EE 418 CRYPTOGRAPHY AND NETWORK SECURITY

Project 1: Clock Skew and Cloaking

Mike Warren, Zachary de la Cruz, Hugo Baldner
0323799 , 1572779 , 1471121

Task 1: Implement the class IDS, Simulation and Read the Texts:

While this Class was originally implemented by Zach we decided that the code would be better analyzed all in one language and so after the creation of it in MATLAB Zach passed the code to Mike who proceeded to swiftly translate it into Python. Mike on the other hand programmed the simulation section and translated the IDS section. Meanwhile Hugo did the reading and answering of the text questions and the compilation of our work. While all this code will be turned in separately in its whole here we will place the snippets that were made by our group:

IDS:

```
94      # ====================== Start of Your Code ========================
95      # TODO: Compute curr_avg_offset_us and curr_acc_offset_us for state-of-the-art IDS
96
97      # Your code goes here.
98
99      #Calulates offset values based on array a
100     aarr = []
101     # Verify range is 2-N and not 1-N
102     for i in range(2, self.N,1):
103       aarr.append(a[i]-(a[1]+(i-1)*prev_mu_T_sec))
104
105
106     #Computes current average offset
107     curr_avg_offset_us = (1/(self.N-1)*np.sum(aarr))
108     test.append(curr_avg_offset_us)
109
110
111     #Establish Current Accumulated Offset
112     curr_acc_offset_us = (prev_acc_offset_us + np.abs(curr_avg_offset_us))
113
114     # ====================== End of Your Code ========================
```

```
117     # ====================== Start of Your Code ========================
118     # TODO: Compute curr_avg_offset_us and curr_acc_offset_us for NTP-based IDS
119
120     # Your code goes here.
121
122     #Calculate current average offset
123     curr_avg_offset_us = self.T_sec - (a[self.N-1] - a0)/self.N
124
125     #Update accumulated offset
126     curr_acc_offset_us = prev_acc_offset_us + self.N*curr_avg_offset_us
127
128     # ====================== End of Your Code ========================
```

```
140     # ====================== Start of Your Code ========================
141     # RLS algorithm
142     # Inputs:
143     #   t[k] -> time_elapsed_sec
144     #   P[k-1] -> prev_P
145     #   S[k-1] -> prev_skew
146     #   e[k] -> curr_error
147     #   lambda -> l
148     #
149     # Outputs:
150     #   P[k] -> curr_P
151     #   S[k] -> curr_skew
152     #
153     # TODO: Implement the RLS algorithm
154
155     # Your code goes here.
156
157     #Set lambda to 0.9995
158     l = 0.9995
159
160     #Establishes G[k]
161     G = (l**-1)*prev_P*time_elapsed_sec/(1+(1**-1)*(time_elapsed_sec**2)*prev_P)
162
163     #Computes P(k)
164     curr_P = (l**-1)*(prev_P-G*time_elapsed_sec*prev_P)
165
166     #Computes S(k)
167     curr_skew = prev_skew+G*curr_error
168
169
170     # ====================== End of Your Code ========================
```

```
200     # ====================== Start of Your Code ========================
201     # TODO: 1) Normalize curr_error_sample, 2) compute curr_L_upper and curr_L_lower
202     # Store the normalized error in `normalized_error`
203
204     # Your code goes here.
205
206     #Normalizes curr_error_sample
207     normalized_error = (curr_error_sample-mu_e)/sigma_e
208
209     #Computes curr_L_upper and curr_L_lower
210     curr_L_upper = max(0, prev_L_upper+normalized_error-kappa)
211     curr_L_lower = max(0, prev_L_lower-normalized_error-kappa)
212
213
214     # ====================== End of Your Code ========================
```

SIMULATION:

```python
# ====================== Start of Your Code ======================
# Example: Plot accumulated offset curve for 0x184.
#  plt.plot(ids['184-sota'].elapsed_time_sec_hist, ids['184-sota'].acc_offset_us_hist, label='0x184')

# Your code goes here.
plt.plot(ids['184-sota'].elapsed_time_sec_hist, ids['184-sota'].acc_offset_us_hist, label='0x184')
plt.plot(ids['3d1-sota'].elapsed_time_sec_hist, ids['3d1-sota'].acc_offset_us_hist, label='0x3d1')
plt.plot(ids['180-sota'].elapsed_time_sec_hist, ids['180-sota'].acc_offset_us_hist, label='0x180')
plt.legend()
plt.title("Accumulated Offset Curve for State-of-The-Art IDS, Batch Size %i" %N)
plt.xlabel("Elapsed Time (sec)")
plt.ylabel("Accumulated Offset (us)")
#plt.savefig('C:\\Users\\sirwi\\Documents\\UW\\EE 418\\Project 1\\Task3sota.png', dpi = 300)
plt.show()

# ====================== End of Your Code ======================
```

