Fall 11-22-2010

Covert DCF - A DCF-Based Covert Timing Channel In 802.11 Networks

Russell Holloway

COVERT DCF: A DCF-BASED COVERT TIMING CHANNEL IN 802.11 NETWORKS

by

RUSSELL D. HOLLOWAY

Under the Direction of Dr. Raheem Beyah

ABSTRACT

Covert channels are becoming more popular as security risks grow in networks. One area that is promising for covert channels is wireless networks, since many use a collision avoidance scheme such as carrier sense multiple access with collision avoidance (CSMA/CA). These schemes often introduce randomness in the network, which provides good cover for a covert timing channel. In this thesis, we use the 802.11 standard as an example to demonstrate a wireless covert channel. In particular, most 802.11 configurations use a distributed coordinated function (DCF) to assist in communications. This DCF uses a random backoff to avoid collisions, which provides the cover for our covert channel. Our timing channel provides great improvements on other recent covert channels in the field of throughput, while maintaining high accuracy. We are able to achieve throughput over *8000 bps* using Covert DCF, or by accepting a throughput of *1800 bps* we can achieve higher covertness and 99% accuracy as well.

INDEX WORDS: Covert channel, MAC misbehavior, Steganography, 802.11 DCF, Wireless LANs

COVERT DCF: A DCF-BASED COVERT TIMING CHANNEL IN 802.11 NETWORKS

by

RUSSELL D. HOLLOWAY

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in the College of Arts and Sciences

Georgia State University

2010

COVERT DCF: A DCF-BASED COVERT TIMING CHANNEL IN 802.11 NETWORKS

by

RUSSELL D. HOLLOWAY

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Raheem Beyah, for all of his guidance and support throughout this process.

I would also like to thank Dr. Anu Bourgeois and Dr. Yingshu Li.

Finally, I would like to thank my family and friends for all of their support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

In today's society, security is becoming increasingly important in our networks. Initially, security was not designed into many of the standards and protocols as the architects of the Internet and standards never expected security to be an issue. However, it has become clear that security is a major concern. We have seen techniques and security patches applied to existing technology, as well as seen new technology and systems introduced with at least some form of security included, at least at the most basic level. Unfortunately, no matter how much security we continue to add, there will always be people looking to bypass that security for various reasons.

One popular method for bypassing security protocols is the use of covert channels. By using covert communication channels, an individual can hide messages and other information within regular traffic, thus slipping by security protocols (in contrast to overt channels, where the message is plain and sent openly). Classically, covert channels are classified as either storage channels or timing channels [1]. Storage channels use some sort of storage medium to hide messages, and timing channels use timing patterns to hide messages within regular communication. In this thesis, we focus on covert timing channels.

Covert timing channels pose a great risk to security as they often bypass all security measures in place altogether. A common example is the Bell-La Padula model, which enforces high access control and security in government [2]. The access control is very secure in this model due to two important mandatory access control rules. However, by creating a covert channel, the access control can be bypassed altogether.

The timing channel we propose is an example of a wireless covert channel that allows for higher bandwidth covert communication in comparison to other covert channels. One use for this channel would be command and control of a wireless botnet. We have seen mobile device usage grow dramatically, and many of them now include 802.11 support. Unfortunately, this growth also leads to additional malicious activity such as the iPhone/Privacy.A attacks which steal personal data from jailbroken iPhones [3]. It is not hard to imagine botnets forming on 802.11 enabled devices in near future, and it is necessary to prepare ourselves and defenses for these new forms of malicious activity.

On the other hand, covert channels could provide an added layer of security. For example, they could be used to store additional information as part of an access control scheme as we demonstrated in [4]. While they should not be used as the sole method for access control since they merely aim to provide security through obscurity, they could provide an additional layer as part of an overall layered security scheme.

Our covert timing channel, Covert DCF, uses the 802.11 medium access control (MAC) contention window (CW) in the distributed coordinated function (DCF) to send information. The CW allows for random backoff to reduce the number of collisions on the wireless medium and also offers the variability required for a covert channel. A sender can intelligently select the CW for each outgoing frame [5], making it appear random over time. However, each choice of the CW can represent a symbol from a codebook, and through these symbols the covert message can be sent. Note, one instance of this (i.e., reducing the random backoff time) is considered mac misbehavior [5]. What we illustrate in this thesis, is that not only can a node improve throughput using mac misbehavior, but it can also send messages covertly.

The receiving node reverses the process. First, the receiving node will calculate the CW window used by the sender (based on gathered knowledge of the current network), and look up the corresponding symbol in the codebook.

This scheme can easily be extended to other distributed wireless networks, since by design they generally include some randomness to avoid collisions on the medium. This randomness is the source of our timing channel.

The rest of this thesis is organized as follows. In Chapter 2, we present related work. A brief overview of the 802.11 DCF is presented in Chapter 3, with an overview of our proposed covert channel following in Chapter 4. Chapter 5 aims at optimizing individual characteristics for our covert channel. Chapter 6 presents our covert channel in detail. We present our analysis and results in Chapter 7. Lastly, Chapter 8 presents our conclusions and we conclude with future work in Chapter 9.

Chapter 2

RELATED WORK

In order to fully implement a covert channel as effectively as possible, we must analyze other covert channel designs, information theory of such channels, and also detection and prevention methods of these channels.

There have been several designs of covert channels. Previous research has introduced a good number of storage channels, with more recent research covering a wider range of protocols and network layers as seen in [6–12]. Several covert timing channels have been introduced as well such as that in [13], where the authors introduce one of the earlier basic timing channels using IP inter-packet arrival time (IAT) patterns. This channel obtained approximately 17 bps. However, it is easy to detect this covert timing channel based on the regularity of the traffic it generates. Cabuk et al. provide an in-depth study of IP covert channel detection in [14].

Sellke et al. demonstrate a TCP timing channel which can also be created in such a way that it can hide among traffic that can be modeled by independent and identically distributed (i.i.d.) random variables in [15]. It requires the sender and receiver to agree upon a seed in advance, and the cumulative distribution function (cdf) to model must have an inverse. However, when the authors make their covert channel traffic computationally indistinguishable within polynomial time from regular traffic, there is a significant decrease in the throughput. They demonstrated a maximum of 84 bps throughput for traffic that is computationally distinguishable from legitimate traffic, or 5bps for the computationally indistiguishable scheme. While being computationally indistinguishable can help hide the covert message, the extremely low throughput detracts from the appeal of this method.

A few papers have also been written on 802.11 covert channel schemes as well. One method we presented in [16] uses rate switching as the covert channel but has a maximum throughput of 96 bps. Furthermore, due to the unreliability of UDP, the rate switching technique used can have a large effect on UDP traffic. A simple covert storage channel at the data link layer using the 802.11 sequence and WEP initial vector fields in the header was introduced in [17].

We can also look at information theory related to covert channels. There are several papers that discuss the capacity and bounds of timing channels. It is shown in [18] that channels modeled by a single-server queue have a capacity greater than the service rate of the queue. Channels with service time distributions which have bounded support are considered in [19], and [20] analyzes the effects of noise, such as time sharing delays of the CPU and I/O. Real-time systems are considered in [21], in which operations are performed at preemption points or within predetermined intervals.

Finally, we can also look at detection and prevention schemes. Because we are performing MAC misbehavior on our sending node, we need to avoid MAC misbehavior detection schemes. Rong et al. [22] presented a method for detecting MAC misbehavior by looking at throughput degradations observed at normal stations. Similar methods for calculating the *traffic gain ratio* and *traffic degradation ratio* were presented in [23]. We presented a scheme using a Naïve Bayes classifier and the IAT in [24]. In [25], the authors present a scheme that looks directly at the number of idle slots and the collision probability calculated on each observed node to determine if it is misbehaving or not.

While there are some detection schemes such as those presented in [26–28] that aim to modify the 802.11 protocol itself to improve detection capabilities, they are out of the scope of this thesis and not immediately applicable to the detection of our channel.

Furthermore, there have also been some mitigation schemes introduced that aim to prevent the use of covert channels without requiring detection of them, as seen in [29, 30]. These schemes usually add some delay or padding to traffic on the network to throw off the timing of covert timing channels. However, these schemes cannot affect our timing channel when used on the local wireless network, since no devices sit between the two nodes on the network. The timing of our packets stems from a fundamental backoff required in current 802.11 networks that cannot be removed without significantly modifying the standard.

