Frequency Hopping Pattern Recognition Algorithms for Wireless Sensor Networks

Min Song, Trent Allison
Department of Electrical and Computer Engineering
Old Dominion University
Norfolk, VA 23529, USA

Abstract

Frequency Hopping Spread Spectrum (FHSS) as a communications technique has also been considered as one of security mechanisms to provide a good level of security against eavesdroppers in wireless sensor networking environment. We claim that the version presently used in wireless sensor communications does not yield a high level of security. In this paper, we develop algorithms to break the FHSS hopping patterns. Simulations are performed to verify the algorithms. Results suggest that wireless sensor networking is not secure in its current form.

1. INTRODUCTION

In Frequency Hopping Spread Spectrum (FHSS), the transmitter broadcasts on one frequency for a small amount of time then switches to another frequency using a known switching algorithm call a hopping code or hopping pattern. The receiver knows the same hopping code so it is able to slide the code past the incoming signal until it synchronies with the sender. Once they are synchronized the transmitter and receiver follow the hopping code to switch frequencies and communicate. The resulting transmission is spread over a large frequency range and therefore appears as noise to other receivers unless they know (or can decipher) the hopping code. This also allows for many devices to share the same frequencies. This greatly improves utilization opposed to having dedicated frequencies for each user. The technique has been used in many applications including wireless sensor networks and US military satellites as a secure and efficient data transfer scheme [3]. The Global Positioning System (GPS), as a particular example, also employs frequency hopping. It uses many more channels than the one evaluated in this paper. During war time the security of the GPS can be increased. This wartime mode uses frequency hopping and is designed to take about 11 years to break. To further ensure protection, the code is changed everyday [4, 6].

The use of FHSS can allow for fairly secure data communications. However, no system is completely secure. This is especially true for wireless sensor communications which are easily vulnerable to eavesdroppers [2]. This paper will explore different techniques to decipher the hopping codes used in wireless sensor networks or WLAN. The system evaluated in this paper is loosely modeled after the wireless communication standard 802.11 [5]. It uses the lower 79 channels of the Industrial, Scientific, and Medical (ISM) band. This band starts at 2.4 GHz with each channel being 1 MHz wide. The hopping code for this example is characterized by the following equation:

$$C(d_i, f_i) = \sum_{i=1}^{I} d_i f_i, \quad I \le 79$$

where f_i is a frequency being used with the corresponding dwell time d_i. The frequencies corresponding to f_i can hop around in any order (i.e., the frequency corresponding to f_{i+1} can be larger or smaller than the frequency corresponding to f_i) but frequencies cannot repeat in a given hopping code. The dwell time at each frequency must be smaller than 400 milliseconds. Due to the constraints of modern synthesizers, the smallest amount of time required for a receiver to switch channels is about one millisecond. Therefore one millisecond will serve as a lower limit of the dwell time. The period of the hopping code and its maximum, minimum, and average values are defined as follows:

$$T = \sum_{i=2}^{I} d_i, \quad 2 \le I \le 79$$

$$T_{\min} = 2 \text{ ms}, d_1 = d_2 = 1 \text{ ms}, I = 2$$

$$T_{\max} = 31.6 \text{ s}, d_i = 400 \text{ ms}, I = 79$$

$$T_{\text{ave}} \approx 8 \text{ s}$$

Given these definitions, techniques for an eavesdropper to derive the hopping code needs to be analyzed and compared. This will prove that frequency hopping in 802.11 does not provide secure communications.

The rest of the paper is organized as follows. Section 2 presents four algorithms to decipher the hopping codes. The numerical results and the comparison of the four algorithms are provided in Section 3. Section 4 introduces the simulator design and analyzes the simulation results. Finally, Section 5 concludes the paper.

2. ALGORITHMS DESIGN

Four algorithms were explored to decipher the hopping codes. The first brute-force method attempts to decode the signal by using every possible hopping code. The second approach observes one frequency at a time to determine the hopping sequence and will be referred to as sequential scanning. The third method monitors each frequency simultaneously and is denoted as parallel scanning. A final effort was made to create a hybrid algorithm using concepts from the first three techniques. Each algorithm is analyzed theoretically and the second and third are simulated using C++ software.

