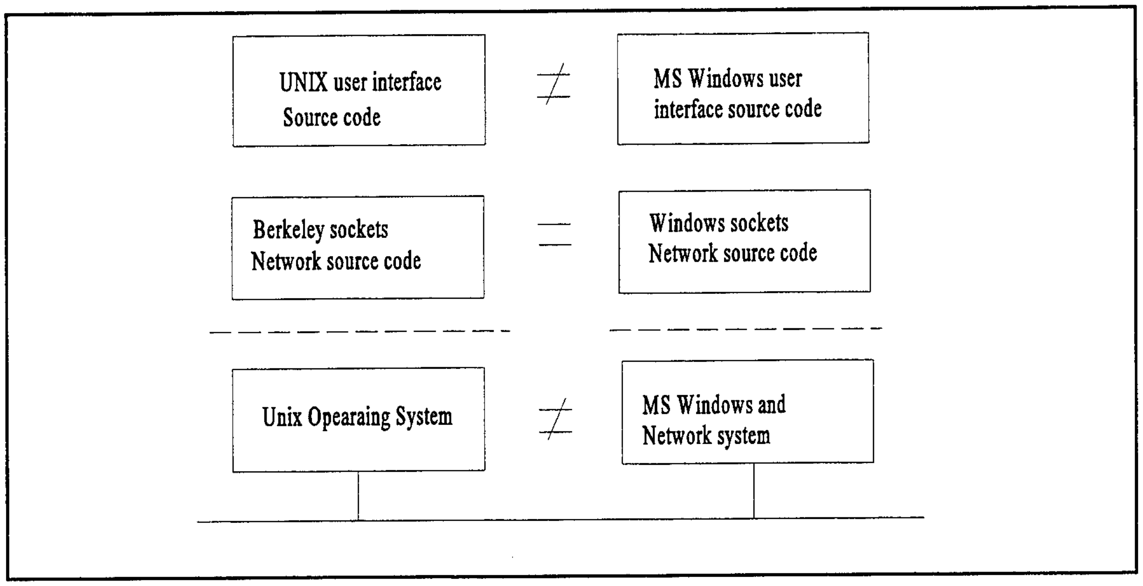


Figure 30 : The Winsock network model in comparison with the OSI model

The benefits that Winsock API (WSA) provides to network application implementors can be summarized as follows:

1. It provides source code portability with Berkeley sockets API. Almost all the functions and procedures used by the two standards are exactly the same (Winsock derives from BSD sockets). Figure 31 shows the source code portability aspect. The only differences account for the new mode of operation (asynchronous mode)



UNIX user interface Source code
≠
MS Windows user interface source code
Berkeley sockets Network source code
=
Windows sockets Network source code
Unix Opearaing System
≠
MS Windows and Network system

Figure 31 : Network source code portability from Berkeley sockets to Winsock

that Winsock defines. This mode was critically advantageous for the non-preemptive Operating Systems (OS) like DOS and Windows 3.1 (BSD sockets support only blocking and non-blocking modes of operation), but the new multitasking Windows 95 and Windows NT OS support network applications designed in the other modes very efficiently.

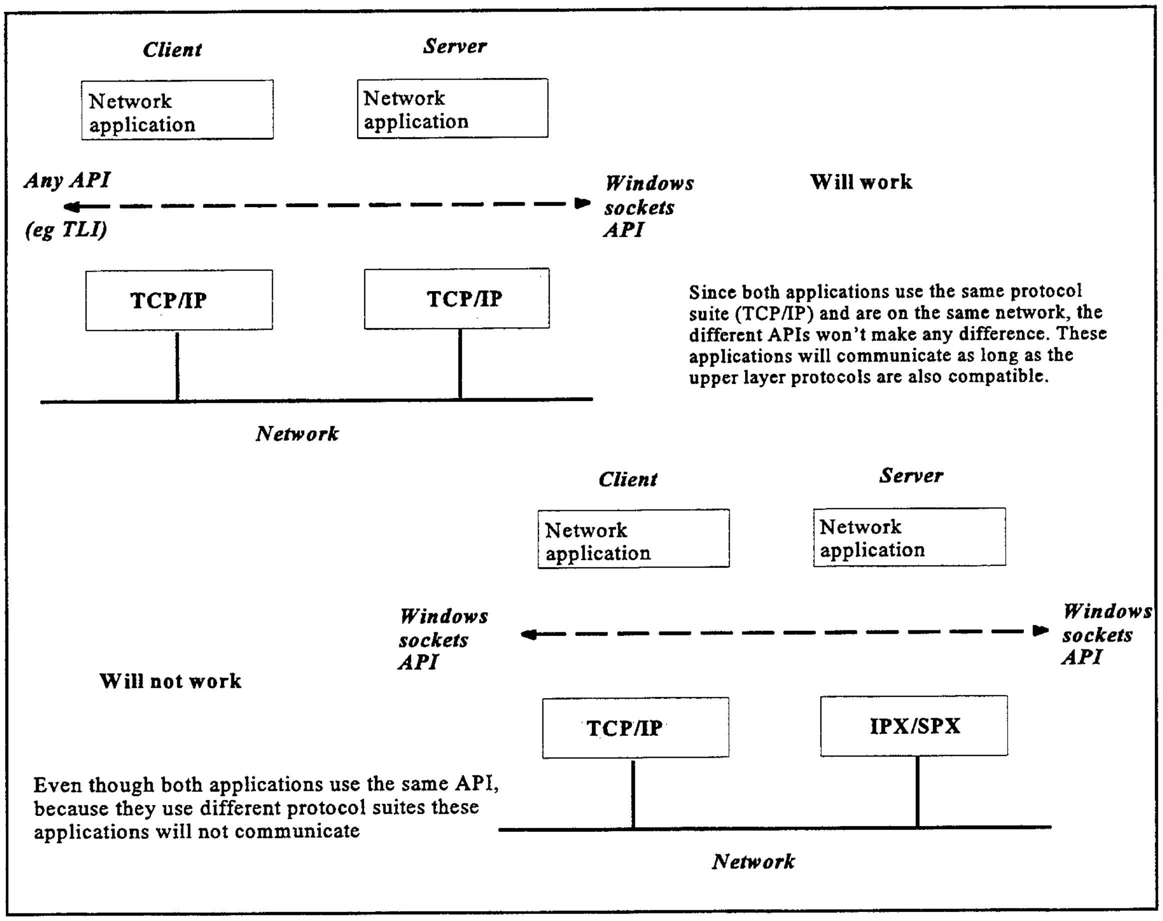
2. It is protocol independent and also network media independent. WSA can provide access to different protocol suites, like: DECNet, AppleTalk, SPX/IPX, OSI, SNA, TCP/IP and many others. WSA provides this independency by supporting dynamic linking. Dynamic link libraries (DLL) are a key feature of MS Windows. They are libraries of executable procedures, with well-defined interfaces. Applications link with them dynamically at run time (rather than statically at compile time). Multiple applications can use a DLL simultaneously (they share code), which means there is only one copy of the DLL code in memory. Another important aspect is that the DLL is separate from the application, so one can be changed with out affecting the other. However, the most important advantage is that DLLs that provide compatible APIs, also provide compatible application binary interface. This means that Winsock implementations have portable executable programs, not just source files. Once the Winsock source code has been compiled and linked, the executable program created will run over any vendor's Winsock-compliant interface.

   Winsock implementations run over any type of network medium : Ethernet, wireless, Token Ring, FDDI etc. Winsock only interacts with the DLLs, and does not need to know any other API mechanism. Proprietary hardware APIs provide interaction between the network interface card (Ethernet, 802.11 etc.) and the multiple protocol stack drivers (ODI, Packet Driver, NDIS etc.) which in turn communicate with the upper protocols (TCP/IP, DECNet, etc.) with standard driver APIs.

3. As mentioned earlier Winsock is an open standard. Open standards make technology accessible. They allow network users and programmers to mix and

match components from different vendors. The TCP/IP suite is the ideal example of an open standard. The TCP/IP protocol suite is responsible for the phenomenal growth of the Internet. Its success is due to its interoperability, which resulted from real-world testing and refinement by protocol stack and application developers involved in its development. Winsock as an open standard, provides a well defined interface, so that one vendor's product can interoperate with other's. The portability of the Winsock code between platforms is really essential.

