Realizing an 802.11-based Covert *Timing* Channel Using *Off-The-Shelf* Wireless Cards

Sakthi V. Radhakrishnan, A. Selcuk Uluagac and Raheem Beyah
CAP Group, School of ECE, Georgia Institute of Technology
Atlanta, GA 30363, USA
{sakthi03, selcuk}@gatech.edu, rbeyah@ece.gatech.edu

Abstract—**By using covert channels, a malicious entity can hide messages within regular traffic and can thereby circumvent security mechanisms. This same method of obfuscation can be used by legitimate users to transmit messages over hostile networks. A promising area for covert channels is wireless networks employing carrier sense multiple access with collision avoidance (CSMA/CA) (e.g., 802.11 networks). These schemes introduce randomness in the network that provides good cover for a covert timing channel. Hence, by exploiting the random back-off in distributed coordination function (DCF) of 802.11, we realize a relatively high bandwidth covert timing channel for 802.11 networks, called Covert-DCF. As opposed to many works in the literature focusing on theory and simulations, Covert-DCF is the first fully implemented covert timing channel for 802.11 MAC using off-the-self wireless cards. In this paper, we introduce the design and implementation of Covert-DCF that is transparent to the users of the shared medium. We also evaluate the performance of Covert-DCF and provide discussions on the feasibility of this technique in a real world scenario.**

Index Terms—**Covert Channels, Covert DCF, Network Steganography, 802.11 Covert Channel, Covert Timing Channel**

I. Introduction

Covert channels are communication channels between a sender and one or more receivers which is generally used for secretive data exchange. This is in contrast to overt communication, where the data is sent over legitimate channels. Using a covert channel, a malicious entity can hide messages and other useful information (e.g., PIN, password) within regular traffic which poses a great security threat.

The idea of covert communication channels has been around as early as the 1970s [1]. Traditionally, *covert communication channels* take two forms: (1) *storage* channels and (2) *timing* channels. With storage channels, the secret data is directly or indirectly embedded into a storage medium; covertness is ”distinguished based on what is sent” [2]. For instance, the secret data can be transmitted by exploiting the unused fields of communication protocol headers [3], or by using packets of different lengths [3]. On the other hand, timing channels use timing patterns or modulate the communication resources over time to hide messages within regular communication. The receiver, then observes these patterns over time to recover the secret data.

This paper presents a covert *timing* channel implemented over the 802.11 MAC protocol. Layer 2 Covert channels is an area of research that is investigated mostly via simulations or using theory. Theory and simulations provide researchers an excellent opportunity to learn the necessary details to accomplish a covert channel and replicate the results. However, when evaluating timing channels, which significantly depend on temporal environmental conditions, simulations and theory provide a limited view into the actual performance of such a channel. For instance, performance issues under collisions, hardware fluctuations, interactions with other applications and protocols on a real testbed or a device would better be observed when a covert channel is actually implemented. Therefore, practical realizations (implementations) allow researchers to better evaluate the feasibility of a proposed covert channel.

On the other hand, studying covert channels at the MAC layer via simulations and theory had been the best option, since wireless drivers were closed source. However, in recent years this has been changing. The number of companies that offer open source versions of their wireless drivers are quickly increasing. Today, almost all major wireless network interface card manufactures, provide open source versions of their wireless drivers.

Leveraging the proliferation of various open source wireless drivers available, in this paper we realize a novel covert timing channel for the 802.11 MAC protocol that employs carrier sense multiple access with collision avoidance (CSMA/CA). Our timing channel is called *Covert-DCF* and its theoretical foundation was originally proposed in [4]. Randomness in events provide a good opportunity of covert channels. In our case, we exploit the random back-off mechanism that exists in the distributed coordination function (DCF) of 802.11 to implement a covert channel. However, in this paper, as opposed to many works in the literature focusing on theory and simulations, we actually realize this covert timing channel. To the best of our knowledge, Covert-DCF is the first fully implemented covert *timing* channel for 802.11 MAC protocol. In this paper, we introduce the design and implementation of Covert-DCF that is transparent to the users of the shared medium. We provide a detailed discussion of the implementation procedures and obstacles faced in realizing this channel. We also evaluate the performance of Covert-DCF on real hardware (i.e., laptops) and include discussions on why this technique would work even under contention from other wireless stations (STAs).

The rest of the paper is organized as follows. The related work is discussed in the next section. Section III provides an overview of the 802.11 MAC protocol. In Section IV, we discuss the communication and threat model and give a brief overview of Cover-DCF. In Section V we articulate the details of our implementation. We evaluate the performance of Covert-DCF on real systems in Section VI. Finally, Section VII concludes the paper and gives the future work.

II. Related Work

The term *covert channel* was first introduced by Lampson [1] under the context of program confinement which tries to ensure data privacy between processes. Although initial studies on covert channels concentrated on secretive data communication inside multi-level secure (MLS) systems, it soon spread into computer networks and related areas.

