Trade-offs Between Energy and Security in Wireless Networks

by

Kerry McKay

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

April 2005

APPROVED:

Professor Fernando C. Colón Osorio, Thesis Advisor

Professor Emmanuel Agu, Thesis Reader

Professor Michael Gennert, Head of Department

Abstract

As the popularity of wireless networks increases, so does the need to protect them. In recent years, many researchers have studied the limitations of the security mechanisms that protect wireless networks. There has also been much research in the power consumption introduced by the network card. Technologies such as CPU and memory are increasing and so is their need for power, but battery technology is increasing at a much slower rate, forming a "battery gap". Because of this, battery capacity plays a major role in the usability of the devices. Although the effect of the network communication on a mobile device's battery has been widely researched, there has been less research on the effect of the security profile on energy usage.

In this thesis, we examine a method for analyzing trade-offs between energy and security proposed by Colón Osorio et al. This research describes a method to identify the most appropriate security profile for a given application, given battery constraints. The same method can also be used to discover the minimum battery capacity to maintain a minimum security profile for a predefined amount of time.

Trade-offs and optimality are analyzed using a cost-energy function, C^E, and security measure, S_M. C^E encompasses the energy required to use countermeasure M against a specific vulnerability, V_i, as well as the energy consumed in bulk transfer. S_M is a numerical representation of the effectiveness of a set of security mechanisms which utilize the set of countermeasures to defend against a set of vulnerabilities. Using C^E and S_M, we can compare different security profiles using a trade-off model. Having defined such a framework, we investigate different instances and examples where the use of the model is helpful in accessing trade-offs between security obtained and energy consumed to achieve such security. This was first examined through an analytical study, followed by experimentation.

The major contributions of this work are an energy-security trade-off model and its empirical validation. This work extends the empirical experimentation done by other researchers such as Potlapally et al., Karri et al., and Stemm and Katz on the relationship between energy and the security of wireless communications in battery-constrained devices.

Acknowledgments

I would like to express my gratitude to my advisor, Professor Fernando C. Colón Osorio, and reader, Professor Emmanuel Agu, for making this work possible. I would also like to thank the WPI System Security Lab (WSSRL) for all their input, feedback, and assistance, especially Justin Whitney, Frank Posluszny III, and Dave Toth. Thanks to Will Chin for his help with the experimental hardware, and Lin Zhong for LabVIEW assistance and clarification on the experimental design on which mine is based. I would also like to extend a heart-felt thanks to my family and friends for their support, without which I would not have been able to complete this work.

This work was partially funded by the Acumen Consulting Group, Inc., Marlboro, MA under grant ACG-2004-01.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The use of wireless networks is continuously increasing. The sales of embedded wireless devices grows 66.2% each year[12]. Table 1.1 shows that hot spots are becoming more frequent in public areas such as airports, hotels, and retail stores. Newer generations of mobile computing equipment come with wireless support standard. In 2003, 55% of laptops sold had embedded wireless support built in [12], and this percentage is expected to grow even more due to technologies like Intel's Centrino chip. Indeed, from corporate networks to home networks, the number of wireless networks and clients is on the rise. As the world becomes more dependent on wireless networks, it needs to improve the mechanisms that protect them.

A key limitation in wireless devices is the battery capacity. While memory and processor technologies double with the introduction of every new semiconductor generation (roughly every 18 months), battery technology is increasing at the much slower rate of 5%-10% per year[17]. This is causing a gap to form between the power required and the battery available (figure 1.1). Some may argue that this is not important, because people often plug in their laptops during wireless network usage. When we say wireless device we are mainly concerned with wireless handheld devices, which are rarely plugged into a power supply during normal usage as their main advantage is unrestricted mobility. Research in the power consumpition of wireless handhelds has been primarily done in three areas:

1. energy utilization of the network interface card
2. overall impact of the NIC on mobile systems
3. power management techniques

Location	**2001**	**2002**	**2003**	**2004**	**2005**
Airports	85	152	292	378	423
Hotels	569	2,274	11,687	22,021	23,663
Retail Outlets	474	11,109	50,287	82,149	85,567
Enterprise Guesting Areas	84	624	1,762	3,708	5,413
Stations and Ports	-	88	623	2,143	3,887
Community Hot Spots	2	266	5,637	20,561	30,659
Others	-	240	790	1,526	2,156
Total Market	1,214	14,752	71,079	132,486	151,768

Source: Gartner Dataquest (June 2003)[10]

Table 1.1: Public WLAN Hot Spot locations worldwide, by type

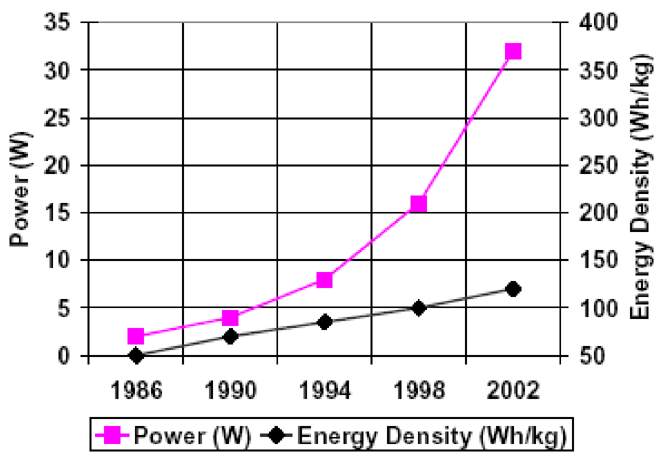


Figure 1.1: growing gap between battery technology and power requirements[17]

There has been some research on the effect of wireless security on the total energy consumption, showing case studies and possible energy optimizations. To our knowledge, there has been no conclusive research on making intelligent trade-offs between security and energy consumption. If trade-offs between security and energy can be represented in a mathematical form, then we can use that information to better choose a security for a given application. This knowledge will lead to optimal energy usage, with respect to the security profile.

The largest source of power drain on a wireless mobile client is packet transmission. Security protocols, specifically the authentication portion, may require many or few transmissions, depending on the protocol. For instance, WEP authentication (discussed in section 2.2.1) contains only two messages sent by the client whereas EAP-based methods require the client to send at least four messages - a minimum 100% increase. The client needs to send a message to get a ticket-granting ticket, and then for every service, a message requesting a ticket and a message for logging into that service are required. The protocol has a direct impact on the number of transmissions, and subsequently on the battery life.

In addition to the cost of transmission, there are large differences in energy consumed by other factors of each protocol. The energy drained by cryptographic computations does matter, as reducing the energy cost will extend the time that a mobile device can be used. Although transmission is the biggest source of energy consumption, finding optimizations with respect to the security profile are advantageous.

In this thesis, we first review current 802.11 security standards and their limitations. We then use a model proposed by Colón Osorio et al.[4] to understand how such protocols affect the energy consumption of a mobile device. More specifically, we attempt to quantify how much additional power is expended by a mobile device int order to achieve a given security profile. This model will be used to evaluate WEP (section 2.2), WPA (section 2.2.1), 802.1x/EAP (section 2.2.2), and CCMP (section 2.2.3). They are first evaluated by analytical methods used to create a hypothesis, and then compared with the empirical measurements of our experiment to support our hypothesis.

1.1 Previous Work

Much research has been conducted on the effects of the wireless card on mobile devices. However, little of that research has focused on the security profile. Based on our literature survey, most of

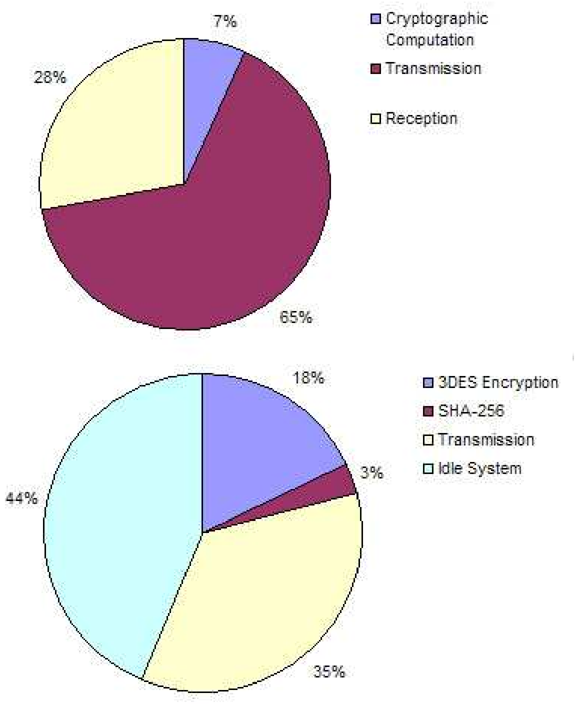


Figure 1.2: Energy consumption described by Karri and Mishra[14]

the work in this area has been done in the transmission of packets and energy profiling.

Stemm and Katz[28] provided us with a model for breaking down energy expended in wireless communication. They examined packets of b bytes, and derived costs for energy used in the idle state (equation 1.1), transmission and reception of packets (equation 1.2), and the total energy (equation 1.3).

$$Idle = I\frac{b}{B} \tag{1.1}$$

$$SendRecv = aE_a + dE_d \tag{1.2}$$

$$Energy = SendRecv + Idle \tag{1.3}$$

In reviewing previous work closely related to our work, that is understanding the impact of security mechanisms on the battery life of mobile devices, one manuscript is worth mentioning. The manuscript at hand is that of Potpally et.al[25] which examined the energy consumed by a PDA to communicate with a secure connection via wireless network. While their paper did not use the same modelling that we employ here, it provided a solid foundation for an experimental structure, as well as data that could be used in the verification of our experimental setup. This paper made an attempt to analyze trade-offs between security and energy, but focused primarily on the key-sizes of encryption algorithms rather than the security of the protocol as a whole.

Karri et al.[14] also had a related work, although they did not attempt to perform any trade-off analysis. This case study measured the energy usage of an encryption algorithm, packet transmission, receiving packets, and the idle state. A sample of their findings is depicted in figure 1.2. They also examined the effect of compression on the power utilization.

As stated previously, we are concerned with the number of messages that must be passed during the authentication portion of the protocol. It follows that we need to take into account the amount of disassociation and reassociation that occurs in a typical mobile session. Several studies have been conducted where students analyze the traffic of their campus network[16][11][29][3]. Tang and Baker traced wireless connections within buildings, as well as a metropolitan area network (MAN). Additionally, a study conducted by U.C. San Diago and Microsoft attempted to characterize user behavior with respect to wireless networks during a conference.

The most comprehensible and applicable of these studies are those by Kotz et al.[16][11] in 2002 and 2004. During their observations of wireless activity on the Dartmouth campus, they gathered sufficient information to identify clients roaming between access points. In their 2004 study, they found that half of all wireless clients roamed between access points. In their previous study, only one third of the clients roamed. Indeed, they found that the number of wireless clients overall and the percentage of those that were mobile had increased in two years. This observation of the rise in roaming sessions supports and strengthens our claim that the cost of reassociation needs to be factored into energy measurements.

1.2 Coverage and Overview of Thesis

In this thesis we provide a limited review of 802.11 security protocols. Specifically, the reviewed protocols are WEP, TKIP, EAP, and CCMP, with descriptions of their encryption schemes as well (RC4, AES, etc). It also includes a short introduction to wireless networks and how they differ from wired networks from a security perspective. The experiment test bed and all of its components are explained, as well as the overall energy equations used.

There are attacks on the 802.11 protocol which are unrelated to the security protocols described. Namely, attacks involve tactics such as access point spoofing and sending packets at certain intervals such that a denial of service is created. These attacks are on the underlying 802.11 protocol and not on the security profiles examined in this paper. They are outside of the scope of this project.

This thesis does not include details or instruction on programming in Microsoft Embedded Visual C++, Microsoft Foundation Classes, or National Instruments LabVIEW. In addition, we do not include specifics of the experiment configuration process. Code used in this thesis is presented in appendices A and C.

The rest of this thesis is organized as follows. Chapter 2 reviews differences between wired and wireless networks, and standard wireless security profiles that exist today. Chapter 3 describes our model for reasoning about energy-security trade-offs. Chapter 4 covers our preliminary analysis, where we study the algorithms used in each protocol and form hypotheses about how the different security mechanisms impact the battery. Our experimental design is illustrated in chapter 5 and the results in chapter 6. Finally, we give a summary and conclusions in chapter7.

Chapter 2

Background

2.1 A Brief Overview of Wireless Network

2.1.1 Summary of 802.11 Protocol

In order to understand the security protocols available for wireless networks, let us first examine the 802.11 protocol. 802.11 is a MAC layer protocol which uses radio frequencies in unlicensed portions of the spectrum, called the Industrial, Scientific, and Medical (ISM) bands. Currently, those frequencies are 2.4 GHZ (802.11b and 802.11g) and 5 GHz (802.11a). The range of each radio's transmission creates a cell. If two access points are nearby, then there cells will overlap and a client may connect to either of them, but not both.

In order for a client to connect to an access point, it first has to authenticate. This authentication is performed by a challenge-response. If authentication is successful, the client then needs to associate with the access point. Should a client wander outside of its current cell, then it will be disconnected and need to associate again. During a mobile session, a client may roam from one access point to another within the same network. Here, the client will need to reassociate with the new access point. When the client resides in an overlap between two such access points, then it may constantly disassociate and reassociate as the signals fluctuate that change which is the stronger access point. The states described here are expressed by figure 2.1

2.1.2 How Does Security on a WLAN Differ from a Wired LAN?

The greatest factor that separates wired and wireless security is the concept of *physical security.* Before Wireless Local Area Networks (WLANs), access to internal networks could be limited to those who were allowed to get in close proximity to machines on the network. Walls and doors protected unauthorized users from gaining access. However, wireless signals leak outside these boundaries. In the earlier years of WLAN deployment, companies would put access points inside their firewalls, allowing anyone in range of the signal to crack their way in. This was known as the "parking lot attack"[2].

Presently, similar tactics are still being employed. "Wardriving" and "Warchalking" are still occurring. In these activities, the goal is to find an open network or breach the security, and gain access to the network. It is not unusual to hear of someone who steals a neighbor's Internet connection over a wireless network. Sometimes, the abuse is minor, unintrusive, and non-consequential. However, this type of breach can have huge consequences attached. In a world where our Internet usage may be subpoenaed, it is not wise to allow others to access a network that we are legally responsible for.

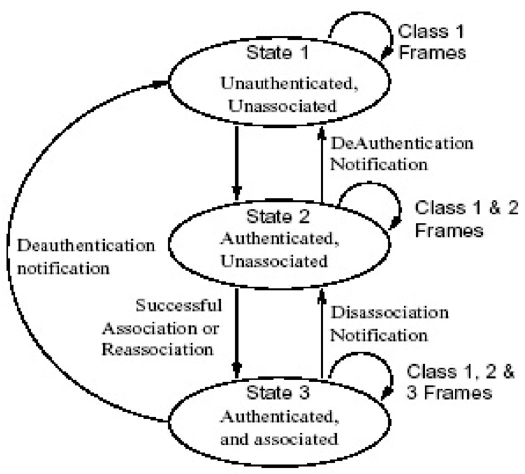

State 1
Unauthenticated, Unassociated
Class 1 Frames
DeAuthentication Notification
State 2
Authenticated, Unassociated
Class 1 & 2 Frames
Deauthentication notification
Successful Association or Reassociation
Disassociation Notification
State 3
Authenticated, and associated
Class 1, 2 & 3 Frames

Figure 2.1: 802.11 State Machine [21]

2.2 Wireless Security Protocols

We shall now review wireless security protocols, in order of appearance on the market.

2.2.1 WEP

The Wired Equivalent Privacy (WEP) protocol was created as a way to ensure the same level of privacy for wireless communication as there is for wired communications. Its goals, as with any security mechanism, is to provide confidentiality, integrity, and availability to the wireless network. Unfortunately, WEP accomplishes none of these goals. It is a very poor protocol and was nearly removed from the 802.11 standard in a vote by the IEEE in June 2001 (54%-46%)[22].

WEP Encryption

The encryption scheme used in WEP is a very simple one: it uses the RC4 stream cipher to generate a pseudo-random keystreams which it XORs with the plaintext to encrypt. To decrypt, XOR the keystreams with the ciphertext.

$$keystream = RC4(IV + key)$$

$$C = P \oplus keystream$$

$$P = C \oplus keystream$$

RC4 is a keyed stream cipher containing two different functions - the key scheduling algorithm (KSA) and the pseudo-random generator algorithm (PRGA)[8]. In WEP, the RC4 key is the concatenation of a 24-bit initialization vector (IV) and the shared secret key common to the access point and all its users. The same RC4 key will always produce the same keystream. Since the only varying piece of this is the IV, that means that there will only be at most 2^{24} different keystreams generated. While that may seem like a large number, it may be exhausted in seven hours of maximum full-frame transmission on an 802.11b network [6]. This small space causes keystreams to repeat, which is in violation of a key concept in the security of stream ciphers - the same keystream should never be used twice.

To help alleviate this problem, the IV space was increased to 128 bits. Unfortunately, this did not solve the problem, since IV's are still reused it was never enforced that more than one IV had to be used in the first place. Vendors could set their devices to only go between 0 and 2^{24}, and the WEP protocol has no way of preventing or detecting this.

The RC4 cipher itself also has security issues. The key scheduling algorithm has been shown to leak information about the key, one byte at a time. By collecting about 60 messages of a special form, an attacker can guess the secret with a high probability of being correct [8].

Integrity Check

The WEP integrity check is weak. It uses a cyclic redundancy check like the one used to detect *random* errors in networking. The distinction between random and intentional changes is very important. The output space of this integrity check value (ICV) is only 32 bits, which is poor for collision resistance. It is unkeyed and linear, so anyone can compute it. Someone could easily change or spoof a packet, and it would go undetected because it has an ICV that matches.

Authentication

WEP uses a simple challenge/response protocol that is also quite poor. The challenge exchange goes as follows[2]:

$$\begin{aligned} AP &\rightarrow client : challenge \\ client &\rightarrow AP : IV, \{challenge, ICV\}_{wepKey} \end{aligned}$$

This is completely unacceptable as an authentication scheme. By capturing the clear challenge, the encrypted challenge, and the IV, an unauthorized user can gain access by doing simple math.

$$keystream = C \oplus P$$

An attacker could gain access to the network without knowing the shared secret.

2.2.2 WPA

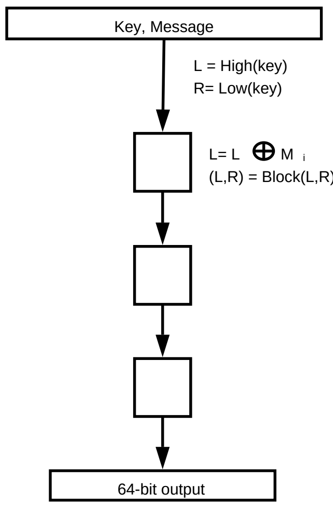
One of the reasons that WEP has remained in the 802.11 standard is that it is widely deployed and implemented in hardware. WI-FI Protected Access (WPA) is a set of improvements over WEP that are compatible with existing hardware.

The Temporal Key Integrity Protocol

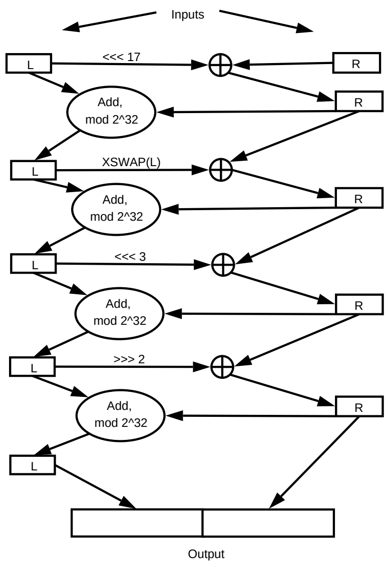
The Temporal Key Integrity Protocol (TKIP) is a modified version of WEP's encryption scheme. Like WEP, it uses the RC4 stream cipher to generate a keystream which is then XORed with the plaintext. What TKIP brings to the table is a way of creating keystreams which are unique to each packet. This is done by mixing the transmitter address (TA) into the key, giving each user a unique key per session, and by using the IV as a counter. If an IV value is not the one expected, then it is discarded. When the IV space is almost exhausted, a new key is negotiated.