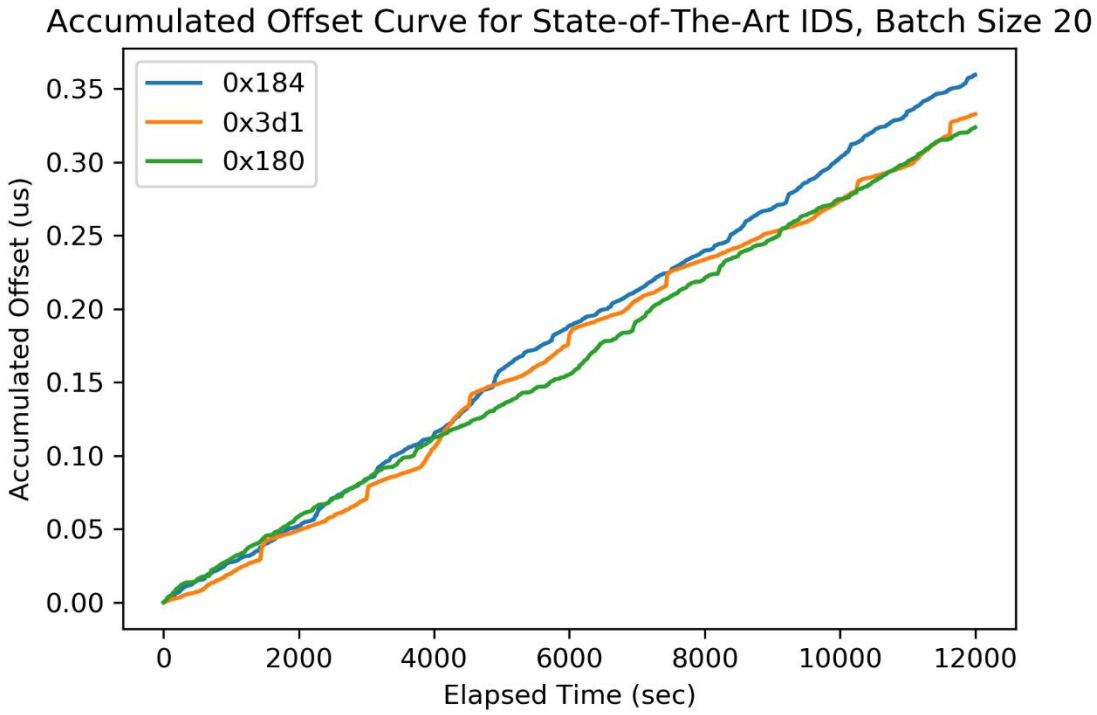
```python
# ====================== Start of Your Code ======================

# Your code goes here
plt.plot(ids['184-ntp'].elapsed_time_sec_hist, ids['184-ntp'].acc_offset_us_hist, label='0x184')
plt.plot(ids['3d1-ntp'].elapsed_time_sec_hist, ids['3d1-ntp'].acc_offset_us_hist, label='0x3d1')
plt.plot(ids['180-ntp'].elapsed_time_sec_hist, ids['180-ntp'].acc_offset_us_hist, label='0x180')
plt.legend()
plt.title("Accumulated Offset Curve for NTP IDS, Batch Size %i" % N)
plt.xlabel("Elapsed Time (sec)")
plt.ylabel("Accumulated Offset (us)")
#plt.savefig('C:\\Users\\sirwi\\Documents\\UW\\EE 418\\Project 1\\Task3NTP.png', dpi = 300)
plt.show()

# ====================== End of Your Code ======================
```

```python
# ====================== Start of Your Code ======================

# Your code goes here.
#Import data files 180 and 3d1
data_184 = import_data('C:\\Users\\sirwi\\Documents\\UW\\EE 418\\Project 1\\src/data/184.txt')
data_3d1 = import_data('C:\\Users\\sirwi\\Documents\\UW\\EE 418\\Project 1\\src/data/3d1.txt')

#Set N size
N = 20

# Construct a new data-set with the first 1000 batches of data_184,
# followed by 1000 batches of data_3d1. That is, the masquerade
# attack occurs at the 1001-st batch.
data_184 = np.asarray(data_184[0:1000 * N]) - data_184[0]  # Relative timestamps
data_3d1 = np.asarray(data_3d1[0:1000 * N]) - data_3d1[0]  # Relative timestamps
data = np.append(data_184, data_184[-1] + 0.1 + data_3d1)  # The 1st spoofed message occurs exactly 0.1 sec
                                # (the period) after the last legitimate message.

ids = IDS(T_sec=0.1, N=N, mode='ntp-based')

batch_num = 2000
for i in range(batch_num):
    batch = np.asarray(data[i * N:(i + 1) * N])
    ids.update(batch)

#Plot results for NTP Masquerade Attack
plt.plot(ids.L_upper_hist, label='Upper Control Limit')
plt.plot(ids.L_lower_hist, label='Lower Control Limit')
plt.xlabel('Number of Batches')
plt.ylabel('Control Limits')
plt.title('Control Limits for NTP IDS Masquerade Attack')
plt.legend()
#plt.savefig('C:\\Users\\sirwi\\Documents\\UW\\EE 418\\Project 1\\Task5MasqueradeNTS.png', dpi = 300)
plt.show()

# ====================== End of Your Code ======================
```

