DECEPTION FRAMEWORK – MULTIPLE INDEXING IN SELF ADAPTATION

By
RELANGI PADMAJA
Master's of Science in Computer Science
Oklahoma State University
Stillwater, OK
2009

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2009

DECEPTION FRAMEWORK – MULTIPLE

INDEXING IN SELF ADAPTATION

Thesis Approved:

Dr. Johnson Thomas

Thesis Adviser

Dr. Xiaolin Li

Dr. Nohpill Park

Dr. A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGMENTS

I thank my advisor Dr. Johnson Thomas for his advice and help during my research, and for his tremendous support. I also thank my parents, brother and sister for all support and encouragement; they always wished best for me. Thanks to Zhu Ling for her help in data collection. I would also like to thank my committee members Dr.Xiaolin Li, Dr.Park and Dr. Subhash Kak, head of department for their encouragement and support.

Most of all, I thank GOD for being with me in every season of my life. I also thank Him for His promise “ That He will never leave me nor forsake me”.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF EQUATIONS

.

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Wireless Sensor Network

Wireless Sensor Networks are used to monitor physical or environmental conditions such as temperature, sound or vibrations co-operatively. These sensors are now finding applications in diverse fields such as in the battlefield, healthcare, space etc. A Wireless Sensor Network (WSN) consists of collections of sensor nodes and a base station. These miniature computers have very basic functionalities with interfaces and components. They are limited in power, computational capability and memory.

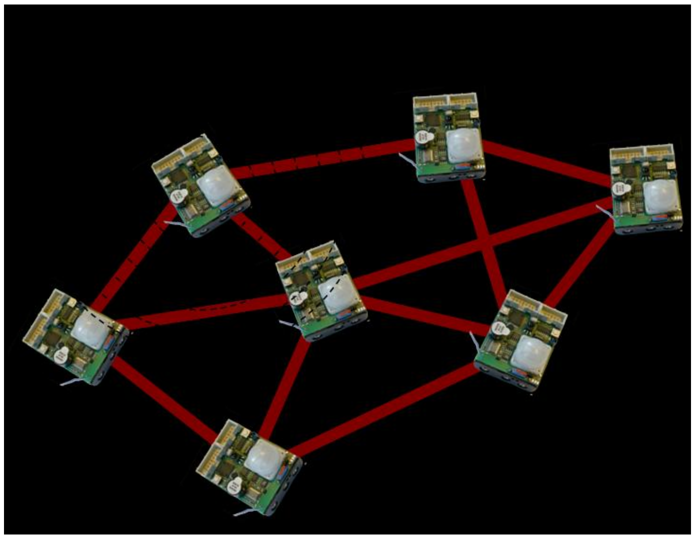


Figure 1 Wireless Sensor Network

1.2 Constraints and Security issues on Wireless Sensor Networks

Wireless Sensor Networks has some constraints. These have limited computational, storage and limited power resources. They are typically deployed in harsh environments and suffer from increased hardware and communication failure rates. These sensors may also be left unattended.

1.3 Security in Wireless Sensor Networks

The resource limitations of sensors mean that complex cryptographic techniques in Wireless Sensor Networks to secure data might be computationally too intensive. Sensor networks are therefore vulnerable to security attacks. Since it is relatively easy for an attacker to break into a sensor network (due to the absence of complex cryptographic techniques), one way of protecting the network is for the sensor nodes to respond to an attacker by sending him false information or by deceiving. Deception will help to manipulate an attacker’s behavior. The objective is to protect the network. For example, if one part of the network is collecting critical data, the objective of deception will be to prevent the attacker from moving to that part of the network. The attacker behavior can be manipulated by sending him data that he expects if it is a sinkhole attack for example, and therefore he will not move to another critical part of the network.

1.4 Deception framework

In deception, the defender or deceiver deceives the attacker by learning the strategy of attack from the attacker and responding to the attacker as anticipated. Besides deceiving the attacker, it also deprives or drains the resources of the attacker. Furthermore, by sending wrong data to the attacker, it secures vital information.

A Deception framework analyzes past attacker behavior and determines the optimum response to deceive the attacker. If an attacker changes his behavior, the network has to adapt to the new strategy of the attacker. We propose an approach to recognize this change in the strategy of an attacker. Once this change is detected the deception framework self-adapts to these changes and modifies its response to the attacker. In this thesis we propose a method called multiple indexing for self adaptation of window size and tolerance value to maximize the performance and accuracy of our deception framework.

In chapter 2, we briefly review the architecture of the Deception Framework. Chapter3 presents the literature review. In chapter 4, proposed approach to self-adaptation is presented. We present an approach to determine the optimum window size and the optimum tolerance value. Chapter 5 presents simulation results and the conclusions are presented in chapter 6.

CHAPTER II

2.1 DECEPTION FRAMEWORK IN WSN

The main objective of the deception framework is to deceive the attacker by providing wrong information to the attacker. We assume that sensor nodes are less powerful then the attacker. When a node has been attacked by a powerful attacker in a network, the deception framework collects the input data from the attacker and analyzes the data. Input data from the attacker is called a request message which is sent from the attacker to the node. During this process, the deception framework will understand the strategy or the pattern of attack.

Once the pattern is recognized the deception framework aims to meet the expectation of the attacker by responding appropriately to the attacker. In this thesis we apply a simple 1:1 request-response pattern. That is, the rate of response will approximately match the rate of the request. The response will contain fake sensor data. Other more complex request-response relationships may be implemented, but this depends on the type of attack. We model a simple sinkhole attack where increasing rate of requests from the attacker will result in increasing rate of responses to the attacker.

Once the request pattern has been analyzed, the defender will predict the response (output data from a node to the attacker) that is expected by the attacker. Once the expected response has been predicted, the deception framework will generate the response to the attacker.

2.2 Design and Architecture

In this section we describe the design and architecture of self adaptation. Once the attacker has been identified, deception takes place in three phases.

1) Initiation of the Environment
2) Deceive
3) Self Adaptation

Figure 2 shows a block diagram of the deception framework architecture.

Phase I: Initiation of an Environment

In the first phase once the attack has been identified, initiation of an environment for deception is prepared by Distributed Deception Agents **(**DDA)**.** DDAs are considered to have more storage memory and are able to handle more complex algorithms. DDAs are the agents in the network which selects the nodes to execute the deception algorithm while the other nodes carry normal communication within the network. The nodes selected to do the deception are called sacrificial nodes. Once the sacrificial nodes are selected, the deception variables are uploaded into the selected sacrificial nodes by the DDA.

Phase II: Deceive

In the second phase, the deception of the attacker begins by analyzing the input data (request message from the attacker). It will predict the future expected request data and its response. Once this is done the deception framework responds to the attacker.

Phase III : Self Adaptation

Self adaptation involves monitoring the attacker behavior, and adapting to the changed behavior of the attacker. As the input data from the attacker is analyzed and compared with the query sequence in phase II, changes in pattern can be identified. Query sequence is the future expected input data which is obtained by finding steady rate of input data. This is used as a reference to compare with the next input data to identify the change in pattern.

Once the change in pattern is identified, in phase III the deception algorithm running on the DDA must change its variables (such as query sequence, window size, tolerance value) to adapt to the new pattern of input data. The query sequence is divided into n number of windows of length window size. Tolerance value is the maximum deviation of input data pattern that is allowed from the query sequence.

When the input data pattern is changed its query sequence length may vary from the previous query sequence length. Hwan Lim, Pak, Kim [3], show that as the length of the query sequence increases its execution time also increases, this is called the window size effect. This degrades the performance of the deception algorithm. Our proposed algorithm has to meet this constraint and increase the performance of self adaptation by adapting variables such as window, and tolerance value to the optimum value.

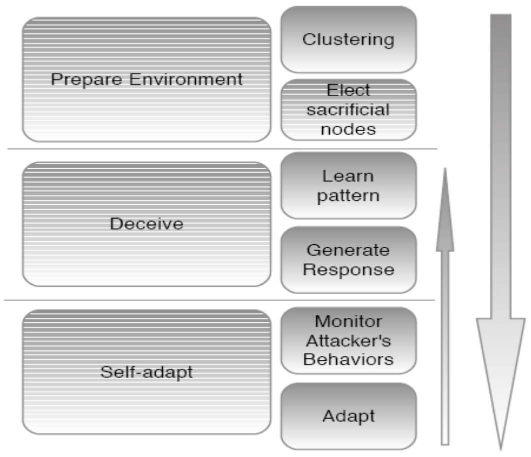


Figure 2 Block diagram of Deception Framework

CHAPTER III

REVIEW OF LITERATURE

In the Deception framework, self adaptation plays a vital role. If the change in the pattern of input data is not detected, the objective of deception may fail Previous work had a fixed window size and fixed tolerance value. The disadvantage with fixed values is performance degradations as the length of the query sequence varies. This is the so-called window size effect [3]. The window size and tolerance value must therefore adapt to the varying query sequence length as there is an optimum window size for each query sequence length. Hwan et. al. [3] introduced a novel method called multiple indexing to decrease the execution time during subsequence matching of two patterns, namely, the input data pattern and the query sequence.

In their studies they found that as the window size increases according to the query sequence, the execution time decreases. To determine the optimum window size many studies were done in subsequence matching. We review two methods that are closest to our approach, the FRM and dual match that calculate optimum window size. However, due to the limitations of sensor networks, these methods are not directly applicable.

3.1 FRM:

FRM [3] uses the concept of a window of fixed length for R^*-tree indexing. It extracts a sliding window size of size *w* from every possible position inside each input data sequence *S* of length len(*S*)($\geqslant w$), and then converts every sliding window into a point in $f(\ll w)$-dimensional space by using DFT. The total number of points extracted from each data sequence *S* is ($len(S) - w + 1$). As a result, a large number of points appear in this way, and thus the storage overhead for storing these points individually also gets large. For alleviating this problem, FRM forms the minimum bounding rectangles (MBR) enclosing multiple points and builds an R^*-tree on these MBRs instead of points. For subsequence matching, FRM extracts p disjoint windows of size *w* from a query sequence of length $len(Q)(\geqslant w)$ where $p = \lfloor len(Q)/w \rfloor$, and then converts every disjoint window into a point in *f*-dimensional space by using DFT. Thus in this method memory storage for each dimensional point is the issue.

3.2 Dual-Match

Dual-Match was proposed in [6] to overcome the weakness of FRM addressed above. Dual-Match extracts windows in the way opposite to FRM: It extracts disjoint windows from data sequences and sliding windows from a query sequence. In Dual-Match, instead of storing the MBRs containing multiple data windows as in FRM, each data window is individually stored in the index. By constructing the index in this manner, Dual-Match can dramatically reduce the number of false alarms, and can obtain search performance much better than FRM. In dual match subsequence match is done by extracting sliding windows for length of query sequence into f- dimensional space by using DFT. Usually, Dual-Match sets the window size *w* *w*hich divides the query MBR to be [(min(*Len*(*Q*)+1)/w]- 1). Index searching step constructs a candidate set by comparing each query window point, thus discarding false alarms.

In [3] fig 3 when the query sequence length is increased with constant window size, execution time is increased simultaneously. In figure 3 when window sizes increases with for a constant query sequence, execution time decreased.

These methods focus highly on variant time series like stock prices, weather forecasting, temperature readings etc., Due to the limitations of sensor networks they are too complex for sensor networks. Our method of self-adaptation is simpler with fewer calculations and need less memory for better performance.

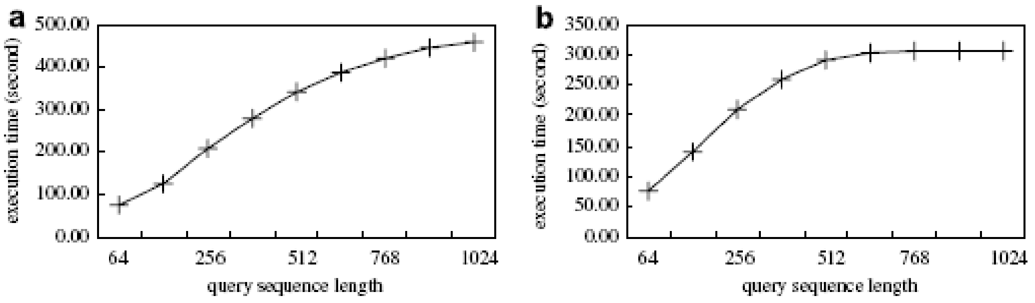


Fig. 1. Performance with different query sequence lengths: (a) FRM; (b) Dual-Match.