Covert channels in computer networks can be broadly classified as either *timing* or *storage* channels. Channels that make use of event timings are classified as covert timing channels and channels that modify some form of storage objects (such as header fields) are classified as covert storage channels.

Storage channels have been proposed in several works in the literature and most of these techniques exploit unused header fields which make them protocol dependent. For example, the technique proposed in [5] works only over ICMP ping request, whereas Covert-DCF works over any traffic stream. Moreover, some of the networks re-normalize unused header fields [6], which in turn will damage covert storage channels that make use of such fields. In our case, this wouldn't be a problem because Covert-DCF is a timing channel, and thus does not use any form of header manipulation.

Though covert timing channels have the advantage of being protocol independent, and being immune to header field re-normalization, they have their own disadvantages. For example, the works discussed in [7]–[9] require that the sender and receiver are in sync with one another. Such synchronizations are not necessary for our timing channel implementation, which is based off one of our previous works [4]. In [4] we use theory and simulations as proof of concept, however, with simulations it is not possible to understand the feasibility or practicality of a covert channel. Though simulations can account for time fluctuations introduced by network and MAC layers, they do not consider the timing noise injected by hardware fluctuations which arise due to variation in processor load and implementation limits [10]. Studying the influence of these factors require a real-time implementation and this served as the primary motivation for this work.

Uniqueness: This work discusses in detail the real-time implementation of an 802.11 MAC protocol based timing channel and practical problems faced in doing so. To the best of our knowledge, there are very few covert timing channels based on layer 2 protocols [8] [11] [12] [13], and none of them provide real-time implementation. They treat the matter theoretically and use simulations for their proposed algorithms. This makes Covert-DCF, the first real-time implementation of a layer 2 timing channel utilizing random back-offs in 802.11 networks.

III. Background: Overview of 802.11 Protocol

We realize Covert-DCF by exploiting the random back-off values in the 802.11 MAC protocol in DCF mode of operation. Hence, in this section, we briefly review the necessary details of the DCF mode of operation.

The Distributed Coordination Function (DCF) uses a listen-before-talk scheme called CSMA/CA. This scheme makes every node in the medium responsible for channel sharing, retransmission and collision avoidance. The major parameters involved in the DCF mode of operation are, Contention Window (CW), DCF Inter Frame Space (DIFS), and Small Inter Frame Space (SIFS), shown in Figure 1. When a node has data to transmit, it senses the channel to check if the channel is busy. If the channel is free, then the sending node waits for DIFS amount of time before transmitting the data. If the channel becomes busy during this period, the sending node choses a value between 0 and CW_{min}, and backs off for that period of time. If the channel becomes busy during this period, the node freezes the back-off countdown timer, and continues to count down once the channel becomes free. If the node counts down to zero while the channel is free, it possesses the channel and transmits the data packet. If the ACK associated with this transmission is not received, the sender assumes a collision and doubles the upper limit on the CW and repeats the entire process again.

IV. Communication & Threat Model

Covert-DCF is a covert timing channel which is implemented over the IEEE 802.11 MAC protocol. In this section, we explain the communication and threat model used in Covert-DCF.

Our model is based on the traditional model for covert communications, the *prisoner problem* [14], [15]. In this model as illustrated in Figure 2, there is a *covert sender* (Alice), a *covert receiver* (Bob), a *warden* (Wendy), and a shared secret. The prisoners Alice and Bob try to communicate an escape plan secretly in spite of the warden.

In our work, Bob and Alice use the random back-off process in the 802.11 DCF to secretly convey messages to each other. Before Alice and Bob can communicate using this covert channel, they need to share a code book. This code book includes the list of back-off values that the covert sender will use and the corresponding symbol associated with it.

Assume that Alice wants to send the word "CAB" over the covert channel to Bob as seen in Figure 3. She first chooses the back-off period associated with symbol 'C', waits for the backs-off amount of time according to the procedures explained in the previous section, sends a data packet, and then does the same for 'A' followed by 'B'. On the other end, Bob observes the back-off values used by Alice by monitoring the wireless traffic. Once Bob has the back-off values, he compares it with the back-off values in the code book to recover the sent message "CAB". The *warden* in our model

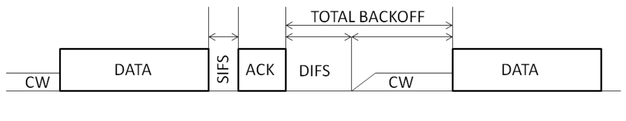


Fig. 1. Packets in the DCF mode of access in IEEE 802.11

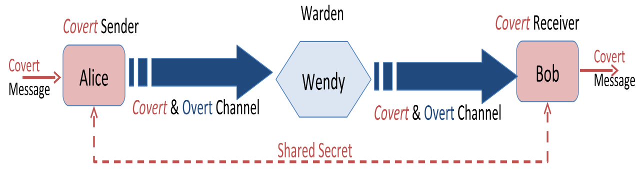


Fig. 2. Communication & Threat Model [14]

represents the person who would try to detect or block this information exchange. Examples of warden include wireless intrusion detection and prevention systems (WIDPS) such as Motorola AirDefense [16], or systems for detecting greedy behavior in wifi networks, such as Domino [17].

