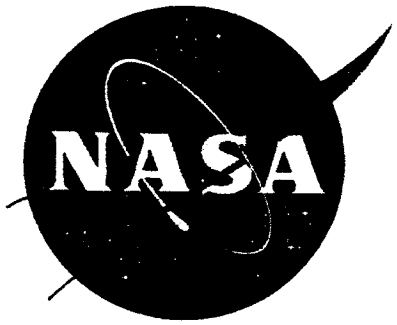
UNCLASSIFIED



Effective Utilization of Commercial Wireless Networking Technology in Planetary Environments: Year 2 Annual Report

Phillip De Leon, Stephen Horan, Deva Borah, and Ray Lyman
New Mexico State University
Klipsch School of Electrical & Computer Engineering
Box 30001, Dept. 3-0
Las Cruces, New Mexico 88003-8001

Annual Report Prepared for the
National Aeronautics and Space Adminstration, Glenn Research Center
under Grant #NAG3-2864

March 2005

UNCLASSIFIED

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01-03-2005	Technical Report	01-05-2004 -- 01-05-2005

4. TITLE AND SUBTITLE

Effective Utilization of Commercial Wireless Networking Technology in Planetary Environments: Year 2 Annual Report

5a. CONTRACT NUMBER

5b. GRANT NUMBER
NAG3-2864

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

DeLeon, Phillip, Horan, Stephen, Borah, Deva, and Lyman, Ray

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

New Mexico State University
Klipsch School of Electrical and Computer Engineering
Box 30001 / Dept. 3-0
Las Cruces, NM 88003-8001

8. PERFORMING ORGANIZATION REPORT NUMBER
NMSU-ECE-05-001

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

NASA Glenn Research Center
Space Systems and Grants Branch, MS 500-319
21000 Brookpark Road
Cleveland, OH 44135-3191
Technical Officer: Michael Caulev

10. SPONSORING/MONITOR'S ACRONYM(S)
NASA/GRC

11. SPONSORING/MONITORING REPORT NUMBER

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited; Distribution: Standard
Availability: NASA CASI (301) 621-0390

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The purpose of this research is to investigate the use of commercial, off-the-shelf wireless networking technology in planetary exploration applications involving rovers and sensor webs. The three objectives of this research project are to: 1) simulate the radio frequency environment of proposed landing sites on Mars using actual topographic data, 2) analyze the performance of current wireless networking standards in the simulated radio frequency environment, and 3) propose modifications to the standards for more efficient utilization. In this annual report, we present our results for the second year of research. During this year, the effort has focussed on the second objective of analyzing the performance of the IEEE 802.11a and IEEE 802.11b wireless networking standards in the simulated radio frequency environment of Mars. The approach builds upon our previous results which deterministically modelled the RF environment at selected sites on Mars using high-resolution topographical data. These results provide critical information regarding antenna coverage patterns, maximum link distances, effects of surface clutter, and multipath effects. Using these previous results, the physical layer of these wireless networking standards has now been simulated and analyzed in the Martian environment. We are looking to extending these results to the and medium access layer next. Our results give us critical information regarding the performance (data rates, packet error rates, link distances, etc.) of IEEE 802.11a/b wireless networks. This information enables a critical examination of how these wireless networks may be utilized in future Mars missions and how they may be possibly modified for more optimal usage.

15. SUBJECT TERMS

Mars communication networks, wireless proximity networks, RF propagation modeling

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19b. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Phillip DeLeon
U	U	U	UU	169	19b. TELEPHONE NUMBER *(Include area code)* (505) 646-3771

Abstract

The purpose of this research is to investigate the use of commercial, off-the-shelf wireless networking technology in planetary exploration applications involving rovers and sensor webs. The three objectives of this research project are to: 1) simulate the radio frequency environment of proposed landing sites on Mars using actual topographic data, 2) analyze the performance of current wireless networking standards in the simulated radio frequency environment, and 3) propose modifications to the standards for more efficient utilization. In this annual report, we present our results for the second year of research. During this year, the effort has focussed on the second objective of analyzing the performance of the IEEE 802.11a and IEEE 802.11b wireless networking standards in the simulated radio frequency environment of Mars. The approach builds upon our previous results which deterministically modelled the RF environment at selected sites on Mars using high-resolution topographical data. These results provide critical information regarding antenna coverage patterns, maximum link distances, effects of surface clutter, and multipath effects. Using these previous results, the physical layer of these wireless networking standards has now been simulated and analyzed in the Martian environment. We are looking to extending these results to the and medium access layer next. Our results give us critical information regarding the performance (data rates, packet error rates, link distances, etc.) of IEEE 802.11a/b wireless networks. This information enables a critical examination of how these wireless networks may be utilized in future Mars missions and how they may be possibly modified for more optimal usage.

Contents

List of Figures

List of Tables

1 Summary

Effective utilization of commercial wireless networking technology such as that based on the IEEE 802.11a and IEEE 802.11b standards, could bring the vision of proximity wireless networks used in future exploration of Mars to reality. However, in its current form, this technology may fall short of expectations due to its power requirements, limited transmission range, significant attenuation in hilly terrain, and possible poor bit error rate performance due to long multipath delays and harsh environments. The three objectives of this three-year research program are to 1) simulate the Radio Frequency (RF) environment of a planetary surface using high-resolution, Digital Elevation Maps of Mars, 2) analyze the performance of current wireless networking standards in the simulated RF environment, and 3) propose modifications to the wireless networking standards for better utilization in the planetary environment.

In year two, we have successfully accomplished the second research objective through the completion of the following tasks:

- Using the Year 1 results regarding antenna coverage patterns, maximum link distances, effects of surface clutter, and multipath effects, we have simulated the physical layers of IEEE 802.11a and IEEE 802.11b wireless networking standards in the Martian environment. The simulations were conducted in MATLAB using the Comm Access Technologies mWLAN toolbox. These results will be used as the basis to begin the study of the behavior of the medium access layer for these protocols for the planetary environment.

- We have measured key parameters (RF signal strength, delay spread, data rates, and packet error rates) in an outdoor IEEE 802.11b wireless network. These measurements were conducted at sites near Tortugas Mountain and Dripping Springs due their similarity to the Mars surface (free of man-made objects, little vegetation, mostly flat with some terrain variation and rocks, etc.). These measurements were compared with expected performance based upon the DEMs for the sites. We judge the agreement to be very good, especially considering the wide variation in measurements that can occur by moving the transmitter or receiver antenna by a small distance.

In this annual report, we present detailed results for the second objective of analyzing the performance of current wireless networking standards in the simulated RF environment. Our results give us critical information regarding the performance (data rates, packet error rates, link distances, etc.) of IEEE 802.11a/b wireless networks. This information enables careful examination of how these wireless networks may be utilized in future Mars missions and how they may be possibly modified for optimal usage. Our work suggests that:

- The results of the physical layer simulations for the Martian surface show that successful communications are possible within a few hundred meters of the transmit antenna when the transmit power is more than a few milliwatts and the antenna heights are fixed at more than 1 meter above the ground. The packet error rate performance of

802.11b without a RAKE receiver seems to be more adversely affected by the multipath conditions than 802.11a. Further, the lowest data rate mode of 802.11a provides the best bit error performance.

- Using higher power in the communications system does not always help the performance of the system. This result is known for terrestrial environments where vegetation and atmospheric phenomena are important. It is also true in sparse environments with negligible atmospheric attenuation as well.
- Transmission power and antenna height can be traded to a certain extent. The desired link Quality of Service and data rate may be more of a driver in link design and performance than transmission power and antenna height.
- Use of a RAKE-type of receiver can significantly improve performance with 802.11b protocols.

In addition, we include results from Year 1 regarding the validation of the power delay profiles. These results were not published in the 2004 annual report.

2 Introduction

NASA's long-term goals for the exploration of Mars include the use of rovers and sensors which intercommunicate through proximity wireless networks. Elements of the network have a short transmission range, low power requirements, low cost, and a relatively short-life span [1]. The performance of any such wireless network depends *fundamentally* on the Radio Frequency (RF) environment. In order to evaluate and optimize the performance of such a wireless network, a basic understanding or model of the channel is important. With such a model, better choices for the modulation and coding schemes, equalizer design, and positioning of network access point antennas can be made [2]. These choices can also affect the overall design and operations of a rover or a sensor web for a planetary environment.

In the first year of the research, we simulated the RF environment of selected sites on Mars using commercial RF planing and propagation software (with appropriate modifications for the Mars environment) and high-resolution Digital Elevation Maps (DEMs) from the Mars Global Surveyor (MGS) mission. The results of the simulation provided critical information regarding antenna coverage patterns, maximum link distances, effects of surface clutter, and multipath effects. Using these results, we now have simulated and analyzed the physical layers of the IEEE 802.11a and IEEE 802.11b wireless networking standards in the Martian environment. Our results give us new information regarding the performance (data rates, packet error rates, link distances, etc.) of the wireless networks. This information enables careful examination of how these wireless networks may be utilized for proximity networks in future Mars missions and how they may be possibly modified for optimal usage. In addition, these results could assist mission planners in network design should such wireless networking technology be adopted. These studies form the basis for our examination of the medium access layers for these protocols that we have now begun and how the protocols may be expected to act in the outdoor planetary environment.

The major accomplishments of the first year's work are detailed in [3] and they include:

- Identified a RF planning and propagation software packages (ATDI's HerTZ Mapper and ICS Telecom) that are used in the cellular telephone industry for network planning and optimization. The software, with modifications developed at NMSU, allows us to import DEMs of Mars and tune key parameters of the propagation model for accurate simulations of RF coverage patterns and multipath.

- Obtained high-resolution (10.4m/pixel) DEMs for the Gusev Crater and Meridiani Planum regions and successfully reformatted this data for import into the software. Within these regions, sites were chosen for their range and topographic features which were expected to yield different RF propagation effects in simulation.

- Computed RF coverage patterns and power delay profiles (multipath) for various communication scenarios at the selected sites. These scenarios include various transmitter and receiver locations, transmission power, surface clutter, and atmospheric phenomena. The basic reference configuration was a 2.4GHz carrier frequency, omnidirectional

antennas at a 1m height above local ground, and 1W radiated power. The clutter layer was synthesized based on the statistical distribution of rock sizes and spacing found in earlier Mars missions.

- Validated the simulation method by modelling a local site and performing actual RF field measurements.
- Utilized the simulation results to answer questions that mission planners might wish to ask such as percentage RF coverage in a region or percent RF coverage as a function of distance from a network access point.

The *major recommendations* from the first year's work are

- *Investigate how the coverage results and power delay profiles may affect communications protocol selection and/or modifications.* These investigations are expected to give further insights into the design of proximity network configurations and potential ways to modify the protocols for better performance.
- *Utilizing the RF models developed in this research, evaluate and improve IEEE 802.11a and 802.11b protocols for use in proximity wireless networks.*
- *Investigate how NASA mission planners might partner with New Mexico State University to make the RF modelling techniques developed in this research more widely known and utilized in the planetary exploration community.* The methods developed in this research could be used to model RF propagation at other sites of interest on the moon, Mars, and even in structures such as the International Space Station.

In this research regarding wireless network simulation on the Martian surface, we assume a proximity network model composed of multiple surface rovers and sensors that communicate via IEEE 802.11a or IEEE 802.1b to a network access point placed on the lander. At the lander, data is filtered, aggregated, and packaged for transmission back to earth via an orbiting relay station [4]. Other assumptions in the network model are 1W transmitted power between the network access point and surface nodes (rovers or sensors), omnidirectional antennas placed at a 1-2m heights, and 2.4GHz or 5GHz, as appropriate for the standard, carrier frequency.

We assume that the standard commercial 802.11 wireless networks are being used. Relevant parameters for the 802.11a and b standards are given in [5] and are summarized in Table 2.1. The 802.11 protocols contain functions related to connection management, link reliability management, and power management. The protocol developers are also looking into functions related to link security as well. The protocol supports two basic modes for mobile terminal operation: infrastructure mode and ad hoc mode. In the infrastructure mode used in this research, the mobile terminals access a wired network via one or more access points. In the ad hoc mode, the mobile terminals can communicate directly with each other without the need for the access point.

Table 2.1: Important 802.11 protocol parameters.

Parameter	*802.11b*	*802.11a*
Operating Frequency	2.4 GHz	5 GHz
Maximum Data Rate	11 Mbps	54 Mbps
Slot Time	20 μs	9 μs
Minimum Contention Window	31	15
Maximum Contention Window	1023	1023
Modulation Format	Complimentary Code Keying (CCK) Direct Sequence Binary Phase Shift Keying (DS-BPSK) Quadrature Phase Shift Keying (QPSK) Packet Binary Convolutional Code (PBCC)	Orthogonal Frequency Division Multiplexing (OFDM)

The wireless protocol needs to be configured so that the medium access layer works to avoid collisions and not just recover from channel collisions. The basic method is to use a modified version of the IEEE truncated binary exponential backoff algorithm. If the channel is busy, the sender backs off an integer multiple of the slot time. The backoff algorithm has variations from the strict application to allow a more fair access of the channel by those stations currently experiencing a backoff wait. In order to avoid the hidden terminal problem, the 802.11 networks can use a request-to-send/clear-to-send mechanism for channel access control and collision avoidance. Because the wireless channel can have relatively poor performance, e.g. a channel BER of 10^{-4}, there is a provision in the 802.11 protocols to allow for fragmentation of the packets. A shorter packet will have a higher probability of successful transmission so packet fragmentation can result in higher throughput in poor channels. Wireless protocols allow for power management through placing devices into sleep modes. When a device is in sleep mode, it cannot send or receive data. The device can have its sleep/awake duty cycle adjusted for specified power management goals. When a node needs to send data to a sleeping mode, the data sender must buffer the data until it detects the sleeping node has reactivated and is ready to receive.

3 Methods, Assumptions, and Procedures

3.1 Introduction

In this section, we first describe the methods, assumptions, and procedures used in validating the power delay profile results using local sites (this work was not included in the 2004 annual report but it did continue into Year 2 of the effort). Next, we describe the main work of Year 2 of the research project, namely simulating and analyzing the physical and medium access layers of IEEE 802.11a/b.

3.2 Validation of Local Power Delay Profile Simulations

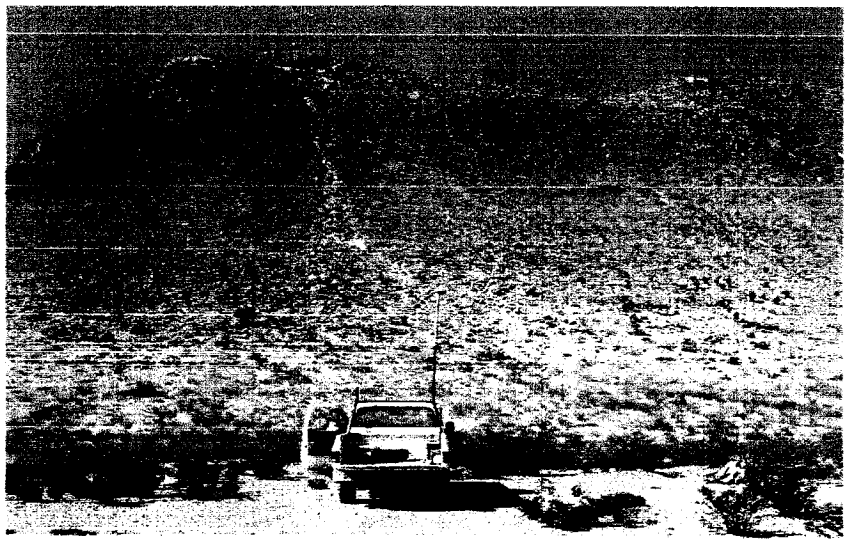
In order to gain confidence in the simulation of a Power Delay Profile (PDP) for any environment, we attempted to measure a detailed channel impulse response at 2.4 GHz at a location where the map data was available and where detailed measurements could be made. However, due to the high-speed instrumentation required to capture a complete, high-resolution response, we instead decided to measure the RF power for a simple reflection off a large-scale topographical feature referenced to a direct path power level. This reflection could then be checked against simulation for proper time delay and power level as a validation. In order that the channel response would not be too complex, we selected local sites which were very flat, had minimal vegetation, and which had a single large sloping or vertical rock face.

3.2.1 Local Site Locations for Reflection Measurements

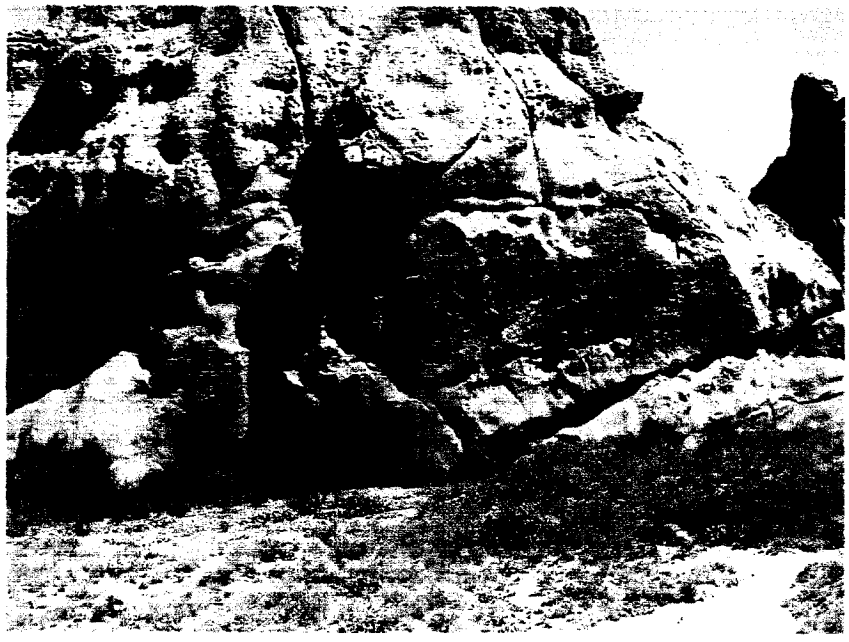
We chose two local sites for reflection measurements: 1) on the east side of Tortugas Mountain (approximately 10 minutes from the NMSU campus) and 2) Dripping Springs State Park (approximately 30 minutes from the NMSU campus). The "reflector" at the Tortugas Mountain site is a portion of the 200 m sloping, east face of the mountain (see Fig. 3.1 (a))and at the Dripping Springs site is a near vertical rock formation north of the visitor center (see Fig. 3.1 (b)). DEMs for these local sites are shown in Fig. 3.2. For these two sites, the coordinates for the transmitter (TX) and receiver (RX) positions are given in Table 3.1 and the basic arrangement of transmitter, receiver, and reflector is shown in Fig. 3.3. For the Tortugas Mountain site, the distance travelled along the reflection path is 424 m which translates to a travel time of 1.4 μs at the speed of light.

3.2.2 Measurement Procedure

In order to measure the reflected RF power off of Tortugas Mountain, we used directional antennas (Cushcraft 24012P) to transmit and receive the 2.4-GHz signal. Directional antennas were required in order to ensure that the received signal was strictly from the reflection off of Tortugas Mountain and not directly from a side lobe of the transmitting antenna. The actual antenna patterns were measured in the NMSU anechoic chamber and this pattern was later used in the simulation.

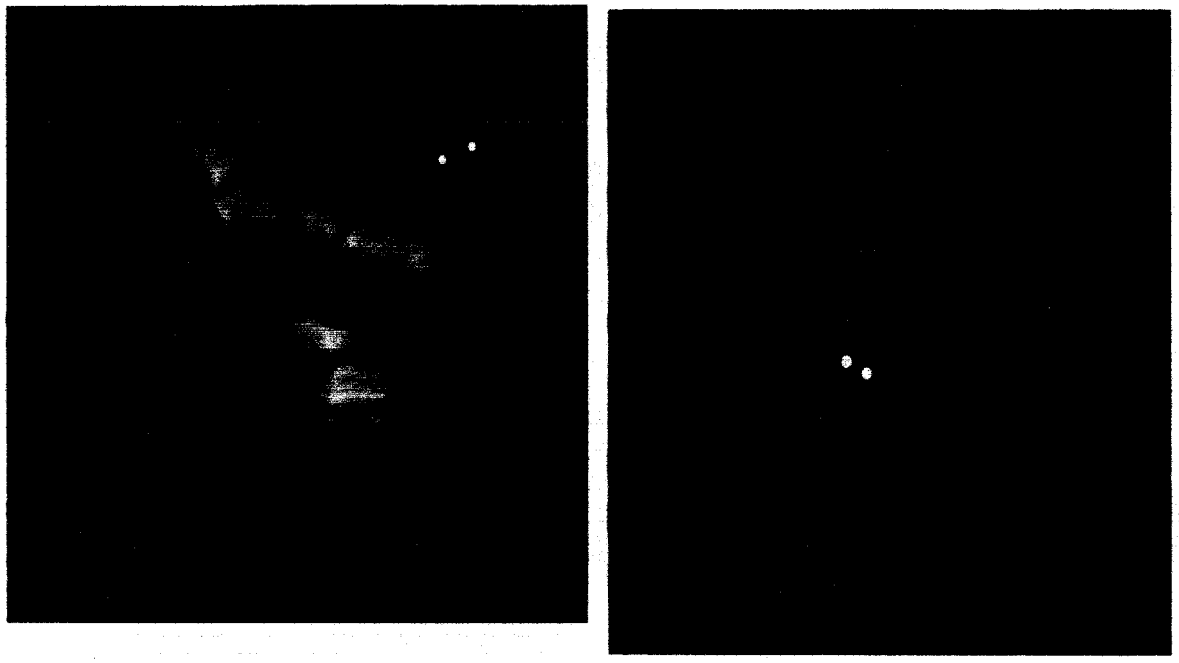


(a)



(b)

Figure 3.1: Reflectors at (a) Tortugas Mountain (sloping face) and (b) Dripping Springs (rock formation).



(a) (b)

Figure 3.2: Digital elevation maps for (a) Tortugas Mountain and (b) Dripping Springs. Approximate locations of TX, RX, and reflection points are denoted on DEMs.

Table 3.1: GPS coordinates for TX and RX positions at local sites.

Site	*Latitude*	*Longitude*	*Elevation*
Tortugas Mountain – TX	32° 17′ 46.1″	−106° 41′ 31.5″	1300 m
Tortugas Mountain – RX	32° 17′ 48.1″	−106° 41′ 26.9″	1309 m
Dripping Springs – TX	32° 19′ 59.8″	−106° 35′ 37.6″	1680 m
Dripping Springs – RX	32° 19′ 59.2″	−106° 35′ 35.4″	1682 m

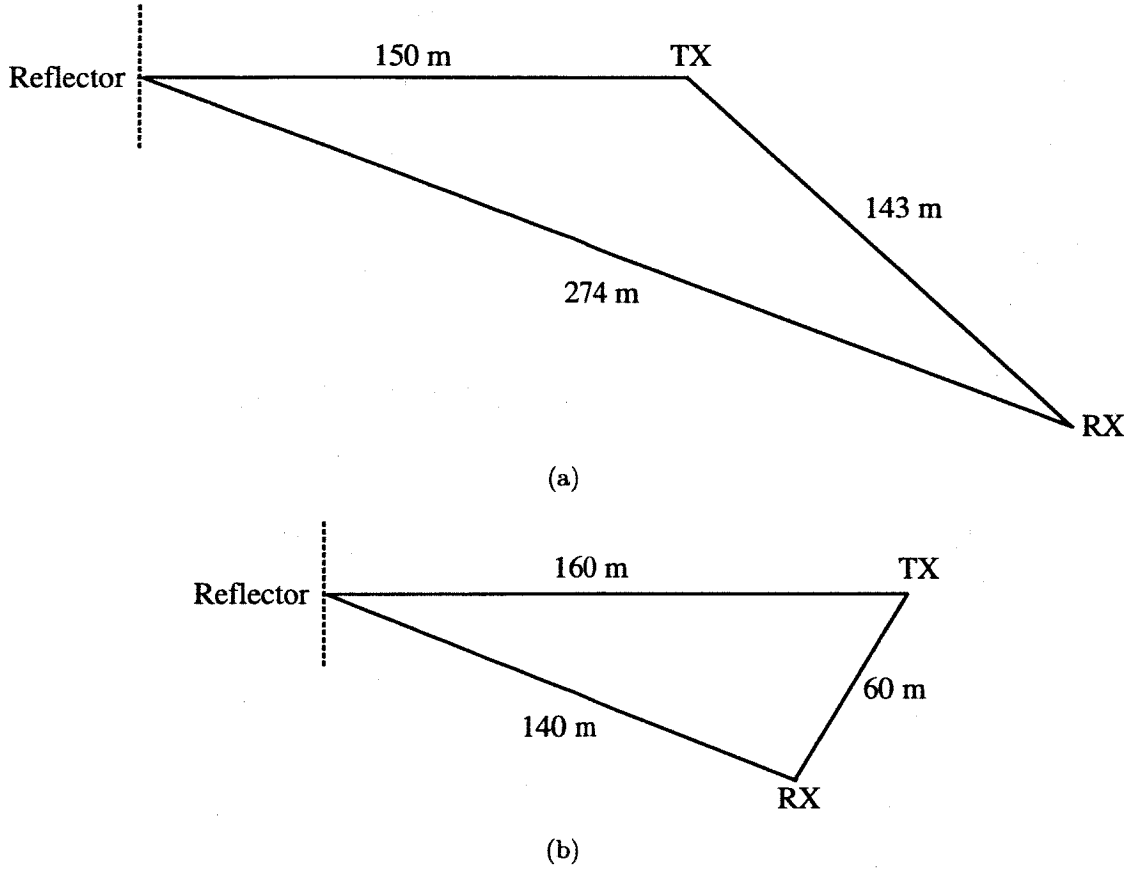


Figure 3.3: Arrangement of transmitter (TX), receiver (RX), and reflector at (a) Tortugas Mountain and (b) Dripping Springs.

For the measurements at Tortugas Mountain, the transmitter antenna was mounted on top of the NMSU telemetry van and the receiver antenna was mounted on a university truck as shown in Figure 3.4 in order to have them each at roughly the same elevations. For the measurements at Dripping Springs, both transmit and receive antenna were mounted on poles approximately 3 m off the ground. The receive antenna is shown in as shown in Figure 3.5. A Microdyne (TSS-2000) signal generator was used as the 2.4-GHz, 100-mW source and an Agilent spectrum analyzer (E4403B) was used to measure power levels at the receive side.

First, the noise floor at the site was found with the spectrum analyzer. Then the transmitter was turned on generate a continuous wave signal with a transmission power of 20 dBm (100 mW) at 2.4 GHz. Second, the direct path measurements were made by pointing the transmit and receive antenna towards each other. Third, the transmit and receive antennas were pointed towards a small target on Tortugas Mountain and RF power measurements were made. In order to determine the level of sidelobe power propagating from transmit to receive antenna, both antennas were pointed up into the sky and sidelobe power was measured.

3.2.3 Simulation of PDPs

The DEMs for the Tortugas Mountain and Dripping Springs sites were converted to ATDI's proprietary format for use with ICS Telecom. The procedure for generating the PDPs using ICS Telecom was given in Appendix C of the 2004 Annual Report.

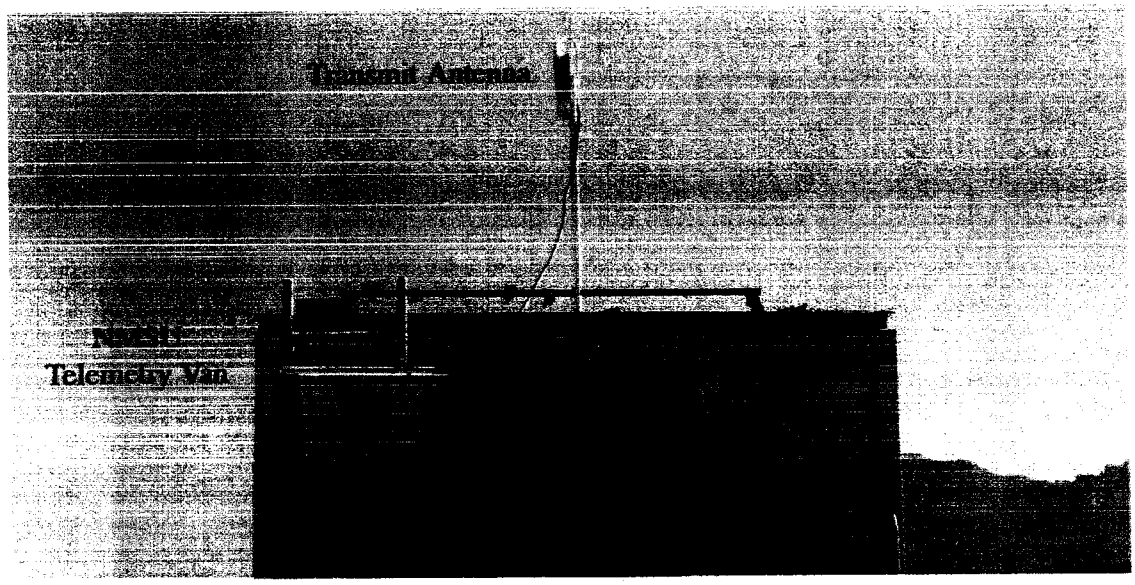
3.3 Measurement of Power Delay Profiles with the YellowJacket 802.11b PLUS

In June 2004, we acquired a YellowJacket 802.11b PLUS measurement instrument from Berkeley Varitronics Systems (BVS). In addition to measuring many IEEE 802.11b-specific parameters, it is also able to measure a power delay profile from an access point to the location of the YellowJacket. The PDP measurement can be geocoded using the on-board GPS unit and logged to a file. Such measurements provide us a capability to develop real-world, space-time RF channel models at 2.4GHz. In this section, we describe how that measurement is made.

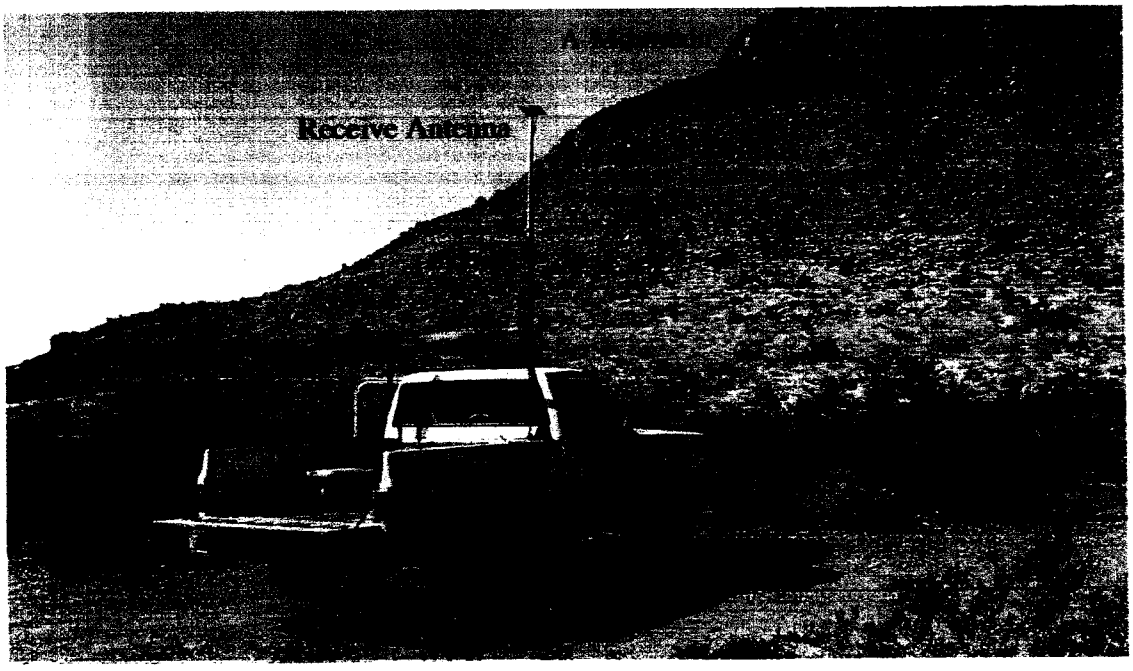
3.3.1 Background Information

Consider a channel with impulse response $h(n)$ with an input (transmitted) random process $u(n)$ and an output (received) random process $y(n)$. Compute the cross-correlation of the received process with the transmitted ($E[*]$ is the expectation operator)

$$\begin{aligned} r_{uy}(n) &= E\left[u(k)y^*(k-n)\right] \\ &= E\left[u(k)\left(\sum_{l=-\infty}^{\infty} h^*(l)u(k-n-l)\right)^*\right] \end{aligned}$$



(a)



(b)

Figure 3.4: Mobile antenna placement. (a) Transmit antenna on the top of the NMSU telemetry van and (b) Receive antenna on university truck at the Tortugas Mountain (A-Mountain) site.

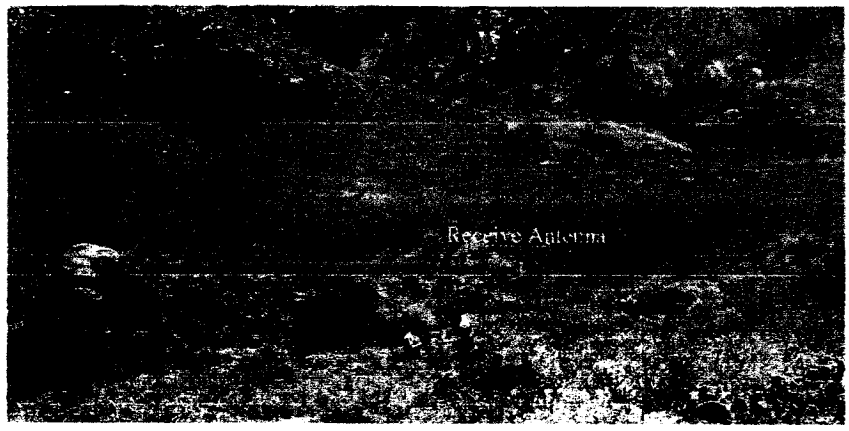


Figure 3.5: Receive antenna at Dripping Springs.

$$\begin{aligned} &= E\left[u(k)\sum_{l=-\infty}^{\infty} h(l)u^*(k-n-l)\right] \\ &= E\left[\sum_{l=-\infty}^{\infty} h(l)u(k)u^*(k-n-l)\right] \\ &= \sum_{l=-\infty}^{\infty} h(l)E\left[u(k)u^*(k-n-l)\right]. \end{aligned} \tag{3.1}$$

All 802.11b packet transmissions start with several repetitions of the unmodulated Barker code. The 802.11b Barker code is 11-bits long and it has excellent correlation properties, i.e.

$$r_u(n) = \delta(n). \tag{3.2}$$

The YellowJacket correlates the received 802.11b packet, $y(n)$ against the Barker code at the start of the received packet. Thus substituting (3.2) into (3.3) shows that the cross-correlations are in fact the impulse response values

$$\begin{aligned} r_{uy}(n) &= \sum_{l=-\infty}^{\infty} h(l)\delta(n-l) \\ &= h(n). \end{aligned} \tag{3.3}$$

The PDP is readily computed from (3.3) by

$$P'(n) = \max|h(n)|^2. \tag{3.4}$$

Finally, YellowJacket normalizes (3.4) so that

$$\begin{aligned} P(n) &= \max P'(n) \\ &= 1.0. \end{aligned} \tag{3.5}$$

3.3.2 Relative Correlation Window

Equation (3.4) is computed in the YellowJacket at a time spacing of 1/4 chip (91 ns), for a span of about 10 chips, i.e. and displayed in the relative correlation window. YellowJacket centers the PDP so that the peak (assumed to be the direct path) is at 0 (hence the relative correlation). Thus for convenience, the relative correlations start at 2 chips and end around +8 chips. This is illustrated in Figure 3.6 for the YellowJacket unit where the display is set to show signal correlation and received signal strength (RSSI).

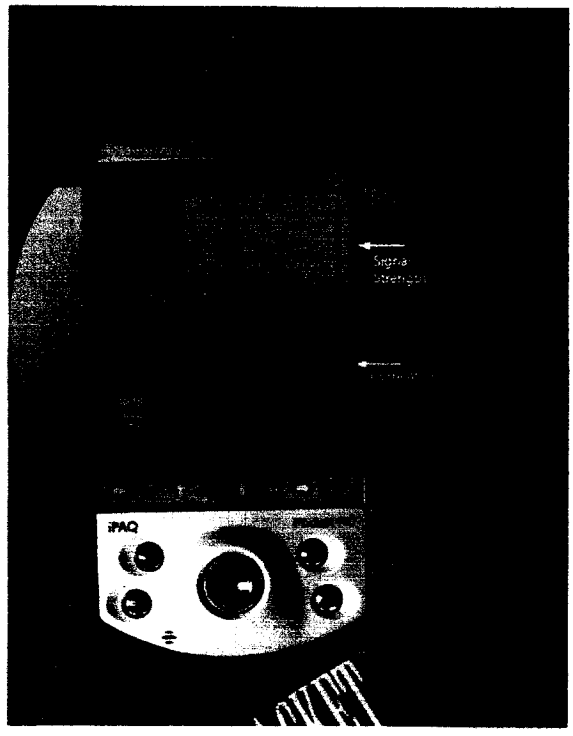


Figure 3.6: YellowJacket display showing the correlation display window plus a Received Signal Strength Indicator (RSSI) window.

3.3.3 Multipath Values in Log File

The first $N = 22$ relative correlations which are displayed are logged to a file. Presumably not all 40 values are stored in order to reduce memory. Thus the time extent of the correlation function is

$$\begin{aligned} \tau &= 22 \times 91\text{ns} \\ &= 2\mu s. \end{aligned} \tag{3.6}$$

3.3.4 RMS Delay Spread

The *rms delay spread*, σ is defined as

$$\sigma = \sqrt{E\left[P^2(n)\right] - \left(E\left[P(n)\right]\right)^2}. \tag{3.7}$$

where E is the expectation operator and $P(n)$ is the received power at time n. YellowJacket computes and displays the rms delay spread using the 22 PDP values as follows. Since we have only a finite amount of data the expectations must be estimated. The second moment, $E\left[P^2(n)\right]$ is estimated with

$$\hat{E}\left[P^2(n)\right] = \frac{\sum_{n=0}^{N-1} P(n)\, n^2}{\sum_{n=0}^{N-1} P(n)\, n} \tag{3.8}$$

where $N = 22$ is the number of PDP samples YellowJacket has stored. The first moment or *mean delay*, $E\left[P(n)\right]$ is estimated with

$$\hat{E}\left[P(n)\right] = \frac{\sum_{n=0}^{N-1} P(n)\, n}{\sum_{n=0}^{N-1} P(n)\, n}. \tag{3.9}$$

Using these estimates, the rms delay spread is calculated as

$$\sigma = T\sqrt{\hat{E}\left[P^2(n)\right] - \left(\hat{E}\left[P(n)\right]\right)^2}. \tag{3.10}$$

where $T = 1/22$ MHz (1/2 x 1/4 chip period?).

3.3.5 Final Notes

The YellowJacket provides a convenient method of measuring real-world PDPs provided the delay spread is not too large so that the values alias. Note that if the propagation delay between the strongest path and multipath reflections is larger that 9 chips, the multipath can "alias"to the start (-2) of the multipath display.

3.4 Simulation of IEEE 802.11a/b Physical Layer using MATLAB

The work in this section is based upon the paper "Performance Evaluation of the IEEE 802.11a and b WLAN Physical Layer on the Martian Surface" by Deva K. Borah, Anirudh Daga, Gaylon Lovelace and Phillip DeLeon. This paper was presented at 2005 IEEE Aerospace Conference.

3.4.1 Summary

The performance of IEEE 802.11a and b WLAN standards on the Martian surface is studied in this task area. The Gusev Crater region and the Meridiani Planum (Hematite) region

are chosen as example sites based on the mission science and mission success criteria. The radio frequency multipath environment is obtained using digital elevation maps from the Mars Global Surveyor mission, taking into account the atmosphere and other factors on the Martian surface. It is observed that IEEE 802.11a performs well in terms of packet error rates at distances up to a few hundred meters from the transmit antenna when the transmit power is 1 W and the antennas are located 1.5 m above the ground. Although the performance of IEEE 802.11b is found to be more adversely affected, its performance too can be improved significantly using a RAKE receiver. It is observed that the lower data rate modes of 802.11a show much better results in terms of bit error rates. However, both 802.11a and b appear to provide effective communications within a few hundred meters of the transmitter in the selected sites considered.

3.4.2 Introduction

Future space missions on the Martian surface may involve multiple rovers collecting data at different locations, and communicating wirelessly with common access points. Such communications have to be reliable, robust and power efficient. Development and testing of such communication technologies from scratch is an expensive proposition. A more cost effective approach would be to adapt existing technology with appropriate modifications. Towards this objective, this paper investigates the physical layer performance of two well known Wireless Local Area Network (WLAN) standards IEEE 802.11a and IEEE802.11b under the Martian environment, and identifies the issues that need to be addressed.

The IEEE 802.11b standard [6] provides data rate options of 1, 2, 5.5 and 11 Mbit/s in the 2.4 GHz band. The modulation options include direct sequence spread spectrum using differential Binary Phase Shift Keying (BPSK), Quadrature Phase Shift Keying (QPSK), Complementary Code Keying (CCK), and Packet Binary Convolutional Code (PBCC). Although primarily designed for indoor office environments, recent studies have shown good performance of 802.11b in outdoor environments [7], [8]. However, the performance with low height rover antennas on the Martian surface, and the performance comparison of 802.11b with respect to 802.11a in the Martian environment are important issues that have not been addressed before and need investigation.

The IEEE 802.11a standard [9] operates in the 5-GHz band and uses the Orthogonal Frequency Division Multiplexing (OFDM) technology. It can support data rates of 6, 9, 12, 18, 24, 36, 48, and 54 Mbit/s. The standard employs convolutional encoder, and uses cyclic prefix of 0.8 micro second duration. This enables it to handle the multipath problem more successfully [10], [11]. However, longer delay spreads, which can happen on the Martian environment with low height antennas and longer transmitter/receiver distances, can severely affect its performance. Therefore, the effects of such delay spreads on the Martian surface require investigation.

In previous work [12], the RF environment of the Martian surface has been extensively studied. In particular, the RF coverage patterns produced from a 2.4 GHz transmitter with 1 W radiated power and 1 m antenna height within Gusev Crater and Meridiani Planum have

been investigated. These simulations use 11 m/pixel digital elevation maps from the Mars Global Surveyor mission. The software used in this study takes into account the propagation factors such as planetary radius, atmospheric density and composition, soil chemistry, etc. The impact of surface clutter (rocks) on RF propagation has also been examined. It has been observed in that study that while significant terrain variation can have a major impact on the coverage, sufficient RF signal power for an IEEE 802.11b link is possible at these sites over several kilometer distances even with low antenna heights.

This analysis uses the received power results from [12] and recent results regarding the simulation of the multipath environment in the performance evaluation of the 802.11 a and b standards. Here, we will study the performance of different data rates for different transmit and receive antenna locations and several sites on Mars. It is observed in the results that multipath environments can severely affect the performance of 802.11b. The use of receivers that mitigate multipath effects, such as RAKE-type receivers, is found to provide significant improvement. The performance of 802.11a is also found to be affected by the multipath environment, especially in the absence of clear line-of-sight. In particular, the higher bit rate modes of IEEE 802.11a are found to be more affected by the multipath effects. Further, when the delay spread exceeds the 0.8 μs cyclic prefix duration, the performance drops rapidly.

This study section is organized as follows. In Section 3.4.3, we provide an overview of the 802.11a and b physical layer (PHY) specifications. The packet structures described in this section are faithfully simulated in our simulation results section. In Section 3.4.5, the site selection criteria on the Martian surface, and the radio frequency multipath calculation are described. In Section 3.4.4, the methodology for preparing MATLAB and the product mLAN for the physical layer simulations is described. Section 4.6 presents the simulation results in terms of packet error rates and bit error rates. A discussion on the interpretations of the simulation results is presented in Section 4.6.6.

3.4.3 IEEE 802.11 WLAN

In this section, we present a brief overview of the physical layer specifications of IEEE 802.11a and b standards.