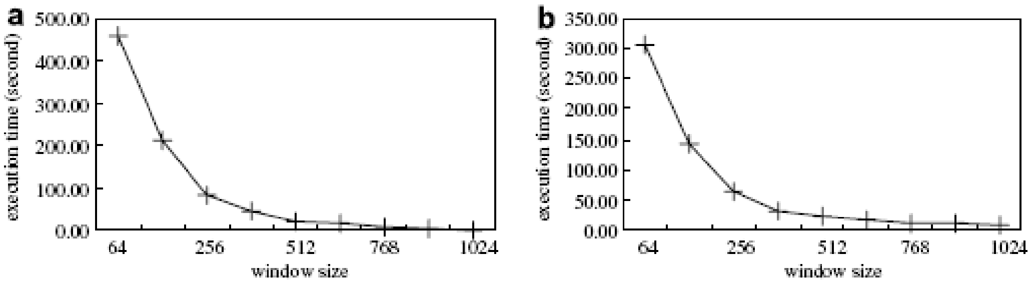


Figure 3   Performance with different window sizes

CHAPTER IV

4.1 SELF ADAPTATION

Self Adaptation is the third and vital feature in Deception Framework. It is defined as adapting to the changed pattern of attack and responding accordingly. Adaptation is necessary to maintain deception and prevent the attacker from suspecting anything. In our work, the objective of deception is to give the attacker what he expects. Self adaptation considers the attackers own strategy to deceive him. It deceives the attacker by responding in a pattern as expected by the attacker. This is learned and predicted from past experiences.

Self adaptation is an important feature because, if the deceiver cannot self-adapt, the purpose of deception is defeated. If the purpose of deception is defeated it may be suspected by the attacker, which will encourage him to choose other ways to attack
We first describe the problems in self adaptation and then discuss approaches to solve them. We later propose an algorithm for self adaptation. Finally this chapter concludes with future work.

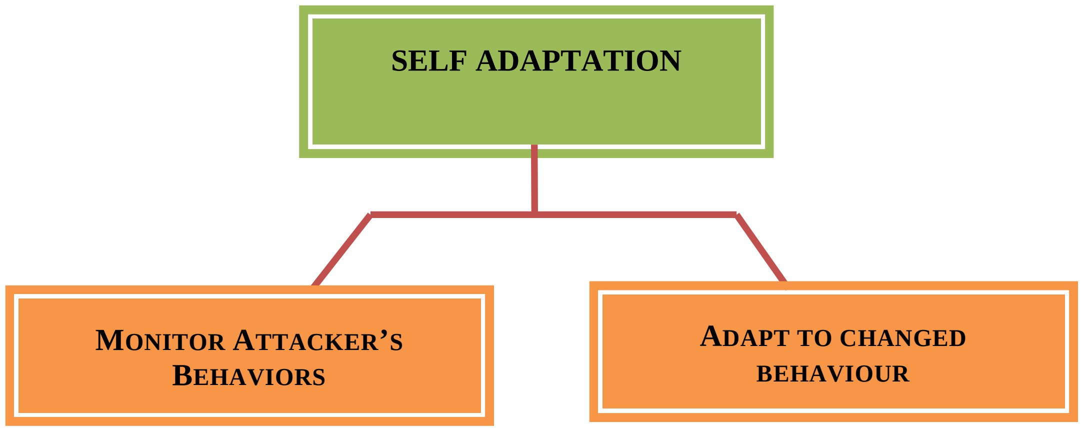


Figure 4 Self Adaptation Block diagram

4.2 Problem in Self Adaptation

In order to self adapt, the deceiver must first be able to identify the changes in pattern and then must know how to respond to the changed pattern. We divide the problem into three steps

1) Monitoring the attacker's behavior and identify the changed pattern
2) Predict the response to the changed pattern
3) Self adapt with the changed pattern.

In this thesis we study the 1st and 3rd points.

Identify changed pattern : To identify the changed pattern the deceiver needs to know

a) The reference pattern or the query sequence. The reference pattern is expected input pattern form the attacker. This is obtained by identifying the steady rate of attack in the input data during the analysis of input data pattern.
b) The present data pattern or data sequence, to which we will compare the reference pattern.

c) The Window size to compare both data and query sequences.

d) The Tolerance value to monitor and identify changes in the data pattern.

<u>Self adapting to the changed pattern :</u> Once the change in pattern is identified, the system needs to adapt to the changes and continue the deception of the attacker. Below are the features which the framework needs to adapt with the changed pattern.

1) Learn the present new pattern which is the requests sent by the attacker

2) Predict the new query sequence which is the expected request rate from the attacker. This is based on the reference or query sequence.

3) Determine the window size for new query sequence to avoid window size effect.(more detailed description of window size is given in section 4.6).

4) Change the tolerance value with the new pattern (which is used to identify change in pattern).

In next two sections we focus on identifying the changes in pattern and adapting to the new pattern.

4.3 IDENTIFIYING CHANGES IN PATTERN OF ATTACK

Any data that is collected in a certain time period can be represented in the form of a pattern. This pattern can be a sine curve or straight line etc, in a time series sequence. Any series of data that has been collected over time and represented graphically is called a time series. In our method, we consider three types of time periods in a time series:

- Time point is the smallest unit of time over which the system collects data, e.g., ten seconds.
- Basic window is a consecutive subsequence of time points over which the system

maintains a digest (i.e., a compressed representation) e.g., two minutes.

- Secondary window is a user-defined consecutive subsequence of basic windows over which the user wants statistics, e.g., an hour.

Both time point and basic windows are system fixed variables. Secondary window is user defined and can be varied.

In this thesis we consider a sinkhole attack in which the attacker claims to be close to the base station to its neighboring nodes. Once the attack has been identified the deception framework has to understand the pattern of attack. The deception framework collects the attacking pattern from the attacker, learns the pattern and predicts the pattern using the methods proposed by Zhang in [2] called *F4*(Fractal FOREcasting) for example. Once the pattern is learned and reaches the steady state we consider the repeating pattern as a query sequence and the steady rate as the objective of the deception. Subsequence matching is a process, in which we search for the pattern in the input data time series which is similar to a query sequence.

Self adaptation is a simpler process than the work done by Faloutsos et al (called FRM) [3] and Dual form [6]. Since our problem is to not to find a similar pattern, but to identify change in pattern which makes our job simpler then pattern matching. Given the resource limitations of sensor networks, we use a simplistic approach which is efficient to identify changes . To recognize the change in pattern we compare the unknown input data time series with the known query sequence. For comparison input data and query sequence are divided into window size. Window size is the length of window to divide both query sequence and input data pattern to compare and identify the change in pattern (a more detailed explanation is given in section 4.6). The difference between both the input and query sequence are compared with tolerance value (section 4.5) to identify the change in pattern.

We assume the attacking data input time series is Nt is given. Nt = {n_1, n_2, n_3,….,n_j } where j is the time point and n is the value at the jth time. Secondary Window is represented by W_i, basic window is represented by w_k. First the steady rate of the input data is calculated by finding the difference between the harmonic mean of two adjacent secondary windows. The input sequence of the steady rate V_{obj} is considered as the query sequence. After the query sequence is obtained, the optimum window size W_{opt} is calculated to avoid the window size effect as explained in the previous section. The optimum tolerance value *T* is also calculated to avoid false alarms. The query sequence and input data is divided into *n* windows of length W_{opt} and each W_{opt} is divided into *i* secondary windows of length W_i . Secondary window | W_i| = *k* where *k* is the number of time points.

$$\text{Harmonic Mean of Secondary Window} = \frac{\text{total no of time points in each secondary window}}{\text{Sum of inverse of total time points}}$$

$$HMean(W_i) = \frac{k}{\sum_{j=1}^{k} \frac{1}{n_{i,j}}} \qquad (4.1)$$

Once the harmonic means of two adjacent secondary windows are obtained, the normalized difference of two adjacent secondary windows are calculated to determine the movement of attack. It enables finding similar fluctuation patterns even though they are not close.

Let *b* be the basic window. Normalized difference (δ) helps us to find the difference between the objective and input data time series. To find the measurement of deviation from V_{obj} we define a term called delta (Δ). Sometimes the Δ value decreases over time.

Decrement is represented by dec(x).

$$\delta = \frac{\text{HMean}(W_{n+b+1}\ldots\ldots\ldots W_{n+2b}) - V_{obj}}{V_{obj}}$$

$$\Delta_{t+1} = \delta + \text{dec}(\Delta_t)$$

$$\text{dec}(\Delta_t) = (1 + \Delta_t)^e - 1 \qquad (4.2)$$

where e is close to zero. The closer the x in dec(x) is to 0, the lower the decreasing rate. The difference δ comes in at a rate of once per secondary window. We define a time series D_t as the difference time series.

$$D_t = \{D_i, D_{i+1}, \ldots\ldots\ldots D_{t+n}\}$$

D_t at every time is zero except at each secondary window. As a result we formalize the cumulative measure of deviation. Δ is the difference of each secondary window to the V_{obj}.

$$\Delta \text{ is} \qquad \Delta_{t+1} = D_{i+1} + \text{dec}(\Delta_t) = D_{i+1} + (1 + \Delta_t)^e \qquad (4.3)$$

If more than 50% of the secondary window in D_t is greater than maximum tolerance value than we consider that length of input sequence as change in pattern.

4.4 Self Adaptation to the Changed pattern

Once the change in pattern is identified, the deception frame has to adopt to the new pattern. There are two issues to be considered: what needs to be adapted and why it has to be adapted. The previous section described how to find the change in pattern.

Once the change in pattern is identified the query sequence (steady rate of the input pattern) also changes. As the query sequence changes, to avoid window size effect we need to change the window size to an optimum value. Therefore, we propose an algorithm which changes window size according to changes in the query sequence.

Similarly the tolerance value of the deception framework should also be changed with the new query sequence. Tolerance value is maximum value of difference that can be allowed for a query sequence of input data time series. As query sequence changes, the new pattern may increase or decrease from the previous pattern. There may also be a phase shift in the input pattern. Hence the tolerance value should change accordingly. If the tolerance value is not changed we may mistake the similar pattern as a change in pattern because tolerance value did not increase or decreases with the increase or decrease in query sequence.

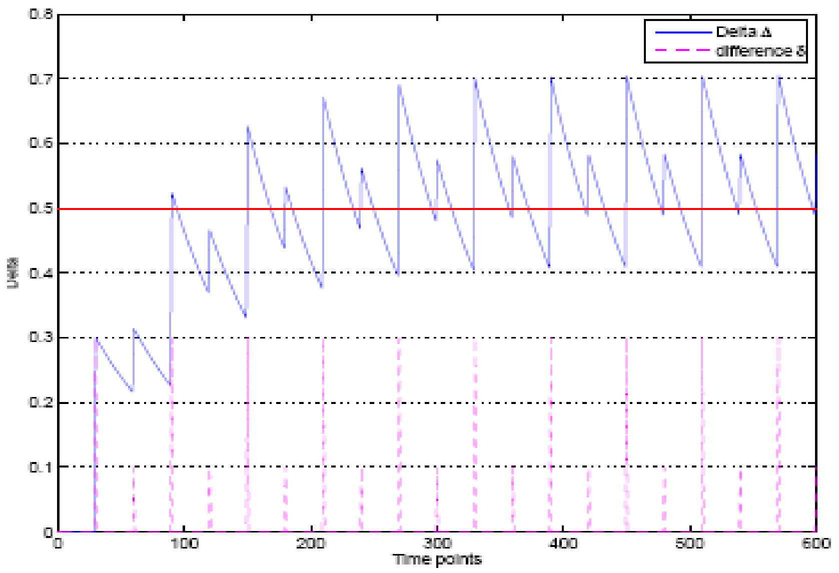
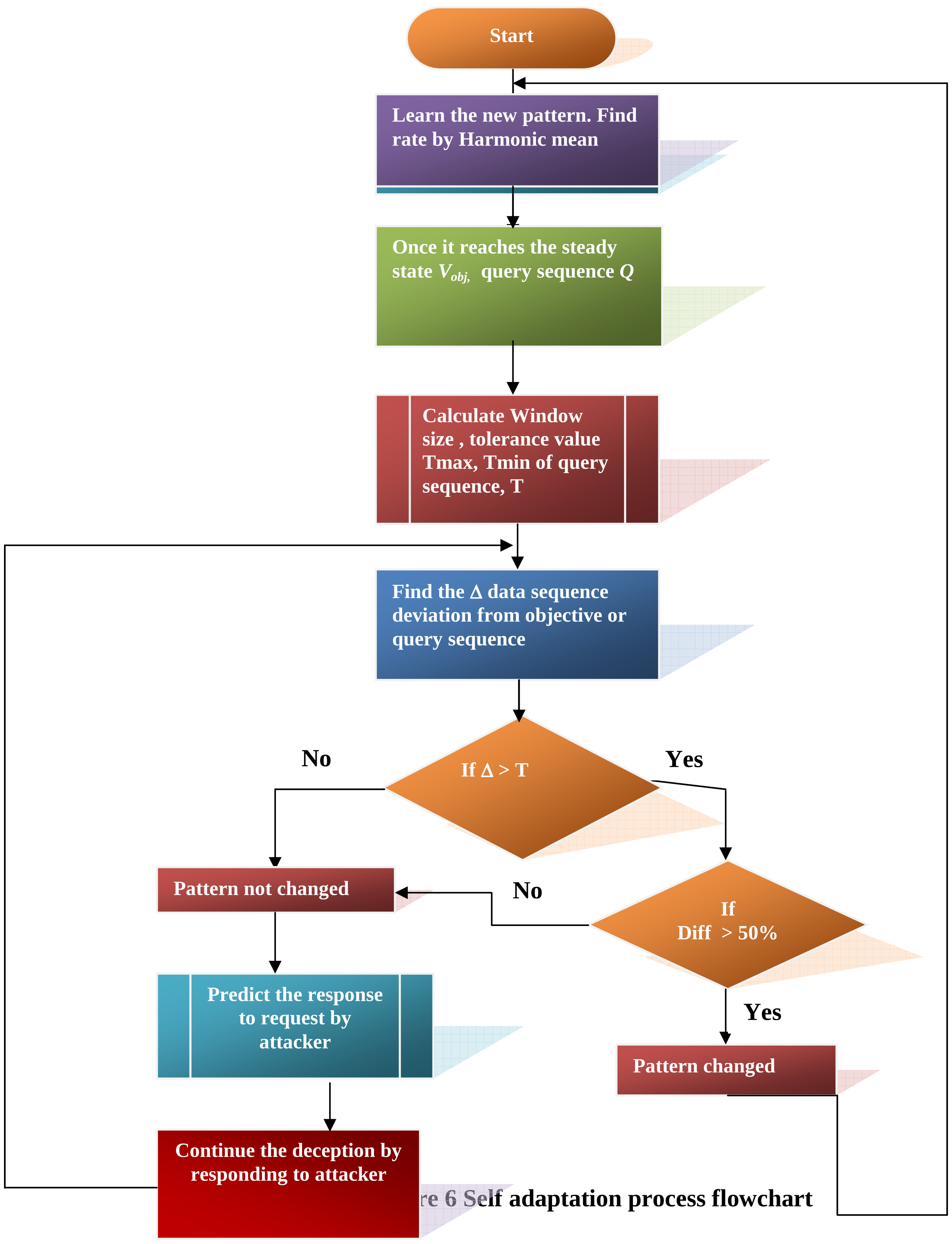


