Fingerprinting 802.11 Implementations via Statistical Analysis of the Duration Field

Or. . . I know what device driver you have by staring at a 16-bit field long enough and squinting.

9/2006

Johnny Cache
johnycsh@802.11mercenary.net

Contents

Chapter 1

Foreword

Abstract: The research presented in this paper provides the reader with a set of algorithms and techniques that enable the user to remotely determine what chipset and device driver an 802.11 device is using. The technique outlined is entirely passive, and given the amount of features that are being considered for inclusion into the 802.11 standard, seems quite likely that it will increase in precision as the standard marches forward.

The implications of this are far ranging. On one hand, the techniques can be used to implement innovative new features in Wireless Intrusion Detection Systems (WIDS). On the other, they can be used to target link layer device driver attacks with much higher precision.

LIST OF FIGURES

LIST OF TABLES

Acknowledgements: I would like to thank Dr. Volpano for his technical as well as editorial expertise. Without his help, this work would be significantly more difficult on the reader. I would also like to thank Joshua Wright and Mike Kershaw for their technical input and contributions to those of us interested in 802.11 research in general. This material is based upon work supported by the National Science Foundation under Grant No. DUE0414102. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

This paper is a subset of the thesis research done while attending the Naval Postgraduate School. The entire thesis is available at www.802.11mercenary.net/~johnycsh/publications.

Chapter 2

Introduction

The adoption of wireless local area networks (WLANs) has exploded in recent years due in large part to standardization by the IEEE and certified WiFi interoperability; see the compilation *Wireless LAN Edition*[1]. Vendors ship products that they claim conform to the IEEE 802.11 standard and in many ways, they do, as the WiFi industry consortium can confirm. Yet products can also vary widely in their implementations of this standard. An implementation usually comprises a software component (the device driver), the hardware (radio chipset), and firmware for that chipset. The combination of the three uniquely identifies the implementation. Invariably, an implementation exhibits some behavior that can be observed or measured and is unique. This behavior is called its 802.11 *fingerprint*. Fingerprints enable us to identify 802.11 implementations.

2.1 Why Fingerprint 802.11?

Some 802.11 implementations have vulnerabilities that make devices that use the wireless technology vulnerable as well. Exploits developed for one implementation may not work for another so an attacker might prefer to identify the implementation first. Then they can choose the appropriate exploit rather than cycling through them and possibly drawing attention to themselves by crashing a device with the wrong exploit.

Fingerprints can also be used in a defensive way. A system administrator may maintain a database of authorized devices approved for use on their WLAN. Typically the devices are identified by their globally-unique 802.11 MAC addresses. But this is insufficient because a MAC address can be easily cloned by an authorized user using an unauthorized device. A better approach is to use an

802.11 fingerprint. Knowing which 802.11 implementations are vulnerable, an administrator can monitor their environment for wireless activity, observe 802.11 fingerprints, and be notified of an authorized user who is using a device with a vulnerable 802.11 implementation even if the device clones the 802.11 MAC address of an authorized, and presumably secure, implementation. There are a variety of monitoring products on the market today, generally called Wireless Intrusion Detection Systems (WIDS), where 802.11 fingerprints could be observed.

2.2 What is 802.11?

802.11 is a link-layer protocol standard ratified by the IEEE. The first version of the standard was ratified in 1997 and the most recent revision was ratified in 1999 and reaffirmed in June 2003 [2]. Alternative data rates and PHY-layer protocols are specified in amendments 802.11b-1999 and 802.11a-1999 respectively. The *Wireless LAN Edition* is a compilation of the standard and its amendments. Many people equate "Wi-Fi" with 802.11. Wi-Fi is a term created by the Wi-Fi assocation[3]. It is quite possible for a device to be Wi-Fi compliant without fully complying with the 802.11 standard.

IEEE Std 802.11 is a Media Access Control (MAC) and Physical Layer (PHY) standard governing wireless local area networks operating in the ISM band which is unlicensed radio spectrum. This required the 802.11 Task Group to deal with problems that have no simple analogy in the wired world.

One of the most obvious problems is the unreliability of a wireless link. The standard operates in unlicensed spectrum and therefore competes with cordless phones and other wireless networks for the medium. Different wireless networks using the same frequency must co-exist. The designers had to take into account various means to stop independent networks from unfairly impacting the performance of each other. The 802.11 standard includes features to address this problem. These include positive acknowledgement with retransmission, and special medium access control frames called Request To Send (RTS) and Clear To Send (CTS).

In summary, the 802.11 standard is in many ways more complicated than its wired-Ethernet counterpart due to issues that arise in a wireless environment. It has to deal with many problems that have no wired-side analogy. Ultimately, it is this complexity that leads implementations to vary, making fingerprinting possible.

2.3 Finding an 802.11 Fingerprint

An implementation comprises a driver, radio chipset, firmware, and possibly some user-space applications. Ideally, one would be able to identify any component of a given implementation and further refine identification of each software component by its version. Whether it is possible to identify these components depends largely upon behaviors not governed by the standard and where they are implemented. As we shall see, there is even deviation from the standard within the industry that presents very useful opportunities for fingerprinting. Developing 802.11 fingerprints is largely an exploratory exercise in determining how an 802.11 implementation behaves uniquely.

The strength of a fingerprint determines whether the components of an implementation can be identified individually. The fingerprints described in this paper afford reliable identification of 802.11 chipsets, drivers, and in some cases, different versions of the same driver. No attempt was made to differentiate firmware versions or userland applications that might influence the behavior of the driver.

One of the most unique aspects of 802.11 implementation fingerprinting is that many characteristics of the implementation are controlled by hardware. However, there is a trend in modern 802.11 chipsets to push more and more functionality into software. Popular examples of these chipsets include products from Atheros and Ralink. Though it seems unlikely, it is quite possible that drivers for software based radio chipsets (such as products from Atheros and RaLink) could be patched, allowing them to mimic the details of other implementations. Doing this would allow an attacker to have his driver or chipset intentionally misidentified, perhaps to sidestep a fingerprint-aware WIDS.

Many other devices however have certain aspects that cannot be controlled from software. The older Prism2 generation of chipsets is the best example of a chipset that operates somewhat independently of the driver.

2.4 Organization of Paper

This paper is organized into the following chapters. Chapter 3 provides a brief overview of the relevant portions of the IEEE 802.11 MAC rules. Chapter 4 covers the algorithms used to analyze the duration field. Chapter 5 analyzes the accuracy of this analysis. Chapter 6 contains future work and concluding remarks. Finally three appendices are also included: Appendix A lists the results for all matching metrics covered in Chapter 5. Appendix B covers implementation details that can be used to validate the techniques and results. Appendix C gives a brief overview on running the associated tools. Appendix D contains detailed information regarding every 802.11 implementation tested.

Chapter 3

Carrier Sensing in the 802.11 MAC

This chapter provides the relevant background of the 802.11 MAC needed to understand the fingerprinting algorithms covered in chapter 4. This background is by no means a complete description of the 802.11 standard.

3.1 802.11 Basics

Standard 802.11-1999 specifies Medium Access Control (MAC) and Physical (PHY) layer protocols. There are two types of MAC protocols described, Point Coordination Function (PCF) and Distributed Coordinated Function (DCF). It is possible to alternate between them. When the PCF is operating, the medium is in a *contention-free period* since the point coordinator, an access point, controls all access to the medium. When end stations compete for the medium, including the access point, they use the DCF MAC protocol. This period is called a *contention period.*

The standard specifies three different frame types: control, management, and data. Control frames are used for medium reservation and acknowledgements, and have a real-time processing requirement. Medium reservation control frames are not confined to a single network; they are intended to be processed by all stations on a given channel even though they may belong to different wireless networks, or *Basic Service Sets* (BSS). These frames carry a duration field that is essentially an announcement of a station's intention to use the medium for a period of time. Stations operating on the same channel should observe the announcement regardless of the BSS to which they belong. Otherwise they risk interference with their own transmissions. In this way, multiple Basic Service

Sets can coexist on the same channel.

MAC management in 802.11 includes authentication and association with an access point. It also includes provisions for locating networks via probe requests and beacon packets. Management frames handle all of these tasks.

Finally, data frames are used to transmit data.

3.2 Physical and Virtual Carrier Sense

The 802.11 standard specifies two ways to determine if the medium is busy. The first is a physical carrier sense. 802.11 specifies that any PHY must provide a technique to sense if the medium is busy. The function in the PHY layer responsible for this is called the clear channel assessment (CCA).

Two clients that belong to the same BSS may not be within radio range of each other. Therefore, neither will be able to detect energy on the medium necessary to do a CCA. Further, it is more efficient in some cases for a client to reserve the medium in advance, for instance, for an acknowledgement which can be sent immediately upon receipt of a frame. Both cases are handled using a virtual carrier sense mechanism. It consists of a Network Allocation Vector (NAV) maintained by each client. The NAV can be thought of as a client's best guess as to how long the medium will be busy. The client's NAV is updated in response to receiving a frame whose *duration field* contains a value that exceeds the current NAV value.