IEEE 802.11

The IEEE 802.11a physical layer is based on orthogonal frequency division multiplexing and operates in the 5-GHz band providing data payload capabilities of 6, 9, 12, 18, 24, 36, 48 and 54 Mbit/s. The different transmission rates are obtained by varying the modulation type and/or the channel coding rates. The system uses 52 subcarriers that are modulated using BPSK, QPSK, 16- or 64- Quadrature Amplitude Modulation (QAM). The error correction coding uses a convolutional encoder with a coding rate of 1/2, 2/3 or 3/4.

The Physical layer Protocol Data Unit (PPDU) format is shown in Fig. 3.7. The Physical Layer Convergence Procedure (PLCP) preamble field consists of 10 repetitions of a short training sequence, and two repetitions of a long training sequence preceded by a Guard Interval (GI). A single BPSK encoded OFDM symbol follows. It contains a 4-bit RATE

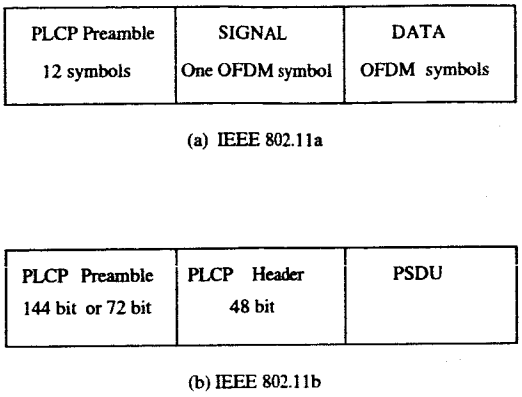


Figure 3.7: PPDU Format for IEEE 802.11a and b.

field, a 12 bit LENGTH field, one reserved bit, one parity bit and 6 'zero' tail bits encoded with a rate 1/2 convolutional code. The DATA portion contains a 16 bit SERVICE field, a Physical Sublayer service Data Unit (PSDU), 6 'zero' tail bits and pad bits, and may constitute multiple OFDM symbols.

The data to be transmitted are scrambled to remove any spectral line from the data. They are then convolutionally encoded with a rate-1/2 encoder with generator polynomials $g_0 = 133_8$, $g_1 = 171_8$, and puncturing is performed if necessary. All encoded data bits are interleaved using two steps. First, consecutive coded bits are mapped to non-adjacent subcarriers. The second step maps consecutive coded bits onto the less and more significant bits of the constellation.

The OFDM symbols are transmitted using a relatively long cyclic prefix of duration $T_{GI} = T_{FFT}/4$, where T_{FFT} is the duration of an OFDM symbol. The duration T_{FFT} equals 3.2 μs. Thus the symbol interval is 4.0 μs. The PLCP preamble duration is 16 μs, and the SIGNAL symbol lasts 4.0 μs.

IEEE 802.11b

The IEEE 802.11b Direct Sequence Spread Spectrum (DSSS) can provide data rates of 1, 2, 5.5 and 11 Mbit/s in the 2.4 GHz band. The basic data rate of 1 Mbit/s is provided using Differential Binary Phase Shift Keying (DBPSK) while the 2 Mbit/s rate uses Differential Quadrature Phase Shift Keying (DQPSK). The above two data rates employ 11 chip long Barker sequences for spreading with a chip rate of 11 MHz.

Higher data rates of 5.5 Mbit/s and 11 Mbit/s are available in 802.11b through the use of complementary code keying at the same chipping rate of 11 Mchips/s. Each CCK symbol consists of 8 complex chips: $e^{j(\phi_1+\phi_2+\phi_3+\phi_4)}$, $e^{j(\phi_1+\phi_3+\phi_4)}$, $e^{j(\phi_1+\phi_2+\phi_4)}$, $-e^{j(\phi_1+\phi_4)}$, $e^{j(\phi_1+\phi_2+\phi_3)}$, $e^{j(\phi_1+\phi_3)}$, $-e^{j(\phi_1+\phi_2)}$, $e^{j\phi_1}$. In the case of 5.5 Mbit/s, 4 bits are transmitted per symbol while in the case of 11 Mbit/s, the number of bits transmitted per symbol is 8. The first two bits are used to compute a phase change for ϕ_1 with respect to phase ϕ_1 of the preceding symbol or the phase of the preceding DQPSK symbol if there is a header to PSDU transition. In the case of 5.5 Mbit/s, the remaining two bits are used to derive the phase ϕ_2, ϕ_3 and ϕ_4, while

the 11 Mbit/s mode uses the remaining 6 bits to compute ϕ_2, ϕ_3 and ϕ_4 based on QPSK. An optional mode replacing CCK modulation with packet binary convolutional coding with a 64-state encoder is also available.

The PPDU format for IEEE 802.11b is also shown in Fig. 3.7. Two different preamble and headers are defined: long PLCP PPDU format and short PLCP PPDU format. The long format contains a 144-bit preamble and a 48-bit header, while the short format contains a 72-bit preamble and a 48-bit header. The preamble contains two fields: synchronization (Sync) and Start Frame Delimiter (SFD). The Sync field is provided to enable the receiver perform necessary synchronization operations. The SFD indicates the start of PHY-dependent parameters within the PLCP preamble. The header consists of signal, service, length and Cyclic Redundancy Code (CRC) fields. The signal field indicates the data rate that is used for the transmission and reception of the PSDU. The service field contains 8 bits, and they carry some information about modulation, symbol clock etc. The length field indicates the number of microseconds required to transmit the PSDU. Finally, a 16 bit CRC protects the signal, service and the length fields. The long PLCP preamble and header are both transmitted using 1 Mbit/s DBPSK modulation. In the case of a short PLCP, the preamble is transmitted using 1 Mbit/s while the header is transmitted using 2 Mbit/s. The transmitted data bits are scrambled at the transmitter and descrambled at the receiver.

3.4.4 MATLAB Simulation Development

We use the capabilities of MATLAB to model the physical layer in this investigation. MATLAB does not have the required tools in its manufacturer's distribution to perform all of the analysis. To augment MATLAB, we acquired the mLAN tools from CommAccess. This section describes the effort required to male these tools usable in the analysis and simulation.

IEEE 802.11a

The IEEE 802.11a physical layer simulation software was developed around the mWLAN Toolbox by CommAccess Technologies. This package provides a large set of MATLAB functions, most of which are sub-functions that need not be called directly. They provide a reasonably complete implementation of the 802.11a OFDM signaling and packet construction. In brief, a block of random data is generated and passed to mWLAN's `PLCP_11a()` function, which returns a complex baseband transmit vector, sampled at 20MHz, representing one packet. After channel effects (noise, fading, multipath, etc) are imposed, the vector is passed to `parse_signal_11a()`, which extracts and checks the transmission parameters encoded in the packet header SIGNAL symbol. An FFT translates back from the time to the frequency domain, and the demodulation and data decoding are performed by the mWLAN functions `DeMod_11a()` and `DeMod_post_11a()`.

A significant amount of "glue logic" code is required to piece together these mWLAN functions. To simplify the effort, ensure repeatability, and guard against mistakes, a top-level simulation script (`sim_ER_11a`) was developed. It reads experiment parameters (e.g. packet size, data rate, E_b/N_0, number of packets to simulate, etc) from a file, so that many different simulation experiments could be run without modifying the simulation source code

itself. Simulation results were logged into result files, and additional scripts were developed to process the results and produce figures and tables.

In addition to the mechanics of simulation (looping through variations and iterations, calling mWLAN functions, storing results, etc), our code implements whatever channel effects are specified in the parameter file. The mWLAN toolbox does not provide any channel modelling functionality. Our software reads in a power delay profile (calculated from Martian terrain maps by ATDI's ICS Telecom RF modelling software). Rayleigh fading is applied, and the result is convolved into the complex baseband transmit vector for the packet. The receiver end of the simulation estimates the channel by a linear least-squares method, and weights the output of the FFT of each received OFDM symbol.

To validate the behavior of these tools, the results of our simulations were compared to published results. Doufexi [11]presented BER-vs-E_b/N_0 results (their Figure 11) for a channel with AWGN plus a 50ns exponentially decaying multipath power delay profile. We ran our simulation with this channel model for the 6, 12, 36 and 54 Mb/s data rates, and obtained very good agreement with the published curves. CommAccess Technologies publish pure AWGN results in the User Manual for their eWWoB product (a GUI workbench based on mWLAN). Their results are consistently 1.25dB (i.e. a factor of 64/48) better than ours. After extended discussions with Dr Song An, who developed mWLAN, we found that the difference is exactly attributable to the treatment of noise power in the non-data OFDM sub-bands, in our respective definitions for E_b/N_0. The distinction is somewhat arbitrary, and in the end, we decided to follow the convention used in Doufexi's published results, which derives from what seems a more intuitive handling of noise power by the DFT of the OFDM receiver structure.

IEEE 802.11b

The physical layer simulations for IEEE 802.11b were performed using the mWLAN toolbox developed by CommAccess Technologies, Inc. The mWLAN toolbox provides several modules associated with the transmission and reception of data at the physical layer. The modules available in mWLAN include:

- Bit Generation module to generate random data bits,
- Square root raised cosine filtering module for pulse shaping (oversampling) the raw data for a given rolloff factor,
- CRC encoding module to encode certain fields in the physical layer header with the CCITT-CRC 16 polynomial,
- Scrambler and descrambler modules to scramble and descramble the data bits in accordance with IEEE 802.11b standard specification,
- Modulation and demodulation modules for the basic data rates of 1 Mbps(DBPSK) and 2 Mbps(DQPSK),

- Modulation and demodulation modules for the high data rates of 5.5 Mbps and 11 Mbps(CCK),
- Modulation and demodulation modules for the PBCC mode (all data rates), and
- CRC check module to verify the validity of the received data.

All these modules are available in the form of Windows Dynamic Linked Libraries(DLLs) and can be accessed using their associated MATLAB functions. Bit error rate and Packet error rate simulations for all the data rates can be performed by writing MATLAB programs which use these functions in a logical order.

The original module for 1 and 2 Mbps modulation had to be changed because it carried out DBPSK and DQPSK encoding after spreading the data with the Barker sequence. The correct way to go was to Barker spread the DBPSK and DQPSK encoded data to preserve the autocorrelation properties of the Barker sequence. Corresponding changes were made in the demodulator module as well. Also, the original module provided for only sub-optimal demodulation of the CCK based 5.5 and 11 Mbps modes. New modules were developed to perform near-optimal demodulation of the 5.5 and 11 Mbps modes.

After making these changes and additions, BER simulations were carried out in AWGN channel to verify the correctness of the modules. The results for the 1 and 2 Mbps schemes agreed very closely with the results published in [13]. The results for the 5.5 Mbps and 11 Mbps schemes agreed very closely with the results published in [14].

Several new modules were developed to incorporate the Martian multipath power delay profiles(PDPs) generated by ICS Telecom. The PDPs obtained from ICS Telecom had a 10ns resolution and had to be fractionally resampled to match the sample time of the oversampled (square root raised cosine filtered) data. The power delay profiles were converted to normalized channel impulse responses to observe the effect of multipath on BER vs Eb/No for various data rates. Modifications were also made to observe the BER and PER as a function of distance. Least squares channel estimation was carried out to estimate the multipath channel impulse response at the receiver. RAKE receiver was implemented, which significantly improved performance.

3.4.5 RF Environment on the Martian Surface

We have used the ICS Telecom software from ATDI [15] to obtain the multipath environment on the Martian surface. DEM files are converted to ATDI's format for the Martian sites (11m/pixel resolution), and are loaded into the software.

Site Selection

We have selected the Gusev Crater and the Meridiani Planum (Hematite) regions [16] as example sites for our study. These two regions are chosen considering the mission science and mission success criteria [16], [17]. The mission science criteria included evidence of water on the Martian surface in the past. The Gusev Crater appears to have been a lake fed by a river at one time. The Meridiani Planum region shows the chemical signature of Hematite

minerals associated with ancient water locations. For mission success, the sites are chosen "near the equator, low in elevation, not too steep, not too rocky, and not too dusty" in addition to other factors. The locations of the selected sites are shown in Table 3.2.

Table 3.2: Sites for WLAN Performance Study.

Site	*Mars Latitude*	*Mars Longitude*
Gusev1 - Site 1	14° 47′ 39.35″ S	176° 1′ 29.18″ E
Gusev1 - Site 2	14° 58′ 41.95″ S	176° 2′ 53.51″ E
Gusev1 - Site 3	15° 11′ 35.66″ S	176° 4′ 31.23″ E
Hematite4 - Site 1	2° 11′ 0.69″ S	−5° 53′ 5.16″ E
Hematite5 - Site 1	1° 52′ 29.16″ S	−5° 25′ 39.59″ E

RF Model

The Irregular Terrain Model (ITM) has been used. It is a general-purpose propagation model for frequencies between 20MHz and 20GHz. This model predicts the median attenuation of a radio signal as a function of distance and the variability of the signal in time and space. The predictions are based on electromagnetic theory and statistical analysis of both terrain features and radio measurements.

The ITM source code has been modified for Martian parameters. Atmospheric attenuation is negligible—actual calculations for a horizontal path on Mars' surface yield attenuation of approximately 10^{-6} dB/Km at 2.5GHz [18]. The ITM source code for propagation on Earth accounts for atmospheric refraction by introducing an "effective radius" multiplier of $K = 1.33$. The effective radius used for Earth is K times Earth's physical radius. Mars' atmosphere is so diffuse, even at the planet's surface, as to resemble a vacuum compared to Earth's. Thus we assume atmospheric refraction is negligible in our study [18], [19]. We set $K = 1$, and use an effective radius equal to Mars' physical radius. We note that in some implementations, an effective curvature (inverse of the effective radius) is used.

3.5 Simulation of IEEE 802.11a/b MAC Layer using OPNET

During the summer of 2004, we considered alternatives for evaluating the MAC-layer performance of wireless LAN protocols on the Martian surface. Ideally, such an evaluation would be tightly coupled with the physical-layer model developed during the initial phase of the research. Simulation software tools which we considered included OPNET, QualNet, NS2 and various MATLAB toolboxes and packages. We chose OPNET because of its ease of use and wide user base. We obtained three licenses and began training during the first summer session.

We discovered that OPNET has limited capabilities in incorporating a physical-layer model in a MAC-layer simulation, so we investigated alternatives for extending its capability in this area. Two basic approaches were considered:

1. Find a reasonable, parameterized physical-layer model, based on previous studies, and incorporate this into the OPNET simulator.

2. Program the physical-layer model using MATLAB, and then provide an interface between this tool and OPNET in order to perform a cosimulation.

The first approach would be preferable if an appropriate physical-layer model could be characterized using a sufficiently small number of parameters. The standard software modules used by OPNET for simulating a wireless channel employ a very simple model, where the probability of packet error is related straightforwardly to the signal-to-noise ratio [20, Ch. 11]. These modules would have to be modified, so that significant effects such as multipath are included in the packet-error computation along with signal strength and noise level. The modification would be accomplished by providing user-defined process modules, written in C, according to a standard procedure supported by OPNET [21]. This approach would be simpler than a full-up cosimulation, and the simulations would probably run much faster. The reason for this is that OPNET views the transmission or reception of a packet as a discrete event. In a cosimulation, each of these events involving a wireless link would give rise to a physical-layer simulation requiring a much finer time granularity, thus multiplying the simulation time.

If a parameterized model proves to be insufficient, then some form of cosimulation will have to be used. To provide an appropriate interface, two options were considered:

1. Use OPNET External-Domain cosimulation tools to exchange data between OPNET and MATLAB [22].

2. Use MATLAB tools for calling the MATLAB engine from the C code used to program the OPNET process modules [23].

The OPNET External-Domain tools proved to be complicated, difficult to use, and not well documented. Further, we were not able to find any literature referring to an OPNET–MATLAB cosimulation that was carried out this way. By contrast, procedures for calling the MATLAB engine from a C-language program are very well described in literature provided by Mathworks, and this approach for linking MATLAB and OPNET has been successfully used by other investigators studying the interaction between MAC and physical layers (see, e.g. Dham [24]).

The approach makes use of the MATLAB *Engine Library* which contains a set of routines that can call MATLAB from a C program. We can write C programs called MATLAB *Engine Programs* that can open and close MATLAB, send data to and from MATLAB, and send commands to be processed in MATLAB. Thus, we may use the *Engine Library* to pass information between OPNET and MATLAB via the C-language user-provided process modules discussed above.

Here is a detailed procedure for preparing an OPNET–MATLAB cosimulation using this approach on Windows XP:

1. Ensure that Microsoft Visual C++ is installed on the computer
2. Open the OPNET Modeler software. Click on Edit→Preferences
3. The preference **bind_shobj_prog** should have the value **bind_so_msvc**
4. The preference **bind_static_prog** should have the value **bind_msvc**
5. Specify the path where the MATLAB Engine Library files are located. This can be done by editing the **bind_shobj_flags** preference. Change the value to **/LIBPATH::C:\MATLAB6p5\extern\lib\win32\microsoft\msvc60**
6. To use the MATLAB engine the following library files have to specified.
   - libeng.lib
   - libmex.lib
   - libmx.lib
   - libmat.lib

   This can be done by changing the value of the **bind_shobj_libs** preference to **libeng.lib libmex.lib libmx.lib libmat.lib**
7. Close the preferences dialog box
8. Open the *Process Model* by clicking on File→New→Process Model
9. Define the state variables by clicking on the **SV** icon and the temporary variables by clicking on the **TV** icon.
10. Click on the **HB** icon and specify the relevant header files.
11. Click on the **Create State** icon (the icon at the top left corner) and place a state in the workspace
12. Enter the C program that will call MATLAB by clicking either on the top half of the state (Enter execs) or the the bottom half of the state(Exit execs)
13. Compile the program to check for compilation errors
14. Click on Intefaces→Process Interfaces. Change the initial value of **begsim intrpt** attribute to **enabled**
15. Associate this process with a node and the node with a network.

3.6 Outdoor IEEE 802.11b Measurements

In order to validate the IEEE 802.11b physical and MAC-layer simulations, actual measurements of IEEE 802.11b must be made in the outdoor environment. These measurements include data rates, bit error rates (BERs), packet error rates (PERs), power delay profiles, signal strength, etc. For the validation work, we considered three environments: free-field, simple reflection, complex terrain. For the free field environment, we conducted experiments at the football stadium parking lot on the NMSU campus. The lot has a gravel surface, is very flat and has virtually no vegetation or man-made objects. The nearest man-made objects are at least 200 m away from our staging area. The simple reflection environment consists of a large (9′ x 6′) steel reflector located on the football stadium parking lot. Finally, we have chosen the Dripping Springs site (La Cueva rock formations) for measuring the IEEE 802.11b protocol performance in a complex terrain. Figure 3.8 shows the measurement equipment on the parking lot and Figure 3.9 shows the reflector. Figure 3.10 shows the topographic map of the setup at La Cueva (rock formations) at Dripping Springs. The complete field test procedure is given in Appendix E.

The measurement setup for the free-field experiments includes a basic IEEE 802.11b wireless access point/router elevated to either 1 m or 2 m height with a laptop hardwired to it. In addition both a laptop (with an IEEE 802.11b wireless card) and the YellowJacket are located at various distances along a radial path from the access point. The entire setup is portable and battery powered. A dataflow from either Transmission Control Protocol/Internet Protocol (TCP/IP) or User Datagram Protocol (UDP) based source is initiated between the laptops using the network bandwidth measurement software application Iperf that is described below. During the dataflow, Iperf can report the data rate and, in the case of UDP, datagram errors. The YellowJacket (YJ) is able to log data such as relative signal strength and relative correlations (power delay profile). In addition, YellowJacket logfiles can be *geocoded* with location and time information. Finally, a set of MATLAB programs were developed to aid in the visualization of YellowJacket logfile data. A short User's Guide is given in Appendix D.

Considerable effort was made to disable all power-saving features on the laptops and to eliminate all unnecessary processes (both operating system and applications) so that sustained and stable data transfer could be made. Without the power-saving features disabled, the results found with Iperf indicated that the data transfer rates were very erratic because the wireless card's RF interface would be cycled on and off.

The use of commercial 802.11x hardware was an expedient and relatively inexpensive way to verify the performance of this protocol. The problems such a wireless network faces in a terrestrial environment can in some cases be translated to a simulated Martian environment. The primary application of 802.11x wireless, in buildings and public places, is not directly applicable and some of the aspects of the design are not best suited for field use (low power, omnidirectional antennas, sealed units, and AC power needs).

The choice of Dell PC laptops was driven by the ATDI HerTZMapper software for Windows. This enabled the experimenters to use a single computer to run simulation models in the field and use the same computer for the data throughput measurements. Otherwise,

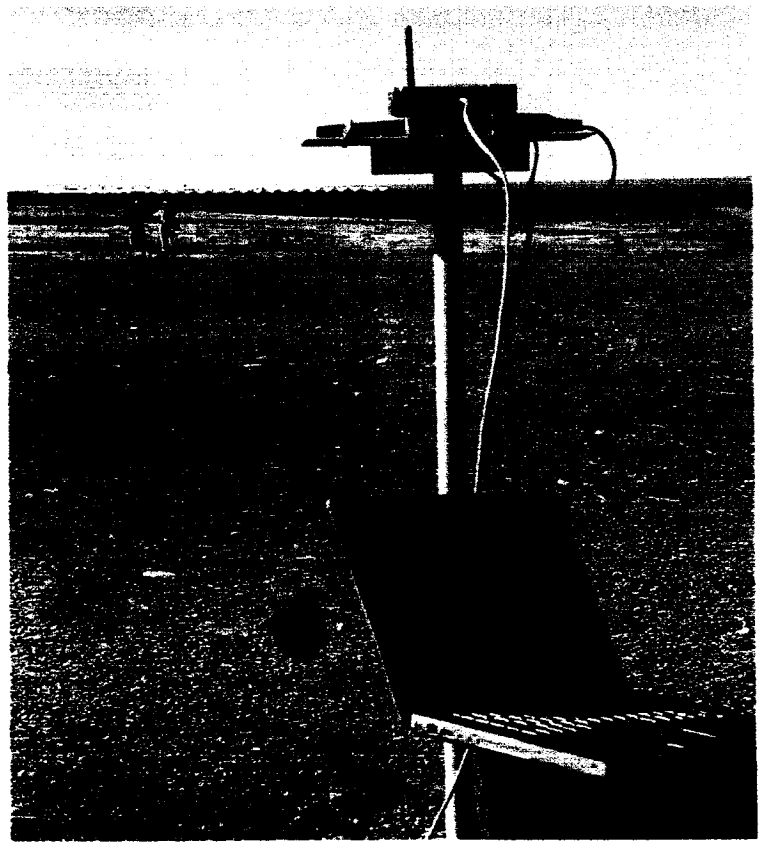
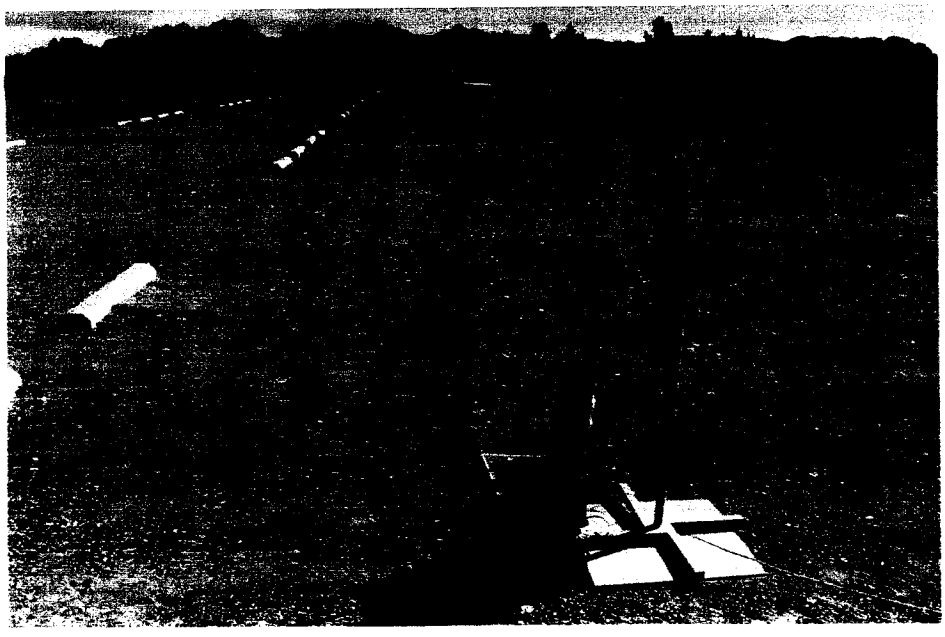


Figure 3.8: Measurements on the NMSU football stadium parking lot. In the foreground is a laptop hardwired to the wireless access point/router elevated to 1 m. In the background are personnel measuring RF data with the YellowJacket and data rates with another laptop. Flags are spaced every 20 m to a distance of 100 m.

(a)



(b)

Figure 3.9: (a) Reflector made of steel bookshelves and (b) setup for measurements in the simple reflection environment.

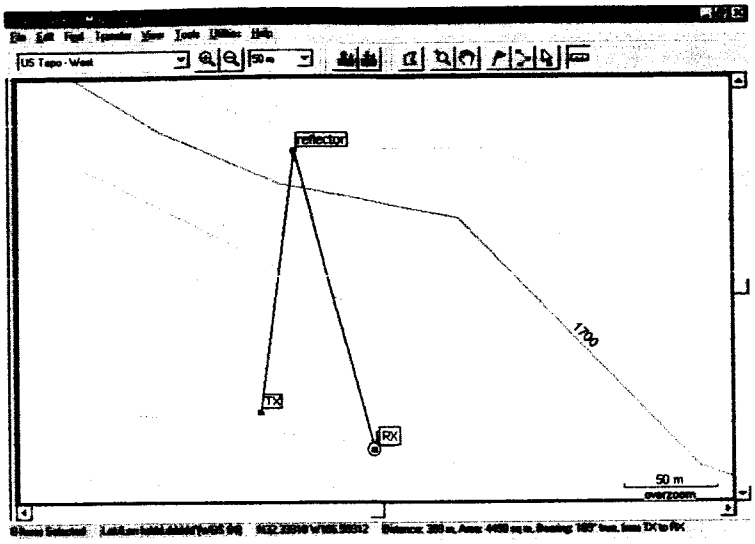


Figure 3.10: Arrangement of IEEE 802.11 access point (TX) and node (RX) at La Cueva (rock formations) in Dripping Springs.

a Mac PowerBook G4 with built-in "Airport" was available. During our testing, we found that the performance of the Apple Airport card (firmware version 9.42) was not as good as the Intel PRO/Wireless 2100 card in the Dell Latitude D600. But the burden of the Microsoft Windows XP operating system on Mac users was a major factor in the time it took to establish reliable, repeatable measurements with our wireless network.

Eventually, we had to establish an independent Wireless Local Area Network (WLAN) with no connection to the rest of the world or internet. We set up Wired Equivalent Privacy (WEP) to keep others from easily accessing our WLAN and adding mystery to our measurements. The Mac OSX has a utility called "Location Manager" that was a great help in changing the connections to the wireless Access Points (AP) or campus internet as needed. The XP OS did not have a similar capability (even on a laptop) and it was difficult, indeed, to switch from a wireless to ethernet link or return. A reboot was usually necessary. *Note:* This may have been a Windows issue that has since been resolved in Service Pack 2.

The necessity of an independent, self-contained, WLAN seems obvious, but our initial field tests brought out even more insidious elements that required tighter control. Our first field test to establish the distance limits of performance with 802.11b showed wild variations in performance, not related to distance. The appearance of periodic deteriorations in transfer rate caused us to suspect that an internal operating system function on the Dell was taking CPU cycles. A suspicion of "spyware" or background network access software created a need to investigate the Windows XP OS in depth. (see, for example, [25] and [26] for extensive additional information on these topics.)

There is a function of the XP OS that checks periodically for the availability of any type of network access, be it Modem, Wireless or Ethernet. This is because the Windows OS back

to Microsoft.com for software updates or other internal requirements. Spyware will attempt to upload information gleaned from a user's activities and report them on a periodic basis to an unannounced host. These effects can range from occasionally annoying to compromising the integrity of the computer. In either case, from the point of view of our experiment, they represent an uncontrolled network access and can affect the measurements made over the network by being a background noise source. Much effort was put in to disable these functions, and then to keep the PC laptop "sanitized" by never letting it connect to the general internet again. This complicated the exchange of data from the laptop to desktop computers in the lab, but we used a USB flash disk to transfer the necessary files and keep the laptop from being violated by connecting over an open Internet.

Further study of how Windows implements the Internet Protocols led to the measurement of Bandwidth Delay Product which is a measure of network performance. This Bandwidth Delay Product is the product of multiplying the "bottleneck bandwidth" (slowest hardware link speed) by the Round Trip Time (RTT). RTT is most readily measured by a network utility called `ping`. With `ping` , an Internet Protocol (IP) address is probed and the response time (in milliseconds) is reported. From the NMSU campus internet, a `ping` to NASA Glenn takes about 80 ms. We found that a `ping` over our WLAN to the Dell, one meter away, took about 300 ms. Clearly, there was something in our link that was amiss. Testing all possible configurations of wired and wireless using the D-Link AP focused the bottleneck at the Dell laptop wireless card only. It seemed a daunting task to determine what, if anything, could be done to locate the level at which this link was failing. There are (at least) two setup or configuration locations for the wireless card: the Windows Network Connections and the Intel ProSet application (see Figure 3.11). Two settings are made available for control - the RF power and the device power management configuration. The default RF power is set to highest, but the power/performance default is set for lowest power (appropriate on a laptop). What is not immediately clear is that this setting is not merely battery power usage, but Intel's implementation of Power Save Polling, an option in 802.11 to "turn off" a node when network usage is minimal. Not all AP's recognize Intel's implementation and apparently we were caught in that situation. It was fortuitous that a simple change of this setting improved the wireless performance of the Dell/Intel laptop from intermittent and unreliable to functionally usable. (see, for example, [27] for further information on this topic.)

In order to create a stable, realtime, testbed for IEEE 802.11 hardware it was necessary to severely restrict the configuration of the Dell laptop, and to a lesser extent, the Mac PowerBook. For best performance, the only non-operating system application running would be the command window where Iperf executes. Simultaneously executing network monitors or Secure Shell functions would allow enough CPU resources to keep the wireless data stream uninterrupted in any non real-time operating system (Windows, Linux, or Unix).

We also found that the UDP protocol in Iperf takes all the available CPU cycles on the Client computer it is running from. This is addressed in a forum response from ncsa. In order to guarantee a continuous data rate, the Iperf program uses "busy waiting" to provide accurate timing and subsequently absorbs all CPU resources to provide the stream. As Iperf is a performance testing tool, it should properly be used temporarily and as the only running

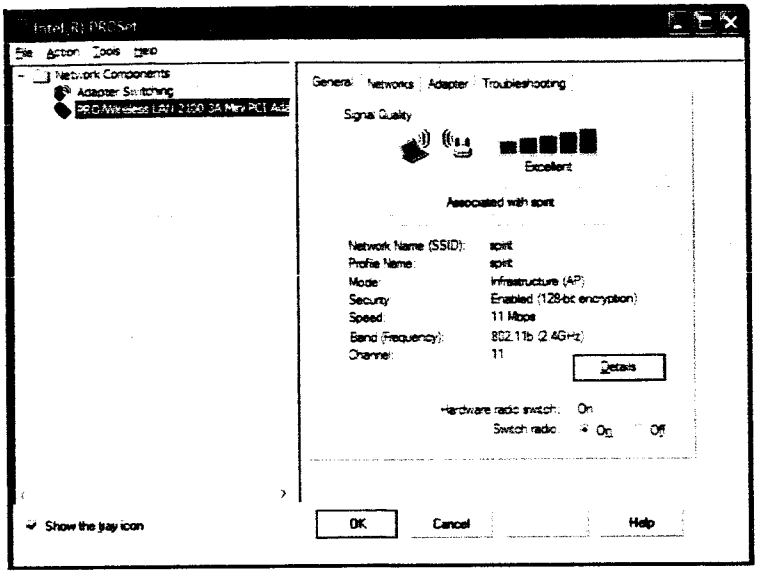


Figure 3.11: Intel Proset WLAN configuration window.

application [28].

Before finding Iperf, we had tried to transfer big data files over 802.1.b using AOL Instant Messenger, iTunes, and Quicktime. While the free Iperf tool might leave us wondering what we're seeing, it is, at least, quite controllable. The first realization we came to was the use of the terms "client" and "server"; they seemed counter-intuitive. Whatever the history of networking terminology, in Iperf the "client" sends the UDP data TO the "server" and, therefore, the client should be connected at the wireless link under test.

3.6.1 The Iperf Network Measurement Bandwidth Tool

Complete information regarding the installation and use of Iperf can be found at

`http://dast.nlanr.net/Projects/Iperf/`

Iperf must be installed on two computers connected (wired or wirelessly) to a LAN or the Internet. One computer is designated as the server and the other as the client. Dataflow originates at the client and terminates at the server. For our setup, the laptop hardwired to the access point is the Iperf client while other laptop is the Iperf server. On UNIX systems, Iperf is executed by typing Iperf commands in a terminal. On Windows systems, Iperf is executed by typing commands in the command line window.

An example of a TCP/IP data flow using Iperf is as follows. On one PC we type:

```
iperf -s -i 1
```

to denote this PC is the Iperf server and to update measurement information every one second. On the other PC we type:

```
iperf -c 10.0.0.1 -i 1
```

to denote this PC is the Iperf client which is connecting to a server whose IP address is assumed to be 10.0.0.1, and to update measurement information every one second.

An example of a UDP data flow using Iperf is as follows. On one PC we type:

```
iperf -s -u -i 1
```

to denote this PC is the Iperf server, UDP dataflow, and to update measurement information every one second. On the other PC we type:

```
iperf -c 10.0.0.1 -u -b 5.5M -t 60 -i 1
```

to denote this PC is the Iperf client whose IP address is assumed to be 10.0.0.1, a UDP flow at 5.5 Mbps for 60 seconds, and to update measurement information every one second. In the setup used in the outdoor measurements, we found that we could only sustain 5.5 Mbps without significant datagram errors in the free-field environment. This appears to be the baseline capacity of the wireless link in our setup.

The data output from Iperf can be redirected to a file for later analysis. Typical data is shown in Figure 3.12:

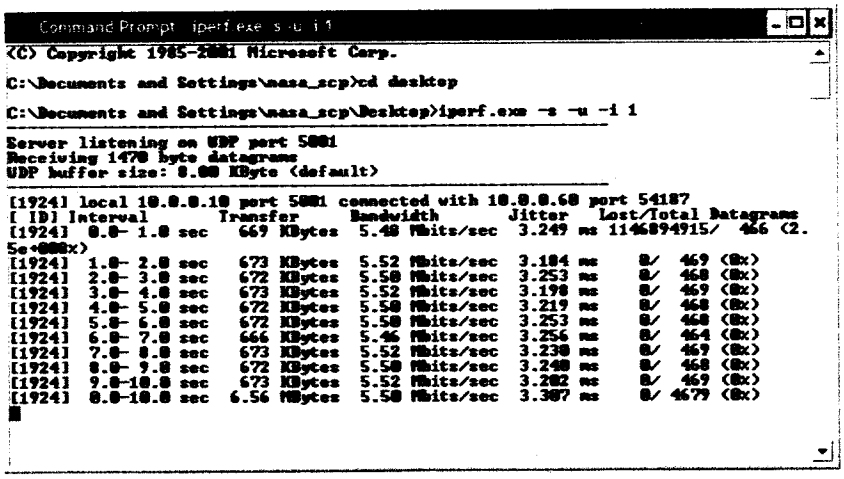


Figure 3.12: Example output from Iperf.

3.6.2 YellowJacket and IEEE 802.11b

The Berkeley Varitronics Systems YellowJacket PLUS b [29]is an 802.11b receiver wrapped around an HP (Compaq) iPAQ (model 5150 in ours) Personal Digital Assistant (PDA). The PLUS version includes a Global Positioning System (GPS) receiver and antenna jack.

Software on the iPAQ provides an interface to the receivers and makes measurements of the 802.11 RF environment along with location data for the position of the YellowJacket

(YJ). The data can be logged on the iPAQ and then transferred to a PC (Windows only) via the built-in infrared (ir) port on the iPAQ. A BAFO brand ir/USB interface is provided by BVS, but we could never get it to work with our Dell Latitude laptop. Since the Windows OS gives several options to configure a new communications device, much frustration was expended just to establish communications with the YJ. We found that the built-in ir port on the Dell laptops was not enabled by the default OS and BIOS settings. With assistance from a systems engineer familiar with the BIOS, we finally established ir communications with the YJ. But, that link is quite slow (is it 9600 baud?) and there are a number of translations that must take place to view YJ data: a translation from ir to file using Microsoft ActiveSync, a translation from PDA format to "desktop" format, a translation from BVS format to Excel using BVS "Chameleon" and finally using MS Excel to create a graph. We eventually wrote custom MATLAB software to analyze the YJ logs.

Some of the features of the YJ don't work as well as we would like, but it is nonetheless a unique and useful tool:

1. The GPS receiver performance is mediocre compared to the handheld units meant for hiking or navigation.
2. The screen snapshot function of the YJ only works for the Spectrum screen.
3. The Marker function to update the log file does not work - this would have been a way to synchronize the YJ and Iperf data.
4. The Relative Correlation screen only logs 22 data points out of the 44 measured.
5. The Delay page does not log at all, although that data can be derived.
6. The unit has an odd behavior with the RSSI display; this signal strength has spikes to a low value whenever the data rate increases toward the 11 Mbps rate.
7. The log files slow their update as the data rate increases. At the management and housekeeping rate of 1 Mbps, the log updates around 8 Hz, but when data is streaming this slows to 1 Hz or less.
8. The Relative Correlation display can easily "alias" or wrap- around when the "chip" delay exceeds 7 chips.
9. Finally, the battery performance of the iPAQ is not appropriate for field equipment.

We learned much about the 802.11b protocol. First, while the advertised speed is 11 Megabits per second (Mbps), we could never achieve that as a channel throughput rate. In fact, it is documented [30] that, due to handshake overhead and synch headers, the maximum possible data throughput rate is only 7.1 Mbps for UDP and 5.9 Mbps for TCP links. We initially tested, using Iperf UDP, with a transfer rate of 7.35 Mbps (empirically derived) in the lab. This was used in the field to stress the link and determine the limits of performance, with distance being the only variable. After that test regimen, we lowered our rate to 5.5

Mbps to have some headroom and get reliable performance out of the link. In our field testing we constantly ran up against the issue of low power versus distance. We needed to increase the TX-RX distance to simulate long multipath or delayed reflections. But, at the same time, the loss of RF power (r-squared free space loss) would sometimes put our signal too near the noise floor. The use of directional antennas was required. The possibility of purchasing an 802.11 RF power booster exists, but the expense was not justified. The Cushcraft S24012P panel antennas offer +12 dBi gain due to the pattern. One aspect of the 802.11 specification serves to meet an FCC requirement that limits the effective radiated power of 802.11b devices - the combination of RF power and antenna gain cannot exceed 4W EIRP [31]. One way the manufacturers hold to this rule is by using an antenna connector that makes it inconvenient to swap antennas on an Access Point. These "reverse gender" SMA connectors keep the AP/antenna combination proprietary. We either replaced the SMA on the AP or used a purchased adaptor to connect to the Cushcraft panels. We were still under the legal limit after the modification.

We learned that two AP's will not connect wirelessly; a "bridge" or "router" is needed for this function. We found that the antennas internal to a laptop might be in an unknown orientation. We did some limited experimentation to find an optimum laptop orientation, which wasn't usually horizontal on a desktop! In the close proximity of an office environment, even the polarization loss from vertical to horizontal may be insignificant. For field use, commercial AP's don't offer enough RF output, and should be possible to power from DC directly; we had to carry bulky DC/AC inverters to run the measly "wall warts" for the AP's.

In order to use two directional antennas, we needed another standard SMA connection to our laptops. The Dell Intel wireless card was not amenable to having an external antenna. We tried a Linksys WUSB11 802/USB dongle, but it never achieved error free performance with the drivers provided. We borrowed a Linksys WET11 Bridge and connected via ethernet (cat5 cable) to the Dell (the Mac is cat5 to the D-link hardwire). Using the SMA gender changer, we could then have two wireless connection to the Cushcraft directional antennas (The YJ antenna is a standard SMA). In our experience, any change or addition to the Dell/Windows laptop was a lesson in frustration for Mac users.

The results of our field testing confirmed most assumptions about the performance of 802.11b. We had incremental surprises and detours.

1. The performance of a commercial AP with its omnidirectional antenna falls off after about 60 meters in the free field.

2. The presence of a reflector about 40m away does not destroy an established 802.11b Iperf link.

3. The presence of a reflector about 90m away, along with the long path/low power situation does affect Iperf performance.

4. The intentional orientation of the RX antenna in a multipath configuration (La Cueva) did result in a substantial degradation of Iperf performance.

4 Results and Discussion

4.1 Introduction

In this section, we describe the results from the validation study and the physical layer simulations. The validation study is based outdoor measurements made in the region around NMSU. The simulation study is to estimate the characteristics of the physical layer performance for the 802.11 a/b protocols on the Martian surface based on the DEM data.

4.2 Validation of Power Delay Profile Simulations

Power Delays Profiles were simulated at the Tortugas Mountain and Dripping Springs sites and a simple RF reflection measurement was also made at these sites. Table 4.1 shows the direct path and reflected path RF measurement results. For Tortugas Mountain, the reflection is 24 dBm less than the power for the direct path (since the sidelobe power levels were below the noise floor, the 24 dBm figure is entirely due to reflected power).

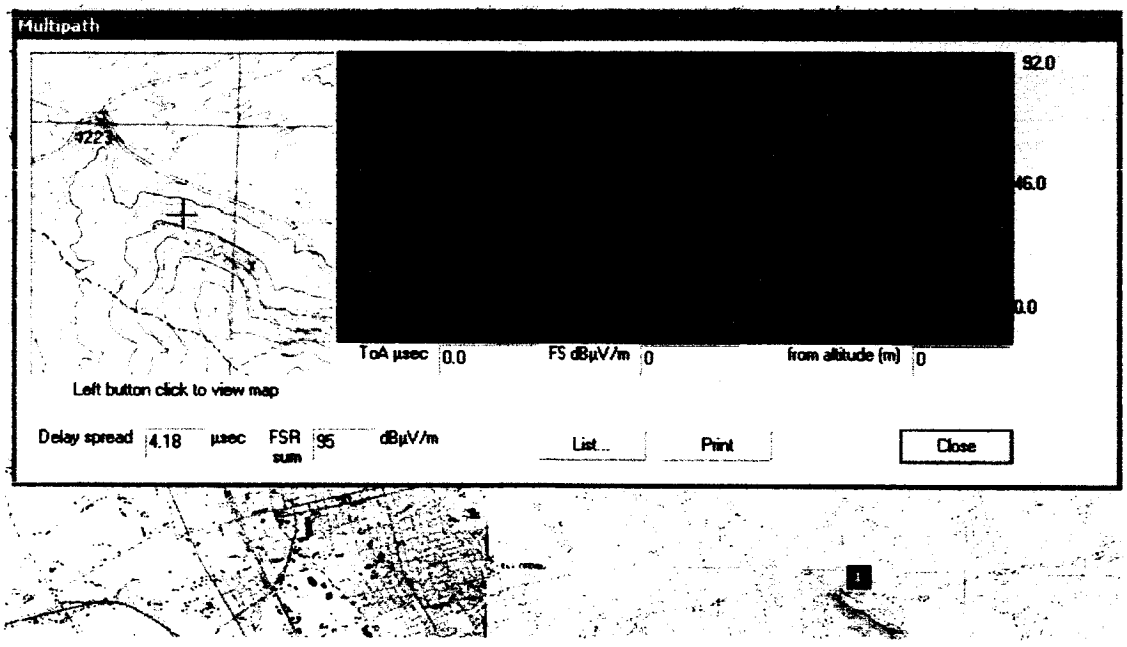
Table 4.1: PDP validation data.