Figure 5 Effect of fixed tolerance value

Figure 5 shows the effect of not changing the tolerance value. Here the old tolerance is 0.5 delta. Around 200 time points, a change in pattern was detected. Because of the change in pattern, the query sequence also changed. However, since our original tolerance value of 0.5 has not changed, it considers the changes after 200 as changes in pattern even though there is no change in the pattern. The tolerance value has therefore to be adapted to the new query sequence. Thus self adaptation is required for more accurate data or performance. Figure 6 flowchart of self adaptation process.

Self Adaptation Process


Start
Learn the new pattern. Find rate by Harmonic mean
Once it reaches the steady state $V_{obj,}$ query sequence Q
Calculate Window size , tolerance value Tmax, Tmin of query sequence, T
Find the Δ data sequence deviation from objective or query sequence
If Δ > T
No
Yes
Pattern not changed
No
If Diff > 50%
Yes
Pattern changed
Predict the response to request by attacker
Continue the deception by responding to attacker

re 6 Self adaptation process flowchart

4.5 Optimal Tolerance value

Tolerance value is a nominal value of maximum difference allowed from the query sequence to the input data. Tolerance value is of vital importance as data traveling in a media like air can face disturbances or noise on its way to a node in the network. Noise might indicate as change in pattern which in fact is not a change in pattern. We name such alarms as false alarms. This tolerance value should be optimum; if Tolerance value is too small then the system may conclude consider a signal with noise as change in pattern. If tolerance value is too large then, the system may neglect a real change of attack as noise resulting in failure of deception.

To obtain optimum tolerance value we need to first calculate the tolerance limit T_{max} and T_{min} for that query sequence. Tolerance limit is the range of maximum and minimum mean average deviation of new query sequence and old query sequence. Tolerance limit T_{max} and T_{min} are like the upper and lower tolerance limits in the new input sequence. Input data values that lie within this range are accepted. Data that goes beyond the tolerance values are considered as change in pattern.

To identify change in pattern we need to know the difference instead of upper limit and lower limit as we are comparing two sequences: query sequence and input sequence.
We call this difference value as tolerance value and it is represented by *T*. The upper bound difference is the difference between the T_{max} and the value of the pattern. Similarly the lower bound difference is the difference between the T_{min} and the value of the pattern. The smaller of these differences is considered to be the value *T*. When Δ (deviation of the time series from objective) is more than *T* the deception framework considers that as change in pattern.

Tolerance Limit:

We first propose a method to calculate tolerance limit from old query sequence and new query sequence. In this method we take mean average deviation of both old query sequence and new query sequence. We multiply the mean average deviation with a constant value sigma where sigma is less than or equal to one. Then by either subtracting or adding the mean average deviation to the old tolerance limit we find tolerance limits T_{max} and T_{min} for the new query sequence. We assume that we know old tolerance limit from past experience.

Let us consider *P*(*t*) as the new query sequence at time *t* , *V*(*t*) as the old query sequence. *n* is the length of the new query sequence. The old tolerance limits which we call old_T_{max} and old_T_{min} and sigma (a constant value) is given.

The Mean Average Deviation (MAD) is calculated as

$$MAD = \frac{\sum |P(t) - V(t)|}{n} \text{ where } 1 \leq \text{t} \leq n \text{ and} \qquad (4.4)$$

n is the total number of values.

$$new\_\ T_{max} = old\_T_{max} + \text{sigma} * MAD$$

$$new\_T_{min} = old\_T_{min} - \text{sigma} * MAD \qquad (4.5)$$

Optimum Tolerance Value *T* :

Once the tolerance limit is obtained we need to know the maximum and minimum value in new query sequence as value_max and value_min respectively. Now subtract the *value_max* from T_{max} and similarly *value_min* from T_{min} as *diff_max* and *diff_min*. Compare both the values and smallest value is considered as T. The reason to consider the smallest value is because the smaller the deviation, the more accurate the data.

$$diff\_max = T_{max} - value\_max$$

$$diff\_min = value\_min - T_{min}$$

$$T = min(diff\_max, diff\_min) \qquad (4.6\ )$$

This method is more accurate because with a fixed tolerance value, sometimes the difference may be more than the defined value as the query sequence changes. This can result in incorrect identification of pattern change. Our method calculates tolerance value according to varying query sequence which give a more accurate tolerance value then fixed values.

Flowchart for Tolerance limit

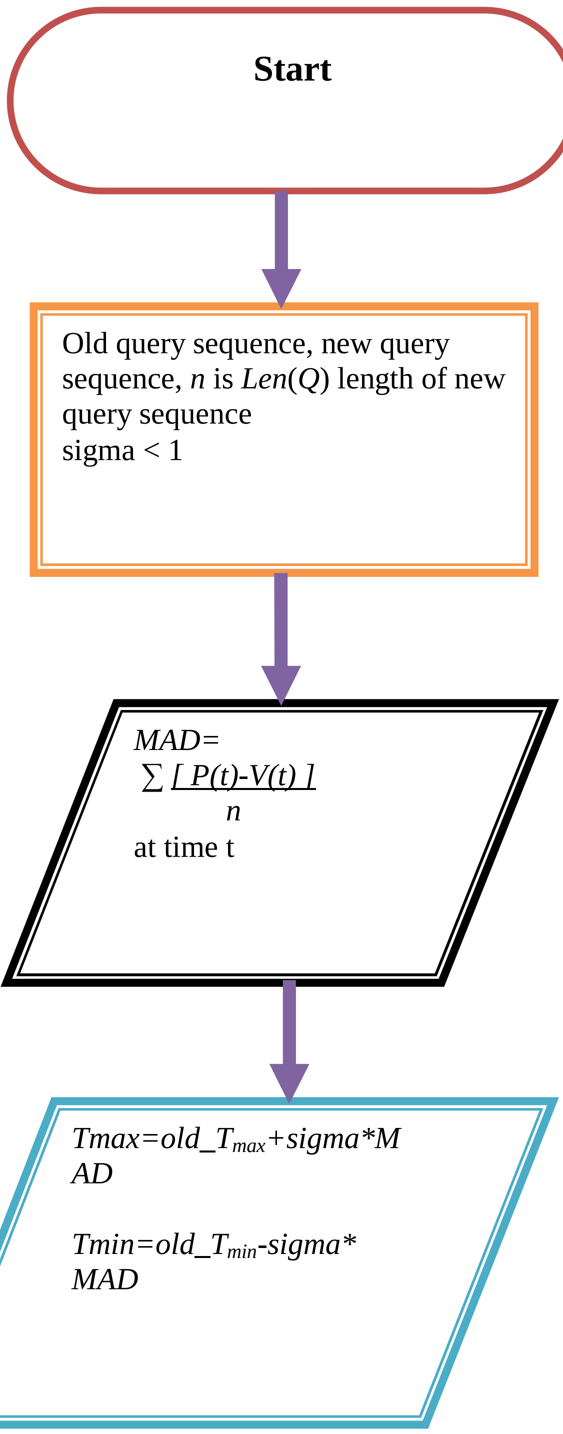


Figure 7 Flowchart for Tolerance limit

4.6 Optimum Window Size

Window size is the length of window used to divide the query sequence into *n* windows. The window size is also used to divide the input data sequence into *x* number of windows. *x* may or may not be equal to *n* since the input data sequence may be of different length than the query sequence. . We next compare the query sequence of *n* windows with input data sequence of *x* windows of length window size to identify change in pattern.

Determining optimum window size is of vital importance. Hwan, Park, Kim [3] show that as the length of the query sequence increases with fixed window size the execution time increases. They conclude that performance deteriorates as the difference between the window size and length of the query sequences get larger. This phenomenon is called the ***window size effect***.

In the deception framework execution time is very important. As the data sent from the attacker to different nodes in the network are directed to sacrificial nodes, sacrificial nodes are overloaded with data which slows down the execution time reducing the effectiveness of deception. Furthermore, the window size effect deteriorates the situation even more. It is therefore important to obtain an optimum window size which can help to decrease the execution time and increase the performance.

In [3] Hwan Lim, Park, Kim proposed a novel approach called index interpolation to overcome this performance degradation. The drawback of the method is the limited resources available in sensors. Index interpolation has complex calculations which involve calculating the cost of window size and based on the cost value, the window size is determined. Moreover the memory requirements are extensive since memory is needed to store the binary tree method used to calculate the window list. We propose a novel method to calculate the optimum window size which is less complex and needs less memory than complex index interpolation.

4.6.1 Calculation of Optimum Window list

Before we determine the optimum window size we calculate the window list. Window list is a set of different sizes of windows which are considered to be optimum for different query sequences. The values of these sizes lie in between the maximum and minimum length of query sequence. These maximum and minimum length of query sequence are obtained from analyzing past data. We assume that our deception framework will have query sequences within these values.

Let *Len*(*Q*) be the length of query sequence, *Len_max*(*Q*) be the maximum length of the query sequence and *Len_min*(*Q*) be the minimum length of the query sequence and *n* the index number to store *n* window in the window list. In [4] lemma 5, the relation between the maximum window size and minimum length of query sequence is given. The maximum window size should be less than half of the minimum length of the query sequence.

If *Len_max*(*Q*) is the maximum length of query sequence that we expect in the system, *Len_min*(*Q*) is the minimum length of query sequence that we expect in the system, window list is W_p, then the minimum window size W_{min} and the maximum window size W_{max} are calculated as follows:

$$W_{max} = \max\{w_\mathrm{i} \mid w_e \leq [\, (Len\_max(Q)+1) / 2 \,]\ (1 \leq e \leq \mathrm{n})\ \}$$

$$W_{min} = \min\{w_i \mid w_e \leq [\,(Len\_min(Q)+1) / 2\,]\ (1 \leq e \leq n)\}$$

$$W_p = < w_1, w_2, \ . \ . \ . \ , w_n > \text{where } W_{min} < w_1 < w_2 < \_\ \_\ \_ < w_n < W_{max}$$

W_{max} and W_{min} are the limits of the window list. The other window sizes are then obtaind by dividing the range W_{max}-W_{min} evenly over the n indices of the window list. Since the maximum value of window size will be less than or equal to half of the query sequence, the chances of window size greater than the length of query sequence is very unlikely. According to the window size effect, as a window size gets nearer to *Len(Q),* the performance improves. However for accuracy the window size should be less than *(Len(Q)+1)/2)*. Therefore we evenly divide the windows between W_{max} and W_{min} into n windows beginning from minimum window size W_{min}.