The duration field is found in nearly every packet. It is not included in Power-Save Poll frames, as the bits are used for the association ID field. Conceptually the duration field of a frame is the amount of time the transmitting client wishes to reserve the medium for itself to send subsequent frames, including any replies expected of the recipient such as acknowledgements. How this value is computed depends on the exact type of frame it is in. The duration field is 16 bits. Therefore the largest value it could reserve the media for is 65,535 microseconds. However the standard explicitly says to ignore any values greater than 32,767.

In a typical scenario where a client is not sending an unfragmented data frame, the duration field will be the amount of time it takes for the inter-frame spacing, combined with the time required for the receiving station to send an ACK packet; in other words, a constant. In management types (such as beacons) or some control types (such as ACKs) no more traffic is needed, and the duration field is set to zero.

In more complicated scenarios involving fragmentation, the duration field will include the time required not only for the inter-frame spacing and ACK, but for the rest of the fragments. When using frames to explicitly reserve the media

(RTS/CTS) it is the duration field that specifies how long the media is reserved. Finally an important aspect of the PCF is implemented by using the duration field to interoperate with stations on the same channel using the DCF.

Chapter 4

Statistical Analysis of Duration Field

As mentioned in chapter 3, the duration field is a 16 bit value which describes how long the station that currently has access to the medium *intends to keep it*, after the current transmission. Even though the duration field is 16 bits wide, it only takes on a few discrete values. Common values are 0 (for packets that are not acknowledged such as management frames broadcast during a Contention Period), and the time it takes for a SIFS (Short Interframe Spacing) interval plus an acknowledgment, used in transmitting unicast data frames.

Variables that can affect the duration field include some parameters of the local Basic Service Set specified in a beacon's fixed flags field. These include short slot time, short pre-amble, and of course, the data rates supported. The net result of this is that ideally a unique fingerprint for a given implementation would be taken across all possible variations of these parameters. For this work, four databases were created. The databases currently have human-friendly names (the name of the AP used to create them). In the future, the number of databases will grow large enough that an algorithmic naming scheme (rates-flags for example) will be employed.

Since the performance of this technique varies with the parameters of the Basic Service Set with which it is associated, a brief introduction to the four networks it was developed and tested against is given below.

Table 4.1 represents data about the four WLANs on which all experiments in this chapter were performed. They were chosen to give a good estimate of real world network deployments. Lexie is a b-only Cisco aironet 350 . Mixed–wrt54g is a rev5 Linksys wrt54g running in mixed mode. Mixed–Airplus is a D-link DI-524, and G–wrt54g is a rev5 Linksys wrt54g in g-only mode. The models of

Name	**Rate**	**Flags**
Lexie	1.0 - 11.0 Mb/sec (b-only)	0x0021 (short pre-amble)
mixed–wrt54g	1.0 - 54.0 (mixed)	0x0401, 0x0001 (disables SST if a b client is in range)
mixed–AirPlus	1.0 - 54.0 (mixed)	0x0421 (SST, short pre-amble)
G–wrt54g	1.0 - 54.0 (G-only)	0x0421 (SST, short-preamble)

Table 4.1: Summary of databases created

the Access Points used are mentioned to give the reader some sense of market representation. The databases generated from each AP are not tied to that specific AP. Clients should respond identically in any BSS with the same set of parameters listed above.

4.1 What is in a Print Database?

The tools and techniques described in this chapter all operate on a surprisingly little amount of information, stored in what we call a *print database*. There is a fingerprint for each implementation. A fingerprint comprises a list of records of the form (packet_type, duration-value, count) which reflects for the given packet type, the number of times the given duration value appeared. All data and management frames are observed while control packets are discarded.

Two example prints from the same database are given in Tables 4.2 and 4.3. Both prints were generated from packet captures done while a client associates, obtains an IP address from DHCP, and proceeds to load a few web pages. With so little activity, there is a remarkable range of behaviors. These two prints were chosen to illustrate the range of behaviors between Atheros and Prism chipsets.

packet-type	**(duration [duration observed frequency / number packets of this type])**
Assoc Request	(314 [2/2])
probe request	(0 [75/77]) (314 [2/77])
Authentication	(314 [2/2])
Data	(162 [167/278]) (0 [111/278])
Null Function	(162 [597/597])

Table 4.2: Implementation-Id: 1 (Atheros, ar5211.sys), database: Lexie

packet-type	**(duration [duration observed frequency / number packets of this type])**
Assoc Request	(258 [13/13])
probe request	(0 [50/50])
Authentication	(53389 [13/13])
Data	(213 [1229/1303]) (0[54/1303]) (223[20/1303])
Null Function	(37554 [16/16])

Table 4.3: Implementation-Id: 9 (Prism-2.5, smc2532w.sys), database: Lexie

Two things stand out immediately from these fingerprints. The first is that the second implementation (the prism2.5 based implementation) uses duration values that are entirely different than those used by the better behaved Atheros card. Secondly, the prism2.5 based implementation uses two illegal duration values. The standard says that any values greater than 32767 should be ignored.

Though these two implementations are different enough that they can be easily distinguished, most of the other sampled implementations fell somewhere between them. To get better resolution, two ratios were introduced: the ratio of packets with a given duration relative to the total number of packets sampled, and the ratio of pairs (packet type, duration) for a given packet type and duration relative to the total number of packets seen of that packet type.

Though these numbers can fluctuate across different samples for the same implementation, they proved to be stable enough to cause an improvement in the algorithms that use them. Tables 4.4 and 4.5 show this information for the Atheros fingerprint above in Table 4.2.

packet-type	**(duration [ratio of packets with this duration, for given packet-type])**
Assoc Request	(314 [100%])
probe request	(0 [97%]) (314 [3%])
Authentication	(314 [100%])
Data	(162 [60%]) (0 [40%])
Null Function	(162 [100%])

Table 4.4: Implementation-Id: 1 (Atheros, ar5211.sys), database: Lexie

duration	ratio of packets with this duration, regardless of packet-type
0	19%
162	80%
314	1%

Table 4.5: Implementation-Id: 1 (Atheros, ar5211.sys), database: Lexie

4.2 The Duration Matching Algorithm

The matching algorithm expects as input a packet capture (pcap) file gathered by sniffing the exchange between an 802.11 NIC and one of the 802.11 Access Points for which a print database has been assembled for a collection of 802.11 implementations. The input is compared against each print in the database using a particular matching metric. We give five matching metrics. Each matching metric produces a scalar quantity measuring the degree of match between the input and a print. The algorithm outputs a list of 802.11 implementations ordered by decreasing degree of match.

The metrics are presented in order of increasing complexity. Values from one metric are not intended to be comparable to values from another.

4.3 SimpleComparison Metric

SimpleCompare is the first of three related metrics, the other two being MediumCompare and ComplexCompare. SimpleCompare is unique in that it compares the input against a print in the database without using any information about other prints in the database. That means that if a certain duration value is incredibly unique, such as the illegal ones only found in prism2 based implementations, it has no opportunity to take this into consideration.

All the metrics presented in this chapter break the fingerprints up into two different sets of data points. The first set is a set of pairs of the form (duration value, count). The second set is a set of triples of the form (packet type, duration value, count). The diagrams below leave the count component of both tuples out for clarity.

SimpleCompare, as well as the other metrics, has three different flavors. It can be computed using just the (duration value, count) pairs, or it can be computed using just the (packet type, duration value, count) triples. Finally the results from both analyses can be combined. Combining the results of these metrics is simply a matter of adding the return values from both metrics.

SimpleCompare utilizes two functions that are used throughout this chapter. They are used to compute the duration ratios in tables 5.2 and 5.3, and are defined as follows.

$$duration\_ratio(p,d) = \frac{\text{\# of packets with packet\_type = p, duration = d}}{\text{\# of total packets with packet\_type = p}}$$

The SimpleCompare metric is defined below. The input packet capture is denoted by L. R, on the other hand, denotes a print in the capture database for a particular 802.11 implementation.

$$duration\_ratio(d) = \frac{\text{\# of packets with duration = d}}{\text{\# of total packets observed}}$$

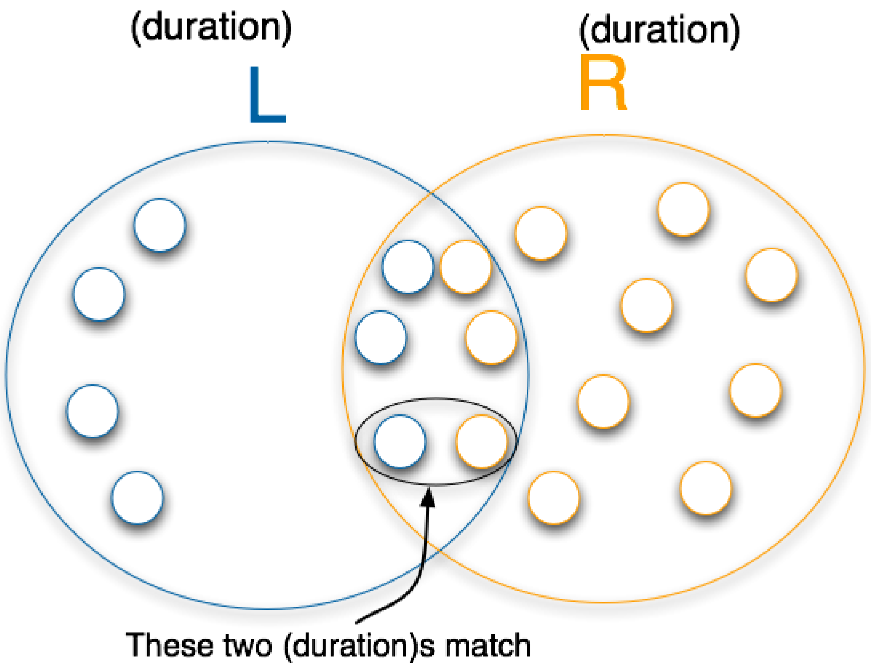

Figure 4.1: SimpleCompare duration-value only analysis

```
sum = 0;
for every duration-value d $\in (L \cap R)$
    sum += 1.0 $- |L.duration\_ratio(d) - R.duration\_ratio(d)|$
return sum;
```

The metric weights common durations that appear in their respective prints at roughly the same rate more heavily than ones that do not. However, Simple-Compare doesn't pay attention to duration values that aren't in the intersection, as illustrated in Figure 4.1, even though the number of values not in the intersection is clearly a strong indicator of how close two prints match. It also doesn't have any idea of how unique any specific duration values are across the entire database.