	Relative Power Level Difference Between Direct and Reflection Paths	Reflection Delay
Tortugas Mountain–Simulation	20 – 25 dBm	0.6 μs
Tortugas Mountain–Measurement	23 dBm	1.4 μs
Dripping Springs–Simulation	26 dBm	0.7 μs
Dripping Springs–Measurement	20.5 dBm	0.87 μs

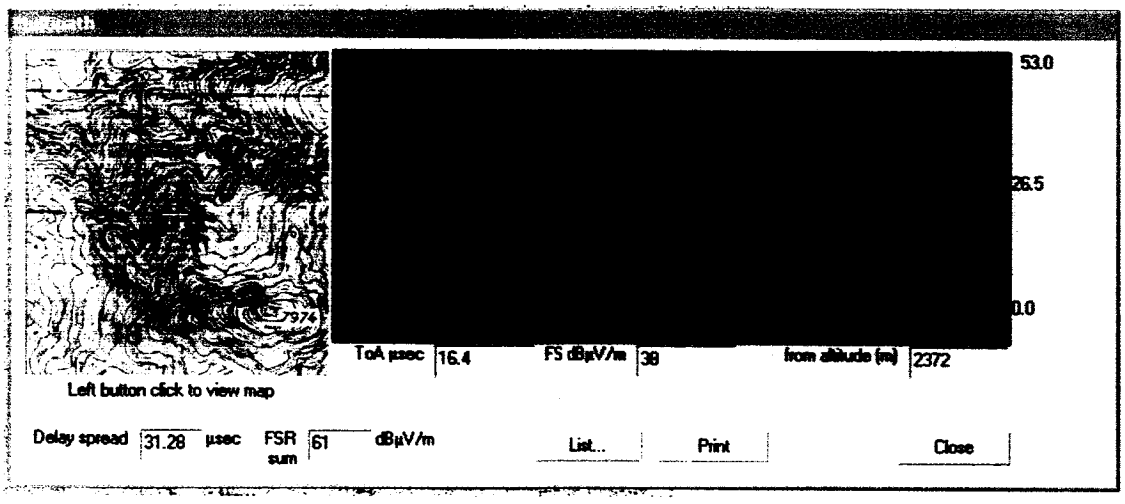
Using the appropriate DEMs, antenna patterns, radiated power levels, and positions of transmit and receive antennas, the power delay profiles were simulated for the local sites. Figure 4.1 gives the PDPs for Tortugas Mountain and Dripping Springs. In these figures, we see for Tortugas Mountain a strong response occurs at approximately 1.4 μs and is 25 dBm down from the direct path and for Dripping Springs a strong response occurs at approximately 0.7 μs and is 26 dBm down from the direct path. This agrees quite closely with measurement data.

4.3 Power Delay Profile Measurement Data

Figure 4.2 shows the PDPs measured using the YellowJacket at the NMSU football stadium parking lot (free field). With the access point at a fixed location, the PDP was measured at 40 m and 80 m distances in the south and east directions (south leg and east leg). The plots show the PDP (relative correlation vs. delay in chips) at various points in time. Figure 4.3 shows the PDPs measured at the NMSU football stadium parking lot in the presence of the reflector. With the access point at a fixed location, we measured the PDP at 40 m with and



(a)



(b)

Figure 4.1: Power delay profile simulation results from (a) Tortugas Mountain and (b) Dripping Springs.

without the reflector. We clearly see in Figure 4.3(b) the secondary path delayed by 3 chip periods relative to the direct path centered at 0 chips.

Figure 4.4 shows the PDPs (relative correlation vs. delay in chips) at various points in time measured using the YellowJacket at the La Cueva rock formations at Dripping Springs (complex multipath). Figure 4.4(a) illustrates the PDP when the antennas were pointed directly at each other. With this setup, there is essentially a single, direct path. Figure 4.4(b) illustrates the PDP when the antennas were pointed at La Cueva. With this setup there is clearly, significant multipath with the assumed direct path centered at 0 chips and secondary reflections around 1.5 chips and -2 chips (aliased).

4.4 Outdoor IEEE802.11b Results–Stadium Parking Lot

Prior to measuring the performance of IEEE802.11b outdoors in a complex multipath environment, measurements were made in the presence of a single reflector (three large steel bookshelves) in the NMSU football stadium parking lot. In the absence of the reflector the parking lot is essentially a free field (see previous subsection). As described in the previous section, a baseline measurement was made without the bookshelf (directional antennas pointed toward each other) and a measurement was made with the reflector (mainlobes pointed at the bookshelf with first sidelobes pointed directly toward each other). The measurements were of network bandwidth (fastest data rate without significant UDP datagram loss for the baseline) and percentage of received packets. Figure 4.5 shows plots of the data for a 100 second time period. For the free field we see virtually no packet loss at a sustained rate of 5.5 Mbps, however, in the presence of a very simple reflector, we have significant packet error rates while attempting the same rate UDP data flow.

4.5 Outdoor IEEE802.11b Results–Dripping Springs

In order to access the performance of IEEE802.11b in the presence of multipath such as that found in an outdoor environment, several tests were conducted at the La Cueva rock formations at Dripping Springs. These tests were described in the previous section. For the direct path test, both directional antennas were pointed straight at each other and the resulting network bandwidth is given in Figure 4.6(a). This result serves as our baseline since there is no multipath involved. For the multipath test, each antenna was pointed at La Cueva which resulted in a strong multipath environment. The resulting network bandwidth is given in Figure 4.6(b).

4.6 Simulated WLAN Performance at Different Sites

Data packets for 802.11a or 802.11b were generated according to the PPDU format shown in Fig. 3.7. The simulation software is developed around the mWLAN toolbox from CommAccess Technologies [32]. The data packets from the transmitter are sent through a random multipath channel generated for the particular transmitter and receiver locations on the

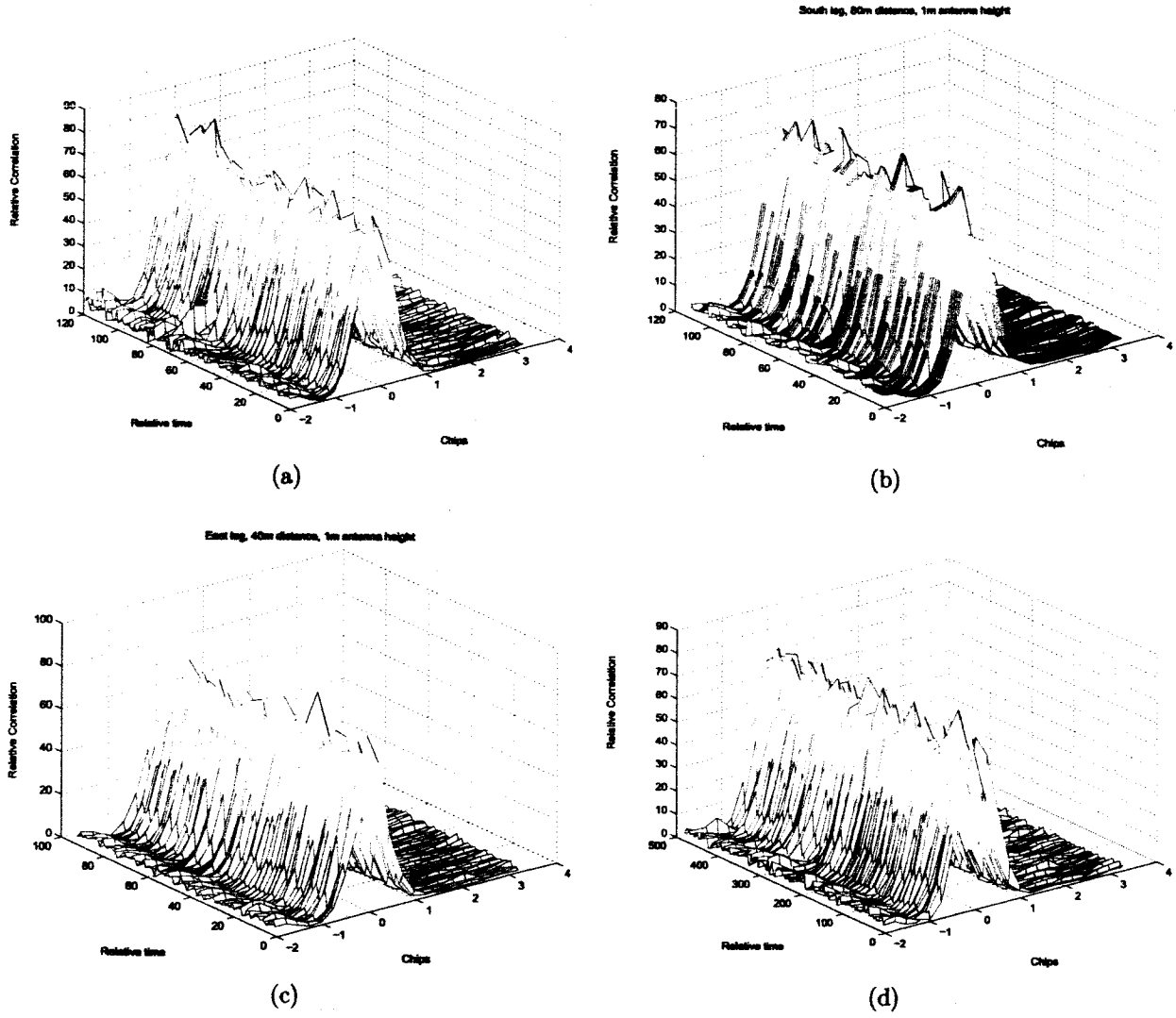
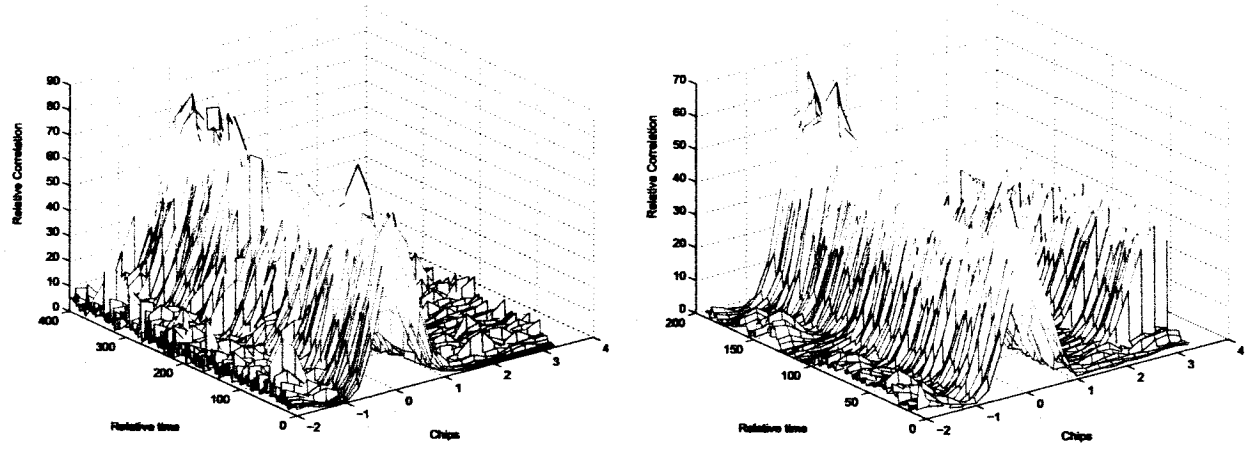
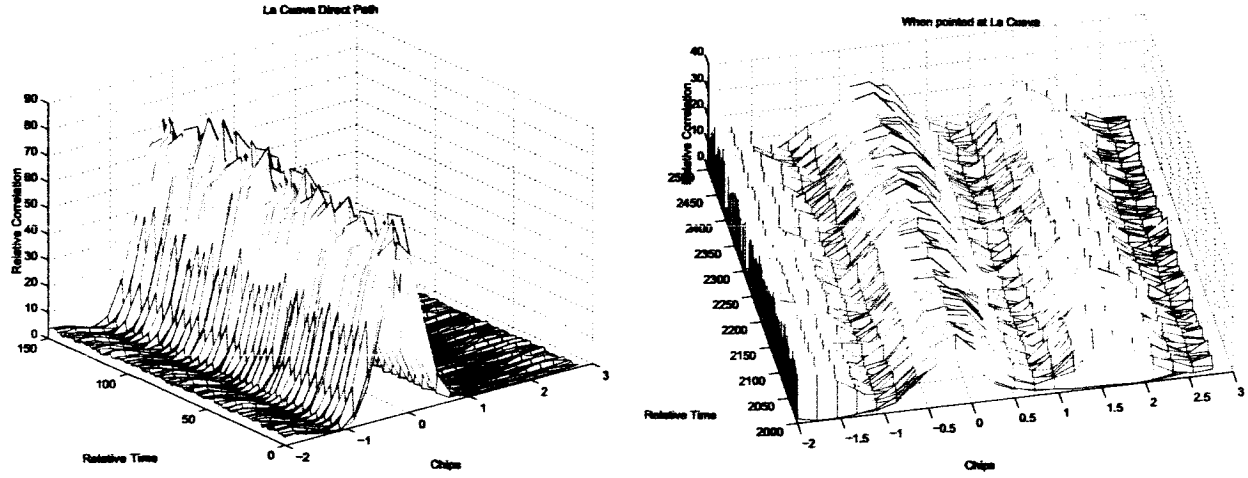


Figure 4.2: Power delay profiles of the NMSU football stadium parking lot (free field) as measured using the YellowJacket. Plots show the PDP (relative correlation vs. delay in chips) at various points in time. (a) South leg, 40 m distance; (b) South leg, 80 m distance; (c) East leg, 40 m distance; and (d) East leg, 80 m distance



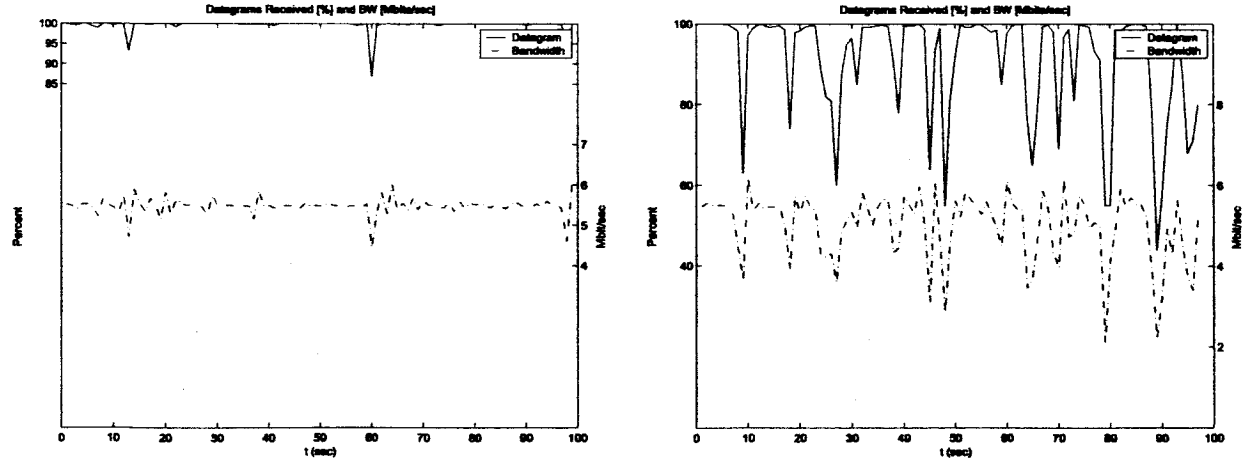
(a) (b)

Figure 4.3: Power delay profiles of the NMSU football stadium parking lot with and without a reflector present. Plots show the PDP (relative correlation vs. delay in chips) at various points in time. (a) without reflector showing a single, direct path centered at 0 chips and (b) with a reflector showing a secondary path at 3 chip periods and a direct path centered at 0 chips.



(a) (b)

Figure 4.4: Power delay profiles of the La Cueva rock formations at Dripping Springs. Plots show the PDP (relative correlation vs. delay in chips) at various points in time. (a) with directional antennas pointed directly toward each other, direct path centered at 0 chips and (b) with directional antennas pointed toward rock formations, direct path centered at 0 chips and multiple secondary paths.



(a) (b)

Figure 4.5: Outdoor IEEE802.11b network bandwidth and percent UDP received packets for (a) free field and (b) simple reflection.

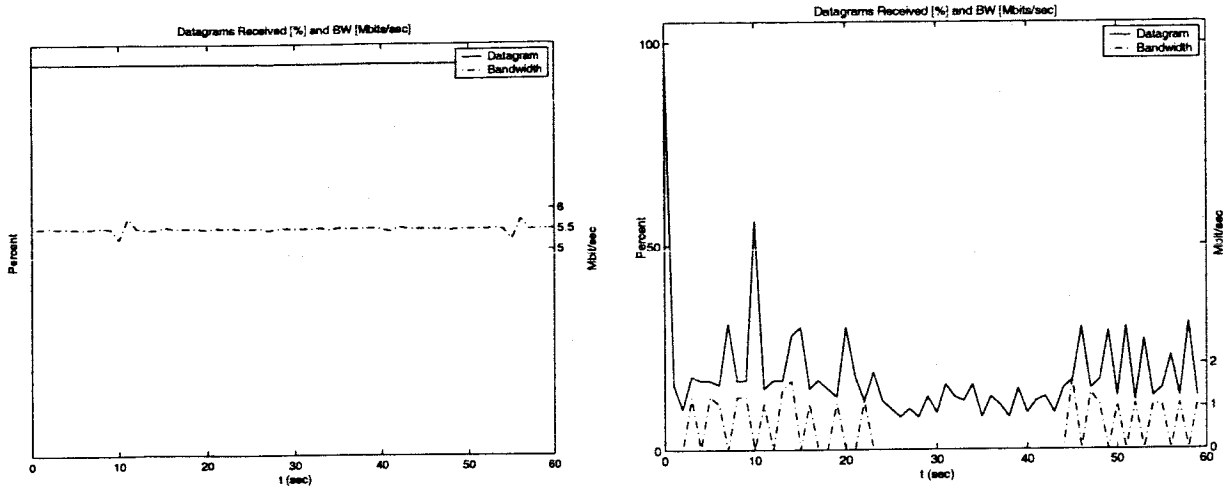


Figure 4.6: Outdoor IEEE802.11b network bandwidth and percent UDP received packets for (a) free field and (b) multipath environment.

Martian surface using the ICS Telecom software. The received packets are processed by the corresponding receiver. The 802.11b receiver's performance is studied with and without a RAKE structure. Note that a RAKE receiver coherently combines different multipath contributions before detection and thus improves performance. For both 802.11a and b, only truncated channel impulse responses are estimated at the receiver using the corresponding PLCP Preamble.

4.6.1 Performance versus distance between the transmitter and the receiver

In order to obtain packet error rate (PER) and bit error rate (BER) results versus distance, it is necessary to estimate both the received signal and the input-referred noise for an 802.11 receiver on the Martian surface. The RF propagation simulations using ICS Telecom provide an estimate of electric field intensity at the receiving antenna. A first-order estimate of receiver noise is based on a noise figure $F_R = 7.2$ dB for a typical 802.11a receiver implementation [33]. Assuming noise figure is measured for a reference temperature $T_0 = 290$ K, the equivalent noise temperature for the Martian receiver may be calculated [34] as $T_R = (F_R - 1)T_0 = 1522$ K. An omnidirectional antenna pattern sees roughly half sky and half surface, so we approximate the brightness temperature (T_b) as $T_b = T_p/2 = 250$ K / 2, where T_p is the physical temperature. Further assuming a radiation efficiency $\eta = 0.9$, we find an equivalent temperature for the antenna of $T_A = \eta T_b + (1 - \eta)T_p = 138$ K. Thus, our simulations use an equivalent noise temperature for the receiver input of $T_{eq} = T_A + T_R = 1560$ K.

The packet error rates (PER) for various distances (d) between the transmitter and

receiver are given in Tables 4.2-4.6. Note that a CRC failure is considered as a packet error in 802.11b while any error in the OFDM SIGNAL symbol constitutes a packet error in 802.11a. Transmit power is 1 W, and antenna height is 1.5 m above the ground, for both 802.11a and b. The 802.11b results in the table are obtained without RAKE. The data rates for 802.11a and b are 12 Mbps and 11 Mbps respectively.

The packet error rate tables show that both 802.11a and b perform well for receivers within several hundred meters from the transmitter. In some cases, we find better packet error performance at a longer distance (d). For example, with 802.11a at Gusev1 Site2, PER at 500 m appears to be better than the PER at 200 m. Similarly, in the case of Hematite4 Site1, the PER at 200 m is better than the PER at 100 m. In the case of Gusev1 Site2, we observe that while the received power is higher at 200 m, the rms delay spread in this case is smaller for $d = 500$ m, resulting in fewer packet errors. A similar comment can be made about the Hematite4 Site1 PER result. In the case of 802.11b, the performance at 100 m is better than the performance at 50 m for Gusev1 Site2 as well as for Hematite4 Site1. We notice a similar phenomenon as observed in the case of 802.11a, that is, although the received power is smaller for 100 m, the rms delay spread becomes smaller too. Thus, it appears that when sufficient power is transmitted (1 W in this case) the multipath effects play a dominant role on the performance of both 802.11a and 802.11b. Finally, we note that the results with very low PER values must be used with caution as they are not statistically significant due to the small number of packet errors observed from transmitting 20,000 data packets.

The effect of distance on the bit error rate (BER) performance is shown in Fig. 4.7 for Gusev1 Site1. The BER result for each distance is an average over four different locations at the same distance. Transmitted power is 1 mW for all cases, and 802.11b results are obtained using a RAKE receiver. The data rate for 802.11a is 12 Mbps and the 802.11b transmits at the rate of 11 Mbps. In the case of 802.11a, the overall BER seems to increase with distance except for a strong dip at 500 m. This BER dip is believed to be due to favorable terrain conditions at that distance and it agrees well with the PER result in Table 4.2. The BER for 802.11b seems to be nearly constant up to distances of 1000 m except for a dip at 500 m similar to 802.11a.

4.6.2 Effect of transmit power on PER

Although Tables 4.2–4.6 show PER results for 1 W of transmit power, it is instructive to study the effects of transmit power on the PER. This is investigated via Table 4.7 for Gusev1 Site1. The table shows that when the transmit power is small, 802.11b seems to be doing better than 802.11a. As the transmit power increases, the performance for both 802.11a and b tend to flatten out for high transmit power, but 802.11a shows much better performance than 802.11b. Note that the rms delay spread for this location is 0.105 μs for 802.11a, and it is much less than the available 0.8 μs guard period. Thus, 802.11a can handle this delay spread quite well, and its performance keeps improving with the transmit power. As the transmit power becomes large, however, the multipath events with delays

Table 4.2: Packet Error Rate Performance at Gusev1 Site1. A '-' indicates zero packet errors, in 20,000 packets.

d (m)	rms delay spread (μs)		Received power (nW)		PER	
	802.11a	802.11b	802.11a	802.11b	802.11a	802.11b
20	0.194	0.268	40.9	79.3	0.0008	0.0983
50	0.144	0.203	38.6	75	0.0004	0.0768
100	0.105	0.155	36.4	71	0.0001	0.0572
200	0.180	0.153	70.0	206	0.0001	0.0281
500	0.091	0.092	61.7	145	-	0.0158
1000	17.3	1.86	.00011	0.0009	1	0.9619

Table 4.3: Packet Error Rate performance at Gusev1 Site2.

d (m)	rms delay spread (μs)		Received power (nW)		PER	
	802.11a	802.11b	802.11a	802.11b	802.11a	802.11b
20	0.146	0.186	38.2	79	0.0003	0.115
50	0.131	0.155	26.7	56	0.0004	0.082
100	0.095	0.126	25.8	54	0.0001	0.032
200	0.713	0.719	0.0822	0.16	0.099	0.51
500	0.472	0.476	0.0114	0.02	0.067	0.53

Table 4.4: Packet Error Rate performance at Gusev1 Site3. A '-' indicates zero packet errors, in 20,000 packets.

d (m)	rms delay spread (μs)		Received power (nW)		PER	
	802.11a	802.11b	802.11a	802.11b	802.11a	802.11b
20	0.143	0.17	52.0	119	0.0002	0.1
50	0.055	0.1	45.2	102	-	0.027
100	0.055	0.065	45.9	103	-	0.016
200	0.070	0.089	34.8	81	0.0001	0.03
500	11.2	9.2	.00001	0.0001	1	0.54
1000	0.742	0.718	.000001	.00003	1	1

Table 4.5: Packet Error Rate Performance at Hematite4 Site1.

d (m)	rms delay spread (μs)		Received power (nW)		PER	
	802.11a	802.11b	802.11a	802.11b	802.11a	802.11b
20	0.741	0.634	59.2	114.27	0.0262	0.2113
50	0.747	0.625	49.1	94.02	0.0272	0.2844
100	0.584	0.564	47.0	80.62	0.0138	0.1667
200	0.289	0.297	29.3	46.07	0.0026	0.1196
500	0.069	0.087	22.5	43.24	0.0001	0.0478
1000	0.696	0.685	.0374	0.167	0.4405	0.3312

Table 4.6: Packet Error Rate Performance at Hematite5 Site1.

d (m)	rms delay spread (μs)		Received power (nW)		PER	
	802.11a	802.11b	802.11a	802.11b	802.11a	802.11b
20	1.031	0.913	45.9	88.54	0.0037	0.1280
50	0.755	0.694	36.2	69.04	0.0012	0.0950
100	0.475	0.498	29.2	55.97	0.0004	0.0724
200	0.178	0.228	28.6	55.30	0.0001	0.0354
500	0.160	0.204	42.3	88.95	0.0004	0.0370
1000	0.287	0.316	2.10^{-7}	2.10^{-6}	1	1

Table 4.7: Effect of Transmit Power on PER for Gusev1 Site1 at a distance of 100 m from the transmitter. The 802.11b receiver is implemented without a RAKE structure.

Transmit Power	PER for 802.11a	PER for 802.11b
1 μW	0.985	0.4183
10 μW	0.381	0.1719
100 μW	0.0225	0.1011
1 mW	0.0021	0.0625
10 mW	4×10^{-4}	0.0612
100 mW	2.5×10^{-4}	0.0555
1 W	2×10^{-4}	0.0516

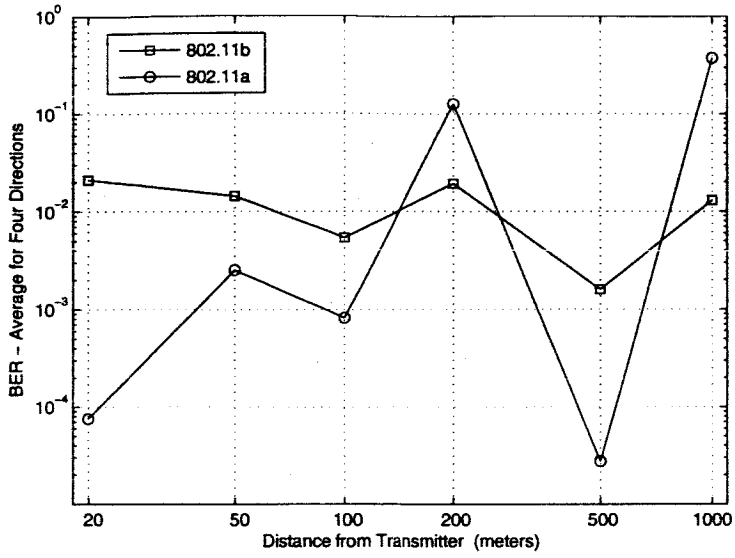


Figure 4.7: BER Performance for 802.11a and b at Gusev1 Site1.

exceeding 0.8 μs start affecting its performance with adjacent symbol interference. This limits the performance improvement. In the case of 802.11b too, the presence of multipath propagation effects does not allow performance improvement beyond a certain value.

4.6.3 BER Performance versus SNR

The bit error rate (BER) performance results versus SNR are shown in Figs. 4.8-4.17 for IEEE802.11a and b. In the case of 802.11a, we notice that lower data rates provide much better BER performance giving several dB advantage over higher rates. However, it is also to be noted that lower rates need to transmit longer than higher rate modes in order to send the same amount of information. We also see that the curves tend to flatten at the higher SNR region as the performance becomes more dominated by the delay spreads. Although the rms delay spread is within 0.8 μs for the cases studied in these figures, there are still multipath channels beyond 0.8 μs producing adjacent symbol interference. The BER performance curves for 802.11b show that multipath channels can severely affect their performance. Figures 4.13-4.17 show results without a RAKE structure. Another interesting observation is that CCK performs better than the other modulations in some cases.

4.6.4 Effect of using RAKE for 802.11b

The use of a RAKE receiver can significantly improve the BER and PER performance for 802.11b. The BER performance improvements can be seen comparing Figs. 4.13 and 4.18 for Gusev1 Site1. The PER performance improvements are summarized in Table 4.8. The table shows that RAKE can provide PER improvement by a factor as high as eight in this

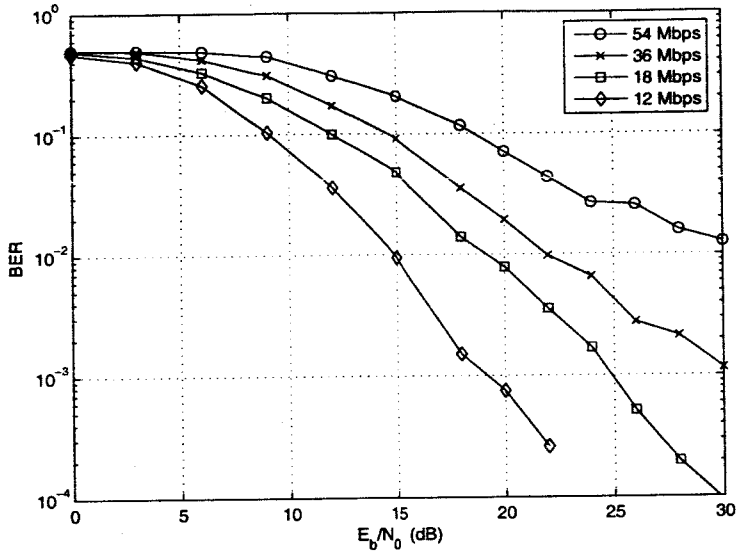


Figure 4.8: BER Performance for 802.11a at Gusev1 Site1.

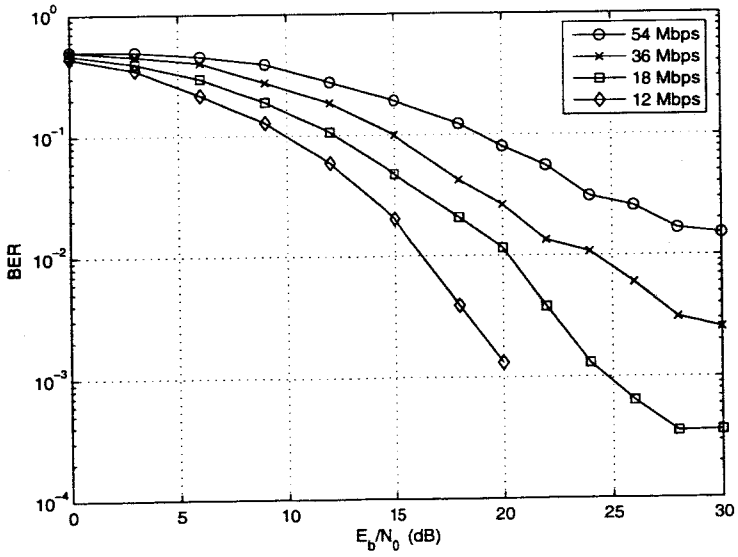


Figure 4.9: BER Performance for 802.11a at Gusev1 Site2.

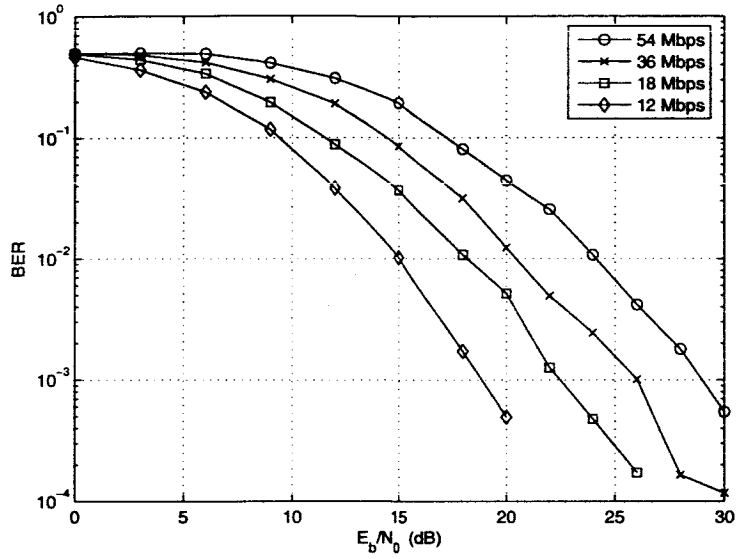


Figure 4.10: BER Performance for 802.11a at Gusev1 Site3.

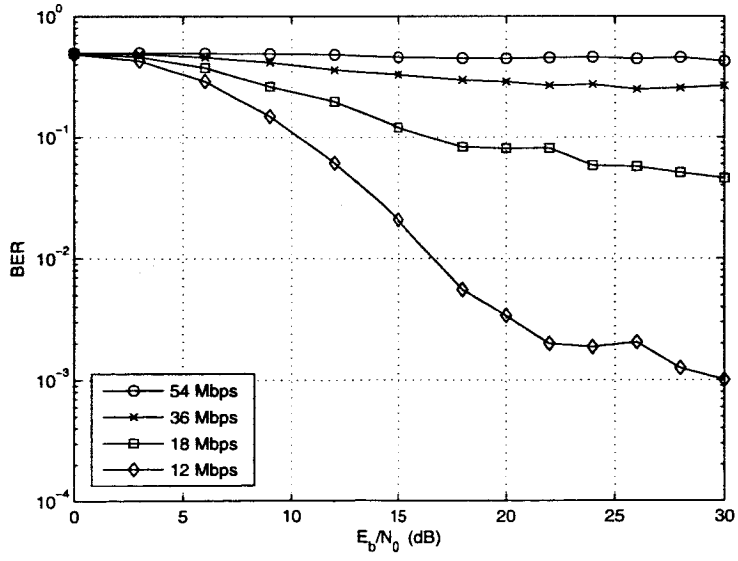


Figure 4.11: BER Performance for 802.11a at Hematite4 Site1.

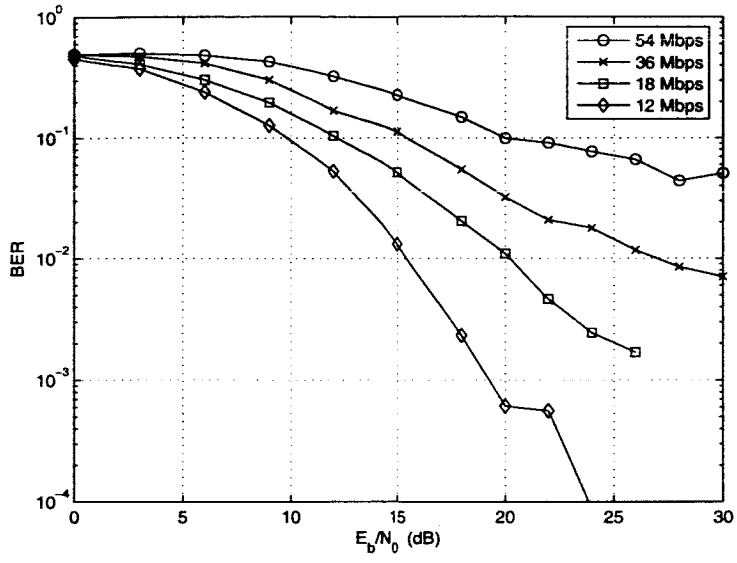


Figure 4.12: BER Performance for 802.11a at Hematite5 Site1.

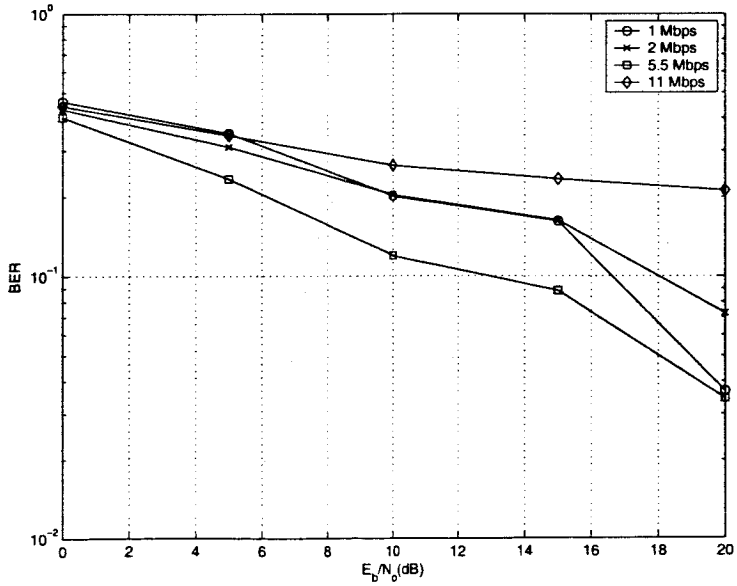


Figure 4.13: BER Performance for 802.11b at Gusev1 Site1 without a RAKE structure.

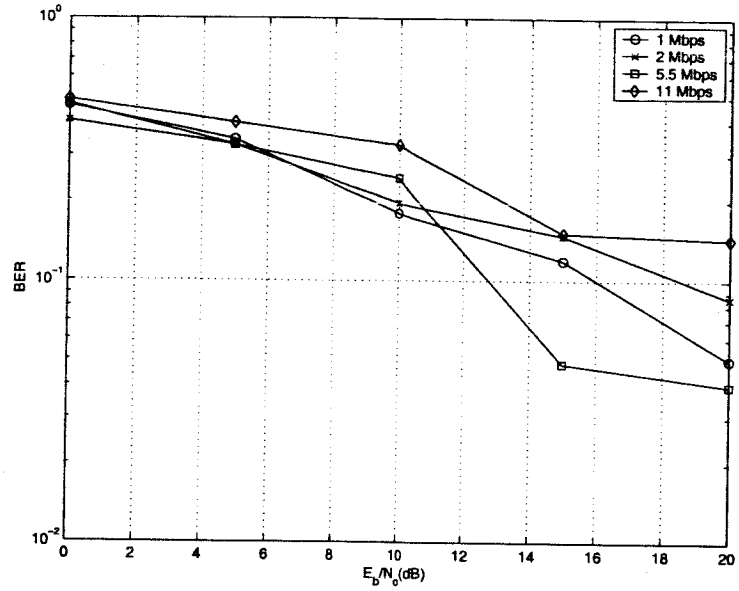


Figure 4.14: BER Performance for 802.11b at Gusev1 Site2 without a RAKE structure.

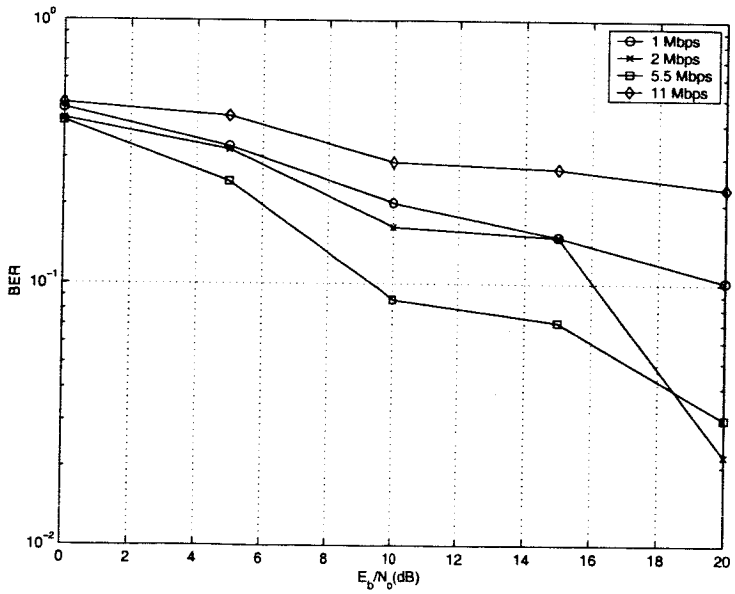


Figure 4.15: BER Performance for 802.11b at Gusev1 Site3 without a RAKE structure.

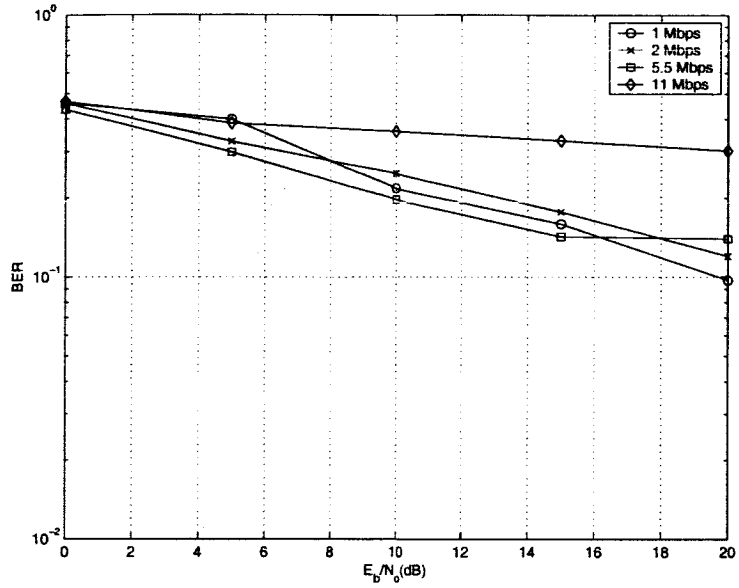


Figure 4.16: BER Performance for 802.11b at Hematite4 Site1 without a RAKE structure.

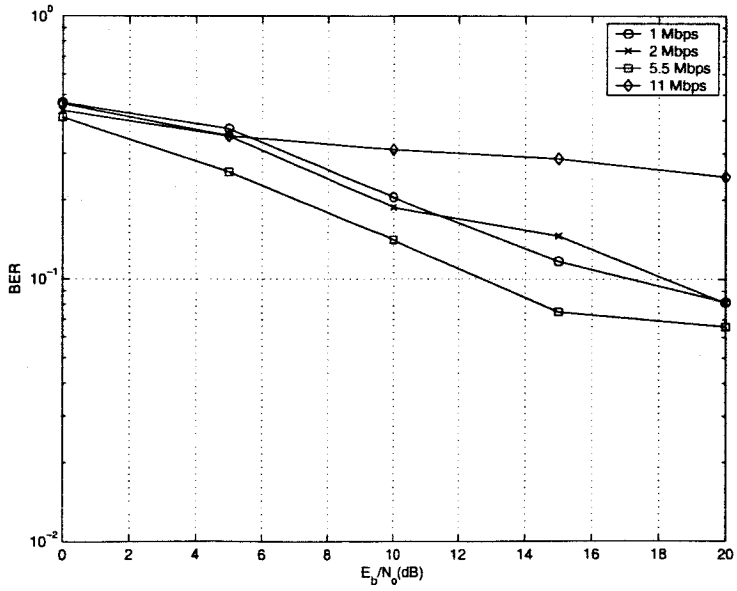


Figure 4.17: BER Performance for 802.11b at Hematite5 Site1 without a RAKE structure.