2.1 Brute-force algorithm

The brute-force technique assumes discrete dwell times with a minimum increment of one millisecond at each frequency. Otherwise, the number of possible hopping codes would be infinite due to the limitless variations of continuous dwell times. Therefore, there are 400 possible dwell times for each channel with many different frequency combinations using up to 79 channels in any order with no repeating channels. The eavesdropping receiver takes each possible hopping code and tries to use it to decode the incoming signal until the correct code is found. A modified version of the brute-force method keeps track of what frequencies are known to be used and known to not be used. In this way the set of possible hopping codes can be reduced and the algorithm converges quicker.

2.2 Sequential scanning algorithm

In sequential scanning, there is a single receiver whose signal detection is connected to a computation unit. The computation unit is responsible for running the sequential scanning algorithm and determining the hopping code. The sequential scanning algorithm has the receiver wait to detect a transmitted signal on one channel at a time. The duration of the wait time is the maximum time it takes to repeat the hopping code sequence (referred to as the hopping code period). If a signal is not received during the maximum hopping code period then that channel is marked as not used. Another channel is then chosen and the receiver listens for the maximum repeat time again. If the receiver detects the transmitted signal, it measures the dwell time at that frequency then continues to monitor that channel until the transmitted signal is detected again. The time between transmissions is recorded as the new hopping code period. An internal counter is reset at the beginning of the repeated transmission and the start time for that frequency is recorded as zero. This point is also considered the starting time of the hopping code and the counter is reset to zero after an amount of time equal to the known period has passed. In this way the receiver stays synchronized with the transmitter based on the first frequency found.

After the receiver detects the first channel it chooses another channel to monitor and waits the known hopping code period for a detected transmission. If the channel does not become active, it is marked as unused and another channel is monitored. If the channel does become active then the start time is recorded based on the clock and the dwell time is measured. After the start time and dwell time is recorded, a new channel is selected. This process continues until the total dwell time of the known channels equals the known period. Since the start times are recorded relative to the synchronized clock, the order of the known frequencies can be determined by simply sorting based on start time. After sorting the frequencies, the receiver initiates the hopping code based on the synchronized clock, the derived order of the frequencies, and the measured dwell time for each frequency.

2.3 Parallel scanning algorithm

In parallel scanning scheme, there is a receiver for each possible channel used in the hopping code. All of the receivers provide a signal detect status to the computation unit. The unit is responsible for running the parallel scanning algorithm and deriving the hopping code. Once the code is determined, the first receiver can be used to decode the incoming signal.

The parallel scanning approach has a receiver for each of the 79 channels. The receivers monitor all the channels simultaneously and the computation unit records all transmission start times based on a common clock. Once the first channel repeats a transmission, the hopping code can be determined. The start times that were recorded from each receiver is used to sort the frequencies and calculate the dwell times for each frequency. In this way, the parallel scanning approach can determine the hopping code in an amount of time slightly larger than the period of that hopping code.

2.4 Hybrid algorithm

The final hybrid algorithm uses a set of parallel receivers (less than 79) that switch through the possible channels. The hopping code is determined in the same way as in the sequential scanning algorithm however it will benefit from monitoring multiple channels simultaneously. All of the channels being scanned are connected to a central computing unit that ensures different receivers do not repeatedly attempt the same frequencies. The central computing unit also keeps the synchronized clock based on the first receiver that detects a transmission. The hopping code can be computed by sorting the start times once the hopping code period, channels used, channel start times, and channel dwell times that have been determined.

3. NUMERICAL RESULTS

3.1 Brute-force algorithm

The brute-force approach is expected to be time consuming and hardware intensive. There are a large number of possible hopping codes to try. This is due to there being up to 79 hops in a hopping code and 1 to 400 millisecond dwell time between each hop. This method is also limited because a discrete dwell time must be assumed. In a physical application this would be very time consuming if each possible hopping code is attempted in a serial fashion. If it were possible to test all the hopping codes in parallel then it would prove to be the fastest technique.