Michael

The TKIP specification also names a new message integrity code (MIC) called Michael. Michael is a non-linear hash function that produces a 64-bit output. Unlike the CRC used in WEP, Michael is keyed. Only those who know the secret can compute a valid hash. However, it should be noted that the output space is still small, allowing the possibility of finding or guessing a valid hash.

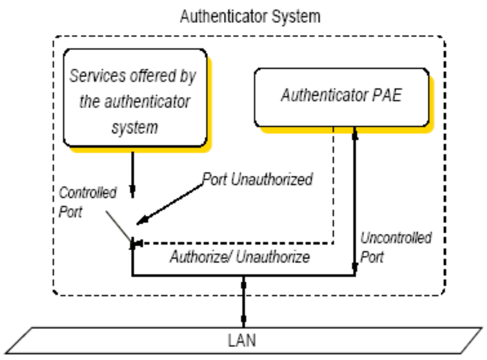


(a) Michael function

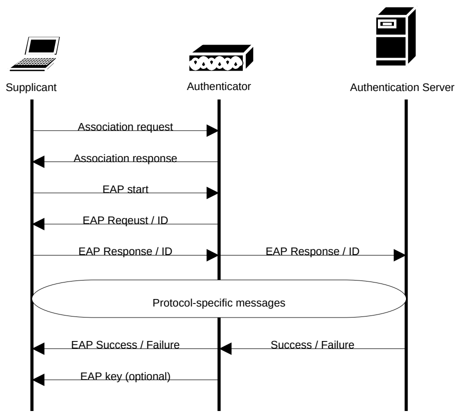


(b) Michael block

Figure 2.2: Michael



(a) Controlled and uncontrolled ports in the authenticator



(b) Basic EAP messages

Figure 2.3: EAP/802.1x Authentication

2.2.3 802.1x/EAP Authentication

802.1x is a flexible framework which has been created for authentication in PPP protocol. This framework can also be applied to a wireless network to allow key distribution for TKIP with existing hardware. 802.1x defines the idea of port-based access control; conceptually access control will require involves two ports: a controlled port and an uncontrolled port. Access to the uncontrolled port can be gained at any time, and this port leads to the authentication service. The controlled port can only be accessed after authentication and authorization have taken place, as denoted by the switch in figure 2.3(a). In wireless networks, the controlled port is the AP's connection to the network, and the uncontrolled port goes to an authentication server, such as RADIUS (remote authentication dial-in user service).

There are three parties identified in this authentication scheme. The supplicant is the entity that wishes to be authenticated (wireless client). The authenticator is the entity with which the supplicant is trying to authenticate (access point). Authentication is provided by the third party, the authentication server, through communication with the authenticator. The supplicant and authenticator send messages over the wireless medium, while the authenticator and authentication server communicate over a wire. The separation of services here is interesting because it is something that was borrowed from the wired world. It is also interesting to note that a wire has actually been introduced into the authentication process.

The extensible authentication protocol (EAP) is an outline for authentication that sits underneath a higher protocol (figure 2.3(b)). For instance, SSL could be used on top of EAP. Protocols which are currently available from vendors deploying TKIP and EAP (Cisco Systems for example) include protocols such as EAP-TLS (transport layer security), LEAP (Cisco's lightweight EAP), EAP-TTLS (tunneled transport layer security), and PEAP (protected EAP). Each variant has its own methods, such as mutual authentication vs. client-only authentication, and certificates vs. username/password.

2.2.4 CCMP

WPA was not intended as a permanent solution to our wireless security dilemma. On June 25, 2004, IEEE TGi approved the newest wireless security standard, 802.11i, which includes a new protocol, Counter CBC-MAC Protocol (CCMP). This new protocol differs dramatically from WEP and TKIP in how it encrypts data. While the previous protocols used the RC4 *stream cipher*, CCMP uses the Advanced Encryption Standard (AES) *block cipher.*

When encoding data with block ciphers, the data is broken up into blocks of fixed and equal length. AES was designed to accomodate 128, 192, or 256-bit blocks, but only blocks of 128 bits are utilized by CCMP. Each block is encrypted independently of the rest and then linked together by a specified *mode of operation.* Typically, a block cipher algorithms contain a forward cipher function for encryption and an inverse cipher function for decryption, and this holds true for AES. However, in CCMP, only the forward cipher function is needed.

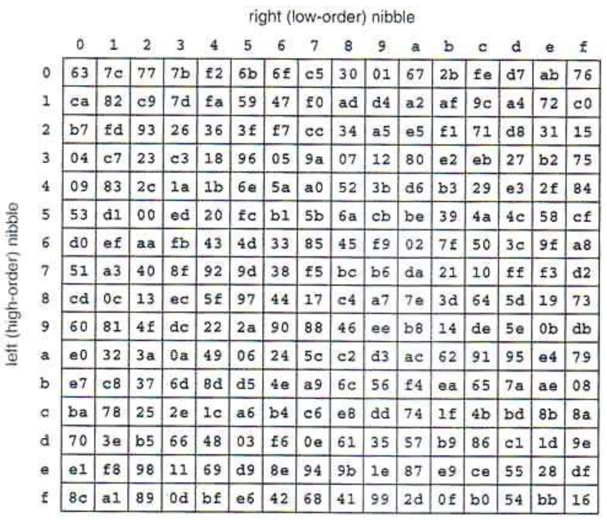
In AES, each data block is stored as two-dimensional array thats size is determined by its length. In CCMP, only 128-bit blocks are used, for both the key and plaintext. Therefore, all data blocks are stored as 4x4 arrays, where each cell contains one byte. Figure 2.2.4 shows this representation of the block.

$b0$	$b4$	$b8$	$b12$
$b1$	$b5$	$b9$	$b13$
$b2$	$b6$	$b10$	$b14$
$b3$	$b7$	$b11$	$b15$

Figure 2.4: AES representation of data

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

(a) AES S-box matrix multiplication [5]



right (low-order) nibble

left (high-order) nibble	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

(b) AES S-box look-up table[15]

AES is comprised of a key scheduling algorithm, plus four additional functions which occur at three different layers.

- key addition
- byte substitution
- shiftRow
- mixColumn

The key addition layer is the simplest. It is a simple bitwise xor of the data with the current key. Here, the current key is one of the unique keys created in the key scheduling process. The substitution layer is also quite straight forward. There is a matrix multiplication known as an S-box, which introduces a non-linear transform of the data. This operations is shown in figure 2.5(a). The S-box can also be executed via look-up table, denoted in figure 2.5(b). For each octet of data, the low-order and high-order nibbles are used as indices, and the result of (high, low) in the S-box then replaces that octet. The diffusion layer consists of two operations, the first of which is the shiftRow operation (figure 2.5(c)). Each row of the data block is shifted a specified amount. Finally, the data undergoes the MixColumn operation (figure 2.5(d)), where each column is multiplied by a 4x4 matrix.

Each round applies all four these functions, except for the final round. In the final round, the mixColumn operation is not applied.

$b0$	$b4$	$b8$	$b12$	no shift
$b1$	$b5$	$b9$	$b13$	$\rightarrow$ 3 positions
$b2$	$b6$	$b10$	$b14$	$\rightarrow$ 2 positions
$b3$	$b7$	$b11$	$b15$	$\rightarrow$ 1 positions

(c) AES ShiftRow operation

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

(d) AES MixComlumn operation

Figure 2.5: AES diffusion layer

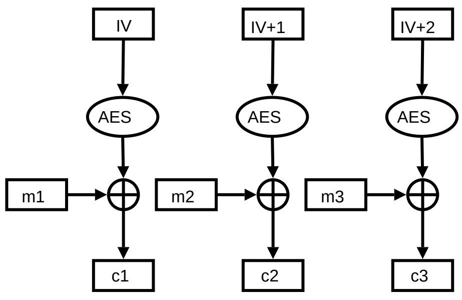
Counter CBC-MAC (CCM) mode is a new mode of operation that was created to both authenticate and encrypt. Here, the term “authenticate” refers to verifying the origin of the message. This new mode combines two old modes. Counter mode (CTR) is used for confidentiality, and cipher block chaining mode (CBC) is used for authentication (figures 2.6(a) and 2.6(b), respectively). CCM uses standard CTR mode, but a variation on the standard usage of CBC known as CBC-MAC. This adaptation uses an IV of zero, and uses the final block (which may or may not be truncated) as a MAC.

There are processes which comprise CCM, generation-encryption and decryption-verification. Generation-encryption takes in a nonce, payload, and associated data (if any), and applies a formatting function that breaks the inputs into blocks. From this state, CBC is applied to produce the MAC, counter blocks are generated, and the plaintext is encrypted with AES in CTR mode.

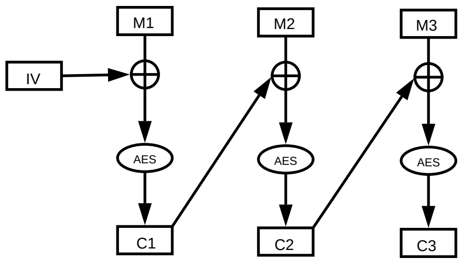
To decrypt a message, the ciphertext is first run through counter mode decryption. This will produce the distorted payload and MAC. The payload, nonce, and associated data are then checked for validity. If they are invalid then an error message is returned. If they are valid, then they are formatted into blocks and the CBC-MAC is applied to verify the MAC tag. If the MAC verifies the payload, then plaintext payload is returned. Otherwise, an error message is returned.

CCMP uses the 802.1x framework for authentication. There are no standard EAP methods that are part of the security standard, but stronger methods are recommended.

The 802.11i security standard includes, in addition to CCMP, WPA. This remains for backward compatibility with legacy wireless devices.



(a) AES in counter mode



(b) AES cipher block chaining mode

Figure 2.6: CTR and CBC modes

Chapter 3

Analyzing Trade-offs Between Energy and Security

3.1 Energy-Security Trade-off Model

Colón Osorio et al.[4] described a model to calculate the energy consumed by using a countermeasure (M_k) to protect against a certain vulnerability (V_i). First, we defined the total energy cost as the sum of energy consumed to launch all countermeasures included in the protocol (equation 3.1).

$$C^E_{total} = \sum_i C^E(M_k, V_i) \tag{3.1}$$

Secondly, we defined the measure of security a given countermeasure M provides as S_M. Because there is currently no known method for empirically measuring the security of a protocol, we use a subjective estimate based on the effort/time needed to break the protocol. This incorporates the key size, known weaknesses, and the availability of tools designed for attacking these protocols.

Finally, with knowledge of C^E and S_M, we can obtain greater insight by applying equation 3.2. This is called the *Countermeasure Energy Quotient.* Given a set of protocols that meet a minimum requirement for security profile or battery lifetime, this quotient allows us to identify the optimal protocol. The optimal protocol is that which maximizes Q_M.

$$Q_M = \frac{S_M}{C^E_{total}} \tag{3.2}$$

3.2 An Instance of the Energy-Security Trade-off Model

This model suggests that vulnerabilities be divided to provide the three services of security. Specifically,

- V_1 = Robustness of the cryptographic algorithm(confidentiality)
- V_2 = Robustness of the integrity check (integrity)
- V_3 = Robustness of the authentication, authorization and access protocol (availability)

Integrity checks and encryption may be grouped together, but for security purposes have been seperated. Authentication, authorization, and access have been split despite the fact that they all are associated with availability. The reason behind this is related to message passing. Some

Vulnerability		64-bit WEP	128-bit WEP	CKIP+MMH	WPA-LEAP	AES-CCM
encryption	key not renegotiated when exhausted	0	0	0	1	1
	known (practical) attacks on cipher	0	0	0	0	1
	key discovery through packet collection	0	0	1	1	0
integrity	birthday attack	1.52588E-05	1.52588E-05	1.5259E-05	0.125	1
	origin not protected	0	0	0	0	1
	bit-flipping attack	0	0	1	1	1
	anyone can compute	0	0	1	1	1
availability	authentication without secret	0	0	0	0	1
	open authentication allowed	0	0	0	1	1
	authenticate hardware, not person	0	0	0	1	1
		1.52588E-05	1.000015259	5.00001526	8.125	11

Table 3.1: Security proxy

protocols, such as WEP, group these operations into one. However, protocols exist where each of these steps requires a message. Protocols which use ticket granting mechanisms, such as Kerberos, are examples of this.

The energy expenditure function associated with each countermeasure M_1, M_2, and M_3, $C^E(M_k, V_i)$, is defined by the protocol itself and the parameters used. For example, in WEP, the countermeasure against V_1 is simply the RC4 stream cipher. In this case, the energy expenditure to achieve the desired level of security is $C^E(K_{length}, V_i) = f(\#\text{ of computations to encrypt})$

3.3 Security Proxy

To our knowledge, there is currently no empirical means of measuring the security of a protocol. In this thesis, we have derived a proxy as an estimate. Our proxy is simply an ordinal scale that ranks security profiles by counting vulnerabilities and the countermeasures against them. It is important to note that because this scale is *ordinal*, the numbers have no meaning on their own. Meaning can only be obtained by saying $x\ R\ y$, where R is a relation. This also means that our quotient, Q_M, is on an ordinal scale.

The aspects of each protocol are rated against the classic categories of attacks. If it withstands the attack, then it receives a 1 in that category. If not, then it receives a 0. For vulnerabilities that are not simply a 'yes' or 'no', but vary in difficulty, such as brute force and birthday attacks, ratios are used to assign a number between 0 and 1. This method of comparison assumes that all vulnerabilities are equal. This is not an accurate assumption, as some vulnerabilities are worse than others. For instance, one vulnerability may only affect a single message by rearranging blocks within it, making it gibberish, while another vulnerability may render the entire network unusable. Clearly, the latter has a more severe impact than the former. While our assumption is not true in practice, we believe that our proxy is sufficient for purposes of illustrating the model.

Chapter 4

Analytical Study

The first part of understanding the relationship between security countermeasures and energy consumption consisted of an analytical study involving WEP, WPA, and CCMP. Each of the computational algorithms was examined for a specified packet size based on RFC information and observations. This study provided insight, but was clearly not sufficient.

In order to perform a valid analysis, we obtained code for 802.11i from an IEEE member [13]. This code includes C files for CCMP MPDU encryption, TKIP key mixing, RC4, and Michael. This code was created to follow the algorithms described in the drafts exactly, not implement any efficiency improvements.

Based on these algorithms, we comprised the graphs shown in figures 4.1 and 4.2. The first contains the cost of encryption and the integrity check, and the latter just the encryption. We can see that for encryption only, AES is the cheapest in terms of computation, while WEP and TKIP are almost the same. This is a because both WEP and TKIP use the RC4 stream cipher, and TKIP only adds a little extra computation for the key mixing.

When the integrity check is factored in, AES and TKIP become the most expensive. This is due to the relatively high cost of the integrity function to that of WEP's.

In addition, we conducted and earlier analysis which contained an estimation of authentication costs, shown in figure 4.3. Unfortunately, the EAP authentication methods that we selected in this analysis were not included in the experiment due to lack of support. However, we can still see that the cost of EAP methods is far greater than that of WEP's authentication.

Based on our preliminary analysis we quickly concluded that the most significant element affecting the energy consumption of a wireless device security protection mechanism will be that associated with authentications. Similarly, we speculated that there will be very little differences across cryptographic protocols from an energy consumption perspective. While only one authentication is required to start a session, weak signals, reassociation, and roaming can all cause more authentications to take place. Therefore, it can be assumed that a session may have multiple authentication handshakes.

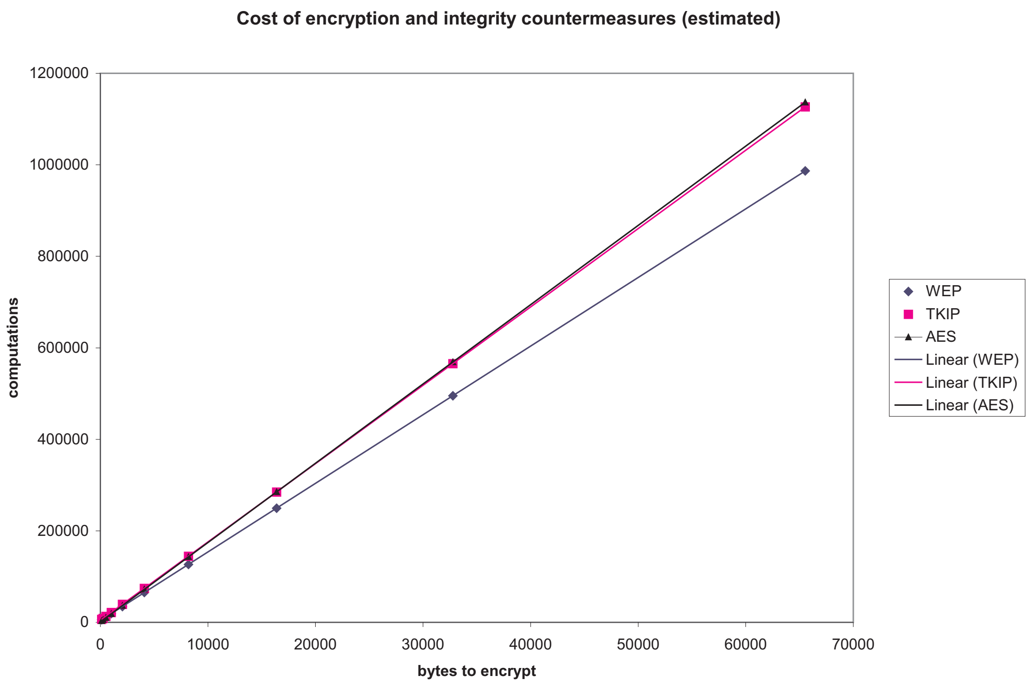


Figure 4.1: computations for confidentiality and integrity countermeasures

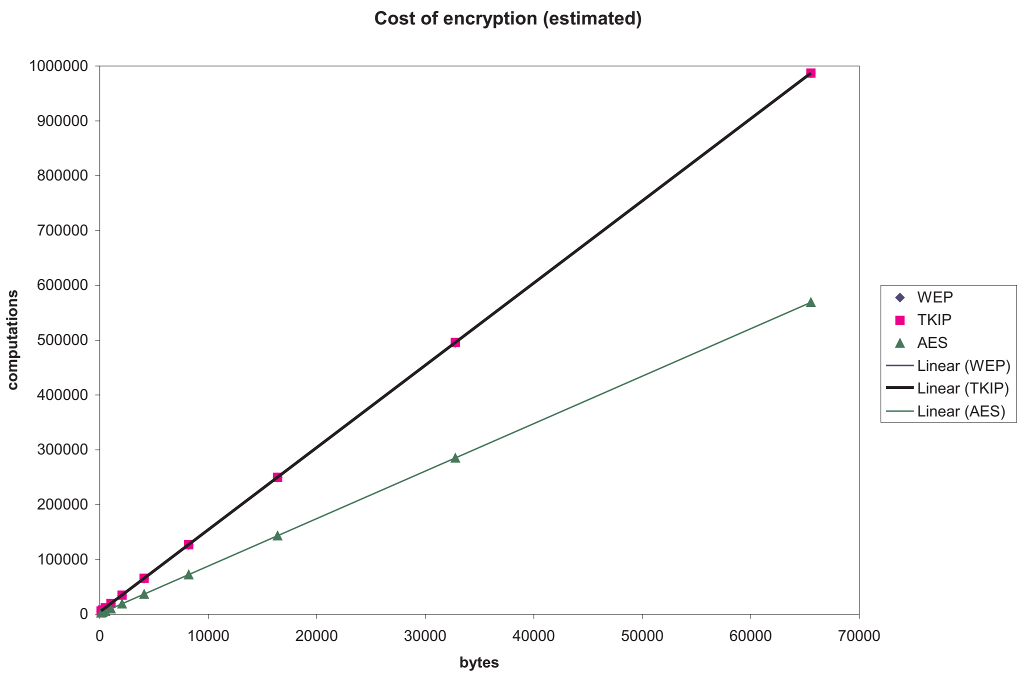
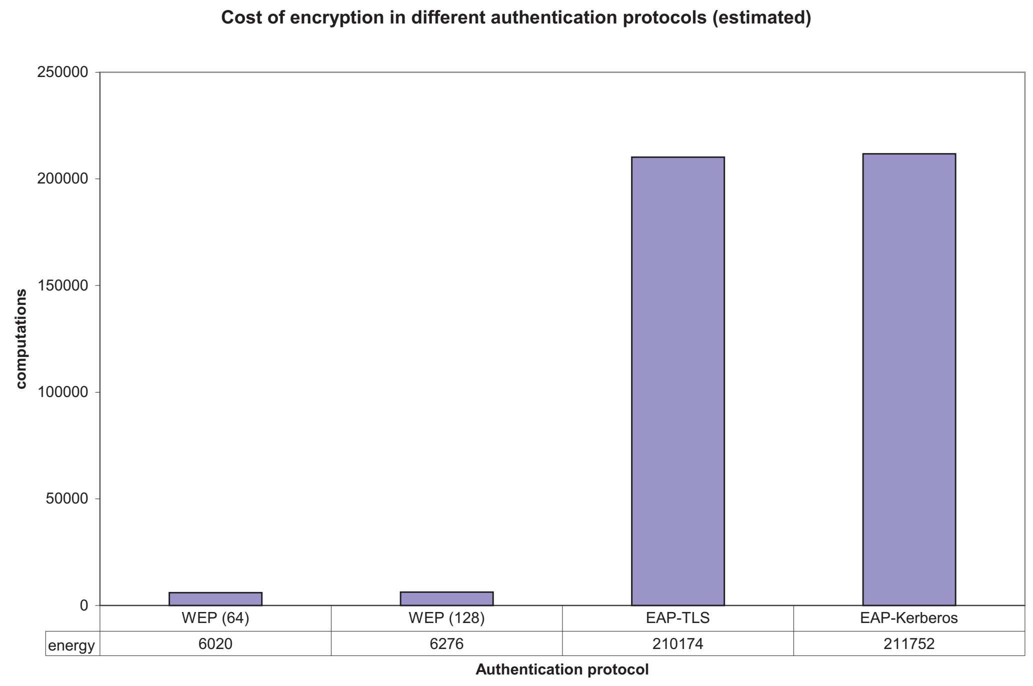


Figure 4.2: computations for confidentiality countermeasures



	WEP (64)	WEP (128)	EAP-TLS	EAP-Kerberos
energy	6020	6276	210174	211752

Figure 4.3: computations for availability countermeasures

Chapter 5

Experiment

5.1 Experiment Overview

The experiment was constructed for a basic scenario where we have a mobile device that wishes to retrieve a web page via the wireless channel. The test bed, depicted in figure 5.2, consists of a wireless client (supplicant), access point (authenticator), and RADIUS server (authentication server). Component information is described in table 5.1

Power measurements were obtained using Labview 7.1 by National Instruments[24]. This product obtains signals via a data acquisition (DAQ) card that connects to the PC. For this experiment, we used a 6062E multifunction DAQ card with a CB-68LP connector block. A Radio Shack Universal Breadboard was used for all wire connections.