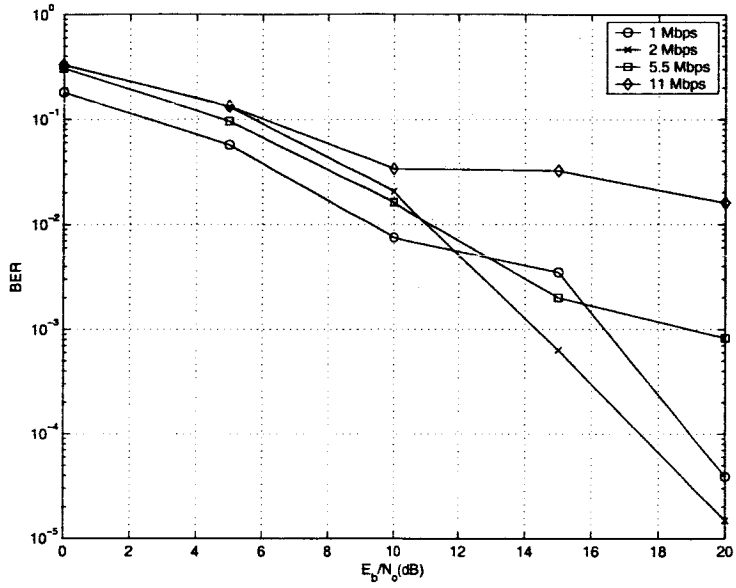


Figure 4.18: BER Performance for 802.11b at Gusev1 Site1 with a RAKE receiver.

case. The performance improvement seems to be generally smaller at very large distances.

4.6.5 Performance versus antenna heights

The antenna heights can affect the performance of both 802.11a and b significantly. An increase in the antenna heights can provide better line-of-sight signals over a rocky terrain and can increase the received power. However, it can result in more delay spreads as well, resulting in decreased performance at the receiver. In the case of 802.11a, we can observe from the PER tables that, of the three sites considered, Gusev1 Site3 has the least rms delay spread at 100 m. Since the received power is too low, the benefit from an increase in the received power becomes significant since the rms delay spread remains much smaller than the guard interval. Thus, when the antenna heights are raised, this site shows significant improvement in performance despite an increase in the rms delay spread value from 0.036 μs (corresponding to antenna height of 0.5 m) to 0.058 μs (corresponding to antenna height of 2.0 m). The results in 802.11b do not show significant improvements with antenna heights as in 802.11a. This may be because the benefit due to more received power is nearly cancelled by the loss due to increased delay spreads. Finally, increasing the heights of the antennas beyond a certain value may be impractical for mobile rovers.

4.6.6 Discussion

There are several interesting observations.

- The received power for 802.11b is always greater than 802.11a. This makes sense since

Table 4.8: Packet Error Rate Performance at three sites in Gusev1 for IEEE 802.11b. The '-' indicates non-availability of results. The transmit power is 1 W and the antenna heights are fixed at 1.5 m above the ground.

d (m)	Site 1		Site 2		Site 3	
	Without RAKE	With RAKE	Without RAKE	With RAKE	Without RAKE	With RAKE
20	0.098	0.024	0.115	0.022	0.10	0.043
50	0.077	0.014	0.082	0.011	0.027	0.008
100	0.057	0.01	0.032	0.008	0.016	0.005
200	0.028	0.005	0.51	0.284	0.03	0.006
500	0.016	0.002	0.53	0.315	0.54	0.234
1000	0.962	0.52	-	-	1.00	0.682

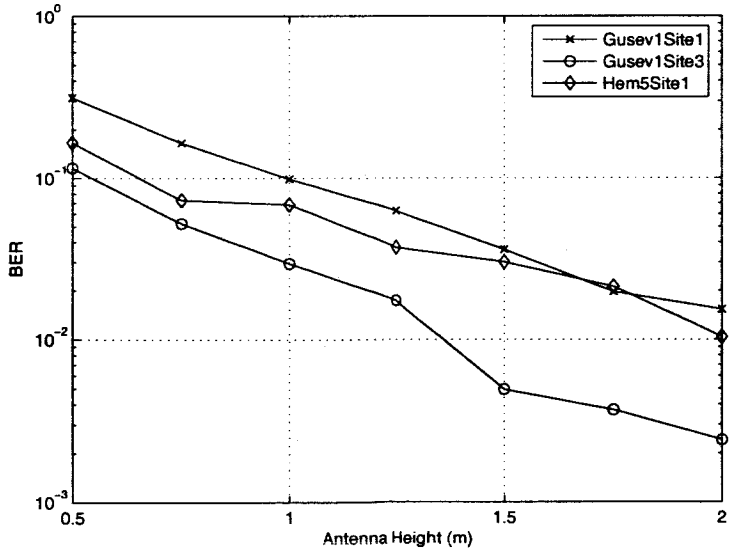


Figure 4.19: BER Performance for 802.11a at Gusev1 Site1. The transmit power is 100 μW, and the distance (d) between the transmitter and the receiver is 100 m.

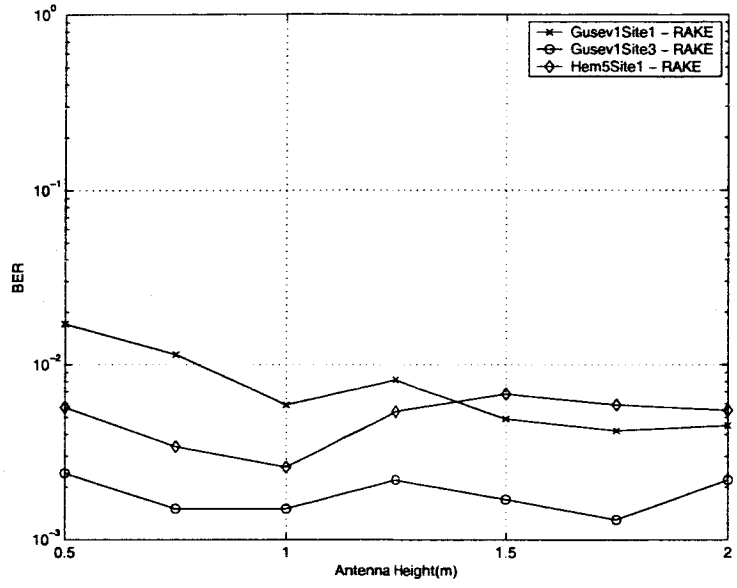


Figure 4.20: BER Performance for 802.11b at Gusev1 Site1 using a RAKE receiver. The transmit power is 100 μW, and the distance (d) between the transmitter and the receiver is 100 m.

the transmit frequency for 802.11a is in the 5 GHz band while the transmit frequency for 802.11b is in the 2.4 GHz band.

- For shorter distances, the rms delay spread for 802.11a seems to be smaller than for 802.11b in the Gusev sites considered. For larger distances, the rms delay spread for 802.11a increases and becomes similar to or larger than the rms delay spread for 802.11b. The behavior seems to be just the opposite at the Hematite sites.

- The performance of 802.11a and b is affected by received power and the existence of multipath propagation conditions. When the received power is too small, we can say that the system is operating in the power limited region. An increase in power in the power constrained region improves the performance. On the other hand, when sufficient power is received, the performance of the system can still be severely degraded due to multipath effects. In this case, we can say that the system is in the multipath limited (or equivalently bandwidth limited) region. In the multipath limited region, the performance of the system does not improve with transmission of any additional power.

- The PER for 802.11a is observed to be smaller than the PER for 802.11b in almost all cases for large transmit power (1 W). For smaller transmit power, 802.11b seems to perform better than 802.11a. Note that the received power for 802.11b is larger than for 802.11a as they use 2.4 GHz and 5 GHz frequency bands respectively. This

higher received power greatly helps 802.11b in this power limited region. A meaningful comparison between the two, however, should include the effects of packet sizes, overheads, possible improvement due to RAKE in 802.11b, and implementation complexity considerations as well.

5 Conclusions

We have continued the effort begun under Year 1 of the grant to begin applying the technology developed for propagation modelling on the surface of Mars using commercial software products and available map data sets. The effort in Year 2 had two major phases: validation of simulation techniques and IEEE 802.11a/b physical layer modelling. The major accomplishments for the second year of this research are as follows:

1. Accomplishment 1. Using the Year 1 results regarding antenna coverage patterns, maximum link distances, effects of surface clutter, and multipath effects, we have simulated the physical layers of IEEE 802.11a and IEEE 802.11b wireless networking standards in the Martian environment. The simulations were conducted in MATLAB using the CommAccess Technologies' mWLAN toolbox. These results will be used as the basis to begin the study of the behavior of the medium access layer for these protocols in the simulated Martian environment.

2. Accomplishment 2. We performed a detailed validation study of the results of the computer-based techniques for estimating the physical layer propagation effects against those measured in the field. In this validation study, we measured key parameters (RF signal strength, delay spread, data rates, and packet error rates) for an outdoor IEEE 802.11b wireless network. These measurements were conducted at sites near Tortugas Mountain and Dripping Springs due their similarity to the Mars surface (free of man-made objects, little vegetation, mostly flat with some terrain variation and rocks, etc.). These measurements were compared with expected performance based upon the DEMs for the sites. We judge the agreement to be very good, especially considering the wide variation in measurements that can occur by moving the transmitter or receiver antenna by a small distance.

From this study effort, we conclude that:

- The results of the physical layer simulations for the Martian surface show that successful IEEE 802.11a/b-based communications are possible within a few hundred meters of the transmit antenna when the transmit power is more than a few milliwatts and the antenna heights are fixed at more than 1 meter above the ground. The packet error rate performance of 802.11b without a RAKE receiver seems to be more adversely affected by the multipath conditions than 802.11a. Further, the lowest data rate mode of 802.11a provides the best bit error performance.

- Using higher power in the communications system does not always help the performance of the system. This result is known for terrestrial environments where vegetation and atmospheric phenomena are important. It is also true in sparse environments with negligible atmospheric attenuation as well.

- Transmission power and antenna height can be traded to a certain extent. The desired link Quality of Service and data rate may be more of a driver in link design and performance than transmission power and antenna height.

- Use of a RAKE-type of receiver can significantly improve performance with 802.11b protocols.

6 Recommendations

Given our research results, we propose the following recommendations:

1. *Continue investigation of the technology to employ the MATLAB physical layer simulations with the OPNET MAC layer simulations.* The incorporation of MATLAB with OPNET to produce a cosimulation is proving to be a difficult but necessary task. However, recent efforts have shown that there appears to be a path to resolving the compatibility issues. Once these issues are resolved, then further progress in seeing the effects of the physical channel on MAC layer performance will follow.

2. *Investigate MAC layer improvements to make outdoor wireless communications more reliable.* The 802.11 protocol suite has a rich set of features that can be investigated for how they can be improved to make the communications more reliable. Most of these features were initially designed for indoor wireless use and not the outdoor environment. Now that we can predict the propagation effects, both link budget and multipath, with a degree of reliability, we have the means to investigate which protocol features are most important for the 802.11 wireless environment.

3. *Investigate the potential for other protocol families to be used in the planetary surface environment.* For example, it is expected that the European Space Agency will consider using the 802.16/large subcarrier number OFDM as the baseline protocol surface communications in their mission studies. With the technology baseline being developed here, we can assess if this protocol suite would make sense when the various propagation factors are considered. The performance can then be benchmarked against the 802.11 simulations for evaluation purposes.

References

[1] "NASA Space Communications Project," Mar. 2004. <http://scp.grc.nasa.gov/portfolio/pn/index.html>.

[2] J. Andersen, T. Rappaport, and S. Yoshida, "Propagation Measurements and Models for Wireless Communications Channels," *IEEE Communications Mag.*, pp. 42–49, Jan. 1995.

[3] P. DeLeon and S. Horan, "Effective utilization of commercial wireless networking technology in planetary environments," Tech. Rep. NMSU-ECE-04-003, New Mexico State University, Las Cruces, NM, March 2004.

[4] D. Hansen, M. Sue, C. Ho, M. Connally, T. Peng, R. Cesarone, and W. Horne, "Frequency Bands for Mars In-Situ Communications," *Proc. IEEE Aerospace Conf.*, 2001.

[5] C. S. R. Murthy and B. S. Manoj, *Ad Hoc Wireless Networks Architectures and Protocols.* Upper Saddle River, NJ: Pearson Education, Inc., 2004. Chap. 2.

[6] *IEEE Std 802.11b-1999 Supplement to ANSI/IEEE Std 802.11, 1999 Edition - Information technology Telecommunications and information exchange between systems Local and metropolitan area networks Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band.* New York, NY: Institute of Electrical and Electronics Engineers, 1999.

[7] C. Steger, P. Radosavljevic, and J. P. Frantz, "Performance of IEEE 802.11b wireless LAN in an emulated mobile channel," in *Proc. IEEE VTC*, (Korea), April 2003.

[8] M. V. Clark, K. K. Leung, B. McNair, and Z. Kostic, "Outdoor IEEE 802.11 cellular networks: Radio link performance," in *Proc. IEEE ICC*, 2002.

[9] *IEEE Std 802.11a-1999 (ISO/IEC 8802-11:1999/Amd 1:2000(E)) Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications - Amendment 1: High-speed Physical Layer in the 5 GHz Band.* New York, NY: Institute of Electrical and Electronics Engineers, 1999.

[10] D. K. Borah, R. Jana, and A. Stamoulis, "Performance evaluation of IEEE 802.11a wireless LANs in the presence of ultra-wideband interference," in *Proc. IEEE Wireless Communications and Networking Conf., WCNC*, (New Orleans), March 2003.

[11] P. K. A. N. A. Doufexi, S. Armour and D. Bull, "A Comparison of HIPERLAN/2 and IEEE 802.11a," *IEEE Commun. Magazine*, vol. 40, pp. 172 – 180, May 2002.

[12] V. Chukkala, P. D. Leon, S. Horan, and V. Velusamy, "Modeling the radio frequency environment of mars for future wireless, networked rovers and sensor webs," in *Proc. IEEE Aerospace Conference*, (Big Sky, MT), 2004.

[13] B. Pearson, "A Condensed Review of Spread Spectrum Techniques for ISM Band Systems." Intersil Corporation, May 2000. Application Note 9820.

[14] "WLAN Design Guide." Agilent Technologies, May 2003.

[15] "ATDI," Mar. 2004. <http://www.atdi.com>.

[16] NASA, "NASA Rovers Slated To Examine Two Intriguing Sites On Mars." NASA News Release 03-137, Apr. 2003. <http://www.nasa.gov/home/hqnews/2003/mar/HP_news_03137.html>.

[17] A. Rayl, "NASA Announces Mars Exploration Rovers' Landing Sites," Apr. 2003. <http://www.planetary.org/html/news/articlearchive/headlines/2003/nasa_mer_sites.htm>.

[18] C. Ho, S. Slobin, M. Sue, and E. Njoku, "Mars Background Noise Temperatures Received by Spacecraft Antennas," *The Interplanetary Network Progress Report*, vol. 42-149, May 2002.

[19] P. McKenna. Private communication, Jul. 2003.

[20] OPNET, *Wireless Module User Guide for Modeler.*

[21] OPNET, *Tutorial: Basic Processes.*

[22] OPNET, *Modeling Concepts*, ch. External System Domain.

[23] Mathworks, *External Interfaces.* Matlab user guide.

[24] V. Dham, "Link establishment in ad hoc networks using smart antennas," Master's thesis, Virginia Polytechnic Institute and State University, January 2003.

[25] "WindowsNetworking," Aug. 2004. <http://windowsnetworking.com/articles_tutorials/trouble/>.

[26] "Ad-Aware," Aug. 2004. <http://www.lavasoft.com/>.

[27] "Intel," Aug. 2004. <http://support.intel.com/support/wireless/wlan/sb/CS-006205.htm>.

[28] "Iperf-users," Sep. 2004. <http://archive.ncsa.uiuc.edu/lists/iperf-users/may03/msg00000.html>.

[29] "BVS," Feb. 2004. <http://www.bvsystems.com/>.

[30] "Atheros," Aug. 2004. <http://atheros.com/pt/papers.html>.

[31] "EIRP Limitations for 802.11 WLANs," Oct. 2004. <http://www.wi-fiplanet.com/tutorials/article.php/1428941>.

[32] "CommAccess," Jan. 2004. <http://www.commaccess.com>.

[33] T. H. Lee, H. Samavati, and H. R. Rategh, "5-GHz CMOS Wireless LANs," *IEEE Trans. on Microwave Theory and Techniques*, vol. 50, pp. 268–279, Jan. 2002.

[34] D. M. Pozar, *Microwave and RF Wireless Systems.* New York: John Wiley & Sons, 2001.

A Research Personnel

Principal Investigators
Dr. Phillip L. DeLeon
Associate Professor
New Mexico State University
Klipsch School of Electrical and Computer Engineering
Las Cruces, New Mexico 88003-8001
(505) 646-3771
pdeleon@nmsu.edu

Dr. Stephen Horan
Professor
New Mexico State University
Klipsch School of Electrical and Computer Engineering
Las Cruces, New Mexico 88003-8001
(505) 646-4856
shoran@nmsu.edu

Co-Investigators
Dr. Deva Borah
Assistant Professor
New Mexico State University
Klipsch School of Electrical and Computer Engineering
Las Cruces, New Mexico 88003-8001
(505) 646-3357
dborah@nmsu.edu

Dr. Raphael Lyman
Assistant Professor
New Mexico State University
Klipsch School of Electrical and Computer Engineering
Las Cruces, New Mexico 88003-8001
(505) 646-3811
rlyman@nmsu.edu

Senior Personnel
Mr. Anirudh Daga
Research Assistant
New Mexico State University
Klipsch School of Electrical and Computer Engineering
Las Cruces, New Mexico 88003-8001

(505) 646-1911
anirudh@nmsu.edu

Mr. Robert Hull
Technical Support
New Mexico State University
Klipsch School of Electrical and Computer Engineering
Las Cruces, New Mexico 88003-8001
(505) 646-1556
rhull@nmsu.edu

Mr. Gaylon Lovelace
Research Assistant
New Mexico State University
Klipsch School of Electrical and Computer Engineering
Las Cruces, New Mexico 88003-8001
(505) 646-1911
glovelac@nmsu.edu

B Publications Resulting from Research

The following publications resulted from this research effort. Many of the results of these theses have been included in this report.

V. Chukkala, P. DeLeon, S. Horan, and V. Velusamy, "Radio Frequency Channel Modeling for Proximity Networks on the Martian Surface," *International Journal of Computer and Telecommunications, in press*, 2005.

V. Chukkula, P. DeLeon, S. Horan, and V. Velusamy, "Modeling the Radio Frequency Environment of Mars for Future Wireless, Networked Rovers and Sensor Webs," in *Proc. 2004 IEEE Aerospace Conf.*, Big Sky, MT., March 2004.

D. Borah, A. Daga, G. Lovelace and P. DeLeon, Performance Evaluation of the IEEE 802.11a and b WLAN Physical Layer on the Martian Surface , in *Proc. 2005 IEEE Aerospace Conf.*, Big Sky, MT., March 2005.

V. Chukkula and P. DeLeon, Simulation and Analysis of the Multipath Environment of Mars, in *Proc. 2005 IEEE Aerospace Conf.*, Big Sky, MT., March 2005.

C Steps for Generating Simulations

C.1 RF Coverage Simulations using HerTZ Mapper

Initial Setup

To create a project in HerTZ Mapper you must have three files available

1. .geo: This file contains the topographic information
2. .img: This is a image file corresponding to the topographic map
3. .pal: This file contains the palette information of the image

Creating a Project

From the Windows *Start menu*, navigate to the *ATDI* software selection within Programs, and choose HerTZ Mapper. Once the package has opened, click on the image on the middle of the screen. From the File menu, you may now choose *Create New File Server* to create a new project, or *Coverage and profile analysis* to open an existing one. The procedure is explained below. The File server window is shown in Fig. C.1.

- Enter a file name for project or select an existing.svr file
- Double-click on Digital Elevation Model, then on Add, and select a .geo file
- Proceed the same way for the Primary Image, and add first the .img file associated to the DEM (then the .pal file)
- "click Open Coverage" & Profile, and choose OK

Transmitter setup

Once you have the files opened in HerTZ Mapper you can click on the DEM or image and select "Add site" to place a transmitter. You can also specify the latitude and longitude of the location to exactly place a Tx at desired location. When you select Add site a Tx parameter window is displayed as one shown in Fig. C.2. The user has to enter all the Tx parameters such as frequency, power and antenna height. Once you choose OK a red square is displayed on DEM and image with a letter on it as shown in Fig. C.3. This red square shows the location of transmitter.

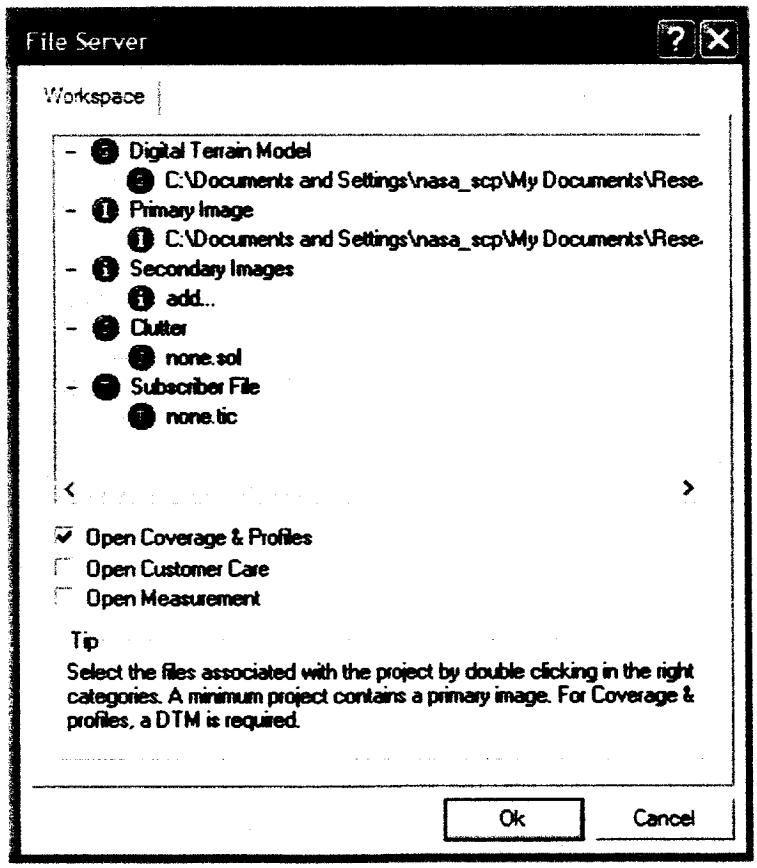


Figure C.1: HerTZ Mapper file server window.

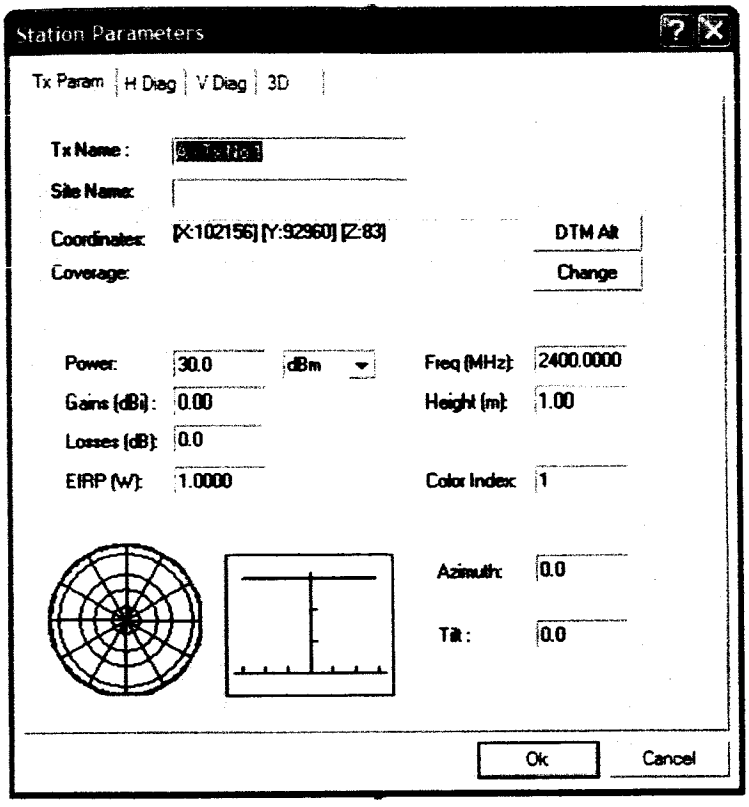


Figure C.2: HerTZ Mapper station parameters.

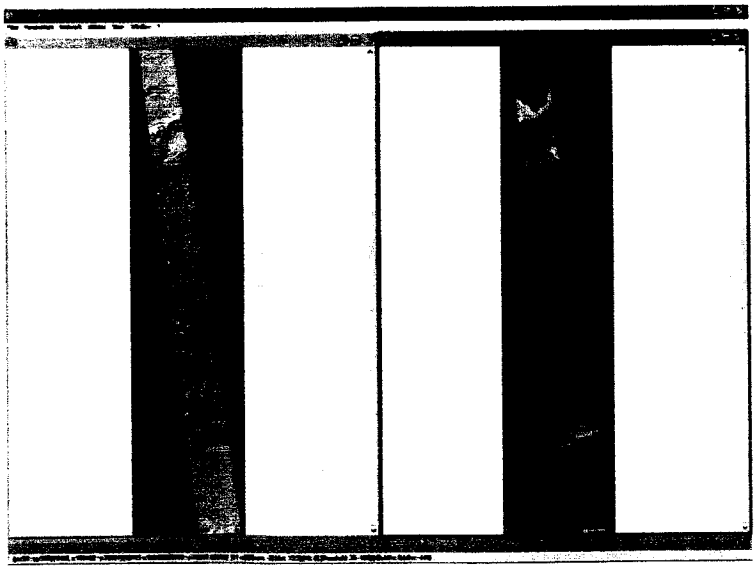


Figure C.3: HerTZ Mapper opening window.

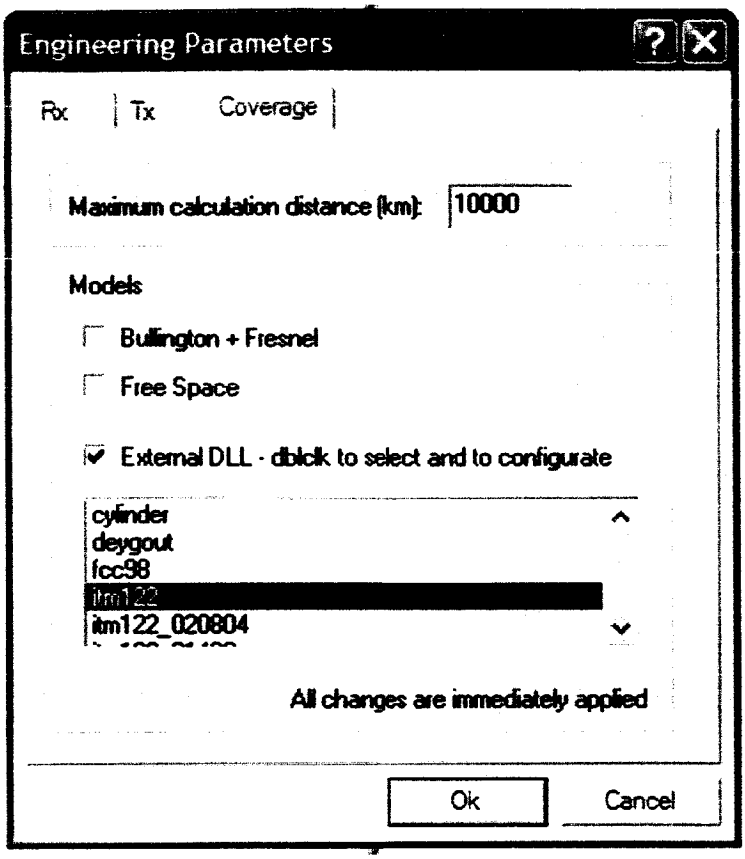


Figure C.4: HerTZ Mapper model selection window.

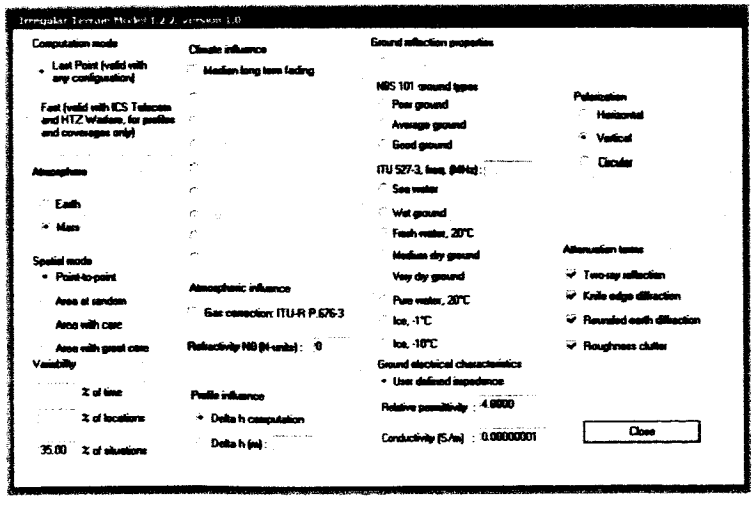


Figure C.5: HerTZ Mapper ITM parameter window.

Propagation Model Setup

To select a propagation model select Parameters form "File Menu" then "Engineering parameters" and then "Coverage". Once you click on the coverage a window is selected with all the propagation models as shown in Fig. C.4. First you have to select "External dll" and then "ITM122". Once you click OK the ITM parameter window is displayed as shown in Fig. C.5. Once you enter all the required values you can close the window to finish the propagation model setup.

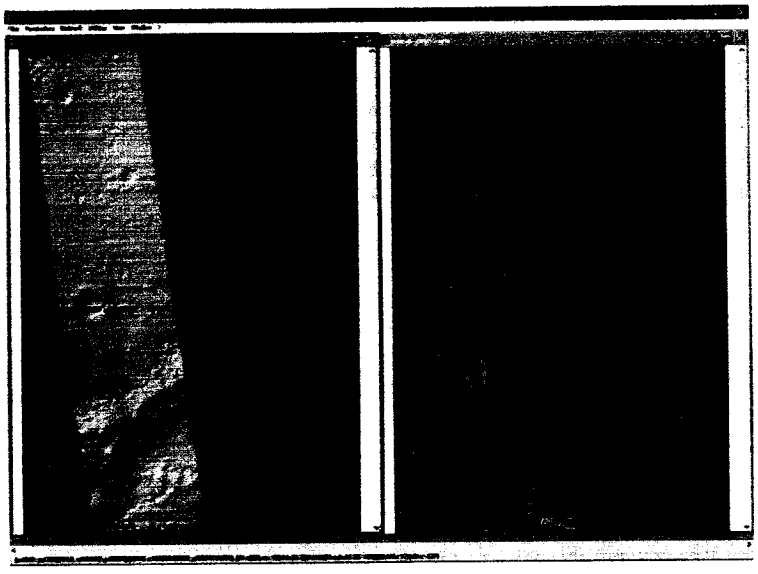


Figure C.6: HerTZ Mapper opening window.

Coverage Calculation and Overlay

First click on the "Transmitter" and then select the Transmitter and then select "coverage" to run a coverage. Clicking on the Coverage option will instruct the software to simulate the site you clicked on, using the parameters and propagation model you have previously set up. The computer performs a simulation. Once finished, you are prompted to save the field strength file. Click "Save" for this file box (it automatically suggests a name). Minimize or close the field strength result window and click on the map. The field strength file can now be displayed in full transparency on top of the image by choosing the "Overlay FLD" option (in the popup box), and selecting the appropriate field file. The color of the field denotes a discrete band of values of field strength. To see the actual value of field, move the mouse cursor over the appropriate area and read the value from the information bar at the bottom of the screen. It appears in the dBV/m bracket. You can modify the colors and field levels in the Parameters/Legend box. Fig. C.6 shows the coverage overlayed on the DEM.

C.2 Power Delay Profile Simulations using ICS Telecom

Initial setup

As mentioned for HerTZ Mapper the ICS Telecom also needs three files (.geo,.img, and .pal) to create a project.

Creating a Project

From the Windows Start menu, navigate to the ATDI software selection within Programs, and choose ICS Telecom. Once the package has opened, From the File menu choose project Manager to create a new project, or to open an existing one.

Once the project Manager window opened click on the 3 dots in front of the DEM, image, and pal options to select the .geo, .img, and .pal file locations. Once all 3 files are selected click save as to create the project file with extension .PJT.

Transmitter setup

Select the point mode from the tool box. Now click on the DEM and then select add Tx/Rx from the popup menu. A parameter window will be displayed. Enter the Tx parameters leaving out the options not applicable for the situation. Once you click OK a square with a letter on it is displayed on DEM to represent the transmitter position.

Propagation Model setup

Select File menu Coverage/Network Calculation/Tx/Rx FS calculation. A window is displayed with Rx antenna height field. Now click on more to get a Advanced coverage parameters window. Click on Model to get Propagation models window. The following three selection have to be made in this window.

In Models box click on 3 dots to select the ITM (tuned dll) from the Models directory within ICS. In climate box enter the corresponding Mars land and sea radius in Km and then click on the 3 dots to select the k factor as 1. In Reflection box enter the reflectance value and then select 3d coverage only option.

PDP calculation

1. Set on the terrain the transmitter network up(Add Tx/Rx option of the point mode left click popup menu).

2. Proceed to the coverage calculation of the network(File menu Coverage/Network calculation/Tx/Rx Fs coverage).

3. Display the composite coverage(File Menu Coverage/Network Analysis/Composite coverage display) and keep the coverage overlaid on the terrain. Keep the Point mode activated.

4. Click over the site of the Virtual receiver In the popup menu, choose option Multipath.

5. In the input box,enter the icon number of the transmitter and the receiver antenna height(in meters).

The simulation is run and will calculate the reflected field strength and the ToA emanating from the designated transmitter to the receiver point touched by the cursor. The result is shown as a graph displayed in the Multipath box. The blue color represents the reflected ray, and the red color indicates the direct ray. In the status bar, the following information regarding the position of the cursor on the graph,is given:

ToA value in μs

Reception threshold(FSR) in dBμV/m

Altitude in meters

The print button opens the Print setup box to print the graph. The close button closes the Multipath box.

D Visualization of Data Acquired with the YellowJacket

D.1 Log File Transfer

The YellowJacket will save measurement data on the iPAQ in a single file named with the .YJ3 extension. The default name is log.yj3. In order to save multiple sessions or measurements, be sure to rename the file and start again, as the YJ will simply append to the existing log file.

The only way to transfer data out of the YJ is to use the infrared port on the top of the iPAQ. This must connect to a corresponding infrared port on a Windows-based PC. The software used for the transfer is Microsoft Active Sync.

1. On the PC, start Microsoft Active Sync, then point the YJ at the ir port of the PC. On the YJ, use the main menu screen to select System, then Communications, then Connect via IR.

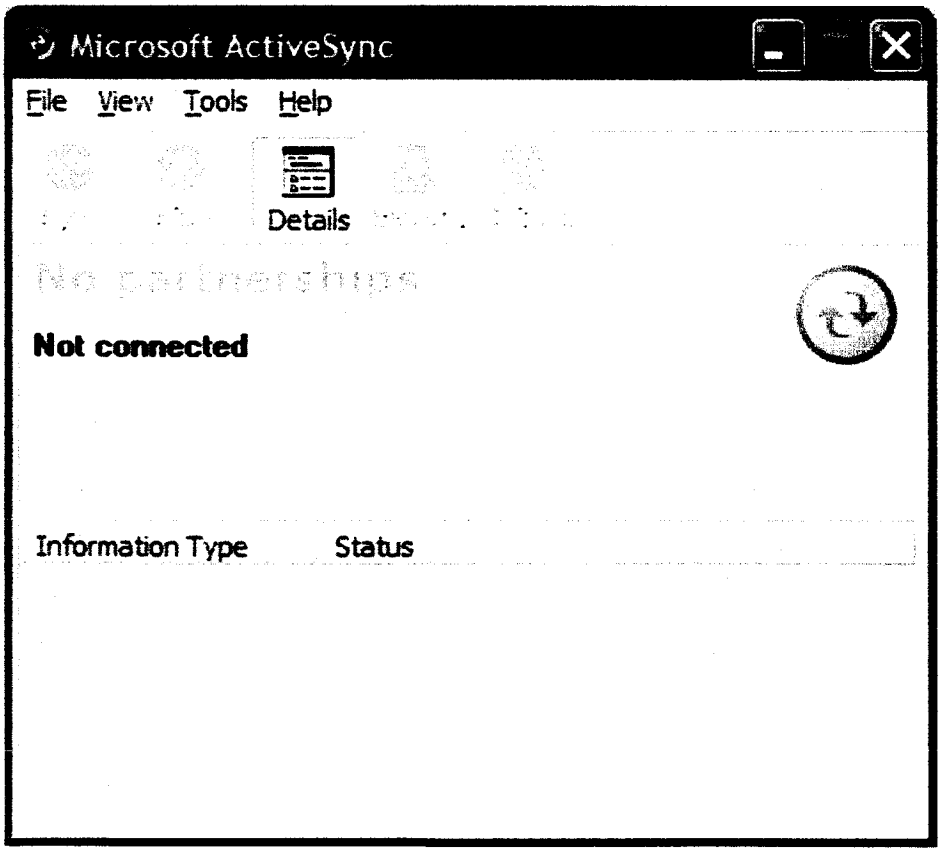


Figure D.7: Snapshot of ActiveSync.

2. When the units "connect", Active Sync will make a noise and ask what type of partnership is desired. This should be a guest connection, not a synched connection.

3. While Active Sync should be able to allow the user to browse the files on the remote unit, it usually crashes. Try the normal "my computer" method to locate the remote unit and look for the log.yj3 files in the home folder. Copy the file(s) to the desktop or appropriate folder.

4. Now that the file has been copied (with translations from PDA to desktop format) to the PC, it must be translated from YJ format to columnar. Use Berkeley Varitronics Systems (BVS) Chameleon software for this.

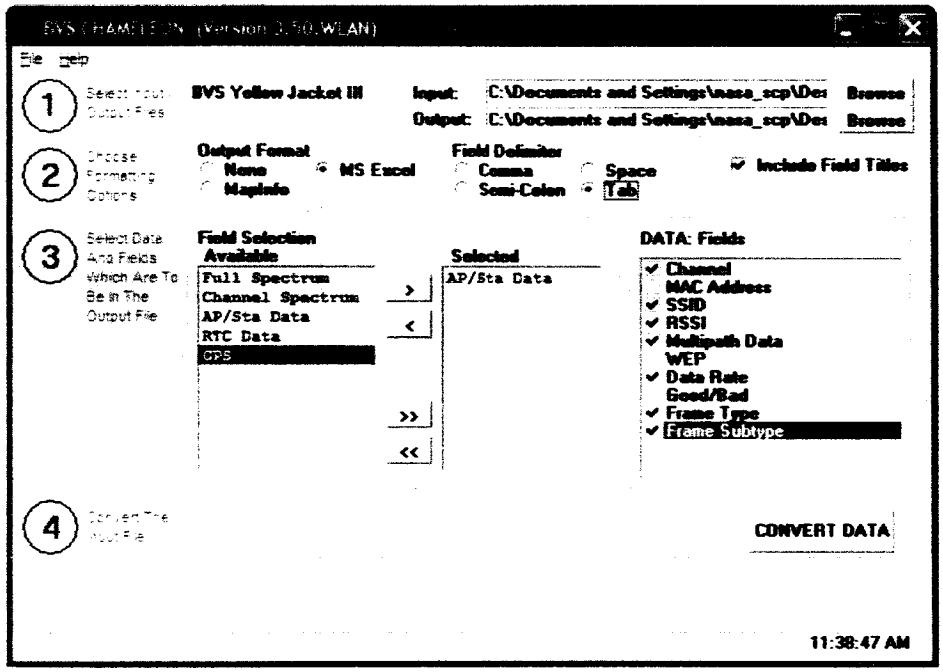


Figure D.8: Snapshot of Chameleon.

5. In Chameleon, follow the steps numbered to the left of the Chameleon screen:

   (a) Browse for the input file. The default screen will search for the .yj3 extension (extensions are not always visible in Windows machines) and the file type: BVS YellowJacket III Log Files.

   (b) Browse for the output file (create a file name) with the default extension .out

   (c) Select Output Format: MS Excel; select Field Delimiter: Tab

   (d) Choose the data source from the "Field Selection Available." The YJ stores the data from the screens in these categories. The most useful are the AP/Sta Data (Access Point station data) and GPS.

(e) Once a source is selected (by placing the selection is the Selected box using the arrows), the DATA: Fields become available. Here, the individual selections are made and will include all sequential samples of the various measurements. Always select a Channel and/or SSID to be sure which station is being measured. The Multipath Data are the first twenty two measurements from the Correlation screen, from which the Delay Spread can be derived.

(f) Use Microsoft Excel to view the Chameleon XXX.out file and put the data in spreadsheet format. The Excel wizard will assist in translating the layout. The file can be saved as XXX.xls and opened in MATLAB (see next section regarding out YellowJacket Toolbox User's Guide). For viewing in Excel, some adjustment of column width will be required.

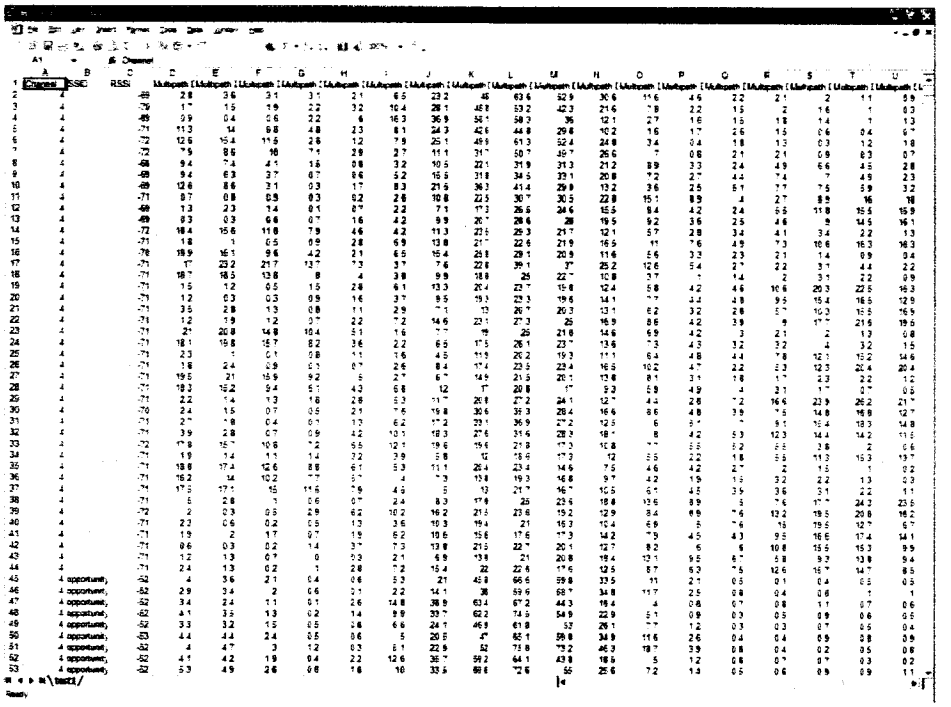


Figure D.9: Snapshot of Excel.

D.2 MATLAB YellowJacket Toolbox User's Guide

Open MATLAB by double clicking on the MATLAB icon as in Fig. D.10.