In order for the parallel brut force method to synchronize with the transmitter it will require half a code period on average. This is based on where the algorithm starts compared to the received signal. The physical hardware to support every combination being testing in tandem would be outlandish. Overall this approach will not be viable. The following is an equation representing all the possible hopping codes.

$$PHC = \sum_{j=2}^{79} \frac{400^j \times 79!}{j(79-j)!} \approx 10^{322},$$

$$PHC_{j=3} = \frac{79 \times 400 \times 78 \times 400 \times 77 \times 400}{3}$$

The equation is based on there being 79 ways to choose f_1 multiplied by 400 ways to choose d_1 multiplied by 79 ways to choose f_2 and so on. The equation is divided by j because of the circular nature of the hopping code. This comes to a huge number. Obviously, trying each possible code would take too long.

A modified brute-force method could keep track of used and unused frequencies to reduce the set of possible hopping codes. This would be done by trying the first frequency in a hopping code. If the frequency is unused then denote it as such and remove all hopping code combinations that use that frequency. If the frequency is detected then mark it as used and try other combinations with that frequency. In this way the possible hopping codes will converge on a result based on the following equation. This approach would still take a large amount of time and is not feasible. The modified brute-force possible hopping codes is:

$$PHC_{\text{remaining}} = \sum_{j=n}^{m} \frac{400^j \times (m-n)!}{j(m-j)!}$$

where m is the number unused frequencies and n is the number of known frequencies.

3.2 Sequential scanning algorithm

The sequential scanning approach will require a small amount of hardware since only one channel is being monitored at a time. The repetitive process will narrow down the hopping code faster than the brut force method but will require a good deal of calculations and iterations. This is a more realistic approach than either brut force method. When there are low numbers of hops the algorithm will suffer from having to wait the max period while looking for a used frequency. When the hopping number is high the period will be determined faster. However, in this case the period will also more than likely be larger and the algorithm will take many repetitions so there will be little benefit. If we consider the average time needed it should follow the equation below. Where the first term is the average amount of time spent trying to find the first frequency plus the second term which is the average time spent eliminating unused frequencies plus the third term which is the average time to find a frequency's dwell time. The average time needed by sequential scanning is:

$$AveTimeSeq = (79 \times 400)\frac{79}{8} + \left(\frac{79}{2} \times \frac{400}{2}\right)\frac{79}{8} + \left(\frac{79}{2} \times \frac{400}{4}\right)\frac{79}{2}$$
$$= 546087.5 \text{ ms}$$

3.3 Parallel scanning algorithm

The parallel scanning technique requires 79 receivers but would determine the hopping code in the shortest amount of time. This method will require slightly more time than the parallel brut force approach but it feasible for implementation. The hopping code will always be determined within one hopping code period plus a maximum of 400 milliseconds. The extra 400 milliseconds is to account for when the sampling began immediately after a 400 millisecond transmission started and therefore the system would have to wait an extra 400 milliseconds to receive that transmission in its entirety. The need for 79 receivers is expensive and the main limiting factor of this approach. The average time needed can be computed with the following equation:

$$AveTimePara = \frac{79}{2} \times \frac{400}{2} + \frac{400}{4} = 8000 \text{ ms}$$

The first term is the average period and the second term is the average amount of time spent waiting for the first frequency. Again, this extra wait time is needed to take in account for when the algorithm starts in the middle of an active frequency.