Some previous schemes can stay covert well with very little throughput. Other schemes have higher throughput yet are easily detected. We seek to develop an 802.11 covert channel that optimizes throughput, accuracy, and covertness as necessary for intended usage. In this work, we are able to achieve throughput of 2500 bps while remaining covert with 85% accuracy, or 1800 bps while remaining covert with 99% accuracy.

Chapter 3

BACKGROUND INFORMATION

The 802.11 MAC uses carrier sense multiple access with collision avoidance (CSMA/CA), which is a DCF aimed at reducing collisions since they are not as easily detected on wireless networks as on wired networks. The process is described in full detail in [31], but below we discuss the essential details necessary for an understanding of our covert channel.

Before a station may transmit, it must sense whether or not the network is busy or not. This check must be performed at both the physical layer and in the network allocation vector (NAV), since two nodes may both be in range of an access point (AP), but not each other.

If the medium is sensed to be busy, then the station must wait until it is no longer in use before attempting to transmit. There is a required waiting period after the medium is no longer busy before transmission is allowed. This serves two purposes. First, certain messages such as acknowledgement (ACK) messages have higher priority than other messages such as a new packet transmission. To provide higher priority, ACK messages have to wait a shorter period of time, the Short Interframe Space (SIFS), to access the medium. Other packets must wait a longer period of time, the DCF Interframe Space (DIFS), in addition to a random period of backoff time, before transmission.

The random backoff time is required to solve the issue where multiple nodes may be waiting for the medium to become free, and thus otherwise would all attempt to use the medium at exactly the same time when the DIFS timer ran out. The random backoff time is calculated by multiplying a randomly selected number of slots from $[0, CW]$ by $slot\_time$, which is a constant value. The random number of slots is chosen in the range of $[0, CW]$ where the value of CW starts at CW_{min} and is doubled every time a collision occurs until $CW = CW_{max}$, at which point it remains at CW_{max} until a successful transmission takes

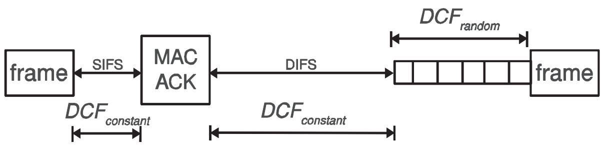


Figure 3.1. Transmission of packet using DCF

place. At this point, CW is set to CW_{min} again. This binary exponential backoff is designed to decrease the probability of collisions while keeping the wait time at a minimum. If at the end of the waiting period the medium is still free, the station may transmit.

Fig. 3.1 shows the events that take place during transmission of a packet. Of notable interest is the fact that outside of the random backoff, there are no other random times. In the next chapter, we use this to our advantage when developing Covert DCF.

In addition to the DCF, there are different physical layers (PHY) that may be used in 802.11 as well. For example, common PHY are direct-sequence (DS), frequency-hopping spread spectrum (FHSS), and orthogonal frequency-division multiplexing (OFDM). Each modulation technique defines the actual length of DIFS, SIFS, slot times, and other timing characteristics. While our proposed work does not modify or work directly at the physical layer of the 802.11 protocol, it is necessary to know which PHY layer is being used in order to perform the necessary calculations for backoff.

Chapter 4

PROPOSED SCHEME

Covert DCF stems from the random backoff that is required to take place when sending new packets. By taking control of this random backoff, we are able to encode different symbols using various backoff values without drawing immediate attention to our channel since the randomness helps disguise the channel.

The sender and receiver agree on a pre-defined codebook which maps symbols from $S = \{s \text{ is a bit string of length } l(s)\}$ to backoff values. For example, we may let $S = \{000_2, 001_2, 010_2, ..., 111_2\}$ which sends three bits of information at a time. Symbol 000_2 may be associated with a backoff of $100\mu s$ and symbol 010_2 with a backoff of $150\mu s$. We describe how we choose the symbols in more detail in Chapters 5 and 6.

Fig. 4.1 illustrates a high level view of a simple case using our covert channel. In this example, the receiving node is on the wireless side of the network, and $S = \{0000_2, ..., 1111_2\}$. The sender chooses to send the phrase BAD. There may or may not be regular nodes communicating on the network.

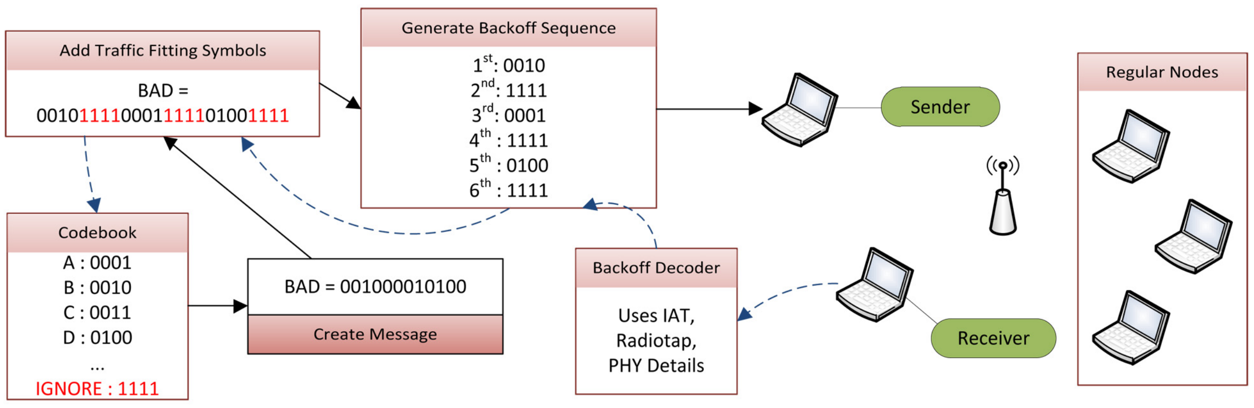


Figure 4.1. High level view of scheme

In the design of Covert DCF, we focus on three important characteristics to consider: throughput, accuracy, and covertness.

Throughput determines how fast we can send data across the channel. To increase throughput, we could increase the baud rate β, which is the number of state changes that take place over any given period of time. In the context of this research, state changes occur with each packet sent. Alternatively, we can change $|S|$, the number of symbols within S. If we let $l(s)$ be the number of bits in element $s \varepsilon S$, then throughput can be calculated as $bps = \beta * l(s)$ for our channel. Chapter 5 discusses optimizing throughput in more detail.

Accuracy is another important aspect to consider. Without high accuracy, the covert channel may not be very useful. In overt communication, including the standard 802.11 protocol, accuracy is often achieved through two-way communication and the use of ACK messages. However, covert channels are often one-way, thus ACK messages are not used and the channel must function without them. We consider accuracy in more detail in Chapter 5 as well.

Finally, we also wanted to consider covertness. After all, it is covertness which separates our channel from regular overt channels. It is important to try to balance throughput, accuracy, and covertness during the design of our channel. We present several methods used to determine covertness in the next chapter.

Chapter 5

MAXIMIZING THROUGHPUT, ACCURACY, AND COVERTNESS

In this chapter, we will analyze and discuss how we can maximize throughput, accuracy, and covertness. We also discuss any performance changes observed as we optimize each. Afterwards, we will devise a strategy of optimizing the channel for any particular use.

5.1 Throughput

In order to maximize throughput, we want to optimize both β and $|S|$. At first glance, an increase in either should increase the throughput.

We can increase β by minimizing the IAT. In this case, we minimize our choice of random backoff. For example, if we have $|S| = 4$, we can minimize the average IAT by using $s_1 = 0, s_2 = 1, s_3 = 2$, and $s_4 = 3$ for each symbol s_i. This will have a smaller average IAT than if we used $s_1 = 0, s_2 = 10, s_3 = 20$, and $s_4 = 30$.

Next, we calculate the average backoff for each set S as $|S|$ increases for any given PHY. By dividing the average backoff by $l(s)$, we are able to determine how long on average each bit takes to transmit, and in turn calculate the bits per second.

