2011 Eighth IEEE International Conference on Mobile Ad-Hoc and Sensor Systems

Covert DCF: A DCF-Based Covert Timing Channel in 802.11 Networks

Russell Holloway
Security Researcher
Counter Threat Unit
Dell SecureWorks
Atlanta, Georgia 30342
rholloway@secureworks.com

Raheem Beyah
Communications Assurance & Performance Group
School of Electrical & Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332
rbeyah@ece.gatech.edu

***Abstract*—Covert communications have been used for many decades. Accordingly, when digital communications moved to the forefront it was natural that covert channels be proposed to operate over these networks. Covert channels are general purpose transmission mediums that can be used for good (e.g., an additional layer of security) or bad (e.g., to conduct various proximity-based attacks in wireless LANs). However, their use has been limited as a result of their low throughput. One area that is promising for covert channels is wireless networks. Specifically, those that employ 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. In this paper, we propose a relatively high bandwidth covert timing channel for 802.11 networks (Covert DCF). We exploit the random backoff in the distributed coordinated function (DCF), used to avoid collisions, to provide cover for our covert timing channel. Covert DCF provides significant improvements over other recent covert channels in the area of throughput, while maintaining high accuracy and remaining undetectable. We are able to covertly achieve throughput of 1800 bps while maintaining 99% accuracy. This throughput is approximately 17 times faster than that of current covert timing channels. Covert DCF is robust in that it can adapt to various network conditions.**

***Keywords*-covert channel, MAC misbehavior, steganography, 802.11 DCF, wireless LANs**

I. 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 new technology and systems introduced with some form of security included, at least at the most basic level. Unfortunately, no matter how much security is added, 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 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 paper, we focus on covert timing channels.

The timing channel we propose is a wireless covert channel that allows for a significantly higher bandwidth covert communication in comparison to other covert channels. This could facilitate various proximity-based WLAN specific attacks. For example, one nefarious use of this channel would be to covertly spread wildfire worms. Wildfire worms spread solely over WLANs by using mobile nodes and overlapping AP cells for inter-AP malware propagation [2]. The authors of [2] illustrate that if a wildfire worm is carefully crafted, it can infect all the vulnerable wifi-connected computers in 80% of APs (this represents thousands of hosts) in the various areas they considered in their work.

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 [3]. 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. However, a sender can intelligently select the CW for each outgoing frame, making it appear random over time. 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 [4]. What we illustrate in this paper, is that not only can a node improve throughput using MAC misbehavior, but it can also send messages covertly.

This scheme can easily be extended to other distributed

978-0-7695-4469-4/11 $26.00 © 2011 IEEE
DOI 10.1109/MASS.2011.60

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 paper is organized as follows. In Section II, we present related work. A brief overview of the 802.11 DCF is presented in Section III, with an overview of our proposed covert channel following in Section IV. Section V discusses optimizing throughput by adjusting our codebook and baud rate, accuracy by analyzing two different network cases, and covertness by avoiding current detection schemes. Section VI presents Covert DCF's design in detail, including the fitting of our traffic to any known empirical distribution. We present our analysis and results in Section VII. Lastly, Section VIII presents our conclusions and future work.

II. Related Work

In order to fully implement a covert channel as effectively as possible, we must analyze other covert channel designs and also detection and prevention methods of these channels. In this section, we will look at previous storage channels and timing channel schemes. Later in Section V, we present additional details on detection and prevention methods.

There have been several designs of covert channels. Previous research has introduced a good number of storage channels, with recent research covering a wider range of protocols. In [5], the authors present a scheme to encode a covert message by binning messages based on length with each bin representing a different symbol, followed by sending normal traffic messages from respective bins as needed. Khan et al. present a method to encode a message using multiple active network connections, where incoming traffic on one connection has one meaning, and incoming traffic on another connection means something different [6]. Ji et al. present an Ethernet covert channel using ARP broadcasts, where messages are represented by the last ω bits of the target IP address [7]. In [8], the authors demonstrate a covert channel by placing messages into the B frames of MPEG-2 streaming video files. A few papers have also been written on 802.11 covert channel schemes. One method we presented in [9] 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 initialization vector fields in the header was introduced in [10]. Our research differs from these previous methods in that we propose a timing channel instead of a storage channel. Further, our throughput far exceeds those of the previous schemes.