In order to determine the power consumed, Labview measured the voltage drop over a 0.47 Ohm resistor to determine the current (equation 5.1). Using Joule's law (equation 5.2), we can determine the total power consumed (in Watts) during the measurement period. First, the current at each point (the sample rate is 1 millisecond) is calculated. The area under this curve is calculated and then multiplied by the voltage supplied by the AC adapter (5 volts). The total energy (Joules) used to perform each transaction is then calculated by multiplying the power consumed by time in seconds (equation 5.3).

$$I = \frac{V}{R} \tag{5.1}$$

$$P = I \times V \tag{5.2}$$

component	**vendor**	**model**	**relevant specs**
web server, RADIUS server, measurement system	Toshiba	Satellite A75-S206	2.8 GHz Mobile Intel Pentium4 Processor 518, 512 MB DDR SDRAM, Windows XP Professional SP2
mobile client	Compaq	iPAQ Pocket PC 3955	400 MHz Intel PXA250 Application Processor, 64 MB RAM, Windows Mobile 2003
access point	Cisco Systems, Inc.	Aironet 1231G	802.11g radio
measurement equipment	National Instruments	Labview 7.1 Academic Starter Kit	DAQ 6062E, CB-68LP connector block

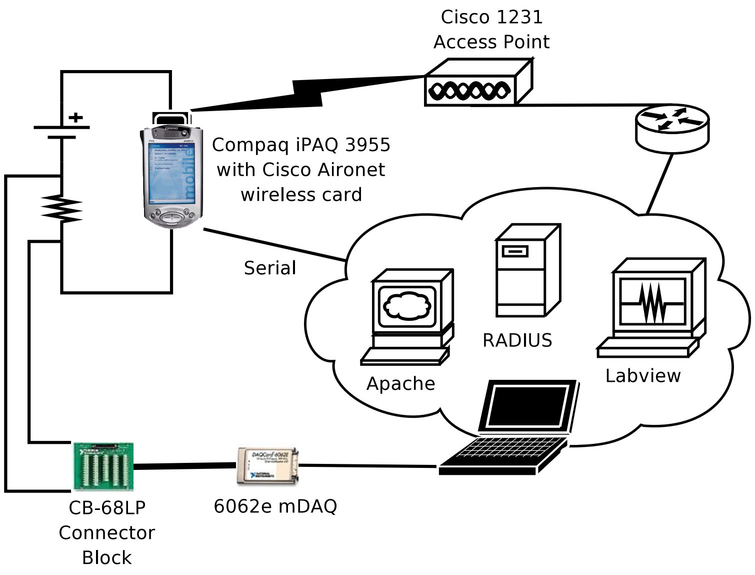


Figure 5.1: Experiment setup

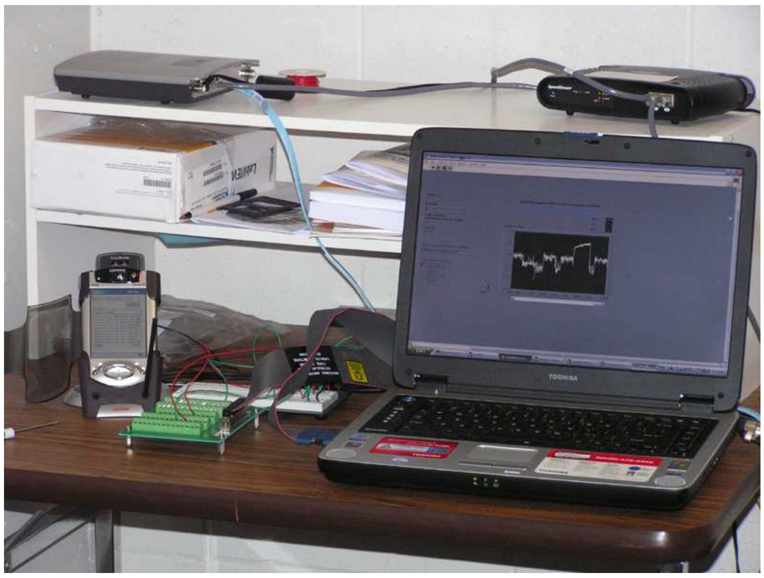


Figure 5.2: Experiment setup

$$E = P \times t \tag{5.3}$$

The iPAQ is connected directly to the measurement system via serial port. This allows us to send signals at the start and stop of each transaction, isolating the exact period that our transaction takes place.

To measure the cost of disconnection and reassociation, we use the access point to kill the connection between itself and the client. This may be done through either the command line interface or the web interface. However, the deauthentication via web interface takes longer to complete, and thus may complete the deauthentication outside of our measurement window. Because of this, we used the CLI.

To deauthenticate the client from the access point, enter enable mode and use the following command (where 000f.8fef.aab2 is our client card's MAC address).

```
WSSRL#clear dot11 client 000f.8fef.aab2
```

5.2 Network Infrastructure

The test bed was completely isolated from WPI's network in order to prevent interference and uncontrolled events (such as changes to the data). To prevent others from accidentally connecting to our test bed, we disabled beacon messages from being transmitted by the access point and enabled MAC filtering.

It was discovered during the experiment that running additional applications on the measurement system increased the frequency of LabVIEW errors. The errors that were encountered occurred because the input buffer was not emptied before more data was written to it. Because data was overridden, some results were invalid and the experiment had to be repeated.

5.3 Software

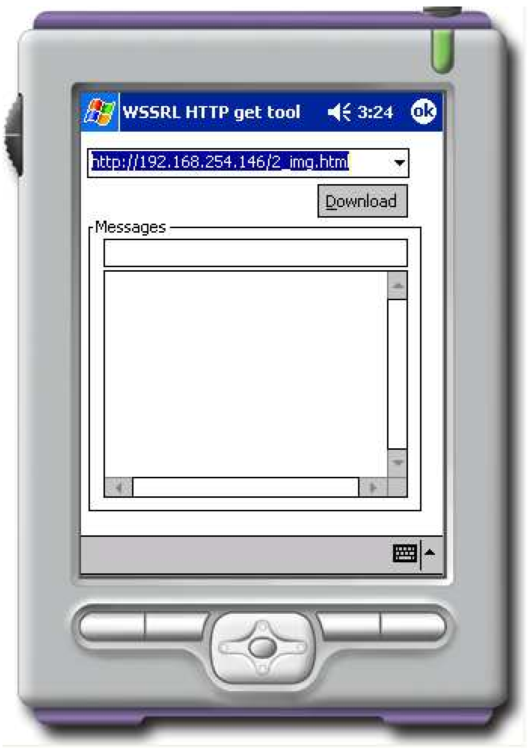
Apache 2, distributed by the Apache group, was chosen as the web server for this experiment. Apache is one of the most commonly used web servers, and is free under the GPL. Our server runs a basic installation which does not include CGI processing or additional features, such as SSL.

During the creation of this experiment, the packet analyzer Ethereal[7] was used to capture TCP and UDP traffic. This was necessary for verification that transactions were completing properly, and was exceedingly useful for troubleshooting. TCP traffic was observed to verify HTTP requests and responses, while UDP captures were used to verify RADIUS transactions.

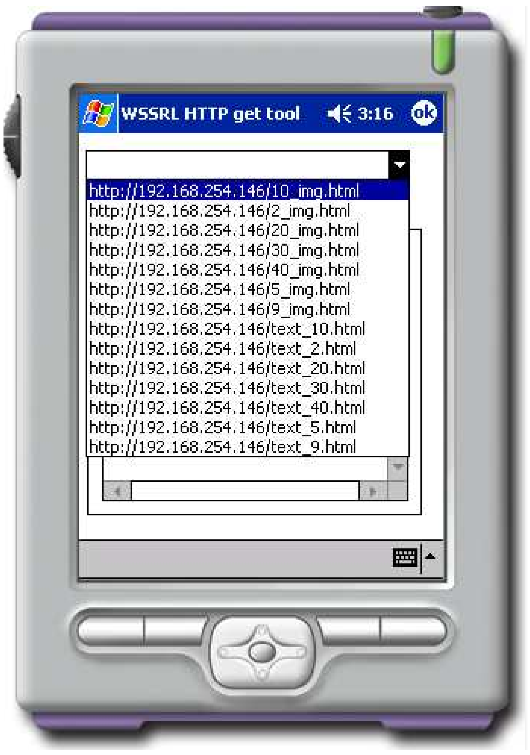
The PDA runs a special browser written for this project. The browser is extremely basic - it was designed with only three functions:

- send get requests to our web server
- receive and display ASCII representation of of objects
- send a signal to Labview at the start and end of every transaction

The address bar is a drop-down list of all the possible pages in the experiment. This removes the need to type in the URL for each scenario, therefore increasing speed and reducing error rate. Once the URL has been selected and the download button has been pressed, the application sends a start signal to Labview, retrieves all objects associated with the URL, and then sends Labview a stop signal. A message is displayed in the single-line text area indicating whether or not the page was successfully downloaded. The multi-line text area displays the ASCII representation of each object.



(a) browser interface



http://192.168.254.146/10_img.html
http://192.168.254.146/2_img.html
http://192.168.254.146/20_img.html
http://192.168.254.146/30_img.html
http://192.168.254.146/40_img.html
http://192.168.254.146/5_img.html
http://192.168.254.146/9_img.html
http://192.168.254.146/text_10.html
http://192.168.254.146/text_2.html
http://192.168.254.146/text_20.html
http://192.168.254.146/text_30.html
http://192.168.254.146/text_40.html
http://192.168.254.146/text_5.html
http://192.168.254.146/text_9.html

(b) dropdown address list

Figure 5.3: The simple browser used for the experiment

This application was developed using Microsoft Embedded Visual C++ [20]. This development environment was selected due to its integration with Microsoft ActiveSync, Microsoft Foundation Classes (MFC), and a variety of sample applications. Their HTTP sample application provided the base code for our tool. The code for this application is located in appendix C.

LabVIEW was used to gather all voltage measurements. The code used was heavily based on an example by National Instruments [23]. Analog input is acquired continuously in between two digital triggers. The recorded waveforms are written to files for later analysis. Our LabVIEW code is located in appendix A .

Once waveforms were obtained, they were run through two perl scripts. The first script, `power.pl`, calculates the power consumed in each run. The second script, `average.pl`, finds the average of the power expenditure in two different ways - over the entire data set, and over the IQR. There were often runs that were unusually long and greatly affected the results, so the IQR average was the one used for our measurements. Perl code used in this experiment is located in appendix B.

The RADIUS (remote authentication dial-in user service) software selected for this project was Funk Steel-belted RADIUS Enterprise Edition (SBR EE)[9]. Funk Software offers a fully-functional free 30-day trial of this server. The selection of this software was based on the supported interoperability with Cisco's products and proprietary protocols (LEAP and EAP-FAST). However, no EAP-FAST functionality could be found. Cisco Systems claimed that SBR EE supported EAP-FAST, but nowhere in Funk's documentation could we find a way to enable it. The authentication methods supported by Funk are LEAP, MD5-Challenge, TLS, and MS-CHAP-V2.

There were three different wireless client programs installed on the PDA, Cisco's Aironet Client Utility (ACU), Funk Odyssey Client for PPC 4.0beta, and the Meetinghouse AEGIS client[19]. The reason for multiple clients lies in the authentication support provided by each one. ACU could not be removed, as doing so also removed the driver for the wireless adapter. The only EAP authentication methods supported by this device are LEAP and EAP-FAST, which could be used in both open and WPA association modes.

Funk's Odyssey client added support for WPA, however, it would not associate with the access point. Upon examination with Ethereal, the problem seemed to lie in the client or AP. The RADIUS server sent the RADIUS ACCEPT message, but the client would always disconnect and start the authentication process over. Odyssey could clearly not be used with WPA, however, it did offer more authentication methods to be used with open 802.1x authentication. Although several other methods were configurable, only MD5-Challenge was common between it and SBR EE.

The AEGIS client also had difficulty with association methods other than open and shared, and was not used in testing. It did not add any configurations that we could not accomplish with the other two client programs, and had the worst interface. It did not give much feedback, making it very difficult to determine what was happening.

5.4 Workload

In any experimental setup of this nature, it is important to capture data while executing workloads which are "closely" representative of actual Internet traffic. Fortunately, over the last several years researchers have studied the problem of accurate representation of Internet workloads. In general, the network community has settled on a model for network traffic which goes under the name of "mice and elephants". In this model, mice are small objects that are transferred often, such as text messages, TCP acknowledgments, etc. Elephants are large objects, such as multimedia files, of which there are fewer occurrences.

Several studies have been conducted which examine network loads and their effects on performance. One such study out of the University of Washington[26] was used to construct the data transmitted during our experiment. In their research, Saroiu et al. compared HTTP traffic

Original data[26]			Experiment Workload	
-	object size (KB)	# of requests	% of listed requests total	instances in workload
1	9	1,412,104	22.305	22
2	2	3,007,720	47.509	48
3	333,000	21	0.0003	0
4	5	1,412,105	22.305	22
5	2,230	1,457	0.023	0
6	20	126,625	2	2
7	20	122,453	1.934	2
8	30	56,842	0.897	1
9	10	143,780	2.271	2
10	40	47,676	0.753	1

Table 5.1: Workload specifications

workload name	object size (KB)	workload construction
text2	2	single 2KB text-only HTML file
text5	5	single 5KB text-only HTML file
text9	9	single 9KB text-only HTML file
text10	10	single 10KB text-only HTML file
text20	20	single 20KB text-only HTML file
text30	20	single 30KB text-only HTML file
text40	40	single 40KB text-only HTML file
2img	2	48 <img src=...> in HTML file
5img	5	22 <img src=...> in HTML file
9img	9	22 <img src=...> in HTML file
10img	10	2 <img src=...> in HTML file
20img	20	2 <img src=...> in HTML file
30img	30	1 <img src=...> in HTML file
40img	40	1 <img src=...> in HTML file

Table 5.2: Workload

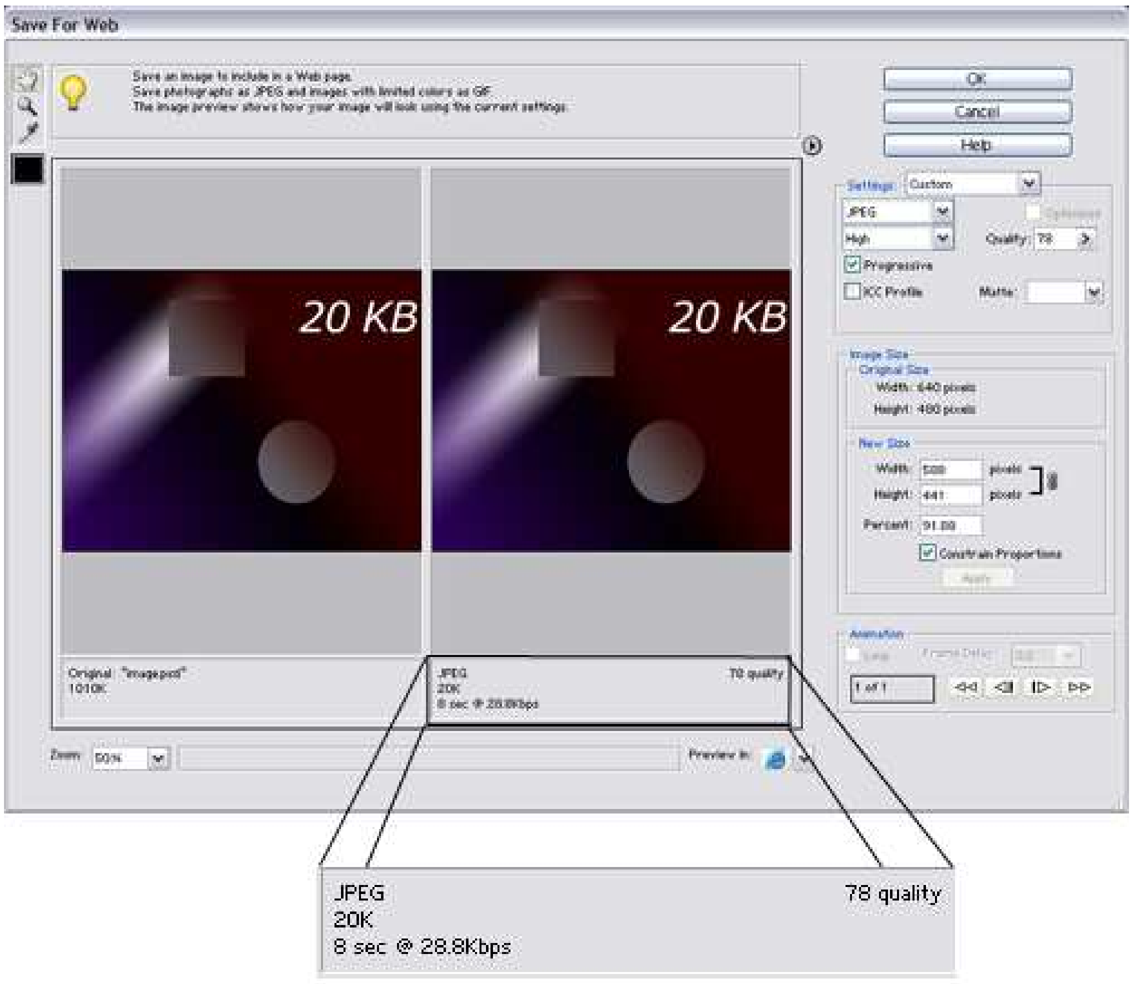
over various applications, such as WWW, Kazaa, and Gnutella. For this experiment, we are only focusing on WWW traffic, since surfing the web is a common use of mobile devices.