More formally, let $S = \{s \text{ is a bit string of length } l(s)\}$ be our symbol set. Furthermore, let our time values be in μs for all calculations. Then the average transmission time $avg\_trans\_time_{l(s)}$ can be calculated as

$$avg\_trans\_time_{l(s)} = \frac{\sum_{k=0}^{2l(s)} trans\_time_k}{2l(s)} \tag{5.1}$$

where

$$\begin{aligned} trans\_time_k =& DIFS + bo_k + TxTime \\ & + SIFS + ACKTxTime \end{aligned} \tag{5.2}$$

for $DIFS$, $SIFS$, and $ACKTxTime$ fixed for any particular PHY. $TxTime$ depends on the PHY along with payload size. Finally, bo_k is the backoff as determined using the scheme mentioned previously, and can be calculated as

$$bo_k = k(slot\_time) \tag{5.3}$$

where k represents the k^{th} symbol in S.

Note that we are only interested in symbol set sizes that make full use of all bits for any $l(s)$ length bit string. That is, we define S such that $|S| = 2^{l(s)}$.

Using the average transmission time, we conclude that the time per bit $tpb_{l(s)}$ can be calculated as

$$tpb_{l(s)} = \frac{avg\_trans\_time_{l(s)}}{l(s)} \tag{5.4}$$

and thus bits per second $bps_{l(s)}$ is

$$bps_{l(s)} = \frac{1\,000\,000\mu s}{tpb_{l(s)}}. \tag{5.5}$$

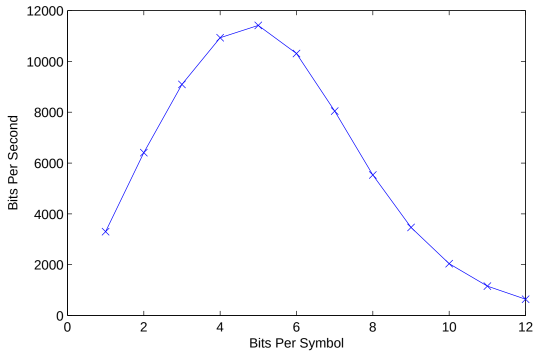


Figure 5.1. Theoretical Throughput

In Fig. 5.1, we see an initial increase in throughput as we increase $|S|$. However, we observe that it peaks around $l(s) = 5$ and then begins to drop. This decrease in throughput is due to the fact that each additional bit we add to s doubles $|S|$. This exponential increase in $|S|$ leads to an exponential increase in the average random backoff. On the other hand, we only see a linear gain in number of bits sent with each additional bit used in S. At approximately $l(s) = 5$ we find that the throughput gains by sending a larger number of bits at a time are outweighed by the significant increase in random backoff.

We find similar results for other PHY. The peak tends to be located when $4 \leq l(s) \leq 6$ depending on the PHY characteristics and average packet size.

These results form curves similar to those seen in [15].

We find that theoretically our maximum throughput is high. We can theoretically obtain bit rates far greater than introduced in other covert channels. We must keep in mind that we have created this scheme specifically to focus on maximizing throughput without regard to the effects on accuracy and covertness. However, as we will see later in the thesis, we can maintain high accuracy, covertness, and high throughput at the same time in many cases.

5.2 Accuracy

When considering accuracy, we must consider how many nodes are sending on the wireless network, where the receiving or listening node is located (wired side or wireless side of the network), and how much error we are allotted when calculating the number of slots used during the backoff.

Since accuracy depends on the ability of the receiving node to properly identify the correct IAT, the number of senders plays an important role. If there is only one sender, the error in decoding the IAT should be minimal since the sending node always obtains the wireless medium when desired. However, in the case of multiple senders, another node may choose a shorter random backoff and access the medium before the malicious node. If this is the case, the malicious node will have to defer transmission until the medium is free again. These other transmissions caused by other nodes can cause increases in the IAT when examining two consecutive packets sent by the malicious node.

Case 1. *single sender, wireless receiver:* In the first case, high accuracy is easy to obtain. Since the malicious node does not compete with other nodes for the medium, there should be minimal delays.

Case 2. *single sender, wired receiver:* Similarly, in single-sender wireless-side, we can also decode the IAT with high accuracy for the same reason as described in Case 1. However, there may be slightly larger delays than in Case 1 due to the additional hop required on the network. We can compensate for these slightly larger delays by using ranges for each symbol instead of single values. That is, instead of using the set $S = \{s \text{ is a bit string}\}$ to represent our symbols we use $\mathcal{A} = \{S \text{ is a unique set of consecutive bit strings}\}$.

Case 3. *multiple senders, wireless receiver:* In this case, we must account for the fact that other nodes may use the medium, thus causing the malicious node to delay transmission. Since the receiving node is located on the wireless side of the network, the receiver can observe all traffic on the wireless medium. If the receiver senses a frame sent by another node (either physically or via the NAV) it can simply disregard the information and adjust its IAT timer accordingly.

Case 4. *multiple senders, wired receiver:* This last case is the most challenging out of the four cases. We cannot assume that the receiving node knows when other nodes transmit on the wireless network. This inability leads to difficulties in calculating the proper IAT since there may or may not have been delays due to other nodes on the wireless network. To handle this situation, we must create our codebook in such a way that received symbols cannot be improperly interpreted.

To do so, we must know the minimum amount of time required for a transmission on the wireless medium. Before using the covert channel, the sending and receiving nodes must know the distribution of packet sizes on the wireless network.

By obtaining the distribution of packet sizes, we can calculate the extra delay added per additional transmission by other nodes. For example, if we know that 95% of packets sent on the network are 1125-1875 bytes in length, then we can calculate with 95% confidence that the amount of time required to send a transmission will be between $225.97\mu s$ and $331.97\mu s$ (using DS PHY in this case). In this example, we obtain that if we let $S = \{0000_2, ..., 1100_2\}$, and the receiver decodes it as a value in that range, then no other packets were sent between our two transmissions, thus the IAT should be decoded correctly. Fig. 5.2 demonstrates the two decoding possibilities for the lower bound of this example. Using this knowledge, we are able to choose the proper codebook and symbol set for the network.

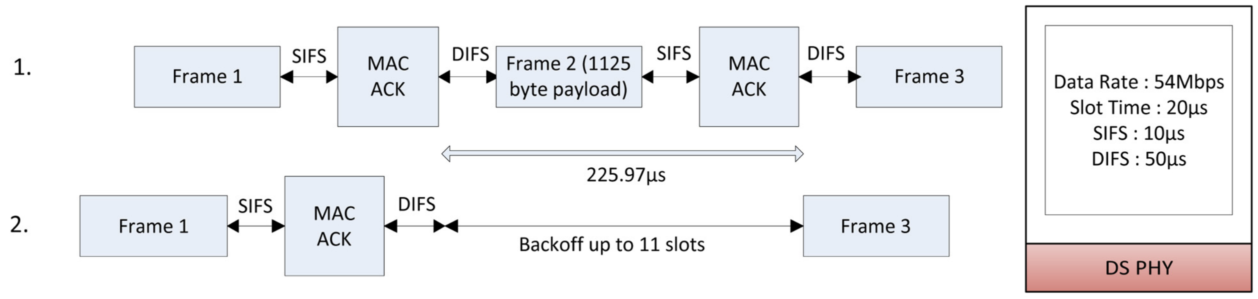
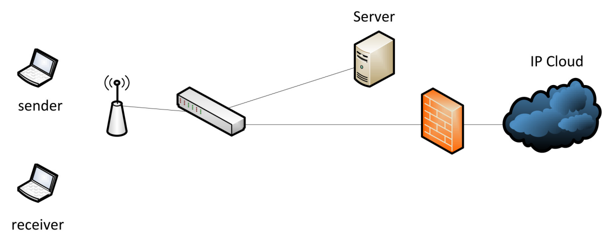


Figure 5.2. Minimum time required to send packet

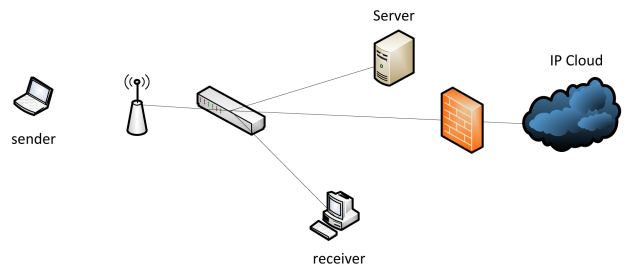
We must take note that this will significantly decrease our choice of S, since we can only use small values for the backoff. In our example above, we cannot make use of all symbols where $l(s) = 4$, since any symbols larger than 1100_2 will require more than $331.97\mu s$ to transmit.