Several covert timing channels have been introduced as in [11]. 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 [12]. 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 [13]. 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 indistinguishable scheme. While being computationally indistinguishable can help hide the covert message, the extremely low throughput detracts from the appeal of this method.

Some previous schemes can stay covert well with very little throughput. Other schemes have higher throughput yet are easily detected. In this work, we develop an 802.11 covert channel that optimizes throughput, accuracy, and covertness as necessary for intended usage. We are able to achieve higher throughput than previous timing channels, fit our traffic to any empirical distribution on a given network, and achieve high accuracy as well.

III. 802.11 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 [14], but below we discuss the essential details necessary for an understanding of our covert channel.

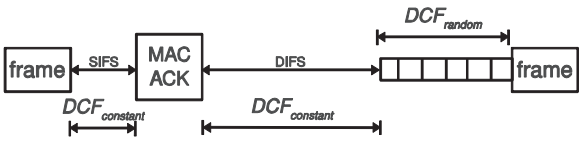


Fig. 1. Transmission of a packet using the DCF

Before a station may transmit, it must sense whether or not the network is busy. 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

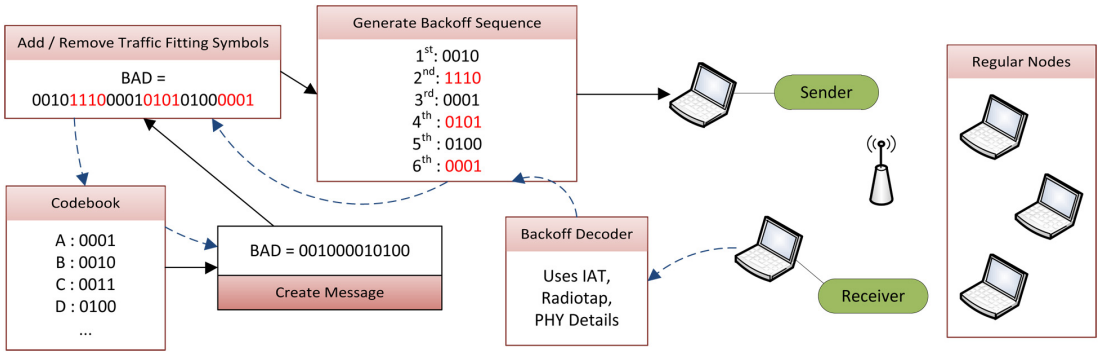


Fig. 2. High level view of Covert DCF

(DIFS), in addition to a random period of backoff time, before transmission.

Second, 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 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. 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 section, 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.

IV. Proposed Scheme

Covert DCF is possible because of the random backoff that is required to take place when sending new packets. By taking control of this random backoff (DCF_{random} in Fig. 1), 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 the set $C = \{s$ is a bit string of length $l(s)\}$ to backoff values. For example, we may let $C = \{000_2, 001_2, 010_2, ..., 111_2\}$ 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 Sections V and VI.

Fig. 2 illustrates a high level view of a simple case using our covert channel. In this example, we let $C = \{0000_2, ..., 1111_2\}$. The sender chooses to send the phrase *BAD*. The sender looks up the letters *B, A,* and *D* in the codebook and combines the associated values to create a sequence. From there, we intermix our sequence values with *traffic-fitting symbols*, a process which is explained in more detail in Section VI. This process allows us to fit our distribution to that of the network in order to remain covert. Once the final sequence is created, the sequence is converted into a series of backoffs which are used in place of the random backoffs for each packet transmission. The decoding steps simply perform the process in reverse, as illustrated by the dashed arrows in Fig. 2.