Table 5.1 shows relevant data from the research of Saroiu et al.[26] This data comes from their top ten bandwidth consuming objects. The data from this study became the basis for our workload creation. In effect, we set out to reproduce workloads which highly correlates the type of objects and traffic experienced by Saroiu, et al.[26] while at the same time making it possible to understand the behavior of a handheld device. For each object, we use the number of requests over the total number of requests in the top ten to discover how many instances of that object should appear.

Once this was done, three types of pages were constructed to model the Internet:

- text-only pages
- text and many smaller images
- text and fewer larger images

The text-only pages are stand-alone (call no additional objects), and there is one page for each object size listed in table 5.2. Seven image pages were constructed for this experiment, also based off the numbers in table 5.2. The HTML pages are the bare minimum, containing only the basic opening and closing HTML tags and the img tags necessary to request each image. The number


Save For Web

Save an image to include in a Web page.
Save photographs as JPEG and images with limited colors as GIF.
The image preview shows how your image will look using the current settings.

Settings: Custom
JPEG
High
Quality: 78
[✓] Progressive
[ ] ICC Profile
Matte:

Image Size
Original Size
Width: 640 pixels
Height: 480 pixels
New Size
Width: 588 pixels
Height: 441 pixels
Percent: 91.88
[✓] Constrain Proportions

Animation
1 of 1

Original: "image.psd"
1010K

JPEG
20K
8 sec @ 28.8Kbps
78 quality

Zoom: 50%

JPEG
20K
8 sec @ 28.8Kbps
78 quality

Figure 5.4: exporting image file for the experiment

of times that an image is called from its accompanying page corresponds with the instance field of table 5.2.

The images used for this experiment were created using Adobe Photoshop Elements [1]. All images are based on the same basic image, but vary in the title layer (which labels the image with it's size for easy identification) and the final dimensions and quality. After each image was finished as a .psd file, it was exported for the web as a jpeg file. In order to achieve the desired file size, the image dimensions and jpeg quality were altered until the file size was correct. Figure 5.4 shows the screen used for this process.

The workload objects can be found in appendix D.

Chapter 6

Results

6.1 Encryption

Figure 6.1 depicts our measurements of workload transfers when varying the encryption cipher. For these measurements, the client adapter was configured using the Cisco ACU. All measurements are taken *after* the client was authenticated and associated, so they convey only the cost of confidentiality and integrity countermeasures.

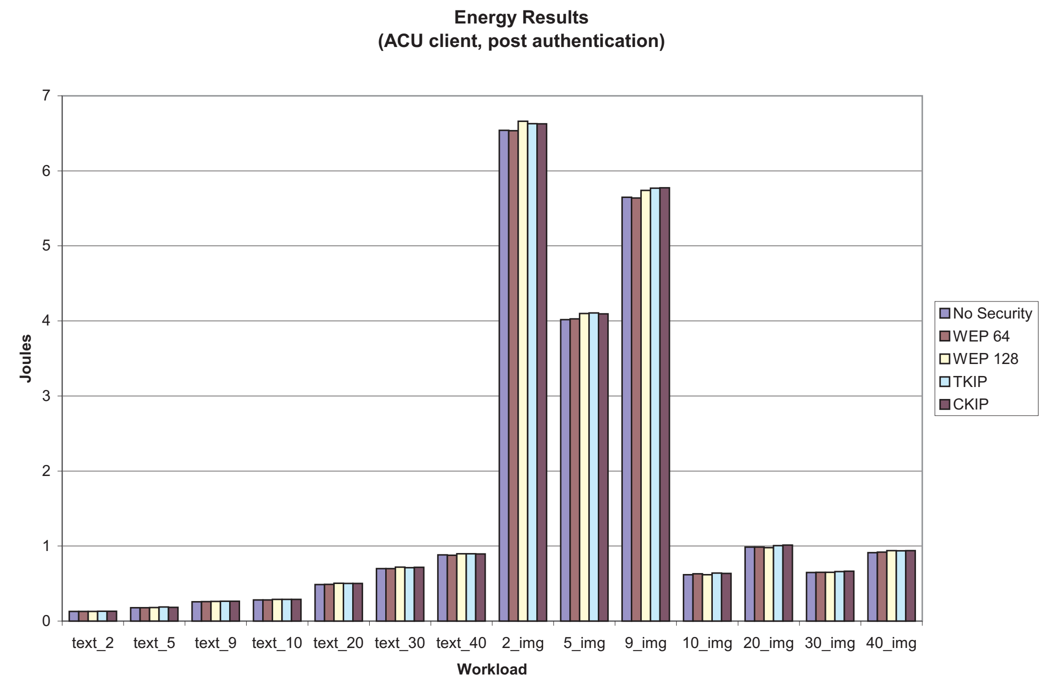


Figure 6.1: Energy used over workloads after association established

From this data, we can see that the impact of encryption on the battery life is very minimal. Workloads which only requested one object, namely the text-only workloads, showed trivial energy differences between profiles. This is no surprise, as all of the ciphers shown here are based on the RC4 stream cipher and RC4 is very cheap in terms of energy. In the workloads that require more requests, specifically the 2img, 5img, and 9img workloads, you can see how the different variations on WEP affect the total energy consumed. In these workloads we can see how the 128-bit ciphers break further away from the rest. The cost of 64-bit WEP remains very close to that of no security.

Our first analysis, showed that adding TKIP key mixing and Michael to 128-bit WEP increased the number of computations by about a 2%. This analysis did not use the same implementations shown previously in chapter 4, but rather an informal diagram on a university website. When we extended our analysis for variable message length using Johnston's implementation[13], as shown in chapter 4, we found the increase to be around 1.13%, for the sizes shown. Here, the largest difference is 1.31%. This may be attributed to a different implementation in the client and access point firmware. It is also likely that all operations in C, which is what we examined in our analysis, do not consume the same amount of energy since some may be broken into more or fewer assembly instructions.

Although the latest firmware for our access point supports AES-CCM, it currently only supports it on certain AP models. Our AP is *not* one of those that supports AES-CCM at the present time, despite the fact that it can be configured on the device. Because of this, we were not able to obtain empirical measurements for this new protocol.

6.2 Authentication

Mobile clients do not necessarily stay connected to the same access point during an entire session. Several factors may cause disconnection to occur. The client may wander outside the range of the access point, the AP may deauthenticate when the authentication period expires, the connection may be dropped due to low signal strength, etc.

In order to see the difference in cost of disconnection, we took measurements with three different authentication types: open, shared, and LEAP. LEAP is configured without WPA key management, as WPA requires TKIP or AES-CCM as a cipher. Additionally, we could not perform open and shared authentication with TKIP or AES-CCM. Therefore, WPA measurements are not grouped with these results. As anticipated, the differences between open and shared authentication are trivial. To close the connection, we deauthenticated the client through the AP's CLI. We took measurements using two different clients, Cisco ACU and Funk Odyssey client, as Odyssey supported additional EAP methods. The results are shown in figures 6.2(a) and 6.2(b), respectively.

Both clients consume approximately the same amount of energy for open and shared authentication. However, in figure 6.2(b), the cost of LEAP authentication is significantly greater than in figure 6.2(a). MD5-Challenge EAP authentication may not be compared between the clients, as ACU does not support this method.

The 2img workload only transferred long enough to inject 3 disconnections. In order to gain more data points, a new workload was created - the disconnect workload. This is simply an extended version of 2img which should be long enough to insert 7 disconnections. However, due to time constraints, only 0-5 disconnections are recorded. The results are shown in figures 6.3(a) and 6.3(b). These graphs also contain trend lines and correlation coefficients. For the Odyssey client, our sample points create a line with little deviation from the data points. The trend lines for the ACU client are not as tight, but still have a high correlation. While the energy usage is increased with this new workload, the trends appears the same. The Odyssey client consumes significantly more energy than ACU for LEAP authentication.

In order to identify the cause of this discrepancy, we performed some traces during disconnection with Ethereal. Traces captured on the wired side (between the access point and RADIUS server) were identical. For the wireless channel, we used an Orinoco wireless card on a laptop running the Knoppix STD distribution. This configuration allowed us to put the card into promiscuous mode and monitor the traffic exchange between the PDA and access point.

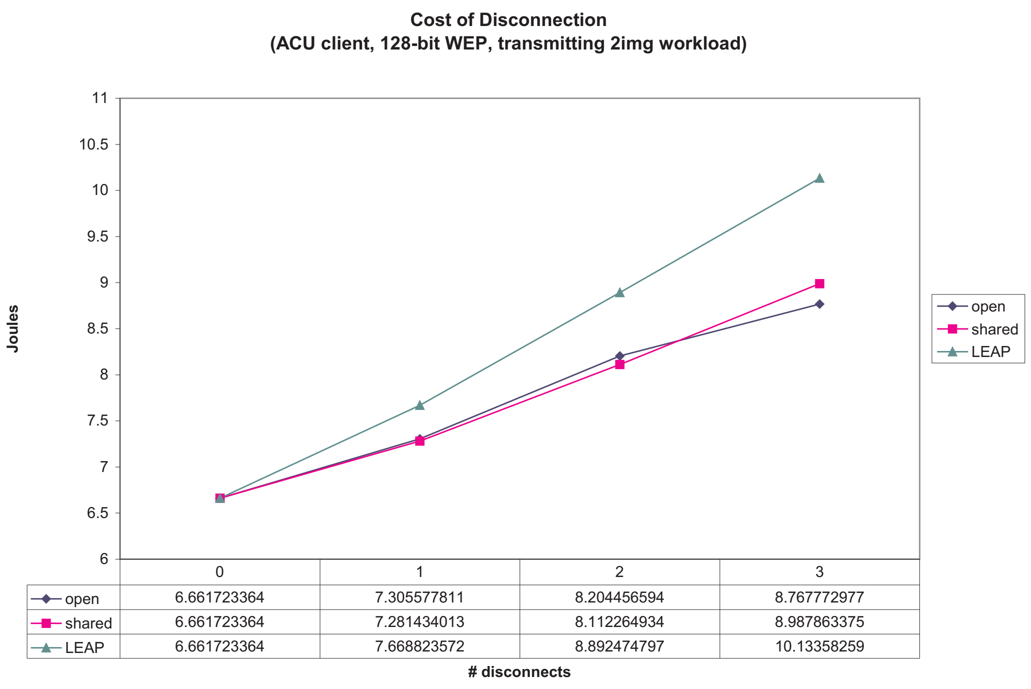
We collected 10 traces for each client, all of which had results similar to those shown in figure

6.4. What we found is that the time between the reassociation request and subsequent WEP-encrypted packet were 3 to 4 seconds apart with the Odyssey client, but only 1 and 2 seconds apart with the ACU client. Although ACU sent more packets (because of the LLC transaction), it completed about 1 to 2 seconds faster than Odyssey. Therefore, we conclude that the discrepancy between the two clients is a result of time difference in the idle state.

While our final experimental setup used Windows Mobile 2003, we had previously used Pocket PC 2002. We upgraded in order to achieve more configurations, as PPC 2002 did not support WPA key management, and therefore did not support some of the newer ciphers. We did, however, take some measurements while PPC 2002 was running on the PDA.

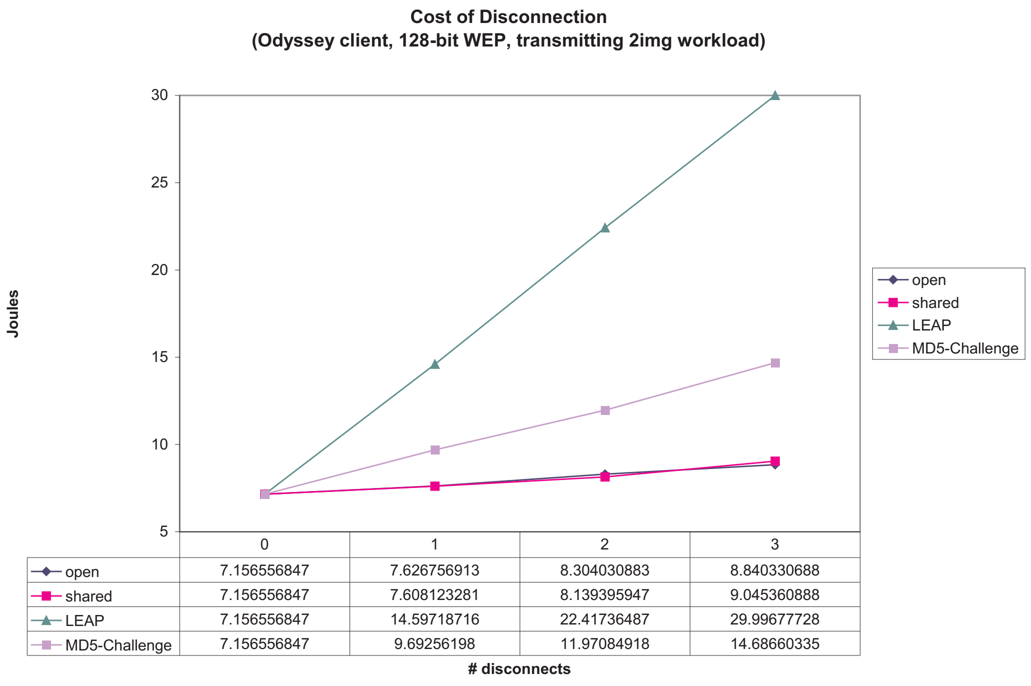
Figure 6.5 shows the results of measurements while the 2img workload was transferring, the client adapter was configured with the Odyssey client, and the PDA had Pocket PC 2002 as its OS. This graph varies greatly from figure 6.2(a), and looks similar to figure 6.2(b). However, the cost of LEAP in figure 6.5 is almost double that of the cost in 6.2(b). We believe that the reason behind this discrepancy lies in the 802.1x support. PPC 2002 requires that a program called "802.1x Backport" be installed to use EAP authentication. Windows Mobile 2003, however includes 802.1x support in the operating system.

As discussed in the analysis, we can assume that multiple authentication exchanges may take place. In fact, a study of a campus WLAN[16] showed that 18% of sessions roam at least once. Of those sessions, 60% roamed within a subnet, which means that they had to reauthenticate with a new access point, but kept the same IP address. The remaining 40% had to undergo the complete association in addition to DHCP process.



	0	1	2	3
◆ open	6.661723364	7.305577811	8.204456594	8.767772977
■ shared	6.661723364	7.281434013	8.112264934	8.987863375
▲ LEAP	6.661723364	7.668823572	8.892474797	10.13358259

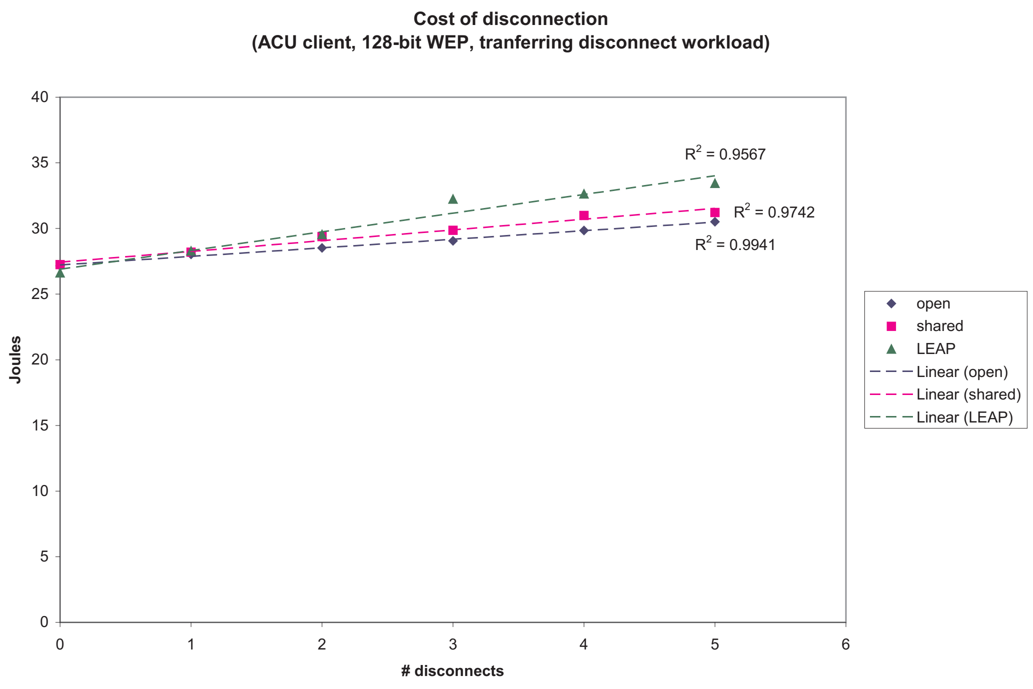
(a) ACU client



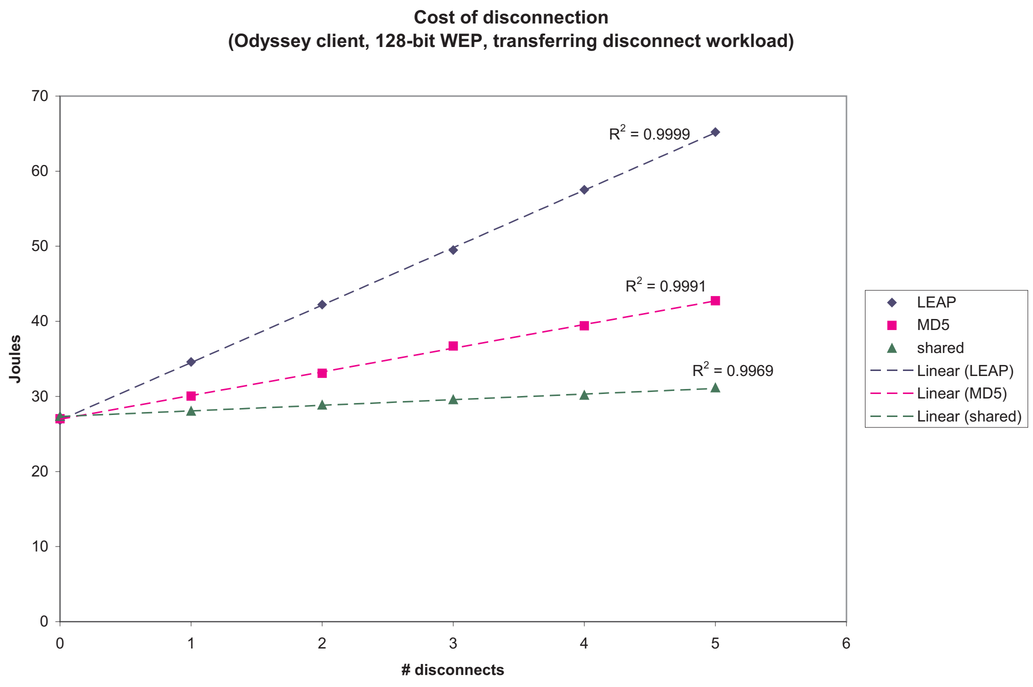
	0	1	2	3
◆ open	7.156556847	7.626756913	8.304030883	8.840330688
■ shared	7.156556847	7.608123281	8.139395947	9.045360888
▲ LEAP	7.156556847	14.59718716	22.41736487	29.99677728
■ MD5-Challenge	7.156556847	9.69256198	11.97084918	14.68660335