We must also mention that the 802.11 DCF protocol by default selects a random backoff in the range of [0, CW]. Thus if we continue to use the same scheme with our chosen CW values, we will see overlap between symbols. For example, if symbol s_1 is represented by $CW = 8$ and s_2 has $CW = 3$, then there is a 44% chance that s_1 and s_2 overlap. Instead, we simply adjust and set our random backoff to the CW value itself. This requires MAC misbehavior on the sending node, but all other nodes remain untouched.

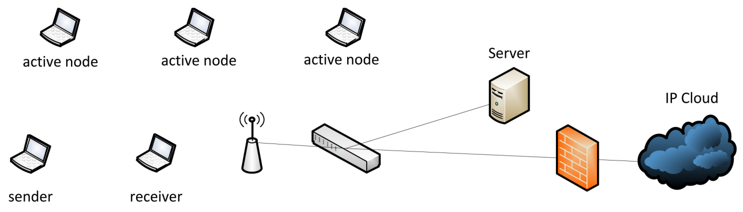
In addition to the location of the sending and receiving nodes, we should also consider how much error we allot the receiver node when calculating the backoff slots. Earlier, we attempted to maximize our throughput by optimizing β. We set each symbol to a single backoff, without any room for error. While this does help improve our throughput on the sending node, it can affect the accuracy on the receiving node. To increase accuracy, we let $\mathcal{A} = \{S \text{ is a unique set of consecutive bit strings}\}$. In other words, instead of using the higher throughput values $s_1 = 0, s_2 = 1, s_3 = 2$, and $s_4 = 3$ we could use $s_1 = 0 - 9, s_2 = 10 - 19, s_3 = 20 - 29$, and $s_4 = 30 - 39$. The former case requires an exact calculation on the number of backoff slots, whereas the latter allows us a little room for error from delays.



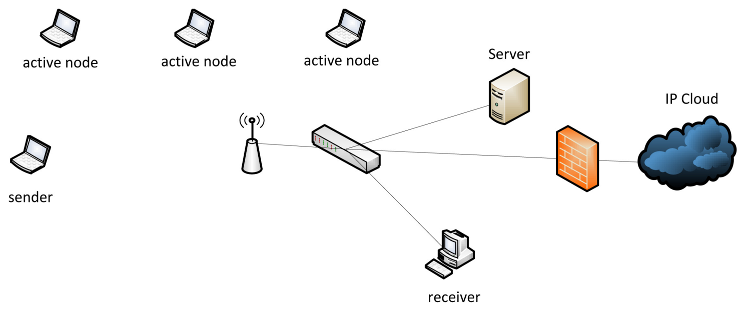
(a) Case 1



(b) Case 2



(c) Case 3



(d) Case 4

Figure 5.3. Accuracy Cases

5.3 Covertness

Lastly, we want to maximize our covertness. There are several angles from which we can analyze covertness.

Covertness is often evaluated based on throughput or traffic distribution. In [15], Sellke et al. demonstrate a covert channel that models a Pareto distribution, similar to Telnet traffic, at the sacrifice of a low throughput. Some channels introduce their own traffic into the network, whereas others are more passive such as PSUDP, which modifies pre-existing DNS queries to embed a storage channel [32].

However, there are also some 802.11 specific detections that we wish to avoid as well. Because we are performing MAC misbehavior on our sending node, we need to avoid MAC misbehavior detection schemes similar to those presented earlier.

Similarly, we need to be able to bypass wireless intrusion detection systems (WIDS) and wireless intrusion prevention systems (WIPS). At the time of this writing, WIDS and WIPS have seen a lot of development in the field of wireless device management (AP management and authorization, rogue access point detection) along with traffic analysis, yet as far as we are aware none currently look for covert channels. Nevertheless, we do not want our malicious node to create an anomaly displayed on traffic analysis tools.

Consider Motorola AirDefense Enterprise [33], which is a popular commercial WIPS solution. As of this writing, AirDefense Enterprise provides down to the minute granularity with regards to traffic data point collection. Thus we need to ensure that our malicious node's data points match those of regular traffic. To do so, we only need to ensure that over any given 60 second window, our traffic averages out similar to that of regular traffic over a given 60 second window.

In this thesis, we base our covert analysis on:

1. Throughput Changes
   (a) sender throughput gain
   (b) legitimate node traffic degradation
   (c) network throughput change
2. Sorted backoff for traffic regularity

3. Two-Sample Kolmogorov-Smirnov test

In all of the above cases, we use data and samples that fit a 60 second window. These tests allow us to assess our covert channel both visually and quantitatively within the capabilities of WIDS / WIPS.

Chapter 6

COVERT DCF PROTOCOL DESIGN

In this Chapter, we will present our protocol used for sending and receiving.

6.1 Covert DCF Sending Protocol

Before we begin using Covert DCF, we must first do some analysis on the network in order to help us increase our accuracy and covertness. First, we capture traffic on the WLAN in order to obtain average throughput, traffic distribution, and packet size distribution. The first two will help us increase our covertness, whereas packet size will help us increase our accuracy. Note that both the receiver and the sender must have this information.

We also need to know which PHY characteristics are being used. We can obtain this information using the Radiotap headers [34]. We must note that this information must also be available to both sender and receiver. As such, if the receiving node is on the wired-side of the network, then we must assume all traffic uses the same PHY throughout the covert communication.

Next, the sender will take the bit sequence that represents the covert message and encode it into the proper delay sequence based on a codebook. The specifics of the codebook will depend on the requirements for the covert channel, and the location of the receiving node.

If the receiving node is on the wireless side of the network and within listening distance of the sending node, then we can use a larger symbol set $|S|$ with a larger baud rate, thus increasing our throughput.

If the receiving node is on the wired side of the network, we must be able to ensure that the messages can be decoded with high accuracy. Other sending nodes on the network may access the medium first, thus causing our IAT to consist of our planned IAT, along with the extra delay involved with the medium being used by other nodes or due to collisions. By choosing an appropriately small S as discussed in Chapter 5, we are able to correctly decode the IAT in this situation.

Furthermore, we may choose to increase our allotted error by using ranges of values instead of single values for our symbols. For networks with heavier delays, a larger range would be required.

Once the sender has S along with the delay sequence, we must ensure that it closely matches the traffic throughput and distribution of other nodes. In order to do this, we rely on *traffic-fitting symbols*. These are symbols in the codebook that have no meaning to the sender or receiver - they are used solely for increasing covertness. Alternatively, traffic-fitting symbols could be time based, such as embedding them in every 5th packet, provided the receiver knows to ignore every 5th packet. Algorithm 1 below presents our scheme for inserting traffic-fitting symbols.

Algorithm 1 Sequence Adjustment

```
 1: for all windows do
 2:    $t \Leftarrow training\_window$
 3:    $m \Leftarrow message\_window$
 4:    for all v in t do
 5:       $t\_prob[v] \Leftarrow \frac{COUNT(v)}{SIZE(t)}$
 6:    end for
 7:    $n \Leftarrow m$
 8:    while $SIZE(n) < window\_size$ do
 9:       for all v in n do
10:          $n\_prob[v] \Leftarrow \frac{COUNT(v)}{SIZE(n)}$
11:       end for
12:       $largest\_diff \Leftarrow MAX(t\_prob[v] - n\_prob[v])$
13:       if $largest\_diff > 0$ then
14:          PUSH n, ld_value
15:       else
16:          for all p in n_prob do
17:             if $p > 0$ then
18:                PUSH n, p_value
19:             end if
20:          end for
21:       end if
22:    end while
23: end for
```

Our goal is to match the empirical cumulative distribution function (ecdf) of the backoffs on our sending node to that of backoffs from a legitimate node. To begin, we need training data from legitimate traffic that contains at least as many data points as our initial sequence of symbols we wish to send. We then break both sets down into windows, which allows us to match distributions within the entire set as well as the over all ecdf.