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, which in turn limits the time in which we perform misbehavior thus lessening chances of detection. 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. Each state change allows the encoding of one symbol. In the context of this work, state changes occur between packets, so the number of state changes is roughly the number of packets per second. Alternatively, we can change the cardinality of C, $|C|$, which represents the number of symbols in C. If we let $l(s)$ be the number of bits in element $s \varepsilon C$, then throughput can be calculated as $bps = \beta * l(s)$ for our channel. Section V discusses optimizing throughput in more detail.

Accuracy is another important aspect to consider. Without high accuracy, the covert channel may not be very useful. We consider accuracy in more detail in Section V, examining two

network topologies that could exist.

Finally, we must consider covertness. After all, it is covertness which separates our channel from regular overt channels. In addition to addressing throughput and accuracy, we present several methods used to determine covertness in the next section. Then, in Section VI we present our scheme for fitting our traffic to any empirical distribution on a given network.

V. Maximizing Throughput, Accuracy, and Covertness

In this section, we will analyze and discuss how we can maximize throughput, accuracy, and covertness.

A. Throughput

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

We can increase β, which can be represented by packets per second, by minimizing the random backoffs chosen for our symbols. Assume we are on a 54Mbps 802.11g network sending an average payload size of 1125 bytes and we send 4 bit symbols ($|C| = 16$) for our covert channel. Also let bo_0 be the associated backoff for symbol s_0, bo_1 be the associated backoff for s_1, and so on. We can maximize β by using $bo_0 = 0, bo_1 = 1, bo_2 = 2, ..., bo_{15} = 15$ for each s_i, resulting in an average of 7.5 slots per backoff. By using the average payload size (and header sizes), wireless data rate, and average number of slots per backoff, we can approximate the number of packets sent each second. Using the figures in this example, we find $\beta = 2160$. Alternatively, assume we used $bo_0 = 0, bo_1 = 10, bo_2 = 20, ..., bo_{15} = 150$, with an average of 75 slots per backoff. In this case $\beta = 551$. Since $bps = \beta * l(s)$, the first case leads to a higher throughput.

To optimize $|C|$, assume we choose to fix β to the optimum value for each C as described above. The specific value of β will vary depending on $|C|$, but the optimum value will always be the one where our symbols are represented by the smallest backoffs possible. We calculate the average backoff for each set C as $|C|$ 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 $C = \{s \text{ is a bit string of length } l(s)\}$ 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

$$avg\_trans\_time_{l(s)} = \frac{\sum_{k=0}^{2^{l(s)}} trans\_time_k}{2^{l(s)}} \quad (1)$$

where

$$\begin{aligned} trans\_time_k = & DIFS + bo_k + TxTime \\ & + SIFS + ACKTxTime \end{aligned} \quad (2)$$

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

$$bo_k = k(slot\_time) \quad (3)$$

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

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 C such that $|C| = 2^{l(s)}$.

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

$$tpb_{l(s)} = \frac{avg\_trans\_time_{l(s)}}{l(s)} \quad (4)$$

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

$$bps_{l(s)} = \frac{1\,000\,000 \mu s}{tpb_{l(s)}}. \quad (5)$$

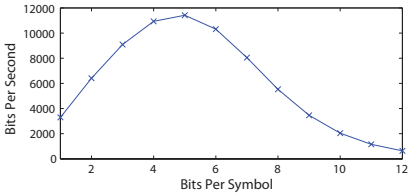


Fig. 3. Theoretical throughput

In Fig. 3, we see an initial increase in throughput as we increase $|C|$. 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 $|C|$. This exponential increase in $|C|$ 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 C. 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 [13].

We can theoretically obtain bit rates far greater than introduced in other covert timing 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 paper, we can maintain high accuracy, covertness, and high throughput at the same time in many cases.