As in Fig. fig:command, type at command prompt

```
>> YJ_program
```

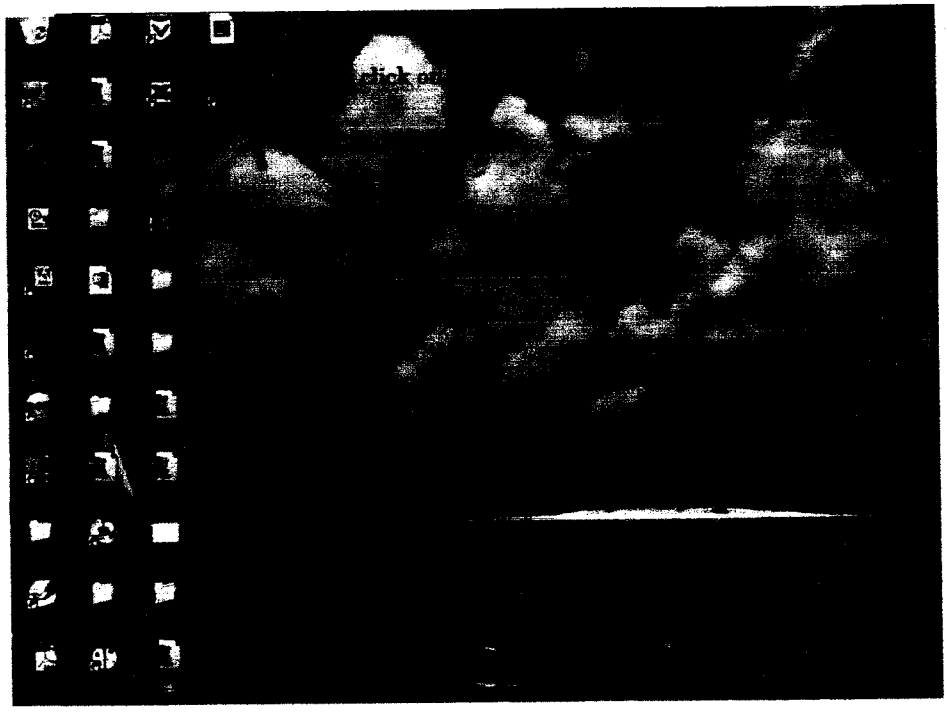


Figure D.10: Snapshot of Windows desktop.

Select YellowJacket data folder by date and antenna height. Then choose which Excel file by distance from AP. See Fig. D.12.

Now the program is running as in Fig. D.13. NOTE: To view Iperf data for location

1. Select file Iperf

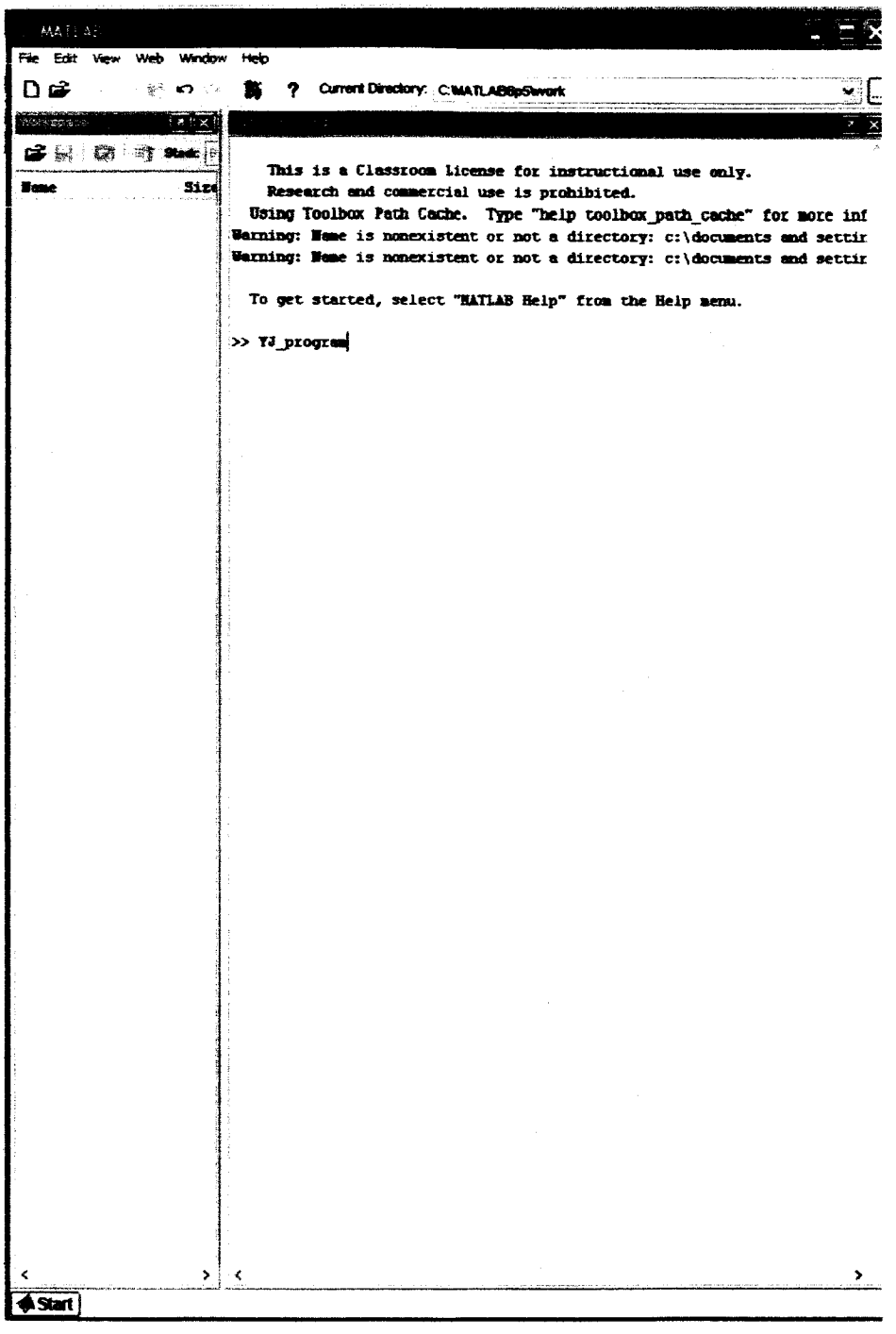


Figure D.11: MATLAB's command window.

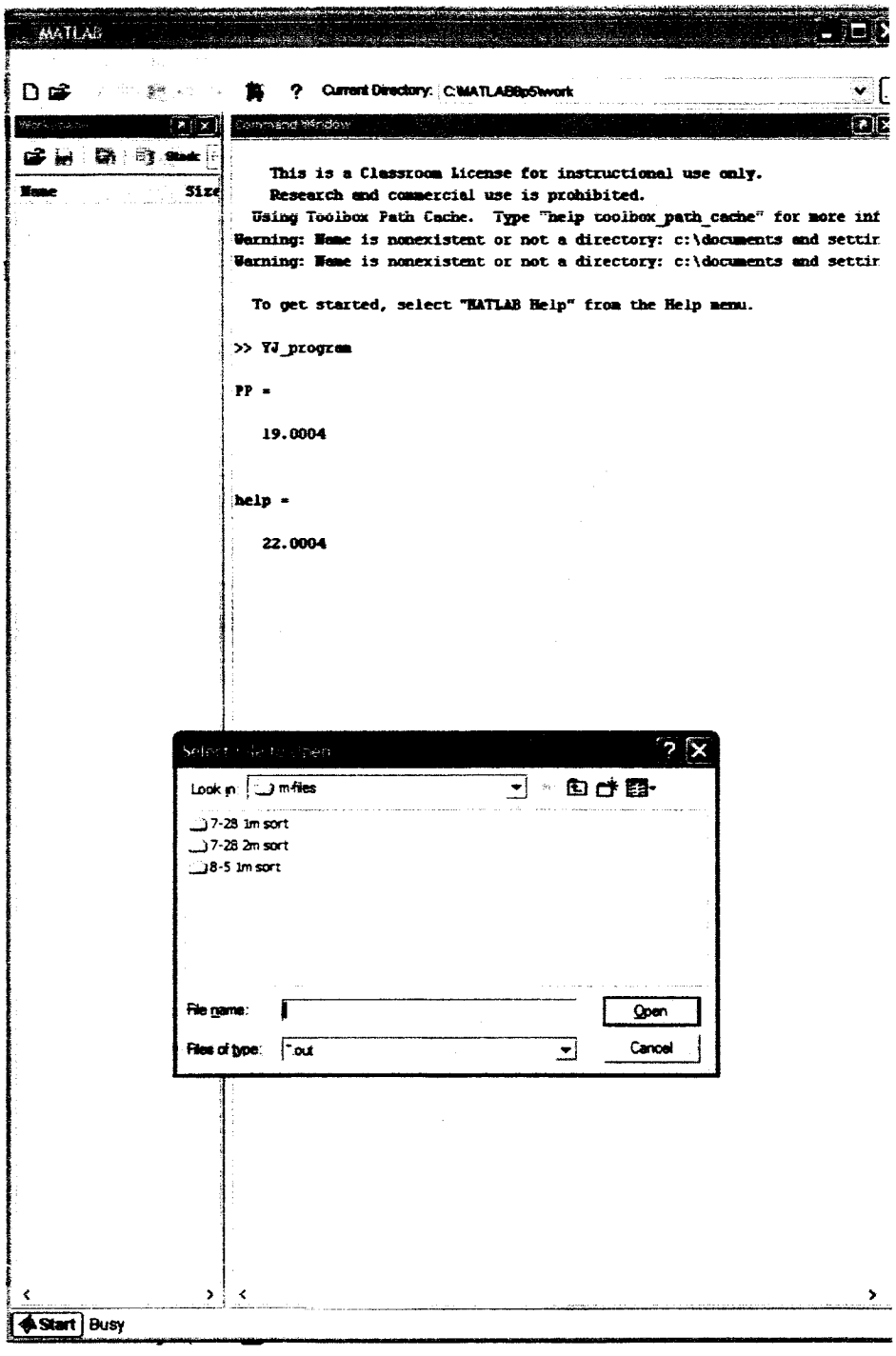


Figure D.12: YellowJacket toolbox program running.

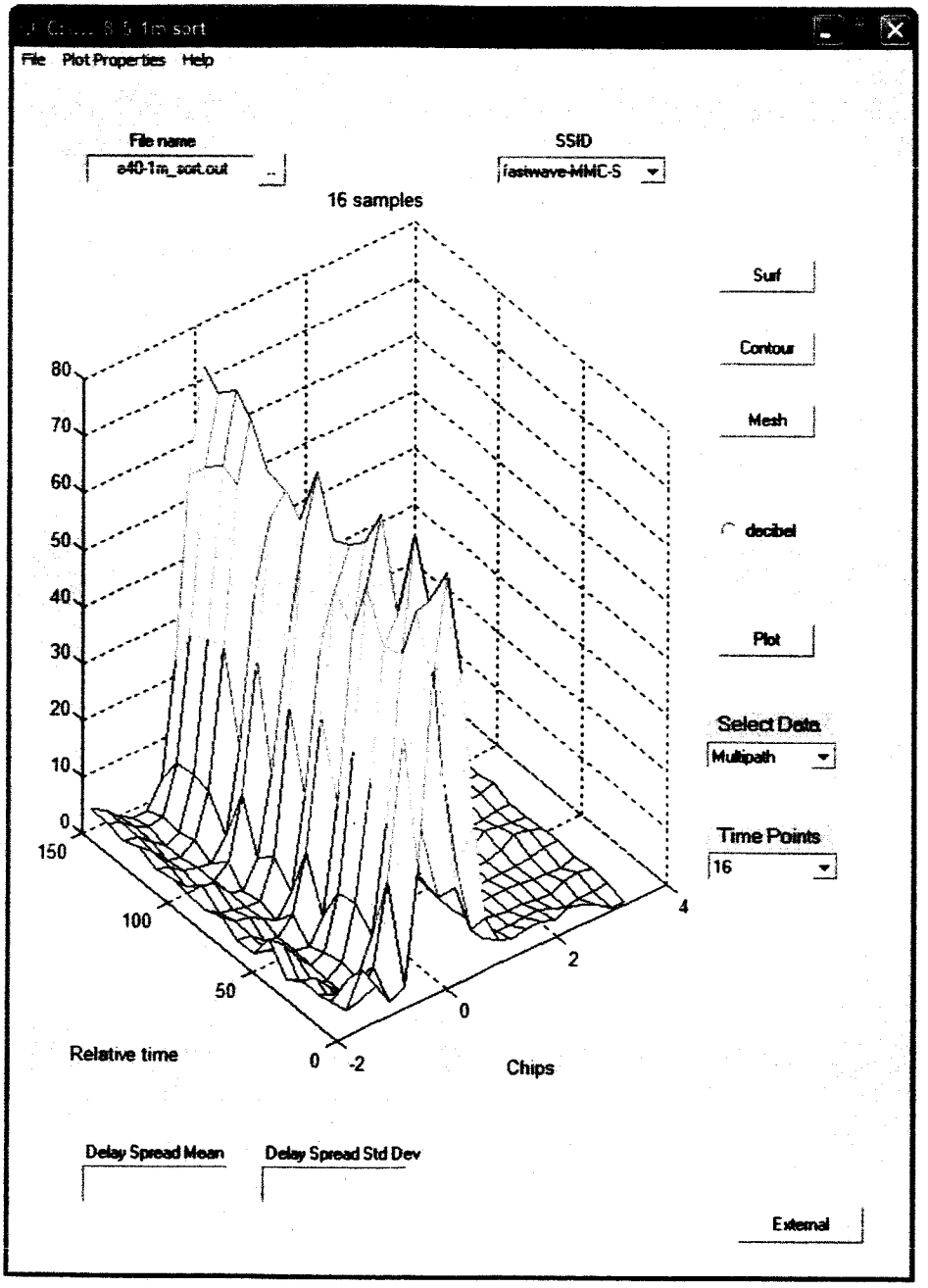


Figure D.13: YellowJacket toolbox program running.

E IEEE 802.11b Field Measurement Test Procedure

E.1 Introduction - Free Field and Simple Reflector Tests

Purpose: To verify IEEE 802.11b WLAN hardware and RF performance. We intend to discover the free field distance that can support data transfer between commercial access points (AP) and standard laptop computer transceivers (built in). In addition, a simple RADAR reflection configuration will be measured to determine the effect of multipath signals on the performance of an 802.11b system.

E.2 Equipment

D-Link DI-524 wireless router / access point (AP)
SSID = "spirit" or "opportunity"

Apple Macintosh PowerBook G4 laptop
SSID = "isidis" or "dsp"

Dell Latitude D600 laptop
SSID = "gusev"

YellowJacket Plus (with GPS) - also called "YJ"
802.11b W-LAN Analysis System
Berkeley Varitronics Systems
s/n 024127

Cushcraft S24012P directional panel antenna(s)

Garmin rino120 FRS radios / GPS
Garmin eTrex GPS receiver

APC SmartUPS 700
uninterruptible power supply (battery powered AC source)
and / or
Xantrex xPower 600
DC-AC power source

software:
"Iperf" network testing utility

E.3 Procedure

Setup

1. Connect equipment to power source as required:

   laptop "isidis"

   D-Link AP

   YJ charger cradle, if needed

2. Establish appropriate connections to AP:

   ethernet cable between Iperf "client" and AP

   RF link between Iperf "server" and AP

   open "SSH Secure Shell Client" window on server to remotely control client: password as listed

   run Iperf as client, -c

3. Verify YJ is acquiring AP by channel and SSID

   under YJ options, clear log file and then enable logging

Iperf data test

1. Setup Iperf server and client UDP command lines:

   (server) iperf -s -u -i 1 xxxlog.txt

   where xxx is log file descriptor (distance, elevation) such as "s40-1m"

   (client) iperf -c 10.0.0.xx -u -b 7.35m -t 120

   where .xx is the wireless IP address of the server computer

   determine clean data rate and substitute for 7.35m in above command line

2. While monitoring distance on the GPS, move away from the AP, watching the data quality on Iperf.

3. Take data at 20 meter increments to limit of system function and data quality. See Figure E.14 for positions.

4. Rename the log file on the YJ, labelling each measurement station

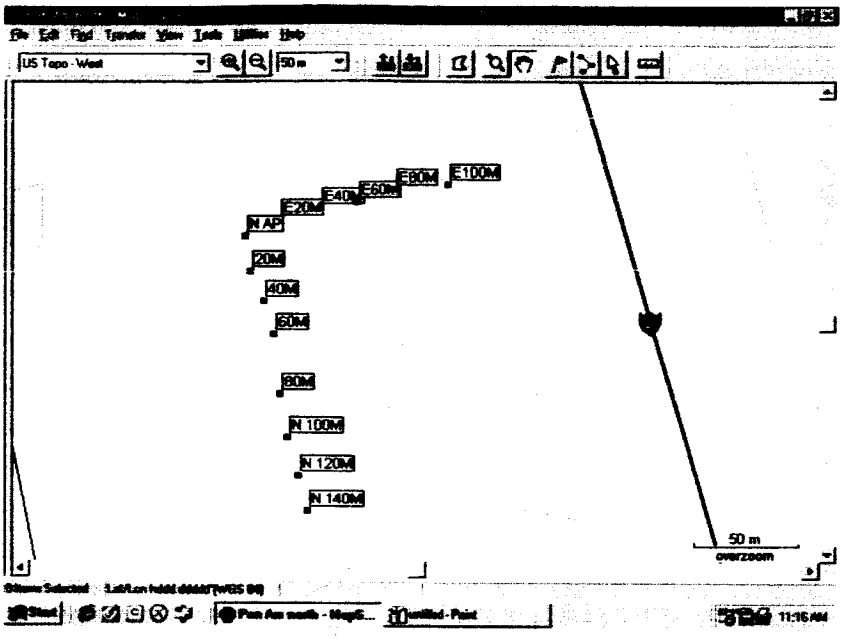


Figure E.14: Wireless node positions at stadium parking lot.

Reflection test

Establish the geometric layout depicted in Figure E.15.

1. Verify wireless direct link between server and client.
2. With YJ at far corner of layout triangle, look for double correlation peak by manipulating orientation or antenna angles.
3. Name the log file on the YJ
4. With double peak showing on YJ, attempt Iperf data run and log results.

E.4 Introduction - Multipath Tests

Purpose: In a terrestrial environment that features massive rock structures capable of causing multipath reflections, determine the signal characteristics and data link effects due to the terrain.

E.5 Equipment

D-Link model DI-524 wireless router / access point (AP)
SSID = "spirit" or "opportunity"

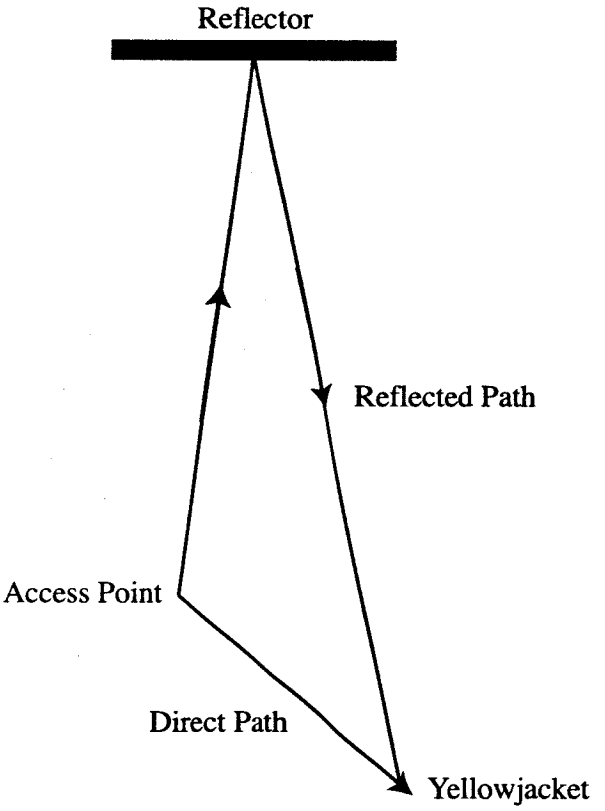


Figure E.15: Outdoor geometry.

Linksys Bridge model WET11
SSID = "opportunity" (WLAN AP)

Apple Macintosh PowerBook G4 laptop
SSID = "isidis"

Dell Latitude D600 laptop
SSID = "gusev"

YellowJacket Plus (with GPS) - also called "YJ"
802.11b W-LAN Analysis System
Berkeley Varitronics Systems
s/n 024127

Cushcraft S24012P directional panel antenna(s)

Garmin rino120 FRS radios / GPS
Garmin eTrex GPS receiver

APC SmartUPS 700
uninterruptible power supply (battery powered AC source)
Xantrex xPower 600
DC-AC power source

software:
"Iperf" network testing utility
"Ping" network connection utility

E.6 Procedure

Setup

1. Connect equipment to power source as required:

   laptop "isidis"

   D-Link AP

   YJ charger cradle, if needed

2. Verify YJ is acquiring AP by channel and SSID

   under YJ options, clear log file and then enable logging

Wireless data test

1. Establish a wireless connection with omnidirectional antennas, using Ping as an example data test (see Figure E.16).
2. Raise antenna to two meter height and evaluate data improvement, if any.
3. For sufficient signal using 50mW AP, connect directional antenna(s) and retest.
4. Find optimal orientation to establish multipath peaks on YJ display, using rock face as reflector. Mark RX spot as GPS waypoint.
5. Name log file on YJ.

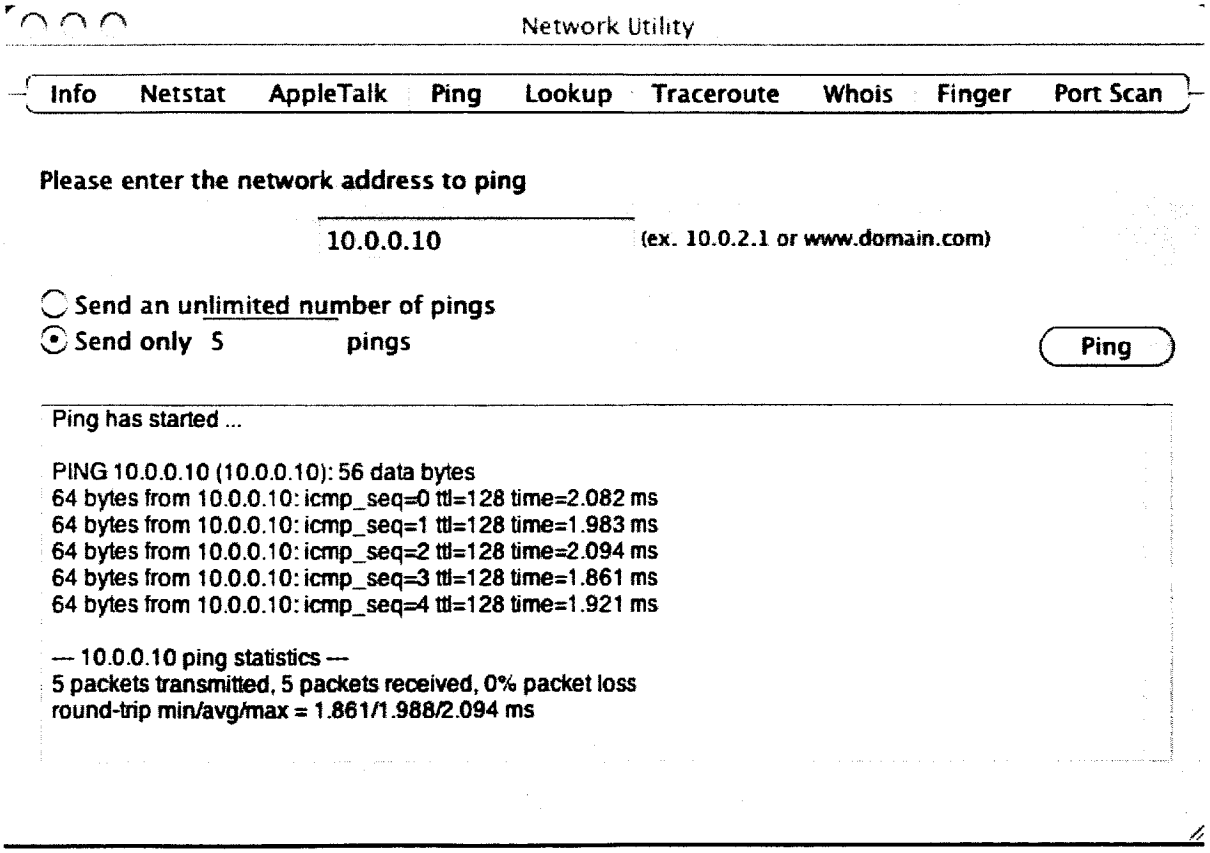


Figure E.16: Sample output from ping test.

Iperf Multipath test

1. Setup UPS power to run laptop and Bridge at RX location.

2. Establish appropriate connections to AP:

   ethernet cable between Iperf "client" and AP at TX location.

   RF link between Iperf "server" and AP using Linksys Bridge as intermediate element (to enable use of directional antenna) at RX location.

3. Carefully, without moving RX antenna position, connect antenna to Bridge in place of YellowJacket.

4. Run Ping to verify wireless link.

5. Setup Iperf server and client UDP command lines:

   (server) iperf -s -u -i 1 xxxlog.txt

   where xxx is log file descriptor.

   (client) iperf -c 20.0.0.xx -u -b 5.5m -t 60

   where .xx is the wireless IP address of the server computer

6. Perform sufficient Iperf runs to establish link behavior under multipath conditions.

F Code Listings

This Appendix lists the major MATLAB code files used in the analysis and simulation work.

Yellow Jacket Code

ask_npts.m

```
1   function answer = ask_npts(handles)
2
3   default = num2str(length(handles.current_data)*.1);
4   prompt = {['How many points(' num2str(length(handles.current_data)) '):']};
5   dlg_title = 'Moving Average filter';
6   num_lines= 1;
7   def = {default}
8   answer  = str2num(char(inputdlg(prompt, ...
9       dlg_title,num_lines,def)));
10  if isempty(answer)
11      answer = .1*length(handles.current_data);
12  end
```

BW_filter.m

```
1   function BW = BW_filter(BW0)
2
3   for i = 1:length(BW0)
4
5       if BW0(i) < 7.5
6           BW(i) = BW0(i);
7       end
8   end
```

---

cell_find.m

```
1   function [labels, index, in_ref] = cell_find(s, char_array)
2   %This program find a string in a cell array and return the index
3
4
5   %==================================
6   %This is not correct
7   %The Data Rate has a extra tab charactar
8   %==================================
```

```
 9
10  [R C]    = size(s);              %define search criteria
11  [Rc Cc] = size(char_array);
12  index = [];                 %Set intial index
13  in_ref = [];
14  ind_end = 1;
15
16  for j = 1:Cc
17
18      for i = 1:C
19
20          x = double(char(s{1,i}));
21
22          if x(1) == 9
23              labels{i} = {char(x(2:length(x)))};
24          else
25              labels{i} = s{1,i};
26          end
27
28
29
30          %If desired string of given headers
31          if length(char(labels{i})) == length(char(char_array(j))) ...
32                          & char(labels{i}) == char(char_array(j))
33
34              %If consecutive headers are the same
35              if i~=1 & length(char(labels{i})) == length(char(labels{i-1})) ...
36                      & char(labels{i}) == char(labels{i-1})
37                  if ind_end == 1;    %describe when consecutive headers began
38                      ind_end = 0;
39                      initial = i-1;
40                      len = length(in_ref);
41                  end
42              else
43                  in_ref{length(in_ref)+1} = [{i} labels{i}];
44              end
45              index(length(index)+1) = i;
46          elseif i >3 & length(char(labels{i-1})) == length(char(labels{i-2})) ...
47                      & char(labels{i-1}) == char(labels{i-2})
48                  if ind_end == 0
49                      in_ref{len} = [{initial} {i-1} labels{i-1}];
50                      ind_end = 1;
51                  end
52          end
53
```

```
54      end%char_array search
55  end%header search
56
57
```

---

cellarray.m

```
1   function names = cellarray(ssid)
2
3   len = length(ssid);
4
5   for i = 1:len
6       names{i,1} = char(ssid{i}{1}{1});
7   end
```

---

col_find.m

```
1   function handles = col_find(handles)
2   %Called from YJ_work1 on line 96
3   %This program read the colums specified in handles.ref
4
5       row = [2 length(handles.data)];%Length of data
6   %Columns-----------
7   %SSID
8       temp = handles.ref{1}{1};
9       handles.col_ssid = temp;
10  %Multipath start
11      temp = handles.ref{2}(1);
12      handles.col_in = temp{1}(1);
13  %Multipath finish
14      temp = handles.ref{2}(2);
15      handles.col_fn = temp{1}(1);
16  %Data Rate
17      temp = handles.ref{3}{1};
18      handles.DR = data_fetch(row,temp,handles.data);
19  %RSSI
20      temp = handles.ref{4}{1};
21      handles.rssi = data_fetch(row,temp,handles.data);
22
23
24      if length(handles.ref)  > 4
```

```
25  %Latitude
26          temp = handles.ref{5}{1};
27          handles.latitude = data_fetch(row,temp,handles.data);
28      end
29      if length(handles.ref) > 5
30  %Longitude
31          temp = handles.ref{6}{1};
32          handles.longitude = data_fetch(row,temp,handles.data);
33      end
34      if length(handles.ref) > 6
35          %Altitude
36          temp = handles.ref{7}{1};
37          handles.Altitude = data_fetch(row,temp,handles.data);
38      end
```

data_fetch.m

```
1   function d = data_fetch(row,col,data)
2
3   [r c] = size(col);
4   %This is for Multiple column data
5   if c >1
6
7       for i = row(1):row(2)
8           for j = col(1):col(2)
9               d(i-row(1)+1,j) = str2num(char(data{i,j}));
10          end
11      end
12  else    %this is for single column data
13      if isempty(str2num(char(data{row(1),col})))
14
15          for i = row(1):row(2)
16              y = char(data{i,col});
17              d(i-row(1)+1) = str2num(y(1:5));
18          end
19      else
20          for i = row(1):row(2)
21              y = char(data{i,col});
22              d(i-row(1)+1) = str2num(y);
23          end
24      end
25  end
26
```

27

---

delay_spread.m

```
1   function d = delay_spread(MP)
2
3
4   N = 22;
5   T = 1/22000000;
6   n=[0:N-1];
7   [numrows,numcols] = size(MP);
8   d = zeros(numrows,1);
9
10  for i = 1:numrows
11      m = sum(MP(i,3:24).*n)/sum(MP(i,3:24));
12      d(i) = T*sqrt(sum(MP(i,3:24).*((n-m).^2))/sum(MP(i,3:24)));
13  end
14  d = d*1e9;
```

---

edit_external_plot.m

```
1   load handles
2
3   PD = handles.MP(:,4:25);
4   [r c] = size(PD)
5   start = 1;
6   stop  = r;
7   figure(23);
8
9   % Specify Range for plot
10  % start = 2000
11  % stop  = 2500
12
13
14
15  % Plot Power Delay Profile
16  X = -2:5/c:3-(4/c);
17  Y = start:stop;
18  mesh(X,Y,PD(Y,:))
19  ylabel('Relative Time');
20  xlabel('Chips');
```

```
21  zlabel('Relative Correlation');
22  title('When pointed at La Cueva');
```

for_data.m

```
1   function  lab = for_data(s)
2
3   %Format data
4   % clear;clc;close all
5   % %file name of data
6   % filename = 'log629_ap_mtlb.out';
7   % disp(filename)
8   % s = importdata(filename,'\t');
9   % disp(' ')
10
11  clc;
12  rows = length(s);
13  %tab = s{1}(1,8);
14  load tab_char   %tab character
15  for j = 1:rows
16
17      index = find(char(s{j}) == tab);      %find tab  character
18
19      len = length(index);
20
21      temp= s{j}(1:index(1)-1);       %save data string (initial)
22      lab{j,1} = cellstr(temp(1,:));  %load data string (initial)
23
24      for i = 1:len-1
25          temp = s{j}(index(i)+1:index(i+1)-1);
26          lab{j,i+1} = cellstr(temp);
27      end
28      lab{j,i+2} = cellstr(s{j}(index(i+1):length(s{j}))); %final
29  end
30
31
32
```

general_iperf.m

```
1   function iperf = general_iperf(file,path)
```

```
2  %
3  %           SPACE DELIMITER
4  %
5  %       USE get_iperf_data.m if possible
6  %=================================================================
7  % The program reads data files taken by iperf
8  % and then literally copied to an Excel spreadsheat
9  % Process:
10 %     1) open text file of iperf data
11 %     2) Ctl+a
12 %     3) Ctl+c
13 %     4) In Excel spreadsheet ctl+v
14 %     5) File Save
15 % Now run this program and the iperf data will be stored as followed
16 % all in the iperf handle, iperf.:
17 %     Time in seconds                (iperf.time)
18 %     Transfer in KBytes             (iperf.transfer)
19 %         Bandwidth in Mbit/sec          (iperf.BW)
20 %         Packet Error Rate in percent  (iperf.PER
21 %
22 % =================================================================
23
24 not_all = 0;  %condition if all data if imported properly
25
26 % [file path] = uigetfile('.xls');
27
28 % import data
29         data = importdata([path file]);
30 %Space delimeter
31     sp_del = double(char(' '));
32 %condition varables
33     garbage = 7;gar = 0;
34 %Main loop
35 for i = 8:length(data)
36     %If data is not empty
37     if ~isempty(data{i})
38         %Read only data lines
39         if char(data{i}(1)) == char('[') & char(data{i}(2)) ~= char(' ') %data line
40             sp_ind = find(data{i} == sp_del);   %Save delimeter col location
41             sp_ind = [0 sp_ind];
42
43             for k = 1:length(sp_ind)
44                 if ~(k == 4 & i < 17)
45                     if sp_ind(k+1)-sp_ind(k) > 1  %Test for multiple spaces
46                         start = sp_ind(k)+1;
```

```
47                          stop  = sp_ind(k+1)-1;
48                          row_data{i-garbage,k-gar} = {data{i}(start:stop)};
49                          if (k == 3 &  i < 18) | (k == 2 & i >= 18)
50                              tmp = double(char(row_data{i-garbage,k-gar}));
51                              ind = find(tmp == 45);
52                              if ~isempty(ind)
53                                  row_data{i-garbage,k-gar} =  {char(tmp(1:ind-1))};
54                              end
55
56                          end
57                      else
58                          gar = gar +1;
59                      end
60                  else
61                      gar = gar +1;
62                  end
63
64                  %PER column
65                  if (k+2) > length(sp_ind)
66                      start = sp_ind(k+1)+2;
67                      stop = length(data{i})-2;
68                      if start > length(data{i}(:))
69                          if ~isempty(char(row_data{i-garbage,11}))
70                              per_tmp = [char(row_data{i-garbage,10}) char(row_data{i-garbage
71                              row_data{i-garbage,k-gar+1} = {num2str(eval(per_tmp)*100)};
72                          else
73                              row_data{i-garbage} = {''};
74                          end
75                      elseif data{i}(start) == '-' | (i-garbage) == 1
76                          row_data{i-garbage,k-gar+1} = {'-1'};
77
78                      else
79                          row_data{i-garbage,k-gar+1} = {data{i}(start:stop)};
80                      end
81                      gar = 0;
82                      break
83
84                  end%End of PER column
85              end%End for loop or the row
86          else %If row contain labels
87              garbage = garbage + 1;
88          end
89          %If statement for last row of data (PER column)
90          if (i == length(data))
91              row_data{i-garbage,12} = row_data{i-garbage,11};
```

```
92          end
93      else
94          not_all = 1;
95      end% End of is data cell row empty
96  end% End of row
97
98  % %If there is a WARNING: in the last row of row_data disp and delete
99  if not_all == 0 & length(char(row_data{i-garbage,2})) == length('WARNING:')
100     j = 1;
101     str = [];
102     while ~isempty(row_data{i-garbage,j})
103         str = [str char(row_data{i-garbage,j}) ' '];
104         row_data{i-garbage,j} = {''};
105         j = j+1
106         if length(row_data) > j
107            break
108        end
109     end
110     disp(str);
111 end
112
113 %\\\\\\\\\\\\\\\   Create iperf handle   ///////////////
114 %Time in seconds
115 iperf.time = abs((iperf_cell_array(row_data,2)));
116 %Transfer in KBytes
117 iperf.transfer = iperf_cell_array(row_data,4);
118 %Bandwidth in Mbit/sec
119 iperf.BW = iperf_cell_array(row_data,6);
120 %Packet Error Rate in percent
121 iperf.PER = iperf_cell_array(row_data,12);
122 %Make data same length
123 if length(iperf.time) ~= length(iperf.PER)
124     len = min([length(iperf.PER) length(iperf.time) ...
125         length(iperf.BW)]);
126     iperf.time = iperf.time(1:len);
127     iperf.transfer = iperf.transfer(1:len);
128     iperf.BW = iperf.BW(1:len);
129     iperf.PER = iperf.PER(1:len);
130 end
131
132 if not_all == 1
133     disp('Matlab could not import all of the data');
134
135 end
```

get_iperf_data.m

```
1
2   % This is called from the menu of YJ_program
3
4   [file path] = uigetfile;
5   ip_fi = file;ip_pa = path;
6   if fliplr(file(length(file)-(0:2))) == 'txt'
7       temp = importdata([path file]);
8       time = temp.data(:,1);
9       BW = temp.data(:,3);
10      PER = temp.data(:,7);
11  else
12      temp = general_iperf(file,path);
13      time = temp.time;
14      BW = temp.BW;
15      PER = temp.PER;
16  end
17
18
19      iperf_numeric = 2;  %Conditon for iperf_gui
20
21      save('iperf_num_wksp','time','BW','PER','iperf_numeric', ...
22          'ip_fi', 'ip_pa')
23      run iperf_gui
```

get_iperf_data_igui.m

```
1
2   % This is called from the menu of iperf_gui under:
3   %     User Input > iperf numeric
4
5   [file path] = uigetfile('*.txt','.xls');
6
7   if fliplr(file(length(file)-(0:2))) == 'txt'
8       temp = importdata([path file]);
9       time = temp.data(:,1);
10      BW = temp.data(:,3);
11      PER = temp.data(:,7);
12  else
13      temp = general_iperf(file,path)
```

```
14        time = temp.time;
15        BW = temp.BW;
16        PER = temp.PER;
17    end
18
19    iperf_numeric = 3;  %Conditon for iperf_gui menu
20
21        save('iperf_num_wksp','time','BW','PER','iperf_numeric', ...
22            'file', 'path')
23        run iperf_gui
```

getfolder.m

```
1   function ret = getfolder(path)
2
3
4   ind = find(path == '\');
5
6   ret = [path(1:3) '...'   path(ind(length(ind)-1):length(path)-1)];
```

iperf_cell_array.m

```
1   function ret = iperf_cell_array(data,j)
2
3   len = length(data);
4
5   for i = 1:len
6       if ~isempty(data{i,j}) %| char(data{i,j}{1})
7           x = char(data{i,j}{1});
8
9           % condtion for time
10          k = find(x == '-');
11
12          if isempty(k)
13
14              if length(x) == length('sec')  ...
15                      & x == 'sec'
16                  ret(i,1) = data{i,j+1}; %Transfer column
17              elseif length(x) ==length('KBytes') ...
18                      & x == 'KBytes'
19                  ret(i,1) = data{i,j+1}; %Bandwidth column
```

```
20              elseif j == 12 & str2num(x) > 100
21                  ret(i,1) = data{i,j+1};
22
23              else
24                  ret{i,1} = x;     %If number
25              end
26
27          else
28              if k == 1
29                  ret{i,1} = '0'; % bad data in file
30
31              else
32                  ret{i,1} = x(1:k-1); %bad data import
33              end
34
35
36          end
37          if isempty(ret{i,1})
38              ret{i,1} = 0;
39          end
40
41      end
42  end
43
44  ret = str2num(char(ret));
45  if j ==2 & ret(len) == 0
46      ret = ret(1:len-1)
47  end
48
49
50  % x=  'sec'
51  % if (length(x) == length('sec') | length('KBytes')) & ...
52  %         (x == 'sec' | 'KBytes')
53  % disp('yes');
54  % end
```