V. Implementation using Commodity Hardware

In this section, we discuss the details of the implementation of Covert-DCF. First, we introduce the hardware and software architecture on which Covert-DCF was realized. Then, we articulate the necessary operations involved at the covert sender and receiver.

A. *Hardware and Software Architecture*

A prototype implementation of a covert timing channel like Covert-DCF could be realized using FPGAs, GNU Radios, and off-the-shelf wireless network interface cards available in laptops. Although the flexibility is higher with the first two options, we opted to implement Covert-DCF using off-the-shelf wireless cards. The primary objective behind this was to evaluate the feasibility and practicality of Covert-DCF with commodity chiptsets which is available in most laptops today.

For our implementation, we chose the Atheros AR5212 chipset based commodity 802.11 a/b/g wireless hardware. This chipset provides a great degree of software control over several aspects of the wireless card and the drivers for the card are open source. As seen in Figure 4, most of the 802.11 functionalities are defined outside the firmware and the hardware access is regulated using a Hardware Abstraction Layer (HAL). In our implementation, we used the Multiband Atheros Driver for Wireless Fidelity (Madwifi) trunk version [18] as the driver for the chipset. The reader is referred to [19] for information about other open source wireless drivers available for IEEE 802.11 network cards.

B. *Covert-DCF Sender Operations*

In order to implement Cover-DCF, the covert sender first needs to convert the symbols to back-off values as explained in Section IV. These values are then used as back-offs during data transmission. In this sub-section, we discuss how this was achieved at the covert sender.

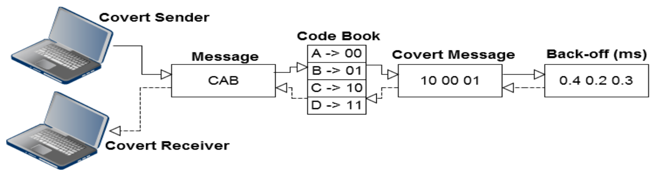


Fig. 3. Covert-DCF Conceptual View

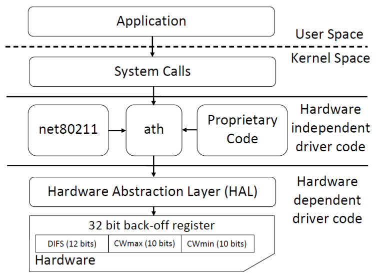


Fig. 4. Hardware & Software Architecture

1) Varying the Total Back-off: In order to send out messages using Covert-DCF, the sender needs to vary the back-off values i.e., the total back-off time interval between an ACK and the next Data packet (see Figure 1). Since the code for back-off and other critical tasks reside in the firmware, we manipulated the values of the hardware registers in the wireless card to obtain control over the back-off values, on a per packet basis. Note that the Madwifi wireless driver uses a single 32bit register to store the values of CW_{min} (10bits), CW_{max} (10bits) and DIFS (12bits) (see Figure 4). The initial values and the details of the register addresses are located in *ar5xxxreg.h*. To manipulate the registers, one can either use simple C functions such as *writel(value, register_address)* or just make use of built-in functions such as *OS_REG_WRITE(ath_hal, address, value)* and *OS_REG_READ(ath_hal, address, value)*. Also, the Madwifi driver that we used had a few built-in proprietary features [20] under the name of *Super A/G* to increase throughput and range of compatible devices. Among the set of features, *BURST* and *FAST-FRAME* hindered the back-off process and we had to disable them in order to get complete control over the back-off.

2) Varying DIFS vs. Contention Window (CW): The ultimate goal of the covert sender is to change the back-off time between when the channel becomes idle (which in most case is after the last ACK packet) and the transmission of the next data packet. As can be seen in Figure 1, one can accomplish this by varying either the DIFS or the congestion window. These are discussed as different cases below.

Case1-Varying CW: In this case, the covert sender was configured to transmit decimal values from 0 to 15 in a loop, and the value of the CW was varied as explained above to reflect these values in the back-off. Each decimal value was transmitted with 15 time slots as buffer in order to reduce erroneous decoding at the receiver. Thus, for a set of 16 symbols, a total of 225 slots were used (symbol 0 uses up no slots). Figure 5(a) represents the back-off values of the covert sender as seen by the covert receiver for this case.