B. Accuracy

When considering accuracy, we must consider how many nodes are sending on the wireless network and how much error we allot ourselves 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 sending node. If this is the case, the sending 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 sending node.

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

Case 2. *multiple senders:* In this case, we must account for the fact that other nodes may use the medium, thus causing the sending node to delay transmission. Since wireless LANs are essentially a broadcast network, the receiver can observe all traffic on the wireless medium, either by positioning itself near the AP or using a high-power 802.11 antenna. If the receiver senses a frame sent by another node, then it can simply disregard the information and adjust its IAT timer accordingly.

Fig. 4 shows the two situations visually by marking where nodes are located in each case.

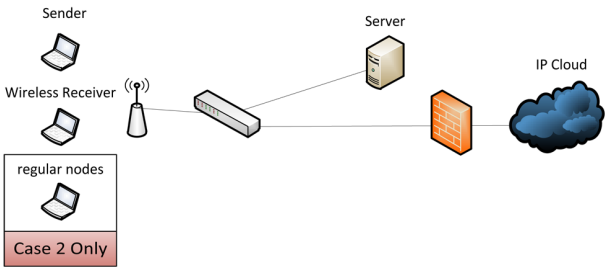


Fig. 4. Diagram illustrating each case

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 default scheme with our chosen CW values, we will see overlap between symbols. For example, if symbol s_1 is set to $CW = 8$ and s_2 has $CW = 3$, then the standard MAC protocol would select $bo_1 \epsilon [0, 8]$ and $bo_2 \epsilon [0, 3]$. This results in 44% chance that s_1 and s_2 overlap in the range of $[0, 3]$. 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 number of sending 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 $\mathcal{A} = \{C$ is a unique set of consecutive bit strings$\}$. As an example, rather than use the higher throughput values $bo_1 = 0, bo_2 = 1, bo_3 = 2$, and $bo_4 = 3$ for symbols, we could use $bo_1 = 0 - 9, bo_2 = 10 - 19, bo_3 = 20 - 29$, and $bo_4 = 30 - 39$.

It is also worth mentioning that the use of error-correcting codes before sending a message is another option for improving accuracy [15]. While this may cause a slight performance hit, it can lead to greater accuracy. Our channel can operate on bits directly, so it is perfectly acceptable to feed it huffman encoded data, compressed data, or use any other encoding technique for that matter.

C. Covertness

Lastly, we want to maximize our covertness. There are several angles from which we can analyze covertness including: observe throughput degradation from several viewpoints, utilize classic MAC misbehavior detection schemes, consider techniques that seek to prevent covert channels, and consider traditional wireless intrusion detection systems (WIDS). We present a description of each below, and evade each to illustrate the stealth of Covert DCF.

Covertness is often evaluated based on throughput or traffic distribution. In [13], 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 [16].

There are also some 802.11 specific detections that we wish to avoid. Because we are performing MAC misbehavior on our sending node, we need to avoid MAC misbehavior detection schemes. Rong et al. [17] 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 [18]. We presented a scheme using a Naïve Bayes classifier and the IAT in [19]. In [20], 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.

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 [21], [22]. These schemes usually add some delay or padding to traffic on the network, and they are usually situated on either the kernel layer to mitigate covert channels created by processes, or on routers or mitigation boxes somewhere along the route. However, kernel layer mitigation techniques would not stop Covert DCF since it is implemented at the MAC layer. Also, adding a mitigation box is infeasible when working solely on the wireless segment due to the broadcast nature of 802.11 networks.

Similarly, we need to be able to bypass wireless intrusion detection systems (WIDS) and wireless intrusion prevention

systems (WIPS). Consider Motorola AirDefense Enterprise [23], which is a popular commercial WIPS solution. Currently, AirDefense Enterprise provides down to the minute granularity with regards to traffic data point collection. Thus we need to ensure that our sending node's data points average out to match those of regular traffic over any given 60 second window.