Iperf_data.m

```
1   function iperf = Iperf_data(iperf)
2   %
3   %           SPACE DELIMITER
4   %
5   %       USE get_iperf_data.m if possible
```

```
6   %=================================================================
7   % The program reads data files taken by iperf
8   % and then literally copied to an Excel spreadsheat
9   % Process:
10  %     1) open text file of iperf data
11  %     2) Ctl+a
12  %     3) Ctl+c
13  %     4) In Excel spreadsheet ctl+v
14  %     5) File Save
15  % Now run this program and the iperf data will be stored as followed
16  % all in the iperf handle, iperf.:
17  %     Time in seconds                (iperf.time)
18  %     Transfer in KBytes             (iperf.transfer)
19  %         Bandwidth in Mbit/sec          (iperf.BW)
20  %         Packet Error Rate in percent  (iperf.PER
21  %
22  % =================================================================
23
24
25  load iperf_loc          %pa -path and fi -file
26  path = [pa 'iperf/'];
27  ind = find(fi == '-')
28  file = [fi(1:ind-1) '.xls'];
29
30
31  not_all = 0;  %condition if all data if imported properly
32
33  % [file path] = uigetfile('.xls');
34
35  % import data
36          data = importdata([path file]);
37  %Space delimeter
38      sp_del = double(char(' '));
39  %condition varables
40      garbage = 7;gar = 0;
41  %Main loop
42  for i = 8:length(data)
43      %If data is not empty
44      if ~isempty(data{i})
45          %Read only data lines
46          if char(data{i}(1)) == char('[') & char(data{i}(2)) ~= char(' ') %data line
47              sp_ind = find(data{i} == sp_del);   %Save delimeter col location
48              sp_ind = [0 sp_ind];
49
50              for k = 1:length(sp_ind)
```

```
51                  if ~(k == 4 & i < 17)
52                      if sp_ind(k+1)-sp_ind(k) > 1  %Test for multiple spaces
53                          start = sp_ind(k)+1;
54                          stop  = sp_ind(k+1)-1;
55                          row_data{i-garbage,k-gar} = {data{i}(start:stop)};
56                          if (k == 3 &  i < 18) | (k == 2 & i >= 18)
57                              tmp = double(char(row_data{i-garbage,k-gar}));
58                              ind = find(tmp == 45);
59                              if ~isempty(ind)
60                                  row_data{i-garbage,k-gar} =  {char(tmp(1:ind-1))};
61                              end
62
63                          end
64                      else
65                          gar = gar +1;
66                      end
67                  else
68                      gar = gar +1;
69                  end
70
71                  %PER column
72                  if (k+2) > length(sp_ind)
73                      start = sp_ind(k+1)+2;
74                      stop = length(data{i})-2;
75                      if start > length(data{i}(:))
76                          if ~isempty(char(row_data{i-garbage,11}))
77                              per_tmp = [char(row_data{i-garbage,10}) char(row_data{i-garbage
78                              row_data{i-garbage,k-gar+1} = {num2str(eval(per_tmp)*100)};
79                          else
80                              row_data{i-garbage} = {''};
81                          end
82                      elseif data{i}(start) == '-' | (i-garbage) == 1
83                          row_data{i-garbage,k-gar+1} = {'-1'};
84
85                      else
86                          row_data{i-garbage,k-gar+1} = {data{i}(start:stop)};
87                      end
88                      gar = 0;
89                      break
90
91                  end%End of PER column
92              end%End for loop or the row
93          else %If row contain labels
94              garbage = garbage + 1;
95          end
```

```
96          %If statement for last row of data (PER column)
97          if (i == length(data))
98              row_data{i-garbage,12} = row_data{i-garbage,11};
99          end
100     else
101         not_all = 1;
102     end% End of is data cell row empty
103 end% End of row
104
105 % %If there is a WARNING: in the last row of row_data disp and delete
106 if not_all == 0 & length(char(row_data{i-garbage,2})) == length('WARNING:')
107     j = 1;
108     str = [];
109     while ~isempty(row_data{i-garbage,j})
110         str = [str char(row_data{i-garbage,j}) ' '];
111         row_data{i-garbage,j} = {''};
112         j = j+1;
113     end
114     disp(str);
115 end
116
117 %\\\\\\\\\\\\\\   Create iperf handle   ////////////////
118 %Time in seconds
119 iperf.time = abs(round(iperf_cell_array(row_data,2)));
120 %Transfer in KBytes
121 iperf.transfer = iperf_cell_array(row_data,4);
122 %Bandwidth in Mbit/sec
123 iperf.BW = iperf_cell_array(row_data,6);
124 %Packet Error Rate in percent
125 iperf.PER = iperf_cell_array(row_data,12);
126 %Make data same length
127 if length(iperf.time) < length(iperf.PER)
128     len = length(iperf.time);
129     iperf.time = iperf.time(1:len);
130     iperf.transfer = iperf.transfer(1:len);
131     iperf.BW = iperf.BW(1:len);
132 elseif length(iperf.PER) < length(iperf.time)
133     len = length(iperf.PER);
134     iperf.time = iperf.time(1:len);
135     iperf.transfer = iperf.transfer(1:len);
136     iperf.BW = iperf.BW(1:len);
137 end
138
139 if not_all == 1
140     disp('Matlab could not import all of the data');
```

```
141
142   end
143
144   % handles.iperf_figure = figure
145   % set(gcf,'Name',handles.filename);
146   % [haxes,h_per,h_bw] = plotyy(iperf.time, iperf.PER,iperf.time, iperf.BW)
147   %
148   % axes(haxes(1))  %PER parameters
149   % ylabel('Packet Error Rate');
150   % ylim([0 100]);
151   % set(h_per,'Color','r');
152   % axes(haxes(2))  %BW parameters
153   % ylabel('Bandwidth [Mbits/sec]')
154   % ylim([0 max(iperf.BW)+20])
155   % set(h_bw,'LineStyle','--');
```

iperf_file.m

```
1
2   % This is called from the menu of YJ_program under:
3   %     File > Iperf
4   % this command finds the iperf data for the current YJ data file
5   load handles;
6   ip_fi = handles.file;
7   ip_pa = handles.path;
8
9   iperf_numeric = 1;  %Conditon for iperf_gui menu
10
11      save('iperf_num_wksp','iperf_numeric', ...
12          'ip_fi', 'ip_pa')
13      run iperf_gui
```

iperf_gui.m

```
1   function varargout = iperf_gui(varargin)
2   % IPERF_GUI M-file for iperf_gui.fig
3   %       IPERF_GUI, by itself, creates a new IPERF_GUI or raises the existing
4   %       singleton*.
5   %
6   %       H = IPERF_GUI returns the handle to a new IPERF_GUI or the handle to
7   %       the existing singleton*.
```

```
8  %
9  %      IPERF_GUI('CALLBACK',hObject,eventData,iperf,...) calls the local
10 %      function named CALLBACK in IPERF_GUI.M with the given input arguments.
11 %
12 %      IPERF_GUI('Property','Value',...) creates a new IPERF_GUI or raises the
13 %      existing singleton*.  Starting from the left, property value pairs are
14 %      applied to the GUI before iperf_gui_OpeningFunction gets called.  An
15 %      unrecognized property name or invalid value makes property application
16 %      stop.  All inputs are passed to iperf_gui_OpeningFcn via varargin.
17 %
18 %      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
19 %      instance to run (singleton)".
20 %
21 % See also: GUIDE, GUIDATA, GUIHANDLES
22
23 % Edit the above text to modify the response to help iperf_gui
24
25 % Last Modified by GUIDE v2.5 01-Nov-2004 16:56:26
26
27 % Begin initialization code - DO NOT EDIT
28
29 gui_Singleton = 1;
30 gui_State = struct('gui_Name',       mfilename, ...
31                    'gui_Singleton',  gui_Singleton, ...
32                    'gui_OpeningFcn', @iperf_gui_OpeningFcn, ...
33                    'gui_OutputFcn',  @iperf_gui_OutputFcn, ...
34                    'gui_LayoutFcn',  [] , ...
35                    'gui_Callback',   []);
36 if nargin & isstr(varargin{1})
37     gui_State.gui_Callback = str2func(varargin{1});
38 end
39
40 if nargout
41     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
42 else
43     gui_mainfcn(gui_State, varargin{:});
44 end
45 % End initialization code - DO NOT EDIT
46
47
48 % --- Executes just before iperf_gui is made visible.
49 function iperf_gui_OpeningFcn(hObject, eventdata, iperf, varargin)
50 % This function has no output args, see OutputFcn.
51 % hObject    handle to figure
52 % eventdata  reserved - to be defined in a future version of MATLAB
```

```
53  % iperf     structure with iperf and user data (see GUIDATA)
54  % varargin   command line arguments to iperf_gui (see VARARGIN)
55
56  %Creat Menu Options0-----------------------------
57
58  load iperf_num_wksp
59
60  if iperf_numeric ~= 3      %If called from YJ_work1
61  %     iperf_gui menu
62      file = uimenu('Label','File');
63      uimenu(file,'Label','Open','Callback','get_iperf_data_igui');
64      uimenu(file,'Label','Save Plot','Callback','save');
65      uimenu(file,'Label','Quit','Callback','exit',...
66          'Separator','on','Accelerator','Q');
67
68      PP   = uimenu('Label','Plot Properties');
69      uimenu(PP, 'Label', 'Grid', 'Callback', 'Grid_stat');
70      uimenu(PP, 'Label', 'Clear Graph', 'Callback', 'cla');
71
72      help = uimenu('Label','Help')
73      uimenu(help,'Label','Help', 'Callback', 'help_Doc');
74  end
75      if iperf_numeric == 1
76          load handles;
77          set(gcf,'Name',handles.filename);
78          iperf.rssi = handles.rssi;
79          iperf = Iperf_data(iperf);
80          set(iperf.rssi_but,'Value',1);
81          set(iperf.rssi_but,'Visible','on')
82
83      else
84          set(gcf,'Name',ip_fi)
85
86          iperf.time = time
87          iperf.BW = BW
88          iperf.PER = PER
89
90          iperf.rssi_val = 0;
91          set(iperf.rssi_but,'Value',0);
92          set(iperf.rssi_but,'Visible','off')
93      end
94
95  delete_file = 0;
96  save('iperf_num_wksp','delete_file');
97
```

```
98
99
100  iperf.BW = BW_filter(iperf.BW);
101
102  save iperf
103  re_plot(iperf)
104
105  % Choose default command line output for iperf_gui
106  %iperf.output = hObject;
107
108  % Update iperf structure
109  guidata(hObject, iperf);
110
111  % UIWAIT makes iperf_gui wait for user response (see UIRESUME)
112  % uiwait(iperf.figure1);
113
114
115  % % --- Outputs from this function are returned to the command line.
116  % function varargout = iperf_gui_OutputFcn(hObject, eventdata, iperf)
117  % % varargout  cell array for returning output args (see VARARGOUT);
118  % % hObject    handle to figure
119  % % eventdata  reserved - to be defined in a future version of MATLAB
120  % % iperf    structure with iperf and user data (see GUIDATA)
121  %
122  % % Get default command line output from iperf structure
123  % varargout{1} = iperf.output;
124
125
126  % --- Executes on button press in txt_per_but.
127  function per_but_Callback(hObject, eventdata, iperf)
128  % hObject    handle to txt_per_but (see GCBO)
129  % eventdata  reserved - to be defined in a future version of MATLAB
130  % iperf    structure with iperf and user data (see GUIDATA)
131
132  re_plot(iperf)
133  % Hint: get(hObject,'Value') returns toggle state of txt_per_but
134
135
136  % --- Executes on button press in txt_rssi_but.
137  function rssi_but_Callback(hObject, eventdata, iperf)
138  % hObject    handle to txt_rssi_but (see GCBO)
139  % eventdata  reserved - to be defined in a future version of MATLAB
140  % iperf    structure with iperf and user data (see GUIDATA)
141  re_plot(iperf)
142  % Hint: get(hObject,'Value') returns toggle state of txt_rssi_but
```

```
143
144
145 % --- Executes on button press in bw_but.
146 function bw_but_Callback(hObject, eventdata, iperf)
147 % hObject    handle to bw_but (see GCBO)
148 % eventdata  reserved - to be defined in a future version of MATLAB
149 % iperf    structure with iperf and user data (see GUIDATA)
150 re_plot(iperf)
151
152
153 % Hint: get(hObject,'Value') returns toggle state of bw_but
154
155 %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
156 %========================================================================
157 function re_plot(iperf)
158 load iperf
159 load handles
160
161     %clear statistics
162     set(iperf.mean_rssi,'String','');
163     set(iperf.mean_bw,'String','');
164     set(iperf.mean_per,'String','');
165     set(iperf.var_rssi,'String','');
166     set(iperf.var_bw,'String','');
167     set(iperf.var_per,'String','');
168     opt = 0;
169     bw_val = get(iperf.bw_but,'Value');
170     per_val = get(iperf.per_but,'Value');
171     rssi_val = get(iperf.rssi_but,'Value');
172     subplot(1,1,1)
173     if rssi_val == 0
174         %clear txt_rssi statistics
175         set(iperf.mean_rssi,'Visible','off');
176         set(iperf.var_rssi,'Visible','off');
177
178         subplot(1,1,1)
179         if (get(iperf.bw_but,'Value') & get(iperf.per_but,'Value')) == 1
180             %Plot TXT_PER and BW information
181             [haxes,h_per,h_bw] = plotyy(iperf.time,100-iperf.PER, ...
182                 iperf.time(1:length(iperf.BW)),iperf.BW);
183             title('Datagrams Received [%] and BW [Mbits/sec]');
184         %Axis Label
185             set(get(haxes(1),'Ylabel'),'String','Percent')
186             set(get(haxes(2),'Ylabel'),'String','Mbit/sec')
187         %Axis limits
```

```
188             ylim(haxes(1),[0 105]);
189             ylim(haxes(2),[0 10]);
190         %Line Properties
191             set(h_per,'Color','b');
192             set(h_bw,'LineStyle','--');
193         %Set GUI Visulation Parameters
194             set(iperf.mean_bw,'Visible','off');
195             set(iperf.var_bw,'Visible','off');
196
197             set(iperf.mean_bw,'Visible','on');
198             set(iperf.var_bw,'Visible','on');
199             set(iperf.mean_bw,'String',num2str(mean(iperf.BW)));
200             set(iperf.var_bw,'String',num2str(var(iperf.BW)));
201
202             set(iperf.mean_per,'Visible','on');
203             set(iperf.var_per,'Visible','on');
204             set(iperf.mean_per,'String',num2str(100-mean(iperf.PER)));
205             set(iperf.var_per,'String',num2str(var(iperf.PER)));
206
207             time = iperf.time;
208             PER = iperf.PER;
209             BW = iperf.BW;
210             file = ip_fi;
211             save('current_data','time','PER','BW','file');
212
213         elseif get(iperf.per_but,'Value') == 1 & get(iperf.bw_but,'Value') ==0
214             %Plot TXT_PER
215             plot(iperf.time,100-iperf.PER);title('Datagrams Received [%]');
216             ylabel('Percent');
217             ylim([0 105]);
218             %Set GUI Visulation Parameters
219             set(iperf.mean_bw,'Visible','off');
220             set(iperf.var_bw,'Visible','off');
221
222             set(iperf.mean_per,'Visible','on');
223             set(iperf.var_per,'Visible','on');
224             set(iperf.mean_per,'String',num2str(100-mean(iperf.PER)));
225             set(iperf.var_per,'String',num2str(var(iperf.PER)));
226
227             time = iperf.time;
228             PER = iperf.PER;
229             file = ip_fi;
230             save('current_data','time','PER','file');
231
232         elseif get(iperf.per_but,'Value') == 0 & get(iperf.bw_but,'Value') ==1
```

```
233             %Plot BW
234             plot(iperf.time,iperf.BW);title('BW [Mbits/sec]');
235             axis([0 length(iperf.time) 0 10]);
236             %Set GUI Visulation Parameters
237             set(iperf.mean_bw,'Visible','on');
238             set(iperf.var_bw,'Visible','on');
239             set(iperf.mean_bw,'String',num2str(mean(iperf.BW)));
240             set(iperf.var_bw,'String',num2str(var(iperf.BW)));
241
242             set(iperf.mean_per,'Visible','off');
243             set(iperf.var_per,'Visible','off');
244
245             time = iperf.time;
246             BW = iperf.BW;
247             file = ip_fi;
248             save('current_data','time','BW','file');
249
250         elseif (get(iperf.bw_but,'Value') & get(iperf.per_but,'Value')) == 0
251             subplot(1,1,1)
252             plot(0,0)
253             title('Nothing Selected');
254             %Set GUI Visulation Parameters
255             set(iperf.mean_per,'Visible','off');
256             set(iperf.var_rssi,'Visible','off');
257             set(iperf.var_bw,'Visible','off');
258             set(iperf.var_per,'Visible','off');
259         end
260     elseif rssi_val == 1
261         %Set txt_rssi statistics
262         set(iperf.mean_rssi,'Visible','on');
263         set(iperf.var_rssi,'Visible','on');
264         set(iperf.mean_rssi,'String',num2str(mean(iperf.rssi)));
265         set(iperf.var_rssi,'String',num2str(var(iperf.rssi)));
266
267         subplot(211)
268         if (get(iperf.bw_but,'Value') & get(iperf.per_but,'Value')) == 1
269
270             [haxes,h_per,h_bw] = plotyy(iperf.time,100-iperf.PER, ...
271                 iperf.time,iperf.BW(1:length(iperf.time)));
272             title('Datagrams Received [%] and BW [Mbits/sec]');
273         %Axis Label
274             set(get(haxes(1),'Ylabel'),'String','Percent')
275             set(get(haxes(2),'Ylabel'),'String','Mbit/sec')
276         %Axis Limits
277             ylim(haxes(1),[0 105]);
```

```
278             ylim(haxes(2),[0 10]);
279         %Line Properties
280             set(h_per,'Color','b');
281             set(h_bw,'LineStyle','--');
282             %Set GUI Visulation Parameters
283             set(iperf.mean_bw,'Visible','on');
284             set(iperf.var_bw,'Visible','on');
285             set(iperf.mean_bw,'String',num2str(mean(iperf.BW)));
286             set(iperf.var_bw,'String',num2str(var(iperf.BW)));
287
288             set(iperf.mean_per,'Visible','on');
289             set(iperf.var_per,'Visible','on');
290             set(iperf.mean_per,'String',num2str(100-mean(iperf.PER)));
291             set(iperf.var_per,'String',num2str(var(iperf.PER)));
292
293             time = iperf.time;
294             PER = iperf.PER;
295             BW = iperf.BW;
296             rssi = iperf.rssi;
297             file = ip_fi;
298             save('current_data','time','PER','BW','rssi','file');
299
300         elseif get(iperf.per_but,'Value') == 1 & get(iperf.bw_but,'Value') ==0
301
302             plot(iperf.time,100-iperf.PER);title('Datagrams Received [%]');
303             ylabel('Percent');
304             ylim([0 105]);
305             %Set GUI Visulation Parameters
306             set(iperf.mean_bw,'Visible','off');
307             set(iperf.var_bw,'Visible','off');
308
309             set(iperf.mean_per,'Visible','on');
310             set(iperf.var_per,'Visible','on');
311             set(iperf.mean_per,'String',num2str(100-mean(iperf.PER)));
312             set(iperf.var_per,'String',num2str(var(iperf.PER)));
313
314             time = iperf.time;
315             PER = iperf.PER;
316             rssi = iperf.rssi;
317             file = ip_fi;
318             save('current_data','time','PER','rssi','file');
319
320         elseif get(iperf.per_but,'Value') == 0 & get(iperf.bw_but,'Value') ==1
321             plot(iperf.time,iperf.BW);title('BW [Mbits/sec]');
322             %Set GUI Visulation Parameters
```

```
323             set(iperf.mean_bw,'Visible','on');
324             set(iperf.var_bw,'Visible','on');
325             set(iperf.mean_bw,'String',num2str(mean(iperf.BW)));
326             set(iperf.var_bw,'String',num2str(var(iperf.BW)));
327
328             set(iperf.mean_per,'Visible','off');
329             set(iperf.var_per,'Visible','off');
330
331             time = iperf.time;
332             BW = iperf.BW;
333             rssi = iperf.rssi;
334             file = ip_fi;
335             save('current_data','time','rssi','BW','file');
336
337         elseif (get(iperf.bw_but,'Value') & get(iperf.per_but,'Value')) == 0
338             subplot(1,1,1)
339             plot(iperf.rssi);title('RSSI');
340             ylabel('dBm')
341             opt = 1;
342             %Set GUI Visulation Parameters
343             set(iperf.mean_bw,'String','');
344             set(iperf.var_bw,'String','');
345
346             set(iperf.mean_per,'String','');
347             set(iperf.var_per,'String','');
348         end
349         if opt == 0
350             subplot(212)
351             plot(iperf.rssi);title('RSSI');
352             ylabel('dBm')
353
354         end
355     end
356
357
358
359 % --- Executes during object creation, after setting all properties.
360 function mean_txt_rssi_CreateFcn(hObject, eventdata, handles)
361 % hObject    handle to mean_txt_rssi (see GCBO)
362 % eventdata  reserved - to be defined in a future version of MATLAB
363 % handles    empty - handles not created until after all CreateFcns called
364
365 % Hint: edit controls usually have a white background on Windows.
366 %       See ISPC and COMPUTER.
367 if ispc
```

```
368         set(hObject,'BackgroundColor','white');
369     else
370         set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
371     end
372
373
374
375     function mean_txt_rssi_Callback(hObject, eventdata, handles)
376     % hObject    handle to mean_txt_rssi (see GCBO)
377     % eventdata  reserved - to be defined in a future version of MATLAB
378     % handles    structure with handles and user data (see GUIDATA)
379
380     % Hints: get(hObject,'String') returns contents of mean_txt_rssi as text
381     %        str2double(get(hObject,'String')) returns contents of mean_txt_rssi as a double
382
383
384     % --- Executes during object creation, after setting all properties.
385     function var_txt_rssi_CreateFcn(hObject, eventdata, handles)
386     % hObject    handle to var_txt_rssi (see GCBO)
387     % eventdata  reserved - to be defined in a future version of MATLAB
388     % handles    empty - handles not created until after all CreateFcns called
389
390     % Hint: edit controls usually have a white background on Windows.
391     %       See ISPC and COMPUTER.
392     if ispc
393         set(hObject,'BackgroundColor','white');
394     else
395         set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
396     end
397
398
399
400     function var_txt_rssi_Callback(hObject, eventdata, handles)
401     % hObject    handle to var_txt_rssi (see GCBO)
402     % eventdata  reserved - to be defined in a future version of MATLAB
403     % handles    structure with handles and user data (see GUIDATA)
404
405     % Hints: get(hObject,'String') returns contents of var_txt_rssi as text
406     %        str2double(get(hObject,'String')) returns contents of var_txt_rssi as a double
407
408
409     % --- Executes during object creation, after setting all properties.
410     function var_bw_CreateFcn(hObject, eventdata, handles)
411     % hObject    handle to var_bw (see GCBO)
412     % eventdata  reserved - to be defined in a future version of MATLAB
```

```
413 % handles    empty - handles not created until after all CreateFcns called
414
415 % Hint: edit controls usually have a white background on Windows.
416 %       See ISPC and COMPUTER.
417 if ispc
418     set(hObject,'BackgroundColor','white');
419 else
420     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
421 end
422
423
424
425 function var_bw_Callback(hObject, eventdata, handles)
426 % hObject    handle to var_bw (see GCBO)
427 % eventdata  reserved - to be defined in a future version of MATLAB
428 % handles    structure with handles and user data (see GUIDATA)
429
430 % Hints: get(hObject,'String') returns contents of var_bw as text
431 %        str2double(get(hObject,'String')) returns contents of var_bw as a double
432
433
434 % --- Executes during object creation, after setting all properties.
435 function mean_bw_CreateFcn(hObject, eventdata, handles)
436 % hObject    handle to mean_bw (see GCBO)
437 % eventdata  reserved - to be defined in a future version of MATLAB
438 % handles    empty - handles not created until after all CreateFcns called
439
440 % Hint: edit controls usually have a white background on Windows.
441 %       See ISPC and COMPUTER.
442 if ispc
443     set(hObject,'BackgroundColor','white');
444 else
445     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
446 end
447
448
449
450 function mean_bw_Callback(hObject, eventdata, handles)
451 % hObject    handle to mean_bw (see GCBO)
452 % eventdata  reserved - to be defined in a future version of MATLAB
453 % handles    structure with handles and user data (see GUIDATA)
454
455 % Hints: get(hObject,'String') returns contents of mean_bw as text
456 %        str2double(get(hObject,'String')) returns contents of mean_bw as a double
457
```

```
458
459 % --- Executes during object creation, after setting all properties.
460 function var_txt_per_CreateFcn(hObject, eventdata, handles)
461 % hObject    handle to var_txt_per (see GCBO)
462 % eventdata  reserved - to be defined in a future version of MATLAB
463 % handles    empty - handles not created until after all CreateFcns called
464
465 % Hint: edit controls usually have a white background on Windows.
466 %       See ISPC and COMPUTER.
467 if ispc
468     set(hObject,'BackgroundColor','white');
469 else
470     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
471 end
472
473
474
475 function var_txt_per_Callback(hObject, eventdata, handles)
476 % hObject    handle to var_txt_per (see GCBO)
477 % eventdata  reserved - to be defined in a future version of MATLAB
478 % handles    structure with handles and user data (see GUIDATA)
479
480 % Hints: get(hObject,'String') returns contents of var_txt_per as text
481 %        str2double(get(hObject,'String')) returns contents of var_txt_per as a double
482
483
484 % --- Executes during object creation, after setting all properties.
485 function mean_txt_per_CreateFcn(hObject, eventdata, handles)
486 % hObject    handle to mean_txt_per (see GCBO)
487 % eventdata  reserved - to be defined in a future version of MATLAB
488 % handles    empty - handles not created until after all CreateFcns called
489
490 % Hint: edit controls usually have a white background on Windows.
491 %       See ISPC and COMPUTER.
492 if ispc
493     set(hObject,'BackgroundColor','white');
494 else
495     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
496 end
497
498
499
500 function mean_txt_per_Callback(hObject, eventdata, handles)
501 % hObject    handle to mean_txt_per (see GCBO)
502 % eventdata  reserved - to be defined in a future version of MATLAB
```

```
503  % handles    structure with handles and user data (see GUIDATA)
504
505  % Hints: get(hObject,'String') returns contents of mean_txt_per as text
506  %        str2double(get(hObject,'String')) returns contents of mean_txt_per as a double
507
508
509  % --- Executes on button press in Ext_fig.
510  function Ext_fig_Callback(hObject, eventdata, handles)
511  % hObject    handle to Ext_fig (see GCBO)
512  % eventdata  reserved - to be defined in a future version of MATLAB
513  % handles    structure with handles and user data (see GUIDATA)
514  load current_data
515
516  name = figure
517
518  set(name,'Name',file)
519  set(name,'NumberTitle','off')
520
521
522  if exist('PER') & exist('BW')
523          %Plot TXT_PER and BW information
524              [haxes,h_per,h_bw] = plotyy(time,100-PER, ...
525                  time(1:length(BW)),BW);
526              title('Datagrams Received [%] and BW [Mbits/sec]');
527          %Axis Label
528              set(get(haxes(1),'Ylabel'),'String','Percent')
529              set(get(haxes(2),'Ylabel'),'String','Mbit/sec')
530          %Axis limits
531              ylim(haxes(1),[0 105]);
532              ylim(haxes(2),[0 10]);
533          %Line Properties
534              set(h_per,'Color','b');
535              set(h_bw,'LineStyle','-.');
536              legend([h_per h_bw],'Datagram','Bandwidth')
537  elseif exist('PER')
538          %Plot TXT_PER
539              plot(time,100-PER);title('Datagrams Received [%]');
540              title('Datagrams Received [%]');
541              ylabel('Percent');
542              ylim([0 105]);
543              legend('Percent')
544  elseif exist('BW')
545          %Plot BW
546              plot(time,BW);title('BW [Mbits/sec]');
547              title('BW [Mbits/sec]');
```

```
548             legend('Bandwidth');
549
550 end
```

iperf_pre.m

```
1  load iperf
2  if ishandle(iperf.figure1)
3      close(iperf.figure1)
4  end
5
6  run iperf_gui;
```

mouse_data.m

```
1  %Sebastian Stewart
2  %================================================
3  %This program is what allows the user to
4  %right click on the delay spread plot and
5  %select function to execute
6  %================================================
7  cmenu = uicontextmenu;
8          mu = handles.DS_mean* ...
9              ones(1,length(handles.current_data));
10
11
12         handles.ds_plot = plot(handles.current_data, 'UIContextMenu',cmenu);
13         hold on;
14         handles.sds_plot = plot(mv_data,'r', 'UIContextMenu',cmenu);
15         handles.mu_plot = plot(mu,'g' ...
16              ,'UIContextMenu',cmenu);
17         hold off;
18         set(cmenu,'Tag',char(handles.ssid_name));
19 %these are the funciton that execute when item are selected
20         cb1 = ['this'];
21         cb2 = ['sds_menu(handles)'];
22         cb3 = ['uiresume'];
23 %These are what get displayed when right click of mouse on object
24         item1 = uimenu(cmenu, 'Label', 'This', ...
25             'Callback', cb1);
26         item2 = uimenu(cmenu, 'Label', 'MA Delay Spread', ...
```

```
27              'Callback', cb2);
28          item3 = uimenu(cmenu, 'Label', 'resume', ...
29              'Callback', cb3);
```

---

new_file.m

```
1
2   [iperf.file iperf.path] = uigetfile('*.txt');
3   set(gcf,'Name',iperf.file);
4   temp = importdata([iperf.path iperf.file]);
5
6   iperf.time = temp.data(:,1);
7   iperf.BW = temp.data(:,3);
8   iperf.PER = temp.data(:,7);
9
10  load handles;
11  iperf.rssi = handles.rssi;
12
13  iperf = Iperf_data(iperf)
14  save iperf
15  re_plot(iperf)
```

---

re_plot.m

---

ssid_findnew.m

```
1   function [ssid_names, ssid_index] = ssid_findnew(data,col)
2
3   %=====================================================
4   %Assumptions The data is sorted by channel and SSID
5   %this program:]
6   %     1) outputs the index of each ssid
7   %     2) output cell array of the ssid in the data
8   %=====================================================
9
10  % initial = 0                    %searching for new ssid name
11  % start = [];
12
```

```
13  % %set initial ssid name
14       ssid_names = {};
15  % %set initial ssid index
16       ssid_index = [];
17
18  %Search all rows
19  for i = 2:length(data)
20
21      if char(data{i,col})
22
23          %if strings are same
24          length(char(data{i,col}));
25          length(char(data{i-1,col}));
26          char(data{i,col});
27          char(data{i-1,col});
28
29          if length(char(data{i,col})) ~= length(char(data{i-1,col}))
30
31              ssid_index(length(ssid_index)+1) = i-1;
32              ssid_index(length(ssid_index)+1) = i;
33              ssid_names{length(ssid_names)+1,1} = data(i,col);
34
35          elseif length(char(data{i,col})) == length(char(data{i-1,col})) ...
36                  & char(data{i,col}) ~= char(data{i-1,col})
37              ssid_index(length(ssid_index)+1) = i-1;
38              ssid_index(length(ssid_index)+1) = i;
39              ssid_names{length(ssid_names)+1,1} = data(i,col);
40          end
41      end
42  end
43  ssid_index(length(ssid_index)+1) = i;
44
45
```

time_sample_data.m

```
1   function handles = time_sample_data(handles)
2   %handles used: row      handles created: tsmpl
3   user_str = {'user'};
4
5   tm_per = .1:.1:1
6   ssid_len = handles.row(2)-handles.row(1)
7   time_pts = unique(ceil(ssid_len.*tm_per))
```

```
 8   handles.tsmpl = time_pts(1)
 9
10   % time_pts = cellstr(time_pts)
11   y = cellstr(time_pts);
12   time_pts = cat(2,y,user_str);
13
14   set(handles.dwn_samp,'String',time_pts)
15
16
```

YJ_work1.m

```
 1   function varargout = YJ_work1(varargin)
 2
 3   %for_data (90,184)
 4   %data_fetch (445)
 5   %time_sample_data (356, 421)
 6   %delay_spread (371,421,439)
 7   %col_find (99,190)
 8   %cell_find (188)
 9
10
11
12
13   % YJ_WORK1 M-file for YJ_work1.fig
14   %      YJ_WORK1, by itself, creates a new YJ_WORK1 or raises the existing
15   %      singleton*.
16   %
17   %      H = YJ_WORK1 returns the handle to a new YJ_WORK1 or the handle to
18   %      the existing singleton*.
19   %
20   %      YJ_WORK1('CALLBACK',hObject,eventData,handles,...) calls the local
21   %      function named CALLBACK in YJ_WORK1.M with the given input arguments.
22   %
23   %      YJ_WORK1('Property','Value',...) creates a new YJ_WORK1 or raises the
24   %      existing singleton*.  Starting from the left, property value pairs are
25   %      applied to the GUI before YJ_work1_OpeningFunction gets called.  An
26   %      unrecognized property name or invalid value makes property application
27   %      stop.  All inputs are passed to YJ_work1_OpeningFcn via varargin.
28   %
29   %      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
30   %      instance to run (singleton)".
31   %
```

```
32  % See also: GUIDE, GUIDATA, GUIHANDLES
33
34  % Edit the above text to modify the response to help YJ_work1
35
36  % Last Modified by GUIDE v2.5 24-Jul-2004 14:29:00
37
38  % Begin initialization code - DO NOT EDIT
39  gui_Singleton = 1;
40  gui_State = struct('gui_Name',       mfilename, ...
41                     'gui_Singleton',  gui_Singleton, ...
42                     'gui_OpeningFcn', @YJ_work1_OpeningFcn, ...
43                     'gui_OutputFcn',  @YJ_work1_OutputFcn, ...
44                     'gui_LayoutFcn',  [] , ...
45                     'gui_Callback',   []);
46  if nargin & isstr(varargin{1})
47      gui_State.gui_Callback = str2func(varargin{1});
48  end
49
50  if nargout
51      [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
52  else
53      gui_mainfcn(gui_State, varargin{:});
54  end
55  % End initialization code - DO NOT EDIT
56
57
58  % --- Executes just before YJ_work1 is made visible.
59  function YJ_work1_OpeningFcn(hObject, eventdata, handles, varargin)
60  % This function has no output args, see OutputFcn.
61  % hObject    handle to figure
62  % eventdata  reserved - to be defined in a future version of MATLAB
63  % handles    structure with handles and user data (see GUIDATA)
64  % varargin   command line arguments to YJ_work1 (see VARARGIN)
65
66  %This is not yet working completly
67  %Creat Menu Options0-----------------------------
68
69  file = uimenu('Label','File');
70      uimenu(file,'Label','Iperf','Callback','iperf_file');
71      uimenu(file,'Label','Save Plot','Callback','save');
72      uimenu(file,'Label','Quit','Callback','exit',...
73             'Separator','on','Accelerator','Q');
74
75  PP   = uimenu('Label','Plot Properties');
76      uimenu(PP, 'Label', 'Grid', 'Callback', 'Grid_stat');
```

```
77      uimenu(PP, 'Label', 'Clear Graph', 'Callback', 'cla');
78
79  User_input = uimenu('Label','User Input');
80      uimenu(User_input,'Label','iperf numeric','Callback','get_iperf_data');
81
82  help = uimenu('Label','Help');
83      uimenu(help,'Label','Help', 'Callback', 'help_Doc');
84
85  %-------------------------------------------------
86
87  %Read file name and path
88      [handles.filename handles.path] = uigetfile('*.out');
89  %Save file name for iperf call
90      fi=handles.filename;    pa=handles.path;
91      save iperf_loc fi pa
92  %Show Path on YJ figure
93      folder = getfolder(handles.path);
94      set(handles.figure1,'Name',folder);
95  %Define file location
96      file_loc = [handles.path handles.filename];
97  %=================================================================
98  %Show file name in filename box
99       set(handles.file,'String',handles.filename);
100
101     h = waitbar(0,'Please wait...');
102
103 % %Import the data as Cell Array
104     s = importdata(file_loc,'\t');
105
106     waitbar(10/100);
107
108 %If structure then convert to Cell Array
109 if isstruct(s)
110         s = struct2cell(s)
111 end
112 % %Discretize the Cell Array by delimiter
113     handles.data = for_data(s);
114
115     waitbar(.5)             %update waitbar
116
117 %Read labels
118     [labels header_index handles.ref] = ...
119         cell_find(handles.data, [{'SSID'} {'Multipath Data'} {'Data Rate'} ...
120         {'RSSI'} {'Latitude (Dec. Deg.)'} {'Longitude (Dec. Deg.)'} ...
121         {'Altitude (FT)'}]);
```

```
122
123
124  handles = col_find(handles);
125
126  %-------------------------------------------
127  %   handles.ref contain columns location of
128  %   cell array specified above
129  %-------------------------------------------
130
131  %SSID header_index has ssid column number in first element
132  waitbar(.75)
133
134  %Find the rows of the rows of each ssid
135  [ssid_names handles.ssid_row] = ssid_findnew(handles.data, handles.col_ssid);
136
137  % Display SSid names is ssid_name_array pop-up menu
138      ssid_names = cellarray(ssid_names);
139      set(handles.ssid_name_array,'String', ...
140          ssid_names);
141
142  %Get first ssid name displayed in handles.ssid_name_array
143  handles = ssid_name_array_Callback(handles.ssid_name_array, eventdata, handles);
144  %Get first Time Point displayed in handles.tsmpl
145  handles = dwn_samp_Callback(handles.dwn_samp,eventdata, handles);
146
147  plot_Callback(hObject, eventdata, handles)
148  waitbar(1)
149  close(h);
150  save handles
151
152  %====GUI STUFF=============================================
153  % |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
154  %\/  \/  \/  \/  \/  \/  \/  \/  \/  \/  \/  \/  \/  \/  \/
155  % Choose default command line output for YJ_work1
156  handles.output = hObject;
157  % Update handles structure
158  guidata(hObject, handles);
159  % UIWAIT makes YJ_work1 wait for user response (see UIRESUME)
160  % uiwait(handles.figure1);
161
162
163  % --- Outputs from this function are returned to the command line.
164  function varargout = YJ_work1_OutputFcn(hObject, eventdata, handles)
165  % varargout  cell array for returning output args (see VARARGOUT);
166  % hObject    handle to figure
```

```
167  % eventdata  reserved - to be defined in a future version of MATLAB
168  % handles    structure with handles and user data (see GUIDATA)
169
170  % Get default command line output from handles structure
171  varargout{1} = handles.output;
172
173
174  %------------------PUSH BOTTONS--------------------
175  %==================================================
176  %==================================================
177
178  % --- Executes on button press in browse.
179  function browse_Callback(hObject, eventdata, handles)
180  clc;
181  load handles
182  %Create Menu Options
183  %-------------------------------------------
184  % f = uimenu('Label','Plot Protperties');
185  %     uimenu(f,'Label','New Figure','Callback','figure');
186  %     uimenu(f,'Label','Save','Callback','save');
187  %     uimenu(f,'Label','Quit','Callback','exit',...
188  %           'Separator','on','Accelerator','Q');
189  %-------------------------------------------
190  file = handles.filename;pa = handles.path ;
191  % save wksp hObject file pa
192  % clear
193  % load wksp
194  % hObject    handle to browse (see GCBO)
195  % eventdata  reserved - to be defined in a future version of MATLAB
196  % handles    structure with handles and user data (see GUIDATA)
197  %set(handles.ssid_name_array,'String',);
198  %\\\\\\\\\\\  DATA DISPLAY PROGRAM  ////////////////////
199
200
201
202  %Read file name and path
203      [handles.filename handles.path] = uigetfile('*.out');
204
205
206      if handles.path ~= 0
207
208          %Show Path on YJ figure
209          folder = getfolder(handles.path);
210          set(handles.figure1,'Name',folder);
211          %Define file location
```

```
212         file_loc = [handles.path handles.filename];
213         %Save file name for iperf call
214         fi=handles.filename;    pa=handles.path;
215         save iperf_loc fi pa
216
217         %================================================================
218         %Show file name in filename box
219         set(handles.file,'String',handles.filename);
220
221         h = waitbar(0,'Please wait...');
222
223         % %Import the data as Cell Array
224         s = importdata(file_loc,'\t');
225
226         waitbar(10/100);
227
228         %If structure then convert to Cell Array
229         if isstruct(s)
230             s = struct2cell(s);
231         end
232         % %Discretize the Cell Array by delimiter
233         handles.data = for_data(s);
234
235         waitbar(.5)             %updata waitbar
236
237         %Read labels
238         [labels header_index handles.ref] = ...
239             cell_find(handles.data, [{'SSID'} {'Multipath Data'} {'Data Rate'} ...
240                 {'RSSI'} {'Latitude (Dec. Deg.)'} {'Longitude (Dec. Deg.)'} ...
241                 {'Altitude (FT)'}]);
242
243
244         handles = col_find(handles);
245
246         %-------------------------------------------
247         %   handles.ref contain columns location of
248         %   cell array specified above
249         %-------------------------------------------
250
251         %SSID header_index has ssid column number in first element
252         waitbar(.75)
253
254         %Find the rows of the rows of each ssid
255         [ssid_names handles.ssid_row] = ssid_findnew(handles.data, handles.col_ssid);
256
```

```
257          % Display SSid names is ssid_name_array pop-up menu
258          ssid_names = cellarray(ssid_names);
259          set(handles.ssid_name_array,'String', ...
260              ssid_names);
261
262          %Get first ssid name displayed in handles.ssid_name_array
263          handles = ssid_name_array_Callback(handles.ssid_name_array, eventdata, handles);
264          %Get first Time Point displayed in handles.tsmpl
265          handles = dwn_samp_Callback(handles.dwn_samp,eventdata, handles);
266
267          waitbar(1)
268          close(h);
269      else
270          handles.filename = file;
271          handles.path = pa;
272      end
273  save handles;
274  %-------------------------------------------------------------------------
275  %=========================================================================
276  %\\\\\\\\\\\\\\\\\\\\   Other plotting functions////////////////
277  % --- Executes on button press in surf.
278  function surf_Callback(hObject, eventdata, handles)
279  % hObject    handle to surf (see GCBO)
280  % eventdata  reserved - to be defined in a future version of MATLAB
281  % handles    structure with handles and user data (see GUIDATA)
282  [r c] = size(handles.current_data);
283  if c > 1
284     x = [-2:0.25:3.25];
285     y = [1:r];
286     surf(x,y,handles.current_data(:,handles.col_in:handles.col_fn));
287     xlabel('Chips');
288     ylabel('Relative time');
289
290  end
291
292  % --- Executes on button press in contour.
293  function contour_Callback(hObject, eventdata, handles)
294  % hObject    handle to contour (see GCBO)
295  % eventdata  reserved - to be defined in a future version of MATLAB
296  % handles    structure with handles and user data (see GUIDATA)
297  [r c] = size(handles.current_data);
298  if c > 1
299     x = [-2:0.25:3.25];
300     y = [1:r];
301     contour(x,y,handles.current_data(:,handles.col_in:handles.col_fn));
```

```
302     xlabel('Chips');
303     ylabel('Relative time');
304 end
305
306
307
308 % --- Executes on button press in mesh.
309 function mesh_Callback(hObject, eventdata, handles)
310 % hObject    handle to mesh (see GCBO)
311 % eventdata  reserved - to be defined in a future version of MATLAB
312 % handles    structure with handles and user data (see GUIDATA)
313 [r c] = size(handles.current_data);
314 if c > 1
315     x = [-2:0.25:3.25];
316     y = [1:r];
317     mesh(x,y,handles.current_data(:,handles.col_in:handles.col_fn));
318     xlabel('Chips');
319     ylabel('Relative time');
320 end
321
322
323 % --- Executes on button press in plot.
324 function plot_Callback(hObject, eventdata, handles)
325 % hObject    handle to plot (see GCBO)
326 % eventdata  reserved - to be defined in a future version of MATLAB
327 % handles    structure with handles and user data (see GUIDATA)
328 clc
329
330 if handles.file
331
332
333     ts = handles.tsmpl;
334
335     [r c] = size(handles.current_data);
336
337 %If Delay Spread is Selected
338     if c == 1
339     %N-point Moving average filter for Delay spread
340         len = length(handles.current_data)
341         npts = ask_npts(handles)
342         num = ones(1,npts)/npts;
343         mv_data = filter(num,1,handles.current_data);
344
345
346         title([num2str(npts) ' MA filter'])
```

```
347         grid on;
348         ylabel('Nanoseconds');
349         xlabel('Relative Time');
350 %Mouse select Property----------------------------------
351         run mouse_data
352 %If Multipath is Selected
353     elseif c > 1
354         h = waitbar(0,'Please wait...');
355     %Get Status of Radio Button
356         val = get(handles.dB,'Value');
357         x = [-2:0.25:3.25];                 %Chip vector
358     %Create down sample index
359         dsi = 1:r/ts:r;
360         y = [round(dsi)];                    %Time Sample vector
361     %Check if Radio buton
362         if val == 1     %In dB
363             z = 10*log10(handles.current_data(y, ...
364                 handles.col_in:handles.col_fn));
365             waitbar(.6)
366         else            %In decimal
367             z = handles.current_data(y, ...
368                 handles.col_in:handles.col_fn);
369             waitbar(.6)
370         end
371
372         axes(handles.axes1)
373         mesh(x,y,z);title([num2str(ts) ' samples'])
374         xlabel('Chips');
375         ylabel('Relative time');
376
377         waitbar(.9)
378         close(h);
379     else
380         title('Could not display data');
381     end
382 end
383 save handles
384
385 %\\\\\\\\\\\\\\\\\\\\ POPUP MENU ////////////////////////////
386 %\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
387 %\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
388
389 % --- Executes during object creation, after setting all properties.
390 function select_data_CreateFcn(hObject, eventdata, handles)
391 % hObject    handle to select_data (see GCBO)
```

```
392  % eventdata  reserved - to be defined in a future version of MATLAB
393  % handles    empty - handles not created until after all CreateFcns called
394
395  guidata(hObject,handles);
396
397  % Hint: popupmenu controls usually have a white background on Windows.
398  %       See ISPC and COMPUTER.
399  if ispc
400      set(hObject,'BackgroundColor','white');
401  else
402      set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
403  end
404
405
406  % --- Executes on selection change in select_data.
407  function handles = select_data_Callback(hObject, eventdata, handles)
408  % hObject    handle to select_data (see GCBO)
409  % eventdata  reserved - to be defined in a future version of MATLAB
410  % handles    structure with handles and user data (see GUIDATA)
411
412  val = get(hObject,'Value');
413  str = get(hObject, 'String');
414  switch str{val};
415  case 'Multipath'
416      [r c] = size(handles.current_data)
417      if r > 10
418          time_sample_data(handles);
419      end
420      handles.current_data = handles.MP;
421  %Clear Delay Spread info
422      set(handles.mean,'String',[]);
423      set(handles.std_dev,'String',[]);
424  case 'Delay Spread'
425      set(handles.dwn_samp,'String',[]);
426      handles.current_data = delay_spread(handles.MP);
427  %Calc mean and standard deviation
428      handles.DS_mean = mean(handles.current_data)
429      set(handles.mean,'String',handles.DS_mean);
430      handles.DS_sigma  = std(handles.current_data)
431      set(handles.std_dev,'String',handles.DS_sigma);
432  end
433  guidata(hObject,handles);
434  % Hints: contents = get(hObject,'String') returns select_data contents as cell array
435  %        contents{get(hObject,'Value')} returns selected item from select_data
436
```

```
437 %=======================SSID NAME=========================
438 %=========================POPUP===========================
439 %=========================================================
440
441 % --- Executes during object creation, after setting all properties.
442 function ssid_name_array_CreateFcn(hObject, eventdata, handles)
443 % hObject    handle to ssid_name_array (see GCBO)
444 % eventdata  reserved - to be defined in a future version of MATLAB
445 % handles    empty - handles not created until after all CreateFcns called
446 guidata(hObject,handles);
447 % Hint: edit controls usually have a white background on Windows.
448 %       See ISPC and COMPUTER.
449
450 if ispc
451     set(hObject,'BackgroundColor','white');
452 else
453     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
454 end
455
456 %THIS IS THE USER SELECT SSID FUNCTION
457 %=========================================================
458 % |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
459 %\/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/
460 function handles = ssid_name_array_Callback(hObject, eventdata, handles)
461 % hObject    handle to ssid_name_array (see GCBO)
462 % eventdata  reserved - to be defined in a future version of MATLAB
463 % handles    structure with handles and user data (see GUIDATA)
464
465 val = get(hObject,'Value');
466 str = get(hObject, 'String');
467 if length(str) < 2 & val >1
468     val = 1;
469 end
470 handles.ssid_name = str(val);
471
472 %handles = col_find(handles)
473
474 %------Use this code until fixed ssid_row from function
475 %------ssid_findnew
476     len = length(handles.ssid_row);
477     if mod(len,2)
478         handles.row = handles.ssid_row(2*val:(2*val+1));
479     else
480         handles.row = handles.ssid_row(2*val-1:(2*val));
481     end
```

```
482
483  %Set the time sample data select options
484
485      handles = time_sample_data(handles);
486
487
488
489  %=====================================================
490  %------------- Multipath data----------------------
491      col = [handles.col_in handles.col_fn];
492      handles.MP = data_fetch(handles.row,col,handles.data);
493  %-------------Set current data---------------------
494  if get(handles.select_data,'Value') == 2
495      handles.current_data = delay_spread(handles.MP);
496      %Calc mean and standard deviation
497      handles.DS_mean = mean(handles.current_data);
498      set(handles.mean,'String',handles.DS_mean);
499      handles.DS_sigma  = std(handles.current_data);
500      set(handles.std_dev,'String',handles.DS_sigma);
501  else
502      handles.current_data = handles.MP;
503  end
504
505
506  %handles.ssidname = get(handles.ssid_name_array,'String');
507  guidata(hObject,handles);
508  % Hints: get(hObject,'String') returns contents of ssid_name_array as text
509  %        str2double(get(hObject,'String')) returns contents of ssid_name_array as a double
510
511  %=======================FILE NAME=========================
512  %=========================================================
513  %=========================================================
514
515  % --- Executes during object creation, after setting all properties.
516  function file_CreateFcn(hObject, eventdata, handles)
517  % hObject    handle to file (see GCBO)
518  % eventdata  reserved - to be defined in a future version of MATLAB
519  % handles    empty - handles not created until after all CreateFcns called
520  guidata(hObject,handles)
521
522  % Hint: edit controls usually have a white background on Windows.
523  %       See ISPC and COMPUTER.
524  if ispc
525      set(hObject,'BackgroundColor','white');
526  else
```

```
527         set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
528     end
529
530
531
532     function file_Callback(hObject, eventdata, handles)
533     % hObject    handle to file (see GCBO)
534     % eventdata  reserved - to be defined in a future version of MATLAB
535     % handles    structure with handles and user data (see GUIDATA)
536     handles.file = get(handles.file,'String');
537     guidata(hObject,handles)
538     % Hints: get(hObject,'String') returns contents of file as text
539     %        str2double(get(hObject,'String')) returns contents of file as a double
540
541     %===============================================================
542     %================EXTERNAL FIGURE================================
543     %===============================================================
544     % --- Executes on button press in ext_fig.
545     function ext_fig_Callback(hObject, eventdata, handles)
546     % hObject    handle to ext_fig (see GCBO)
547     % eventdata  reserved - to be defined in a future version of MATLAB
548     % handles    structure with handles and user data (see GUIDATA)
549     name = figure
550     file  = handles.filename;
551     ssid = handles.ssid_name{1};
552     set(name,'Name',handles.filename)
553     set(name,'NumberTitle','off')
554
555         val = get(handles.dwn_samp,'Value');
556         str = get(handles.dwn_samp, 'String');
557     %Convert to integer
558
559         ts = handles.tsmpl;
560
561         [r c] = size(handles.current_data);
562         if c == 1
563             plot(handles.current_data)
564             hold on;
565             plot(handles.DS_mean*ones(1,length(handles.current_data)),'g');
566             hold off;
567             title([' Delay Spread of ' file(1:length(file)-9)]);
568         elseif c > 1
569     %Get Status of Radio Button
570             val = get(handles.dB,'Value')
571             x = [-2:0.25:3.25];                 %Chip vector
```

```
572  %Create down sample index
573          dsi = 1:r/ts:r;
574          y = [round(dsi)];                  %Time Sample vector
575  %Check if Radio buton
576          if val == 1     %In dB
577              z = 10*log10(handles.current_data(y, ...
578                  handles.col_in:handles.col_fn));
579          else            %In decimal
580              z = handles.current_data(y, ...
581                  handles.col_in:handles.col_fn);
582          end
583
584          mesh(x,y,z);
585          title([' Power Delay Profile of ' file(1:length(file)-9)]);
586      else
587          title(['Could not display data for ' ssid]);
588       end
589          xlabel('Chips');
590          ylabel('Relative time');
591          zlabel(['Relative Correlation of ' ssid]);
592
593  % --- Executes during object creation, after setting all properties.
594  function mean_CreateFcn(hObject, eventdata, handles)
595  % hObject    handle to mean (see GCBO)
596  % eventdata  reserved - to be defined in a future version of MATLAB
597  % handles    empty - handles not created until after all CreateFcns called
598
599  % Hint: edit controls usually have a white background on Windows.
600  %       See ISPC and COMPUTER.
601  if ispc
602      set(hObject,'BackgroundColor','white');
603  else
604      set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
605  end
606
607
608
609  function mean_Callback(hObject, eventdata, handles)
610  % hObject    handle to mean (see GCBO)
611  % eventdata  reserved - to be defined in a future version of MATLAB
612  % handles    structure with handles and user data (see GUIDATA)
613
614  % Hints: get(hObject,'String') returns contents of mean as text
615  %        str2double(get(hObject,'String')) returns contents of mean as a double
616
```

```
617
618 % --- Executes during object creation, after setting all properties.
619 function std_dev_CreateFcn(hObject, eventdata, handles)
620 % hObject    handle to std_dev (see GCBO)
621 % eventdata  reserved - to be defined in a future version of MATLAB
622 % handles    empty - handles not created until after all CreateFcns called
623
624 % Hint: edit controls usually have a white background on Windows.
625 %       See ISPC and COMPUTER.
626 if ispc
627     set(hObject,'BackgroundColor','white');
628 else
629     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
630 end
631
632
633
634 function std_dev_Callback(hObject, eventdata, handles)
635 % hObject    handle to std_dev (see GCBO)
636 % eventdata  reserved - to be defined in a future version of MATLAB
637 % handles    structure with handles and user data (see GUIDATA)
638
639 % Hints: get(hObject,'String') returns contents of std_dev as text
640 %        str2double(get(hObject,'String')) returns contents of std_dev as a double
641
642
643 % --- Executes on button press in dB.
644 function dB_Callback(hObject, eventdata, handles)
645 % hObject    handle to dB (see GCBO)
646 % eventdata  reserved - to be defined in a future version of MATLAB
647 % handles    structure with handles and user data (see GUIDATA)
648 val = get(handles.select_data,'Value');
649 if  val == 1
650 plot_Callback(hObject, eventdata, handles)
651 end
652 % Hint: get(hObject,'Value') returns toggle state of dB
653
654
655 % --- Executes during object creation, after setting all properties.
656 function dwn_samp_CreateFcn(hObject, eventdata, handles)
657 % hObject    handle to dwn_samp (see GCBO)
658 % eventdata  reserved - to be defined in a future version of MATLAB
659 % handles    empty - handles not created until after all CreateFcns called
660
661 % Hint: popupmenu controls usually have a white background on Windows.
```

```
662   %       See ISPC and COMPUTER.
663   if ispc
664       set(hObject,'BackgroundColor','white');
665   else
666       set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
667   end
668
669
670   % --- Executes on selection change in dwn_samp.
671   function handles = dwn_samp_Callback(hObject, eventdata, handles)
672   % hObject    handle to dwn_samp (see GCBO)
673   % eventdata  reserved - to be defined in a future version of MATLAB
674   % handles    structure with handles and user data (see GUIDATA)
675   %--------------------------------------------------------------------
676   %This functin get the displayed value on Time Points
677   %   handles.tsmpl
678   %--------------------------------------------------------------------
679   val = get(hObject,'Value');
680   str = get(hObject,'String');
681
682   opt = 1;
683
684   switch str{val};
685   case 'user'
686   %Ask user to input number of samples desired
687       tot = handles.row(2)-handles.row(1);
688       answer = tot+1;
689       while answer <= 1 || answer >= tot
690
691           prompt = {['Enter number between 1:' num2str(tot)]};
692           dlg_title = 'Time Points';
693           num_lines= 1;
694           answer  = str2num(char(inputdlg(prompt,dlg_title,num_lines)));
695
696           if answer <= 1 || answer >= tot
697               errordlg(['Please enter number between 1 and ' ...
698                       num2str(tot)]);
699               waitfor(gcf);
700           end
701       end
702
703       handles.tsmpl = answer;
704       opt = 0;
705   end
706
```

```
707   if opt == 1
708       handles.tsmpl = str2num(char(str(val)));
709   end
710   %plot_Callback(hObject, eventdata, handles)
711   guidata(hObject,handles)
712
713
714   % Hints: contents = get(hObject,'String') returns dwn_samp contents as cell array
715   %        contents{get(hObject,'Value')} returns selected item from dwn_samp
716
717
```