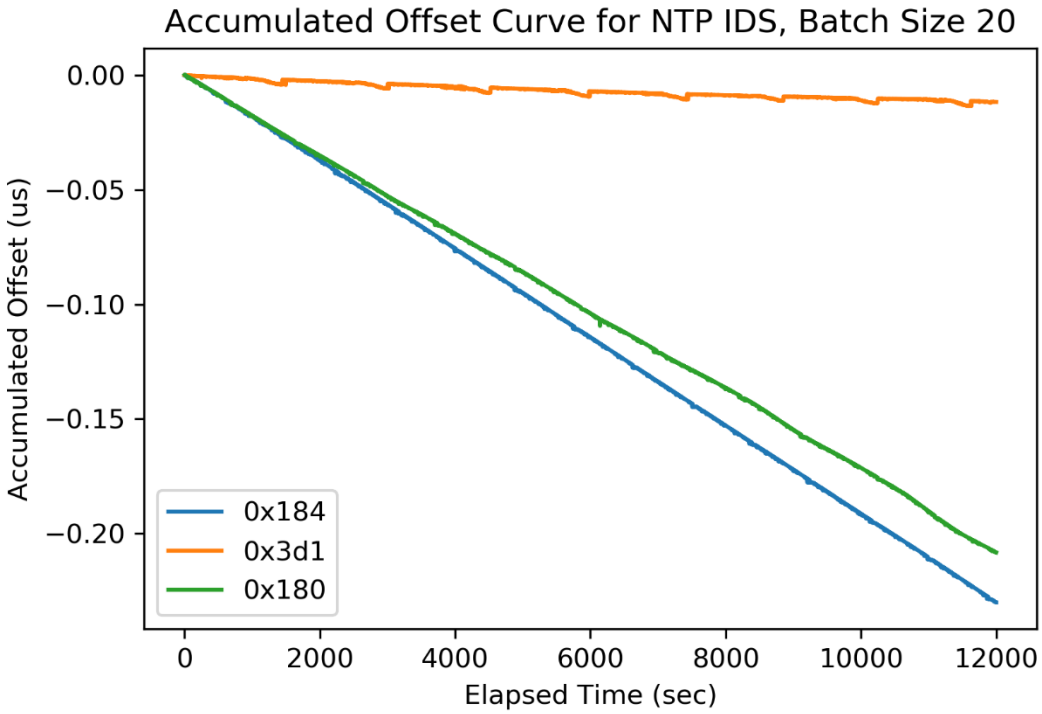
```python
# ====================== Start of Your Code ======================

# Your code goes here.
#Import data
data_184 = import_data('C:\\Users\\sirwi\\Documents\\UW\\EE 418\\Project 1\\src/data/184.txt')
data_180 = import_data('C:\\Users\\sirwi\\Documents\\UW\\EE 418\\Project 1\\src/data/180.txt')

#Set N size
N = 20

# Construct a new data-set with the first 1000 batches of data_184,
# followed by 1000 batches of data_180. That is, the cloaking
# attack occurs at the 1001-st batch.
data_184 = np.asarray(data_184[0:1000 * N]) - data_184[0]  # Relative timestamps
data_180 = np.asarray(data_180[0:1000 * N]) - data_180[0]  # Relative timestamps
data = np.append(data_184, data_184[-1] + 0.100029 + data_180)  # The 1st spoofed message occurs exactly 0.1 sec + 29 us
                                # (the period) after the last legitimate message.

ids = IDS(T_sec=0.1, N=N, mode='state-of-the-art')

batch_num = 2000
for i in range(batch_num):
    batch = np.asarray(data[i * N:(i + 1) * N])
    ids.update(batch)

#Plot results of SOTA Cloaking Attack
plt.plot(ids.L_upper_hist, label='Upper Control Limit')
plt.plot(ids.L_lower_hist, label='Lower Control Limit')
plt.xlabel('Number of Batches')
plt.ylabel('Control Limits')
plt.title('Control Limits for State-of-The-Art IDS Cloaking')
plt.legend()
#plt.savefig('C:\\Users\\sirwi\\Documents\\UW\\EE 418\\Project 1\\Task6CloakingSOTA.png', dpi = 300)
plt.show()
# ====================== End of Your Code ======================
```

```python
# ====================== Start of Your Code ======================

# Your code goes here.
#Import data
data_184 = import_data('C:\\Users\\sirwi\\Documents\\UW\\EE 418\\Project 1\\src/data/184.txt')
data_180 = import_data('C:\\Users\\sirwi\\Documents\\UW\\EE 418\\Project 1\\src/data/180.txt')

#Set N size
N = 20

# Construct a new data-set with the first 1000 batches of data_184,
# followed by 1000 batches of data_180. That is, the cloaking
# attack occurs at the 1001-st batch.
data_184 = np.asarray(data_184[0:1000 * N]) - data_184[0]  # Relative timestamps
data_180 = np.asarray(data_180[0:1000 * N]) - data_180[0]  # Relative timestamps
data = np.append(data_184, data_184[-1] + 0.100029 + data_180)  # The 1st spoofed message occurs exactly 0.1 sec + 29 us
                                # (the period) after the last legitimate message.

ids = IDS(T_sec=0.1, N=N, mode='ntp-based')

batch_num = 2000
for i in range(batch_num):
    batch = np.asarray(data[i * N:(i + 1) * N])
    ids.update(batch)

#Plot results of NTP Cloaking Attack
plt.plot(ids.L_upper_hist, label='Upper Control Limit')
plt.plot(ids.L_lower_hist, label='Lower Control Limit')
plt.xlabel('Number of Batches')
plt.ylabel('Control Limits')
plt.title('Control Limits for NTP IDS Cloaking')
plt.legend()
#plt.savefig('C:\\Users\\sirwi\\Documents\\UW\\EE 418\\Project 1\\Task6CloakingNTP.png', dpi = 300)
plt.show()
# ====================== End of Your Code ======================
```