In this paper, we perform the following tests for covertness as seen in previous academic works:

1) Throughput changes
   a) sender throughput gain
   b) legitimate node traffic degradation
   c) network throughput change
2) Visual distribution matching
3) Two-Sample Kolmogorov-Smirnov test

In all of the above cases, we use data and samples that fit a 60 second window in order to align our academic covert tests with the capabilities of current industry WIDS / WIPS (this is an adjustable parameter should the granularity change). These tests allow us to assess our covert channel both visually and quantitatively within the capabilities of WIDS / WIPS and academic detection schemes. We present the details for this process in the next section.

VI. Covert DCF Protocol Design

In this section, we will present our protocol used for sending and receiving. In order to use Covert DCF, we must make the following assumptions:

- sender and receiver both can obtain information on current network status such as packet size distribution, traffic distribution, average throughput, and PHY characteristics. All of this can be calculated on-the-fly.
- sender and receiver have agreed upon a codebook (5 bit symbols, 6 bit symbols, etc.) and traffic mix ratio.
- receiver can hear all communication on the wireless network, either by positioning near the AP or using a cheap high-gain antenna.

These assumptions will be mentioned in more detail as we present the protocol and are in line with assumptions made by other timing channels in that they also require prior communication between sender and receiver.

A. 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. At a minimum, we must capture as many packets as we intend to send in order to create our traffic-fitting symbols properly. 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 [24].

Next, the sender will take the bit sequence that represents the covert message and encode it into the proper delay sequence based on a pre-determined codebook.

Once the sender has the initial 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 that will be sent and ignored, solely used for distribution matching. Algorithm 1 below presents our scheme for inserting traffic-fitting symbols.