At first, this lack of a global perspective on the relative likeliness of seeing duration values seemed that it would hinder this algorithm significantly. Consider the case when a prism2 sample is input that uses all the same illegal duration values as the one stored in the database, but at very different rates. Simple-Compare lacks the information to realize that the illegal values identify a prism2 implementation, and could grade this sample incorrectly.

At this point, SimpleCompare is also ignoring the packet type in which the duration values appear. This can cause two problems. One is that two different implementations use the same duration value, but in consistently different packet types (probe requests versus association responses for example). The other is that the ratio that duration values are used across all packet types fluctuate largely across packet samples, but the rate is much more consistent when confined to a particular packet type. Both of these problems are addressed by considering the packet types when looking at durations.

We can reuse SimpleCompare except this time we run it against the (packet type, duration) pairs, as illustrated below.

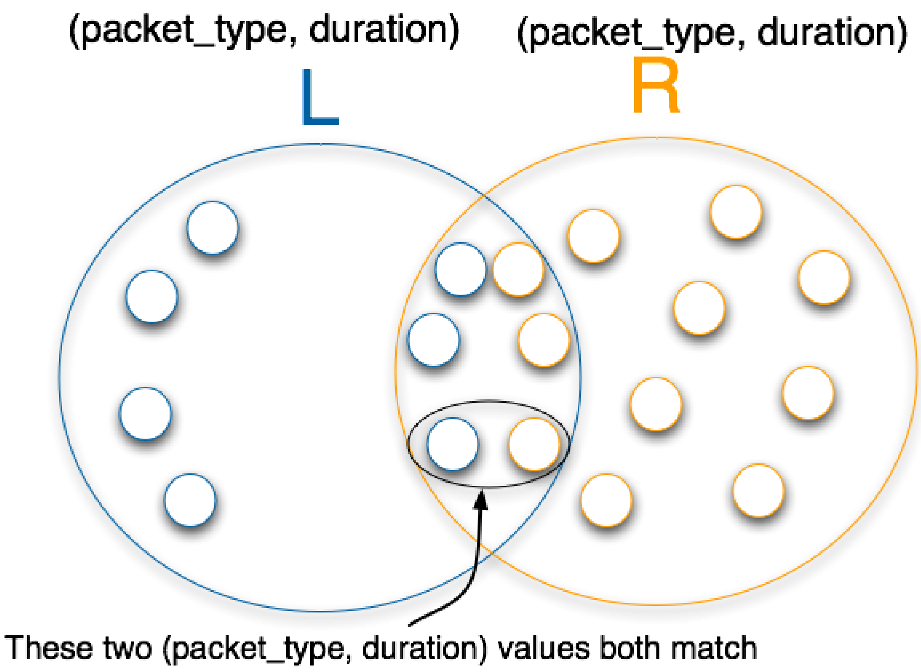

Figure 4.2: SimpleCompare (packet type, duration) analysis

```
sum = 0;
for every pair(packet_type p, duration-value d) $\in (L \cap R)$
    sum += $1.0 - |L.duration\_ratio(p, d) - R.duration\_ratio(p, d)|$
return sum;
```

4.4 MediumCompare Metric

SimpleCompare does not account for highly-unique duration values. MediumCompare was created as an alternative to deal more intelligently with such duration values. Intuitively, if two prints both use duration values that are globally unique (i.e. illegal values generated by prism2-based implementations) then this should count more than matching very common values such as 0.

Like SimpleCompare, the MediumCompare metric compares an input pcap with every print in the database except that for each print in the database, it also considers global duration uniqueness by examining the rest of the database. It computes one of two weights, either duration uniqueness, or packet type duration uniqueness, depending on the data set as follows.

When computing duration uniqueness the metric counts the total number of unique (implementation, duration value) pairs in the entire database. This does not take into account how often an individual duration value appears in packets for a given implementation. Rather, it counts how often a duration value is used across all implementations. If two implementations both use duration value 314, but one uses it 1% of the time, and the other uses it 80% of the time, both of these implementations will contribute the same amount to duration uniqueness.

$$duration\_uniqueness(d) = \frac{\text{\# of unique (implementation, duration) tuples}}{\text{\# of unique (implementation, duration = d) tuples}}$$

Similarly packet type duration uniqueness is computed by counting the total number of unique (implementation, packet type duration) values across the entire database.

$$duration\_uniqueness(p, d) = \frac{\text{\# of unique (implementation, packet\_type, duration) tuples}}{\text{\# of unique (implementation, packet\_type = p, duration = d) tuples}}$$

Once these two values have been computed MediumCompare is very similar to SimpleCompare.

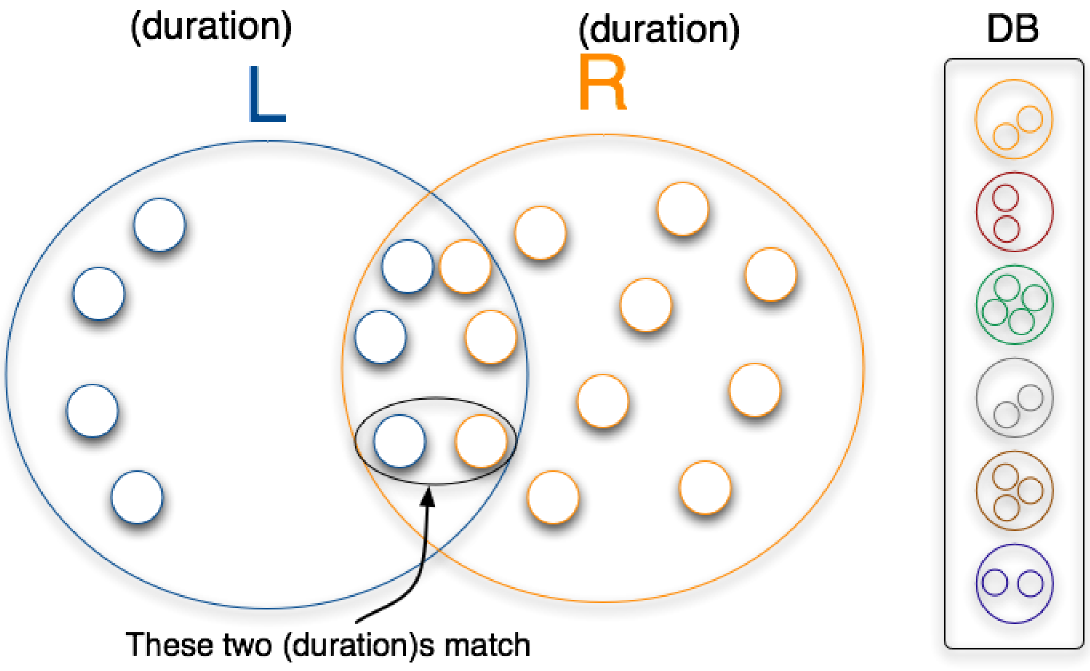

Figure 4.3: MediumCompare duration-value only analysis

```
sum = 0;
for every duration-value d $\in (L \cap R)$
    sum += **duration_uniqueness(d)** $*$
    $1.0 - |L.duration\_ratio(d) - R.duration\_ratio(d)|$
return sum;
```