IEEE 802.11a Simulation Code

sim_ER_11a.m

```
1   % SIM_ER_11A.M
2   % This is the top-level script for IEEE 802.11a PHY layer simulations,
3   % using the mWLAN Toolbox by CommAccess Technologies.
4   %
5   % To run a set of simulations, just modify the 'simName' variable below,
6   % so that 'RunParams_<simName>.m' is the name of the script file
7   % which sets all the simulation parameters for all the simulation runs
8   % to be done in this set.
9   %
10  % Some terminology to make sense of the nested iterations:
11  %  Calling 'sim_ER_11a.m' once executes a 'simulation set'.
12  %  A simulation set entails N 'simulation runs.'
13  %  Each simulation run simulates a bunch of packets, one by one.
14  %  The number of packets in a run is upper-bounded by the 'MaxPkts'
15  %   parameter in the 'RunParams_X.m' file.  But if 'enough' errors
16  %   occur earlier, then the run will stop before sim'ing all MaxPkts
17  %   packets.  'Enough' is defined by the 'MinErrPkts' parameter,
18  %   and corresponds to the number of pkts with errors, and not just
19  %   the number of bit errors, so that one pkt with 1000 bit errors
20  %   will not end the run.
21  %  The 'vsEbNo' flag, set in the 'RunParams_X.m' file, determines
22  %   how noise is calculated.  If 'vsEbNo', then noise is normalized
23  %   to produce BER-vs-EbNo curves.  If not, then noise is based on
24  %   an absolute-scale estimate of the noise on Mars, and the signal
25  %   power is the absolute power 'txPwr' (Watts) from 'RunParams_X.m'.
26  %  The 'skipDemod' flag allows faster simulation, if you only want
27  %   Pkt Error Rate, based on correct reception of the pkt header.
```

```
28  %   If 'skipDemod', then only the SIGNAL symbol in the header is
29  %   demodulated/decoded.  The data is ignored.  This, together with
30  %   specifying a very short 'pktLen', results in much better sim
31  %   times, when only PER is desired.
32  %
33  % This script produces a (very large) diary file 'simName.txt'.
34  % After each run in the set, a result table will be dumped into the
35  % diary file.  The format of the result table is described in the
36  % comments for the dumpResultTable() function.  The result tables
37  % are cumulative throughout the diary file, so that the table at
38  % the very end of the diary file summarizes all the results for
39  % all the runs in the entire diary file.  This final table should
40  % be extracted into a managably-sized 'Results_simName.m' file
41  % (containing only the one table), and a 'SumAndPlot_simName.m'
42  % or 'SumAndTab_simName.m' script can be customized to produce
43  % whatever table/plot is desired.
44  %
45  % For discussion about the calculation of noise power on Mars, read
46  %  'abacus1/nasa_scp/Simulations/80211a/PowerDistanceEtc/pwr.pdf'.
47  % and the subsection
48  %  '4. PERFORMANCE AT DIFFERENT SITES'
49  %   'Performance versus distance between the transmitter and the receiver'
50  % in the 2005 IEEE Aero Conference paper:
51  %  'abacus1/nasa_scp/Documents/IEEE_Aero_Conf_2005/PerfEval11Final/docfinal.pdf'.
52  %
53  % 2004, Gaylon R Lovelace, New Mexico State University
54  %
55
56  clear all, format compact, close all;
57  randn('state',sum(100*clock));
58
59  simName = 'pwrCheck';
60
61  diary( [ simName '.txt' ] );   datestr(now)
62
63  skipDemod = 0;  % default to normal demod
64  vsEbNo = 0;  % default to SNR determined from ICS Telecom
65  eval( [ 'RunParams_' simName ] );  % sets an Nx9 cell array with all the simulation paramet
66  runParamNums = cell2mat(runParams(:,1:8));
67  nRuns = size( runParams, 1 );
68
69  magicNums = setMagicNums;          % sets a bunch of 802.11a specific magic numbers
70
71  pktCums = zeros(nRuns,1);   bitCums = pktCums;  errBitCums = pktCums;  nErrs = 0;
72  berrPktCums = pktCums;  perrPktCums = pktCums;  serrPktCums = pktCums;  aerrPktCums = pktCu
```

```
73  rmsDelays = pktCums;  rxPowers = pktCums;  estEbNoDBs = pktCums;
74
75  for runNdx = [ 1 : nRuns ]
76
77          Mbps        = cell2mat( runParams(runNdx,2) );
78          tx_rateMode = Mbps2rateMode( Mbps );
79          tx_codeRate = magicNums.RATE_dependent(magicNums.xMODE(tx_rateMode+1),3);
80          tx_N_BPSC   = magicNums.RATE_dependent(magicNums.xMODE(tx_rateMode+1),4);
81          tx_N_CBPS   = magicNums.RATE_dependent(magicNums.xMODE(tx_rateMode+1),5);
82          EbNoDB      = cell2mat( runParams(runNdx,3) );
83          distance    = cell2mat( runParams(runNdx,4) );
84          tx_PwrScalar= cell2mat( runParams(runNdx,5) );
85          tx_nBytes   = cell2mat( runParams(runNdx,6) );
86          maxPkts     = cell2mat( runParams(runNdx,7) );
87          minErrPkts  = cell2mat( runParams(runNdx,8) );
88          chnlType    = char( runParams(runNdx,9) );
89          pdpFile     = char( runParams(runNdx,10) );
90
91          [ dateStr, V_m ] = readPDP( [ '..\PDP\' pdpFile '.csv' ], chnlType, 10e-9 );
92          V_m = downsamplePDP( V_m, 5, 500 ) .* sqrt(tx_PwrScalar);
93
94
95          % Constants for Mars Propagation
96          c = 3e8;
97          lambda = c / 5.25e9;                    % IEEE 802.11a ==> 5.25 GHz
98          Gr = 1.0;                               % Isotropic Rx Antenna
99          kBoltz = 1.38e-23;
100         Teq = 1560;                             % Equivalent Noise Temp Kelvins
101
102         % Convert from E-Field (V/m) to Rxr Input Voltage RMS:
103         rxVrms = V_m/10 * ( sqrt(Gr) * lambda / (pi*sqrt(480)) );
104         if (vsEbNo)
105                 rxVrms = rxVrms' / sqrt( abs( rxVrms' * rxVrms ) );    % normalize
106         end
107
108         % Ensure at least 64 samples in h(t)
109         if (length(rxVrms)<64);  rxVrms = [ rxVrms(:)', zeros(1,64) ];  rxVrms = rxVrms(1:6
110
111         if (vsEbNo)
112         % tx_wave is normd so  Es = 1.000  -->  reqd_No = 1 / EsNoLinear;
113         EbNoLinear = 10 .^ ( EbNoDB / 10 );
114             awgnSigma = sqrt( 1 / ( EbNoLinear * tx_N_CBPS * tx_codeRate ) );
115                 awgnSigma = awgnSigma * sqrt(80);  % because is done below, for tx_wave
116         else
117                 % Compute noise power on Mars
```

```
118                 awgnSigma = sqrt( kBoltz * Teq * 20e6 / 2 );
119             end
120
121         rmsDelays(runNdx) = rmsDelayCalc( rxVrms, 1/20e6, 1 );
122         rxPowers(runNdx) = ( rxVrms(:)' * rxVrms(:) );
123             rxVrms = rxVrms(:);
124             leadPwr = ( rxVrms(1:16)' * rxVrms(1:16) );
125             tailPwr = ( rxVrms(17:end)' * rxVrms(17:end) );
126             totalPwr = ( rxVrms(:)' * rxVrms(:) );
127             disp( 'leadPwr  tailPwr  totalPwr' );
128             disp( [ leadPwr  tailPwr  totalPwr ] );
129
130             for repNdx = [ 1 : maxPkts ]
131                 disp( sprintf( 'runNdx=%d;  AntHt=%gm;  EbNoDB=%g;  Mbps=%d;  Rep#%d;  %s', runI
132
133                     tx_PSDU   = bitgen( 8*tx_nBytes );
134                     tx_wave   = plcp_11a( tx_rateMode, tx_PSDU );
135
136
137
138                     % Renormalize tx_wave to SampleEnergy=1.0, rather than OFDMSymbolEnergy=1.0
139                     tx_wave = tx_wave * sqrt(80);
140                     h_t = rxVrms;
141
142
143                     % AWGN Channel is just one tap.
144                     %h_t = zeros(1,64);  h_t(1) = 1.0;
145
146                     % Fading Model:  h(t) = h(t) * Rayleigh ampl * uniform phase
147                     h_t = ( sqrt(0.5) * [1 j] * randn(2,length(h_t)) ) .* h_t(:)';
148
149                     rx_wave = conv( tx_wave, h_t );
150                     %rx_wave = tx_wave;
151
152             % For debug purposes, estimate Eb/No
153             estEbNoLinear = ( var(rx_wave) / awgnSigma^2 ) / ( tx_N_CBPS * tx_codeRate / 80 );
154
155                 awgnoise = awgnSigma * sqrt(1/2) * [1 j] * ( randn(2,length(rx_wave)) );
156
157                     rx_wave = rx_wave + awgnoise;
158
159
160                     h_t_hat = magicNums.LSE_mtrx * [rx_wave(1:(magicNums.preambLen)).'];
161                     % Equalization at rxr will require transform of whichever impulse response
162                     H_k_hat = fft( h_t_hat(1:64) );
```

```
163
164
165             % Up to this point, everything is in MKS units (Volts, meters, Watts, etc)
166             % The receiver simulation below has no gain control.  Except for premultipl
167             % tx_wave by sqrt(80) (to make it apples-to-apples with the noise), everyth
168             % done above is un-done by H_k_hat in the receiver.  Scale the rx'd signal
169             % together to 'set the receiver preamp gain'.
170             rx_wave = rx_wave / sqrt(80);     % Only for 'real power' multipath via ICS
171
172             signalStart = 320 + 16 + 1;
173             [ rx_rateMode, rx_nSyms, rx_nBytes, signalErrFlags, signalErrMsg ] = ...
174                 processSIGNAL( rx_wave(signalStart:(signalStart+63)), magicNums, H_k_ha
175
176         rx_rateMode = tx_rateMode;  % for BER purposes, ignore the SIGNAL symbol contents
177         rx_nBytes   = tx_nBytes;
178
179         if (~skipDemod)
180                 [ serviceBits, dataBits ] = processData( rx_wave, rx_rateMode, rx_nBytes, m
181                 end
182
183                 if(~skipDemod), nErrs = sum( dataBits ~= tx_PSDU ); end;
184                 pktCums(runNdx)       = pktCums(runNdx)     + 1;
185                 bitCums(runNdx)       = bitCums(runNdx)     + 8*tx_nBytes;
186                 errBitCums(runNdx)    = errBitCums(runNdx) + nErrs;
187                 berrPktCums(runNdx)   = berrPktCums(runNdx) + (nErrs>0);
188                 perrPktCums(runNdx)   = perrPktCums(runNdx) + bitand( signalErrFlags, 1 );
189                 serrPktCums(runNdx)   = serrPktCums(runNdx) + ( signalErrFlags > 1 );
190                 aerrPktCums(runNdx)   = aerrPktCums(runNdx) + ( (nErrs>0) || (signalErrFlags
191         estEbNoDBs(runNdx)    = estEbNoDBs(runNdx) + estEbNoLinear;  % not in dBs until late
192
193         if( length(signalErrMsg) > 0 )
194                     disp( [ '  ' signalErrMsg ] )
195         end
196
197                 statStr = sprintf( '  This pkt: %d/%d bits.    ', nErrs, 8*tx_nBytes );
198                 statStr = [ statStr sprintf( 'Cumulative: %d/%d errPkts & %d/%d bits.', aer
199         disp( statStr );
200
201                 if ( aerrPktCums(runNdx) >= minErrPkts )
202                         % Have plenty of errors for significance.  Skip remaining packets i
203                         break;
204                 end
205
206         end % repNdx
207
```

```
208         estEbNoDBs(runNdx) = 10 * log10( estEbNoDBs(runNdx) / pktCums(runNdx) );
209
210         %save( sprintf( '%s_%s_curve%02d.mat', simName, chnlType, runParamNums(runNdx,1) ) )
211
212         resultMatrix = [ errBitCums, berrPktCums, perrPktCums, serrPktCums, aerrPktCums, ..
213                          bitCums, pktCums, rmsDelays, rxPowers, estEbNoDBs ];
214         resultTable = dumpResultTable( resultMatrix, runParams );
215         disp( resultTable );
216
217 end % runNdx
218
219 diary off;
220
```

Mbps2rateMode.m

```
1  function rateMode = Mbps2rateMode( Mbps )
2  %MBPS2RATEMODE  Convert 802.11a Mbit/s to a mWLAN rateMode value
3  % 48 54 12 18 24 36  6  9   Mb/s
4  %  0  1  2  3  4  5  6  7   rateMode
5  %
6  % Subfunction for SIM_ER_11A.M
7  %
8  % 2004, Gaylon R Lovelace, New Mexico State University
9  %
10
11 table = [
12 48 54 12 18 24 36  6  9   %      Mb/s
13  0  1  2  3  4  5  6  7   %  rateMode
14 ];
15 ndx = find( Mbps == table(1,:) );
16 rateMode = table(2,ndx);
```

setMagicNums.m

```
1  function [magicNums] = setMagicNums()
2  %SETMAGICNUMS  Returns a struct full of magic numbers for use in 802.11a Rcvr
3  % Subfunction for SIM_ER_11A.M
4  %
5  % 2004, Gaylon R Lovelace, New Mexico State University
6  %
```

```
 7
 8  load( 'tx_wave_preamble' );  tx_wave_preamble = tx_wave_preamble(:);
 9  chnlLen = 64;   preambLen = length(tx_wave_preamble);
10  A = convmtx( tx_wave_preamble, chnlLen );
11  A = A([1:preambLen],:);
12  % don't use the zero-prefixed rows at bottom; they overlap with the SIGNAL symbol.
13
14  magicNums = struct( ...
15  'pnVector', [ ...
16       1  1  1  1 -1 -1 -1  1 -1 -1 -1 -1  1 ...
17       1 -1  1 -1 -1  1  1 -1  1  1 -1  1  1 ...
18       1  1  1  1 -1  1  1  1 -1  1  1 -1 -1 ...
19       1  1  1 -1  1 -1 -1 -1  1 -1  1 -1 -1 ...
20       1 -1 -1  1  1  1  1  1 -1 -1  1  1 -1 ...
21      -1  1 -1  1 -1  1  1 -1 -1 -1  1  1 -1 ...
22      -1 -1 -1  1 -1 -1  1 -1  1  1  1  1 -1 ...
23       1 -1  1 -1  1 -1 -1 -1 -1 -1  1 -1  1 ...
24       1 -1  1 -1  1  1  1 -1 -1  1 -1 -1 -1 ...
25       1  1  1 -1 -1 -1 -1 -1 -1 -1], ...
26  'u64', [0:1:63], ...
27  'i_symbol', [8 22 44 58], ...
28  'i_data', [1:5 7:19 21:26 28:33 35:47 49:53], ...
29  'v_symbol', [1 -1 1 1], ...
30  'v1', [28:1:53], ...
31  'v2', [2:1:27], ...
32  'v3', [1:1:26], ...
33  'v4', [39:1:64], ...
34  'RATE_dependent', [ ...
35     6  1 1/2 1 48  24; ...
36     9  1 3/4 1 48  36; ...
37     12 2 1/2 2 96  48; ...
38     18 2 3/4 2 96  72; ...
39     24 3 1/2 4 192 96; ...
40     36 3 3/4 4 192 144; ...
41     48 4 2/3 6 288 192; ...
42     54 4 3/4 6 288 216], ...
43  'xMODE', [7 8 3 4 5 6 1 2], ...
44  'am', [1./sqrt([1 1 2 2 10 10 42 42])], ...
45  'preambLen', preambLen, ...
46  'LSE_mtrx', inv(A'*A)*A' ...
47  );
48
49  % 'D_hat', [ zeros(1,53) ], ...
50  % 'S_hat', [ zeros(1,48) ], ...
51  % 'kpn', [1], ...
```

```
52  % 'u80', [80], ...
53
54
55
56  %, ...
57  %'tx_wave_preamble', tx_wave_preamble
```

readPDP.m

```
1   function [ dateStr, V_m ] = readPDP( fileName, grepTxt, Ts )
2   %READPDP  Read a delay profile from an ICS Telecom CSV file
3   %    [ dateStr, V_m ] = readPDP( fileName, grepTxt, Ts )
4   %    opens the named file, and reads the comma separated data from
5   %    all lines which begin with the text string GREPTXT.
6   %    DATESTR returns the timestamp from the first line of the file.
7   %    V_M returns the E-Field Intensity profile, converted from dBuV/m
8   %    into linear units of Volts/meter.  Zeros are inserted for all
9   %    missing time intervals, including those prior to the first
10  %    tap in the data.  The elements of V_M correspond to Times-of-
11  %    Arrival of [ 0 : Ts : maxTime ] where maxTime is the maximum ToA
12  %    in the file data.
13  %
14  %    An error will result if any of the Times-of-Arrival given in
15  %    the file are not an integer multiple of Ts.
16  %
17  %    The CSV file should match the following six-column format:
18  %        5/10/2004 10:17,,,,,
19  %
20  %        Subscriber,#,X,Y,FS dBV/m, ToA s
21  %        rx20m1,6,102319.2031,92928,102,0.07
22  %        rx20m1,6,102319.2031,92928,99,0.1
23  %        rx20m1,6,102319.2031,92928,100,0.11
24  %        rx100m1,4,102319.2031,92855.20313,102,0.31
25  %        rx100m1,4,102319.2031,92855.20313,93,0.32
26  %        rx100m1,4,102319.2031,92855.20313,99,0.33
27  %        rx100m1,4,102319.2031,92855.20313,100,0.35
28  %
29  %    For this example, READPDP( fileName, 'rx100m1', 10e-9 ) will return:
30  %    DATESTR = '5/10/2004 10:17';
31  %    V_M = 36x1 array = [ zeros(1,31), 10^(102/20), 10^(93/20), 10^(99/20), 0, 10^(100/20) ]
32  %    TOA_S = 36x1 array = [ 0 : 10e-9 : 350e-9 ]';
33  %
34  %
```

```
35  % Subfunction for SIM_ER_11A.M
36  %
37  % 2004, Gaylon R Lovelace, New Mexico State University
38  %
39
40          fid = fopen( fileName );
41          dateStr = fgetl( fid );
42          dateStr = dateStr(1:(end-5));
43
44  %       fgetl( fid );  fgetl( fid );   % skip blank line & column headers
45
46          state = 2;
47          k = 0;
48          data = repmat( [ -1 -1 ], 8, 1 );   % create 'empty' data array
49          while( state > 0 )
50                  oneLine = fgetl( fid );
51                  if ( oneLine == -1 )
52                          state = 0;
53                  elseif ( isempty( oneLine ) )
54                          % do nothing; skip blank lines
55                  else
56                          c = find( oneLine == ',' );
57                          txt = oneLine(1:(c(1)-1));
58                          if ( strcmpi( txt, grepTxt ) )
59                                  k = k + 1;
60                                  state = 1;
61                                  data(k,1) = str2num( oneLine((c(4)+1):(c(5)-1)) );
62                                  data(k,2) = str2num( oneLine((c(5)+1):end) );
63                          elseif ( state == 1 )
64                                  %% did have 'matching' lines, but now are past them
65                                  state = 0;
66                          end
67                  end
68          end
69
70          fclose( fid );
71
72          data = data(1:k,:);
73
74          t_over_T = 1 + round( ( data(:,2) * 1e-6 ) / Ts );
75          V_m = zeros( max(t_over_T), 1 );
76          V_m(t_over_T) = 10.^( ( data(:,1) / 20 ) - 6 );
77
78
79  return
```

```
80
81
82          %% Example of how to use this fctn
83          lambda = 3e8 / 5.25e9;                  % IEEE 802.11a ==> 5.25 GHz
84          kBoltz = 1.38e23;
85          Teq = 1560;                             % Equivalent Noise Temp Kelvins
86          tx_wave = plcp_11a( rateMode, PSDU );
87          tx_wave = tx_wave * sqrt(80);
88          pdp = readPDP( fileName, tag, 10e-9 );  % Read V/m profile from file.
89          pdp = pdp(min(find(pdp)):end);          % Discard leading zeros
90          % h(t) = pdp * Rayleigh ampl * uniform phase
91          h_t = ( sqrt(0.5) * [ 1 j ] * randn(2,length(pdp)) ) .* pdp;
92          h_t = h_t * ( lambda / (pi*sqrt(480)) );
93          rx_wave = conv( tx_wave, h_t );
94          awgnSigma = sqrt( kBoltz * Teq * 20e6 / 2 );
95      awgnoise = awgnSigma * sqrt(1/2) * [1;j] * ( randn(2,length(tx_wave)) );
96          rx_wave = rx_wave + awgnoise;
```

downsamplePDP.m

```
1   function V_m = downsamplePDP( raw_V_m, M, N )
2   %DOWNSAMPLEPDP  Filter & downsample a delay profile from ICS Telecom
3   %    V_m = readPDP( raw_V_m, M, N )  takes a PDP from readPDP(), and
4   %    lowpass filters it with an Nth order filter, and downsamples it
5   %    by a factor of M.  V_m is normalized to have total energy equal
6   %    to that of raw_V_m.
7   %
8   % Subfunction for SIM_ER_11A.M
9   %
10  % 2004, Gaylon R Lovelace, New Mexico State University
11  %
12
13  myFilt = fir1( N, 1/M );
14
15  first = round( min( find( raw_V_m ) ) + (N/2) );
16  last = round( first + length( raw_V_m ) );
17  V_m = filter( myFilt, 1, [ raw_V_m; zeros(first+N/2,1) ] );
18
19  V_m = V_m(first:M:last);
20
21  V_m = V_m * sqrt( ( raw_V_m' * raw_V_m ) / ( V_m' * V_m ) );
22
```

processData.m

```
1  function [ serviceBits, dataBits ] = processData( rx_wave, rx_rateMode, rx_nBytes, magicNum:
2  %PROCESSDATA Process the 802.11a data symbols
3  % RX_WAVE is the entire received baseband signal, including preamble & SIGNAL symbol.
4  % The actual data symbols start at sample 401.
5  %    320 samples = skip over the PLCP Preamble (16us * 20MHz);
6  %    80 samples skips over the SIGNAL symbol (4.0us * 20MHz)
7  %    1 = MATLAB index is one-based & we want first sample AFTER the SIGNAL symbol
8  % MAGICNUMS = setMagicNums();
9  % H_k = 64-sample FFT of multipath channel impulse response
10 % RX_RATEMODE, RX_NBYTES should come from the SIGNAL symbol, or from knowledge
11 % of the transmitter.
12 % RX_RATEMODE should be  0=48Mbps, 1=54, 2=12, 3=18, 4=24,      5=36, 6=6, 7=9Mbps
13 %                        These are just bits R1-R4 of the SIGNAL field
14 %                        magicNums.xMODE can sort these into ascending bitrate
15 %
16 % Subfunction for SIM_ER_11A.M
17 %
18 % 2004, Gaylon R Lovelace, New Mexico State University
19 %
20
21         rx_codeRate = magicNums.RATE_dependent(magicNums.xMODE(rx_rateMode+1),3);
22         rx_N_BPSC   = magicNums.RATE_dependent(magicNums.xMODE(rx_rateMode+1),4);
23         rx_N_CBPS   = magicNums.RATE_dependent(magicNums.xMODE(rx_rateMode+1),5);
24         rx_AMref    = magicNums.am(           magicNums.xMODE(rx_rateMode+1)  );
25     rx_nSym     = ceil( ( 16 + 8*rx_nBytes + 6 ) / rx_codeRate / rx_N_CBPS );
26
27
28     demod_seq=[];  soft_seq=[];  stepsize=1;
29
30         kpn = 1;
31         samplesPerSymbol = 80;
32         npt = 320 + 16 + 1;  % skip preamble & SIGNAL symbol
33         for k=1:rx_nSym
34             D_hat=zeros(1,53);  S_hat=zeros(1,48);
35             kpn = kpn + 1;
36             npt = npt + samplesPerSymbol;
37             nn = npt + magicNums.u64;
38             z = rx_wave(nn);
39             y = fft(z) ./ H_k(:).';
40             D_hat(magicNums.v1) = y(magicNums.v2);
41             D_hat(magicNums.v3) = y(magicNums.v4);
42             S_hat = D_hat(magicNums.i_data);
43             W = sum( y(magicNums.i_symbol) .* magicNums.v_symbol ) * ( magicNums.pnVector(kp
```

```
44                  Qbits=4; % number of bits for the quantizer
45                  [ Brx soft ] = DeMod_11a( rx_N_BPSC, Qbits, S_hat/rx_AMref );
46                  demod_seq=[demod_seq Brx]; %size(demod_seq)
47                  soft_seq=[soft_seq soft];
48          end
49
50          Rcv_bits = Demod_post_11a( rx_codeRate, rx_N_BPSC, rx_N_CBPS, demod_seq, soft_seq )
51
52          serviceBits = Rcv_bits(1:16);
53          dataBits = Rcv_bits(17:(16+8*rx_nBytes));
54
```

processSIGNAL.m

```
1   function [ rx_rateMode, rx_nSyms, rx_nBytes, signalErrFlags, signalErrMsg ] = processSIGNAL
2   %PROCESSSIGNAL Process the 802.11a SIGNAL symbol
3   % SIGNALsym should probably be 64 simulation samples, beginning at
4   % sample number 320+16+1.
5   %    320 samples = skip over the PLCP Preamble (16us * 20MHz);
6   %    16 samples skips over the cyclic prefix of the SIGNAL symbol (0.8us * 20MHz)
7   %    1 = MATLAB index is one-based & we want first sample AFTER the cyclic prefix
8   % MAGICNUMS = setMagicNums();
9   % H_k = 64-sample FFT of multipath channel impulse response
10  % TX_RATEMODE, TX_NBYTES are compared against contents of SIGNAL sym, to find errs
11  % TX_RATEMODE should be  0=48Mbps, 1=54, 2=12, 3=18, 4=24,      5=36, 6=6, 7=9Mbps
12  %                        These are just bits R1-R4 of the SIGNAL field
13  %                        magicNums.xMODE can sort these into ascending bitrate
14  % SIGNALERRFLAGS is an integer carrying a 3-bit field:
15  %    0x01 =>  parity error in SIGNAL symbol
16  %    0x02 =>  rx_rateMode ~= tx_rateMode
17  %    0x04 =>  rx_nBytes ~= tx_nBytes
18  % SIGNALERRMSG is a text string conveying the errors flagged by SIGNALERRFLAGS
19  %
20  % Subfunction for SIM_ER_11A.M
21  %
22  % 2004, Gaylon R Lovelace, New Mexico State University
23  %
24
25          y = fft(SIGNALsym) ./ H_k(:).';
26          D_hat = zeros(1,64);
27          D_hat(magicNums.v1) = y(magicNums.v2);    % time/freq mapping per 802.11a Fig 109
28          D_hat(magicNums.v3) = y(magicNums.v4);    % dunno why it's done in two steps
29          S_hat = D_hat(magicNums.i_data);          % skip elements from the NULL subcarrs
```

```
30          S_signal = round(real(S_hat));
31          [ rx_rateMode rx_nSyms rx_nBytes rx_prtchkpass ] = parse_signal_11a( S_signal );
32
33          tx_codeRate  = magicNums.RATE_dependent(magicNums.xMODE(tx_rateMode+1),3);
34          tx_N_BPSC    = magicNums.RATE_dependent(magicNums.xMODE(tx_rateMode+1),4);
35          tx_N_CBPS    = magicNums.RATE_dependent(magicNums.xMODE(tx_rateMode+1),5);
36      tx_nSyms     = ceil( ( 16 + 8*tx_nBytes + 6 ) / tx_codeRate / tx_N_CBPS );
37
38              % Just confirm that mWLAN:parse_signal_11a() uses the same nBytes-->nSymbols
39              % formula that I just wrote above.  If nBytes & rateMode are both right, then
40                  % nSymbols should also agree.
41              if ( ( rx_rateMode == tx_rateMode ) && ...
42                   ( rx_nSyms ~= ceil( ( 16 + 8*rx_nBytes + 6 ) / tx_codeRate / tx_N_CBPS ) )
43                          error( [ 'nSymbols calc mismatch:  rx_nSyms=' num2str(rx_nSyms) ...
44                                   '';  rx_nBytes=' num2str(rx_nBytes) ';  rx_rateMode=' num2s
45                  end
46
47      errParity    = ( rx_prtchkpass ~= 1 );
48      errRateMode  = ( rx_rateMode ~= tx_rateMode );
49      errNBytes    = ( rx_nBytes ~= tx_nBytes );
50      signalErrFlags = errParity + 2*errRateMode + 4*errNBytes;
51
52      signalErrMsg = '';
53          signalValuesErr = ( ( rx_rateMode ~= tx_rateMode ) || ( rx_nSyms ~= tx_nSyms ) || (
54      if ( errParity )
55              signalErrMsg = 'Parity.  ';
56      end
57      if ( errRateMode )
58                  signalErrMsg = [ signalErrMsg '   rateMode: tx=' num2str(tx_rateMode) ' rx=
59          end
60      if ( errNBytes )
61                  signalErrMsg = [ signalErrMsg '   nBytes: tx=' num2str(tx_nBytes) ' rx=' nu
62          end
63      if ( signalErrFlags )
64          signalErrMsg = [ 'SIGNAL symbol error:  ' signalErrMsg ];
65      end
66
```

rmsDelayCalc.m

```
1   function [ rmsDelay, meanExcess ] = rmsDelayCalc( profile, Ts, isVoltage )
2   %RMSDELAYCALC  Calculates multipath delay stats for Delay Profile
3   %    [ rmsDelay, meanExcess ] = rmsDelayCalc( profile, Ts, isVoltage )
```

```
4   %     takes a delay profile and calculates the mean excess delay and
5   %     the RMS delay spread.
6   %     If ISVOLTAGE is FALSE or omitted, then the delay profile is
7   %     taken to be power vs time.  If ISVOLTAGE is TRUE, then the
8   %     profile is taken as voltage (or field strength, etc) vs time.
9   %     In either case, the profile is assumed to be on a linear scale,
10  %     rather than dB, and uniformly sampled at rate 1/Ts.
11  %
12  % Subfunction for SIM_ER_11A.M
13  %
14  % 2004, Gaylon R Lovelace, New Mexico State University
15  %
16
17  firstNonzero = min( find( profile ) );
18  taus = [ 0 : 1 : length(profile)-1 ]' .* Ts;
19  tau_0 = taus(firstNonzero);
20  taus = taus - tau_0;
21
22  if ( (nargin<3) || (isVoltage~=0) )
23          pdp = profile.^2;
24  else
25          pdp = profile;
26  end
27
28  meanExcess = sum( pdp(:) .* taus    ) / sum( pdp(:) );
29  tauSqr_bar = sum( pdp(:) .* taus.^2 ) / sum( pdp(:) );
30  rmsDelay = sqrt( tauSqr_bar - meanExcess^2 );
31
```

---

dumpResultTable.m

```
1   function resultTable = dumpResultTable( resultMatrix, runParams )
2   %DUMPRESULTTABLE Produce a wide NxX string matrix with results from a set of sim'n runs
3   % A simulation set consists of N simulation runs.
4   % Each simulation run consists of many data packets.
5   % Each run may in a set may do a different number of pkts.
6   % Each run ends up producing ten scalar results,
7   % so the N runs, together, produce the N-x-10 results matrix,
8   % which is described column-by-column as:
9   % resultMatrix = [ ...
10  %     errBitCums,  ...  % 01 % Total number of bad data bits
11  %     berrPktCums, ...  % 02 % Number of pkts with >= 1 bad data bit
12  %     perrPktCums, ...  % 03 % Number of pkts with bad SIGNAL parity
```

```
13  %     serrPktCums, ...  % 04 % Number of pkts with bad SIGNAL contents
14  %     aerrPktCums, ...  % 05 % Number of pkts with any of above 3 errs
15  %     bitCums,     ...  % 06 % Total number of data bits transferred
16  %     pktCums,     ...  % 07 % Total number of pkts transferred
17  %     rmsDelays,   ...  % 08 % RMS Delay Spread for the channel used
18  %     rxPowers,    ...  % 09 % Avg rx/tx power ratio for the channel used
19  %     estEbNoDBs,  ...  % 10 % Avg rx'd Eb/No for the channel & rateMode used
20  % ];
21  % All the 'instructions' for a simulation set are contained in
22  % a big N-x-9 cell array:    (must be cell array so last two columns can be strings)
23  % runParams = { ...
24  %     CurveNum,    ...  % 01 % Allows e.g. 24 runs to plot as 3 BER curves each 8 points
25  %     Mbps,             ...  % 02 % Used to spec which 802.11a rateMode
26  %     EbNoDB,      ...  % 03 % Independent variable for plots. Set zero if plotting vs dist
27  %     Distance,    ...  % 04 % Independent variable for plots. Set zero if plotting vs EbNo
28  %     pktLen,      ...  % 05 % User data bytes per 802.11a packet
29  %     txPwr,       ...  % 06 % Scales the ICS Telecom PDP (assumed to be for 1.0 W tx pwr).
30  %     MaxPkts,     ...  % 07 % Stop after M pkts, even if found few or no errors
31  %     MinErrPkts,  ...  % 08 % Stop early, if find this many pkts with >= 1 error.
32  %     Channel,     ...  % 09 % 'Subscriber' string ID from ICS Telecom PDP *.csv file
33  %     pdpFile      ...  % 10 % Filename of ICS Telecom PDP *.csv file
34  % };
35  %      1       2       3       4       5       6       7        8        9         10
36  % beCums pbeCums ppeCums pseCums paeCums    nBits   nPkts       BER rmsDelay    rxPower estI
37  %
38  % Subfunction for SIM_ER_11A.M
39  %
40  % 2004, Gaylon R Lovelace, New Mexico State University
41  %
42
43  %resultMatrix = [ errBitCums, berrPktCums, perrPktCums, serrPktCums, aerrPktCums, bitCums, ]
44
45  nRuns = size( resultMatrix, 1 );
46  if ( nRuns ~= size( runParams, 1 ) );  error( 'RESULTMATRIX, RUNPARAMS be same height.' );
47
48  runParamNums = cell2mat( runParams(:,[1:8]) );
49  resultMatrix(:,8) = resultMatrix(:,8) * 1e9;  % Display rmsDelaySpread in nanoSecs
50  % don't % resultMatrix(:,9) = resultMatrix(:,9) * 1e9;  % Display rxPower in nanoWatts
51  nonzeroBits = find( resultMatrix(:,6) );
52  bers = resultMatrix(:,6);  % BERs default to zero (for runs not completed, etc)
53  bers(nonzeroBits) = resultMatrix(nonzeroBits,1) ./ resultMatrix(nonzeroBits,6);
54
55  resultTable = [ '% ' datestr(now) ];
56  resultTable = strvcat( resultTable, '%      1       2       3       4       5       6
57  resultTable = strvcat( resultTable, '% beCums pbeCums ppeCums pseCums paeCums    nBits   nPl
```

```
formatStr = '%8d%8d%8d%8d%8d%9d%8d%10.3g%9.1f %10.3g%8.1f%10d%6d%8.2f%9.4g%9.4g%9d%9d%9d%13
for k = [ 1 : nRuns ]
        tmpStr = [ '''' char(runParams(k,9)) '''' ];
        thisLine = sprintf( formatStr, resultMatrix(k,[1:7]), bers(k), resultMatrix(k,[8:10
        resultTable = strvcat( resultTable, thisLine );
end