3.4 Hybrid algorithm

The hybrid algorithm uses the best features of two aforementioned techniques. Since it may not be cost

effective to have 79 receivers, the serial scanning technique is used on multiple channels. This allows for the same algorithm to be run in parallel thus reducing the needed time directly by the number of receivers used. By just adding one receiver the serial scanning algorithm's average time will be decreased by a factor of two. The equation governing the hybrid algorithm's average time is the same as the average time needed for sequential scanning but divided by M receivers. The average time needed by the hybrid scanning algorithm is:

$$AveTimeHyb = \frac{(79\times 400)\frac{79}{8} + \left(\frac{79}{2}\times\frac{400}{2}\right)\frac{79}{8} + \left(\frac{79}{2}\times\frac{400}{4}\right)\frac{79}{2}}{M} = \frac{546087.5}{M} \text{ ms}$$

3.5 Comparison of four algorithms

The brute-force method is mainly discussed to illustrate that there is a very large number of possible frequency hopping codes. Trying each one is futile but the process of analyzing the possibility yields a better understanding of the problem. A simulation of the modified brute-force method would be intriguing. However, the shear number of possible codes makes it impossible on a standard computer.

The serial scanning technique is the main focus of this paper since it is the most feasible implementation. It was realized late in the simulation design that if the serial scanning technique reduced the maximum period time when frequencies were eliminated in the beginning then it would improve performance. For example, the max period starts as 79×400 = 31600 ms. If a frequency is tried and is not used then the max period could be reduced to 79× 400 – 400 = 31200 ms. This would constitute an improvement of 3.6%. The maximum period for sequential scanning would be $T_{\max} = 79\times400 - 400\times u$, where u is the number of unused channels. The average time needed by sequential scanning with new max period is:

$$AveTimeSeq = \left(79\times400 - 400\frac{79}{16}\right)\frac{79}{8} + \left(\frac{79}{2}\times\frac{400}{2}\right)\frac{79}{8} + \left(\frac{79}{2}\times\frac{400}{4}\right)\frac{79}{2} = 526584.4 \text{ ms}$$

The parallel scanning technique is straight forward and the fastest. The drawback is that it requires 79 receivers. This is why the hybrid of serial and parallel scanning would provide a cost effective solution to reducing the time needed to break the hopping code. However, the serial scanning technique alone only takes ~9 minutes on average to decipher the code. This makes the serial scanning technique the most viable solution for breaking hopping codes.

4. SIMULATOR DESIGN AND SIMULATIONS

A C++ program was developed to simulate the sequential scanning and parallel scanning techniques. Fig. 1 illustrates the flow of the simulation software. The gray optional path is typically commented out but provides useful information as to what the original hopping code and derived hopping look like. The main functions are defined as follows.

main() – This is the main driver that invokes the other functions. It iteratively executes the sequential and parallel scanning algorithms on random hopping code inputs. The main() functional also records time needed by the algorithms to break the hopping code. The mean and standard deviation of these data sets are calculated over all the trials and displayed. This function also calls the srand() function to initialize the pseudo random number generator function rand() for later use. The time() function is used as input to srand() to provoke randomness based on the time of execution and insure the same hopping codes are not generated each time [1].

create_code(parameters[], int&) – This function takes in a parameters structure representing the original hopping code. It randomly generates the hopping code to be deciphered by the other algorithms and passes it back to the main() function. It first randomly selects the number of hops which is also passed back to main(). It then randomly selects a frequency and associated random dwell time. The frequencies used are recorded and taken from the random selection possibilities to ensure there are no repeats. This process of randomly selecting a frequency and dwell time is repeated until the hopping code has the randomly selected number hops. The random numbers are selected using the rand() function. Simple modulo was not used to determine the random numbers because it only relies on the lower bit of the rand() output and may not be as random as desired. Instead, the rand() output is multiplied by the range needed then divided by RAND_MAX+1. This eliminates the problem of relying on the low bits of the rand() function and can be offset by simply adding a number to the result [1].

find_serial(parameters[], parameters[], int) – This function is passed the sequential derived hopping code, the original hopping code, and the number of hops in the original code. The original hopping code starting frequency and the place in its associated dwell time is randomly selected. The current frequency, current dwell time, and number of hops are used to traverse the original hopping code. The code is repeated continuously as it would if it were being used by a transmitter.