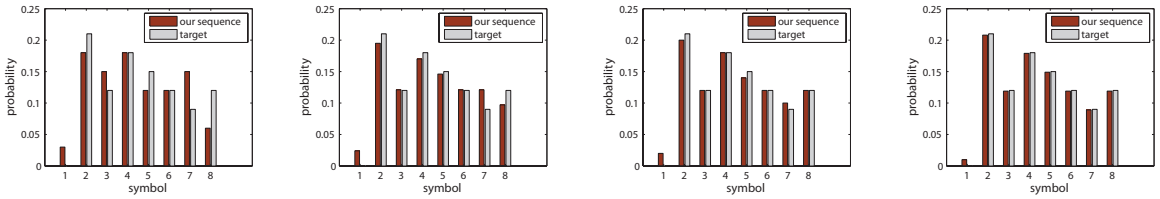
Algorithm 1 Sequence Adjustment

```
for all windows do
    t ⇐ training_window
    m ⇐ message_window
    for all v in t do
        t_prob[v] ⇐ COUNT(v) / SIZE(t)
    end for
    n ⇐ m
    while SIZE(n) < window_size do
        for all v in n do
            n_prob[v] ⇐ COUNT(v) / SIZE(n)
        end for
        largest_diff ⇐ MAX(t_prob[v] − n_prob[v])
        if largest_diff > 0 then
            PUSH n, ld_value
        else
            for all p in n_prob do
                if p > 0 then
                    PUSH n, p_value
                end if
            end for
        end if
    end while
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 overall 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.

To step through Algorithm 1, let t be initialized to the training window from sample traffic and m to the message window from the traffic we which to send. We also initialize a new window, n, to the message window and will modify this window as we progress through the algorithm. Let v represent the individual values within the windows. First, we calculate the probabilities from sample data and store them in t_prob. Next, we loop through the new window in it's entirety, calcuating the current probabilities and storing them in n_prob. We find the largest probability difference between the new window and training window, storing it in



(a) original comparison (b) after two rounds (c) after 4 rounds (d) after 8 rounds

Fig. 5. Distribution fitting example using traffic-fitting symbols

$largest\_diff$. If this is a positive value, then we must append that largest difference value, ld_value, to the new window. If negative, we append the other values associated with a positive difference p, represented as p_value.

Fig. 5 demonstrates a simplified version of this process. Initially, symbol s_7 has the largest percentage difference. Our sequence has a larger probability than the target data. Since we cannot remove symbols from our data, we increase the population size by increasing the probabilities of *other symbols*, thus decreasing the probability of s_7. We only add to symbols which have a lower probability than the target probability. Fig. 5(b) shows the result after two rounds.

In round 4, s_8 has the largest probability difference. Here, our sequence probability is lower than the target probability. We can simply add more of s_8 to the population to adjust this. Fig. 5(c) shows the result after four rounds.

This process continues, always adjusting the symbol with the largest probability difference in each round. In this example, our sequence is fairly close to the target sequence after only 8 rounds of adjustments, as shown in Fig. 5(d).

When inserting our traffic fitting symbols, we do so using a position based method. That is, we may send 1 message symbol per 3 traffic-fitting symbols, resulting in a 25% message:traffic-fitting symbol mix ratio. This ratio must be known on both ends so that the proper packets will be thrown out upon decoding. If the ratio needs to change due to a change in network conditions, this would need to be communicated to the receiver as well, either through the covert channel itself or using a side channel.

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 [13]. 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. As mentioned earlier, current WIDS implementations have a granularity of 60 seconds, so they will also not be able to detect abnormalities within the backoffs since we average them out over a 60 second window.

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

To send our newly created covert sequence, we step 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.

B. Covert DCF Receiving Protocol

The decoding node listens for all packets on the network and can store them in a packet capture file using standard tools such as Wireshark or Tcpdump. All calculations can be performed against the capture file, thus no modification is typically needed of the decoding node. Knowing the MAC 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.

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 2.

Algorithm 2 Symbol Decoding

Require: $prev\_recv\_time \Leftarrow 0$
Ensure: list of decoded symbols
for all packets received **do**
  $curr\_recv\_time \Leftarrow NOW$
  $IAT \Leftarrow curr\_recv\_time - prev\_recv\_time$
  $bo\_time \Leftarrow IAT$ {initialize}
  $bo\_time \Leftarrow bo\_time - difs\_time$