Network code portability is sometimes misinterpreted. Network programmers, and Winsock application users must understand that Winsock applications (as well as other network API applications) running over different protocol suites will not communicate with each other (Figure 32). This is not a problem imposed by the API specification itself, but



Client
Server
Network application
Network application
Any API
(eg TLI)
Windows sockets API
Will work
TCP/IP
TCP/IP
Since both applications use the same protocol suite (TCP/IP) and are on the same network, the different APIs won't make any difference. These applications will communicate as long as the upper layer protocols are also compatible.
Network
Client
Server
Network application
Network application
Windows sockets API
Windows sockets API
Will not work
TCP/IP
IPX/SPX
Even though both applications use the same API, because they use different protocol suites these applications will not communicate
Network

Figure 32 : Protocols and APIs interoperation

rather the way computer networks and computers communicate with each other. Communication between two people speaking different languages will fail even though the interface (voice) is common. Figure 32 shows how protocols and APIs interoperate [Ref. 16].

B. WINSOCK PROGRAMMING MODEL

Winsock API is a kind of "programmatic plug" to any network. The socket concept is the basis of Winsock (and of Berkeley sockets) programming. A **socket** is an endpoint of communication, created in software, and equivalent to a computer's network (hardware) interface. It allows a network application to "plug into" the network (metaphorically). Typically, there is only one physical network interface on a computer, but the number of sockets can be far more. There is a one-for-many correspondence. Many sockets (software communication endpoints) can use a single network interface simultaneously.

There are two types of endpoints (sockets) : clients and servers. By definition, a client sends the first packet, and the server receives it. This assumption helps network code sketching and writing and does not represent the actual functionality of the client-server relationship. Winsock client and server applications are generally characterized by their role during initial communication phase. After initial contact, either the client or the server is capable of sending and receiving data (they are both piers). The services these applications provide can reverse this relationship any time after their first communication between each other. For the UXO detection communication protocol, specified in chapter IV, machine A represents the client (sends the first message always) and machine B (rotary vehicle) the server.

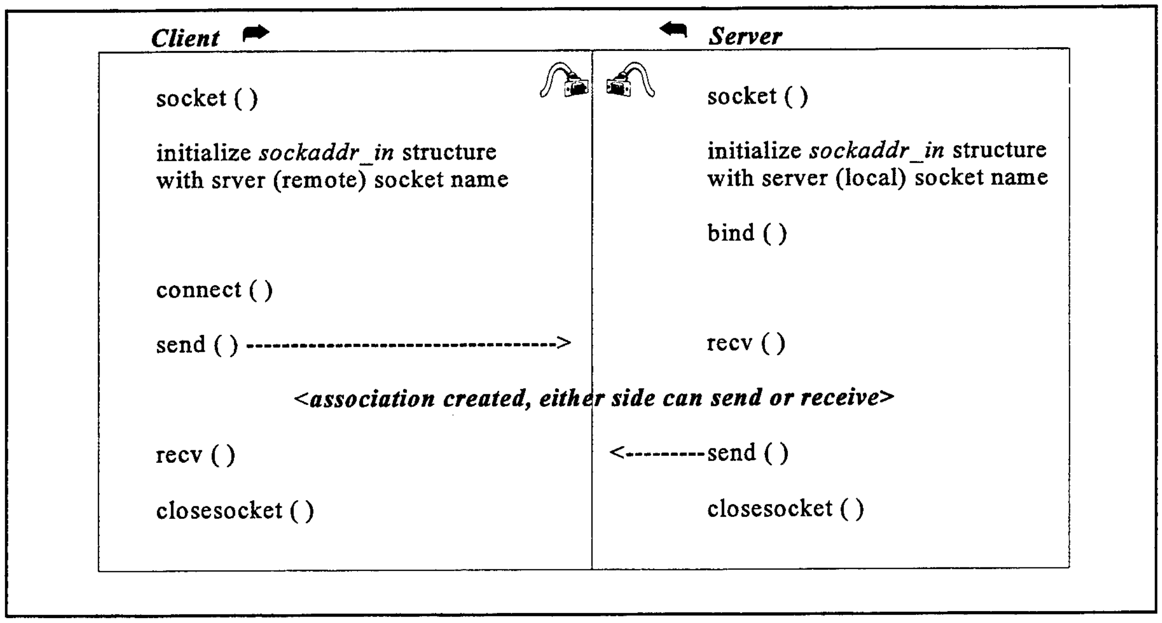
All network applications (clients and servers) usually follow five programming steps (in both Winsock and Berkeley sockets implementations):

1. Open a socket
2. Name the socket
3. Associate the socket with another socket (clients with servers)
4. Send and receive data between sockets, and
5. Close the socket

The communication protocol specified for the UXO detection project is implemented

(Winsock application) on top of the User Datagram Protocol (UDP). The choice has been made in the basis of simplicity and efficiency. The UDP transport has low overhead, so it provides efficiency that can result in performance benefits, and it is easy to use and implement. However, UDP connectionless transport, also called **datagram service**, is unreliable because it neither guarantees packet delivery nor preserves the packet sequence. Although datagram service is unreliable, datagram applications need not be unreliable. Datagram applications can implement the services that provide the missing reliability. This is exactly the service that the two FSM specifications, drawn in Figures 24 and 25, provide for the UXO detection project. They provide positive acknowledgment with retransmission (with time-outs) service, and data sequencing service, so that a receiver can resequence data when needed and detect and discard duplicate data packets. A generic programming model for UDP clients and servers, following the Winsock (or Berkeley sockets) convention appears in Figure 33. [Ref. 4, 16]

Figure 33 shows the generic model followed by the UDP implementation of the communication protocols drawn in Chapter IV. The ground control station is the client application, initializing the communication exchange by asking some data (search results by



Client
Server
socket ( )
initialize sockaddr_in structure with srver (remote) socket name
connect ( )
send ( ) ---------------------------------->
recv ( )
closesocket ( )
socket ( )
initialize sockaddr_in structure with server (local) socket name
bind ( )
recv ( )
<association created, either side can send or receive>
<---------send ( )
closesocket ( )

Figure 33 : Connectionless (UDP) network application sketch (set the remote socket name once)

the DR message or status information by the SR message), and the rotary vehicle becomes the server application that provides this information.

The main data exchange branch of this protocol appears in Appendix A in Winsock ANSI C code implementation. Appendix B presents the winsock.h header file, that includes most of the definitions, functions and procedures used in the code implementation. Some conventions and definitions come from the **windows.h** header file, which is a part of Windows OS itself. This header file is not presented in the thesis research, as it is of no particular interest for network programming.

VI. CONCLUSION

Many modern applications and projects cannot be served by traditional wired-based networking technologies. Wireless communications provide an effective solution for projects that require mobility and especially for those for which implementation spans multiple heterogeneous geographic locations. The mine/UXO detection and clearing project belongs to this category. In this thesis, the physical (hardware) and logical (software) architecture of a wireless LAN that will accommodate the mine/UXO detection project needs is analyzed.

A. CONTRIBUTIONS

Based on the characteristics and the topologies of traditional wireless LAN configurations, a stand alone wireless network is proposed. The major contributions of the thesis on the wireless network configuration are the following:

1. The thesis investigated the proper wireless modulation technique. The wireless LAN should utilize microwave Spread Spectrum modulation. This technology enhances mobility and assures immunity to interference for every wireless LAN implementation that adopts the regulations and provisions of the two regulatory agencies: the FCC and the IEEE (802.11 standard).
2. The thesis proposed a series of wireless devices that have the characteristics needed to accommodate the UXO project requirements. Counterbalancing between the limitations posted by the FCC and IEEE 802.11 regulations and the desired performance and communications distance coverage, the thesis proposed a series (Table 7) of wireless products, through survey in the current network market, that best accommodate the project needs.
3. The thesis specified the wireless protocols that implement the proposed technology (Spread Spectrum) providing multiple access capability to the wireless communication medium for a number of stations simultaneously (MACA and MACAW).
4. The communication protocol between the two main operating stations of the UXO project (Ground Control Station and rotary vehicle robot) was developed and

specified by a FSM model, analyzed for error free operation by the use of timers and positive acknowledgments with retransmissions (Alternating Bit protocol), and finally verified using reachability analysis diagrams.

5. The communication protocol was implemented (in ANSI C code) as an OSI layer application protocol, by the use of Windows sockets network API.

B. SUGGESTIONS FOR FUTURE WORK

In this thesis the communication protocol between the two main stations performing UXO detection and clearing was designed and implemented. The nature of the UXO project suggests the use of additional semi-autonomous vehicles (like the air vehicle) in the future. Additional communication protocols as well as broadcasting or multicasting capabilities, at least for the ground control station (communications coordinator), should be considered as a communications improvement. The use of the User Datagram Protocol (UDP) transport by this thesis, in the proposed communication protocol supports broadcasting and multicasting.