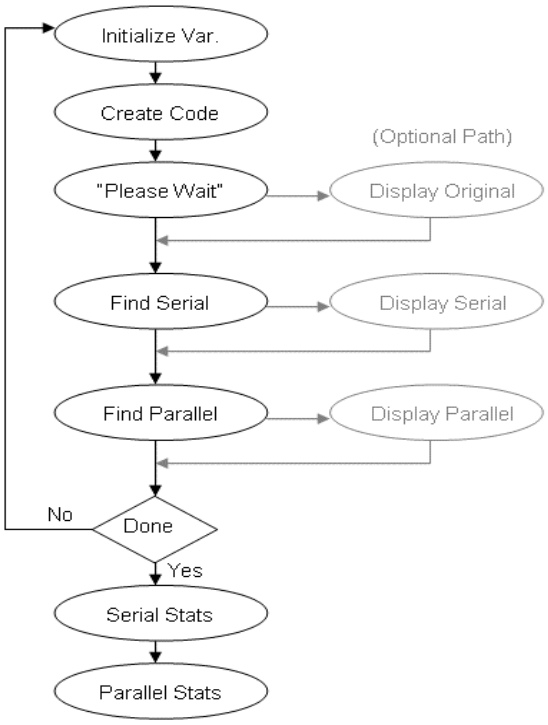


Fig. 1 Simulation program flow diagram

The sequential scanning algorithm is executed using the current state of the original code. The frequencies selected for scanning are chosen randomly opposed to sequentially in a hope that it will help derive the random hopping code faster. Frequencies that have been tried are removed from the random selection to avoid repeating tests. A random frequency is selected to start scanning. If it is detected within the maximum hopping code period (79×400) then it is marked as used, the clock is reset, the start time for that frequency is recorded as zero, and its dwell time is measured. The algorithm then continues monitoring that frequency until the transmission is repeated. This time between transmissions is recorded to be the hopping code period. After this point, the clock is reset after each period to keep it synchronized with the first frequency found. The next frequency is randomly selected and monitored. If it is detected within a period's time, then its dwell time is measured, it is marked as used, and the next frequency is randomly selected.

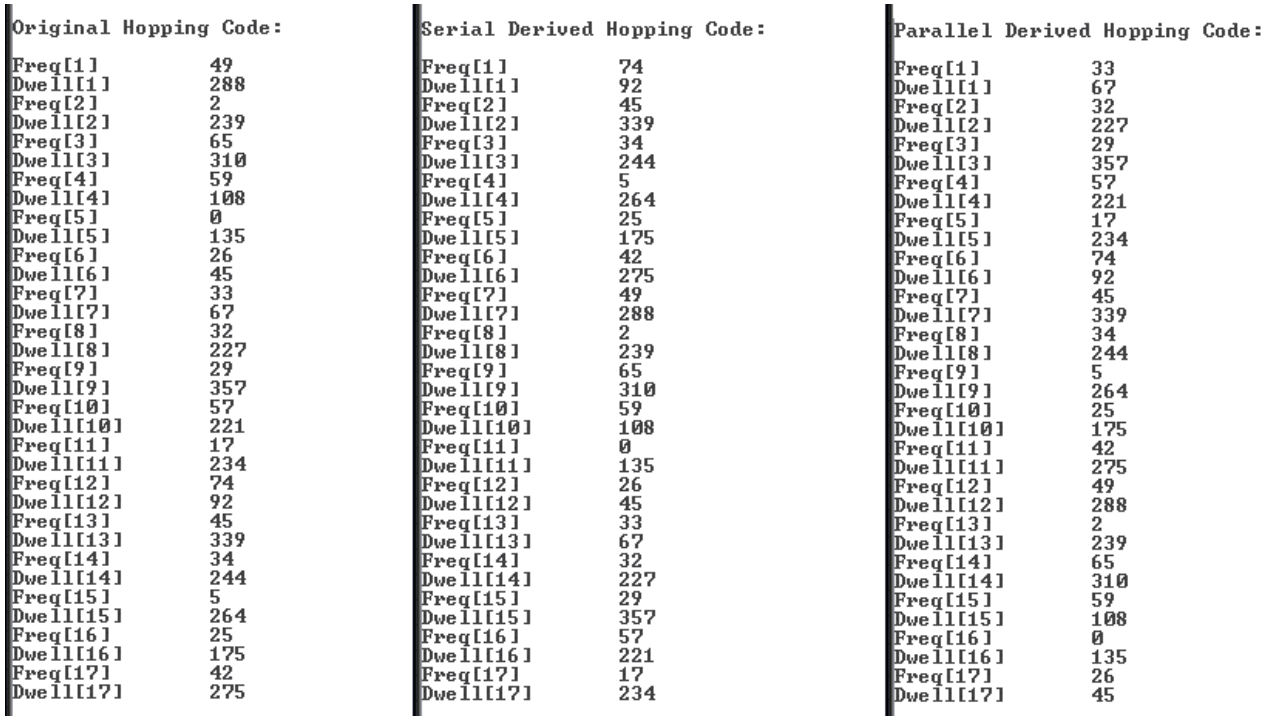
Detection of a signal involves detecting a transition from inactive to active to ensure that the receiver did not start detecting it in the middle of a transmission and thus record an invalid dwell time. If a frequency does not become active in a period's time then it is marked as unused and the next frequency is randomly selected. The known dwell times of the used frequencies are added and compared to the period to determine if all the frequencies have been found. After all the hops are defined, a sorting algorithm is used to copy them in order into the sequential derived structure based on their relative start times. The derived hopping code and the time need to find the code is returned.