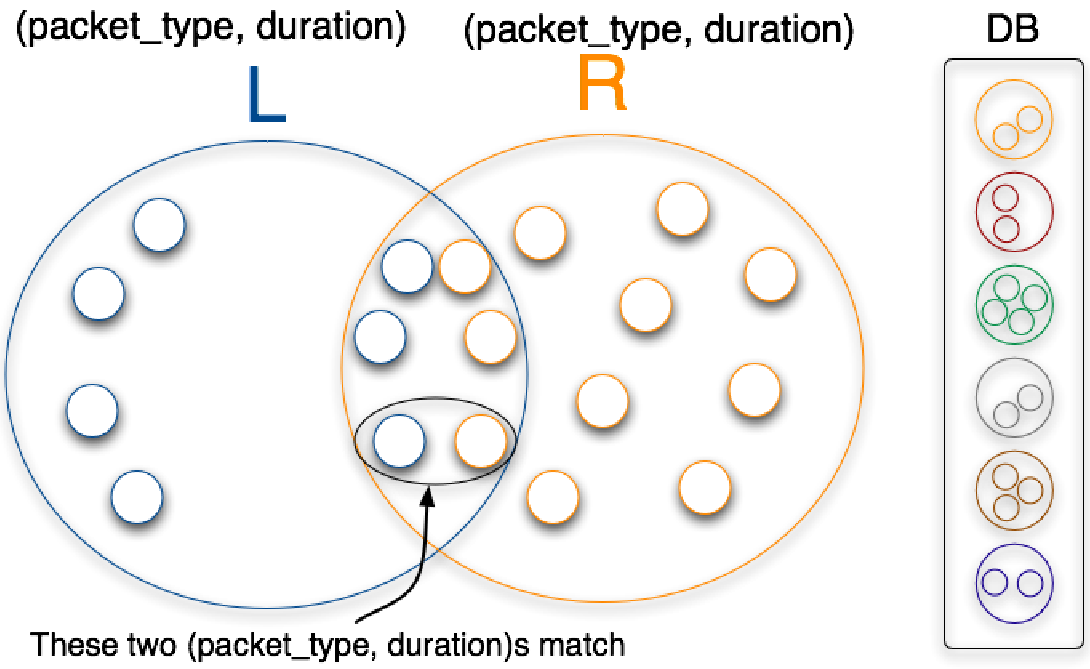

Figure 4.4: MediumCompare (packet_type, duration) analysis

```
sum = 0;
for every packet_type p, duration-value d $\in (L \cap R)$
    sum += **duration_uniqueness(p,d)** $*$
    $1.0 - |L.duration\_ratio(p,d) - R.duration\_ratio(p,d)|$
return sum;
```

4.5 ComplexCompare Metric

Notice that the MediumCompare and SimpleCompare metrics ignore durations outside the intersection. One might think that such information would improve a fingerprinting capability, however, we found this is not the case. To illustrate, a metric called ComplexCompare was investigated. It was designed to take into account all the data points that don't fall in the intersection of two prints. ComplexCompare computes the metric that MediumCompare does and then visits every data point not in the intersection of the prints, computing duration uniqueness, or packet type duration uniqueness and then subtracting this value from the metric. The motivation for this behavior is that if L contains very unique durations and R doesn't, then the metric should be decreased proportionally by the uniqueness of these values.

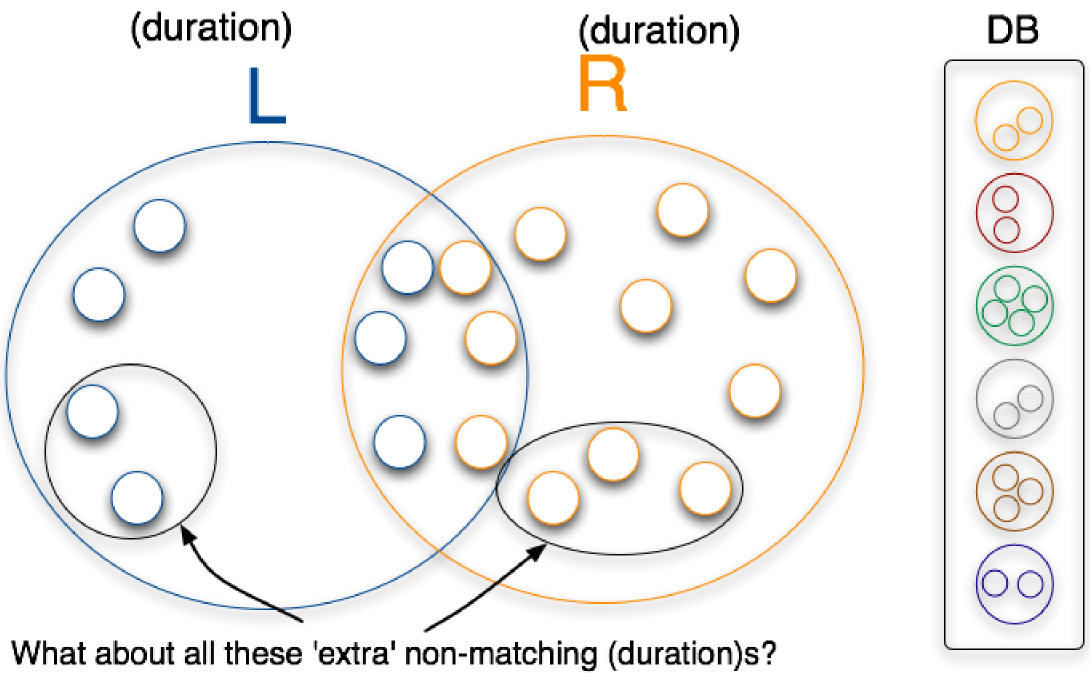

Figure 4.5: ComplexCompare duration-value only analysis

```
ret = MediumCompare(L,R);
sum = 0;
for every duration-value d $\notin (L \cap R)$
    sum += **duration_uniqueness(d)**
return ret - sum;
```

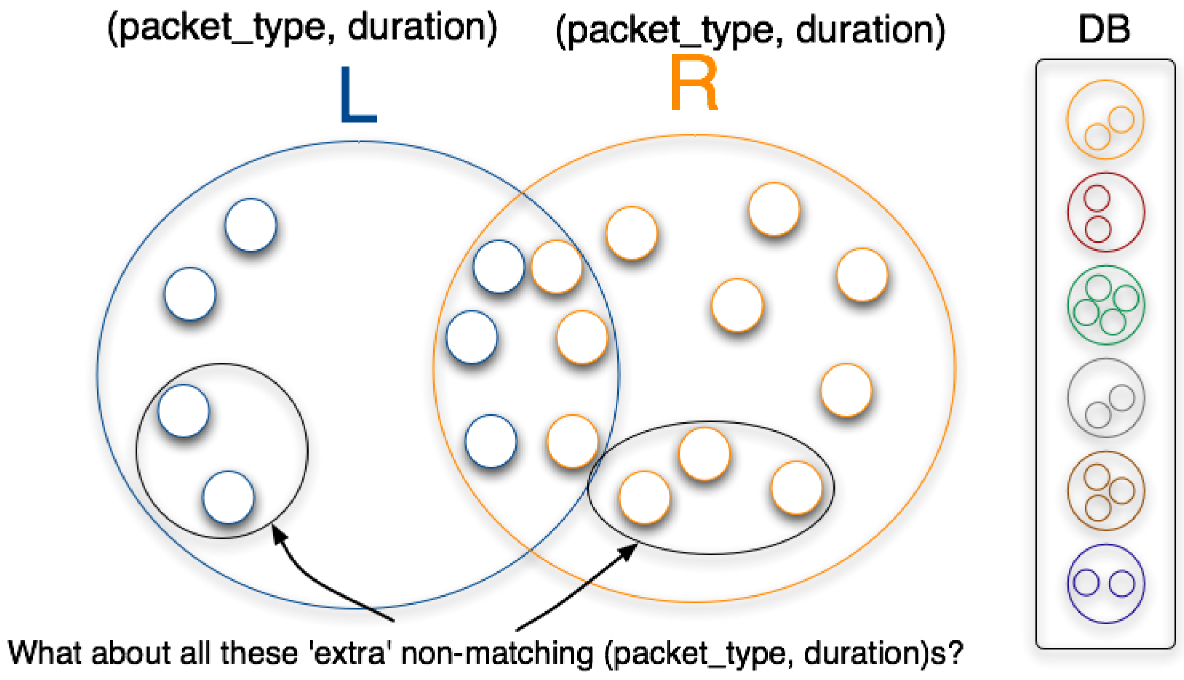
(packet_type, duration)
L
(packet_type, duration)
R
DB
What about all these 'extra' non-matching (packet_type, duration)s?

Figure 4.6: ComplexCompare (packet_type, duration) analysis

```
ret = MediumCompare(L,R);
sum = 0;
for every packet_type p, duration-value d $\notin (L \cap R)$
    sum += **packet_type_duration_uniqueness(p, d)**
return ret - sum;
```

4.6 BayesCompare Metric

BayesCompare was created as an attempt to use a well understood rule to classify 802.11 implementations. In document classification, the problem is that of given a set W of words appearing in a document, classify the document as belonging to one of several categories. One takes the category to be the category C that maximizes $P(W|C)\ P(C)$. The conditional probability $P(W|C)$ comes from a training set of documents known to be in category C. If we take W to be the set of durations occurring in a given packet capture that we want to identify by implementation then $P(W|C)$ becomes the probability of W occurring in a capture given that the capture comes from 802.11 implementation C.

Classification in this manner is only as good as the training set (print database). A given training set may not yet know that implementation C can produce duration D. Hence $P(W|C)$, which is approximated from the training set, is zero when W contains D even though W may contain another duration that uniquely identifies C. Further, approximating $P(C)$ is problematic, as it is the probability of seeing a given 802.11 implementation. One might approximate it by perhaps chipset market share but this would be somewhat inaccurate because it ignores the fact that a device driver is part of an 802.11 implementation we wish to identify. Getting an accurate approximation of it is difficult so we chose to ignore it. This of course puts the metric at a slight disadvantage compared to the other metrics, as we shall see.