The code implementation of the proposed communication protocol (Winsock application) can be further improved by the addition of a rigorous network error analysis and handling. The current implementation does not handle all failures and errors it encounters gracefully. When errors occur notifying the user and aborting the connection is the common strategy. A rigorous error analysis is a potential field for a future research.

Finally, the new capabilities (ease of implementation, complete network packages, real time code testing etc.) and the flexibility that Java network programming language developed during the last years indicates a new tool for simple and elegant network application implementations. Many Internet industry watchers predict that the software of the future will use networks (specially wireless) and local resources in ways that may seem radical by today's standards (like the Winsock and Berkeley sockets network standards). The Java language is a modern application development language designed specifically for the distributed, network applications of the future.

APPENDIX A. MAIN DATA EXCHANGE CLIENT AND SERVER

/* The **Data_Block_Excng ( )** is the client application that implements the main data exchange branch (ground station's functionality) of the communication protocol showed as a FSM in Figure 24. The code follows the Winsock programming conventions. The client initiates association with the server by sending a DR message. This message is implemented as a two byte character string, stored in a control output buffer. The server can then initiate the data block exchange branch (Alternating Bit sequence) or respond with a NR message. The data blocks are organized in strings of characters 1500 bytes long. The first byte of each message (for data blocks or control messages) does not contain information, but is rather for message identification. Examination of the first byte of the incoming data (stored in the clients input buffers) determines protocol continuation. Acknowledgments of data blocks are implemented by 1-byte characters '0' and '1'. After the first iteration of the data block exchange loop, the STOP SENDING condition (internal event) is examined. This event is implemented by a function call that opens a file (the 'Cmmnd_file') and examines if the user or another application has issued a STOP SENDING command. Time-outs are implemented by the use of Winsock setsockopt ( ) function (nOptVal parameter). Because Winsock accepts for all applications time-out values larger than 500 ms, some hypothetical values (not the ones calculated in Chapter IV) relatively corresponding to the actual delay of each time-out are used in this implementation. The code follows the basic Winsock client/server conventions. */

```
/*----------------- Header files ----------------*/
#include <windows.h>
#include <winsock.h>
#include <winsockx.h>
#include <stdio.h>

#define BUFSIZE 1500 ;
#define PORT 600 ;        /* port number for the server's address
                                                   structure */


/*------------- Global variables ----------------*/
extern SOCKET s;
SOCKADDR_IN stRmtName ;
SOCKADDR_IN stRmtName ;
static char InBuffer[BUFSIZE] ;         /* input and */
static char CntrlBuf[ ] = "DR" ;        /* control buffers */
static char CntrlBuffer = 'S'
extern HFILE hFile                      /* file handle */
int nRet ;
int Buflength = 1500 ;            /* data buffer */
BOOL STOP_SENDING = FALSE ;

*/sequence number for data blocks and acknowledgments */
extern char frame_expected ='0' ;
```

```
char ackBuffer ;              /* acknowledgments buffer */

int timeOutOne = 500 ;        /* timeout values */
int timeOutThree = 5000 ;



/*------------- Function Prototypes -------------*/
/* increment the sequence number of the data blocks and
   their corresponding acknowledgments */
char inc (char) ;

int StopSending (void) ;
/*-----------------------------------------------*/

/* main program loop */

void Data_Block_Excng ( ) {

s = socket(PF_INET, SOCK_DGRAM, 0) ;      /* get a UDP socket */
if (s == INVALID_SOCKET) {
      /* if error occurs close connection and socket   */
      (WSAperror (WSAGetLastError ( ), "socket") ;
      nRet = closesocket (s) ;
      if (nRet == SOCKET_ERROR) {
      /* report the error to the user and abord program*/
            WSAperror (WSAGetLastError ( ), "closesocket") ;
            return ;
   }
}

/* initialize destination (server's) socket address structure */

stRmtName.sin_family = PF_INET ;         /* TCP/IP suite */
stRmtName.sin_port = htons(PORT) ;    /* port number in network
                                                     order */
stRmtName.sin_addr = INADDR_ANY ;  /* request the stack to assign
                              the local IP address automatically */

/* connect to server (just to inform the network system that this UDP
connection will send datagrams to the same destination socket);  */

nRet = connect (s, (LPSOCKADDR)&stRmtName, sizeof (SOCKADDR) ) ;

/* on a UDP socket connect ( ) does not fail, because the socket does
not access the network */

/* set the T_O (1) time-out value */
int nOptVal =timeOutOne ;
setsockopt (INVALID_SOCKET, SOL_SOCKET, SO_RCVTIMEO, (char
FAR*)&nOptVal, sizeof (int));

/* main data exchange loop */
for (;;) {

send (s, (LPSTR)&CntrlBuf, _fstrlen (CntrlBuf), 0) ;
nRet = recv (s, (LPSTR)&InBuffer, Buflength, 0) ;
```

```
if (nRet == SOCKET_ERROR) {
      nWSAerror = WSAGetLastError ( ) ;
      if (nWSAerror == WSATIMEDOUT) {
       /* we had a time-out on a blocking operation and we want
                              the application to cancel it */
            if (WSAIsBlocking ( ) ) {     /* determine if a blocking
                                             call is in progress */
                  /* cancel the blocking call */
                  WSACancelBlockingCall ( ) ;
       }
            continue ;
      }else {
      closesocket (s) ;        /* close socket and abord */
      return ;
 }
/* if no time-out occured */
while (!STOP_SENDING) {

/* a 'NR' message was received ? */
if (InBuffer[0] == 110 || InBuffer[0] ==78) {
      nOptVal = timeOutThree ;
      setsockopt (INVALID_SOCKET, SOL_SOCKET, SO_RCVTIMEO, (char
      FAR*)&nOptVal, sizeof (int)) ;
      nRet = recv (s, (LPSTR)&InBuffer, Buflength, 0) ;
      if (nRet == SOCKET_ERROR) {
            nWSAerror = WSAGetLastError ( ) ;
            if  (nWSAerror == WSATIMEDOUT) {
                  WSACancelBlockingCall ( ) ; /* cancel blocking
                              call and return to state 0 */
                  wsprintf ("Time-out(3). Returning to state 0") ;
                  break ;     /* get out of the while ( ) loop */
           } else {
            /*report the error to the user */
            WSAperror (WSAErr, " recv ( ) " ) ;
            closesocket (s) ;
           return ;                 /* fatal error, end program */
            }
      goto data ;
     }
}


data :
/* data block exchange branch-check for correct sequence first */

if (InBuffer[0] == frame_expected) {
      ackBuffer = frame_expected ;
      send (s, (LPSTR)&ackBuffer, 1, 0) ; /* send proper ACK */
      inc (frame_expected) ;
} else {
      ackBuffer = frame_expected ;
    STOP_SENDING = StopSending ( ) ;
}
recv (s, (LPSTR)&InBuffer, Buflength, 0) ;

}            /* while loop ends */
```

```
send (send (s, (LPSTR)&CntrlBuffer, 1, 0) ; /* send a STOP_SENDING
                                                  message */
break ;
}                  /* for loop ends */

closesocket (s) ;
return ;
}         /* Data_Block_Excng ends */



/*------------- Function definitions ---------------*/

/**********************/
char inc (char)
/**********************/
{
if (frame_expected == '0') {
      frame_expected = '1' ;
}else{
      frame_expected = '0'    }
return frame_expected ;
}

/**********************/
int stopSending ( )
/**********************/
{
HFILE hFile ;       /* file descriptor */
char a ;
SOCKET s ;
hFile =_lopen(Cmmnd_file, 0) ;    /* open Cmmnd_file for read */
a =_lread (hFile, StopSendBuffer, 1) ; /* read a 1-byte character
                                                        command */
if (a == HFILE_ERROR) {
      MessageBox (hWinMain, "Error reading Cmmnd_file") ;
      closesocket (s) ;
      return ;
} else {
if ( StopSendBuffer == 's' || StopSendBuffer == 'S' ) {
      return TRUE;
else
      return FALSE;
    }
}
/*-----------------------------------------------------*/
```

```
/*    The Data_Excng ( ) is the server application that implements
the main data exchange branch (rotary vehicle's) of the
communication protocol showed as a FSM in Figure 25. The server's
code also follows the Winsock programming conventions. The server
is implemented as an endless loop that ends upon reception of a
STOP_SENDING (1-byte character) message. During this loop the server
application continuously opens the 'DataFile' to determine data
block availability  (passed in this file by another application
program) and load the output buffers.  Data blocks are again
character strings of 1500 bytes. The first character (byte) always
determines the sequence number of the data block and contains no
actual information. The NR (not ready) message is implemented by a
control character buffer that holds the 'N' character. Time-outs are
again implemented by the use of the setsockopt ( ) function. To
compensate for the delays of the code implementation of Figure's 23
protocol, the T_O(3) timer has a 1 sec greater value than the same
timer in the client application program.
      The server's service port has been assigned just for the
purpose of code completion and belongs in the typical range of user-
defined services as indicated by the Internet Assigned Numbers
Authority (IANA) (revision RFC 1700)   */



#include <windows.h>
#include <winsock.h>
#include <winsockx.h>
#include <stdio.h>
#define BUFSIZE 1500 ;
#define PORT 3600 ;     /* the port that the server listens for
                                               connections */

/*------------- Global variables -----------------*/
SOCKET s;
SOCKADDR_IN stLclName ;
SOCKADDR_IN stRmtName ;
static char OutBuffer[BUFSIZE] ;    /* output data (blocks) buffer */
static char CntrlBuf = 'N' ;  /* the NR message */
static char InBuffer [2] ;    /* ACKs and control messages */
char ackTempBuffer [20] ;
extern HFILE hFile            /* file handle for open data file  */

int nRet ;
int nOptVal ;
int Buflength = 1500 ;
BOOL STOP_SENDING = FALSE ;
BOOL time_out_with_out_data = FALSE ;
extern char next_frame_to_send  = '0' ; /* frame sequence No.*/
char ackBuffer ;
int timeOutFour = 500 ;  /* time-out values for T_O(4), T_O(3)*/
int timeOutThree = 5000 ;
```

```
/*------------- Function Prototypes ---------------*/
char inc (char) ;
/*-------------------------------------------------*/

/* main program begins */

void Data_Excng ( ) {

s = socket(PF_INET, SOCK_DGRAM, 0) ; /* get a UDP socket */
if (s == INVALID_SOCKET) {
/* if it is invalid socket report error to user */
      (WSAperror (WSAGetLastError ( ), "socket") ;
      nRet = closesocket (s) ; /* close connection and socket */
      return ;
   }

/* initialize local (server's) socket address structure, to provide for
the client assosiation */

stLclName.sin_family = PF_INET ;          /* TCP/IP suite */
stLclName.sin_port = htons (PORT) ;    /* server's port number in
                                                network order */
stLclName.sin_addr = INADDR_ANY ;    /* request the stack to assign the
                                   local IP address automatically */

/* name the local socket with the values in the sockaddr_in structure */
nRet = bind (s,(LPSOCKADDR)&stLclName, sizeof (struct sockaddr));

/* since the server's socket name is unique (port number and IP
address), this function will not fail */
/* other applications will not try to bound to the same socket name */

/* an endless loop implementing the server */
for ( ;; ) {

nRet = recv (s, (LPSTR)&InBuffer, 2, 0 ) ;  /* infinite blooking hook */

/* waiting for the DR message */
if  ((nRet == 2) && (InBuffer [0] == 'D' && InBuffer[1] == 'R')) {
      while ( !STOP_SENDING) {
      hFile = _lopen(Datafile, 0); /* open file containing the
                                   data blocks for read */

      if ( hFile == HFILE_ERROR) {
            wsprintf (achTempBuf, "Unable to open the Datafile ");
            MessageBox (hWinMain, (LPSTR)achTempBuf ,"Error
            reading Datafile") ;
            _lclose (hFile) ;
            closesocket (s) ;       /* close socket and abord */
            return ; }

      /* if file is readable */
      fRet =_lread (hFile, OutBuffer, 1500) ;   /* read 1500
                               bytes in the output buffers */
   if  (!fRet ) {          /* check if we have data to send */

                                   /* if the file is empty */
            for ( ;; ) {

             /* send a NR message ('N' character) to the client */
             send  (s,(LPSTR)&CntrlBuf , _fstrlen(CntrlBuf),0);
             InBuffer [0] = 0;
             InBuffer [1] = 0;        /* clear input buffers */

             /* set time-out T_O(3) */
             nOptVal = timeOutThree +1000 ;
             setsockopt (INVALID_SOCKET, SOL_SOCKET, SO_RCVTIMEO,
             (char FAR*)&nOptVal, sizeof (int)) ;
             nret = recv (s, (LPSTR)&InBuffer, 2, 0 ) ;
             if  ((nRet == 2) && (InBuffer [0] == 'D' && InBuffer[1] ==
             'R'))
                    continue ;
                    else if (nRet == SOCKET_ERROR) {
                    nWSAerror = WSAGetLastError ( ) ;
                    if (nWSAerror == WSATIMEDOUT) {
                    WSACancelBlockingCall ( ) ;
                    fRet =_lread (hFile, OutBuffer, 1500) ;

                           /* check for data availability */
                    if (!fRet)
                           break ;
                    else {time_out_with_out_data = TRUE ;
                           break ;  }
             else {WSAperror (WSAErr, " recv ( ) " ) ;
                    closesocket (s) ;
                    return ;   }
                }
           }

data :                     /* we have data to send */

if (time_out_with_out_data) break ;
OutBuffer [0] = next_frame_to_send ;

while ( !STOP_SENDING ) { /* sending data loop */

   send (s, (LPSTR)&OutBuffer, 1500, 1) ;
   nOptVal = timeOutFour ;     /* set time-out T_O(4) */
   setsockopt (INVALID_SOCKET, SOL_SOCKET, SO_RCVTIMEO, (char
   FAR*)&nOptVal, sizeof (int)) ;
   nRet = recv (s, (LPSTR)&InBuffer, 1, 0 ) ;
   if   (nRet == SOCKET_ERROR) {
        nWSAerror = WSAGetLastError ( ) ;
        if  (nWSAerror == WSATIMEDOUT) {
             WSACancelBlockingCall ( ) ;
             continue ; }
       else {WSAperror (WSAErr, " recv ( ) " ) ;
             closesocket (s) ;
             return ;   }
if  ( InBuffer [0] == 'S' || InBuffer [0] == 's' ) {
       STOP_SENDING = TRUE; }
if   ( InBuffer [0] != next_frame_to_send ) {     /* we are OK */
      break ;}
     }

if ( time_out_with_out_data) {
     break ; }
inc (next_frame_to_send) ;
  }
}
return ;
}
/* end of Data_Excng ( ) */


/*------------- Function definitions ---------------*/

/**********************/
char inc (char)
/**********************/
{
if (next_frame_to_send == '0') {
   next_frame_to_send = '1' ;
} else {
   next_frame_to_send = '0'    }
return next_frame_to_send ;
}
```