Task 2: Plot accumulated offset curves as function of the elapsed time for the three messages:



Task 3: Repeat Task 2 for the NTP-based IDS with N = 20. Compare the slopes of the curves from Tasks 2 and 3, and comment on the similarity of the three messages in terms of estimated clock skew from the perspective of the IDS:

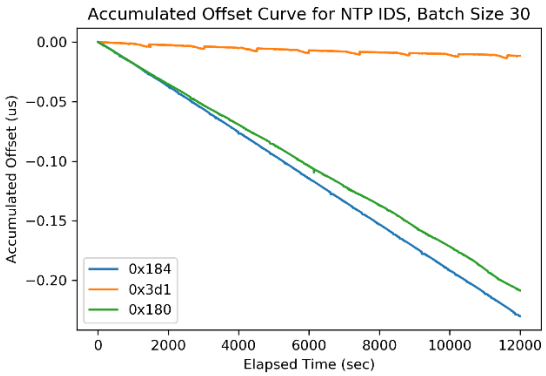


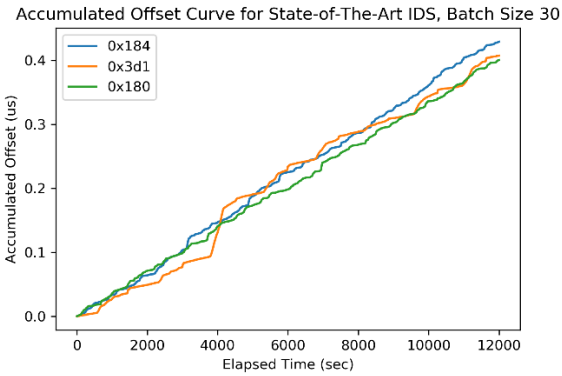
We see very clearly in the NTP IDS the differences between the Offset curves with signals from 184 and 180 having relatively similar offsets while signal 3d1 clearly has a different offset.