(b) Odyssey client

Figure 6.2: Transfer of 2img workload with disconnection

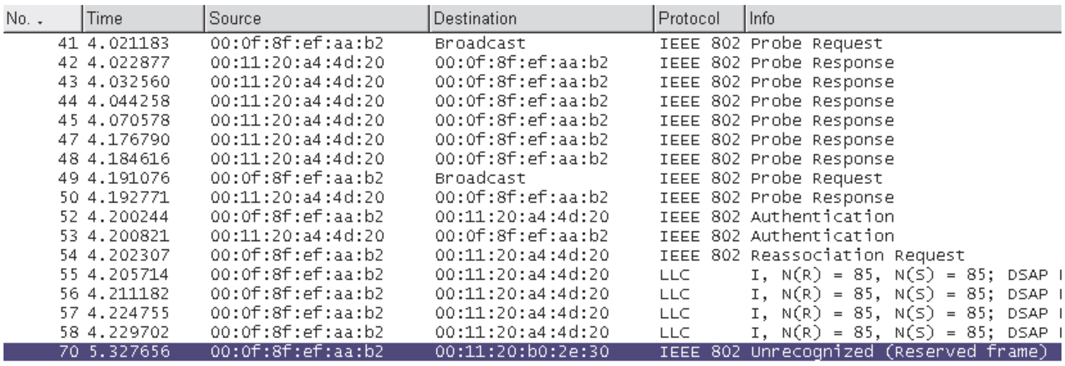


(a) ACU client



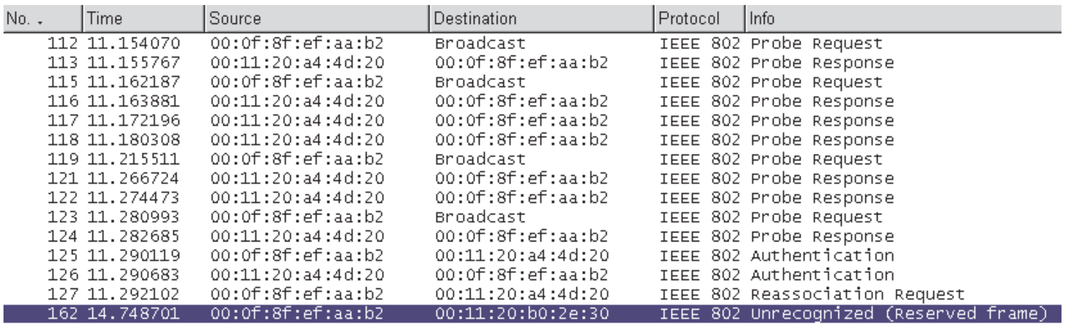
(b) Odyssey client

Figure 6.3: Transfer of disconnect workload with disconnection



```
No. .  Time        Source             Destination        Protocol  Info
    41 4.021183    00:0f:8f:ef:aa:b2  Broadcast          IEEE 802  Probe Request
    42 4.022877    00:11:20:a4:4d:20  00:0f:8f:ef:aa:b2  IEEE 802  Probe Response
    43 4.032560    00:11:20:a4:4d:20  00:0f:8f:ef:aa:b2  IEEE 802  Probe Response
    44 4.044258    00:11:20:a4:4d:20  00:0f:8f:ef:aa:b2  IEEE 802  Probe Response
    45 4.070578    00:11:20:a4:4d:20  00:0f:8f:ef:aa:b2  IEEE 802  Probe Response
    47 4.176790    00:11:20:a4:4d:20  00:0f:8f:ef:aa:b2  IEEE 802  Probe Response
    48 4.184616    00:11:20:a4:4d:20  00:0f:8f:ef:aa:b2  IEEE 802  Probe Response
    49 4.191076    00:0f:8f:ef:aa:b2  Broadcast          IEEE 802  Probe Request
    50 4.192771    00:11:20:a4:4d:20  00:0f:8f:ef:aa:b2  IEEE 802  Probe Response
    52 4.200244    00:0f:8f:ef:aa:b2  00:11:20:a4:4d:20  IEEE 802  Authentication
    53 4.200821    00:11:20:a4:4d:20  00:0f:8f:ef:aa:b2  IEEE 802  Authentication
    54 4.202307    00:0f:8f:ef:aa:b2  00:11:20:a4:4d:20  IEEE 802  Reassociation Request
    55 4.205714    00:0f:8f:ef:aa:b2  00:11:20:a4:4d:20  LLC       I, N(R) = 85, N(S) = 85; DSAP
    56 4.211182    00:0f:8f:ef:aa:b2  00:11:20:a4:4d:20  LLC       I, N(R) = 85, N(S) = 85; DSAP
    57 4.224755    00:0f:8f:ef:aa:b2  00:11:20:a4:4d:20  LLC       I, N(R) = 85, N(S) = 85; DSAP
    58 4.229702    00:0f:8f:ef:aa:b2  00:11:20:a4:4d:20  LLC       I, N(R) = 85, N(S) = 85; DSAP
    70 5.327656    00:0f:8f:ef:aa:b2  00:11:20:b0:2e:30  IEEE 802  Unrecognized (Reserved frame)
```

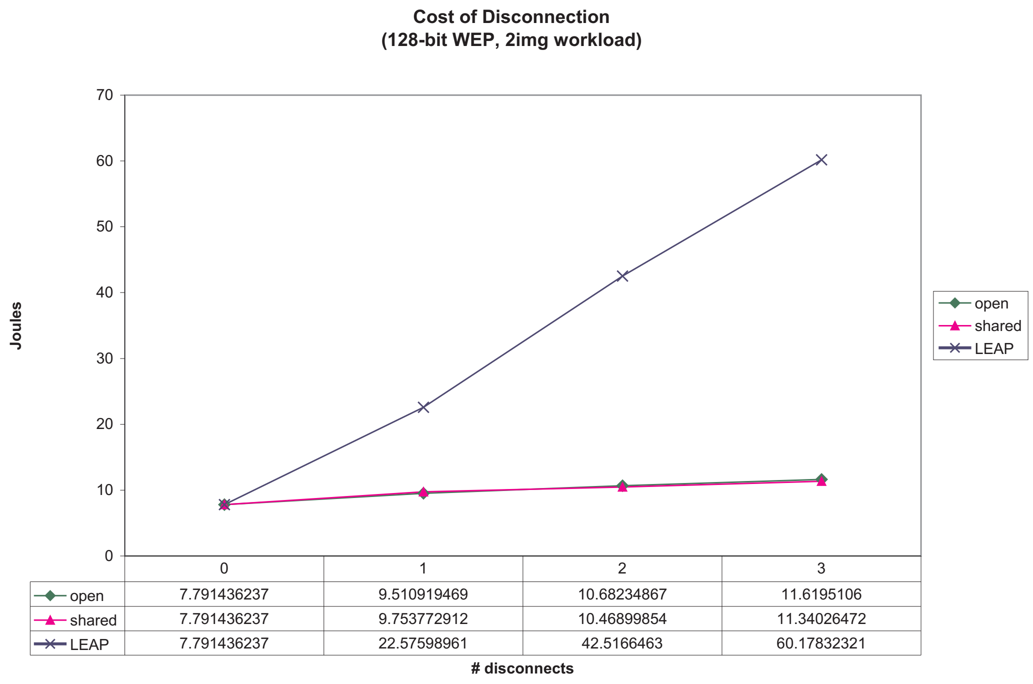
(a) ACU client



```
No. .  Time        Source             Destination        Protocol  Info
   112 11.154070   00:0f:8f:ef:aa:b2  Broadcast          IEEE 802  Probe Request
   113 11.155767   00:11:20:a4:4d:20  00:0f:8f:ef:aa:b2  IEEE 802  Probe Response
   115 11.162187   00:0f:8f:ef:aa:b2  Broadcast          IEEE 802  Probe Request
   116 11.163881   00:11:20:a4:4d:20  00:0f:8f:ef:aa:b2  IEEE 802  Probe Response
   117 11.172196   00:11:20:a4:4d:20  00:0f:8f:ef:aa:b2  IEEE 802  Probe Response
   118 11.180308   00:11:20:a4:4d:20  00:0f:8f:ef:aa:b2  IEEE 802  Probe Response
   119 11.215511   00:0f:8f:ef:aa:b2  Broadcast          IEEE 802  Probe Request
   121 11.266724   00:11:20:a4:4d:20  00:0f:8f:ef:aa:b2  IEEE 802  Probe Response
   122 11.274473   00:11:20:a4:4d:20  00:0f:8f:ef:aa:b2  IEEE 802  Probe Response
   123 11.280993   00:0f:8f:ef:aa:b2  Broadcast          IEEE 802  Probe Request
   124 11.282685   00:11:20:a4:4d:20  00:0f:8f:ef:aa:b2  IEEE 802  Probe Response
   125 11.290119   00:0f:8f:ef:aa:b2  00:11:20:a4:4d:20  IEEE 802  Authentication
   126 11.290683   00:11:20:a4:4d:20  00:0f:8f:ef:aa:b2  IEEE 802  Authentication
   127 11.292102   00:0f:8f:ef:aa:b2  00:11:20:a4:4d:20  IEEE 802  Reassociation Request
   162 14.748701   00:0f:8f:ef:aa:b2  00:11:20:b0:2e:30  IEEE 802  Unrecognized (Reserved frame)
```

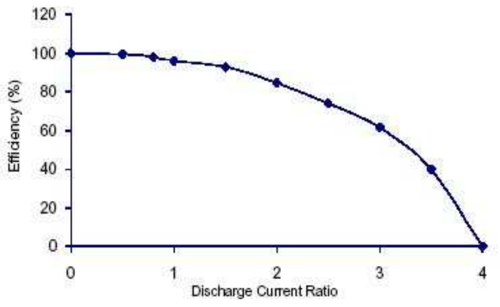
(b) Odyssey client

Figure 6.4: wireless traces of LEAP reauthentication

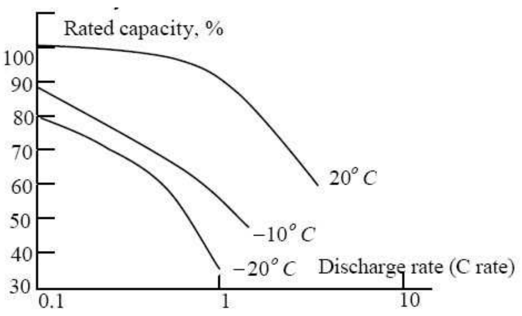


	0	1	2	3
◆ open	7.791436237	9.510919469	10.68234867	11.6195106
▲ shared	7.791436237	9.753772912	10.46899854	11.34026472
× LEAP	7.791436237	22.57598961	42.5166463	60.17832321

Figure 6.5: Transfer of 2img workload with disconnection (Pocket PC 2002, Odyssey client)



(a) Discharge rate vs. battery efficiency[27]



(b) Battery capacity vs. discharge rate[18]

Figure 6.6: Rate of battery discharge

6.3 Effect on Battery Life

The primary battery on our handheld device has a life of 1400mAh. The use of the wireless card requires that the expansion pack also have a battery, which provides an additional 920mAh. Equations 6.1 and 6.2 show the translation of these battery capacities into Joules. Both are rated with 3.7V. This accounts for an energy capacity of 30,902.4 Joules.

$$\begin{aligned} 3.7V \times 1400mAh &= 5,180mWh \\ 5,180mWh \times 3600\frac{s}{h} \times \frac{1W}{1000mW} &= 18,648Ws \\ &= 18,648J \end{aligned} \tag{6.1}$$

$$\begin{aligned} 3.7V \times 920mAh &= 3,404mWh \\ 3,404mWh \times 3600\frac{s}{h} \times \frac{1W}{1000mW} &= 12,254.4Ws \\ &= 12,254.4J \end{aligned} \tag{6.2}$$

These estimates follow the assumption that battery power dissipates linearly. This is not true in practice. Battery capacity, in fact, varies with the discharge rate (figure 6.6(a)). Figure 6.6(b) shows a similar curve, with the percentage of the rated capacity and different temperatures.

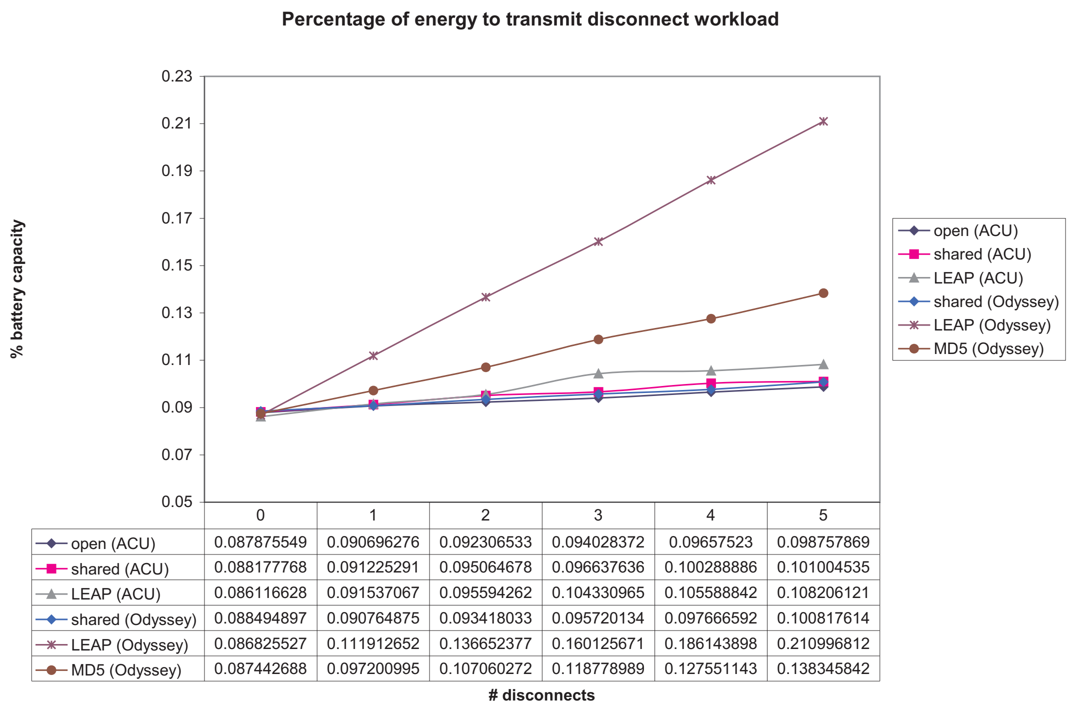
With these capacity values, we can now estimate the percentage of the battery that was consumed during our experiment. Because we do not have the discharge rate available, we will assume that the battery is at full capacity for each calculation.

Figure 6.7 depicts the percentage of the battery's total energy consumed while transferring the disconnect workload, with 0 to 5 disconnections occurring. From these results, we can determine the approximate cost, in terms of battery percentage, for each reauthentication. These approximations are shown in table 6.1.

	open (ACU)	shared (ACU)	LEAP (ACU)	shared (Odyssey)	MD5 (Odyssey)	LEAP (Odyssey)
% battery capacity	0.0021	0.0027	0.0046	0.0024	0.0102	0.0248

Table 6.1: Percent battery used per reauthentication

We can see that open authentication with the ACU client has the lowest energy cost, at 0.0021%. The client would have to be disconnected approximately 47,000 times in order for the entire battery

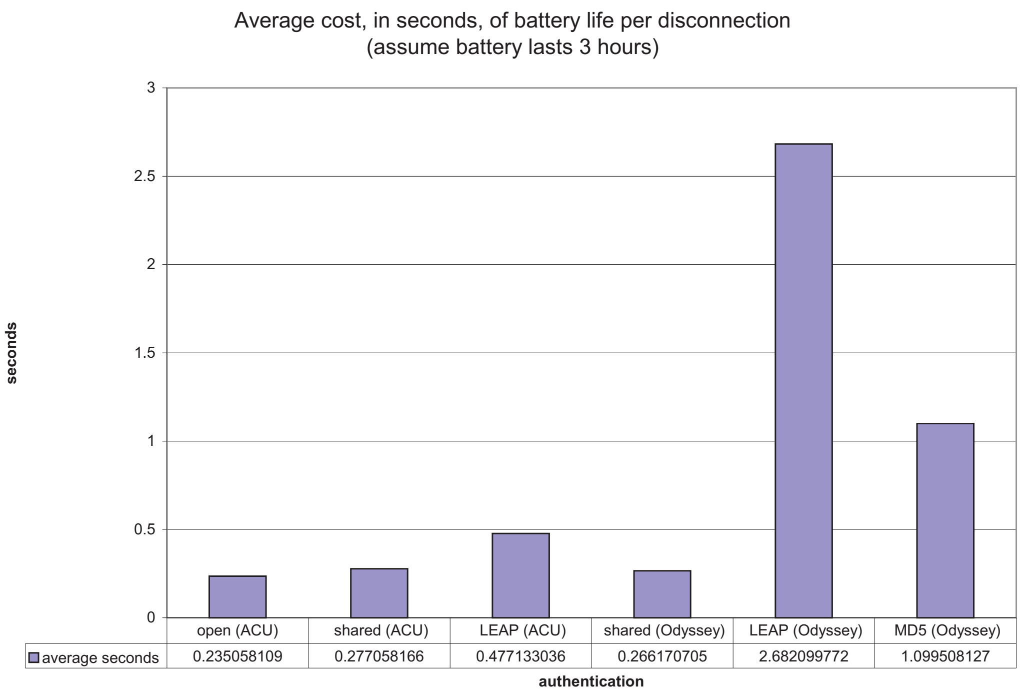


	0	1	2	3	4	5
open (ACU)	0.087875549	0.090696276	0.092306533	0.094028372	0.09657523	0.098757869
shared (ACU)	0.088177768	0.091225291	0.095064678	0.096637636	0.100288886	0.101004535
LEAP (ACU)	0.086116628	0.091537067	0.095594262	0.104330965	0.105588842	0.108206121
shared (Odyssey)	0.088494897	0.090764875	0.093418033	0.095720134	0.097666592	0.100817614
LEAP (Odyssey)	0.086825527	0.111912652	0.136652377	0.160125671	0.186143898	0.210996812
MD5 (Odyssey)	0.087442688	0.097200995	0.107060272	0.118778989	0.127551143	0.138345842

Figure 6.7: Percent of energy consumed by transfer of disconnect workload with deauthentication

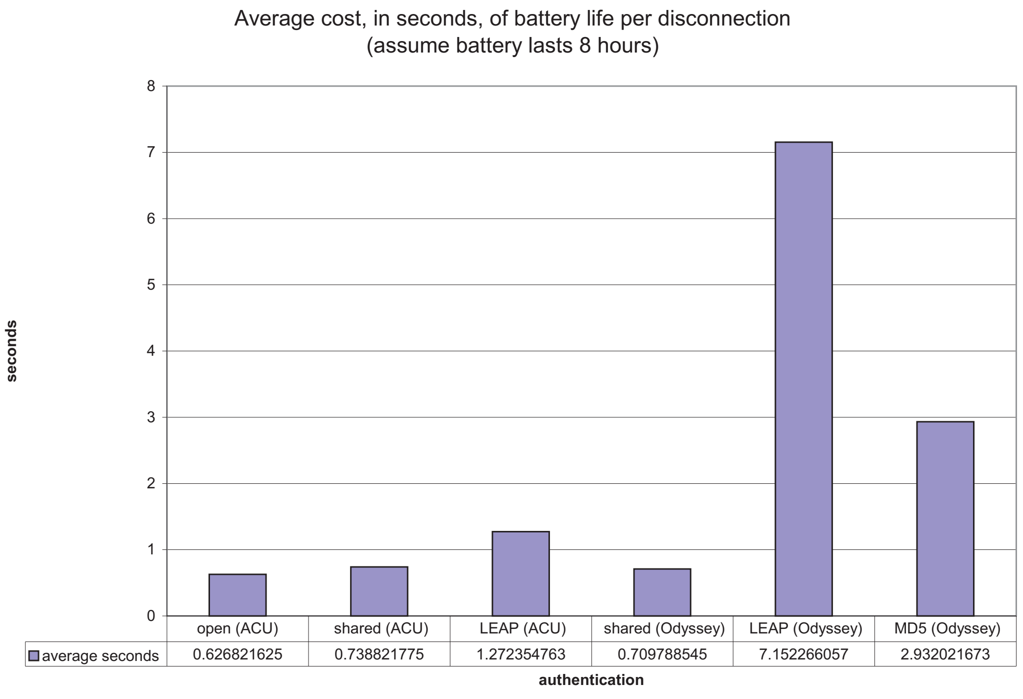
to be used. On the other end of the spectrum, LEAP authentication with the Odyssey client uses 0.0248% of the battery for each authentication. Under this profile, 4,000 disconnections will utlize the entire battery. In practice, both of these numbers would be lower as the battery capacity will reduce with each disconnection, and the battery will discharge at a faster rate. However, we can still see that LEAP with the Odyssey client exhausts that battery in the order of 10 times faster than open authentication.