APPENDIX B. WINSOCK HEADER FILE AND DEFINITIONS

```
/* WINSOCK.H--definitions to be used with the WINSOCK.DLL
 * This header file corresponds to version 1.1 of the Windows Sockets
 * specification.
 * This file includes parts which are Copyright (c) 1982-1986 Regents
 * of the University of California.  All rights reserved.  The
 * Berkeley Software License Agreement specifies the terms and
 * conditions for redistribution.
 */

#ifndef _WINSOCKAPI_
#define _WINSOCKAPI_

/*
 * Pull in WINDOWS.H if necessary
 */
#ifndef _INC_WINDOWS
#include <windows.h>
#endif /* _INC_WINDOWS */

/*
 * Basic system type definitions, taken from the BSD file sys/types.h.
 */
typedef unsigned char   u_char;
typedef unsigned short  u_short;
typedef unsigned int    u_int;
typedef unsigned long   u_long;

/*
 * The new type to be used in all
 * instances which refer to sockets.
 */
typedef u_int           SOCKET;

/*
 * Select uses arrays of SOCKETs.  These macros manipulate such
 * arrays.  FD_SETSIZE may be defined by the user before including
 * this file, but the default here should be >= 64.
 *
 * CAVEAT IMPLEMENTOR and USER: THESE MACROS AND TYPES MUST BE
 * INCLUDED IN WINSOCK.H EXACTLY AS SHOWN HERE.
 */
#ifndef FD_SETSIZE
#define FD_SETSIZE      64
#endif /* FD_SETSIZE */

typedef struct fd_set {
        u_short fd_count;               /* how many are SET? */
        SOCKET  fd_array[FD_SETSIZE];   /* an array of SOCKETs */
}fd_set;

#ifdef __cplusplus
extern "C" {
#endif
extern int PASCAL FAR __WSAFDIsSet(SOCKET, fd_set FAR *);

#ifdef __cplusplus
}
#endif

#define FD_CLR(fd, set) do {\
    u_int __i; \
    for (__i = 0; __i < ((fd_set FAR *)(set))->fd_count ; __i++) {\
        if (((fd_set FAR *)(set))->fd_array[__i] == fd) {\
            while (__i < ((fd_set FAR *)(set))->fd_count-1) {\
                ((fd_set FAR *)(set))->fd_array[__i] = \
                    ((fd_set FAR *)(set))->fd_array[__i+1]; \
                __i++; \
            }\
            ((fd_set FAR *)(set))->fd_count--; \
            break; \
        }\
    }\
}while(0)

#define FD_SET(fd, set) do {\
    if (((fd_set FAR *)(set))->fd_count < FD_SETSIZE) \
        ((fd_set FAR *)(set))->fd_array[((fd_set FAR *)(set))->fd_count++]=fd;\
}while(0)

#define FD_ZERO(set) (((fd_set FAR *)(set))->fd_count=0)

#define FD_ISSET(fd, set) __WSAFDIsSet((SOCKET)fd, (fd_set FAR *)set)

/*
 * Structure used in select() call, taken from the BSD file sys/time.h.
 */
struct timeval {
        long    tv_sec;         /* seconds */
        long    tv_usec;        /* and microseconds */
};

/*
 * Operations on timevals.
 *
 * NB: timercmp does not work for >= or <=.
 */
#define timerisset(tvp)         ((tvp)->tv_sec || (tvp)->tv_usec)
#define timercmp(tvp, uvp, cmp) \
        ((tvp)->tv_sec cmp (uvp)->tv_sec || \
         (tvp)->tv_sec == (uvp)->tv_sec && (tvp)->tv_usec cmp (uvp)->tv_usec)
#define timerclear(tvp)         (tvp)->tv_sec = (tvp)->tv_usec = 0

/*
 * Commands for ioctlsocket(),  taken from the BSD file fcntl.h.
 *
 *
 * Ioctl's have the command encoded in the lower word,
 * and the size of any in or out parameters in the upper
 * word.  The high 2 bits of the upper word are used
 * to encode the in/out status of the parameter; for now
 * we restrict parameters to at most 128 bytes.
 */
#define IOCPARM_MASK    0x7f            /* parameters must be < 128 bytes */
#define IOC_VOID        0x20000000      /* no parameters */
#define IOC_OUT         0x40000000      /* copy out parameters */
#define IOC_IN          0x80000000      /* copy in parameters */
#define IOC_INOUT       (IOC_IN|IOC_OUT)
                                        /* 0x20000000 distinguishes new &
                                           old ioctl's */
#define _IO(x,y)        (IOC_VOID|(x<<8)|y)

#define _IOR(x,y,t)     (IOC_OUT|(((long)sizeof(t)&IOCPARM_MASK)<<16)|(x<<8)|y)

#define _IOW(x,y,t)     (IOC_IN|(((long)sizeof(t)&IOCPARM_MASK)<<16)|(x<<8)|y)

#define FIONREAD    _IOR('f', 127, u_long) /* get # bytes to read */
#define FIONBIO     _IOW('f', 126, u_long) /* set/clear non-blocking i/o */
#define FIOASYNC    _IOW('f', 125, u_long) /* set/clear async i/o */

/* Socket I/O Controls */
#define SIOCSHIWAT  _IOW('s',  0, u_long)  /* set high watermark */
#define SIOCGHIWAT  _IOR('s',  1, u_long)  /* get high watermark */
#define SIOCSLOWAT  _IOW('s',  2, u_long)  /* set low watermark */
#define SIOCGLOWAT  _IOR('s',  3, u_long)  /* get low watermark */
#define SIOCATMARK  _IOR('s',  7, u_long)  /* at oob mark? */

/*
 * Structures returned by network data base library, taken from the
 * BSD file netdb.h.  All addresses are supplied in host order, and
 * returned in network order (suitable for use in system calls).
 */

struct  hostent {
        char    FAR * h_name;           /* official name of host */
        char    FAR * FAR * h_aliases;  /* alias list */
        short   h_addrtype;             /* host address type */
        short   h_length;               /* length of address */
        char    FAR * FAR * h_addr_list; /* list of addresses */
#define h_addr  h_addr_list[0]          /* address, for backward compat */
};

/*
 * It is assumed here that a network number
 * fits in 32 bits.
 */
struct  netent {
        char    FAR * n_name;           /* official name of net */
        char    FAR * FAR * n_aliases;  /* alias list */
        short   n_addrtype;             /* net address type */
        u_long  n_net;                  /* network # */
};

struct  servent {
        char    FAR * s_name;           /* official service name */
        char    FAR * FAR * s_aliases;  /* alias list */
        short   s_port;                 /* port # */
        char    FAR * s_proto;          /* protocol to use */
};
```

```
struct  protoent {
        char    FAR * p_name;           /* official protocol name */
        char    FAR * FAR * p_aliases;  /* alias list */
        short   p_proto;                /* protocol # */
};

/*
 * Constants and structures defined by the internet system,
 * Per RFC 790, September 1981, taken from the BSD file netinet/in.h.
 */

/*
 * Protocols
 */
#define IPPROTO_IP              0               /* dummy for IP */
#define IPPROTO_ICMP            1               /* control message protocol */
#define IPPROTO_GGP             2               /* gateway^2 (deprecated) */
#define IPPROTO_TCP             6               /* tcp */
#define IPPROTO_PUP             12              /* pup */
#define IPPROTO_UDP             17              /* user datagram protocol */
#define IPPROTO_IDP             22              /* xns idp */
#define IPPROTO_ND              77              /* UNOFFICIAL net disk proto */

#define IPPROTO_RAW             255             /* raw IP packet */
#define IPPROTO_MAX             256

/*
 * Port/socket numbers: network standard functions
 */
#define IPPORT_ECHO             7
#define IPPORT_DISCARD          9
#define IPPORT_SYSTAT           11
#define IPPORT_DAYTIME          13
#define IPPORT_NETSTAT          15
#define IPPORT_FTP              21
#define IPPORT_TELNET           23
#define IPPORT_SMTP             25
#define IPPORT_TIMESERVER       37
#define IPPORT_NAMESERVER       42
#define IPPORT_WHOIS            43