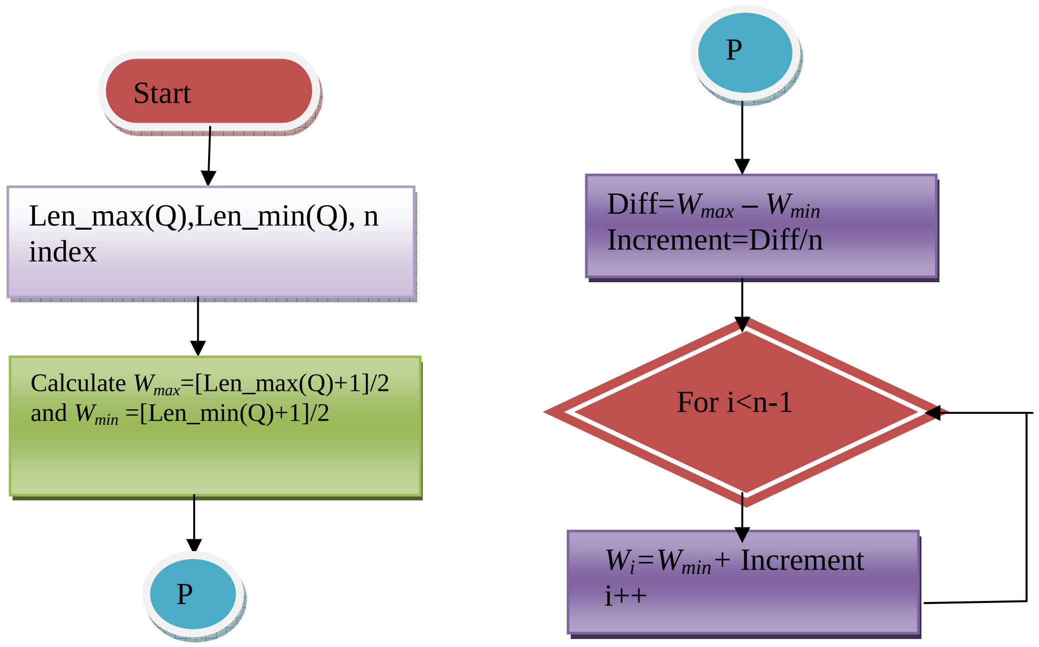

Start
Len_max(Q),Len_min(Q), n index
Calculate W_{max}=[Len_max(Q)+1]/2 and W_{min} =[Len_min(Q)+1]/2
P
P
Diff=$W_{max} - W_{min}$
Increment=Diff/n
For i<n-1
W_i=W_{min}+ Increment
i++

Figure 8 Flowchart for calculating the Window list

4.6.2 Choosing the Optimum Window size

In this section we suggest a procedure to choose the optimum window size for a given length of query sequence from the above calculated window list. The maximum window size for a given query sequence should be less than half the length of the query sequence (from lemma 5 in [4]). If the length of the present query sequence is *Len* (*Q*), maximum window size is

$$w_{max} = [Len(Q)+1]/2 \qquad (4.7)$$

w_{max} is the maximum window size into which a given current query sequence can be divided. w_e is the value in the window list that is lower than, but closest to w_{max}. To obtain the optimum window size we select the window value w_e from the window list (4.6.1) which is nearest to w_{max}.

$$W_{opt} = w_e \leq w_{max} \quad \text{Where } w_e \in \quad W_p = \{w_1, w_2, w_3 \ldots\ldots w_e, \ldots w_n\}$$

For example, a window list has 9 indexes of windows 10 24 38 52 66 80 94 108 122. From equation (4.7) if new query sequence is of length =80 then $w_{max} = 40$ then optimum window size W_{opt}=38. From Hwan et. al. [3] we know that as the number of indexes of the window list increase, the execution time decreases which is show in figure 9.

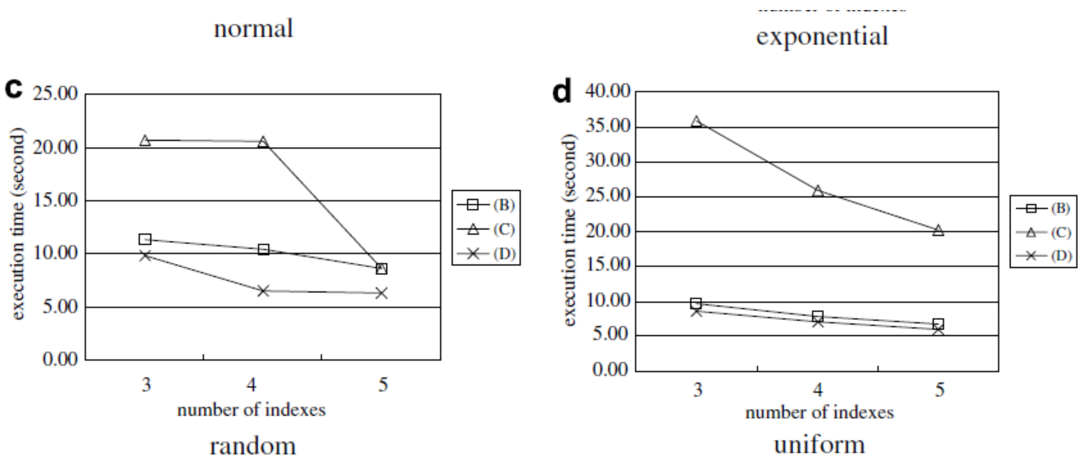


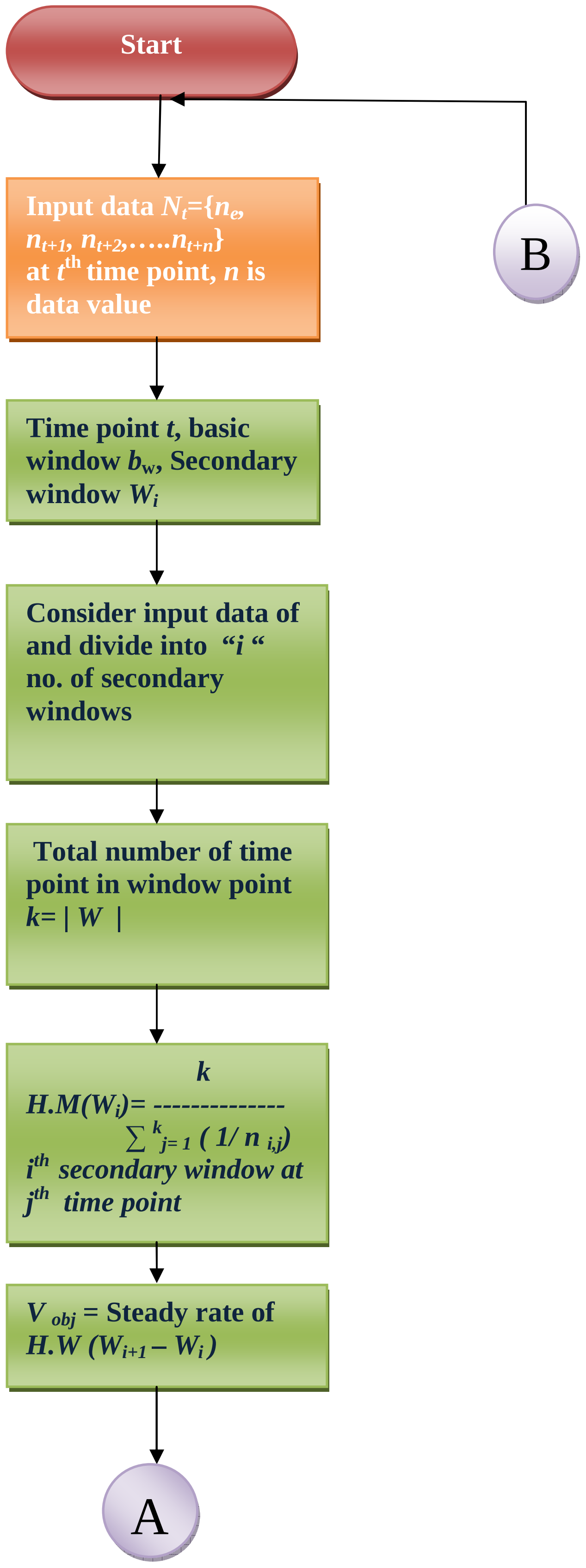
Figure 9 Performance with different number of index sizes

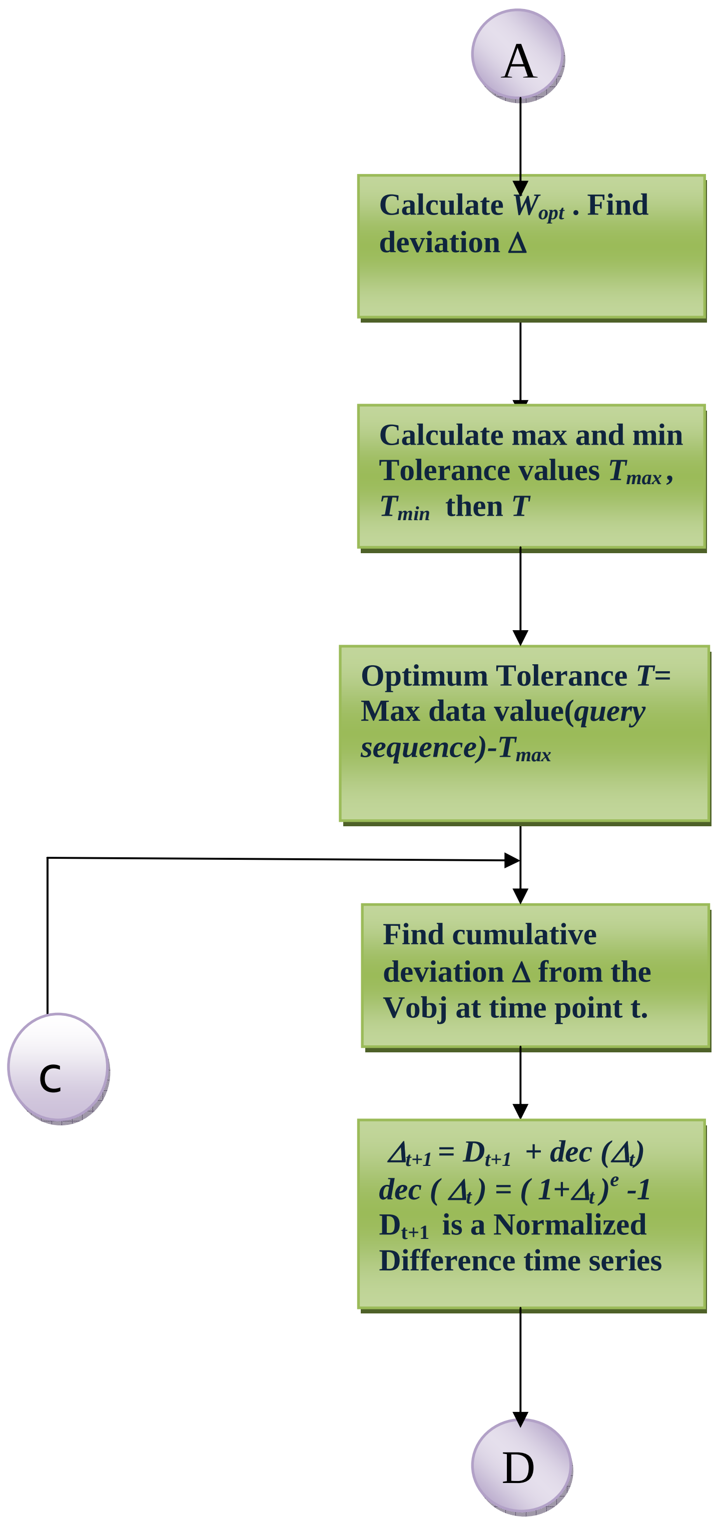
In Fig 9 performance analysis of increasing indexes and execution time is shown as given [3]. As the number of indexes of the window list increase, the execution performance becomes better. Determining the optimum index size is left for future work.

In figure 10 the self-adaptation process is explained. It is divided into three sections (a), (b), (c). Each flowchart from each section is connected by connectors A, B, C, D. In (a) the self-adaptation process to find the Harmonic mean between two adjacent secondary windows is given. In (b) the optimum window size and optimum tolerance value is calculated. In (c) Δ (deviation from the V_{obj}) is compared with T. A counter variable in this section is used to keep track of the number of deviations to identify change in pattern.

Figure 10 FLOWCHART OF IDENTIFYING PATTERN CHANGE

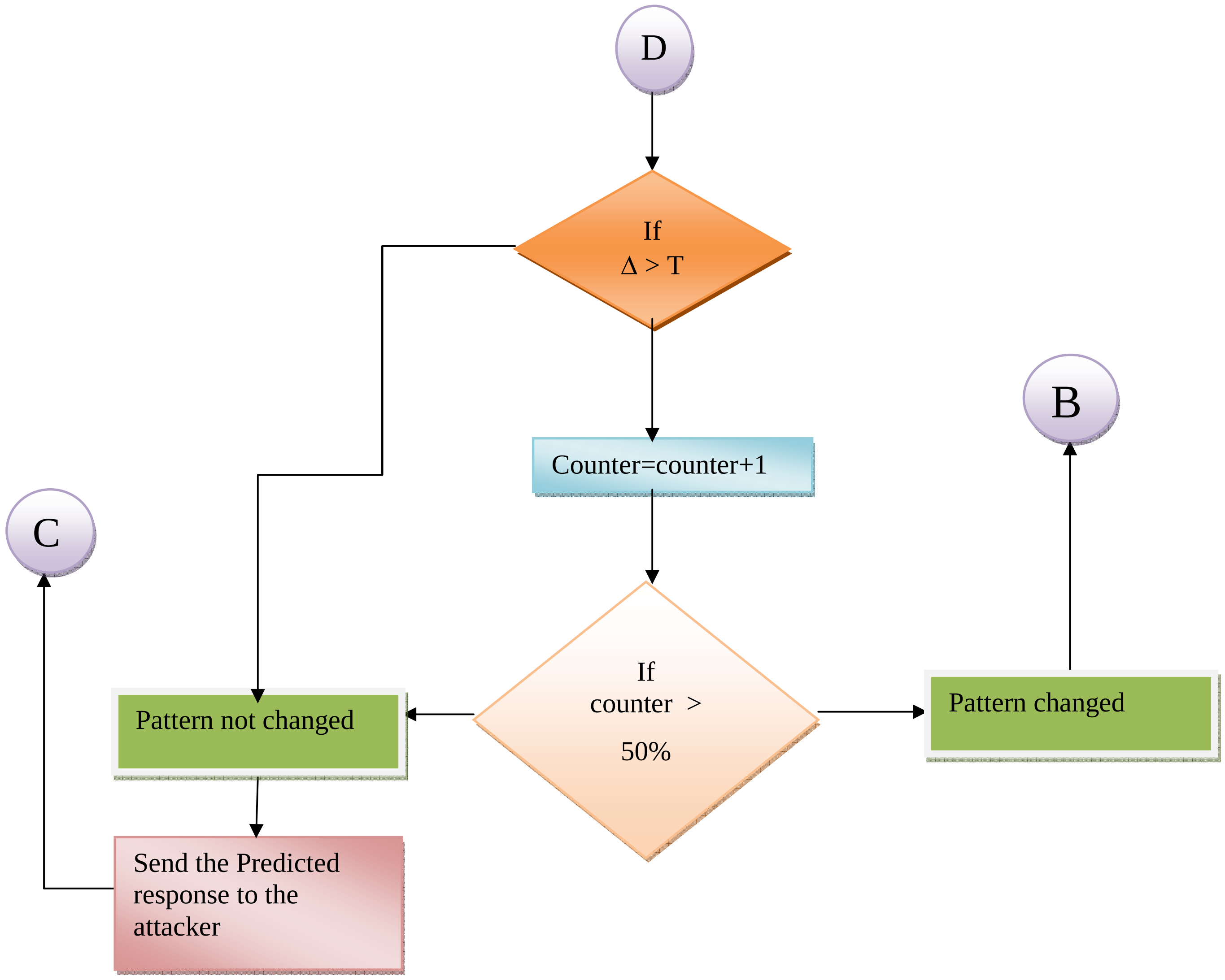
(a)