Let X be an 802.11 implementation for which a fingerprint exists in the print database. Let L be the duration fingerprint arising from an input pcap file. We want the probability that the input pcap file originated with implementation X given L: $P(X|L)$. Using Bayes rule, $P(X|L) = (P(L|X)P(X))/P(L)$. The idea here is to use these conditional probabilities to rank the degree of a match between L and each fingerprint in the print database. Therefore, we did not compute $P(L)$ for a given input pcap as it is constant across all fingerprints in the database. Of course probability P(X) is not constant across all fingerprints but computing it is problematic, as discussed above. Therefore, we didn't compute it as part of the conditional probability. Further, to simplify things, we approximated $P(L|X)$ as the product $P(d_1|X) * P(d_2|X) * \cdots * P(d_n|X)$ where $d_1, d_2, \ldots$ are the distinct durations that appear in L. This assumes that the individual duration values in L occur independently which one can argue isn't true since the durations occur in sequence for certain control frames, for instance, duration values in ACK, RTS and CTS frames. But as mentioned previously, control frames are ignored in fingerprints.

If L denotes the fingerprint arising from an input pcap file and R a fingerprint in the print database then we take the preceding product to be Π R.duration_ratio(d) where d ranges over all durations in L. And when taking into account packet types in which durations occur, it becomes Π R.duration_ratio(p, d) where p and d range over all packet types and durations respectively where duration d occurs in a packet of type p in L.

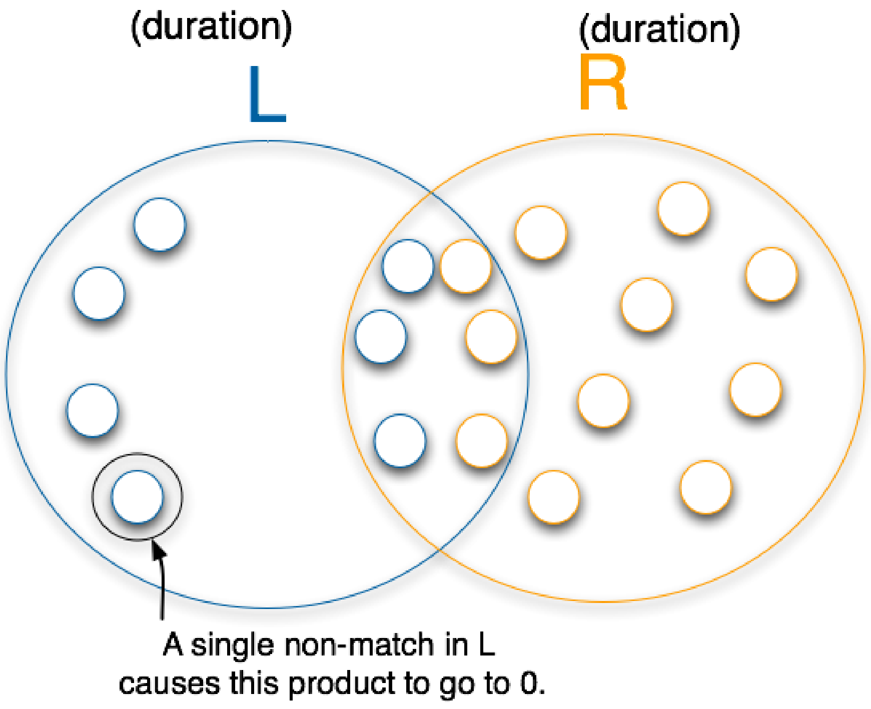

Figure 4.7: BayesCompare duration value only analysis

```
ret = 1.0
for every duration-value d $\in L$
    ret *= **R.duration_ratio(d)**
return ret;
```

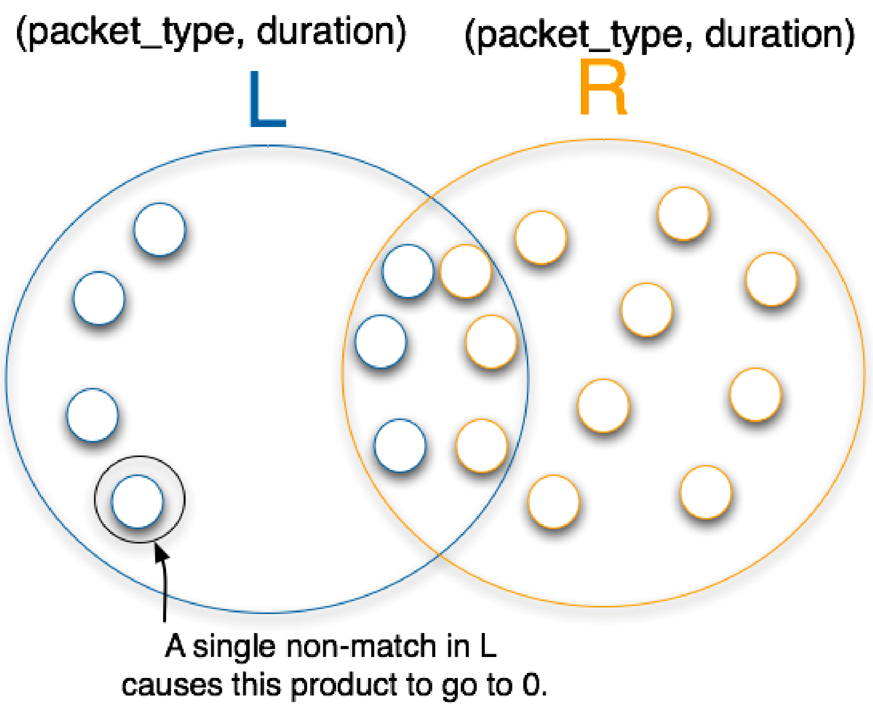

Figure 4.8: BayesCompare (packet_type, duration) analysis

```
ret = 1.0
for every packet_type p, duration-value d $\in L$
    ret *= **R.duration_ratio(p, d)**
return ret;
```

4.7 Modified BayesCompare Metric

Another variant of BayesCompare was investigated. As pointed out above, conditional probability $P(L|X)$ can become zero if L has a duration that has not yet been learned to be producible by implementation X, perhaps because the print database hasn't been updated for some time. So another version was explored where only duration values that fall in the intersection of an input fingerprint L and a database fingerprint R are included in the calculation of $P(L|X)$. So the product becomes Π R.duration_ratio(d) where d ranges over all durations in L ∩ R, and Π R.duration_ratio(p, d) where p and d range over all packet types and durations respectively where duration d occurs in a packet of type p in L ∩ R.

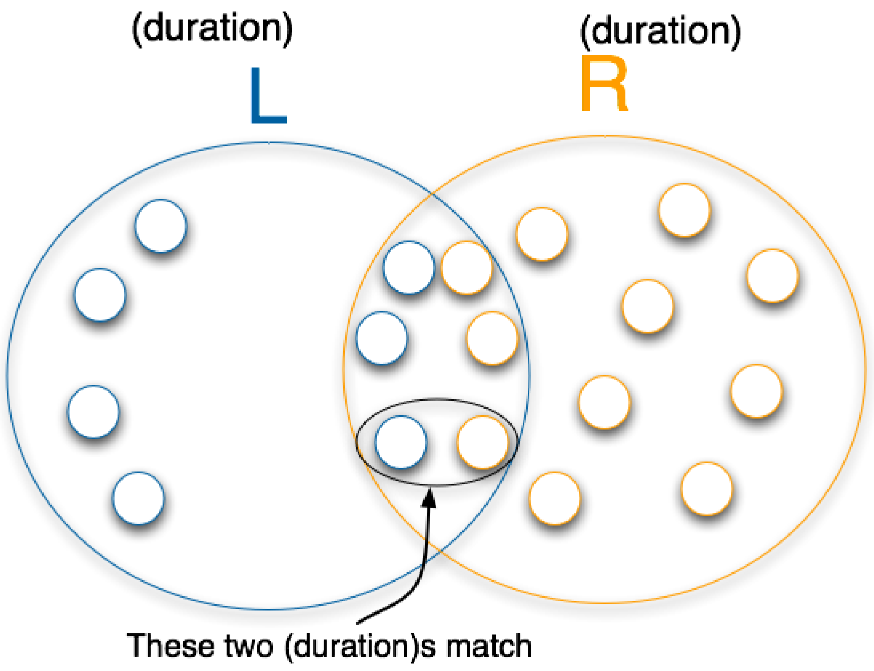

Figure 4.9: BayesCompare-Modified duration value only analysis

```
ret = 1.0
for every duration-value d $\in (L \cap R)$
    ret *= **R.duration_ratio(d)**
return ret;
```