Case2-Varying DIFS: In this case, instead of using CW, the DIFS was used to reflect the symbol that had to be transmitted. Figure 5(b) shows the back-off values that were recorded at the covert receiver for this case. Note that the value of CW was fixed at 0 slots during this experimentation.

From these figures, we see that changing DIFS shows a more stable back-off variation than using CW. The primary

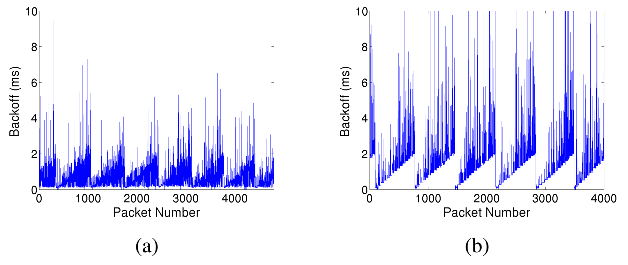


Fig. 5. (a) Case 1: Varying CW; (b) Case 2: Varying DIFS

reason for this is because, unlike DIFS, the value of CW is not always used for back-off during a packet transmission. For example, if a node wants to send a packet and it senses the channel to be free, then it waits for DIFS amount of time and sends out the packets (considering the channel is still free). Hence, the value of CW is never used in such a scenario. As explained in Section III, the value of CW is relevant only when the channel is sensed to be busy or when the node experiences a packet loss. This gives rise to inconsistent total back-off behavior when the value of CW is manipulated. However, since DIFS is always used, the back-off behavior is more consistent as shown in Figure 5(b). One should also note that we were not able to observe the same effect relevant to varying DIFS vs. CW in our earlier work [4], which treated the concept of Covert-DCF via simulations solely.

C. Covert-DCF Receiver Operations

In Covert-DCF, the receiver first monitors the wireless channel, and then filters the back-offs of the covert sender to extract the secret message. In this sub-section, we discuss these operations in further detail.

1) Monitoring the Covert Sender's Back-offs: The primary task at the receivers' side is to monitor the back-off values of the sender. The first step to do this is to put the wireless interface into *monitor* mode. In this mode, a wireless card will capture all packets that are sent over the air (even if they are not addresses to it). Although it is trivial to put a wireless device into promiscuous mode, it should be noted that this process requires a working combination of wireless chipset, packet capturing software, and wireless device driver. For our experimentation, we used, an *AR5212* based wireless chipset, *tcpdump* for packet capturing, and Madwifi as the wireless driver, over Ubuntu 11.04 (linux kernel 2.6.38). Using such a setup, the covert receiver records all the traffic that it hears over the air. This capture is then passed through a Python script which detects and computes the back-off values used by the covert sender. In order to do this, our script identifies packets sent out by the covert sender and computes the back-off used, by simply subtracting the time of the previously seen ACK packet from the time when the data packet was sent (Figure 1).

2) Filtering: In order to improve reliability, we introduce redundant transmissions at the sender (each symbol is sent x number of times) which is explained in Section VI-C. Once the back-off values are collected, the covert receiver applies a series of basic processing techniques to decode the covert message. The basic steps involved in this process are *grouping*, *quantizing*, and *smoothing*. In the grouping phase, *s_size* number of packets are taken for every iteration and the minimum value of back-off in that sample is found and stored. This is analogous to sampling, however in this case, a number of packets are used to decide on a symbol value instead of a single sampled packet. The grouping process essentially yields back-off values that look similar to a pulse amplitude modulated signal that is distorted due to channel characteristics. Before translating the back-off values into symbols, these values are quantized or rounded to the nearest value of back-off available on the code book. The values obtained at the end of quantization are further smoothened using Algorithm 1, to reduce errors caused by timing fluctuations in the back-off values (noticeable in Figure 5(b)). Once smoothened, a simple lookup is used to decode the message.

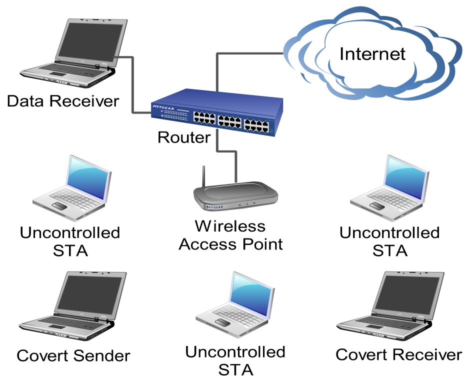


Fig. 6. Covert-DCF test topology

Algorithm 1 Smoothing

```
1: $fq = []$; $q \leftarrow quantized\ data$
2: for $(i = 1 \rightarrow len(q - 1$ step $2)$ do
3:     if $(q(i) == q(i + 1)$ then
4:         $fq.append(q(i))$
5:     end if
6: end for
7: return $fq$
```

VI. Performance Evaluation

In this section, we evaluate the performance of Covert-DCF on a real systems. We begin with a feasibility analysis of Covert-DCF and move on to discuss methods to increase the throughput of our technique. Finally, we introduce a generic analytical expression incorporating various tunable parameters, to help predict and tune the throughput of Covert-DCF.