  $bo\_time \Leftarrow bo\_time - \frac{rcvd\_pk\_size}{rcvd\_frame\_drate}$ {TxTime}
  $bo\_time \Leftarrow bo\_time - sifs\_time$
  $bo\_time \Leftarrow bo\_time - \frac{ack\_pk\_size}{rcvd\_frame\_drate}$ {AckTxTime}
  $decoded\_slots \Leftarrow \frac{bo\_time}{slot\_time}$
  if $remote\_addr = sender\_addr$ **then**
    $decoded\_slots \Leftarrow decoded\_slots + offset$
    $offset \Leftarrow 0$
    print $decoded\_slots$
  else
    $offset \Leftarrow offset + decoded\_slots$
  end if
  $prev\_recv\_time \Leftarrow curr\_recv\_time$
end for

The receiver can 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 resets $offset$ 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.

VII. Evaluation of Covert DCF

To test Covert DCF, we used OPNET to simulate our covert channel. Like other papers, we chose simulations over an experiment due to the fact that currently most network interface cards do not allow tuning of several MAC parameters[1] via software, instead using firmware for these parameters [25].

Our simulation consisted of a wireless segment consisting of our sending node, one receiving node, one access point, and 15 regular wireless nodes. The receiving node may be positioned near the AP or use a cheap high-gain antenna in order to see all traffic on the wireless segment, which would prevent the hidden node problem from causing calculation errors due to missed traffic. The AP was connected to a switch, which also had connections to Ethernet servers and the IP cloud.

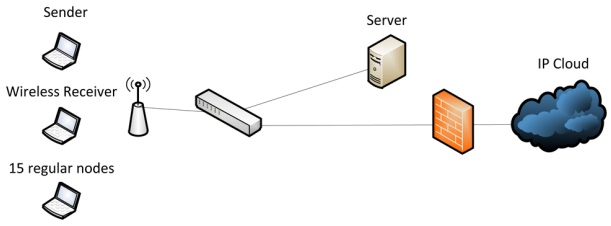


Fig. 6. OPNET Network Layout

For our simulations, we chose to use the 802.11g PHY characteristics, since it is both widely deployed 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 high data rate covert timing channel. SFTP can also provide the back to back traffic.

We modified the wlan_mac process model in OPNET in order to test our channel. 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 reads a message from a standard text file, and using the steps as described in the previous section, created the covert sequence. The receiving node was 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 $|C|$ up to $l(s) = 12$.

A. Throughput

Our simulations show that we are able to obtain high throughput using our channel. Fig. 7 shows a comparison of our theoretical throughput as presented in Section V, and the simulated throughput from sending an ASCII file in DEFLATE compressed and CAST5 encrypted versions. We can see that our channel performed similar to our expected performance.

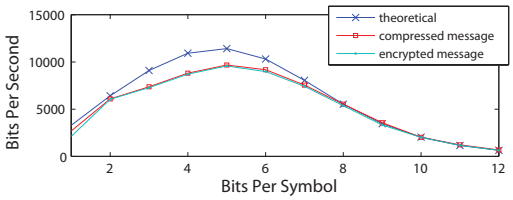


Fig. 7. Covert channel throughput comparison (ASCII text)

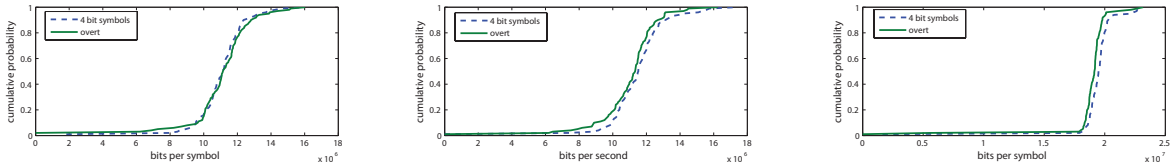
B. Covertness

As mentioned in Section V, we will analyze the covertness of our channel by looking at throughput changes, backoff distribution matching, and two-sample Kolmogorov-Smirnov tests.

1) Throughput Changes: If we do not attempt to control our covertness and aim solely for maximum throughput, we will decrease network performance of other nodes, similar to performing MAC misbehavior on the network. However, if we are smart about our usage of the channel, we can make it much more difficult to detect, as seen in Fig. 8. By embedding some *traffic-fitting symbols* as seen in Section VI, 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 $8000bps$ with 85% accuracy. We show how to raise accuracy to 99% shortly. It is up to the user to determine how important throughput, accuracy, and covertness are for the channel.

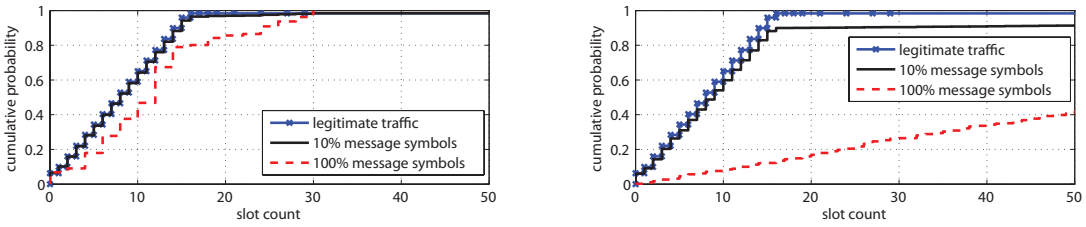
2) Backoff Distribution Matching: We can also compare our traffic distribution to that of a legitimate node as seen in [13]. If the distributions are similar, then it is possible that the traffic was generated using the same method. Fig. 9 compares the empirical distributions of calculated backoffs from our covert node to those from a legitimate node. We compared traffic from a legitimate node, our covert sequence with a 10% mix ratio (1 message symbol per 9 traffic-fitting symbols), and our covert sequence with 100% mix ratio (raw

[1]MadWifi allows tuning of the CW_max and CW_min when setting up the transmit queues, but requires a card level reset each time the value is changed. It currently does not allow the user to change the floor value of the random value generator selecting a number of slots to backoff, instead using the value of 0 stored in firmware.



(a) Sending node throughput (b) Legitimate node throughput (c) Network throughput

Fig. 8. Standard channel throughput variations for covert sender