For each window, we calculate the probabilities of each backoff from our training data. That is, we may discover that a value of *5 slot times* occurs 15% of the time, and a value of *20 slot times* only occurs 5% of the time. We then also calculate the probabilities of our current sequence we intend to send - this is initialized to the initial sequence and will grow as we insert traffic-fitting symbols. Next, we adjust the probabilities as needed, but we must note that we cannot remove symbols from our new sequence, as we rely on them for our message. We can only add to the sequence.

In our example, say a value of 5 slot times occurs 15% of the time in our training data and only 10% of the time in our covert sequence. In this case, we simply need to insert more values of 5 slots into our sequence until it reaches the correct probability.

On the other hand, consider if the same value occurs 15% of the time in the training data and 20% of the time in our sequence. We cannot simply remove the extra, since we rely on them for our message. Instead, we increase the probability of *other symbols*, which increases the population size thus decreasing the probability of our target value. In particular, we choose to increase the probability of other symbols which are already too low - this way we help give them a higher probability and decrease the probability of our target value at the same time.

When inserting our traffic fitting symbols, we do so using the placement based method. We may send 1 covert packet per 3 traffic-fitting symbols. This ratio must be known on both ends so that the proper packets will be thrown out upon decoding.

After one round of this adjustment, our sequence ecdf should be slightly closer to that of our target ecdf. We repeat this process for multiple rounds, each time re-calcuating the probabilities of our covert sequence (since they will change each round), and after enough rounds the ecdf of our covert sequence window and our training window will be similar. If our initial sequence were similar to begin with, it will take fewer rounds than if it were quite dissimilar.

Using this method, it allows us to match the ecdf of any training data, regardless of if there is a known distribution with an inverse that matches it, which is a method seen in [15]. When performing our tests, we were unable to find a good fit to a known distribution that also had an inverse function for our training data.

At this point, we should have a sequence that will have a high throughput, accuracy, and covertness. A malicious process on the sending node uses this delay sequence for subsequent traffic. The traffic can be destined for any location, as long as the receiving node is en route.

Algorithm 2 Backoff Selection

```
Require: $use\_covert \Leftarrow true$
Require: $delay\_seq \Leftarrow$ symbol sequence
Require: $seq\_pos \Leftarrow -1$
Require: $seq\_len \Leftarrow len(delay\_seq) - 1$
 1: for all packets to send do
 2:   if $use\_covert$ is $true$ then
 3:     if $short\_retry\_count + long\_retry\_count = 0$ or $perform\_cw$ is $true$ then
 4:       $seq\_pos \Leftarrow seq\_pos + 1$
 5:     end if
 6:     $backoff\_slots \Leftarrow delay\_seq[seq\_pos]$
 7:     if seq_pos = seq_len then
 8:       $use\_covert \Leftarrow false$
 9:     end if
10:   else
11:     $backoff\_slots \Leftarrow rand(0, CW)$ {use standard procedure}
12:   end if{send packet}
13: end for
```

To send, we implement Algorithm 2 in the MAC layer. This algorithm steps through our sequence of delays sending them appropriately. If we are resending a packet for any reason such as a collision, then we resend it using the appropriate delay. Once the entire message has been sent, we return to using the standard MAC protocol.

6.2 Covert DCF Receiving Protocol

The decoding node listens for all packets on the network. Knowing the MAC address or IP address of the sending node, the decoder can properly choose when to decode a symbol and when to wait until the appropriate packet is processed to decode the symbol. Note that the decoder must be either on the same basic service set (BSS), which consists of all stations sharing a single AP, as the sending node (using the MAC address) or on the network between the sender and destination address (using the IP address).

First, the receiver must calculate the IAT. Next, the receiver calculates the backoff in terms of μs and from that obtains the number of backoff slots used.

To obtain the backoff slot count, the decoder initializes backoff time $bo\_time = IAT$. Next, the decoder subtracts DIFS, SIFS, payload transmission time $TxTime$, and ACK transmission time $ACKTxTime$. At the end of this process, we can convert the bo_time to actual slots (thus symbols) by dividing it by the $slot\_time$ for the given PHY. This process can be seen in Algorithm 3.

Algorithm 3 Symbol Decoding

```
Require: $prev\_recv\_time \Leftarrow 0$
Ensure: list of decoded symbols
 1: for all packets received do
 2:    $curr\_recv\_time \Leftarrow NOW$
 3:    $IAT \Leftarrow curr\_recv\_time - prev\_recv\_time$
 4:    $bo\_time \Leftarrow IAT$ {initialize}
 5:    $bo\_time \Leftarrow bo\_time - difs\_time$
 6:    $bo\_time \Leftarrow bo\_time - \frac{rcvd\_pk\_size}{rcvd\_frame\_drate}$ {TxTime}
 7:    $bo\_time \Leftarrow bo\_time - sifs\_time$
 8:    $bo\_time \Leftarrow bo\_time - \frac{ack\_pk\_size}{rcvd\_frame\_drate}$ {AckTxTime}
 9:    $decoded\_slots \Leftarrow \frac{bo\_time}{slot\_time}$
10:    if $remote\_addr = sender\_addr$ then
11:       $decoded\_slots \Leftarrow decoded\_slots + offset$
12:       $offset \Leftarrow 0$
13:       print $decoded\_slots$
14:    else
15:       $offset \Leftarrow offset + decoded\_slots$
16:    end if
17:    $prev\_recv\_time \Leftarrow curr\_recv\_time$
18: end for
```

If the receiver is on the same BSS as the sending node, the receiver can also compensate for other traffic on the network to ensure the accuracy remains high regardless of other traffic. To do so, if a packet is detected from a node other than the sending node, then the receiver calculates the $decoded\_slots$ as described above and adds the resulting value to $offset$. When a packet arrives from the sending node, the receiving node adds $offset$ to the number of decoded slots and reset $offset$ to 0. By using this offset, we are able to calculate the correct backoff that was initially chosen by the sending node even though there may have been other traffic on the network.

Once the receiver has obtained the set of symbols sent by the sending node, the receiver looks these symbols up in the pre-determined codebook to determine the values represented by each symbol. If traffic-fitting symbols have been used, the receiver ignores those symbols.

Chapter 7

IMPLEMENTATION, PERFORMANCE EVALUATION AND NUMERICAL RESULTS

7.1 Implementation

To test Covert DCF, we used OPNET to simulate our covert channel. Our simulation consisted of a BSS consisting of our sending node, one wireless receiving node, one wired receiving node one hop away, one access point, and 15 regular wireless nodes. The AP was connected to a switch, which also had connections to Ethernet servers and the IP cloud, as illustrated in Fig. 7.1.

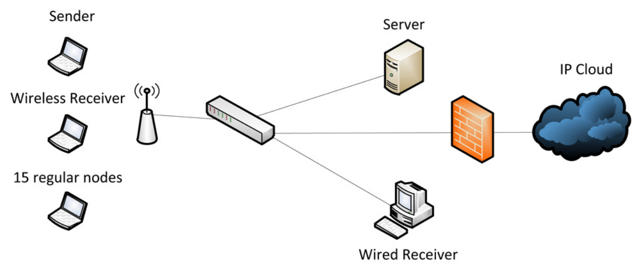


Figure 7.1. OPNET Network Layout

For our simulations, we chose to use the 802.11g PHY characteristics, since it is both widely depoloyed and offers 54Mbps data rate. At the application layer, we used FTP as it can provide many packets back to back which is necessary for a covert timing channel. However, we also did some analysis on other protocols such as HTTP and present those results as well.

We ran 100 trials for each configuration of $|S|$ up to $l(s) = 12$, and a summary of our results can be found in Table 7.1.

We modified the wlan_mac process model in OPNET in order to test our channel.