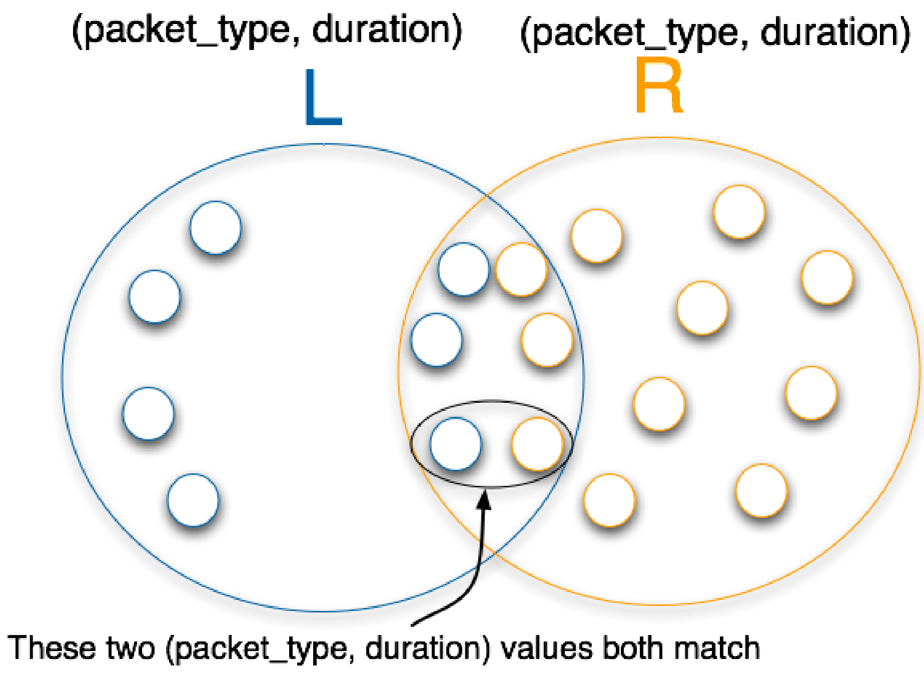

Figure 4.10: BayesCompare-Modified (packet-type, duration) analysis

```
ret = 1.0
for every packet_type p, duration-value d $\in (L \cap R)$
    ret *= **R.duration_ratio(p, d)**
return ret;
```

Chapter 5

Results

This section presents performance results for each of the duration-based matching metrics described in chapter 4. To compare the performance of these metrics, a rating system was devised as follows. Each metric was exercised across four print databases using three packet capture samples s_1, s_2, and s_3 as input for each 802.11 implementation. We define for each 802.11 implementation I, a success probability R_I for a matching metric M. It is the probability that M correctly identifies a sample, that is, identifies that sample as originating with I when it does indeed originate with I.

For example, consider Table 5.1. This print database has 13 fingerprints hence there are 13 entries. The table was produced by using the MediumCompare metric on a particular sample. The tables tells us that this metric believes the sample originated with the Broadcom-MiniPCI (ID 10 in the table) since it has rank zero. But this is incorrect. The sample originated with the Apple-Airport Extreme (ID 5), which has rank "1". So we take as SimpleCompare's probability of succeeding when the sample originates with Apple-Airport Extreme to be (13 - rank)/13 or (13 - 1)/13 since the correct implementation is given rank "1" by the metric.

Now since there are three samples, we extend R_I for a metric M to be $R_I = [(13 - s_1 rank) + (13 - s_2 rank) + (13 - s_3 rank)]/(3 * 13)$ (eq 5.1) where s_i rank is the rank assigned by M to the 802.11 implementation I that actually produced sample s_i. If the probability that I occurs is P_I then the success rate of M is the unconditional probability of success given by $P_{I1} * R_{I1} + P_{I2} * R_{I2} + \cdots + P_{I13} * R_{I13}$

Each term in this sum is the product of the probability of seeing a sample from one of the 13 implementations and the probability of M succeeding to identify it in that case. So M could have a good overall success rate even though it performs badly when trying to identify a sample as belonging to some 802.11 implementation if that implementation doesn't arise often. However, we shall assume that implementations are equally likely to occur. In that case, the sum above becomes $(R_{I1} + R_{I2} + \cdots + R_{I13})/13$.

rank	**score**	**ID**	**Model**	**chipset**	**driver**
0	79.03	10	Broadcom-MiniPCI	BCM4318	bcmwl5.sys
1	78.91	5	Apple-Airport Extreme	BCM4318	AppleAirport2.kext
2	73.51	6	Zonet-ZEW1520	BCM4306	bcmwl5.sys
3	56.03	7	Intel-IPW220BG	IPW2200BG	w29n51.sys
4	54.74	13	Cisco-Aironet-350	Prism2	pcx500.sys
5	53.06	11	Sony-PSP	unknown	unknown
6	47.19	8	D-Link-dwl-g122	RA2570	rt2500usb.sys
7	39.95	4	Proxim-Orinoco Silver	AR5212	ntpr11ag.sys
8	39.55	3	Proxim-Orinoco Silver	AR5211	ntpr11ag.sys
9	39.47	2	Proxim-Orinoco Silver	AR5212	ntpr11ag.sys
10	38.53	1	Linksys-WPC55AG	AR5212	ar5211.sys
11	28.55	12	Nintendo-DS	unknown	unknown
12	22.61	9	SMC-2532W-B	Prism2.5	smc2532w.sys

Table 5.1: Ordered list generated from a matching metric

5.1 SimpleCompare

The following tables show how well SimpleCompare did against all four databases. The number of samples represents how many pcap files the input fingerprints were computed across. 1-sample means that the fingerprint was computed only from the first sample for a given implementation, while 3-sample means all three pcap files were used to generate the print.

Table 5.2 below shows how well SimpleCompare does when it is only analyzing durations, not (packet_type, duration) pairs. Table 5.3 shows how well Simple-Compare does when it only analyzed (packet_type, duration) pairs. Table 5.4 shows the results when both techniques are combined.

Though combining the two techniques did not improve the overall average, it did

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.9724	0.9546	0.9745	0.9115
2-samples	0.9783	0.9408	0.9630	0.8854
1-samples	0.9586	0.9408	0.9583	0.8333
Average	0.9698	0.9454	0.9653	0.8767
Total Average		**0.9393**		

Table 5.2: SimpleCompare, duration values only

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.9921	0.9606	0.9769	0.9688
2-samples	0.9901	0.9645	0.9861	0.9479
1-samples	0.9744	0.9586	0.9745	0.9531
Average	0.9855	0.9612	0.9792	0.9566
Total Average		**0.9706**		

Table 5.3: SimpleCompare, (packet_type, duration) pairs only

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.9901	0.9882	0.9884	0.9531
2-samples	0.9882	0.9684	0.9861	0.9531
1-samples	0.9744	0.9625	0.9769	0.9115
Average	0.9842	0.9730	0.9838	0.9392
Total Average		**0.9701**		

Table 5.4: SimpleCompare combined.

have one important effect. In the combined table, scores consistently increase with sample size, across all databases. This is not the case in either of the two tables preceding it. This is a very desirable property, and could arguably be worth the minor price paid in overall accuracy.

5.2 MediumCompare

Although MediumCompare has significantly more information at its disposal than SimpleCompare (since MediumCompare gets the entire print database over which to compute weights) it only improved its best-case score by .0017 relative to SimpleCompare. This seems to indicate that while knowing certain duration values are highly unique, the implementations that used them identified them enough already that the extra weight given to them wasn't needed in general.

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.9921	0.9684	0.9907	0.9635
2-samples	0.9901	0.9625	0.9884	0.9375
1-samples	0.9882	0.9546	0.9861	0.9427
Average	0.9901	0.9618	0.9884	0.9479
Total Average		**0.9721**		

Table 5.5: MediumCompare, (packet_type, duration) pairs only

5.3 ComplexCompare

ComplexCompare did not improve upon its predecssors, performing consistently worse then Simple or MediumCompare. In fact, no algorithm tested that attempted to take into consideration duration values that don't match ever made an improvement upon those that simply ignored them.

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.9744	0.9566	0.9722	0.8958
2-samples	0.9763	0.9507	0.9722	0.9062
1-samples	0.9803	0.9507	0.9931	0.9323
Average	0.9770	0.9527	0.9792	0.9114
Total Average		**0.9551**		

Table 5.6: ComplexCompare, (packet_type, duration) pairs only

5.4 BayesCompare

Considering the significant disadvantage that BayesCompare is at relative to the other metrics, it performed quite well. It is quite possible that in practice BayesCompare could be the most accurate. This could be accomplished by mapping the probability of seeing particular chipset, device-driver implementation back to the marketshare of the chipset. This optimization is not implemented in the current system, and both flavors of BayesCompare perform consistently worse than the other metrics presented.

5.5 Modified BayesCompare

The Modified BayesCompare did consistently worse than BayesCompare. This seems to indicate that contrary to our original suspicion, having the conditional probabilities go to zero when an unknown duration value is encountered is a good idea.

5.6 Results Summary

A table representing a summary of the algorithms' performance is below. It is interesting to note that while MediumCompare out-performed SimpleCompare, it only did so by a small margin. This seems to indicate that SimpleCompare has little trouble identifying the implementations that use globally unique duration values, even though SimpleCompare is unaware of the uniqueness.