A. Covert-DCF Test Setup

Figure 6 shows the network topology that was used to test CovertDCF. The testbed comprises of a 802.3 backbone router (NETGEAR FVS338), a wireless access point (AP) (ZyXEL G-570S), covert sender and receiver laptops (Lenovo C100) equipped with Cisco Aironet PCMCIA cards, running Ubuntu 11.04 (kernel 2.6.38), and a data receiver. In this setup, the covert sender and the receiver are connected to the same wireless network and the sender's application produces a UDP stream addressed to the data receiver. It should be noted that our tests were performed in the presence of other uncontrolled wireless users in the background. The discussion

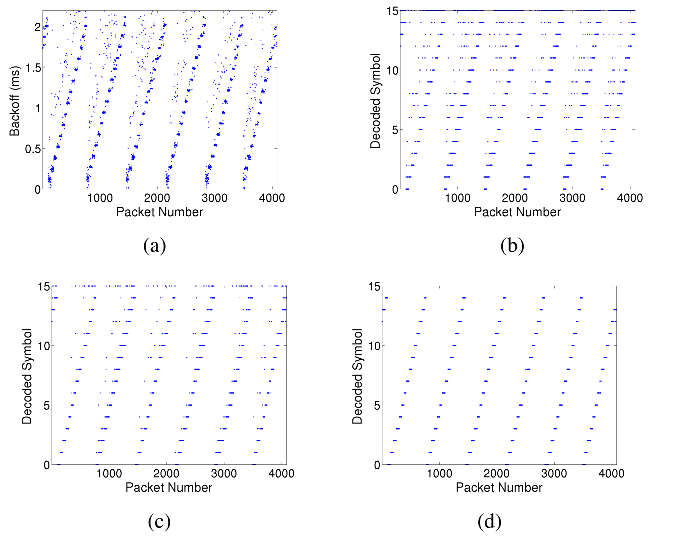


Fig. 7. (a) Captured back-off values; (b) Quantized back-off values; (c) *min()* over 2 consecutive packets; (d) *min()* over 6 consecutive packets

in Section VI-E explains why our technique would work even when there is contention from other wireless stations.

B. Feasibility of Covert-DCF

As explained in Section V, the covert receiver goes through monitoring and filtering operations to extract the symbols sent by the covert sender. In this sub-section, we evaluate the feasibility/performance of Covert-DCF by presenting results obtained through real-time experimentations.

In our test, the code-book consists of decimal values from 0 to 15 and each symbol is represented by 4 bits. Each symbol is associated with a back-off value (i.e., number of time slots). The covert sender transmits all the code book values in a loop with 15 time slots per symbol. The maximum number of slots that can be assigned to DIFS back-off is 255. Hence, a total of 255 slots were split into 16 equal segments (each representing one of the 16 symbols).

Figure 7(a) shows the captured back-off values and Figure 7(b) shows the quantized and decoded values. The x-axis is the packet number in both the figures while the y-axis represents back-off in Figure 7(a) and decoded symbol in Figure 7(b). As can be observed in these figures, without the quantization process, it is hard to extract the symbols. After quantization, the symbols become clear, however, it still contains many errors resulting from fluctuations in the observed back-off. These errors can be reduced by taking the *min()* over s_size consecutive recorded back-off values. This works reasonably well, because the timing fluctuations in the observed back-off values are biased towards values that are higher than the intended back-off (Figure 5(b)). Figure 7(c) and 7(d) presents the decoded values for two different values of s_size 2 and 6. As can be seen, the errors in the decoded values decreases as the size of s_size is increased.

C. Practical Throughput Improvement

In this sub-section, we discuss how the performance of Covert-DCF can be fine-tuned. Specifically, we are interested in configuring Covert-DCF to achieve a higher throughput. One potential way to increase the throughput is to increase the number of bits per symbol. However, as we analyze in this sub-section, decreasing the number of bits per symbol has a positive impact in our configuration as opposed to what would be expected.

In the configuration discussed in Section VI-B, 255 slots were used in total to represent 16 different symbols with 4 bits per symbol configuration. Considering that all 16 symbols are equally likely, the average back-off required to transmit one symbol will be 112.5 slots. This implies 28.125 slots per bit. If we reduce the bits per symbol to 2, we would just split the 255 slots into 4 segments. This will increase the buffer space between the symbols, but its impact on reliability will not be significant because the magnitude of timing fluctuations in the back-off values are too large to be handled by any reasonable buffer size. Therefore, we can consider a 2 bits per symbol configuration where we can reduce the total number of back-off slots utilized to 50 and fix 10 slot for each symbol. Since 4 symbols are possible (2 bits per symbol), this new configuration would occupy 40 out of the 50 slots. The remaining 10 slots is used as a clearance from the bottom so that we do not employ a low back-off (possibly lesser than DIFS), which can trigger an alert. Given this configuration, the average back-off per symbol would be 25 slots which implies a 12.5 slot time per bit.