#define IPPORT_MTP              57

/*
 * Port/socket numbers: host specific functions
 */
#define IPPORT_TFTP             69
#define IPPORT_RJE              77
#define IPPORT_FINGER           79
#define IPPORT_TTYLINK          87
#define IPPORT_SUPDUP           95

/*
 * UNIX TCP sockets
 */
#define IPPORT_EXECSERVER       512
#define IPPORT_LOGINSERVER      513
#define IPPORT_CMDSERVER        514
#define IPPORT_EFSSERVER        520
```

```
/*
 * UNIX UDP sockets
 */
#define IPPORT_BIFFUDP          512
#define IPPORT_WHOSERVER        513
#define IPPORT_ROUTESERVER      520
                                        /* 520+1 also used */

/*
 * Ports < IPPORT_RESERVED are reserved for
 * privileged processes (e.g. root).
 */
#define IPPORT_RESERVED         1024

/*
 * Link numbers
 */
#define IMPLINK_IP              155
#define IMPLINK_LOWEXPER        156
#define IMPLINK_HIGHEXPER       158

/*
 * Internet address (old style... should be updated)
 */
struct in_addr {
        union {
                struct {u_char s_b1,s_b2,s_b3,s_b4; }S_un_b;
                struct {u_short s_w1,s_w2; }S_un_w;
                u_long S_addr;
        }S_un;
#define s_addr  S_un.S_addr
                                /* can be used for most tcp & ip code */
#define s_host  S_un.S_un_b.s_b2
                                /* host on imp */
#define s_net   S_un.S_un_b.s_b1
                                /* network */
#define s_imp   S_un.S_un_w.s_w2
                                /* imp */
#define s_impno S_un.S_un_b.s_b4
                                /* imp # */
#define s_lh    S_un.S_un_b.s_b3
                                /* logical host */
};

/*
 * Definitions of bits in internet address integers.
 * On subnets, the decomposition of addresses to host and net parts
 * is done according to subnet mask, not the masks here.
 */
#define IN_CLASSA(i)            (((long)(i) & 0x80000000) == 0)
#define IN_CLASSA_NET           0xff000000
#define IN_CLASSA_NSHIFT        24
#define IN_CLASSA_HOST          0x00ffffff
#define IN_CLASSA_MAX           128

#define IN_CLASSB(i)            (((long)(i) & 0xc0000000) == 0x80000000)
#define IN_CLASSB_NET           0xffff0000
#define IN_CLASSB_NSHIFT        16
```

```
#define IN_CLASSB_HOST          0x0000ffff
#define IN_CLASSB_MAX           65536

#define IN_CLASSC(i)            (((long)(i) & 0xe0000000) == 0xc0000000)
#define IN_CLASSC_NET           0xffffff00
#define IN_CLASSC_NSHIFT        8
#define IN_CLASSC_HOST          0x000000ff

#define INADDR_ANY              (u_long)0x00000000
#define INADDR_LOOPBACK         0x7f000001
#define INADDR_BROADCAST        (u_long)0xffffffff
#define INADDR_NONE             0xffffffff

/*
 * Socket address, internet style.
 */
struct sockaddr_in {
        short   sin_family;
        u_short sin_port;
        struct  in_addr sin_addr;
        char    sin_zero[8];
};

#define WSADESCRIPTION_LEN      256
#define WSASYS_STATUS_LEN       128

typedef struct WSAData {
        WORD                    wVersion;
        WORD                    wHighVersion;
        char                    szDescription[WSADESCRIPTION_LEN+1];
        char                    szSystemStatus[WSASYS_STATUS_LEN+1];
        unsigned short          iMaxSockets;
        unsigned short          iMaxUdpDg;
        char FAR *              lpVendorInfo;
}WSADATA;

typedef WSADATA FAR *LPWSADATA;

/*
 * Options for use with [gs]etsockopt at the IP level.
 */
#define IP_OPTIONS      1               /* set/get IP per-packet options */

/*
 * Definitions related to sockets: types, address families, options,
 * taken from the BSD file sys/socket.h.
 */

/*
 * This is used instead of -1, since the
 * SOCKET type is unsigned.
 */
#define INVALID_SOCKET  (SOCKET)(~0)
#define SOCKET_ERROR            (-1)

/*
 * Types
 */
```

```
#define SOCK_STREAM     1               /* stream socket */
#define SOCK_DGRAM      2               /* datagram socket */
#define SOCK_RAW        3               /* raw-protocol interface */
#define SOCK_RDM        4               /* reliably-delivered message */
#define SOCK_SEQPACKET  5               /* sequenced packet stream */

/*
 * Option flags per-socket.
 */
#define SO_DEBUG        0x0001          /* turn on debugging info recording */
#define SO_ACCEPTCONN   0x0002          /* socket has had listen() */
#define SO_REUSEADDR    0x0004          /* allow local address reuse */
#define SO_KEEPALIVE    0x0008          /* keep connections alive */
#define SO_DONTROUTE    0x0010          /* just use interface addresses */
#define SO_BROADCAST    0x0020          /* permit sending of broadcast msgs */
#define SO_USELOOPBACK  0x0040          /* bypass hardware when possible */
#define SO_LINGER       0x0080          /* linger on close if data present */
#define SO_OOBINLINE    0x0100          /* leave received OOB data in line */

#define SO_DONTLINGER   (u_int)(~SO_LINGER)

/*
 * Additional options.
 */
#define SO_SNDBUF       0x1001          /* send buffer size */
#define SO_RCVBUF       0x1002          /* receive buffer size */
#define SO_SNDLOWAT     0x1003          /* send low-water mark */
#define SO_RCVLOWAT     0x1004          /* receive low-water mark */
#define SO_SNDTIMEO     0x1005          /* send timeout */
#define SO_RCVTIMEO     0x1006          /* receive timeout */
#define SO_ERROR        0x1007          /* get error status and clear */
#define SO_TYPE         0x1008          /* get socket type */

/*
 * TCP options.
 */
#define TCP_NODELAY     0x0001

/*
 * Address families.
 */
#define AF_UNSPEC       0               /* unspecified */
#define AF_UNIX         1               /* local to host (pipes, portals) */
#define AF_INET         2               /* internetwork: UDP, TCP, etc. */
#define AF_IMPLINK      3               /* arpanet imp addresses */
#define AF_PUP          4               /* pup protocols: e.g. BSP */
#define AF_CHAOS        5               /* mit CHAOS protocols */
#define AF_NS           6               /* XEROX NS protocols */
#define AF_IPX          6               /* IPX and SPX */
#define AF_ISO          7               /* ISO protocols */
#define AF_OSI          AF_ISO          /* OSI is ISO */
#define AF_ECMA         8               /* european computer manufacturers */
#define AF_DATAKIT      9               /* datakit protocols */
#define AF_CCITT        10              /* CCITT protocols, X.25 etc */
#define AF_SNA          11              /* IBM SNA */
#define AF_DECnet       12              /* DECnet */
#define AF_DLI          13              /* Direct data link interface */
#define AF_LAT          14              /* LAT */
```

```
#define AF_HYLINK       15              /* NSC Hyperchannel */
#define AF_APPLETALK    16              /* AppleTalk */
#define AF_NETBIOS      17              /* NetBios-style addresses */

#define AF_MAX          18

/*
 * Structure used by kernel to store most
 * addresses.
 */
struct sockaddr {
        u_short sa_family;              /* address family */
        char    sa_data[14];            /* up to 14 bytes of direct address */
};

/*
 * Structure used by kernel to pass protocol
 * information in raw sockets.
 */
struct sockproto {