Metric	**dur**	**type, dur**	**combined**
SimpleCompare	0.9393	**0.9706**	0.9701
MediumCompare	0.9381	**0.9721**	0.9621
ComplexCompare	0.9370	**0.9551**	0.9500
BayesCompare	0.8456	0.9190	**0.9209**
BayesCompare-modified	0.2866	**0.9243**	0.7502

Table 5.7: Results summary

Chapter 6

Conclusions and Future Work

The techniques outlined in this paper provide passive fingerprinting algorithms that consistently perform at 95% or high accuracy across many implementations and different 802.11 network parameters. It also demonstrated a new level of resolution by uniquely identifying different version of the same driver in many cases. These techniques can be used to target link layer device driver exploits with a high degree of reliability, or integrated into WIDS to help enforce access control.

6.1 Other Matching Metrics

An entirely different type of metric, dubbed FuzzyCompare, was also developed. Fuzzy Compare works by comparing every (packet_type, duration) tuple in a print (L) *to every other tuple* in the other print, R. For each comparison it modifies the score based on a set of coefficients and the global uniqueness of the current duration value.

The interesting aspect about this algorithm is that the coefficients were actually brute-forced by another program to (a modification to duration-print-grader) to find the best possible combination of coefficients. This lead it to produce impressive results, but it couldn't be shown that the coefficients generated would generalize well to data sets with unknown inputs.

FuzzyCompare extended the notion of a fingerprint to include whether or not certain implementations make use of the various flag bits inside the 802.11 header. This really simplified down to tracking which implementations utilize power savings, as the rest of the flags were always unused. Tracking a few more bits seemed to give FuzzyCompare a significant advantage over the other algorithms which strictly analyzed the duration field. Such a hybrid technique will probably yield better real world results. The 802.11e QOS amendment looks like it will provide more bits for this type of analysis.

6.2 Future Work - MAC vs. PHY Fingerprinting

The 802.11 standard is responsible not only for specifying the media access controls of wireless networks, but also the physical (PHY) layer as well. This paper focuses on analyzing the MAC portion of the standard, but one could imagine a tool that analyzes aspects of the PHY for unique signatures.

Such a device would need the ability to analyze the frequency that 802.11 operates in (2.4GHz, 5GHz or the rarely-implemented IR band). Since the goal of the device is to be able to analyze what typical consumer level cards are doing, it would likely need components capable of measuring physical characteristics of the medium with higher levels of precision than that available on commercially-available 802.11 cards. Likely candidates for such a device include measuring the type of preamble used in 802.11 frames and the thresholds used by cards to detect that the medium is busy.

This paper has demonstrated that it is possible to remotely determine which 802.11 implementation generated traffic by analyzing a small sample taken during the association phase. The technique outlined in this paper achieved a new level of resolution that makes it possible to identify not just chipsets, but also device drivers and their versions in many cases.

Appendix A

Complete Results

A.1 Duration Analysis Results

These tables show the results from every experiment conducted using the matching metrics outlined in chapter 5. The values in the tables are the success rate of a matching metric across an entire database.

A.1.1 SimpleCompare Results

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.9724	0.9546	0.9745	0.9115
2-samples	0.9783	0.9408	0.9630	0.8854
1-samples	0.9586	0.9408	0.9583	0.8333
Average	0.9698	0.9454	0.9653	0.8767
Total Average		**0.9393**		

Table A.1: SimpleCompare, duration values only

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.9921	0.9606	0.9769	0.9688
2-samples	0.9901	0.9645	0.9861	0.9479
1-samples	0.9744	0.9586	0.9745	0.9531
Average	0.9855	0.9612	0.9792	0.9566
Total Average		**0.9706**		

Table A.2: SimpleCompare, (packet_type, duration) pairs only

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.9901	0.9882	0.9884	0.9531
2-samples	0.9882	0.9684	0.9861	0.9531
1-samples	0.9744	0.9625	0.9769	0.9115
Average	0.9842	0.9730	0.9838	0.9392
Total Average		**0.9701**		

Table A.3: SimpleCompare combined.

A.1.2 MediumCompare Results

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.9724	0.9606	0.9745	0.9062
2-samples	0.9783	0.9369	0.9630	0.8802
1-samples	0.9606	0.9408	0.9560	0.8281
Average	0.9704	0.9461	0.9645	0.8715
Total Average		**0.9381**		

Table A.4: MediumCompare, duration values only

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.9921	0.9684	0.9907	0.9635
2-samples	0.9901	0.9625	0.9884	0.9375
1-samples	0.9882	0.9546	0.9861	0.9427
Average	0.9901	0.9618	0.9884	0.9479
Total Average		**0.9721**		

Table A.5: MediumCompare, (packet_type, duration) pairs only

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.9862	0.9842	0.9884	0.9375
2-samples	0.9862	0.9645	0.9792	0.9323
1-samples	0.9842	0.9527	0.9745	0.8750
Average	0.9855	0.9671	0.9807	0.9149
Total Average		**0.9621**		

Table A.6: MediumCompare combined.

A.1.3 ComplexCompare Results

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.9684	0.9448	0.9606	0.8906
2-samples	0.9704	0.9290	0.9560	0.8802
1-samples	0.9665	0.9349	0.9722	0.8698
Average	0.9684	0.9362	0.9629	0.8802
Total Average		**0.9370**		

Table A.7: ComplexCompare, duration values only

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.9744	0.9566	0.9722	0.8958
2-samples	0.9763	0.9507	0.9722	0.9062
1-samples	0.9803	0.9507	0.9931	0.9323
Average	0.9770	0.9527	0.9792	0.9114
Total Average		**0.9551**		

Table A.8: ComplexCompare, (packet_type, duration) pairs only

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.9744	0.9606	0.9745	0.8854
2-samples	0.9744	0.9448	0.9745	0.8906
1-samples	0.9763	0.9448	0.9884	0.9115
Average	0.9750	0.9501	0.9791	0.8958
Total Average		**0.9500**		

Table A.9: ComplexCompare combined.

A.1.4 BayesCompare Results

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.9211	0.8698	0.9028	0.7396
2-samples	0.9191	0.8659	0.9051	0.7292
1-samples	0.9034	0.8639	0.8241	0.7031
Average	0.9145	0.8665	0.8773	0.7240
Total Average		**0.8456**		

Table A.10: BayesCompare, duration values only

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.9566	0.9329	0.9745	0.9375
2-samples	0.9566	0.9132	0.9745	0.8698
1-samples	0.9310	0.8935	0.8750	0.8125
Average	0.9481	0.9132	0.9413	0.8733
Total Average		**0.9190**		

Table A.11: BayesCompare, (packet_type, duration) pairs only

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.9329	0.9290	0.9745	0.9375
2-samples	0.9310	0.9211	0.9745	0.9219
1-samples	0.9152	0.9172	0.8727	0.8229
Average	0.9264	0.9224	0.9406	0.8941
Total Average		**0.9209**		

Table A.12: BayesCompare combined.

A.1.5 BayesCompare-Modified Results

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.3136	0.2643	0.2569	0.3229
2-samples	0.2959	0.2446	0.2593	0.3125
1-samples	0.2919	0.2485	0.2639	0.3646
Average	0.3005	0.2525	0.2600	0.3333
Total Average		**0.2866**		

Table A.13: BayesCompare-modified, duration values only

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.9546	0.9191	0.9699	0.9219
2-samples	0.9487	0.9093	0.9699	0.8906
1-samples	0.9290	0.8817	0.9421	0.8542
Average	0.9441	0.9034	0.9606	0.8889
Total Average		**0.9243**		

Table A.14: BayesCompare-modified, (packet_type, duration) pairs only

	lexie	**mixed–wrt54g**	**mixed-AirPlus**	**G–wrt54g**
3-samples	0.7692	0.7416	0.8009	0.8281
2-samples	0.7318	0.7318	0.7870	0.7917
1-samples	0.6746	0.6982	0.7083	0.7396
Average	0.7252	0.7239	0.7654	0.7865
Total Average		**0.7502**		

Table A.15: BayesCompare-modified combined.

A.1.6 Duration Analysis Results Summary

Metric	**dur**	**type, dur**	**combined**
SimpleCompare	0.9393	**0.9706**	0.9701
MediumCompare	0.9381	**0.9721**	0.9621
ComplexCompare	0.9370	**0.9551**	0.9500
BayesCompare	0.8456	0.9190	**0.9209**
BayesCompare-modified	0.2866	**0.9243**	0.7502

Table A.16: Results summary

Appendix B

Implementation Considerations

B.1 PCAP Creation for Duration Analysis

Pcaps created for this project were intentionally not generated by any sort of highly automated process. Captures were created of all cards being powered on and searching for a network before joining. After joining they loaded between 4 and 20 webpages. In one database (G–wrt54g) the capture was run explicitly until 5000 packets had been received (representing the high end of data sampled). The results generated were not significantly better than those databases where the packet captures were stopped in an ad-hoc manner using less data.

The implications of these considerations is that the prints currently created are not strictly representative of clients that are already associated to a network. These prints best represent the behavior of clients around a small window of time centered on them associating to a network. Though this period of time is not very packet-intensive, a lot of important information is gleaned from the duration values contained in the management frames that are exchanged. When implementing this technique in the wild the best thing to do is probably only examine packets exchanged within a window around client association. Merely sampling packets once association has happened will not yield as diverse results.