In Figure 8(a), we compare a lower backoff-per-symbol (0.6 ms) configuration with a higher backoff-per-symbol (1.45 ms) configuration. As seen in this figure, 50 packets with the lower backoff-per-symbol configuration are sent faster than the configuration with higher backoff-per-symbol. This shows that, by decreasing the back-off associated with a symbol one could increase the throughput of Covert-DCF. A more detailed study on the throughput behavior of Covert-DCF is provided in Section VI-F.

Figure 8(b) shows the back-off values that were observed

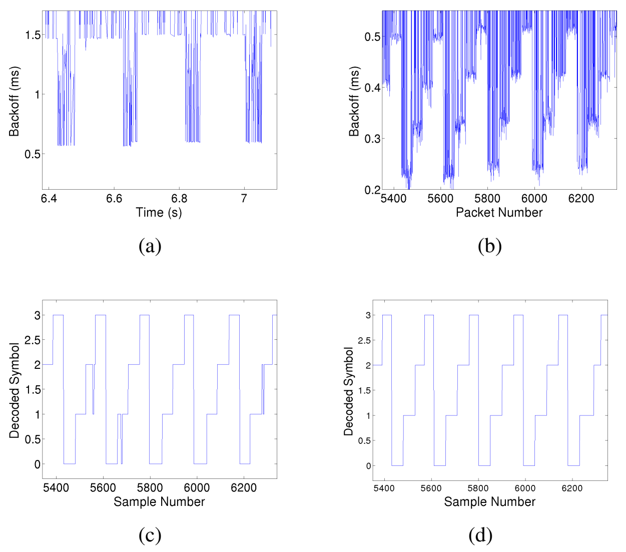
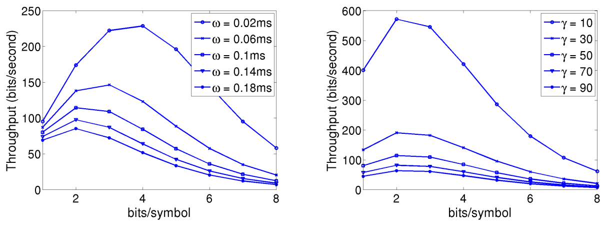


Fig. 8. (a) Variation in packet rate with value of back-off; (b) Plot of back-off values for 2 bits per symbol configuration; (c) After applying Quantization to back-off values; (d) After smoothing the back-off values (Algorithm 1)



(a) Guard Band and bits/symbol (b) Redundancy and bits/symbol

Fig. 9. Variation in throughput with respect to various Parameters. Guard Band(ω) = 0.1ms, Redundancy(γ) = 50, δ = 0.2ms

at the covert receiver, when we used the aforementioned 2 bits per symbol configuration. Due to the 2 slot clearance that was provided at the sender, the back-off value of the symbols start from 0.2 ms and increase by 0.1 ms per symbol. The observed back-off values were then passed through the grouping and quantization processes explained in Section V, to get the decoded symbols. Although the decoding process was able to recover the correct symbols as seen in Figure 8(c), one can still observe small errors in the decoded symbols. These errors are removed using the smoothening algorithm (Algorithm 1), which results in perfect decoding, as shown in Figure 8(d).

Identifying the throughput increase with the 2 bits per symbol configuration, we continue to use the same configuration for evaluating the performance of Covert-DCF. With this configuration, the maximum throughput achieved by Covert-DCF was 2.28Kbps. However, in order to create an effective covert channel with proper reliability, redundancy (50 packets per symbol) was introduced. Hence, the usable throughput decreased to 46bps. On the other hand, in an earlier work [4], the performance of which was only evaluated via simulations in OPNET, reported a data rate of 5Kbps and 1.89Kbps for two different configurations. So, the actual realization of Covert-DCF in this paper shows that real experimentation with real hardware achieves only half of the simulated throughput. And, the usable is much less than the half. Therefore, practical implementations in this work allowed us to better evaluate the feasibility of our earlier proposed covert communication channel. Our future work will include exploring different methods to increase the achievable throughput.