        u_short sp_family;              /* address family */
        u_short sp_protocol;            /* protocol */
};

/*
 * Protocol families, same as address families for now.
 */
#define PF_UNSPEC       AF_UNSPEC
#define PF_UNIX         AF_UNIX
#define PF_INET         AF_INET
#define PF_IMPLINK      AF_IMPLINK
#define PF_PUP          AF_PUP
#define PF_CHAOS        AF_CHAOS
#define PF_NS           AF_NS
#define PF_IPX          AF_IPX
#define PF_ISO          AF_ISO
#define PF_OSI          AF_OSI
#define PF_ECMA         AF_ECMA
#define PF_DATAKIT      AF_DATAKIT
#define PF_CCITT        AF_CCITT
#define PF_SNA          AF_SNA
#define PF_DECnet       AF_DECnet
#define PF_DLI          AF_DLI
#define PF_LAT          AF_LAT
#define PF_HYLINK       AF_HYLINK
#define PF_APPLETALK    AF_APPLETALK

#define PF_MAX          AF_MAX

/*
 * Structure used for manipulating linger option.
 */
struct  linger {
        u_short l_onoff;                /* option on/off */
        u_short l_linger;               /* linger time */
};

/*
```

```
 * Level number for (get/set)sockopt() to apply to socket itself.
 */
#define SOL_SOCKET      0xffff          /* options for socket level */

/*
 * Maximum queue length specifiable by listen.
 */
#define SOMAXCONN       5

#define MSG_OOB         0x1             /* process out-of-band data */
#define MSG_PEEK        0x2             /* peek at incoming message */
#define MSG_DONTROUTE   0x4             /* send without using routing tables */

#define MSG_MAXIOVLEN   16

/*
 * Define constant based on rfc883, used by gethostbyxxxx() calls.
 */
#define MAXGETHOSTSTRUCT        1024

/*
 * Define flags to be used with the WSAAsyncSelect() call.
 */
#define FD_READ         0x01
#define FD_WRITE        0x02
#define FD_OOB          0x04
#define FD_ACCEPT       0x08
#define FD_CONNECT      0x10
#define FD_CLOSE        0x20

/*
 * All Windows Sockets error constants are biased by WSABASEERR from
 * the "normal"
 */
#define WSABASEERR              10000
/*
 * Windows Sockets definitions of regular Microsoft C error constants
 */
#define WSAEINTR                (WSABASEERR+4)
#define WSAEBADF                (WSABASEERR+9)
#define WSAEACCES               (WSABASEERR+13)
#define WSAEFAULT               (WSABASEERR+14)
#define WSAEINVAL               (WSABASEERR+22)
#define WSAEMFILE               (WSABASEERR+24)

/*
 * Windows Sockets definitions of regular Berkeley error constants
 */
#define WSAEWOULDBLOCK          (WSABASEERR+35)
#define WSAEINPROGRESS          (WSABASEERR+36)
#define WSAEALREADY             (WSABASEERR+37)
#define WSAENOTSOCK             (WSABASEERR+38)
#define WSAEDESTADDRREQ         (WSABASEERR+39)
#define WSAEMSGSIZE             (WSABASEERR+40)
#define WSAEPROTOTYPE           (WSABASEERR+41)
#define WSAENOPROTOOPT          (WSABASEERR+42)
#define WSAEPROTONOSUPPORT      (WSABASEERR+43)
#define WSAESOCKTNOSUPPORT      (WSABASEERR+44)
```

```
#define WSAEOPNOTSUPP           (WSABASEERR+45)
#define WSAEPFNOSUPPORT         (WSABASEERR+46)
#define WSAEAFNOSUPPORT         (WSABASEERR+47)
#define WSAEADDRINUSE           (WSABASEERR+48)
#define WSAEADDRNOTAVAIL        (WSABASEERR+49)
#define WSAENETDOWN             (WSABASEERR+50)
#define WSAENETUNREACH          (WSABASEERR+51)
#define WSAENETRESET            (WSABASEERR+52)
#define WSAECONNABORTED         (WSABASEERR+53)
#define WSAECONNRESET           (WSABASEERR+54)
#define WSAENOBUFS              (WSABASEERR+55)
#define WSAEISCONN              (WSABASEERR+56)
#define WSAENOTCONN             (WSABASEERR+57)
#define WSAESHUTDOWN            (WSABASEERR+58)
#define WSAETOOMANYREFS         (WSABASEERR+59)
#define WSAETIMEDOUT            (WSABASEERR+60)
#define WSAECONNREFUSED         (WSABASEERR+61)
#define WSAELOOP                (WSABASEERR+62)
#define WSAENAMETOOLONG         (WSABASEERR+63)
#define WSAEHOSTDOWN            (WSABASEERR+64)
#define WSAEHOSTUNREACH         (WSABASEERR+65)
#define WSAENOTEMPTY            (WSABASEERR+66)
#define WSAEPROCLIM             (WSABASEERR+67)
#define WSAEUSERS               (WSABASEERR+68)
#define WSAEDQUOT               (WSABASEERR+69)
#define WSAESTALE               (WSABASEERR+70)
#define WSAEREMOTE              (WSABASEERR+71)

/*
 * Extended Windows Sockets error constant definitions
 */
#define WSASYSNOTREADY          (WSABASEERR+91)
#define WSAVERNOTSUPPORTED      (WSABASEERR+92)
#define WSANOTINITIALISED       (WSABASEERR+93)

/*
 * Error return codes from gethostbyname() and gethostbyaddr()
 * (when using the resolver). Note that these errors are
 * retrieved via WSAGetLastError() and must therefore follow
 * the rules for avoiding clashes with error numbers from
 * specific implementations or language run-time systems.
 * For this reason the codes are based at WSABASEERR+1001.
 * Note also that [WSA]NO_ADDRESS is defined only for
 * compatibility purposes.
 */

#define h_errno         WSAGetLastError()

/* Authoritative Answer: Host not found */
#define WSAHOST_NOT_FOUND       (WSABASEERR+1001)
#define HOST_NOT_FOUND          WSAHOST_NOT_FOUND

/* Non-Authoritative: Host not found, or SERVERFAIL */
#define WSATRY_AGAIN            (WSABASEERR+1002)
#define TRY_AGAIN               WSATRY_AGAIN

/* Non recoverable errors, FORMERR, REFUSED, NOTIMP */
#define WSANO_RECOVERY          (WSABASEERR+1003)
```

```
#define NO_RECOVERY             WSANO_RECOVERY

/* Valid name, no data record of requested type */
#define WSANO_DATA              (WSABASEERR+1004)
#define NO_DATA                 WSANO_DATA

/* no address, look for MX record */
#define WSANO_ADDRESS           WSANO_DATA
#define NO_ADDRESS              WSANO_ADDRESS

/*
 * Windows Sockets errors redefined as regular Berkeley error constants
 */
#define EWOULDBLOCK             WSAEWOULDBLOCK
#define EINPROGRESS             WSAEINPROGRESS
#define EALREADY                WSAEALREADY
#define ENOTSOCK                WSAENOTSOCK
#define EDESTADDRREQ            WSAEDESTADDRREQ
#define EMSGSIZE                WSAEMSGSIZE
#define EPROTOTYPE              WSAEPROTOTYPE
#define ENOPROTOOPT             WSAENOPROTOOPT
#define EPROTONOSUPPORT         WSAEPROTONOSUPPORT
#define ESOCKTNOSUPPORT         WSAESOCKTNOSUPPORT
#define EOPNOTSUPP              WSAEOPNOTSUPP
#define EPFNOSUPPORT            WSAEPFNOSUPPORT
#define EAFNOSUPPORT            WSAEAFNOSUPPORT
#define EADDRINUSE              WSAEADDRINUSE
#define EADDRNOTAVAIL           WSAEADDRNOTAVAIL
#define ENETDOWN                WSAENETDOWN
#define ENETUNREACH             WSAENETUNREACH
#define ENETRESET               WSAENETRESET
#define ECONNABORTED            WSAECONNABORTED
#define ECONNRESET              WSAECONNRESET
#define ENOBUFS                 WSAENOBUFS
#define EISCONN                 WSAEISCONN
#define ENOTCONN                WSAENOTCONN
#define ESHUTDOWN               WSAESHUTDOWN
#define ETOOMANYREFS            WSAETOOMANYREFS
#define ETIMEDOUT               WSAETIMEDOUT
#define ECONNREFUSED            WSAECONNREFUSED
#define ELOOP                   WSAELOOP
#define ENAMETOOLONG            WSAENAMETOOLONG
#define EHOSTDOWN               WSAEHOSTDOWN
#define EHOSTUNREACH            WSAEHOSTUNREACH
#define ENOTEMPTY               WSAENOTEMPTY
#define EPROCLIM                WSAEPROCLIM