The cost of each disconnection, in terms of time, is dependent on the frequency of usage. A PDA, for example, may last 12 days without charging if it is not turned on. If it is in constant use, however, it may only last 3 hours. Figures 6.8 and 6.9 give estimates of the time cost of each disconnection, assuming 3 and 8 hours of battery life, respectively. These graphs show the average delta between disconnection measurements, not the cost of transferring the workload with disconnection. In this data, we see that the authentication profile that consumes the greatest amount of energy only takes seconds off the battery life. It also shows that the longer the battery life, the greater the energy impact of each reauthentication. Longer usage will all require more authentication, as authentication expires after a fixed amount of time. Therefore, authentication will tend to have a higher cost when the mobile device must be in use for longer periods of time.



	open (ACU)	shared (ACU)	LEAP (ACU)	shared (Odyssey)	LEAP (Odyssey)	MD5 (Odyssey)
▨ average seconds	0.235058109	0.277058166	0.477133036	0.266170705	2.682099772	1.099508127

Figure 6.8: Estimated time cost of disconnection for 3 hours of battery life



	open (ACU)	shared (ACU)	LEAP (ACU)	shared (Odyssey)	LEAP (Odyssey)	MD5 (Odyssey)
▨ average seconds	0.626821625	0.738821775	1.272354763	0.709788545	7.152266057	2.932021673

Figure 6.9: Estimated time cost of disconnection for 8 hours of battery life

workload: 2img			
Profile	SM	CE	Q
none	0	0	0
WEP 64	1.52588E-05	0.770240511	1.98104E-05
WEP 128	1.000015259	0.895729664	1.1164253
CKIP+MMH	5.000015259	0.861278412	5.805341442
WPA-LEAP	8.125	1.245524641	6.523355485

workload: 5img			
Profile	SM	CE	Q
none	0	0	0
WEP 64	1.52588E-05	0.787005339	1.93884E-05
WEP 128	1.000015259	0.858286429	1.16512999
CKIP+MMH	5.000015259	0.853118731	5.86086681
WPA-LEAP	8.125	1.246642802	6.517504444

workload: 9img			
Profile	SM	CE	Q
none	0	0	0
WEP 64	1.52588E-05	0.766275188	1.99129E-05
WEP 128	1.000015259	0.867573113	1.152658195
CKIP+MMH	5.000015259	0.901505405	5.546295378
WPA-LEAP	8.125	1.278303043	6.35608281

workload: 10img			
Profile	SM	CE	Q
none	0	0	0
WEP 64	1.52588E-05	0.787970407	1.93647E-05
WEP 128	1.000015259	0.775659462	1.289245226
CKIP+MMH	5.000015259	0.793237619	6.303300724
WPA-LEAP	8.125	1.17920814	6.890217022

workload: 20img			
Profile	SM	CE	Q
none	0	0	0
WEP 64	1.52588E-05	0.776328505	1.96551E-05
WEP 128	1.000015259	0.768444326	1.301350306
CKIP+MMH	5.000015259	0.802717582	6.228859776
WPA-LEAP	8.125	1.177750564	6.898744306

workload: 30img			
Profile	SM	CE	Q
none	0	0	0
WEP 64	1.52588E-05	0.779827861	1.95669E-05
WEP 128	1.000015259	0.779227161	1.283342405
CKIP+MMH	5.000015259	0.79340064	6.302005577
WPA-LEAP	8.125	1.171906383	6.93314766

workload: 40img			
Profile	SM	CE	Q
none	0	0	0
WEP 64	1.52588E-05	0.78215137	1.95087E-05
WEP 128	1.000015259	0.801750908	1.247289212
CKIP+MMH	5.000015259	0.801681715	6.236908196
WPA-LEAP	8.125	1.18176792	6.8752924

Figure 6.10: Results applied to the trade-off model (image workloads)

6.4 Application of the Trade-off Model

The data shown in section 6.1 provides us with all information necessary to illustrate the usage of our trade-off model. Figures 6.10 and 6.11 show the resulting calculations of applying one authentication and one transfer of each workload. The WEP results assume shared key authentication. In all cases, the quotient follows the intuition that more secure profiles have higher countermeasure-energy quotient values. Of course, these results are highly dependent on our proxy, and trends may change with a more comprehensive and accurate measure of security.

Examining the results for workload "text2", we can see how putting restrictions on parameter values yields the most appropriate protocol. If the transfer of this workload were limited to 1J, then CKIP with MMH would be the best choice, as it gives the most security for that energy constraint. Were a minimum security profile of 5 required, then the best option would be WPA with LEAP authentication. Combining these two restraints so that both a minimum profile of 5 a maximum energy consumption of 1J were required, then CKIP+MMH would be the only option of those presented here.

workload: text2			
Profile	SM	CE	Q
none	0	0	0
WEP 64	1.52588E-05	0.776336518	1.97E-05
WEP 128	1.000015259	0.776833221	1.287297
CKIP+MMH	5.000015259	0.777515253	6.430762
WPA-LEAP	8.125	1.160459201	7.001539

workload: text5			
Profile	SM	CE	Q
none	0	0	0
WEP 64	1.52588E-05	0.775072703	1.97E-05
WEP 128	1.000015259	0.776978357	1.287057
CKIP+MMH	5.000015259	0.780319861	6.407648
WPA-LEAP	8.125	1.165493172	6.971298

workload: text9			
Profile	SM	CE	Q
none	0	0	0
WEP 64	1.52588E-05	0.777134496	1.96E-05
WEP 128	1.000015259	0.779142175	1.283482
CKIP+MMH	5.000015259	0.782672134	6.388391
WPA-LEAP	8.125	1.163273358	6.984601

workload: text10			
Profile	SM	CE	Q
none	0	0	0
WEP 64	1.52588E-05	0.776139023	1.97E-05
WEP 128	1.000015259	0.781897125	1.27896
CKIP+MMH	5.000015259	0.78210542	6.39302
WPA-LEAP	8.125	1.164283272	6.978542

workload: text20			
Profile	SM	CE	Q
none	0	0	0
WEP 64	1.52588E-05	0.7773188	1.96E-05
WEP 128	1.000015259	0.790407784	1.265189
CKIP+MMH	5.000015259	0.787880451	6.34616
WPA-LEAP	8.125	1.169980582	6.94456

workload: text30			
Profile	SM	CE	Q
none	0	0	0
WEP 64	1.52588E-05	0.775360082	1.97E-05
WEP 128	1.000015259	0.795495689	1.257097
CKIP+MMH	5.000015259	0.793349112	6.302415
WPA-LEAP	8.125	1.170352306	6.942354

workload: text40			
Profile	SM	CE	Q
none	0	0	0
WEP 64	1.52588E-05	0.770252035	1.98E-05
WEP 128	1.000015259	0.789624674	1.266444
CKIP+MMH	5.000015259	0.788877697	6.338137
WPA-LEAP	8.125	1.17252514	6.929489

Figure 6.11: Results applied to the trade-off model (text-only workloads)

Chapter 7

Conclusions

7.1 Summary

This work has shown that intelligent reasoning about trade-offs between energy and security is an important research area. We reviewed the current limitations of security protocols utilized in wireless networks. A preliminary analysis of energy-security trade-offs was conducted, examining WEP, WPA and CCMP. The results yielded by this analysis suggested that there is an increase in energy consumption when increasing the security of a particular protocol (i.e. increasing the key size), but does not necessarily increase when changing to a more secure protocol. Because this analysis was done simply by algorithm review, we also obtained empirical measurements via experimentation. We measured energy consumed to retrieve web pages over a wireless network with each of the security profiles described.

From this experiment, we found that the cost of the different encryption algorithms did not vary significantly for our workload. The biggest factor in the encryption measurements was the number of transmissions that took place, which was dependent on the workload. The cost of authentication, however, did have a significant impact on the battery life. The highest cost authentication mechanisms are the EAP methods introduced in the later standards and are considered to provide a higher level of security. Applying limitations to energy and security parameters to our results shows how the model can be applied to the selection of a security profile based on energy restrictions.

Most of the current work of enhancing the robustness of wireless security protocols has adopted techniques from wired-world counterparts. Clearly, such reasoning has the desired effect of increased security; however, as our work has shown, such an approach has detrimental effects on the utility of the wireless device. Namely, it accelerates the depletion of battery life. Our work suggests that such consideration should be of importance moving forward in this area.

7.1.1 Future Work

The work presented in this thesis demonstrates how the trade-off model allows for optimal security profile selection, with respect to energy. While we have obtained empirical measurements for energy utilization, this is not the case with the security achieved by each profile. Indeed, the method that was used is not entirely accurate, as there is currently no known way of empirically measuring the security of a protocol. In order to obtain a better metric, it is necessary that we can more accurately calculate the security provided by a given profile. The problem of measuring the security of a protocol is very difficult one that has been heavily researched over the years. While we may not be able to solve it for our purposes, we must, at minimum, derive a better security proxy that more accurately reflects the differences and subtleties between the security of different profiles.

Currently, we know that lost connections and reassociation affects the number of messages sent and therefore has an impact on the overall energy consumption. However, we do not know how many associations and reassociations are normal within a single session. Further experimentation is required that measures the number of reassociations that take place on an access point, both over a campus and corporate network. This information will be used to modify our model and analytic equations to better incorporate the number of messages and energy spent for authentication and authorization during a typical session.

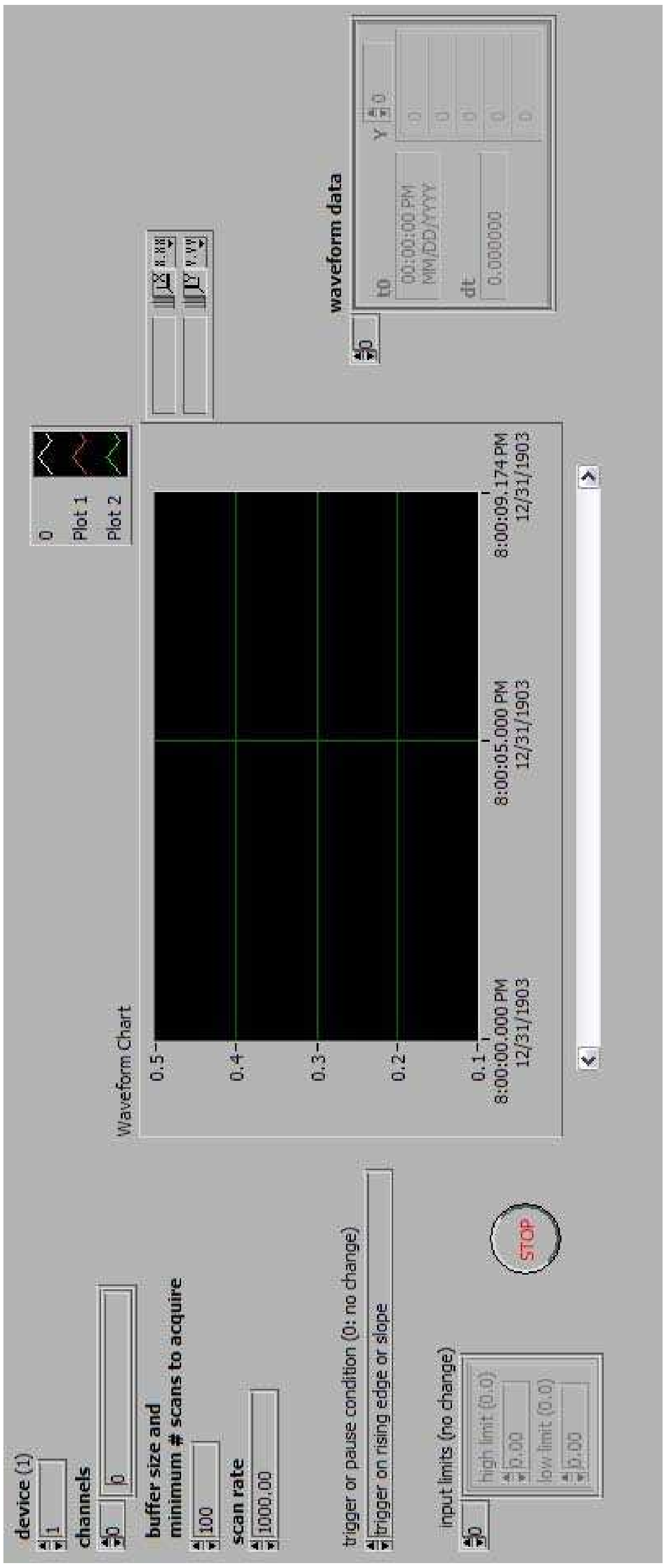
Appendix A

LabVIEW Code

This code is heavily based on an example by National Instruments [23]. The main changes are

- The analog input reads data in and outputs it as a waveform instead of scaled data
- The VI prints waveform data out to a file
- Default values for buffer size and scan rate were increased

Figure A.1 shows the GUI, also referred to as the front panel, of our virtual instrument. Figure A.2 shows the logic that the instrument follows.


device (1)
1

channels
0

buffer size and minimum # scans to acquire
100

scan rate
1000.00

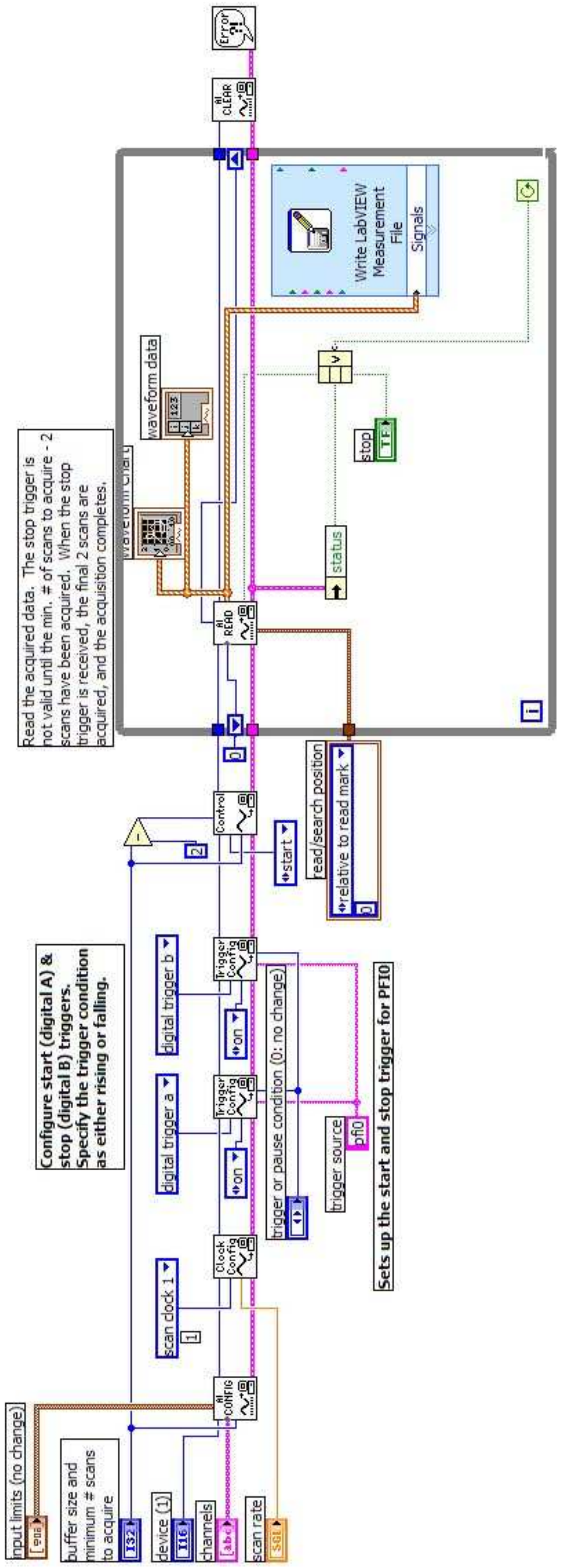
trigger or pause condition (0: no change)
trigger on rising edge or slope

input limits (no change)
high limit (0.0) 0.00
low limit (0.0) 0.00

waveform data
t0 00:00:00 PM MM/DD/YYYY
dt 0.000000
Y 0

STOP
Waveform Chart
0.5
0.4
0.3
0.2
0.1
8:00:00.000 PM 12/31/1903
8:00:05.000 PM 12/31/1903
8:00:09.174 PM 12/31/1903
0
Plot 1
Plot 2

Figure A.1: LabVIEW code: front panel


Read the acquired data. The stop trigger is not valid until the min. # of scans to acquire - 2 scans have been acquired. When the stop trigger is received, the final 2 scans are acquired, and the acquisition completes.

Configure start (digital A) & stop (digital B) triggers. Specify the trigger condition as either rising or falling.

Sets up the start and stop trigger for PFI0

Figure A.2: LabVIEW code: block diagram

Appendix B

Perl Code

```perl
#!/usr/bin/perl -w

################################################################################
# power.pl
# Author: Kerry McKay
#
# This script calculates the instantaneous power from a waveform captured by
# the virtual instrument. For each point, it calculates the integral using rectangle
# approximation. This script outputs important information at the bottom of the
# output file. This line is used later by average.pl.
#
# Voltage files should all be placed in the same directory. This script takes
# that directory as an argument and calculates the inst. power for every point
# in every file that is present in $dir.
################################################################################

my $dir = "";
my $rate = .001;

if (@ARGV != 1) {
  print "\nUsage:\n";
  print "\tshortpower.pl <directory name>\n\n";
  exit(0);
} else {
  $dir = $ARGV[0];
}

print "running on directory $dir\n";

# open directory $dir
opendir(DIR,$dir) or die("failed to open directory $dir: $!\n");

# create a directory for the results
mkdir("$dir-pwr", 0777) || die "cannot mkdir $dir/pwr: $!";

while (defined($file = readdir(DIR))) {
  if ($file =~ /^\./) {
    # don't do anything
  } else {
    # open each file in $dir for reading
    print "Reading file: $file\n";
    $path = "$dir/$file";
    open(FILE, "< $path") or die "Couldn't open $path for reading: $!\n";

    # create and open file for output
    ($base, $ext) = split(/\./, $file);
    $outfile = "$dir-pwr/$base-pwr.$ext";
    open(OUTFILE, "> $outfile") or die "Couldn't open $outfile for writing: $!\n";

    # store the entire file in an array
    @lines = <FILE>;

    #grab the timestamp of the first reading
    $i = 0; # counter
    $first = 0;
    do {
      if ($lines[$i] =~ m/((\d)+\.(\d)+)(\t)+((\d)+\.(\d)+)/) {
        $first = $1;
      }
      $i = $i+1;
    } until ($first =~ m/((\d)+\.(\d)+)/);

    print OUTFILE "starting at time $first:\n\n";

    print OUTFILE "Calculating for file: $file\n";
    print OUTFILE "\ntimestamp\tvoltage reading\t\tcurrent\t\tcurrent area\n";

    # initialize
    $tvolts = 0; #total volts
    $ttime = 0; #total time
    $current = 0; #instant current
    $tcurrent = 0; #total current
    $acurrent = 0; #area under the curve (estimated)



    # prevtime holds the value of the last recorded time. This is needed for the summation
    $prevtime = $first;

    foreach $line (@lines) {
      if ($line =~ /[a-zA-Z]+/) {
        #line of text. skip
      } elsif ($line =~ /(\t)+((\d)+\.(\d)+)/) {
        # grab timestamp and voltage reading
        ($time,  $volt) = split(/\t+/, $line);

        if ($time < $prevtime) {
          # Ignore this. Sometimes Labview repeats one of the early lines at the end.
          # Using this line will alter the results.
        } else {
          # strip the newline off of power reading
          $volt =~ s/\n//;

          #  C = V/R.
          $current = $volt/(0.47);

          # calculate running totals
          $tcurrent += $current;
          $acurrent += $current/1000; #each value represents 1 millisecond
          $tvolts = $tvolts + $volt;

          print OUTFILE "\n$time\t$volt\t\t$current\t\t$acurrent";

          # update prevtime to current timestamp
          $prevtime = $time;
        }
      }
    }

    # get total time
    $ttime = $prevtime-$first; # use $prevtime, $time may be from a repeated line

    $energy = $acurrent * 5; #5 volts is being supplied by AC adapter

    # print totals for each column
    print OUTFILE "\nTOTALS:\n";
    print OUTFILE "\ntime\tvoltage reading\t\tcurrent\t\tcurrent area\t\tjoules\n";
    print OUTFILE "\n$ttime\t$tvolts\t\t$current\t\t$acurrent\t\t$energy";

    # close open filehandles
    close(OUTFILE);
    close(FILE);
  }
}

closedir(DIR);
```

```
#!/usr/bin/perl -w

########################################################################
# average.pl
# Author: Kerry McKay
# v2
#
# This script is to be run after power.pl. It dives into each result
# directory and averages the joules consumed
########################################################################

my $dir = "";

if (@ARGV != 1) {
  print "\nUsage:\n";
  print "\tanalysis.pl <directory name>\n\n";
  exit(0);
} else {
  $dir = $ARGV[0];
}

print "running on directory $dir\n";
$jtotal = 0;
$javerage = 0;

# open directory $dir
opendir(DIR,$dir) or die("failed to open directory $dir: $!\n");

# create and open file for output
$outfile = "$dir-averaged.txt";
open(OUTFILE, "> $outfile") or die "Couldn't open $outfile for writing: $!\n";
print OUTFILE "\ntime\tvolt\t\tcurrent\t\tacurrent\t\tmicrojoules";

$size= 0; # holds the number of result files

# stores all the joule values to be used later for stats
$joulelist = "";

while (defined($file = readdir(DIR))) {
  if ($file =~ /^\./) {
    # don't do anything
  } else {

    # open each file in $dir for reading
    print "Reading file: $file\n";
    $path = "$dir/$file";
    open(FILE, "< $path") or die "Couldn't open $path for reading: $!\n";

    # store the entire file in an array, in reverse order so that the
    # important line is on the top

    $last = ""; #last result
    while ($line = <FILE>) {
      # we' re just waiting for the last line here
      $last = $line;
    }

    print "$last\n";

    ($time,  $volt, $current, $acurrent, $joules) = split(/\t+/, $last);

    # strip the \n off of joules
    $joules =~ s/\n//;

    #print OUTFILE "time: $time\t\tcurrent: $current\t\tjoules: $joules\n";
    print OUTFILE "\n$time\t$volt\t\t$current\t\t$acurrent\t\t$joules";
    $jtotal = $jtotal + $joules;
    $size++;

    #store the joule info
    $joulelist .= "$joules,";

    close(FILE);
  }

}

$javerage = $jtotal / $size;

print OUTFILE "\n\nraw results:\n";
print OUTFILE "TOTAL MICROJOULES:\t$jtotal\n";
print OUTFILE "AVERAGE MICROJOULES:\t$javerage\n";

# store the data lists into arrays
@joulearray = split(/,/, $joulelist);

#sort the data
@joulearray = sort {$a <=> $b} @joulearray;

$numruns = @joulearray; # the number of measurements that took place
print "runs: $numruns";

# Get the IQR
$q1 = int($numruns*.25);
$q3 = int($numruns*.75);
#store all values in the IQR in an array
@jouleiqr = @joulearray[$q1..$q3-1];

print "\n\nMICROJOULE IQR:\n";
foreach $line (@jouleiqr){
  print "$line\n";
}

# and now, for the IQR averages
```

```
$javerage = 0;

foreach $joule (@jouleiqr){
  $javerage += $joule;
}
$javerage = $javerage/($q3-$q1);

print OUTFILE "\n\nIQR results:\n";
print OUTFILE "AVERAGE MICROJOULES:\t$javerage\n";

# print the sorted data data list
print OUTFILE "\n\nJoules:\n";
foreach $j (@joulearray) {
  print OUTFILE "$j\n";
}

close(OUTFILE);
closedir(DIR);
```