Start
Input data $N_t=\{n_e, n_{t+1}, n_{t+2},\ldots..n_{t+n}\}$ at t^{th} time point, n is data value
B
Time point t, basic window b_w, Secondary window W_i
Consider input data of and divide into "i" no. of secondary windows
Total number of time point in window point $k=|W|$
$H.M(W_i)=\frac{k}{\sum_{j=1}^{k}(1/n_{i,j})}$
i^{th} secondary window at j^{th} time point
V_{obj} = Steady rate of $H.W(W_{i+1}-W_i)$
A


A
Calculate W_{opt} . Find deviation Δ
Calculate max and min Tolerance values T_{max} , T_{min} then T
Optimum Tolerance T= Max data value(*query sequence*)-T_{max}
Find cumulative deviation Δ from the Vobj at time point t.
$\Delta_{t+1} = D_{t+1} + dec(\Delta_t)$
$dec(\Delta_t) = (1+\Delta_t)^e - 1$
D_{t+1} is a Normalized Difference time series
C
D

(b)

Figure 10 Flowchart of Pattern change in Self adaptation (continued)



(c)

Figure 10 Flowchart of Pattern change in Self adaptation (continued)

4.7 Algorithm

4.7.1 Self Adaptation

Step 1: Given data sequence $N_t = \{n_t, n_{t+1}, \ldots n_{t+n}\}$, secondary window W_i, basic window w_k,

Step 2: Find the steady rate V_{obj} of the sequence by finding the harmonic mean of the data sequence.

Step 3: Find the query sequence which is a steady rate sequence.

Step 4: Calculate the window size and tolerance limit for the query sequence as the W_{opt} and T_{max} and T_{min}.

Step 5: To avoid false alarms we use dec(x) function, if increment c ,

if $(\delta < c)$ then inc = δ ; else

if $(\delta > c)$ then inc = c ;

Where δ is normal deviation to calculate the movement of attack.

Step 6: Find the Δ difference of data from the query sequence

$\Delta_{t+1} = \delta + dec (\Delta_t)$

$D_{t+1} + dec (\Delta_t)$

$dec (\Delta_t) = (1+\Delta_t)^e -1$

D_{t+1} is a Normalized Difference time series

Step 7: To find the Optimum tolerance (allowed deviation from query sequence)

T = (Maximum input data value in query sequence $-T_{max}$)

Step 8: If more than 50% of the values in window sizes > tolerance value consider as change in pattern.

if $(\Delta \geq T) > 50\%$.

Step 9: If Step 8 true then pattern is changed so repeat the Step 1 to Step 8

Step 10: If Step 8 is false then send the predicted response and continue deception

4.7.2 Calculate Window size list and choose optimum window.

Step1: Declaration : *Len_max(Q)* , *Len_min(Q),* index[*n*] ,

Step 2: Given minimum and maximum length of query sequence of same multiple.

Step 3: Calculate maximum possible window size and minimum possible window size by

$W_{max}= [\ Len\_max(Q) +1\ ] / 2$

$W_{min}= [Len\_min(Q)+1]/2$

Step4: Evenly divide into n window sizes beginning from W_{min} to W_{max}.

Step 5: Store in values in n indexes of window list array

Selecting Optimum window size

Step 6: Given *Len*(*Q*) then calculate $w_{max}=(Len(Q)+1)/2$

Step 7: Choose from *n* windows in window size list value closest to the max_win value as optimum window size.

4.7.3 Find Optimum Tolerance value

Step 1: Give old query sequence, new query sequence, sigma value which is less than 1, length of query sequence Len(Q).

Step 2: Find the mean average deviation (MAD) of new query sequence to the old query sequence of length Len(Q).

Step 3: old_T_{max} + sigma * MAD= new_T_{max}.

old_T_{min} - sigma * MAD= new_T_{min}.

Step 4: Find the maximum and minimum displacement value in query sequence. Let is be *value_max, value_min.*

Step 5: $diff\_max = new\_T_{max} - value\_max$

$diff\_min = value\_min - new\_T_{min}$

Step 6: Select the minimum diff from the *diff_max* and *diff_min.* This is the optimum tolerance value for our current query sequence.

$T = min(diff\_max, diff\_min)$

CHAPTER V

SIMULATION RESULTS

5.1 Network simulation

The proposed deception framework includes nodes and a base station. In fig 5.1 all the nodes in the grey color send data to the base station (red color node). The attacker (black color node) attacks its neighboring nodes saying that he is the base station. As a result all the nodes near this region sends data to the attacker. This is a sink hole attack. In fig 5.1 the surrounding gray nodes within the dotted circle are sacrificial nodes which deceive the attacker.

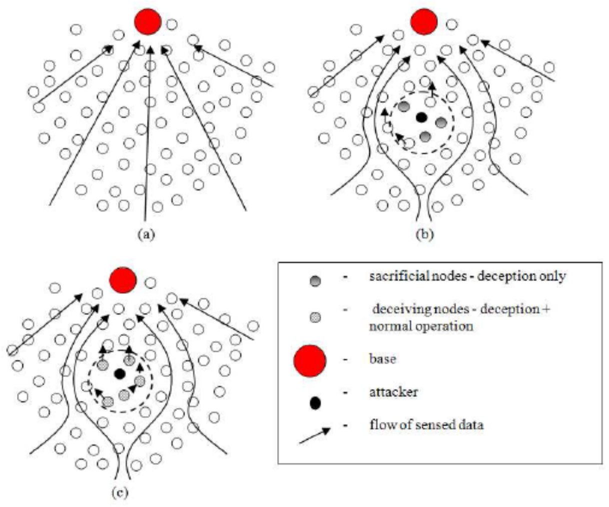

(a)
(b)
(c)
sacrificial nodes - deception only
deceiving nodes - deception + normal operation
base
attacker
flow of sensed data

Figure 11   Deception Scenario

5.2 Simulatin

The sink hole attack was simulated using the Tossim Simulator [1] and NesC Compiler. To run the program for determining the pattern change and its execution time we used Hp PC dv6000 with following configuration

- RAM : 1.0 GB
- Processor : 1.50 GHz
- Operating System : 32 bits Windows XP
- BCC C++ Compiler
- Text pad

Tossim Simulator

The Tossim simulator [1] is a discrete event simulator for TinyOS sensor networks. Instead of compiling TinyOS application on a mote you can compile it in the Tossim framework. Tossim provides a basis for real test-bed deployment. Tossim is developed at UC, Berkely and is widely used in the research community.

5.3 Attacking Strategy - Attacker

Sink Hole Attack:

In the Sink hole attack, the attacker pretends to be the base station or having the

best route to the real base station. This makes all nodes which are closer to the sinkhole to divert their messages to the attacker until the sinkhole stops sending messages to the nodes in the network. The data for the sinkhole was collected by Ling Zhu.

Attacker Scheme:

In a sink-hole attack, the attacker claims himself as the base station with a highest link quality to his neighboring nodes. The attacker changes his frequency of sending the routing message**.** In this attack we generated three types of attacking patterns. The attacker is sending message packets to different nodes in a random frequency, sine curve pattern, and in an increasing pattern. The attacker is node 1 and the base station node 0. The format of the packets were in decimal format such as [Msg type: Source Node, Local Time of Source node] where Msg type is routing message, S is the source node, local time is the time at which the source node (the attacker) sends message to neighboring nodes. For example [S: 1, 4] means Node 1 forwarded a packet at the 4th second. We first look at the input patterns from the attacker.

Input data 1:

In figure 12, the x-axis is the time points, y axis is no of packets sent by the attacker. The attacker initially sends advertisements or routing messages at a slow rate of 1 packet for every 16 seconds for the first 600 seconds or 10 minutes. Then the attacker increases transmission to a rate of 4 packets per 16 seconds from time 600 second to 1000 second (6.6 minutes). Finally the attacker changes transmission to 8 packets, 4 packets and 2 packets into sine wave pattern per every 16 seconds, from time 1000 second to 3000 second (33 minutes). In input 1 the attacker frequency was 0.25 and suddenly increases to frequency 2. Frequency is the rate of number of packets broadcasted per a time period. The y-axis is the ratio of attacker frequency to normal frequency. A normal node transmits one packet every 4 seconds. In the first 600 seconds, the attacker sends 1 packet every 16 seconds; hence the ratio is 0.25 as shown in the graph.

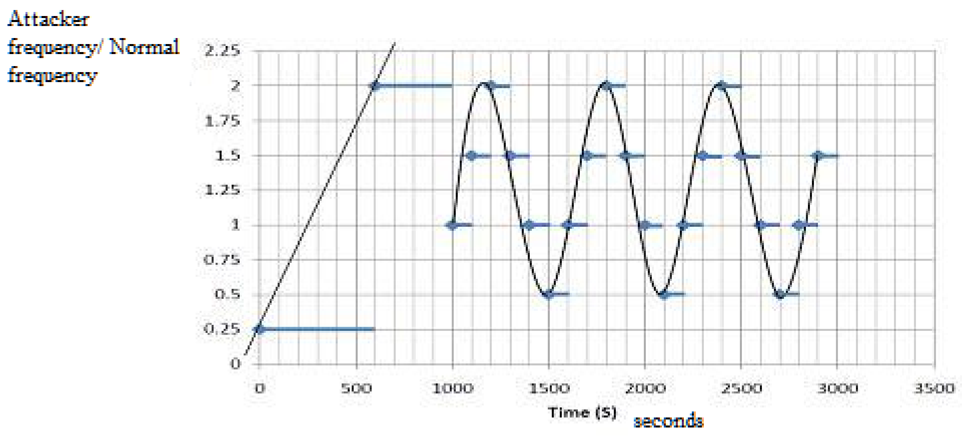


Figure 12 Attacker input data requests 1

<u>**Input data 2:**</u>

In fig 13, the x-axis is time points, y axis is no of packets sent by the attacker. The attacker sends advertisement or routing packets at a slow rate of 1 packet for every 6 seconds for the first 150 seconds. Then the attacker changes transmission to a rate of 2 packets per 6 seconds from 150 seconds to 270 seconds. Later the attacker increases the rate to 3 packets per 6 seconds from 270 seconds to 420 seconds. Transmission rate then decreases to a rate of 2 packets per 6 seconds from 420 seconds to 516 seconds. Finally the transmission changes to a sine curve of 2 packets to 1 packet for every alternate second from 516 seconds to 1104 seconds.