D. Security Issues

Covert-DCF is secure against most of the security framework that are in place to detect malicious activities over legitimate channels. However, wireless intrusion detection and prevention systems (WIDPS) such as Motorola AirDefence Enterprise [16], or systems for detecting greedy behavior in wifi networks, such as DOMINO [17] can counter our technique. For instance, consider Motorola AirDefense Enterprise [16], which is a popular commercial WIDPS solution. This system monitors the average back-off values of all the wireless nodes and reports any irregularities such as jamming. In order to address this concern, carefully chosen traffic fitting symbols can be inserted in between consecutive data symbols similar to [4]. In such an approach, the values of traffic fitting symbols can be chosen such that the average value of back-off for the covert sender matches closely with that of the overt senders. By doing this, the covert sender can fully camouflage the timing channel. However in order to do this, a covert sender needs to constantly monitor the average back-off values of overt nodes in its WLAN. We are planning to adapt this traffic fitting approach in our future work.

E. Contention Issues

Since Covert-DCF is a timing channel, MAC layer contention can be seen as a possible hindrance to its operation. For example, if Covert-DCF used CW to manipulate back-off, then the assigned value of back-off might not count down to zero in a single stretch. This is because, the back-off timer associated with CW freezes whenever the channel becomes busy, and counts down only when the channel is sensed to be free. This results in shorter 'observed' back-off values than what was intended. However, our implementation uses DIFS, instead of CW, to manipulate the value of back-off. Unlike CW, DIFS has to count down to zero in a single stretch, which ensures that the observed value of back-off is always equal to or greater than the intended value of back-off, even when there is contention from other wireless stations in the network. This can be processed to decode the message as explained in Section V-C.

F. Throughput Formulation

In this sub-section, we generalize the performance of Covert-DCF to aid in fine tuning the performance of our technique. Specifically, we are interested in providing an analytical expression for predicting the throughput of our covert channel.

The throughput, ψ, of Covert-DCF can be expressed as follows:

$$\psi = \frac{\beta}{\{\sum_{i=0}^{2^{\beta}-1}(\frac{\omega}{2^{\beta}} * i + \delta)\} * \gamma} \quad (1)$$

where

- β - Number of bits represented by a symbol.
- γ - Redundancy for simple reliability. It is the number of times a symbol is re-transmitted over the covert channel.
- ω - Timing separation between consecutive symbols (Guard Band).
- δ - Minimum back-off time over which the manipulated back-off variations happen.

Without loss of generality, assuming a similar environment for the covert sender and the covert receiver, we show in Figure 9 the throughput performance of Covert-DCF for variations in a number of configurable parameters. The x-axis represents β while y-axis is the maximum achievable throughput.

The general observation that can be made from Figure 9 is that throughput decreases with increase in the guard band and redundancy. However, one needs increased guard band and redundancy for better reliability. Therefore, these parameters need to be configured with care.

Moreover, the line representing a guard band of 0.1ms in Figure 9(a) represents the parameters used in our implementation. It is interesting to observe that the throughput shown in

Figure 9(a) at 2 bits/symbol and guard band of 0.1ms is higher in comparison to what was achieved in our implementation in this paper. This is due to the fact that Equation 1 does not account for a) Data and ACK packet transmission times; b) Processing time at the access point before it responds with an ACK. However, these additional timing overheads are encountered in a real-time scenario, which thereby results in a reduced throughput.

Another important observation that can be made from Figure 9(a) is that throughput peaks at different values of bits/symbol for different values of guard band. Therefore, the number of bits/symbol (for a given value of guard band) must be properly chosen in order to achieve maximum throughput. For the value of guard band chosen in our implementation (which was 0.1ms), the peak rate is achieved at 2 and 3 bits/symbol, as shown in Figure 9(a). We chose 2 bits/symbol (in lieu of 3) so that it would provide an easier way to split bytes or characters into symbols. We also observe a drastic increase in throughput when the redundancy is decreased. Studying the variation in symbol error rate with variation in guard band and redundancy factor will help in further optimizing the throughput.

Finally, Figure 9(b) shows the negative impact of redundancy on throughput. Clearly, it can be observed that throughput decreases exponentially as redundancy increases. This is one of the major factors which decreases the achievable throughput. Therefore, a further investigation of guard band's and redundancy's contribution to reliability is needed. Moreover, the use of error correction codes in lieu of simple redundancy is another avenue of further exploration.

VII. Conclusion and Future Work

Covert communication channels can be utilized for both malicious purposes by hackers to pass valuable information (e.g., PIN, password) or for benign purposes by legitimate users to transmit messages over hostile networks.

In this paper, as opposed to many works in the literature focusing on theory and simulations, we actually realized a covert *timing* channel for 802.11 networks, called Covert-DCF, using *off-the-shelf wireless cards* available almost in all laptops. To the best of our knowledge, Covert-DCF is *the first fully implemented* covert timing channel for the 802.11 MAC protocol. We also provided an in-depth discussion of the design and implementation details of Covert-DCF, with clear explanations to important design/implementation decisions that were made.