find_parallel(parameters[], parameters[], int) – This function behaves similarly to the find_serial() function in that it passes back the parallel derived hopping code and takes in the original hopping code and number of hops. A random starting point in the original hopping code is selected and the state of the hopping code is repeatedly determined as before. A free running clock is started for use in determining start and dwell times. The parallel algorithm looks for the transition from inactive to active for each frequency in the original hopping code. Once that transition is found, the start time is recorded based on the clock value. The dwell time is then calculated by waiting for the transition from active to inactive and subtracting the start time from the present clock time. That frequency and dwell time is the copied into the structure representing the parallel derived hopping code. The entire code has been determined once the first frequency that was detected repeats.

Fig. 2 shows a screen shot of the output from the simulation program. The hopping code shown has 17 hops. The 1st frequency in the serial derived hopping code aligns with the 12th frequency of the original and the 1st frequency of the parallel-derived hopping code matches the 7th frequency of the original. This offset is due to the algorithms starting in a random place with respect to the original hopping code. The circular sequence in each case matches the original indicating that the serial and parallel algorithms are being performed correctly.

Each time the program iterates, the serial and parallel algorithms return the time needed to break the hopping code. These values are averaged to find the mean time and then the standard deviation is calculated. The simulation program was setup to run 1000 trials on the two algorithms. The first test took any random hopping code as input. The second only took codes with 2 to 39 hops. The third used hopping codes with 40 to 79 hops. The results are shown in Table 1.