Appendix C

Tool Usage

While implementing the algorithms outlined in the chapter 4, three important tools were created, duration-print-generator, duration-print-matcher, and duration-print-grader. This section gives an example of using these tools, as well as how they work. duration-print-generator simply takes in an input pcap and a MAC address, computes all the values outlined in the previous chapters, and writes them out to disk (a .prnt file) duration-print-matcher takes an input pcap, MAC address to fingerprint, and a set of previously computed prints (the print database). It then computes the print for the input pcap and finds the closest match. The following table shows the output of an example duration-print-matcher run. In this case duration-print-matcher is attempting to determine what implementation best maps to the card with the MAC address 00:0a:95:f3:2f:ab in the 5-1-lexie.pcap, against all of the saved prints in the print-db/lexie directory. The filename 5-1-lexie indicates that this pcap is the first sample from implementation-id 5. duration-print-matcher mis-identifies this pcap, as the correct implementation is not at the top of the list.

```
./duration-print-matcher -a 00:0A:95:F3:2F:AB -p
./print-db/lexie/pcaps/5-1-lexie.pcap -P ./print-db/lexie/
```

rank	**score**	**ID**	**model**	**chipset**	**driver**
0	79.03	10	Broadcom-MiniPCI	BCM4318	bcmwl5.sys
1	78.91	5	Apple-Airport Extreme	BCM4318	AppleAirport2.kext
2	73.51	6	Zonet-ZEW1520	BCM4306	bcmwl5.sys
3	56.03	7	Intel-IPW220BG	IPW2200BG	w29n51.sys
4	54.74	13	Cisco-Aironet-350	Prism2	pcx500.sys
5	53.06	11	Sony-PSP	unknown	unknown
6	47.19	8	D-Link-dwl-g122	RA2570	rt2500usb.sys
7	39.95	4	Proxim-Orinoco Silver	AR5212	ntpr11ag.sys
8	39.55	3	Proxim-Orinoco Silver	AR5211	ntpr11ag.sys
9	39.47	2	Proxim-Orinoco Silver	AR5212	ntpr11ag.sys
10	38.53	1	Linksys-WPC55AG	AR5212	ar5211.sys
11	28.55	12	Nintendo-DS	unknown	unknown
12	22.61	9	SMC-2532W-B	Prism2.5	smc2532w.sys

Table C.1: Sample output from duration-print-matcher

C.1 Duration-Print-Grader

duration-print-grader performs the same analysis as duration-print-matcher, however it does it on a much larger scale. Every implementation included in a database had 3 different captures taken of it associating to a network. duration-print-grader takes all these pcaps, attempts to match them to the database of prints on disk, and keeps track of the amount of error in terms of distance down the sorted list the correct print for that pcap is found. A table representing the output of duration-print-grader is below C.2.

ID	**s1**	**s2**	**s3**	**chipset, driver**	**success rate**
1	0	0	0	Atheros AR5212 ar5211.sys	39/39 (1.000)
2	0	0	1	Atheros AR5212 ntpr11ag.sys	38/ 39 (0.974)
3	0	0	2	Atheros AR5211 ntpr11ag.sys	37/ 39 (0.949)
4	2	0	0	Atheros AR5212 ntpr11ag.sys	37/ 39 (0.949)
5	1	1	0	Broadcom BCM4318 AppleAirport2.kext	37/ 39 (0.949)
6	0	0	0	Broadcom BCM4306 bcmwl5.sys	39/ 39 (1.000)
7	0	0	0	Intel IPW2200BG w29n51.sys	39/ 39 (1.000)
8	0	0	0	RaLink RA2570 rt2500usb.sys	39/ 39 (1.000)
9	0	0	0	Intersil Prism2.5 smc2532w.sys	39/ 39 (1.000)
10	0	0	0	Broadcom BCM4318 bcmwl5.sys	39/ 39 (1.000)
11	0	0	0	unknown unknown unknown	39/ 39 (1.000)
12	0	0	0	unknown unknown unknown	39/ 39 (1.000)
13	0	0	0	Intersil Prism2 pcx500.sys	39/ 39 (1.000)

Table C.2: output from: ./duration-print-grader -P ./print-db/lexie/

–num errors across DB: 7 success rate across DB: 12.820513 / 13 = 0.9862

Samples s1,s2,s3 refer to the three sample pcaps for a given implementation. The first sample in the row for ID 5 corresponds to the previous example from duration-print-matcher. This has value of 1 because the correct print was 1 deep in the list for sample 1.

The column on the right is the success rate of the specified algorithm for a single implementation. It is computed using equation 5.1 which can be expressed as:

$accuracy = \frac{(num\_implementations\_in\_db)*(num\_samples)-misplacement\_distances}{(num\_implementations\_in\_db)*(num\_samples)}$

Where misplacement distance is the sum of the ranks for the three samples For instance, for the airport extreme (implementation ID #5) we get the following accuracy:

$\frac{(13*3)-2}{13*3} = \frac{37}{39} = 0.949$

Chapter 6 covers the details, but by taking the weighted average of these individual success rates where the rate is the likelihood of seeing an implementation, we can compute a success rate across the entire database. When using duration-print-grader the likelihood of seeing an implementation is constant, and the individual success rates are all weighted equally. In the example above, the success rate across the database turns out to be 12.805555 / 13 = 0.9850.

Appendix D

Comprehensive Device Driver Information

The following table details all of the 802.11 implementations tested in this study. Every implementation excluding the Apple Airport Extreme was test on Windows XP SP2. The Airport card was tested on OSX 10.4.

ID	**MAC, model, chipset files**	**files**	**details**
1	00:12:17:79:1C:B0 Linksys WPC55AG v1.2 Atheros AR5212	ar5211.sys	Driver Date: 7/12/2004 Provider: Atheros Communications Inc/Linksys*. File version 3.3.0.1561 Copyright 2001-2004 Atheros Communications, Inc. Signed: Microsoft Windows Hardware Compatability
2	00:20:A6:4C:D9:4A Proxim Orinoco Silver 8481-WD Atheros AR5212	ntpr11ag.sys	Driver Date: 8/5/2004 Provider: Atheros Communications Inc. File version 3.1.2.219 Copyright 2001-2004 Atheros Communications, Inc. Signed: Microsoft Windows Hardware Compatability

3	00:20:A6:4B:DD:85 Proxim Orinoco Silver 8461-05 Atheros AR5211	same as above	
4	00:20:A6:51:EC:09 Proxim Orinoco Silver 8471-WD Atheros AR5212	same as above	
5	00:0A:95:F3:2F:AB Apple AirPort Extreme Broadcom BCM4318	AppleAirport2-bcm4318	Version: 404.2
6	00:14:a5:06:8F:E6 Zonet ZEW1520 Broadcom BCM-4306	BCMWL5.sys	Driver Date: 1/23/2004 Provider: Broadcom. File version 3.50.21.10 Copyright 1998-2003 Broadcom Corporation. Signed: Microsoft Windows Hardware Compatability
7	00:0E:35:E9:C9:5B Intel PRO/Wireless 2200BG	w29n51.sys	W29NCPA.dll W29MLRes.dl Driver Date: 9/12/2005 Provider: intel File Version: 9003-9 Driver Copyright: Intel 2004 Signed: Microsoft Windows Hardware Compatability
8	00:13:46:E3:B4:2C D-Link dwl-g122 Ralink RA2570	rt2500usb.sys	Driver Date: 4/1/2004 Provider: D-Link/Ralink Driver Version: 1.0.0.0 Signed: Microsoft Windows Hardware Compatability
9	00:04:E2:80:2C:21 SMC 2532W-B Prism 2.5	smc2532w.sys	Driver Date: 10/20/2003 Provider: SMC Driver Version: 3.1.3.0 Copyright: 2003 SMC Networks, Inc. Signed: No.

10	00:14:A4:2A:9E:58 Broadcom 802.11g miniPCI BCM4318	bcmwl5.sys	Driver Date: 12/22/2004 Provider: Broadcom Driver Version: 3.100.46.0 Copyright: 1998-2004, Broadcom Corporation. Signed: Microsoft Windows Hardware Compatability
11	00:14:A4:7f:84:67 Sony PSP	unknown	PSP firmware version 2.50
12	00:09:BF:9D:59:C9 Nintendo DS	unknown	NA
13	00:0D:29:02:44:B8 Cisco aironet-350	pcx500.sys	Driver Provider: Microsoft Driver Date: 7/1/2001 Driver Version: 7.29.0.0 Digital Signer: Microsoft Windows Publisher
14	00:0E:35:E9:C9:5B Intel PRO/Wireless 2200BG	w29n51.sys Netw2c32.dll Netw2r32.dll	Driver Date: 6/26/2006 Provider: intel File Version: 9.0.4.17 Copyright: Intel 2004 Signed: Microsoft Windows Hardware Compatibility

Bibliography

[1] IEEE Wireless LAN Edition. A compilation based on IEEE Std 802.11-1999 (R2003) and its amendments, IEEE Press, 2003.

[2] IEEE std. 802.11, standards for local and metropolitan area networks. 1999.

[3] WiFi (Wireless Fidelity): http://www.wi-fi.org