(a) 4 bit symbol backoff (b) 7 bit symbol backoff

Fig. 9. Empirical CDF comparisons of backoff times

message without traffic fitting symbols) for both 4 bit symbols and 7 bit symbols.

In Fig. 9(a), we can see that our covert sequence with a 10% mix ratio is nearly identical to that of legitimate traffic. The 100% mix ratio traffic, while somewhat similar, is still easily detected. This demonstrates that traffic-fitting symbols are required in order to remain covert.

Fig. 9(b) shows that at the 10% ratio with 7 bit symbols, the distributions are again similar. At the 100% mix ratio, the distributions are quite different. This demonstrates that not only does the mix ratio play an important role in distribution matching, but $|C|$ plays an important role as well. If the backoffs generated from the codebook are naturally similar to those of legitimate traffic, then it is much easier to fit the distribution than if they are naturally quite dissimilar. The optimum value for $|C|$ will vary depending on each specific network.

3) Two-Sample Kolmogorov-Smirnov Test: To further our analysis, 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. While we can compare the distributions visually, performing a K-S test allows us to obtain quantitative results and automate the process. The K-S test generates a test statistic representing the distance between two distributions and allows one to accept or reject a null hypothesis based on the results. For the K-S test, we use a significance level of $\alpha = 0.05$ with the null hypothesis being that the two samples are from the same distribution. If α is greater than p, the probability that the two curves come from the same distribution, then we reject the null hypothesis.

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 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. While viewing the eCDF graphs, we may or may not have accepted the 7 bit symbol sequence with a 10% mix ratio. A K-S test quantifies this comparison, and in both cases, we must reject the null hypothesis at the 5% significance level.

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 congestion, 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, obtaining 1890 bps.

C. 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.

VIII. Conclusions and Future Work

In this paper, 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. We were able to obtain over 8000 bps throughput with approximately 85% accuracy, or by slightly modifying our coding we were able to increase accuracy to over 99% while maintaining a throughput over 5000 bps. Adding in covertness drops our throughput down to $1800bps$, but we are still able to maintain over 99% accuracy at this rate while operating in a more covert fashion. In all cases, the network was under moderate load from regular nodes as well.

In comparison with other covert channels, Covert DCF offers a good alternative when the covert channel is to be used on the wireless LAN.

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.

Acknowledgements

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

References

[1] "Common methodology for information technology security evaluation," July 2009, `http://www.commoncriteriaportal.org`.

[2] P. Akritidis, W. Y. Chin, V. T. Lam, S. Sidiroglou, and K. G. Anagnostakis, "Proximity breeds danger: Emerging threats in metro-area wireless networks," in *In Proceedings of the 16 th USENIX Security Symposium*, 2007, pp. 323–338.

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

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

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

[11] 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.

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

[13] 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.

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

[15] J. Wu, Y. Wang, L. Ding, and X. Liao, "Improving performance of network covert timing channel through huffman coding," *Mathematical and Computer Modelling*, vol. In Press, Corrected Proof, pp. –, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/B6V0V-526MS25-1/2/b3178658ce252356baba2311768c9834

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

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

[25] J. Choi, A. W. Min, and K. G. Shin, "A lightweight passive online detection method for pinpointing misbehavior in wlans," *IEEE Transactions on Mobile Computing*, vol. 99, no. PrePrints, 2010.