It is clear that the parallel scanning algorithm surpasses the sequential scanning. The average time needed for the serial scanning algorithm to find the hopping code is approximately 8.3 minutes while the parallel scanning method has an average of 7.9 seconds. The standard deviations are large mainly due to the receiver algorithm not knowing how many hops are used. When codes with a low number of hops are selected, the algorithms must hunt through many unused frequencies which consumes time. When only the upper half of the number of possible hopping codes is used, the overall times increase and the standard deviations are reduced.



```
Original Hopping Code:

Freq[1]         49
Dwell[1]        288
Freq[2]         2
Dwell[2]        239
Freq[3]         65
Dwell[3]        310
Freq[4]         59
Dwell[4]        108
Freq[5]         0
Dwell[5]        135
Freq[6]         26
Dwell[6]        45
Freq[7]         33
Dwell[7]        67
Freq[8]         32
Dwell[8]        227
Freq[9]         29
Dwell[9]        357
Freq[10]        57
Dwell[10]       221
Freq[11]        17
Dwell[11]       234
Freq[12]        74
Dwell[12]       92
Freq[13]        45
Dwell[13]       339
Freq[14]        34
Dwell[14]       244
Freq[15]        5
Dwell[15]       264
Freq[16]        25
Dwell[16]       175
Freq[17]        42
Dwell[17]       275
```

```
Serial Derived Hopping Code:

Freq[1]         74
Dwell[1]        92
Freq[2]         45
Dwell[2]        339
Freq[3]         34
Dwell[3]        244
Freq[4]         5
Dwell[4]        264
Freq[5]         25
Dwell[5]        175
Freq[6]         42
Dwell[6]        275
Freq[7]         49
Dwell[7]        288
Freq[8]         2
Dwell[8]        239
Freq[9]         65
Dwell[9]        310
Freq[10]        59
Dwell[10]       108
Freq[11]        0
Dwell[11]       135
Freq[12]        26
Dwell[12]       45
Freq[13]        33
Dwell[13]       67
Freq[14]        32
Dwell[14]       227
Freq[15]        29
Dwell[15]       357
Freq[16]        57
Dwell[16]       221
Freq[17]        17
Dwell[17]       234
```

```
Parallel Derived Hopping Code:

Freq[1]         33
Dwell[1]        67
Freq[2]         32
Dwell[2]        227
Freq[3]         29
Dwell[3]        357
Freq[4]         57
Dwell[4]        221
Freq[5]         17
Dwell[5]        234
Freq[6]         74
Dwell[6]        92
Freq[7]         45
Dwell[7]        339
Freq[8]         34
Dwell[8]        244
Freq[9]         5
Dwell[9]        264
Freq[10]        25
Dwell[10]       175
Freq[11]        42
Dwell[11]       275
Freq[12]        49
Dwell[12]       288
Freq[13]        2
Dwell[13]       239
Freq[14]        65
Dwell[14]       310
Freq[15]        59
Dwell[15]       108
Freq[16]        0
Dwell[16]       135
Freq[17]        26
Dwell[17]       45
```

Fig. 2 A screen shot of the output from the simulation program

Table 1. Simulation Results with Variable Hops

Possible # of Hops	Serial Mean	Serial St. Dev.	Parallel Mean	Parallel St. Dev.
2 to 79 (all)	495.158 s	177.483	7.941 s	4.662
2 to 39 (lower)	414.652 s	226.267	4.121 s	2.286
40 to 79 (upper)	605.163 s	68.997.7	12.093 s	2.411

5. CONCLUSIONS

We have designed four algorithms to detect the hopping code used in FHSS. Both analytical and simulations results suggest that the hopping code can be easily broken in as little as eight seconds on average given the proper parallel computing equipment. This indicates that FHSS itself does not provide any security. We believe that an unknown number of channels and a completely random hopping pattern (e.g., allowing channels to be repeated in a code and assigning a random amount of lifetime for each channel) are necessary to make the hopping code harder to predict. Furthermore, more sophisticated security protocols must be present at different networking layers to ensure secure communications in wireless sensor networks.

REFERENCES

[1] Jacobs, Bob, C++ Tutorials For Beginners and Cheapskates, 2003, available at http://www.robertjacobs.fsnet.co.uk/random.htm.

[2] M. Song, and S. Wigginton, "Frequency Hopping Pattern Detection in Wireless Ad Hoc Networks," *Proc. of International Conference on Information Technology*: *Coding and Computing*, pp. 633-638, April 4-6, 2005, Las Vegas.

[3] Couey, Anna, How "The Bad Boy Of Music" and "The Most Beautiful Girl In The World" Catalyzed A Wireless Revolution - in 1941, 1997, available at http://people.deas.harvard.edu/~jones/cscie129/lectures/lecture7/hedy/lemarr.htm.

[4] White House Press Release, Statement by the President Regarding the States' Decision to Stop Degrading Global Positioning System Accuracy, Office of the Press Secretary, 2000.

[5] Naftali Chayat, "Frequency Hopping Spread Spectrum PHY of the 802.11 Wireless LAN," http://grouper.ieee.org/groups/802/11/Tutorial/FH.pdf.

[6] Couch II, Leon W., "Digital And Analog Communication Systems," 3rd Edition, Macmillan Publishing Company, 1990.