```

RunParams_BERDistAvg.m

This last file is one sample parameter file. Similar files are used to define all the different simulation sets we did. This one in particular sets up a long set of simulations to find BER-vs-(Tx-Rx Distance) averaged over several spots at the Gusev1, Site1 location.

```
runParams = {
%       1       2       3         4         5           6         7          8         9           10
% CurveNum   Mbps  EbNoDB  Distance   txPwr      pktLen   MaxPkts  MinErrPkts  Channel       pdpF
        1      12       0      1000    0.001        128       500      10000   'rx1000m1'   'DistA
        1      12       0       500    0.001        128       500      10000    'rx500m1'   'DistA
        1      12       0       200    0.001        128       500      10000    'rx200m1'   'DistA
        1      12       0       100    0.001        128       500      10000    'rx100m1'   'DistA
        1      12       0        50    0.001        128       500      10000     'rx50m1'   'DistA
        1      12       0        20    0.001        128       500      10000     'rx20m1'   'DistA
%
        2      12       0      1000    0.001        128       500      10000   'rx1000m2'   'DistA
        2      12       0       500    0.001        128       500      10000    'rx500m2'   'DistA
        2      12       0       200    0.001        128       500      10000    'rx200m2'   'DistA
        2      12       0       100    0.001        128       500      10000    'rx100m2'   'DistA
        2      12       0        50    0.001        128       500      10000     'rx50m2'   'DistA
        2      12       0        20    0.001        128       500      10000     'rx20m2'   'DistA
%
        3      12       0      1000    0.001        128       500      10000   'rx1000m3'   'DistA
        3      12       0       500    0.001        128       500      10000    'rx500m3'   'DistA
        3      12       0       200    0.001        128       500      10000    'rx200m3'   'DistA
        3      12       0       100    0.001        128       500      10000    'rx100m3'   'DistA
        3      12       0        50    0.001        128       500      10000     'rx50m3'   'DistA
        3      12       0        20    0.001        128       500      10000     'rx20m3'   'DistA
%
        4      12       0      1000    0.001        128       500      10000   'rx1000m4'   'DistA
        4      12       0       500    0.001        128       500      10000    'rx500m4'   'DistA
        4      12       0       200    0.001        128       500      10000    'rx200m4'   'DistA
        4      12       0       100    0.001        128       500      10000    'rx100m4'   'DistA
        4      12       0        50    0.001        128       500      10000     'rx50m4'   'DistA
```

```
30        4       12        0        20       0.001    128      500      10000       'rx20m4'    'DistA'
31    };
32    skipDemod = 0;  % demodulate data to find BER
33    vsEbNo = 0;  % SNR determined from ICS Telecom
34
```

IEEE 802.11b Simulation Code

MultipathEbNoSim.m

```
1   %Program to simulate BER as a function of Eb/No using Martian Multipath
2   %Data
3
4   % IEEE 802.11a/b/g Toolbox, v.3.02, March 2004
5   % All rights reserved, (c) CommAccess Technologies, 2001-2004.
6
7   % Modified by Dr. Deva K. Borah, NMSU
8   % Modified by Anirudh Daga, NMSU
9
10  clear;format compact; close all;clc;
11  opt_mode=1;% opt_mode = 1 for 1 Mbps, 2 for 2 Mbps, 3 for 5.5 Mbps, 4 for 11 Mbps
12  rake = 0;
13  barker=[1 -1 1 1 -1 1 1 1 -1 -1 -1];
14  CCK_PBCCtx=0;% 0 for CCK, 1 for PBCC
15  cckdemodtype=1; %0 for suboptimal CCK demod, 1 for near-optimal
16  coherent=0;% 1 for coherent demod, 0 for otherwise
17
18  p=18;% p = the number of sample per channel symbol
19  chiprate=11;  % Mega cps
20  deltaT=1/chiprate/p;
21  nsample=p;
22
23  %Pulse shaping sequence
24  Nspan=5;
25  ns=p;
26  alpha=0.35;
27  hTx=SRRC(alpha,ns,Nspan);
28  hRx=hTx/nsample;  % This scaling operation is simply for the purpose of plotting.
29
30  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
31  %                ICS TELECOM POWER DELAY PROFILE INCORPORATION                  %
32  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
33  t1 = 'C:\DocumentsPapers\NASA Documents\PDPSimulationDataOct2004\Gusev3_Site1_100m.txt';
```

```
34  % t2 = 'C:\DocumentsPapers\NASA Documents\PDPSimulationDataOct2004\Gusev3_Site2_100m.txt';
35  % t3 = 'C:\DocumentsPapers\NASA Documents\PDPSimulationDataOct2004\Gusev1_Site1_100m.txt';
36  files = strvcat(t1);
37  for kk = 1:size(files,1)
38  PowerDelayProfile = dlmread(files(kk,:));
39  ImpulseResponse = PDP2IMP(PowerDelayProfile)';
40  ImpulseResponse = Missing_Time_Insertion(ImpulseResponse);
41  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
42  %              OVERSAMPLING THE MULTIPATH CHANNEL IMPULSE RESPONSE              %
43  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
44  ImpulseResponse = (ImpulseResponse(:,1)/norm(ImpulseResponse(:,1)))';
45  hf1 = resample(ImpulseResponse,99,50,1);
46  hf1 = hf1/norm(hf1);
47  IRLength = length(ImpulseResponse);
48
49  if(opt_mode>4), opt_mode=4; end
50  if(opt_mode<1), opt_mode=1; end  % protection
51
52  % start waveform generation
53  L_S=1;  % 1 for the long preamble, 0 for the short preamble
54
55  if (opt_mode==1) Rate_modetx=10; end;
56  if (opt_mode==2) Rate_modetx=20; end;
57  if (opt_mode==3) Rate_modetx=55; end;
58  if (opt_mode==4) Rate_modetx=110; end;
59
60  if (opt_mode==3)
61      [codevec55,bitvalvec55,phchodd55,phchevn55,phvec55]=cck55demodsupport;
62  end;
63
64  if (opt_mode==4)
65      [codevec,bitvalvec,phchodd,phchevn,phvec]=cck11demodsupport;
66  end;
67
68  if(CCK_PBCCtx == 1)
69      ph0=pi;
70  end % for (PBCC) BER simulation purpose
71
72  nByte=100;
73  snrdb = 1000;
74  for xx = 1:length(snrdb)
75      tic
76      bitcounter=0;
77      wrongcounter=0;
78
```

```
79      for yy = 1:1000
80          tic
81          files(kk,:)
82          nBittx=nByte*8;
83          PSDU=BitGen(nBittx);
84          PSDULength = length(PSDU);
85          b23=[0 CCK_PBCCtx];
86          %Generate the scrambled spread data
87          [IQsequence,xBit,ph0] = PLCP_11b(opt_mode,L_S,PSDU,b23);
88          if (rake == 1)
89              TxdSpreadPreamble = IQsequence(1:128*11);
90          end
91          nIQ=length(IQsequence);
92
93          if(b23(2)==1)
94              tap_11b=[1 0 1 1 0 1 1 1 1 1 1 1 0 1];
95              VitrB_ini(tap_11b)
96          end
97
98          %Generate Rayleigh fading
99
100         rayleigh_fading = (1/sqrt(2))*(randn(size(hf1)) + j*randn(size(hf1)));
101         hf = hf1.*rayleigh_fading;
102         CIR = conv(conv(hf,hTx),hRx);
103         CIR = CIR(10*ns+1:ns:length(CIR)-10*ns);
104
105         % generate Tx waveform
106
107         S=zeros(1,ns*nIQ);
108         S(1:ns:ns*nIQ)=IQsequence;
109         S=conv(S,hTx);
110         S1=conv(S,hf);
111
112         % Add noise to signal samples
113
114         nx=length(S1);
115
116         if (opt_mode==1) noisestd=sqrt(p*11/(2*10^(snrdb(xx)/10))); end;
117         if (opt_mode==2) noisestd=sqrt(p*11/(4*10^(snrdb(xx)/10))); end;
118         if (opt_mode==3) noisestd=sqrt(p/(10^(snrdb(xx)/10))); end;
119         if (opt_mode==4) noisestd=sqrt(p/(2*10^(snrdb(xx)/10))); end;
120
121         wi=randn(1,nx)*noisestd;
122         wq=randn(1,nx)*noisestd;
123
```

```
124         if (opt_mode == 1) S1 = S1 + wi; else S1 = S1+(wi+j*wq); end;
125
126         % Receiver section starts
127
128         Sr=conv(S1,hRx);
129         ExPar_11b1
130         rcv1 = Sr(10*ns+1:ns:length(Sr)-10*ns);
131         nr=length(rcv1);
132
133         if (rake == 1)
134             RxdSpreadPreamble = rcv1(1:length(TxdSpreadPreamble)+length(rcv1)-nIQ);
135             hhat = ChannelEst_NoFor(RxdSpreadPreamble,TxdSpreadPreamble);
136             [maxval,maxind]=max(hhat);
137             k_header_end = (144+48)*11+1;
138         elseif (rake == 0)
139             [maxval,maxind] = max(CIR);
140             k_header_end=(144+48)*11+1+(maxind-1);
141         end
142
143
144
145         rcv3=rcv1(k_header_end:length(rcv1));
146
147         if ((Rate_modetx==55)&(cckdemodtype==1))
148
149             if (rake == 1)
150                 d_rcv = CCK55DemodRake_NoFor(rcv3,nBittx,codevec55,bitvalvec55,...
151                         phchodd55,phchevn55,phvec55,ph0,PSDULength,hhat);
152             elseif (rake == 0)
153                 d_rcv = cck55demod(rcv3,nBittx,codevec55,bitvalvec55,phchodd55,...
154                         phchevn55,phvec55,ph0);
155             end
156
157         elseif ((Rate_modetx==110)&(cckdemodtype==1))
158
159             if (rake == 1)
160                 d_rcv=CCK11DemodRake_NoFor(rcv3,nBittx,codevec,bitvalvec,...
161                       phchodd,phchevn,phvec,ph0,PSDULength,hhat);
162             elseif (rake == 0)
163                 d_rcv=cck11demod(rcv3,nBittx,codevec,bitvalvec,phchodd,...
164                       phchevn,phvec,ph0);
165             end
166         else
167
168             if (rake == 1)
```

```
169                 d_rcv=Demod_11bv4(Rate_modetx,nBittx,ph0,CCK_PBCCtx,coherent,...
170                       rcv3,hhat,PSDULength);
171             elseif (rake == 0)
172                 d_rcv=Demod_11bv2(Rate_modetx,nBittx,ph0,CCK_PBCCtx,coherent,rcv3);
173             end
174
175         end;
176
177         dd_rcv=Descrambler_11b(d_rcv(1:1:nBittx));
178         Err=(dd_rcv(1:nBittx)~=PSDU);
179         ErrorKnt=sum(Err);
180         bitcounter = bitcounter + nBittx;
181         wrongcounter = wrongcounter + ErrorKnt
182         ber = wrongcounter/bitcounter
183         yy
184         snrdb(xx)
185          if (snrdb(xx)<10 && wrongcounter >= 20000)
186              break;
187         end
188         if (snrdb(xx)>10 && wrongcounter >= 10000)
189             break;
190         end
191         toc
192     end
193     finber(xx) = ber;
194     toc
195 end
196 BitErrRate(kk,:) = finber;
197 end
```

---

mainprog_11b_mtdist.m

```
1  % Program to simulate BER as a function of distance using Martian multipath data
2
3  % IEEE 802.11a/b/g Toolbox, v.3.02, March 2004
4  % All rights reserved, (c) CommAccess Technologies, 2001-2004.
5
6  % Modified by Dr. Deva K. Borah, NMSU
7  % Modified by Anirudh Daga, NMSU
8
9  clear all; format compact; close all;clc;
10 barker=[1 -1 1 1 -1 1 1 1 -1 -1 -1];
11 rake = 1;
```

```
12  opt_mode=4;% opt_mode = 1 for 1 Mbps, 2 for 2 Mbps, 3 for 5.5 Mbps, 4 for 11 Mbps
13  CCK_PBCCtx=0;% 0 for CCK, 1 for PBCC
14  cckdemodtype=1; %0 for suboptimal CCK demod, 1 for near-optimal
15  coherent=0;% 1 for coherent demod, 0 for otherwise
16
17  p=18;% p = the number of sample per channel symbol
18  chiprate=11;  % Mega cps
19  deltaT=1/chiprate/p;
20  nsample=p;
21  % Pulse shaping sequence
22
23  Nspan=5;
24  ns=p;
25  alpha=0.35;
26  hTx=SRRC(alpha,ns,Nspan);
27  hRx=hTx/nsample;
28  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
29  %                ICS TELECOM POWER DELAY PROFILE INCORPORATION                    %
30  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
31  PowerDelayProfile = dlmread('C:\DocumentsPapers\NASA Documents\...
32                                PDPSimulationDataOct2004\Gusev1_Site1_100m.txt');
33  [ImpulseResponse,TruePower] = PDP2IMPDistance(PowerDelayProfile);
34  ImpulseResponse = Missing_Time_Insertion(ImpulseResponse);
35  ImpulseResponse = ImpulseResponse(:,1)';
36
37  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
38  %               OVERSAMPLING THE MULTIPATH CHANNEL IMPULSE RESPONSE              %
39  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
40  hf1 = resample(ImpulseResponse,99,50,1);
41  hf1 = (norm(ImpulseResponse)/norm(hf1)).*hf1;
42  IRLength = length(ImpulseResponse);
43
44  if(opt_mode>4), opt_mode=4; end
45  if(opt_mode<1), opt_mode=1; end  % protection
46
47  if (opt_mode==1) Rate_modetx=10; end;
48  if (opt_mode==2) Rate_modetx=20; end;
49  if (opt_mode==3) Rate_modetx=55; end;
50  if (opt_mode==4) Rate_modetx=110; end;
51
52  if (opt_mode==3)
53      [codevec55,bitvalvec55,phchodd55,phchevn55,phvec55]=cck55demodsupport;
54  end;
55
56  if (opt_mode==4)
```

```
57      [codevec,bitvalvec,phchodd,phchevn,phvec]=cck11demodsupport;
58  end;
59
60  if(CCK_PBCCtx == 1),
61      ph0=pi;
62  end % for (PBCC) BER simulation purpose
63
64  nByte=500;
65  bitcounter=0;
66  wrongcounter=0;
67
68  for yy = 1:5000
69      randn('state',sum(100*clock))
70      tic
71      % start waveform generation
72      L_S=1;  % 1 for the long preamble, 0 for the short preamble
73      nBittx=nByte*8;
74      PSDU=BitGen(nBittx);
75      PSDULength = length(PSDU);
76      b23=[0 CCK_PBCCtx];
77
78      [IQsequence,xBit,ph0]=PLCP_11b(opt_mode,L_S,PSDU,b23);
79      TxdSpreadPreamble = IQsequence(1:128*11);
80      nIQ=length(IQsequence);
81
82      if(b23(2)==1)
83          tap_11b=[1 0 1 1 0 1 1 1 1 1 1 1 0 1];
84          VitrB_ini(tap_11b)
85      end
86
87      %Generate Rayleigh fading
88
89      rayleigh_fading = (1/sqrt(2))*(randn(size(hf1)) + j*randn(size(hf1)));
90      hf = hf1.*rayleigh_fading;
91      CIR = conv(conv(hf,hTx),hRx);
92      CIR = CIR(10*ns+1:ns:length(CIR)-10*ns);
93
94      % generate Tx waveform
95
96      S=zeros(1,ns*nIQ);
97      S(1:ns:ns*nIQ)=IQsequence;
98      S=conv(S,hTx);
99      S=sqrt(1e-3)*S;
100     S1=conv(S,hf);
101     nx=length(S1);
```

```
102
103     % Add noise to signal samples
104     sigma = 4.8663e-7;
105     wi=randn(1,nx)*sigma; wq=randn(1,nx)*sigma;
106     if (opt_mode == 1) S2 = S1 + wi; else S2 = S1 + (wi+j*wq);
107
108     %Receiver section starts
109     Sr=conv(S2,hRx);
110     rcv1 = Sr(10*ns+1:ns:length(Sr)-10*ns);
111     if (rake == 1)
112         RxdSpreadPreamble = rcv1(1:length(TxdSpreadPreamble)+length(rcv1)-nIQ-1);
113         hhat = ChannelEst_NoFor(RxdSpreadPreamble,TxdSpreadPreamble);
114         [maxval,maxind]=max(hhat);
115     elseif (rake == 0)
116         [maxval,maxind] = max(abs(CIR));
117     end
118     if (rake == 0)
119         k_header_end=(144+48)*11+1+(maxind-1);
120     elseif (rake == 1)
121         k_header_end=(144+48)*11+1;
122     end
123     nr=length(rcv1);
124
125     rcv3=rcv1(k_header_end:length(rcv1));
126
127     if ((Rate_modetx==55)&(cckdemodtype==1))
128         if (rake == 0)
129             [d_rcv] = cck55demod(rcv3,nBittx,codevec55,bitvalvec55,...
130                       phchodd55,phchevn55,phvec55,ph0);
131         elseif (rake == 1)
132             [d_rcv] = CCK55DemodRake_NoFor(rcv3,nBittx,codevec55,bitvalvec55,...
133                       phchodd55,phchevn55,phvec55,ph0,PSDULength,hhat);
134         end
135
136     elseif ((Rate_modetx==110)&(cckdemodtype==1))
137         if (rake == 0)
138             [d_rcv]=cck11demod(rcv3,nBittx,codevec,bitvalvec,phchodd,...
139                    phchevn,phvec,ph0);
140         elseif (rake == 1)
141             [d_rcv]=CCK11DemodRake_NoFor(rcv3,nBittx,codevec,bitvalvec,...
142                    phchodd,phchevn,phvec,ph0,PSDULength,hhat);
143         end
144     else
145         if (rake == 0)
146             d_rcv=Demod_11bv2(Rate_modetx,nBittx,ph0,CCK_PBCCtx,coherent,rcv3);
```

```
147           elseif (rake == 1)
148               d_rcv=Demod_11bv4(Rate_modetx,nBittx,ph0,CCK_PBCCtx,coherent,...
149                     rcv3,hhat,PSDULength);
150           end
151       end;
152
153       dd_rcv=Descrambler_11b(d_rcv(1:1:nBittx));
154       Err=(dd_rcv(1:nBittx)~=PSDU);
155       ErrorKnt=sum(Err);
156       bitcounter = bitcounter + nBittx;
157       wrongcounter = wrongcounter + ErrorKnt
158       ber = wrongcounter/bitcounter
159       yy
160
161       if (wrongcounter >= 10000)
162           break;
163       end
164
165       toc
166   end
167
```

PERDistanceRake_NoFor.m

```
1   clear all; format compact; close all;clc;
2
3
4   % IEEE 802.11a/b/g Toolbox, v.3.02, March 2004
5   % All rights reserved, (c) CommAccess Technologies, 2001-2004.
6
7   % Modified by Anirudh Daga, NMSU
8
9   %
10  opt_mode=1;% opt_mode = 1 for 1 Mbps, 2 for 2 Mbps, 3 for 5.5 Mbps, 4 for 11 Mbps
11  rake = 0;
12  CCK_PBCCtx=0;% 0 for CCK, 1 for PBCC
13  cckdemodtype=1; %0 for suboptimal CCK demod, 1 for near-optimal
14  coherent=0;% 1 for coherent demod, 0 for otherwise
15
16  p=18;% p = the number of sample per channel symbol
17  chiprate=11;  % Mega cps
18  deltaT=1/chiprate/p;
19  nsample=p;
```

```
% Pulse shaping sequence

Nspan=5;
ns=p;
alpha=0.35;
hTx=SRRC(alpha,ns,Nspan);
hRx=hTx/nsample;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    ICS TELECOM POWER DELAY PROFILE INCORPORATION            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PowerDelayProfile = dlmread('C:\DocumentsPapers\NASA Documents\...
                        PDPSimulationDataOct2004\Gusev1_Site1_1000m.txt');
[ImpulseResponse,TruePower] = PDP2IMPDistance(PowerDelayProfile);
TruePower
ImpulseResponse = Missing_Time_Insertion(ImpulseResponse);
ImpulseResponse = ImpulseResponse(:,1)';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                 OVERSAMPLING THE MULTIPATH CHANNEL IMPULSE RESPONSE           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
hf1 = resample(ImpulseResponse,99,50,1);
hf1 = (norm(ImpulseResponse)/norm(hf1)).*hf1;
IRLength = length(ImpulseResponse);

if(opt_mode>4), opt_mode=4; end
if(opt_mode<1), opt_mode=1; end  % protection

if (opt_mode==1) Rate_modetx=10; end;
if (opt_mode==2) Rate_modetx=20; end;
if (opt_mode==3) Rate_modetx=55; end;
if (opt_mode==4) Rate_modetx=110; end;

if (opt_mode==3)
    [codevec55,bitvalvec55,phchodd55,phchevn55,phvec55]=cck55demodsupport;
end;

if (opt_mode==4)
    [codevec,bitvalvec,phchodd,phchevn,phvec]=cck11demodsupport;
end;

if(CCK_PBCCtx == 1),
    ph0=pi;
end % for (PBCC) BER simulation purpose

nByte=50;
```

```
65  packetcounter=0;
66  wrongcounter=0;
67
68  for yy = 1:1000
69          % start waveform generation
70      L_S=1;  % 1 for the long preamble, 0 for the short preamble
71      nBittx=nByte*8;
72      PSDU=BitGen(nBittx);
73      PSDULength = length(PSDU);
74      b23=[0 CCK_PBCCtx];
75
76      [IQsequence,xBit,ph0]=PLCP_11b(opt_mode,L_S,PSDU,b23);
77      TxdSpreadPreamble = IQsequence(1:128*11);
78      nIQ=length(IQsequence);
79
80      if(b23(2)==1)
81          tap_11b=[1 0 1 1 0 1 1 1 1 1 1 1 0 1];
82          VitrB_ini(tap_11b)
83      end
84
85      %Generate Rayleigh fading
86
87      rayleigh_fading = (1/sqrt(2))*(randn(size(hf1)) + j*randn(size(hf1)));
88      hf = hf1.*rayleigh_fading;
89      CIR = conv(conv(hf,hTx),hRx);
90      CIR = CIR(10*ns+1:ns:length(CIR)-10*ns);
91      %
92      % generate Tx waveform
93
94      S=zeros(1,ns*nIQ);
95      S(1:ns:ns*nIQ)=IQsequence;
96      S=conv(S,hTx);
97      S1=conv(S,hf);
98      nx=length(S1);
99
100
101     % Add noise to signal samples
102     sigma = 4.8663e-7;
103     wi=randn(1,nx)*sigma; wq=randn(1,nx)*sigma;
104
105     if (opt_mode == 1) S2 = S1 + wi; else S2=S1+(wi+j*wq);
106
107     %Receiver section starts
108     Sr=conv(S2,hRx);
109
```

```
110       ExPar_11b1
111
112       rcv1 = Sr(10*ns+1:ns:length(Sr)-10*ns);
113
114       if (rake == 1)
115           RxdSpreadPreamble = rcv1(1:length(TxdSpreadPreamble)+length(rcv1)-nIQ);
116           hhat = ChannelEst_NoFor(RxdSpreadPreamble,TxdSpreadPreamble);
117           [maxval,maxind] = max(hhat);
118       elseif (rake == 0)
119           [maxval,maxind]=max(CIR);
120       end
121
122       if (rake == 0)
123           rcv1 = rcv1(1:2112);
124       elseif (rake == 1)
125           rcv2 = rcv1(1:2112 + length(hhat) -1);
126           temp = repmat((1:length(hhat))',1,2112) + repmat(0:2112-1,length(hhat),1);
127           rcv3 = conj(repmat(hhat(:),1,2112)).*rcv2(temp);
128           rcv1 = sum(rcv3);
129       end
130
131       nr=length(rcv1);
132
133       ExPar_11b2v2
134
135       z=(T==1);
136
137       b=0;
138
139       signal=(z(b+1:b+8)==1);
140
141       Rate_mode=sum(signal .* (2 .^ (7:-1:0)));
142
143       service=(z(b+9:b+16)==1);
144
145       CCK_PBCCrx=service(4);
146
147       leng=(z(b+17:b+32)==1);
148
149       CRC=(z(b+33:b+48)==1);%[PPDU(177:192)]
150
151       CRC=ones(1,16)-CRC;
152
153       T=[signal service leng CRC];
154
```

```
155       checksum=CRCck_11b(T)
156
157       if(checksum ~=0)
158           wrongcounter = wrongcounter + 1;
159       end
160       wrongcounter
161       packetcounter = packetcounter + 1
162       per = wrongcounter/packetcounter
163       if wrongcounter > 1000; break; end;
164       yy
165   end
166
```

PDP2IMP.m

```
1   %Function to convert Martian PDP into channel impulse response
2   %By Anirudh Daga, NMSU
3
4   function [ImpulseResponse] = PDP2IMP(PowerDelayProfile)
5
6   PowerDelayProfile(:,1) = PowerDelayProfile(:,1) - 144.82;
7   PowerDelayProfile(:,1) = (1e-3).*(10.^(.1*(PowerDelayProfile(:,1))));
8   PowerDelayProfile(:,2) = PowerDelayProfile(:,2) - PowerDelayProfile(1,2);
9   PowerDelayProfile = PowerDelayProfile';imp = zeros(size(PowerDelayProfile));
10  imp(1,:) = sqrt(PowerDelayProfile(1,:));imp(2,:) = PowerDelayProfile(2,:);
11  ImpulseResponse = imp;
```

PDP2IMPDistance.m

```
1   %Function to obtain channel impulse response and true power from Martian PDPs at a given
2   %distance
3   %By Anirudh Daga, NMSU
4
5   function [ImpulseResponse,TruePower] = PDP2IMPDistance(PowerDelayProfile)
6
7   PowerDelayProfile(:,1) = PowerDelayProfile(:,1) - 144.82;
8   PowerDelayProfile(:,1) = (1e-3).*(10.^(.1*(PowerDelayProfile(:,1))));
9   TruePower = sum(PowerDelayProfile(:,1));
10  PowerDelayProfile = PowerDelayProfile';
11  imp = zeros(size(PowerDelayProfile));
12  imp(1,:) = sqrt(PowerDelayProfile(1,:));
```

```
13  imp(2,:) = PowerDelayProfile(2,:);
14  ImpulseResponse = imp';
```

Missing_Time_Insertion.m

```
1   %Missing time insertion to fill in zeros
2   %By Anirudh Daga, NMSU
3
4   function [PowerDelayProfile] = Missing_Time_Insertion(pdp)
5
6   pdp(:,2) = rounddec(pdp(:,2),2);
7   b(:,2) = [pdp(1,2):0.01:pdp(end,2)]';
8   b(:,2) = rounddec(b(:,2),2);
9   [dum,idx] = ismember(pdp(:,2),b(:,2));
10  b(idx,1) = pdp(:,1);
11  PowerDelayProfile = b;
```

rmsdelayspread.m

```
1   % Program to calculate RMS Delay Spread from PDP
2   clear; close all; clc;format long;
3
4   pdp = dlmread('C:\DocumentsPapers\NASA Documents\...
5         PDPSimulationDataOct2004\AntennaHeight\Gusev1...
6         Site3\Gusev1_Site3_2m_11b.txt');
7   pdp(:,1) = pdp(:,1)-144.82;
8   pdp(:,1) = (1e-3).*(10.^(.1*(pdp(:,1))));
9   pdp = pdp.';
10  pdp(2,:) = (1e-6)*(pdp(2,:));
11  taubar = sum(pdp(1,:).*pdp(2,:))/sum(pdp(1,:));
12  tausqrbar = sum(pdp(1,:).*(((pdp(2,:)).^2)))/sum(pdp(1,:));
13  rmsdsp = sqrt(tausqrbar - (taubar^2))
```

ChannelEst_NoFor.m

```
1   % Channel Estimation
2   % By Anirudh Daga, NMSU
3
4
5   function [hhat] = ChannelEst_NoFor(RxdSpreadPreamble,TxdSpreadPreamble)
```

```
6
7   n = length(TxdSpreadPreamble);
8   phi = zeros(1,n);
9
10  for j= 0:n-1
11     ind = 1 : n-j;
12     phi(j+1) = sum(TxdSpreadPreamble(ind) .* TxdSpreadPreamble(ind + j));
13  end
14
15
16  autocorr = toeplitz(phi);
17
18  IRLengthChipRate = length(RxdSpreadPreamble)-length(TxdSpreadPreamble)+1;
19
20  temp = repmat((1:IRLengthChipRate)',1,length(TxdSpreadPreamble))...
21         + repmat(0:length(TxdSpreadPreamble)-1,IRLengthChipRate,1);
22  path = RxdSpreadPreamble(temp);
23  corr = (path*TxdSpreadPreamble.').';
24
25  Rbb = autocorr(1:IRLengthChipRate,1:IRLengthChipRate);
26  hhat = (inv(Rbb)*corr')';
27
```

---

cck11demodsupport.m

```
1   %By Dr. Deva K. Borah, NMSU
2   %Please do not distribute without the permission of the author
3   function [codevec,bitvalvec,phchodd,phchevn,phvec]=cck11demodsupport(void);
4   totalbitvecs=2^8;
5   veccntr=zeros(1,8);
6   veccntr(1)=-1;
7   for hypono=1:totalbitvecs
8       veccntr(1)=veccntr(1)+1;
9        if (veccntr(1)>=2)
10       veccntr(1)=0; flowing=1;
11       for jj=2:8
12       if (flowing==1)
13       veccntr(jj)=veccntr(jj)+1;
14       if (veccntr(jj)>=2) veccntr(jj)=0; flowing=1; else flowing=0; end;
15       end; %if flowing==1
16       end; %for jj
17       end; %if veccntr(1)>=2
18    bitvalvec(hypono,:)=veccntr;
```

```
19
20  %Generate QPSK symbols
21  for ii=1:4
22  bitval=2*bitvalvec(hypono,2*(ii-1)+1)+bitvalvec(hypono,2*ii);
23  switch bitval
24  case 0, phvec(hypono,ii)=0;
25  case 1, phvec(hypono,ii)=pi/2;
26  case 2, phvec(hypono,ii)=pi;
27  case 3, phvec(hypono,ii)=3*pi/2;
28  end; %switch
29  end; %for ii
30  codevec(hypono,1:8)=[exp(j*(phvec(hypono,1)+phvec(hypono,2)+phvec(hypono,3)+...
31          phvec(hypono,4))), exp(j*(phvec(hypono,1)+phvec(hypono,3)+...
32          phvec(hypono,4))), exp(j*(phvec(hypono,1)+phvec(hypono,2)+...
33          phvec(hypono,4))), -1*exp(j*(phvec(hypono,1)+phvec(hypono,4))), ...
34          exp(j*(phvec(hypono,1)+phvec(hypono,2)+phvec(hypono,3))), ...
35          exp(j*(phvec(hypono,1)+phvec(hypono,3))), -1*exp(j*(phvec(hypono,1)+...
36          phvec(hypono,2))), exp(j*phvec(hypono,1))];
37   end; %for bitno
38
39  %Define even and odd phase changes
40  phchevn(1)=0;
41  phchevn(2)=pi/2;
42  phchevn(3)=3*pi/2;
43  phchevn(4)=pi;
44  phchodd=phchevn+pi;
45
```

cck11demod.m

```
1   % CCK - 11 Mbps near optimal demodulation
2
3   %By Dr. Deva K. Borah, NMSU
4   %Please do not distribute without the permission of the author
5
6   function [d_rcv]=cck11demod(rcv3,nBittx,codevec,bitvalvec,phchodd,phchevn,phvec,ph0);
7
8
9   lblocks=nBittx/8;
10  prevfirstph=0;
11  rcv3 = rcv3(1:nBittx);
12  rcv3=rcv3.*exp(-j*ph0);
13  codevec = round(codevec);
```

```
14  for ii=1:lblocks
15      testvec=rcv3((ii-1)*8+1:ii*8);
16      testmat=repmat(testvec,256,1);
17      testmatstatus=conj(testmat).*codevec;
18      [maxval,maxind]=max(sum(real(testmatstatus).'));
19      detbits((ii-1)*8+3:ii*8)=bitvalvec(maxind,3:8);
20      firstphtx=phvec(maxind,1);
21      phchange=firstphtx-prevfirstph;
22      if (mod(ii,2)==0)
23          [minval,minind]=min(abs(exp(j*phchodd)-exp(j*phchange)).^2);
24      else
25          [minval,minind]=min(abs(exp(j*phchevn)-exp(j*phchange)).^2);
26      end; %if
27
28      secbit=mod(minind-1,2);
29      firstbit=floor((minind-1)/2);
30      detbits((ii-1)*8+1:(ii-1)*8+2)=[firstbit,secbit];
31      prevfirstph=firstphtx;
32
33  end; %for ii
34  d_rcv=detbits;
```

cck55demodsupport.m

```
1   %By Dr. Deva K. Borah, NMSU
2   %Modified by Anirudh Daga, NMSU
3   %Please do not distribute without the permission of the author
4
5   function [codevec,bitvalvec,phchodd,phchevn,phvec]=cck55demodsupport(void);
6
7   totalbitvecs=2^4;
8   bitvalvec = de2bi([0:totalbitvecs-1]);
9   for hypono=1:totalbitvecs
10  %Generate QPSK symbols
11          bitval=2*bitvalvec(hypono,1)+bitvalvec(hypono,2);
12          switch bitval
13              case 0, phvec(hypono,1)=0;
14              case 1, phvec(hypono,1)=pi/2;
15              case 2, phvec(hypono,1)=3*pi/2;
16              case 3, phvec(hypono,1)=pi;
17          end; %switch
18      phvec(hypono,2) = pi*bitvalvec(hypono,3) + pi/2;
19      phvec(hypono,3) = 0;
```

```
20      phvec(hypono,4) = pi*bitvalvec(hypono,4);
21      codevec(hypono,1:8)=[exp(j*(phvec(hypono,1)+phvec(hypono,2)+phvec(hypono,3)+...
22              phvec(hypono,4))), exp(j*(phvec(hypono,1)+phvec(hypono,3)+...
23              phvec(hypono,4))), exp(j*(phvec(hypono,1)+phvec(hypono,2)+...
24              phvec(hypono,4))), -1*exp(j*(phvec(hypono,1)+phvec(hypono,4))), ...
25              exp(j*(phvec(hypono,1)+phvec(hypono,2)+phvec(hypono,3))), ...
26              exp(j*(phvec(hypono,1)+phvec(hypono,3))), -1*exp(j*(phvec(hypono,1)+...
27              phvec(hypono,2))), exp(j*phvec(hypono,1))];
28  end; %for bitno
29
30  %Define even and odd phase changes
31  phchevn(1)=0;
32  phchevn(2)=pi/2;
33  phchevn(3)=3*pi/2;
34  phchevn(4)=pi;
35  phchodd=phchevn+pi;
36
```

---

cck55demod.m

```
1   %CCK - 5.5 Mbps near optimal demodulation
2
3   %By Dr. Deva K. Borah, NMSU
4   %Modified by Anirudh Daga, NMSU
5   %Please do not distribute without the permission of the author
6
7   function [d_rcv]=cck55demod(rcv3,nBittx,codevec,bitvalvec,phchodd,phchevn,phvec,ph0);
8
9
10  lblocks=nBittx/4;
11  prevfirstph=0;
12
13  rcv3 = rcv3(1:2*nBittx);
14  rcv3=rcv3.*exp(-j*ph0);
15  for ii=1:lblocks
16      testvec=rcv3((ii-1)*8+1:ii*8);
17      testmat=repmat(testvec,16,1);
18      testmatstatus=conj(testmat).*codevec;
19      [maxval,maxind]=max(sum(real(testmatstatus).'));
20      detbits((ii-1)*4+3:ii*4)=bitvalvec(maxind,3:4);
21      firstphtx=phvec(maxind,1);
22      phchange=firstphtx-prevfirstph;
23      if (mod(ii,2)==0)
```

```
24          [minval,minind]=min(abs(exp(j*phchodd)-exp(j*phchange)).^2);
25      else
26          [minval,minind]=min(abs(exp(j*phchevn)-exp(j*phchange)).^2);
27      end; %if
28
29      secbit=mod(minind-1,2);
30      firstbit=floor((minind-1)/2);
31      detbits((ii-1)*4+1:(ii-1)*4+2)=[firstbit,secbit];
32      prevfirstph=firstphtx;
33
34  end; %for ii
35  d_rcv=detbits;
```

CCK11DemodRake_NoFor.m

```
1   %CCK - 11 Mbps near optimal demodulation (with RAKE)
2
3   %By Dr. Deva K. Borah, NMSU
4   %Modified by Anirudh Daga,NMSU
5   %Please do not distribute without the permission of the author
6
7   function [d_rcv]=CCK11DemodRake_NoFor(rcv3,nBittx,codevec,bitvalvec,...
8                   phchodd,phchevn,phvec,ph0,PSDULength,hhat);
9
10
11  lblocks=nBittx/8;
12  prevfirstph=0;
13
14  rcv = rcv3(1:PSDULength+length(hhat)-1);
15  temp = repmat((1:length(hhat))',1,PSDULength) + repmat(0:PSDULength-1,length(hhat),1);
16  rcv2 = conj(repmat(hhat(:),1,PSDULength)).*rcv(temp);
17  rcv3 = sum(rcv2);
18  rcv3 = rcv3.*exp(-j*ph0);
19
20  codevec = round(codevec);
21  for ii=1:lblocks
22      testvec=rcv3((ii-1)*8+1:ii*8);
23      testmat=repmat(testvec,256,1);
24      testmatstatus=conj(testmat).*codevec;
25      [maxval,maxind]=max(sum(real(testmatstatus).'));
26      detbits((ii-1)*8+3:ii*8)=bitvalvec(maxind,3:8);
27      firstphtx=phvec(maxind,1);
28      phchange=firstphtx-prevfirstph;
```

```
29      if (mod(ii,2)==0)
30          [minval,minind]=min(abs(exp(j*phchodd)-exp(j*phchange)).^2);
31      else
32          [minval,minind]=min(abs(exp(j*phchevn)-exp(j*phchange)).^2);
33      end; %if
34
35      secbit=mod(minind-1,2);
36      firstbit=floor((minind-1)/2);
37      detbits((ii-1)*8+1:(ii-1)*8+2)=[firstbit,secbit];
38      prevfirstph=firstphtx;
39
40  end; %for ii
41  d_rcv=detbits;
```

CCK55DemodeRake_NoFor.m

```
1   %CCK - 5.5 Mbps near optimal demodulation (with RAKE)
2
3   %By Dr. Deva K. Borah, NMSU
4   %Modified by Anirudh Daga, NMSU
5   %Please do not distribute without the permission of the author
6
7   function [d_rcv]=CCK55DemodRake_NoFor(rcv3,nBittx,codevec,bitvalvec,...
8                   phchodd,phchevn,phvec,ph0,PSDULength,hhat);
9
10
11  lblocks=nBittx/4;
12  prevfirstph=0;
13
14  rcv = rcv3(1:PSDULength*2+length(hhat)-1);
15
16  temp = repmat((1:length(hhat))',1,PSDULength*2)...
17         + repmat(0:PSDULength*2 - 1,length(hhat),1);
18  rcv2 = conj(repmat(hhat(:),1,PSDULength*2)).*rcv(temp);
19  rcv3 = sum(rcv2);
20  rcv3 = rcv3.*exp(-j*ph0);
21
22  for ii=1:lblocks
23      testvec=rcv3((ii-1)*8+1:ii*8);
24      testmat=repmat(testvec,16,1);
25      testmatstatus=conj(testmat).*codevec;
26      [maxval,maxind]=max(sum(real(testmatstatus).'));
27      detbits((ii-1)*4+3:ii*4)=bitvalvec(maxind,3:4);
```

```
28        firstphtx=phvec(maxind,1);
29        phchange=firstphtx-prevfirstph;
30        if (mod(ii,2)==0)
31            [minval,minind]=min(abs(exp(j*phchodd)-exp(j*phchange)).^2);
32        else
33            [minval,minind]=min(abs(exp(j*phchevn)-exp(j*phchange)).^2);
34        end; %if
35
36        secbit=mod(minind-1,2);
37        firstbit=floor((minind-1)/2);
38        detbits((ii-1)*4+1:(ii-1)*4+2)=[firstbit,secbit];
39        prevfirstph=firstphtx;
40
41    end; %for ii
42    d_rcv=detbits;
```

---

rounddec.m

```
1   function y = rounddec(x, n)
2   %ROUNDDEC Round to a specified number of decimals.
3   %
4   %   Y = ROUNDDEC(X, N) rounds the elements of X to N decimals.
5   %
6   %   For instance, rounddec(10*sqrt(2) + i*pi/10, 4) returns
7   %   14.1421 + 0.3142i
8   %
9   %   See also: ROUND, FIX, FLOOR, CEIL, ROUNDDIG, TRUNCDEC, TRUNCDIG.
10
11  %   Author:      Peter J. Acklam
12  %   Time-stamp:  2004-09-22 20:06:46 +0200
13  %   E-mail:      pjacklam@online.no
14  %   URL:         http://home.online.no/~pjacklam
15
16      % Check number of input arguments.
17      error(nargchk(2, 2, nargin));
18
19      % Quick exit if either argument is empty.
20      if isempty(x) || isempty(n)
21         y = [];
22         return
23      end
24
25      % Get size of input arguments.
```

```
26      size_x   = size(x);
27      size_n   = size(n);
28      scalar_x = all(size_x == 1);          % True if x is a scalar.
29      scalar_n = all(size_n == 1);          % True if n is a scalar.
30
31      % Check size of input arguments.
32      if ~scalar_x && ~scalar_n && ~isequal(size_x, size_n)
33         error(['When both arguments are non-scalars they must have' ...
34                ' the same size']);
35      end
36
37      f = 10.^n;
38      y = round(x .* f) ./ f;
```

List of Symbols, Abbreviations, and Acronyms

$E[*]$	–	Expectation operator
h	–	Antenna height
$P(n)$	–	received power at index n

μs	–	microseconds (10^{-6} seconds)
σ_τ	–	rms delay
τ_k	–	Time delay
$\bar{\tau}_k$	–	Mean time delay

AP	–	Access Point
BPSK	–	Binary Phase Shift Keying
BER	–	Bit Error Rate
BVS	–	Berkeley Varitronics Systems
CCK	–	Complementary Code Keying
CRC	–	Cyclic Redundancy Code
dBi	–	deciBels referenced to an isotropic (unity gain) antenna
dBm	–	deciBels referenced to 1 milliwatt
DBPSK	–	Differential Binary Phase Shift Keying
DEM	–	Digital Elevation Model
DQPSK	–	Differential Quadrature Phase Shift Keying
DSSS	–	Direct Sequence Spread Spectrum
EIRP	–	Effective Isotropic Radiated Power
GHz	–	GigaHertz (10^9 Hz)
GI	–	Guard Interval
GPS	–	Global Positioning System
IEEE	–	Institute of Electrical and Electronic Engineers
IP	–	Internet Protocol
ITM	–	Irregular Terrain Model
LAN	–	Local Area Network
m	–	meter
MAC	–	Media Access Control
MGS	–	Mars Global Surveyer
mW	–	milliWatts (10^{-3} Watts)
ms	–	milliseconds (10^{-3} seconds)

NMSU	–	New Mexico State University
ns	–	nanoseconds (10^{-9} seconds)
OFDM	–	Orthogonal Frequency Division Multiplexing
PBCC	–	Packet Binary Convolutional Code
PHY	–	Physical Layer
PDA	–	Personal Digital Assistant
PER	–	Packet Error Rate
PDP	–	Power Delay Profile
PLCP	–	Physical Layer Convergence Procedure
PPDU	–	Physical layer Protocol Data Unit
PSDU	–	Physical Sublayer service Data Unit
QAM	–	Quadrature Amplitude Modulation
QoS	–	Quality of Service
QPSK	–	Quadrature Phase Shift Keying
RF	–	Radio Frequency
RSSI	–	Received Signal Strength Indicator
RTT	–	Round Trip Time
RX	–	Receiver
SFD	–	Start Frame Delimiter
TCP	–	Transmission Control Protocol
TX	–	Transmitter
UDP	–	User Datagram Protocol
W	–	Watt
WEP	–	Wired Equivalent Privacy
WLAN	–	Wireless Local Area Network