Appendix C

Embedded Visual C++ Code

For the sake of brevity, only the main source file is included.

```
// httpDlg.cpp : implementation file
//
// This is the main file of our browser. The code is heavily based
// on the "http" sample that comes with Embedded Visual Tools 3.0

#include "stdafx.h"
#include "Afxinet.h"
#include "http.h"
#include "httpDlg.h"
#include "Winbase.h"


/* disaplying the objects as they download adds additional time
* and uses additional energy. Therefore, they are not shown in the
* experiment. However, they may be enabled for debugging purposes
* by setting DEBUG to true. */
#define DEBUG false

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

enum {
  kErrorMin  =0x10000000,
  kError1    =0x10000001,
  kError2    =0x10000010
};

/////////////////////////////////////////////////////////////////////////////
// CHttpSession object

// Http wants to use its own derivative of the CHttpSession class
// just so it can implement an OnStatusCallback() override.
// This code is direct from the sample
CMyHttpSession::CMyHttpSession(LPCTSTR pszAppName, DWORD dwContext, int nMethod)
  : CInternetSession(pszAppName, dwContext, nMethod)
{
}

// This code is direct from the sample
void CMyHttpSession::OnStatusCallback(DWORD dwContext, DWORD dwInternetStatus,
  LPVOID /* lpvStatusInfomration */, DWORD /* dwStatusInformationLen */)
{
  CHttpDlg* pDlg = (CHttpDlg*)dwContext;
  if(!pDlg)
    return;

  switch(dwInternetStatus)
  {
    case INTERNET_STATUS_RESOLVING_NAME:
      if (DEBUG) {
        pDlg->SetDlgItemText(IDC_MESSAGE, CString((LPCTSTR)IDS_IPLOOKING));
      }
      break;
    case INTERNET_STATUS_NAME_RESOLVED:
      if (DEBUG) {
        pDlg->SetDlgItemText(IDC_MESSAGE, CString((LPCTSTR)IDS_IPFOUND));
      }
      break;
    case INTERNET_STATUS_CONNECTING_TO_SERVER:
      if (DEBUG) {
        pDlg->SetDlgItemText(IDC_MESSAGE, CString((LPCTSTR)IDS_CONNECTING));
      }
      break;
    case INTERNET_STATUS_CONNECTED_TO_SERVER:
      if (DEBUG) {
        pDlg->SetDlgItemText(IDC_MESSAGE, CString((LPCTSTR)IDS_CONNECTED));
      }
      break;
    case INTERNET_STATUS_SENDING_REQUEST:
      if (DEBUG) {
        pDlg->SetDlgItemText(IDC_MESSAGE, CString((LPCTSTR)IDS_REQUEST));
      }
      break;
    case INTERNET_STATUS_REQUEST_SENT:
      if (DEBUG) {
        pDlg->SetDlgItemText(IDC_MESSAGE, CString((LPCTSTR)IDS_SENT));
      }
      break;
    case INTERNET_STATUS_RECEIVING_RESPONSE:
      if (DEBUG) {
        pDlg->SetDlgItemText(IDC_MESSAGE, CString((LPCTSTR)IDS_WAITING));
      }
      break;
    case INTERNET_STATUS_RESPONSE_RECEIVED:
      if (DEBUG) {
        pDlg->SetDlgItemText(IDC_MESSAGE, CString((LPCTSTR)IDS_RECEIVED));
      }
      break;
    case INTERNET_STATUS_CLOSING_CONNECTION:
      if (DEBUG) {
        pDlg->SetDlgItemText(IDC_MESSAGE, CString((LPCTSTR)IDS_CLOSING));
      }
      break;
    case INTERNET_STATUS_CONNECTION_CLOSED:
      if (DEBUG) {
        pDlg->SetDlgItemText(IDC_MESSAGE, CString((LPCTSTR)IDS_CLOSED));
      }
      break;
    case INTERNET_STATUS_HANDLE_CREATED:
      if (DEBUG) {
        pDlg->SetDlgItemText(IDC_MESSAGE, CString((LPCTSTR)IDS_CONNECTED2));
      }
      break;
    case INTERNET_STATUS_HANDLE_CLOSING:
      if (DEBUG) {
        pDlg->SetDlgItemText(IDC_MESSAGE, CString((LPCTSTR)IDS_DISCONNECTED));
      }
      break;
    case INTERNET_STATUS_REQUEST_COMPLETE:
      if (DEBUG) {
        pDlg->SetDlgItemText(IDC_MESSAGE, CString((LPCTSTR)IDS_COMPLETED));
      }
      break;
    case INTERNET_STATUS_REDIRECT:
      if (DEBUG) {
        pDlg->SetDlgItemText(IDC_MESSAGE, CString((LPCTSTR)IDS_REDIRECTED));
      }
      break;
  }
}

// This code is direct from the sample
void ThrowHttpException(int nCode)
{
  CInternetException* pEx = new CInternetException(nCode);
  THROW(pEx);
}


/////////////////////////////////////////////////////////////////////////////
// CHttpDlg dialog

CHttpDlg::CHttpDlg(CWnd* pParent /*=NULL*/)
  : CDialog(CHttpDlg::IDD, pParent)
{
  //{{AFX_DATA_INIT(CHttpDlg)
```

```
    // NOTE: the ClassWizard will add member initialization here
  //}}AFX_DATA_INIT
  // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
  m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CHttpDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CHttpDlg)
    // NOTE: the ClassWizard will add DDX and DDV calls here
  //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CHttpDlg, CDialog)
  //{{AFX_MSG_MAP(CHttpDlg)
  ON_BN_CLICKED(IDC_DOWNLOADPAGE, OnDownloadpage)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CHttpDlg message handlers
//
// Code from this point on is new or modified
BOOL CHttpDlg::OnInitDialog()
{
  CDialog::OnInitDialog();

  // Set the icon for this dialog.  The framework does this automatically
  //  when the application's main window is not a dialog
  SetIcon(m_hIcon, TRUE);      // Set big icon
  SetIcon(m_hIcon, FALSE);    // Set small icon

  CenterWindow(GetDesktopWindow());  // center to the hpc screen

  // populate the drop down list with the experiment pages
  CComboBox *pPage = (CComboBox*)GetDlgItem(IDC_ADDRESS);
  ASSERT(pPage != NULL);

  pPage->AddString(CString("http://192.168.254.146/2_img.html"));
  pPage->AddString(CString("http://192.168.254.146/5_img.html"));
  pPage->AddString(CString("http://192.168.254.146/9_img.html"));
  pPage->AddString(CString("http://192.168.254.146/10_img.html"));
  pPage->AddString(CString("http://192.168.254.146/20_img.html"));
  pPage->AddString(CString("http://192.168.254.146/30_img.html"));
  pPage->AddString(CString("http://192.168.254.146/40_img.html"));
  pPage->AddString(CString("http://192.168.254.146/text_2.html"));
  pPage->AddString(CString("http://192.168.254.146/text_5.html"));
  pPage->AddString(CString("http://192.168.254.146/text_9.html"));
  pPage->AddString(CString("http://192.168.254.146/text_10.html"));
  pPage->AddString(CString("http://192.168.254.146/text_20.html"));
  pPage->AddString(CString("http://192.168.254.146/text_30.html"));
  pPage->AddString(CString("http://192.168.254.146/text_40.html"));
  pPage->AddString(CString("http://192.168.254.146/disconnect.html"));

  return TRUE;
}


void CHttpDlg::OnDownloadpage()
{
  int myReturn = 0;
  static HANDLE hPort = INVALID_HANDLE_VALUE;

  TCHAR szAddress[INTERNET_MAX_HOST_NAME_LENGTH];
  GetDlgItemText(IDC_ADDRESS, szAddress, INTERNET_MAX_HOST_NAME_LENGTH - 1);

  SetDlgItemText(IDC_MESSAGE, _T("Download in progess"));
  SetDlgItemText(IDC_EDIT_PAGE, _T(""));

  // open the serial connection
  openPort(hPort);
  if (hPort == INVALID_HANDLE_VALUE ) {
    MessageBox (TEXT("openPort failed"),
            TEXT("Error"), MB_OK);
  }

  portConf(hPort);


  TCHAR szAddress2[INTERNET_MAX_HOST_NAME_LENGTH];

  char baseurl[256]="http://192.168.254.146/";
  char t[4];        //for converting from int to char

  if (CString(szAddress) == "http://192.168.254.146/2_img.html") {
    for (int run=1; run<=100; run++) {

      //send start signal
      portWrite(hPort);

      if (DownloadPage(szAddress) !=0) {
        SetDlgItemText(IDC_EDIT_PAGE, CString("Download failed"));
        CloseHandle(hPort);
        exit(0);
      }

      for (int i=1; i<=48; i++){
        sprintf(t, "%i", i);  //so that CString likes us
        CString url = CString("http://192.168.254.146/2k-") + t + ".jpg";
        lstrcpy(szAddress2, url);
        if (DownloadPage(szAddress2) != 0) {
          SetDlgItemText(IDC_EDIT_PAGE, CString("Download failed"));
          CloseHandle(hPort);
```

```
            exit(0);
          }
        }

        //send stop signal
        portWrite(hPort);

        Sleep(500);  //a small break so that the serial signals aren't too close
      }
      SetDlgItemText(IDC_MESSAGE, CString("Download Complete"));

    } else if (CString(szAddress) == "http://192.168.254.146/5_img.html"){
      for (int run=1; run<=100; run++) {

        //send start signal
        portWrite(hPort);

        if (DownloadPage(szAddress) !=0) {
          SetDlgItemText(IDC_EDIT_PAGE, CString("Download failed"));
          CloseHandle(hPort);
          exit(0);
        }

        for (int i=1; i<=22; i++){
          sprintf(t, "%i", i);  //so that CString likes us
          CString url = CString("http://192.168.254.146/5k-") + t + ".jpg";
          lstrcpy(szAddress2, url);
          if (DownloadPage(szAddress2) != 0) {
            SetDlgItemText(IDC_EDIT_PAGE, CString("Download failed"));
            CloseHandle(hPort);
            exit(0);
          }
        }

        //send stop signal
        portWrite(hPort);

        Sleep(500);  //a small break so that the serial signals aren't too close
      }
      SetDlgItemText(IDC_MESSAGE, CString("Download Complete"));

    } else if (CString(szAddress) == "http://192.168.254.146/9_img.html"){
      for (int run=1; run<=100; run++) {

        //send start signal
        portWrite(hPort);

        if (DownloadPage(szAddress) !=0) {
          SetDlgItemText(IDC_EDIT_PAGE, CString("Download failed"));
          CloseHandle(hPort);
          exit(0);
        }
        for (int i=1; i<=22; i++){
          sprintf(t, "%i", i);  //so that CString likes us
          CString url = CString("http://192.168.254.146/9k-") + t + ".jpg";
          lstrcpy(szAddress2, url);
          if (DownloadPage(szAddress2) != 0) {
            SetDlgItemText(IDC_EDIT_PAGE, CString("Download failed"));
            CloseHandle(hPort);
            exit(0);
          }
        }

        //send stop signal
        portWrite(hPort);

        Sleep(500);  //a small break so that the serial signals aren't too close
      }
      SetDlgItemText(IDC_MESSAGE, CString("Download Complete"));

    } else if (CString(szAddress) == "http://192.168.254.146/10_img.html"){
      for (int run=1; run<=100; run++) {

        //send start signal
        portWrite(hPort);

        if (DownloadPage(szAddress) !=0) {
          SetDlgItemText(IDC_EDIT_PAGE, CString("Download failed"));
          CloseHandle(hPort);
          exit(0);
        }

        for (int i=1; i<=2; i++){
          sprintf(t, "%i", i);  //so that CString likes us
          CString url = CString("http://192.168.254.146/10k-") + t + ".jpg";
          lstrcpy(szAddress2, url);
          if (DownloadPage(szAddress2) != 0) {
            SetDlgItemText(IDC_EDIT_PAGE, CString("Download failed"));
            CloseHandle(hPort);
            exit(0);
          }
        }

        //send stop signal
        portWrite(hPort);

        Sleep(500);  //a small break so that the serial signals aren't too close
      }
      SetDlgItemText(IDC_MESSAGE, CString("Download Complete"));

    } else if (CString(szAddress) == "http://192.168.254.146/20_img.html"){
      for (int run=1; run<=100; run++) {
        //send start signal
        portWrite(hPort);

        if (DownloadPage(szAddress) !=0) {
          SetDlgItemText(IDC_EDIT_PAGE, CString("Download failed"));
          CloseHandle(hPort);
          exit(0);
        }

        for (int i=1; i<=2; i++){
          sprintf(t, "%i", i);  //so that CString likes us
          CString url = CString("http://192.168.254.146/20k-") + t + ".jpg";
          lstrcpy(szAddress2, url);
          if (DownloadPage(szAddress2) != 0) {
            SetDlgItemText(IDC_EDIT_PAGE, CString("Download failed"));
            CloseHandle(hPort);
            exit(0);
          }
        }

        //send stop signal
        portWrite(hPort);

        Sleep(500);  //a small break so that the serial signals aren't too close
      }
      SetDlgItemText(IDC_MESSAGE, CString("Download Complete"));

    } else if (CString(szAddress) == "http://192.168.254.146/30_img.html"){
      /* The start and stop signals are too close in this case. Run each 10 times
      * and divide it by 10 later */
      for (int run=1; run<=100; run++) {

        //send start signal
        portWrite(hPort);

        for (int j=1; j<=9; j++) {
          if (DownloadPage(szAddress) !=0) {
            SetDlgItemText(IDC_EDIT_PAGE, CString("Download failed"));
            CloseHandle(hPort);
            exit(0);
          } else {
            for (int i=1; i<=1; i++){
              sprintf(t, "%i", i);  //so that CString likes us
              CString url = CString("http://192.168.254.146/30k-") + t + ".jpg";
              lstrcpy(szAddress2, url);
              if (DownloadPage(szAddress2) != 0) {
                SetDlgItemText(IDC_EDIT_PAGE, CString("Download failed"));
                CloseHandle(hPort);
                exit(0);
              }
            }
          }
        }
        //send stop signal
        portWrite(hPort);

        Sleep(500);  //a small break so that the serial signals aren't too close
      }
      SetDlgItemText(IDC_MESSAGE, CString("Download Complete-divide by 10"));

    } else if (CString(szAddress) == "http://192.168.254.146/40_img.html"){

      /* The start and stop signals are too close in this case. Run each 10 times
      * and divide it by 10 later */
      for (int run=1; run<=100; run++) {

        //send start signal
        portWrite(hPort);

        for (int j=1; j<=10; j++) {
          if (DownloadPage(szAddress) !=0) {
            SetDlgItemText(IDC_EDIT_PAGE, CString("Download failed"));
            CloseHandle(hPort);
            exit(0);
          } else {
            for (int i=1; i<=1; i++){
              sprintf(t, "%i", i);  //so that CString likes us
              CString url = CString("http://192.168.254.146/40k-") + t + ".jpg";
              lstrcpy(szAddress2, url);
              if (DownloadPage(szAddress2) != 0) {
                SetDlgItemText(IDC_EDIT_PAGE, CString("Download failed"));
                CloseHandle(hPort);
                exit(0);
              }
            }
          }
        }
        //send stop signal
        portWrite(hPort);

        Sleep(500);  //a small break so that the serial signals aren't too close
      }
      SetDlgItemText(IDC_MESSAGE, CString("Download Complete-divide by 10"));

    } else if (CString(szAddress) == "http://192.168.254.146/disconnect.html"){
      /* there actually is no file "disconnect.html". This case requests text2.html
      * 200 times. This should allow us enough time to insert several disconnections.
      */
      for (int run=1; run<=100; run++) {

        //send start signal
        portWrite(hPort);

        for (int j=1; j<=200; j++) {
          CString url = CString("http://192.168.254.146/text_2.html");
          lstrcpy(szAddress2, url);
          if (DownloadPage(szAddress2) != 0) {
            SetDlgItemText(IDC_EDIT_PAGE, CString("Download failed"));
            CloseHandle(hPort);
            exit(0);
          }
        }
        //send stop signal
        portWrite(hPort);

        Sleep(500);  //a small break so that the serial signals aren't too close
      }
      SetDlgItemText(IDC_MESSAGE, CString("Download Complete"));

    } else {
      /* The url is a text only page. Because the start and stop signals are too
      * close for Labview to recognize, we need to request the page multiple times,
      * and then divide the result by that many times afterwards. This last part
      * must be done manually after the perl script that calculates average energy
      * is run. */

      for (int i=1; i<=1; i++) {

        //send start signal
        portWrite(hPort);

        for (int j=1; j<=10; j++) {
          if (DownloadPage(szAddress) !=0) {
            SetDlgItemText(IDC_EDIT_PAGE, CString("Download failed"));
            CloseHandle(hPort);
            exit(0);
          }
        }

        //send stop signal
        portWrite(hPort);

        Sleep(500);  //a small break so that the serial signals aren't too close
      }
      SetDlgItemText(IDC_MESSAGE, CString("Download Complete-divide by 10"));
    }

    //close handle to serial port
    CloseHandle (hPort);

  }


  /************************************************
  * DownloadPage
  *
```

```
* This function was taken from the http sample program
* that comes with MS Embedded Visual Tools 3.0. It
* has very minor modifications to the original.
*************************************************/
int CHttpDlg::DownloadPage(LPTSTR szAddress)
{

  int   nRetCode = 0;
  DWORD dwAccessType = PRE_CONFIG_INTERNET_ACCESS;
  const TCHAR szHeaders[] = _T("Accept: text/*\r\nUser-Agent: WSSRL_get_tool\r\n");
  BOOL  bSuccess = TRUE;

  // don't allow caching. If we do, then we'll get 304 responses and our test will be bad.
  DWORD dwHttpRequestFlags = INTERNET_FLAG_DONT_CACHE;

  CMyHttpSession session(CString("WSSRL"), (DWORD)this, dwAccessType);
  CHttpConnection* pServer = NULL;
  CHttpFile* pFile = NULL;
  TRY
  {
    // check to see if this is a reasonable URL
    CString strServerName;
    CString strObject;
    INTERNET_PORT nPort;
    DWORD dwServiceType;

    if (!AfxParseURL(szAddress, dwServiceType, strServerName, strObject, nPort) ||
      dwServiceType != INTERNET_SERVICE_HTTP)
    {
      SetDlgItemText(IDC_EDIT_PAGE, CString((LPCTSTR)IDS_ERROR1));
      ThrowHttpException(kError1);
    }

    session.EnableStatusCallback(TRUE);

    pServer = session.GetHttpConnection(strServerName, nPort);

    pFile = pServer->OpenRequest(CHttpConnection::HTTP_VERB_GET, strObject, NULL,
     (DWORD)this, NULL, NULL, dwHttpRequestFlags);
    pFile->AddRequestHeaders(szHeaders);
    pFile->SendRequest();

    DWORD dwRet;
    pFile->QueryInfoStatusCode(dwRet);

    if (dwRet == HTTP_STATUS_DENIED)
    {
      MessageBox (L"Access to the secured http site is denied!", L"Error", MB_OK);
      return 0;
    }

    CString strNewLocation;
```