Moreover, we evaluated the feasibility and practicality of Covert-DCF with commodity wifi chipsets available in almost all laptops. Due to hardware fluctuations at the sender and dropped packets at the receiver (at the kernel level), the actual realization of Covert-DCF in this paper showed that real experimentation with real laptops and real wireless cards could only achieve the half of the simulated throughput. And, the usable is much less than that. Hence, practical implementations allowed us to better judge the feasibility of the earlier proposed covert communication channel.

In future, we intend to a) explore different methods (e.g. error correction codes) to increase the data rate; b) study the impact of variation in parameters such as number of surrounding wireless users (channel contention), sending rate, etc, on the Bit Error Rate (BER); c) adopt the use of traffic fitting symbols; d) study the performance in multiple-sender single-receiver, and multiple-sender multiple-receiver scenarios; g) implement a tool which provides a friendly interface for operating our covert timing channel.

Acknowledgments

This work was partly supported by NSF Grant No. CAREER-CNS-0545667 844144.

References

[1] B. W. Lampson, "A note on the confinement problem," *Commun. ACM*, vol. 16, no. 10, pp. 613–615, Oct. 1973.

[2] C. Girling, "Covert channels in LAN's," *Software Engineering, IEEE Transactions on*, vol. SE-13, no. 2, pp. 292 – 296, feb. 1987.

[3] S. Zander, G. Armitage, and P. Branch, "A survey of covert channels and countermeasures in computer network protocols," *Communications Surveys Tutorials, IEEE*, vol. 9, no. 3, pp. 44 –57, quarter 2007.

[4] R. Holloway and R. Beyah, "Covert DCF: A DCF-based covert timing channel in 802.11 networks," in *2011 IEEE 8th International Conference on Mobile Adhoc and Sensor Systems (MASS)*, 2011, pp. 570–579.

[5] B. Ray and S. Mishra, "Secure and reliable covert channel," in *Cyber Security and Information Intellidence Research Workshop*, May 2008.

[6] A. Singh, A. L. M. Santos, O. Nordstrom, and C. Lu, "Stateless model for the prevention of malicious tunnels," in *International Journal of Computer and Applications*, vol. 28, no. 3, 2006.

[7] M. A. Padlipsky, D. W. Snow, and P. A. Karger, "Limitations of end-to-end encryption in secure computer networks," Aug 1978, tech. Rep. ESD-TR-78-158, Mitre Corporation.

[8] C. G. Girling, "Covert channels in LAN's," in *IEEE Transaction on Software Engineering*, vol. SE-13, no. 2, Feb 1987, pp. 292–296.

[9] S. Cabuk, C. E. Brodley, and C. Shields, "IP covert timing channels: Design and detection," in *11th ACM Conference on Computer and Communications Security (CCS)*, Oct 2004, pp. 178–187.

[10] G. Bianchi, A. Di Stefano, C. Giaconia, L. Scalia, G. Terrazzino, and I. Tinnirello, "Experimental assessment of the backoff behavior of commercial ieee 802.11b network cards," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, may 2007, pp. 1181 –1189.

[11] T. Handel and M. Sandford, "Hiding data in the OSI network model," in *1st International Workshop on Information Hiding*, 1966, pp. 23–38.

[12] S. Li and A. Ephremides, "A covert channel in MAC protocols based on splitting algorithms," in *Wireless Communications and Networking Conference (WCNC), IEEE International Conference on*, 2005, pp. 1168–1173.

[13] Z. Wang, J. Deng, and R. B. Lee, "Mutual anonumous communications: A new covert channel based on splitting tree MAC," in *INFOCOM 2007, 26th IEEE International Conference on Computer Communication*, 2007, pp. 2531–2535.

[14] "The prisoners' problem and the subliminal channel." in *Advances in Cryptology: Proceedings of CRYPTO '83*, 1983, pp. 51–67.

[15] T. Handel and M. Sandford, "Hiding data in the OSI network model," in *Information Hiding*, ser. Lecture Notes in Computer Science, R. Anderson, Ed. Springer Berlin / Heidelberg, 1996, vol. 1174, pp. 23–38.

[16] "Motorola airdefence enterprise," 2010, http://www.airdefence.net.

[17] M. Raya, J.-P. Hubaux, and I. Aad, "DOMINO: A system to detect greedy behavior in IEEE 802.11 hotspots," in *2nd International Conference on Mobile Systems, Applications, and Services*, jun 2004, pp. 84–97.

[18] Http://madwifi-project.org/svn/madwifi/trunk.

[19] M. Vipin and S. Srikanth, "Analysis of open source drivers for ieee 802.11 WLANs," in *Wireless Communication and Sensor Computing, 2010. ICWCSC 2010. International Conference on*, jan. 2010, pp. 1 –5.

[20] Http://madwifi-project.org/wiki/ChipsetFeatures/SuperAG.