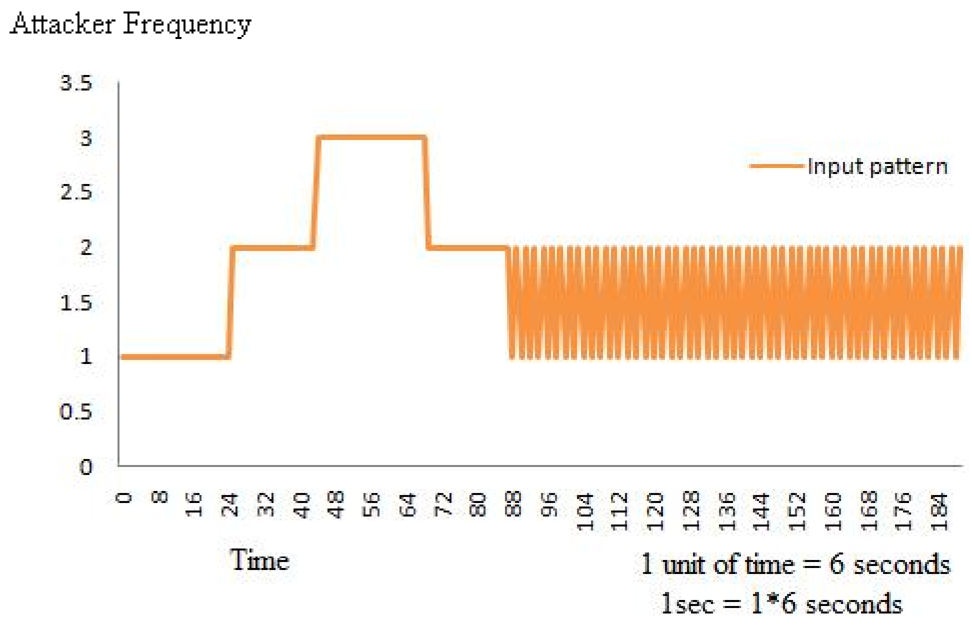


Figure 13 Attacker input data 2 with changing frequency

5.4 Determining Change in pattern – Simulation results

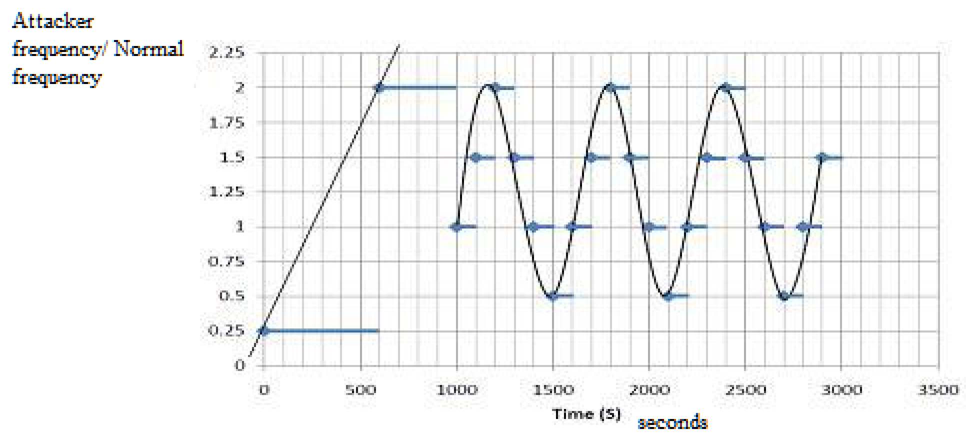
In this section we analyze the results of above inputs by applying the proposed algorithm of self adaptation with multiple indexing and tolerance limit. The aim is to show the effectiveness of the multiple indexing methodology used over the fixed window based approach and the effectiveness of optimum tolerance value over a fixed tolerance value.

Results:

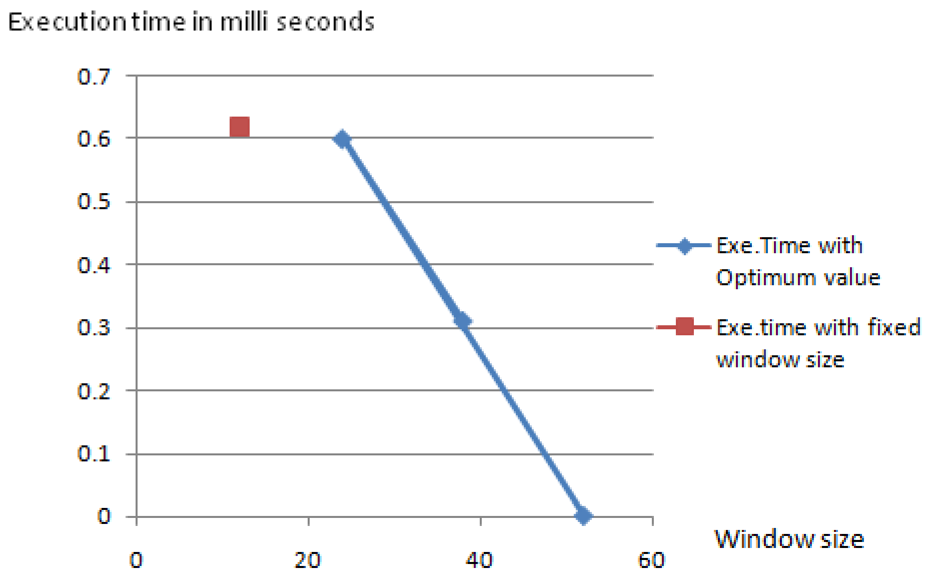
In this following section we analyse the results of the above input patterns. We compared the efficiency of fixed and optimum values of window size, and tolerance value.

5.4.1 Fixed Window Size Vs Optimum Window Size

Results of Input 1 data.



a) Attacker input data requests 1



b) Execution time in Optimum window size vs Fixed window size

Figure 14 Graph of Execution time with fixed window size vs optimum window size

Table 1: Optimum window size				
W(opt)	Time(milli seconds)	Len(Q)	T(old)	T(new)
24	0.6	50	Max=5	Max=2
38	0.31	80	Min=3	Min=0
52	0	120		

Table 1 : Execution time vs Optimum Window size

Table 2: Fixed Window size				
W(fix)	Time (Milli seconds)	Len(Q)	T(old)	T(new)
12	0.62	80	Max=5	Max=2
52	1.41	80	Min=3	Min=0

Table 2 : Execution time vs Fixed Window size

Fig 14 shows the result of input1 where the x axis represents window sizes and the y axis represents execution time to identify a change in pattern in milliseconds. Execution time is the time at which a change in pattern is identified. The red point in the figure represents execution time with fixed window size at *Len*(*Q*)=80 (length of query sequence). The blue line represents the execution times for different lengths of *Len*(*Q*) and the different optimum window sizes generated for different lengths of query sequence *Len*(Q).In tables 1and 2, values of the execution time for Input 1 data pattern at change in input pattern from 0.25 frequency to 2 frequency is shown .

First the attacker sends 1 packet for every 16 seconds and suddenly increases to 4 packets for every 16 seconds. Later he changes his pattern to sine curve with decrease in the number of packests from 8 packets to 4 packets. Let us consider one change in the pattern which occurs when there is a change from 0.25 frequency to 2. In table 1 the execution time to identify this change in pattern with differnet optimum window sizes are given. When window size is 24 execution time is 0.6 milli second to identify the change in pattern. When *Len*(*Q*) increased to 80, the window size changed to 38 and this took 0.31 milliseconds to identify the change which is half the previous execution time. In table 2, using the same length of query sequence Len(Q)=80 we randomly chose 12 and 52 as fixed window size. Our fixed value should not be equal to our optimum window size value.   When fixed window size =12 the time taken to identify the change in pattern was 0.62 seconds which is double the optimum window size execution time. When fixed window size=52 it did not detect the first three changes.but detected change in pattern (the fourth change which is the sine wave) at 1.41 milli seconds execution time. Hence the change from 0.25 to 2 to 1 to 1.5 was not detected. The change detected was at the sine curve pattern. Therefore based upon the above results we can say that the optimum window size improved the performance by halving the execution time.

Results of Input 2

In previous paragraph we observed that the execution time for optimum window size to identify the change in pattern is half the execution time for fixed window size. In this paragraph we will study if the optimum window size, besides reducing the execution time identifies all changes in the pattern correctly.

In this experiment we are considering one input 2 data pattern and assuming that we know the length of query sequence Len(Q). Optimum window size is calculated by our algorithm. We need to detect all changes in pattern and consider the time point value in the window at which the change in pattern is identified. For example, in figure 15 if W_{opt} = 24, assume the change in pattern is detected at time point 48 (which is optimum window 2), The execution time is then considered to be at the end of the second window. In other words, the execution time is defined to be the time at the end of the window that detects a change in pattern. This is the time considered in table 3 where the window is fixed and in table 4 where the window is optimum.

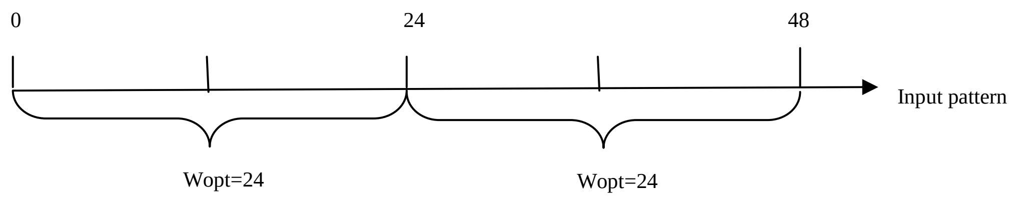
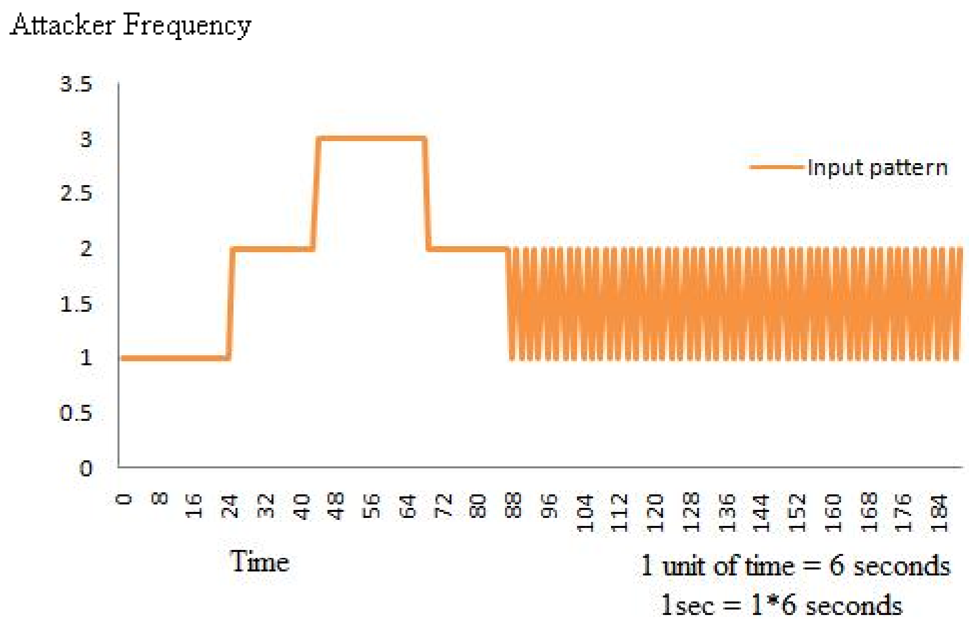
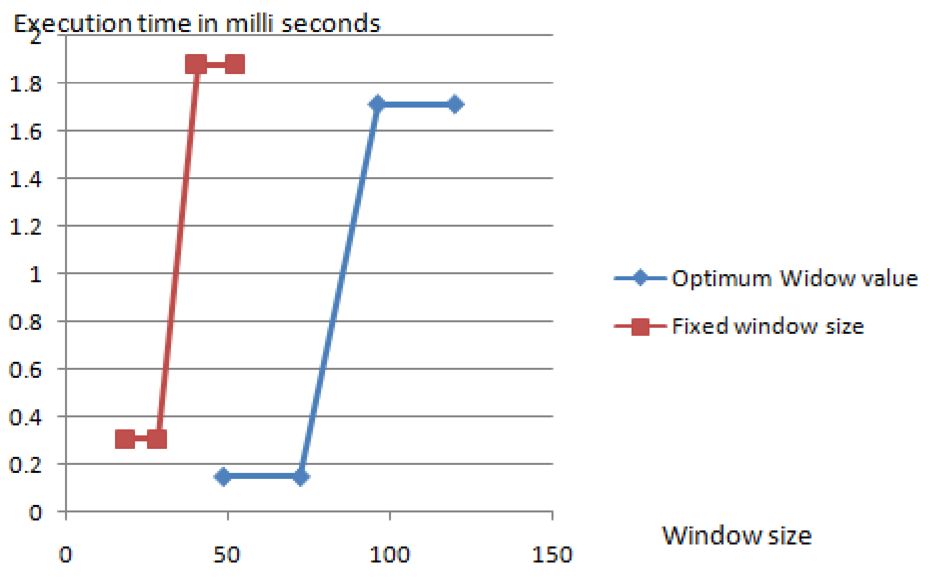


Figure 15 Consideration of time point in optimum window size



a) Input 2 from attacker with changing frequency



b) Execution time for input 2 with Fixed and optimum window size

Figure 16 Performance analysis of Optimum window size to Fixed window size

Table 3: Fixed window size =12		
Window	Time	Len(Q)
24	0.15	50
36	0.15	50
48	1.71	50
60	1.71	50

Table 3: Execution time vs Fixed Window size

Table 4: Optimum window size =24		
Window	Time	Len(Q)
48	0.15	50
72	0.15	50
96	1.71	50
120	1.71	50