#define EUSERS                  WSAEUSERS
#define EDQUOT                  WSAEDQUOT
#define ESTALE                  WSAESTALE
#define EREMOTE                 WSAEREMOTE

/* Socket function prototypes */

#ifdef __cplusplus
extern "C" {
#endif

SOCKET PASCAL FAR accept (SOCKET s, struct sockaddr FAR *addr,
                              int FAR *addrlen);

int PASCAL FAR bind (SOCKET s, const struct sockaddr FAR *addr, int namelen);

int PASCAL FAR closesocket (SOCKET s);

int PASCAL FAR connect (SOCKET s, const struct sockaddr FAR *name, int namelen);

int PASCAL FAR ioctlsocket (SOCKET s, long cmd, u_long FAR *argp);

int PASCAL FAR getpeername (SOCKET s, struct sockaddr FAR *name,
                            int FAR * namelen);

int PASCAL FAR getsockname (SOCKET s, struct sockaddr FAR *name,
                            int FAR * namelen);

int PASCAL FAR getsockopt (SOCKET s, int level, int optname,
                           char FAR * optval, int FAR *optlen);

u_long PASCAL FAR htonl (u_long hostlong);

u_short PASCAL FAR htons (u_short hostshort);

unsigned long PASCAL FAR inet_addr (const char FAR * cp);

char FAR * PASCAL FAR inet_ntoa (struct in_addr in);

int PASCAL FAR listen (SOCKET s, int backlog);

u_long PASCAL FAR ntohl (u_long netlong);

u_short PASCAL FAR ntohs (u_short netshort);

int PASCAL FAR recv (SOCKET s, char FAR * buf, int len, int flags);

int PASCAL FAR recvfrom (SOCKET s, char FAR * buf, int len, int flags,
                         struct sockaddr FAR *from, int FAR * fromlen);

int PASCAL FAR select (int nfds, fd_set FAR *readfds, fd_set FAR *writefds,
                       fd_set FAR *exceptfds, const struct timeval FAR *timeout);

int PASCAL FAR send (SOCKET s, const char FAR * buf, int len, int flags);

int PASCAL FAR sendto (SOCKET s, const char FAR * buf, int len, int flags,
                       const struct sockaddr FAR *to, int tolen);

int PASCAL FAR setsockopt (SOCKET s, int level, int optname,
                           const char FAR * optval, int optlen);

int PASCAL FAR shutdown (SOCKET s, int how);

SOCKET PASCAL FAR socket (int af, int type, int protocol);

/* Database function prototypes */

struct hostent FAR * PASCAL FAR gethostbyaddr(const char FAR * addr,
                                              int len, int type);
```

```
struct hostent FAR * PASCAL FAR gethostbyname(const char FAR * name);

int PASCAL FAR gethostname (char FAR * name, int namelen);

struct servent FAR * PASCAL FAR getservbyport(int port, const char FAR * proto);

struct servent FAR * PASCAL FAR getservbyname(const char FAR * name,
                                              const char FAR * proto);

struct protoent FAR * PASCAL FAR getprotobynumber(int proto);

struct protoent FAR * PASCAL FAR getprotobyname(const char FAR * name);

/* Microsoft Windows Extension function prototypes */

int PASCAL FAR WSAStartup(WORD wVersionRequired, LPWSADATA lpWSAData);

int PASCAL FAR WSACleanup(void);

void PASCAL FAR WSASetLastError(int iError);

int PASCAL FAR WSAGetLastError(void);

BOOL PASCAL FAR WSAIsBlocking(void);

int PASCAL FAR WSAUnhookBlockingHook(void);

FARPROC PASCAL FAR WSASetBlockingHook(FARPROC lpBlockFunc);

int PASCAL FAR WSACancelBlockingCall(void);

HANDLE PASCAL FAR WSAAsyncGetServByName(HWND hWnd, u_int wMsg,
                                        const char FAR * name,
                                        const char FAR * proto,
                                        char FAR * buf, int buflen);

HANDLE PASCAL FAR WSAAsyncGetServByPort(HWND hWnd, u_int wMsg, int port,
                                       const char FAR * proto, char FAR * buf,
                                        int buflen);

HANDLE PASCAL FAR WSAAsyncGetProtoByName(HWND hWnd, u_int wMsg,
                                        const char FAR * name, char FAR * buf,
                                         int buflen);

HANDLE PASCAL FAR WSAAsyncGetProtoByNumber(HWND hWnd, u_int wMsg,
                                           int number, char FAR * buf,
                                           int buflen);

HANDLE PASCAL FAR WSAAsyncGetHostByName(HWND hWnd, u_int wMsg,
                                        const char FAR * name, char FAR * buf,
                                        int buflen);

HANDLE PASCAL FAR WSAAsyncGetHostByAddr(HWND hWnd, u_int wMsg,
                                      const char FAR * addr, int len, int type,
                                        char FAR * buf, int buflen);

int PASCAL FAR WSACancelAsyncRequest(HANDLE hAsyncTaskHandle);
int PASCAL FAR WSAAsyncSelect(SOCKET s, HWND hWnd, u_int wMsg,
                               long lEvent);

#ifdef __cplusplus
}
#endif

/* Microsoft Windows Extended data types */
typedef struct sockaddr SOCKADDR;
typedef struct sockaddr *PSOCKADDR;
typedef struct sockaddr FAR *LPSOCKADDR;

typedef struct sockaddr_in SOCKADDR_IN;
typedef struct sockaddr_in *PSOCKADDR_IN;
typedef struct sockaddr_in FAR *LPSOCKADDR_IN;

typedef struct linger LINGER;
typedef struct linger *PLINGER;
typedef struct linger FAR *LPLINGER;

typedef struct in_addr IN_ADDR;
typedef struct in_addr *PIN_ADDR;
typedef struct in_addr FAR *LPIN_ADDR;

typedef struct fd_set FD_SET;
typedef struct fd_set *PFD_SET;
typedef struct fd_set FAR *LPFD_SET;

typedef struct hostent HOSTENT;
typedef struct hostent *PHOSTENT;
typedef struct hostent FAR *LPHOSTENT;

typedef struct servent SERVENT;
typedef struct servent *PSERVENT;
typedef struct servent FAR *LPSERVENT;

typedef struct protoent PROTOENT;
typedef struct protoent *PPROTOENT;
typedef struct protoent FAR *LPPROTOENT;

typedef struct timeval TIMEVAL;
typedef struct timeval *PTIMEVAL;
typedef struct timeval FAR *LPTIMEVAL;

/*
 * Windows message parameter composition and decomposition
 * macros.
 *
 * WSAMAKEASYNCREPLY is intended for use by the Windows Sockets implementation
 * when constructing the response to a WSAAsyncGetXByY() routine.
 */
#define WSAMAKEASYNCREPLY(buflen,error)     MAKELONG(buflen,error)
/*
 * WSAMAKESELECTREPLY is intended for use by the Windows Sockets implementation
 * when constructing the response to WSAAsyncSelect().
 */
#define WSAMAKESELECTREPLY(event,error)     MAKELONG(event,error)
/*
```

```
 * WSAGETASYNCBUFLEN is intended for use by the Windows Sockets application
 * to extract the buffer length from the lParam in the response
 * to a WSAGetXByY().
 */
#define WSAGETASYNCBUFLEN(lParam)           LOWORD(lParam)
/*
 * WSAGETASYNCERROR is intended for use by the Windows Sockets application
 * to extract the error code from the lParam in the response
 * to a WSAGetXByY().
 */
#define WSAGETASYNCERROR(lParam)            HIWORD(lParam)
/*
 * WSAGETSELECTEVENT is intended for use by the Windows Sockets application
 * to extract the event code from the lParam in the response
 * to a WSAAsyncSelect().
 */
#define WSAGETSELECTEVENT(lParam)           LOWORD(lParam)
/*
 * WSAGETSELECTERROR is intended for use by the Windows Sockets application
 * to extract the error code from the lParam in the response
 * to a WSAAsyncSelect().
 */
#define WSAGETSELECTERROR(lParam)           HIWORD(lParam)

#endif  /* _WINSOCKAPI_ */
```