Summary of Simulations			
Throughput	Covert	Accuracy	Receiver
8600bps	No	85%	wireless
5500bps	No	99%	wireless
2500bps	Yes	85%	wireless
1800bps	Yes	99%	wireless
5500bps	No	27%	wired
900bps	No	99%	wired
140bps	Yes	99%	wired

Table 7.1. Summary of simulations

In addition, we made the necessary changes to wlan_mac_dispatch, wlan_workstation, and wlan_station. We added an option to "enable" or "disable" covert channel functionality within each node. Using this switch, we enabled the covert channel on our sending node and receiving node and disabled it on all other nodes (thus leaving the original configuration for those nodes).

Our sending node read a message from a standard text file, and using the steps as described in the previous chapter, created the covert sequence. The receiving nodes were set to packet capture mode for all packets and decoded the sequence as we have previously discussed. We ran 100 trials for each configuration of $|S|$ up to $l(s) = 12$.

7.2 Throughput

Our simulations show that we are able to obtain high throughput using our channel. Fig. 7.2 shows a comparison of our theoretical throughput as presented in Chapter 5, and the simulated throughput from sending an ASCII file in plain text, compressed, and CAST5 encrypted versions. We can see that our channel performed similar to our expected performance. However, it can be observed that some data points for $l(s) > 8$ during the plain text transmission are actually *slightly higher* than our predicted rates. This can be explained by distribution of backoffs used for the plain text file. The theoretical throughput was based on sending completely *random* symbols, whereas our codebook mapping of our plain text file consisting of English language ASCII characters, mapped to the lower valued symbols from S rather than the entire range for ease of calculations. Both compressing and encrypting the message remove this information from the file, thus they follow the theoretical value more closely.

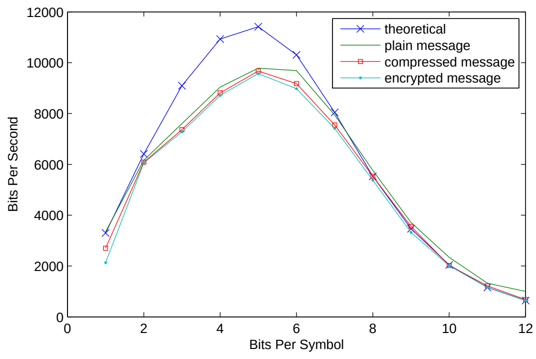


Figure 7.2. Covert channel throughput comparison (ASCII text)

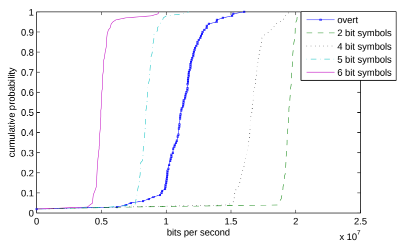
7.3 Covertness

As mentioned in Chapter 5, we will analyze the covertness of our channel by looking at throughput changes, sorted backoffs, and two-sample Kolmogorov-Smirnov tests.

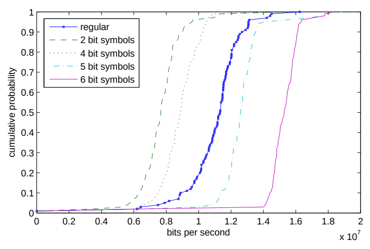
7.3.1 Throughput Changes

We will first show that throughput changes can occur using our channel if we assume an *aggressive sender* that does not aim to be covert. We will then demonstrate how we can improve the channel to have little effect on the throughput of the standard communication channel.

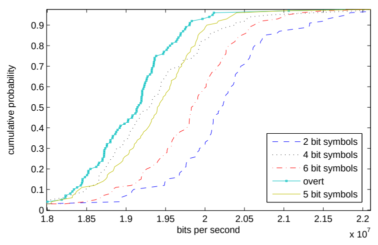
Fig. 7.3 shows various throughput variations of the standard channel communication assuming we have a very aggressive sender, using a 60 second window. Fig. 7.3(a) shows us that as we vary $|S|$, our standard channel throughput varies as well. This makes sense, because if we use a smaller $|S|$ and use small backoffs, then we will win the contention on the network more often than other nodes, thus getting an unfair usage of the network. We can see that between $l(s) = 4$ and $l(s) = 5$ is where our node *should* be if it were not performing MAC misbehavior. Using $l(s) = 3$ yields throughput increases similar to those in [35] for $\alpha = 0.5$ due to the fact that they use the same set of backoffs. Similarly, Fig. 7.3(b) shows the traffic degradation of legitimate nodes when Covert DCF is in use.



(a) Malicious node throughput

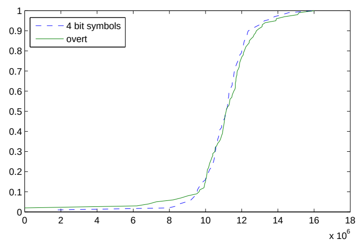


(b) Legitimate node throughput

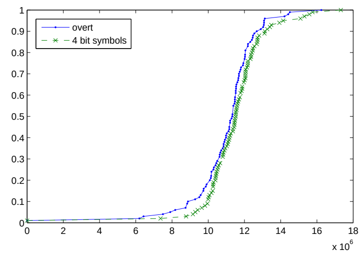


(c) Network throughput

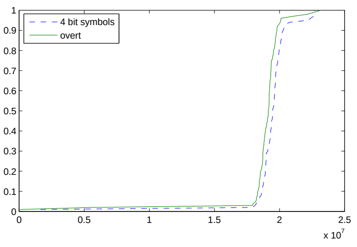
Figure 7.3. Standard channel throughput variations for aggressive sender



(a) Malicious node throughput



(b) Legitimate node throughput



(c) Network throughput

Figure 7.4. Standard channel throughput variations for covert sender

However, in Fig. 7.3(c) we see that if our malicious node is very aggressive, the overall network throughput can actually increase. This is due to the fact that our malicious node easily wins the contention, and thus on average the backoffs used on the network are much smaller.

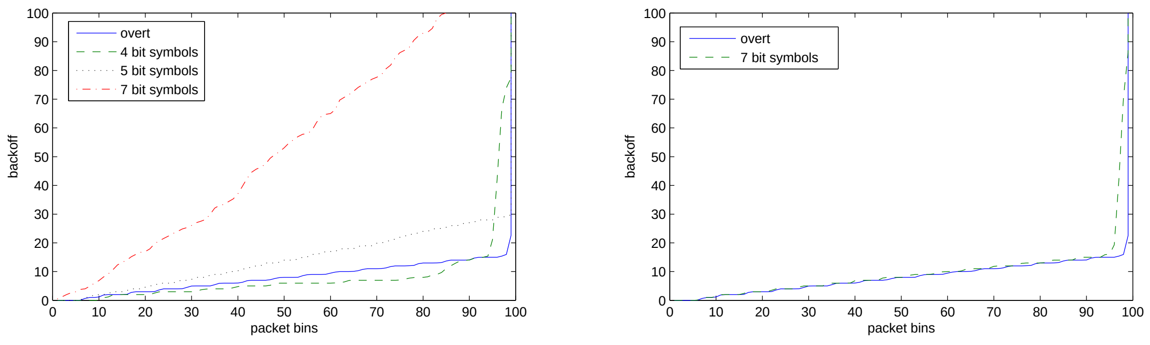
In all graphs in Fig. 7.3, we observe that the closest matches to legitimate traffic falls somewhere between 4 bit symbol and 5 bit symbol usage.

If we are smart about our usage of the channel, however, we can make it much more difficult to detect, as seen in 7.4. By embedding some *traffic-fitting symbols*, or backoffs that carry no meaning, within our covert message, we are able to get a much closer fit. This does degrade Covert DCF throughput in exchange for additional covertness. Using this scheme, our throughput drops to roughly $2500bps$ from over $8000bps$. It is up to the user to determine how important throughput, accuracy, and covertness are for the channel.

7.3.2 Sorted Backoffs

We can also look at the sorted backoff times as presented in [13]. In [13], the authors used the sorted backoffs to look for steps in the graph, which represent regularity in the IAT times. It also provides yet another method for viewing and comparing the distribution of packets. Fig. 7.5(a) shows the sorted backoff times for aggressive usage of the covert channel. We can see that if we send 4 bits at a time, the sorted backoffs follow much closer to that of overt traffic than if we send 7 bits at a time. This is because 4 bit symbols align to 16 backoffs, and by default overt traffic initially selects a backoff from $[0, 15]$ on 802.11g networks. If each symbol occupied two backoff slots to increase accuracy, then 3 bit symbols offer the best match. However, 7 bit symbols are easily detected if heavily used, since many more backoffs are used than in overt communication.