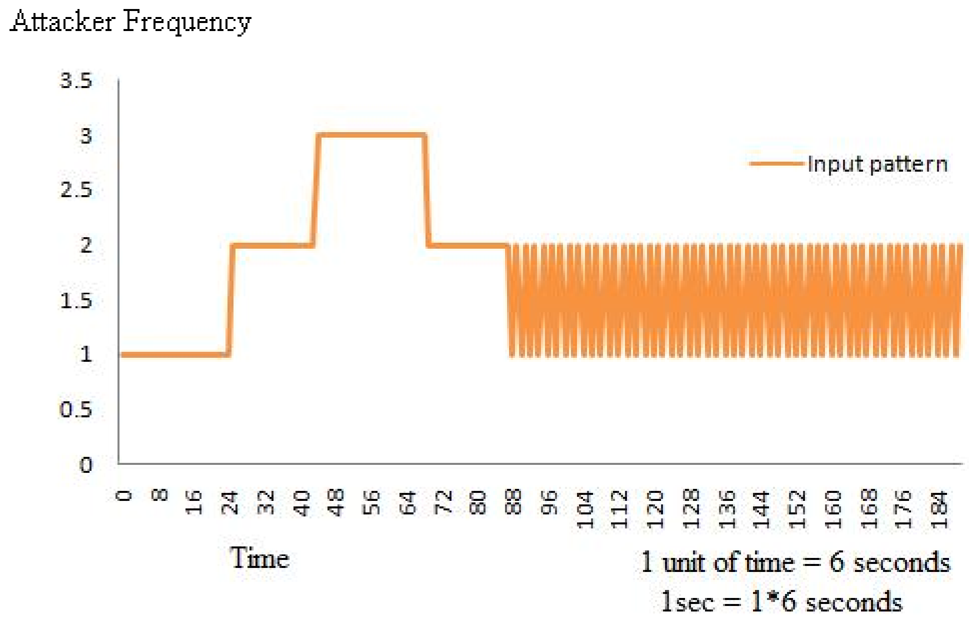
Table 4 Execution time vs Optimum window size

Fig 16 shows the result of input 2 where the x axis represents the secondary window and the y axis represents execution time to identify a change in pattern in milliseconds. Length of query sequence *Len*(*Q*)=50. The red line in the figure represents execution time to identify the change in the pattern for a fixed window size. The blue line represents the execution times to identify changes in pattern with optimum window size. In tables 3 and 4, values of the execution time to identify changes in pattern for data Input 2 using fixed window and optimum window is shown.

If we consider the values from table 4 using optimum window size 24 change in patterns are identified at different time points; at 48 time point (second optimum window), 72 time point (third optimum window), 96 time point (fourth optimum window), 102 time point (fifth optimum window). In table 3 with fixed window size 12 changes in pattern are identified at 4 different windows 24, 36, 48, 60 (second, third, fourth, fifth fixed windows respectively). If execution time is compared, optimum window size execution times at 0.15 & 1.71 are half the execution times 0.31, 1.88 of fixed window size. Even though both fixed and optimum window sizes identifies all the changes in pattern, execution time differs. Since execution time is an important aspect in network sensor we consider our method as optimum method to increase the performance by decresing the execution time.

5.4.2 Fixed Tolerance value Vs Optimum Tolerance Value

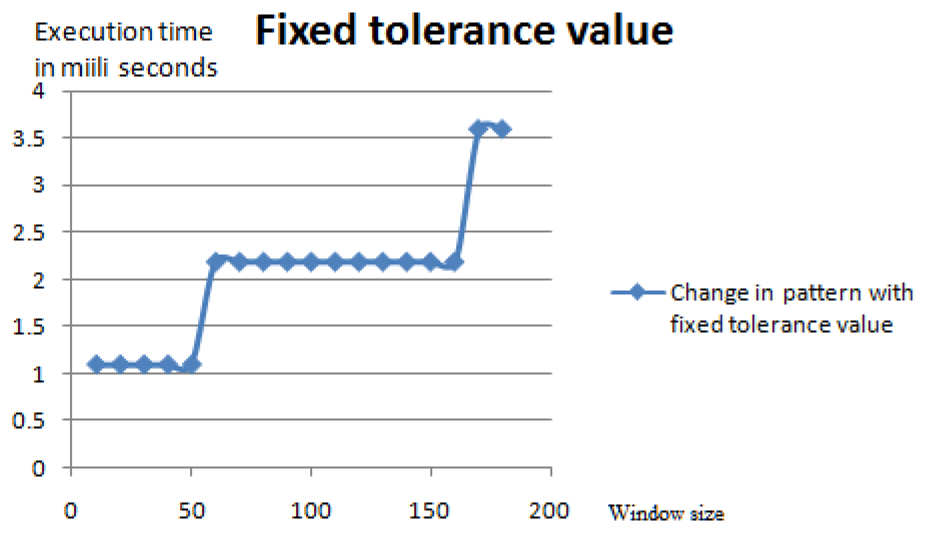
In this section results of input 2 data pattern are used to compare the accuracy of identifying the change in pattern with optimum tolerance value and fixed tolerance value.



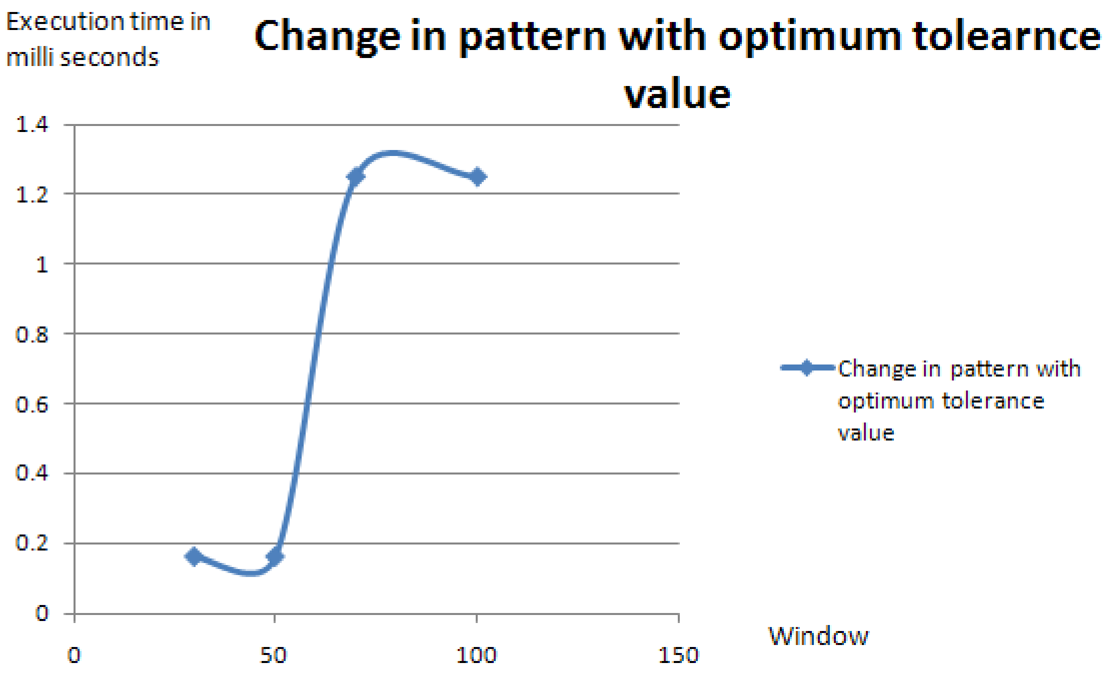
a) Input 2 from the attacker with changing frequency

Fig 17 (b) and (c) shows the results for a fixed window size of 10 using data input 2. Here the x axis represents the window (refer figure 17 (a)) and the y axis represents execution time to identify a change in pattern in milliseconds. In (b) we considered fixed tolerance value of more than 5 change in pattern were identified at 20, 30, 40, 50, 60 etc. time points. In 17 (c) changes in pattern were identified four times at 25, 40, 68, 90 time points.

\



b) Change in pattern identified with fix tolerance value



c)Change in pattern with Optimum tolerance value

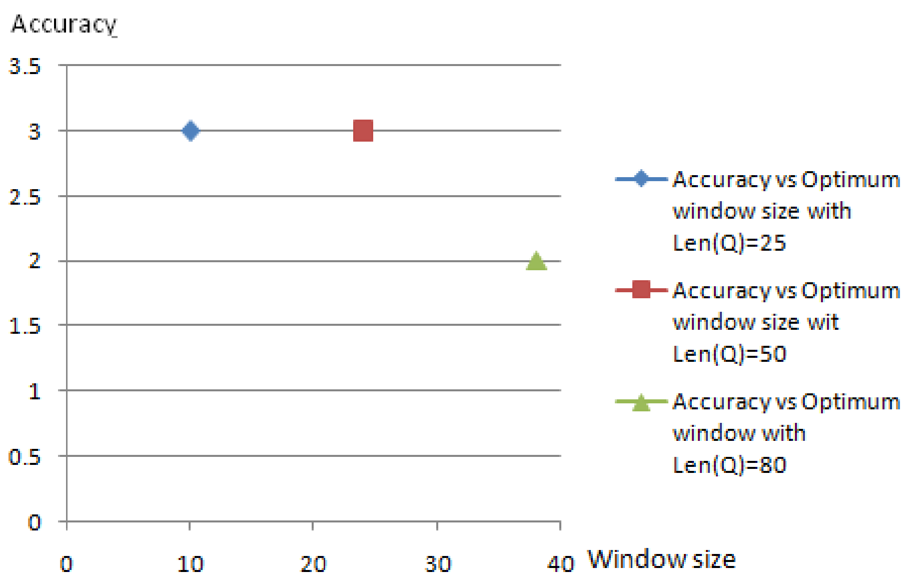
Figure 17 Performance analysis of Optimum tolerance value to Fixed tolerance

In 17(a) input pattern we only see 4 changes in pattern at time points 25, 40 , 68 and 90 This is approximately equal to 17(c) with uses optimum tolerance value. Therefore we can conclude that accuracy of optimum tolerance value is better than the accuracy of fixed tolerance value.

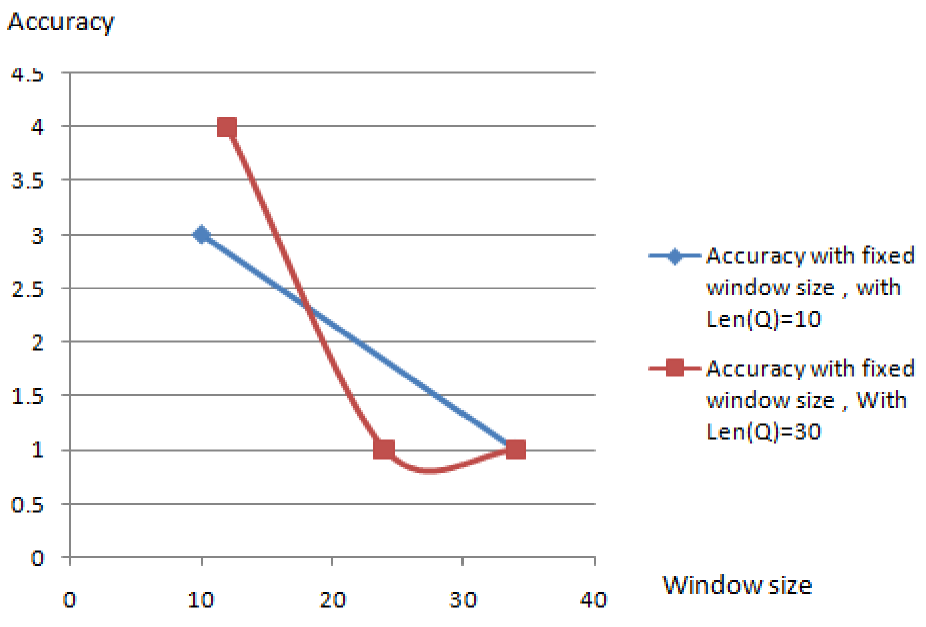
From the above results with window size and tolerance value we can say that optimum tolerance value does improve the accuracy of data and optimum window size does improve the performance by decreasing execution time.