LIST OF REFERENCES

1. Bharghavan, V., Demers, A., Shenker, S., and Zhang, L., "MACAW : A Media Access Protocol for Wireless LANs," *Proceedings SIGCOMM '94 Conf.,* ACM, pp. 212-225, 1994.

2. Davis, P.T. and McGuffin, G.R., *Wireless Local Area Networks,* New York, McGraw-Hill, 1995.

3. Geier, J., *Wireless Networking Handbook,* New Riders Publishing, Indianapolis, Indiana, 1996.

4. Hall, M., Towfiq, M., Arnold, G., Treadwell, D., and Sanders, H., "Windows Sockets. An Open Interface for Network Programming under Microsoft Windows," *http://www.stardust.com/wsresource/winsock1/winsock.html,* Winsock ver. 1.1 documentation, January, 1993, accessed site October 1996.

5. Holzmann, Gerard J., *Design and Validation of Computer Protocols,* Prentice Hall Publishing Co., Atlanta, Georgia, 1991.

6. Institute of Electrical and Electronics Engineers, Inc., IEEE P802.11, *Draft Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification,* New York, 1996.

7. Institute of Electrical and Electronics Engineers, Inc., "ISO/IEC Standard for Wireless LAN," LIAISON STATEMENT, *IEEE P802.11*, March, 1996.

8. Jabbari, B., Colombo, G., Nakajima, A., and Kulkarni, J., "Network Issues for Wireless Communications," *IEEE Commun. Magazine,* vol. 33, pp. 88-98, January, 1995.

9. Kanayama, Y., and Yun, X., "Research on a Semi-Autonomous Ground and Aerial Vehicle System for Mine/UXO Detection and Clearing," Proposal for Research, Naval Postgraduate School, Monterey, California, 1996.

10. Karn, P., "MACA - A New Channel Access Protocol for Packet Radio," *ARRL/CRRL Amateur Radio Ninth Computer Networking Conf.,* pp. 134-140, 1990.

11. Lundy, G.M. and Miller, R.E., "Specification and Analysis of a Data Transfer Protocol Using Systems of Communicating Machines," *Distributed Computing,* Springer-Verlag, December 1991.

12. Lundy, G.M. and Miller, R.E., "Specification and Analysis of a General Data Transfer Protocol," Tech Rep GIT-88/12, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, Georgia, 1988.

13. Lundy, G.M., "Tutorial on Communicating Finite State Machines," Class Notes, Department of Computer Science, Naval Postgraduate School, Monterey, California, February 1991.

14. Omnibyte, "Taurus 68040/68060 based Dual-Bus VMEbus Single Board Computer," *Users Manual,* Omnibyte, March, 1995.

15. OTC Telecom, "AirEzy900™ Plug and Play Wireless Ethernet LAN," User's Manual, by OTC Telecom, Inc., San Jose, California, 1996.

16. Quinn, B., Shute, D., *Windows™ Sockets Network Programming,* Reading, Massachusetts, Addison-Wesley, November, 1995.

17. Stallings , W., *Data and Computer Communications,* 4th ed., New York, Macmillan, 1994.

18. Stevens, W.R., *TCP/IP Illustrated,* Vol. 1, Reading, Massachusetts, Addison-Wesley, 1994.

19. Stremler, F., G., *Introduction to Communication Systems,* 3d edition, Addison-Wesley, October, 1992.

20. Tanenbaum, A.S, *Computer Networks,* 3d ed., Upper Saddle River, New Jersey, Prentice Hall, 1996.

21. Viterbi, A.J., *CDMA Principles of Spread Spectrum Communication,* Reading, Massachusetts, Addison-Wesley, 1995.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   875 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Chairman, Code EC 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5101

4. Dr. G. M. Lundy, Code CS/Ln 3
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943-5101

5. Dr. Xiaoping Yun, Code EC/Yx 3
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5101

6. Dr. Yutaka Kanayama, Code CS/Ka 1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California 93943-5101

7. Dr. Geoff Xie, Code CS 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943-5101

8. Lt. Alexander J. Bekas 2
   Laskaridou 160, Kallithea 176 75
   Athens, GREECE