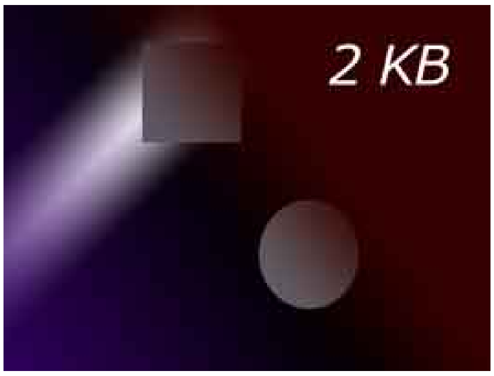
```
pFile->QueryInfo(HTTP_QUERY_RAW_HEADERS_CRLF, strNewLocation);

// were we redirected?
// these response status codes come from WININET.H

if (dwRet == HTTP_STATUS_MOVED ||
  dwRet == HTTP_STATUS_REDIRECT ||
  dwRet == HTTP_STATUS_REDIRECT_METHOD)
{
  CString strNewLocation;
  pFile->QueryInfo(HTTP_QUERY_RAW_HEADERS_CRLF, strNewLocation);

  int nPlace = strNewLocation.Find(_T("Location: "));
  if (nPlace == -1)
  {
    SetDlgItemText(IDC_EDIT_PAGE, CString((LPCTSTR)IDS_ERROR2) );
    ThrowHttpException(kError2);
  }

  strNewLocation = strNewLocation.Mid(nPlace + 10);
  nPlace = strNewLocation.Find('\n');
  if (nPlace > 0)
    strNewLocation = strNewLocation.Left(nPlace);

  // close up the redirected site
  pFile->Close();
  delete pFile;
  pServer->Close();
  delete pServer;

  CString csMsg = CString((LPCTSTR)IDS_CAUTION) + strNewLocation;
  SetDlgItemText(IDC_MESSAGE, csMsg);

  // figure out what the old place was
  if (!AfxParseURL(strNewLocation, dwServiceType, strServerName, strObject, nPort))
  {
    SetDlgItemText(IDC_EDIT_PAGE, CString((LPCTSTR)IDS_ERROR3));
    ThrowHttpException(kError2);
  }

  if (dwServiceType != INTERNET_SERVICE_HTTP)
  {
    SetDlgItemText(IDC_EDIT_PAGE, CString((LPCTSTR)IDS_ERROR4));
    ThrowHttpException(kError2);
  }

  // try again at the new location
  pServer = session.GetHttpConnection(strServerName, nPort);
  pFile = pServer->OpenRequest(CHttpConnection::HTTP_VERB_GET,
    strObject, NULL, (DWORD)this, NULL, NULL, dwHttpRequestFlags);
  pFile->AddRequestHeaders(szHeaders);
  pFile->SendRequest();
```

```
    pFile->QueryInfoStatusCode(dwRet);
    if (dwRet != HTTP_STATUS_OK)
      ThrowHttpException(kError2);
  }

  TCHAR* szWEBPage = new TCHAR[MAX_WEBPAGE_SIZE+1];
  if(szWEBPage)
  {
    szWEBPage[0] = L'\0';

    TCHAR* sz     = new TCHAR[BUFFER_SIZE+1];
    TCHAR* szwBuf = new TCHAR[(BUFFER_SIZE+1)*2];

    sz[0] = L'\0';
    szwBuf[0] = L'\0';
    int n = 0;
    pFile->SetReadBufferSize(BUFFER_SIZE*2);
    while (pFile->ReadString(sz, BUFFER_SIZE))
    {
      wce_AsciiToWide(szwBuf, (const char*)sz);

      n += _tcslen(szwBuf);
      if(n >= MAX_WEBPAGE_SIZE)
        break;
      _tcscat(szWEBPage, szwBuf);
    }
    delete [] sz;
    delete [] szwBuf;

    /* this print will add lots of time to the test run.
    * Only enable it for debugging purposes */
    if (DEBUG) {
      SetDlgItemText(IDC_EDIT_PAGE, szWEBPage);
    }
  }



  delete [] szWEBPage;
  pFile->Close();
  pServer->Close();
}
CATCH (CInternetException,  pEx)
{
  // catch things wrong with parameters, etc
  if (pEx->m_dwError < kErrorMin)
  {
    TCHAR szError[MAX_PATH]=TEXT("\0");
    pEx->GetErrorMessage(szError,MAX_PATH,NULL);
    SetDlgItemText(IDC_EDIT_PAGE, szError);
  }
```

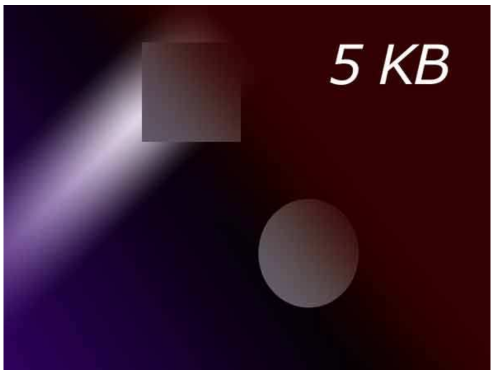
```
    bSuccess = FALSE;
  }
  AND_CATCH (CMemoryException,  pMemory)
  {
    // catch things wrong with memory
    SetDlgItemText(IDC_EDIT_PAGE,CString((LPCTSTR)IDS_MEMORYEXCEPTION));
    pMemory->Delete();
    bSuccess = FALSE;
  }
  END_CATCH_ALL

  if (pFile != NULL)
    delete pFile;
  if (pServer != NULL)
    delete pServer;
  session.Close();

  if(bSuccess && DEBUG) {
    SetDlgItemText(IDC_MESSAGE, CString((LPCTSTR)IDS_DOWNLOADED));
  }

  return nRetCode;
}




/***********************************************
* Serial Communication Code
*
* This is the code that we need in order to
* establish serial communication. This lets us
* acquire precisely the time period that we're
* interested in.
***********************************************/
void CHttpDlg::openPort (HANDLE &hPort) {
  // Open the serial port.

  LPCTSTR lpszPortName = TEXT("COM1:");
  hPort = CreateFile (lpszPortName, // Pointer to the name of the port
                      GENERIC_READ | GENERIC_WRITE,
                                    // Access (read-write) mode
                      0,            // Share mode
                      NULL,         // Pointer to the security attribute
                      CREATE_ALWAYS,// How to open the serial port
                      0,            // Port attributes
                      NULL);        // Handle to port with attribute
                                    // to copy
}

void CHttpDlg::portConf (HANDLE &hPort) {
  DCB PortDCB;
  DWORD dwError;

  // Initialize the DCBlength member.
  PortDCB.DCBlength = sizeof (DCB);

  // Get the default port setting information.
  GetCommState (hPort, &PortDCB);

  // Change the DCB structure settings.
  PortDCB.BaudRate = 9600;              // Current baud
  PortDCB.fBinary = TRUE;               // Binary mode; no EOF check
  PortDCB.fParity = TRUE;               // Enable parity checking
  PortDCB.fOutxCtsFlow = FALSE;         // No CTS output flow control
  PortDCB.fOutxDsrFlow = FALSE;         // No DSR output flow control
  PortDCB.fDtrControl = FALSE;
                                      // DTR flow control type
  PortDCB.fDsrSensitivity = FALSE;      // DSR sensitivity
  PortDCB.fTXContinueOnXoff = FALSE;     // XOFF continues Tx
  PortDCB.fOutX = FALSE;                // No XON/XOFF out flow control
  PortDCB.fInX = FALSE;                 // No XON/XOFF in flow control
  PortDCB.fErrorChar = FALSE;           // Disable error replacement
  PortDCB.fNull = FALSE;                // Disable null stripping
  PortDCB.fRtsControl = FALSE;
                                      // RTS flow control
  PortDCB.fAbortOnError = FALSE;        // Do not abort reads/writes on
                                      // error
  PortDCB.ByteSize = 8;                 // Number of bits/byte, 4-8
  PortDCB.Parity = 0;            // 0-4=no,odd,even,mark,space
  PortDCB.StopBits = 0;        // 0,1,2 = 1, 1.5, 2

  // Configure the port according to the specifications of the DCB
  // structure.
  if (!SetCommState (hPort, &PortDCB))
  {
    // Could not create the read thread.
    MessageBox (TEXT("Unable to configure the serial port"),
              TEXT("Error"), MB_OK);
    dwError = GetLastError ();
  }
}

void CHttpDlg::portWrite (HANDLE &hPort) {
  char bytes[] = "  ";
  DWORD dwNumBytesWritten = 0;
  CString prev;

  if (!WriteFile (hPort, &bytes[0], 1, &dwNumBytesWritten, NULL)) {
    // write failed
    SetDlgItemText(IDC_EDIT_PAGE, CString("Failed to send serial signal\n"));
  }

}
```

Appendix D

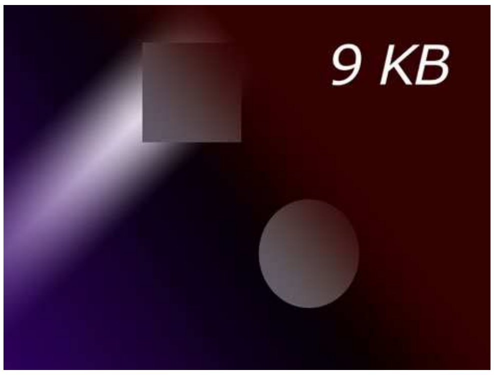
Workload



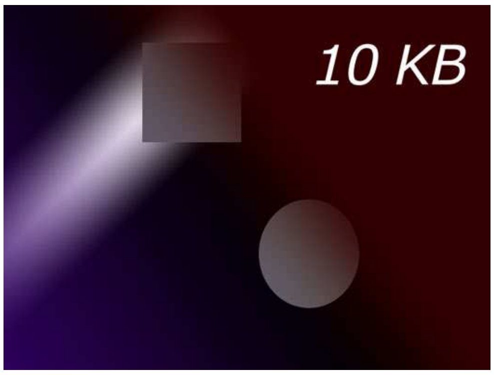
(a) 2 KB JPG image



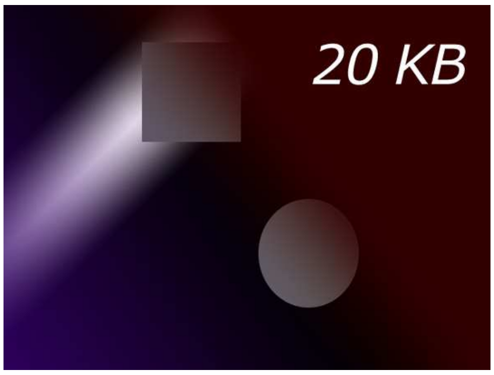
(b) 5 KB JPG image



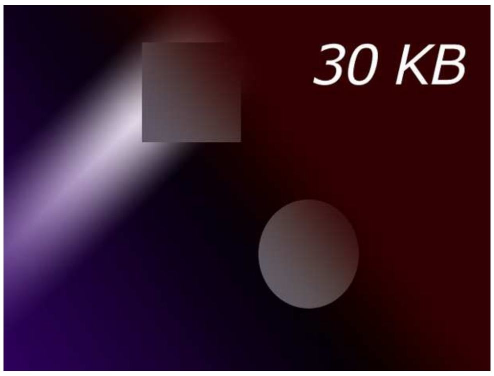
(c) 9 KB JPG image



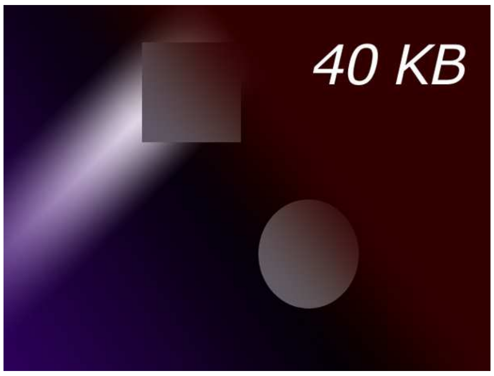
(d) 10 KB JPG image



(e) 20 KB JPG image



(f) 30 KB JPG image



(g) 40 KB JPG image

Appendix E

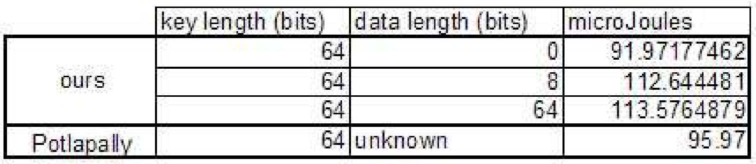
Calibration

In order to validate our experiment, we performed a calibration test using the results of related paper from Princeton University by Potlapally et al.[25]. This study was selected for calibration because it is the basis for the experimental structure described in chapter 5. Because of this, the experiments had much in common.

Although this paper performed its measurements on OpenSSL, they provided basic measurements for RC4; namely energy used during key setup and energy used to encrypt a byte of data. These are the numbers that we performed our calibration against. In order to capture the current used by only the computation without the interference of other factors, namely energy associated with the wireless card, a new application was created solely for this test. This application, like our browser, transmits signals over the serial connection to start and stop the measurements. Radio buttons allow the user to select RC4 key setup or RC4 encryption.

Potlapally et. al	Our experiment
Compaq iPAQ H3670 (SA-11000 Strongarm)	Compaq iPAQ H3955 (PAX250)
Familiar Linux	Windows Mobile 2003
SCB-68 connector block	CB-68LP connector block

Table E.1: Hardware and software differences between the Princeton experiment and our experiment



	key length (bits)	data length (bits)	microJoules
ours	64	0	91.97177462
	64	8	112.644481
	64	64	113.5764879
Potlapally	64	unknown	95.97

Table E.2: calibration results

Bibliography

[1] Adobe systems incorporated. http://www.adobe.com.

[2] Arbaugh, W., Shankar, N., and Wan, Y. J. Your 802.11 wireless network has no clothes. In *First IEEE International Conference on Wireless LANs and Home Networks* (Dec. 2001).

[3] Balachandran, A., Voelker, G. M., Bahl, P., and Rangan, P. V. Characterizing user behavior and network performance in a public wireless lan. In *Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems* (2002), ACM Press, pp. 195–205.

[4] Colón Osorio, F. C., Agu, E., and McKay, K. Measuring energy-security tradeoffs in wireless networks. In *Proceedings of the IEEE 24th International Performance Computing and Communications Conference* (Phoenix, AZ, April 2005).

[5] Daemen, J., and Rijmen, V. Aes proposal: Rijndael (ammended), Mar. 1999.

[6] Edney, J., and Arbaugh, W. A. *Real 802.11 Security: Wi-Fi Protected Access and 802.11i.* Addison Wesley, July 2003.

[7] Ethereal network protocol analyzer. http://www.ethereal.com.

[8] Fluhrer, S. R., Mantin, I., and Shamir, A. Weaknesses in the key scheduling algorithm of rc4. In *Revised Papers from the 8th Annual International Workshop on Selected Areas in Cryptography* (2001), Springer-Verlag, pp. 1–24.

[9] Funk software. http://funk.com.

[10] Gartner, Inc. Number wlan hot spots growing. Web article, June 2003. http://www4.gartner.com/5_about/press_releases/pr30june2003a.jsp.

[11] Henderson, T., Kotz, D., and Abyzov, I. The changing usage of a mature campus-wide wireless network. In *Proceedings of the 10th annual international conference on Mobile computing and networking* (2004), ACM Press, pp. 187–201.

[12] In-Stat/MDR. Embedded wi-fi market undergoing major shift. Web article, Aug 2004. http://www.instat.com/press.asp?ID=1059&sku=IN0401345WS.

[13] Johnston, D. Assorted 802.11 related crypto algorithms, 2002. https://www.deadhat.com/wlancrypto/.

[14] Karri, R., and Mishra, P. Optimizing the energy consumed by secure wireless sessions: wireless transport layer security case study. *Mob. Netw. Appl. 8*, 2 (2003), 177–185.

[15] Kaufman, C., Perlman, R., and Speciner, M. *Network Security: Private Communication in a Public World.* Prentice Hall, 2002, ch. 13 and 14.

[16] Kotz, D., and Essien, K. Analysis of a campus-wide wireless network. In *Proceedings of the 8th annual international conference on Mobile computing and networking* (2002), ACM Press, pp. 107–118.

[17] Lahiri, K., Raghunathan, A., Dey, S., and Panigrahi, D. Battery driven system design: a new frontier in low power design, 2002.

[18] Luo, J., and Jha, N. K. Battery-aware static scheduling for distributed real-time embedded systems. In *Design Automation Conference* (2001), pp. 444–449.

[19] Meetinghouse data communications. http://mtghouse.com.

[20] Microsoft corporation. http://www.microsoft.com.

[21] Mishra, A., and Arbaugh, W. A. An initial security analysis of the ieee 802.1x standard.

[22] Nanda, S. Wireless insecurity / how johnny can hack your wep protected 802.11b network! Tech. rep., Dartmouth College, 2002.

[23] National Instruments. Continuously acquiring analog signals using a digital start and stop trigger. ftp://ftp.ni.com/contrib/epd/B123AE0CBB01111EE034080020E74861/AI_Start-Stop_D-Trig.vi.

[24] National instruments corp. http://www.ni.com.

[25] Potlapally, N. R., Ravi, S., Raghunath, A., and Jha, N. K. Analyzing the energy consumption of security protocols. In *Proceedings of the 2003 international symposium on Low power electronics and design* (2003), ACM Press, pp. 30–35.

[26] Saroiu, S., Gummadi, K. P., Dunn, R. J., Gribble, S. D., and Levy, H. M. An analysis of internet content delivery systems. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation* (Dec 2002), USENIX Association.

[27] Simunic, T., Benini, L., and Micheli, G. D. Energy-efficient design of battery-powered embedded systems. In *Proc. Int. Symp. Low Power Electronics & Design* (August 1999), pp. 212–217.

[28] Stemm, M., and Katz, R. H. Measuring and reducing energy consumption of network interfaces in hand-held devices. In *IEICE Transactions on Communications* (Aug. 1997), vol. E80-B(8), pp. 1125–31.

[29] Tang, D., and Baker, M. Analysis of a local-area wireless network. In *Proceedings of the 6th annual international conference on Mobile computing and networking* (2000), ACM Press, pp. 1–10.