Again, we can be smarter about our usage. By implementing our traffic-fitting symbols, we can see that it is much more difficult to detect. Fig. 7.5(b) shows a comparison of the sorted backoffs for overt communication and the previously easily detected 7 bit symbol covert channel when we sacrifice throughput for covertness.



(a) Aggressive usage (b) Overt usage

Figure 7.5. Sorted backoff times

7.3.3 Two-Sample Kolmogorov-Smirnov Test

Lastly, we perform some quantitative analysis by comparing the empirical cumulative distribution functions (ecdf) of legitimate traffic slot times and our covert traffic slot times. These come directly from the IAT and random backoff. We then perform a Kolmogorov-Smirnov (K-S) test to determine if it is possible if the two sets of data come from the same distribution. For the K-S test, we state that the null hypothesis is that the two samples come from the same distribution, and we use a significance level of $\alpha = 0.05$.

Fig. 7.6 shows the ecdf for legitimate traffic, traffic consisting of 10% message symbols and 90% traffic-fitting symbols, and traffic consisting of 100% message symbols without any traffic-fitting symbols.

We can see that the legitimate traffic distribution and the 10% message symbol traffic distribution are nearly identical, whereas the message without traffic-fitting symbols stands out.

Furthermore, if we run a K-S test comparing the legitimate traffic vs. 10% message symbol traffic and legitimate traffic vs. 100% message symbol traffic, we obtain the p-values 0.6 and 0.001 respectively. Similarly, the K-S test statistics for these were 0.04 and 0.13. Thus with at the 10% rate, we cannot reject the null hypothesis, but we do reject it if we do not include traffic-fitting symbols.

However, as we increase the number of bits sent per symbol, it becomes more difficult to fit the distribution. We must insert many more traffic-fitting symbols to align the two distributions. Fig. 7.7 shows the same tests performed using 7 bit symbols. We can see that

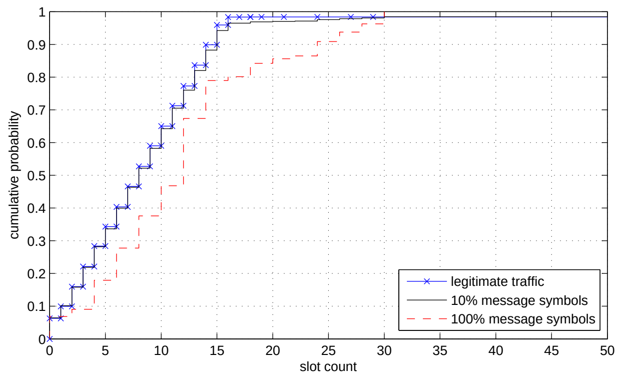


Figure 7.6. 4 Bit Symbol Backoff - Empirical CDF

while the 10% rate is much closer to legitimate traffic than the 100% rate, it still is not quite aligned with the original distribution. In both cases, we must reject the null hypothesis at the 5% significance level.

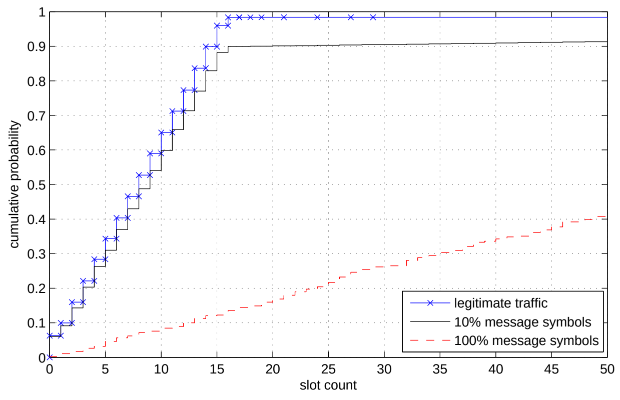


Figure 7.7. 7 Bit Symbol Backoff - Empirical CDF

In this sample network traffic, it was possible to remain covert with 4 bit symbols at the 10% and 25% rates using $\alpha = 0.05$. On a network with more delays, the distributions of 5, 6, or 7 bit symbols may align with legitimate traffic backoffs more closely, thus allowing those to be used as well.

We must note that as we increase the number of traffic-fitting symbols, we do lose throughput since we must send more symbols. At 10%, the length of our covert sequence is 10 times longer than it would be without traffic-fitting symbols, and we obtained throughput of 1367 bps. Similarly, at the 25% rate, the covert sequence will be 4 times longer than without traffic fitting-symbols 1890 bps.

7.4 Accuracy

To maximize throughput, we chose to separate our symbols by a single slot time in order to increase β as described earlier. Using this scheme, we found that we were able to achieve 85% accuracy while the network was under heavy load by adjusting the offset value for traffic generated by other nodes. However, further inspection showed that many of the values that were decoded incorrectly were only off by one slot time.

Thus, by allowing each symbol to cover a range of values (as few as two) rather than a single backoff value, we could easily increase the accuracy. In this case, we let each symbol occupy two backoff slot positions. In doing so, our accuracy went from 85% to over 99% for the same message. We must note that this adjustment can also decrease β and thus the throughput. Our high throughput dropped from an average of $8600bps$ to an average of $5500bps$ without traffic-fitting symbols. We could achieve $1890bps$ with 99% accuracy with traffic-fitting symbols.

Following our previous findings, we analyzed our results from the wired receiver. We continued to use $\mathcal{A}$ as our symbol set varying $|\mathcal{A}|$. Similar to above, we found that we could not decode the sequence perfectly using single values. Not surprisingly, we also discovered that letting each symbol be represented by a range of two values also failed to have high accuracy due to additional network delays. Using a range of two for each symbol only gave us an accuracy of 27%. However, by setting our range to fifteen instead of two, we were able to bring our accuracy back up to over 99%. In order to gain this higher accuracy, we had to accept a throughput of only $900bps$.

7.5 Light Traffic Protocols

As mentioned earlier, we used FTP at the application layer to facilitate back to back traffic, which is necessary for a covert timing channel. Without this type of traffic, it is impossible to tell whether or not a given IAT was planned due to the timing channel or simply delays at the user or application level.

However, we can make an educated guess whether or not delays were from the covert timing channel or user delays by using a timing threshhold. For example, if our maximum delay used in the covert timing channel is $500\mu s$ and the receiving node detects a delay of 3 seconds (after processing any calculated offset), then it is safe to assume this is not part of the covert timing channel, so we toss the packet.

So let our threshhold value be

$$threshhold = DIFS + SIFS + TxTime + AckTxTime + (max\_slots * slot\_time).$$

At the sending node, if two packets are not back to back, then the receiver does not encode the next symbol for the covert channel. At the receiving node, if a packet arrives above the threshhold value, the receiver ignores it since the packet did not immediately follow the previous packet. Otherwise, the receiver assumes the two packets were back to back and decodes the symbol accordingly.

Using this scheme we can still obtain high accuracy and covertness, but throughput is *significantly reduced* if the application level does not generate heavy back-to-back traffic. We tested using an assortment of light usage traffic (HTTP, SMTP, POP3, etc), and throughput was approximately $0.03bps$ - extremely low. A 5kb text file takes approximately 10 days to transmit at this rate. However, we noticed that by increasing $|S|$, we could obtain faster throughputs. By using 10 bits per symbol instead of 5, we were able to get a 50% increase in throughput. Since we must wait so long for two packets to be sent back to back, it makes sense to send as much data as possible. We just need to ensure that we do not increase it too much, or else application level delays and covert timing delays may get confused. We simply need to ensure that our threshhold is high enough to account for all symbols and low enough that it does not allow for application level delays to fall below it.

We must note that without heavy back to back traffic, *any covert timing channel would be affected*, not just Covert DCF. Timing channels rely on the timing between packets, so the packets must exist and be ready to be sent when needed.

Chapter 8

CONCLUSIONS

In this thesis, we introduced a scheme to implement a covert timing channel that is applicable for wireless networks that make use of random backoff in order to avoid collisions. We provided an analysis of aspects resulting in a good covert channel. We demonstrated our channel using the 802.11g wireless protocol through analysis and simulation.