5.4.3 Accuracy vs Window sizes

In this section we study the accuracy of identifying the change in pattern with optimum window size and fixed window size. We considered the range of value for accuracy from 4 to 1. If the accuracy is 4 which means that all four pattern changes have been identified, that is, the detection rate. If the accuracy is 1 then only 1 of the four pattern changes are detected.



a) Accuracy vs Optimum Window size



b) Accuracy vs Fixed window size

Figure 18 Accuracy of data vs window sizes with different length of query sequences.

a) Accuracy Vs Optimum window size		
Optimum Window size	**Accuracy**	**Len(Q)**
10	3	25
24	3	50
38	2	80

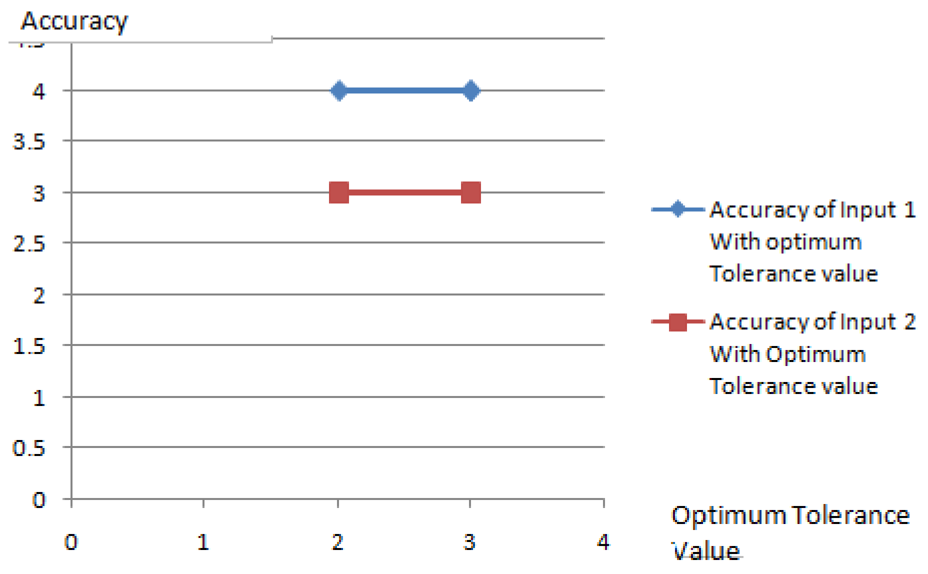
b) Accuracy Vs Fixed window size		
Fixed Window size	**Accuracy**	**Len(Q)**
12	4	30
24	1	30
34	1	30
34	1	10
10	3	10

Table 5 Accuracy measured with Fixed window size and Optimum window size

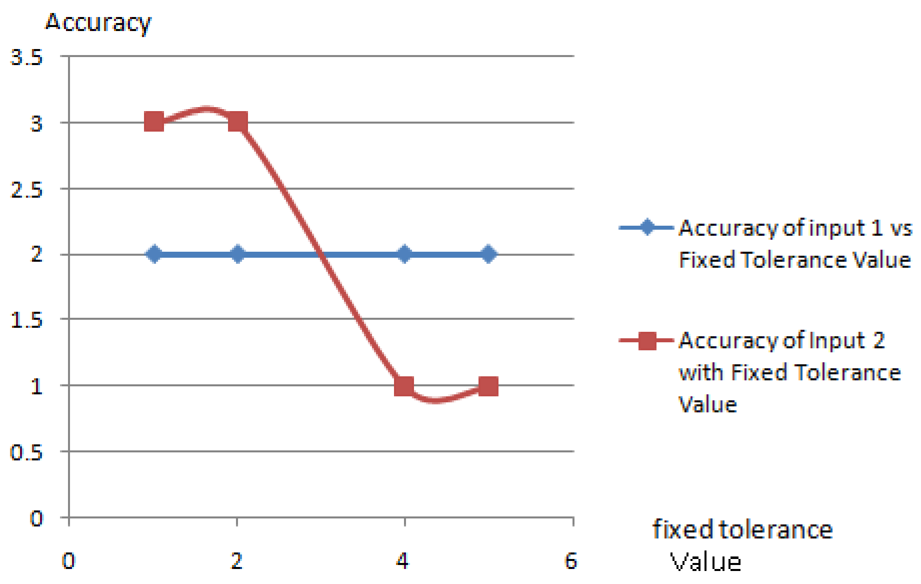
Figure 18 gives the accuracy vs window size graph where x-axis measures the window sizes and y-axis measures the accuracy of data obtained. In table 5 (a) optimum window size is considered where the accuracy of detection obtained is 3 or 2. In 18 (b) for fixed window size the accuracy obtained between 4 or 1. The accuracy of detection decreases as the fixed window size increases. At small window sizes the detection rate is very high, but there will be a lot of false alarms. From above fig 18 the overall accuracy of optimum window size is therefore more than the accuracy of the fixed window size.

5.4.4 Accuracy vs Tolerance Value

In figure 19 we compare the accuracy of identifying the change in pattern with optimum tolerance value and fixed tolerance value. We considered the range of value for accuracy from 4 to 1. If the accuracy is 4 which means that all four pattern changes have been identified, that is, the detection rate. If the accuracy is 1 then only 1 of the four pattern changes are detected.



a) Accuracy vs Optimum Tolerance Value



b) Accuracy vs Fixed Tolerance values

Figure 19 Accuracy vs Optimum Tolerance value and Fixed tolerance value.

a) Accuracy vs Optimum tolerance value				
Old Tolerance limit	Tolerance Limit	Tolerance value	Accuracy of Input 1	Accuracy of Input 2
2	2	0	4	3
5	3	1	4	3
7	3	1	4	3

b) Accuracy with Fixed Tolerance Value		
Fixed Tolerance	Accuracy of Input 1	Accuracy of Input 2
1	2	3
2	2	3
4	2	1
5	2	1

Table 6 Accuracy vs Optimum tolerance and fixed tolerance value.

When the optimum tolerance value is considered the accuracy of identifying the change in pattern is more accurate when compared to fixed tolerance value. In fig 19 (a) accuracy is obtained with optimum tolerance values. In fig 19 (b) accuracy is calculated with different fixed tolerance values. For the fixed tolerance values, the tolerance value depends on the previous tolerance value. As the tolerance value increases the accuracy of detecting pattern change decreased. From the above results we can conclude that optimum tolerance value is more accurate in identifying the change in pattern compared to a fixed tolerance value. For both input 1 and input 2, we roughly estimate the detection rate using optimum tolerance value to be approximately 80% better than the detection rate using fixed tolerance value.

In table 7 based up on the above results we estimate the detection accuracy percentage with optimum window size and optimum tolerance value. This value will be different for different input data patterns.

Table 7 Performance analysis of fixed & optimum of window size and tolerance values.

	Fixed values	Optimum Values	Results
Window size	Eg : Window =12	Eg: Optimum = 24, 38 etc changes according to length of query sequence	Optimum window size shows 50 % (approx) performance increase over Fixed window size.
Tolerance value	Eg: Max=5, min =3	Eg: Max= 2, Min = 1	Optimum window size gives 80% (approx) more accuracy of detection over Fixed window size

5.5 Adaptation

After the change in pattern is detected, self adaptation algorithm adapts to the new pattern by changing the query sequence, optimum window size based on the length of the new query sequence and the tolerance value for better performance and more accurate detection of pattern change.

Our results show that if the query sequence is small then a fixed optimum window is sufficient and there is no need to determine an optimum window. However, in general small attacks of small query sequences are unlikely in which the proposed approach is executed. One concern not considered in this work is the index of the window size list.

As the number of indexes increase, the execution time decreases [3]. We leave for future work to find the optimum value of index.

CHAPTER VI

CONCLUSION

In this thesis, we proposed a method to increase the performance of self-adaptation in a deception framework. In deception framework whenever the attacker in a network tries to attack neighboring nodes, the deception framework after detecting the attack, tries to outsmart the attacker by self adapting his strategies and responding as the attacker anticipates. To increase the performance we proposed a method to calculate the optimum window size and optimum tolerance value. This method increases the performance by reducing the execution time and increasing the accuracy of detecting pattern changes. The results show that using our approach, the execution time to identify pattern changes by using an optimum window size decreased by 50% and the accuracy of detection using an optimum tolerance value increased by 80%.

There are a number of deficiencies with the proposed approach which can be addressed in future work. As the number of indexes of the window size list increases the performance of the deception framework also increases. The index size is directly proportional to the execution time. Larger number of indexes means more memory is needed. Determining the optimum index number for window size list is left for future work. The proposed approach should also be applied to other attacks besides the sink-hole attack.

CHAPTER VII

DEFINITIONS

Time point:	The smallest unit of time over which the system collects data. eg: ten seconds.
Basic window:	A consecutive subsequence of time points over which the system maintains a digest e.g., two minutes.
Secondary window:	a user-defined consecutive subsequence of basic windows over which the user wants statistics e.g., an hour.
Data Sequence:	Input data values collected from the attacker over a time period.
Query Sequence:	Steady rate of expected input pattern which is used to compare with the input data pattern to identify change in pattern.
Tolerance limit:	The permitted range of maximum and minimum value in input data pattern.
Tolerance value:	Maximum deviation allowed in the current input pattern from the query sequence.
Window size:	Window size is length into which query sequence and input data sequence are divided to identify the pattern change.

REFERENCES

[1] Phil L., Nelson L., Matt W., “TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications”, Proceedings First ACM Conference on Embedded Networked Sensor Systems (SenSys'03), pp. 126-137, 1, 2003.

[2] Ruiyi Z.M.V.M., Johnson P.T., “ Deception framework for sensor networks", in Proceedings 3rd International Conference on Security and Privacy in Communication Networks (SecureComm 2007), 2007.

[3] Seung-Hwan L., Heejin P., & Sang-Wook K., “Using multiple indexes for efficient subsequences matching in time series databases”, Information Sciences, Vol. 177, No 24, pp. 5691-5706, 2007.

[4] Yang-Sae M., Kyu-Young W.,& Wook-Shin H., “General match: a subsequence matching method in time series databases based on generalized windows”, Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, pp. 382-393, 2002.

[5] Kil R.M., Seon Hee P., & Seunghwan K., “Optimum window size for time series prediction”, Proceedings of the 19th Annual International Conference of the IEEE, Vol 4, No 30 Oct-Nov 2, 1997, pp. 1421-1424, 1997.

[6] Yang-Sae M., Kyu-Young W., & Woong-Kee L., “Efficient time-series subsequence matching using duality in constructing windows”, Information Systems, Vol. 26, No 4, pp. 279-293, 2001.

BIBILOGRAPHY

[7] Wook-Shin H., Jinsoo L., Yang-Sae M., and Haifeng J., “Ranked Subsequence matching in time-series databases”, Proceedings of the 33rd International Conference on Very Large Data Bases, pp. 423-434, 2007.

[8] Deepayan C., Christos F., “F4: Large-Scale Automated Forecasting Using Fractals”, Proceedings of the 11th International Conference on Information and Knowledge Management, pp. 2-9, 2002.

[9] Tripti N., Veena B., “ Time series: similarity search and its application”, www.iitk.ac.in/ime/veena/PAPERS/icsci05.pdf, [last accessed-April 2009]

[10] Ranganathan M., Faloutosos C., and Manolopoulos Y., “Fast subsequence matching in time-series databases”, Proceedings of the ACMSIGMOD Conference on Management of Data, pp. 419-429, 1994.

[11] Guttman A., “R-trees: A dynamic index structure for spatial searching”, Proceedings of ACM SIGMOD, pp. 47–57, 1984.

[12] Zhang S., Perng C.S., Wang H., and D. S. Parker S.D., “Landmark: A new technique for similarity based pattern querying in time series databases”, Proceedings 16th International Conference of Data Engineering, pp. 33-42, 2000.

[14] Lazos L., and Poovendran R., “Secure broadcast in energy-aware wireless sensor networks," Proceedings IEEE International Symposium on Advances in Wireless Communications (ISWC'02), 2002.

[15] Wood A.D., and Stankovic A.J., “Denial of service in sensor networks," Computer, vol. 35, no. 10, pp. 54-62, 2002.

[16] Project T.H., “ Know Your Enemy”, Boston: Addison-Wesley, 2002.

[17] Rowe N., “Designing good deceptions in defense of information systems," Proceedings of the 20th Annual Computer Security applications Conference, pp. 418-427, 2004.

[18] Rowe N., “A model of deception during cyber-attacks on information systems", Proceedings IEEE First Symposium on Multi-Agent Security and Survivability, pp. 21-30, 2004.

[19] Cohen F., “A mathematical structure of simple defensive network deceptions, "Computers and Security, vol. 19, no. 6, pp. 520 -528, 2000.

[20] Rakesh A., King-Ip L., Harpreet S., Kyuseok S., “Fast similarity search in the presence of noise, scaling and translation in time-series databases”, Proceedings of 21st VLDB Conference, pp. 490–501, 1995.

RELANGI PADMAJA

Candidate for the Degree of

Master of Science in Computer Science

Thesis: DECEPTION FRAMEWORK – MULTIPLE INDEXING IN SELF ADAPTATION

Major Field: Computer Science

Biographical:

Personal Data:

Born in Hyderabad, Andhra Pradesh, India on February 13,1982.

Education:

Completed the requirements for the Master of Science in Computer Science at Oklahoma State University, Stillwater, Oklahoma in July, 2009.

Received the B.S degree from JNTU University, Andhra Pradesh, India, 2003, in Computer Science.

Name: Padmaja Relangi Date of Degree: July, 2009

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: DECEPTION FRAMEWORK – MULTIPLE INDEXING IN SELF ADAPTATION

Pages in Study: 58 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study:

In this thesis we propose deception as a security mechanism. In deception an attacker's behavior is manipulated by sending him misleading information. One of the critical phases in deception is self-adaptation where the defender has to adapt to the changing pattern of the attacker and respond accordingly. However, determining the change pattern must be efficient and accurate. In this thesis we propose a novel algorithm that is based on a windowing scheme and a tolerance value. Our approach aims to find the optimum window size and tolerance value from an efficiency and accuracy perspective.

Findings and Conclusions:

In this thesis we derive the optimum window size and optimum tolerance value which is adaptable to the reference (query sequence) or attack. Simulation results show that using our approach, execution time with optimum window value is less when compared to a fixed window size. Similarly, identifying a change in pattern is more accurate with the proposed optimum tolerance than with a fixed tolerance value.

ADVISER'S APPROVAL: Dr.Johnson Thomas