Thus far, we have shown that it is possible to obtain throughput for our covert timing channel far greater than that of previous covert timing channels. Our channel also maintains good accuracy and can operate covertly as well. When both the sending and receiving node are on the same wireless network, we were able to obtain over 8000 bps throughput with approximately 85% accuracy, or by slightly modifying our code we were able to increase accuracy to over 99% while maintaining a throughput over 5000 bps. Adding in covertness drops our throughput to 1800*bps*, but we are still able to maintain over 99% accuracy at this rate while operating in a more covert fashion. In both cases, the network was under heavy load from regular nodes as well. When the receiving node was an additional hop away on the wired network, we were still able to maintain over 99% accuracy but at the sacrifice of higher throughput. We were only able to maintain approximately 900*bps* throughput in this case. This figure drops futher if the need to be covert is high.

In comparison with other covert channels, Covert DCF offers a good alternative when the covert channel is to be used over shorter distances such as wireless LANs or within a couple hops of the wireless LAN.

Chapter 9

FUTURE WORK

In the future, we intend to perform additional investigations of methods to increase the covertness of Covert DCF and additional analysis for the detection of Covert DCF. Covert channels will continue to be an interesting area of research in the foreseeable future, and methods of increasing the covertness of our channel and other covert channels will be exciting to explore.

REFERENCES

[1] "Common methodology for information technology security evaluation," July 2009.

[2] D. E. Bell and L. J. LaPadula, "Secure computer systems: Mathematical foundations," MITRE Corporation, Tech. Rep., March 1973.

[3] Intego, "Hacker tool copies personal info from iphones," November 2009, `http://www.intego.com`.

[4] T. Calhoun, R. Newman, and R. Beyah, "Authentication in 802.11 lans using a covert side channel," *IEEE International Conference on Communications (ICC).*, pp. 1 –6, jun. 2009.

[5] P. Kyasanur and N. Vaidya, "Detection and handling of mac layer misbehavior in wireless networks," in *International Conference on Dependable Systems and Networks (DSN).*, jun. 2003, pp. 173 – 182.

[6] L. Ji, H. Liang, Y. Song, and X. Niu, "A normal-traffic network covert channel," in *International Conference on Computational Intelligence and Security (CIS).*, vol. 1, dec. 2009, pp. 499 –503.

[7] H. Khan, Y. Javed, F. Mirza, and S. Khayam, "Embedding a covert channel in active network connections," in *IEEE Global Telecommunications Conference (GLOBECOM).*, nov. 2009, pp. 1 –6.

[8] X. Luo, E. Chan, and R. Chang, "Clack: A network covert channel based on partial acknowledgment encoding," in *IEEE International Conference on Communications (ICC).*, jun. 2009, pp. 1 –5.

[9] L. Ji, Y. Fan, and C. Ma, "Covert channel for local area network," in *IEEE International Conference on Wireless Communications, Networking and Information Security (WCNIS).*, jun. 2010, pp. 316 –319.

[10] H. Zhao, Y. Q. Shi, and N. Ansari, "Hiding data in multimedia streaming over networks," in *Eighth Annual Communication Networks and Services Research Conference (CNSR).*, may. 2010, pp. 50 –55.

[11] P. Basu and T. Bhowmik, "On embedding of text in audio a case of steganography," in *International Conference on Recent Trends in Information, Telecommunication and Computing (ITC).*, mar. 2010, pp. 203 –206.

[12] L. Frikha and Z. Trabelsi, "A new covert channel in wifi networks," in *Third International Conference on Risks and Security of Internet and Systems (CRiSIS).*, oct. 2008, pp. 255 –260.

[13] S. Cabuk, C. E. Brodley, and C. Shields, "Ip covert timing channels: design and detection," in *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security.* New York, NY, USA: ACM, 2004, pp. 178–187.

[14] S. Cabuk, C. E. Brodley, and C. Shields, "Ip covert channel detection," *ACM Trans. Inf. Syst. Secur.*, vol. 12, no. 4, pp. 1–29, 2009.

[15] S. Sellke, C.-C. Wang, S. Bagchi, and N. Shroff, “Tcp/ip timing channels: Theory to implementation,” in *INFOCOM 2009, IEEE*, April 2009, pp. 2204–2212.

[16] T. Calhoun, X. Cao, Y. Li, and R. Beyah, “An 802.11 mac layer covert channel,” *Wireless Communications and Mobile Computing.*

[17] L. Frikha, Z. Trabelsi, and W. El-Hajj, “Implementation of a covert channel in the 802.11 header,” *Wireless Communications and Mobile Computing*, 2008.

[18] V. Anantharam and S. Verdu, “Bits through queues,” *IEEE Transactions on Information Theory.*, vol. 42, no. 1, pp. 4–18, Jan 1996.

[19] S. H. Sellke, C.-C. Wang, N. Shroff, and S. Bagchi, “Capacity bounds on timing channels with bounded service times,” in *IEEE International Symposium on Information Theory (ISIT).*, June 2007, pp. 981–985.

[20] I. Moskowitz and A. Miller, “The channel capacity of a certain noisy timing channel,” *IEEE Transactions on Information Theory.*, vol. 38, no. 4, pp. 1339–1344, Jul 1992.

[21] J. Son and J. Alves-Foss, “Covert timing channel capacity of rate monotonic real-time scheduling algorithm in mls systems,” 2006.

[22] Y. Rong, S.-K. Lee, and H.-A. Choi, “Detecting stations cheating on backoff rules in 802.11 networks using sequential analysis,” in *25th IEEE International Conference on Computer Communications (INFOCOM).*, apr. 2006, pp. 1 –13.

[23] Z. Lu, C. Wang, and W. Wang, “On the impact of backoff misbehaving nodes in ieee 802.11 networks,” in *IEEE International Conference on Communications (ICC).*, may. 2010, pp. 1 –5.

[24] A. Venkatarama, C. Corbett, and R. Beyah, “A wired-side approach to mac misbehavior detection,” in *IEEE International Conference on Communications (ICC).*, may. 2010, pp. 1 –6.

[25] A. Toledo and X. Wang, “Robust detection of selfish misbehavior in wireless networks,” *IEEE Journal on Selected Areas in Communications.*, vol. 25, no. 6, pp. 1124 –1134, aug. 2007.

[26] M. Raya, I. Aad, J.-P. Hubaux, and A. El Fawal, “Domino: Detecting mac layer greedy behavior in ieee 802.11 hotspots,” *IEEE Transactions on Mobile Computing.*, vol. 5, no. 12, pp. 1691 –1705, dec. 2006.

[27] P. Kyasanur and N. Vaidya, “Selfish mac layer misbehavior in wireless networks,” *IEEE Transactions on Mobile Computing.*, vol. 4, no. 5, pp. 502 – 516, sep. 2005.

[28] L. Guang, C. Assi, and A. Benslimane, “Enhancing ieee 802.11 random backoff in selfish environments,” *IEEE Transactions on Vehicular Technology.*, vol. 57, no. 3, pp. 1806 –1822, may. 2008.

[29] Y. Wang, P. Chen, Y. Ge, B. Mao, and L. Xie, “Traffic controller: A practical approach to block network covert timing channel,” in *International Conference on Availability, Reliability and Security (ARES).*, mar. 2009, pp. 349 –354.

[30] A. Askarov, D. Zhang, and A. C. Myers, "Predictive black-box mitigation of timing channels," in *Proceedings of the 17th ACM conference on Computer and Communications Security (CCS).* New York, NY, USA: ACM, 2010, pp. 297–307.

[31] "IEEE Std 802.11-2007," 2007.

[32] K. Born, "Psudp: A passive approach to network-wide covert communication," in *Black Hat USA*, 2010.

[33] "Motorola airdefense enterprise," 2010, `http://www.airdefense.net`.

[34] "Radiotap," March 2010, `http://www.radiotap.org`.

[35] V. Giri and N. Jaggi, "Mac layer misbehavior effectiveness and collective aggressive reaction approach," in *IEEE Sarnoff Symposium.*, apr. 2010, pp. 1 –5.