Compare this the previous plot where it was very difficult to detect differences in offsets. Another clear difference is the NTP offset seems to be much more stable; as it produces the same slope almost always for the signals.

This bodes badly for the SOTA as if we cannot truly tell the difference between the regular signals how can we be expected to detect difference in masked signals.

Task 4: Repeat Tasks 2 and 3 with N = 30. Comparing the four figures from Tasks 2 through 3, comment on the consistency of clock skew estimation (i.e., the slope of the curve for the same message should be the same regardless of N) for the state-of-the-art and the NTP-based IDSs:

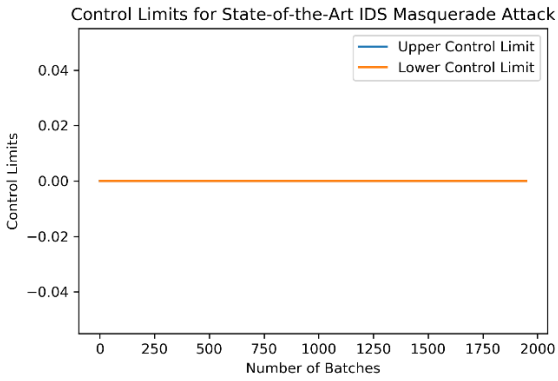


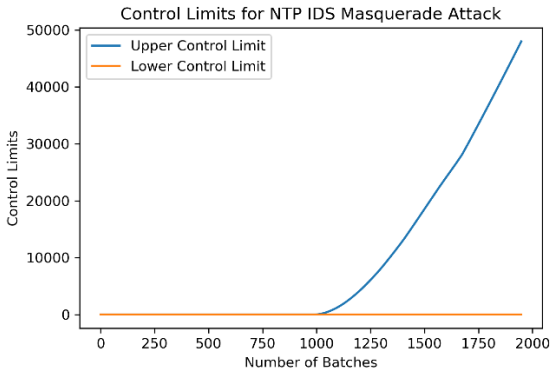


Using Batch 30 we see once again very pro NTP offset calculation results. The curves from this offset accumulation curve are seemingly unchanged. Meanwhile we clearly see a change within the SOTA curve for signal 3d1 there seems to be a much larger jump at around the 4000 second mark of the offset accumulation.

Once again this inconsistency when using different batch sizes bodes badly for SOTA offset calculation.

Task 5: Simulate the masquerade attack in Scenario 1. In this simulation, we first feed 1000 batches of arrival timestamps of message 0x184 to the IDS, and then feed 1000 batches of arrival timestamps of message 0x3d1 to it (batch size is 20). That is, the masquerade attack occurs at the 1001-st batch, and is detected if either upper or lower control limit exceeds the threshold T = 5. Plot control limits as function of number of batches for the state-of-the-art and the NTP-based IDSs. Which IDS can detect the masquerade attack? Why?

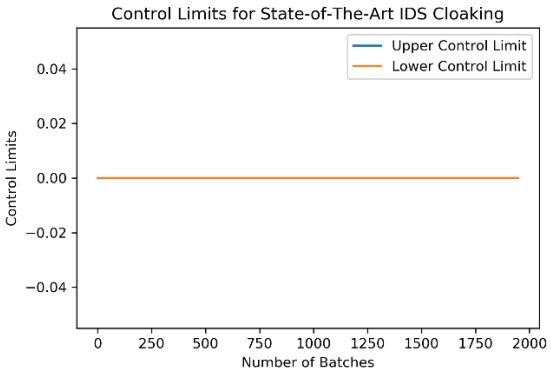


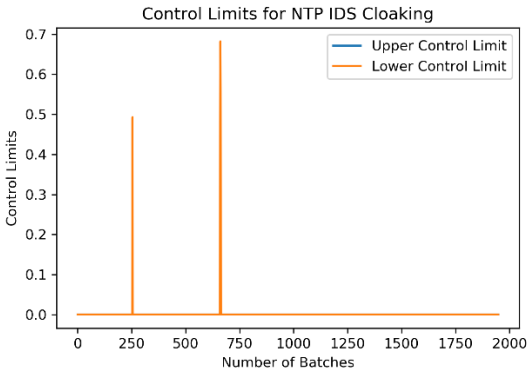


Here we clearly see the previously stated incompetence of SOTA offset calculation being brought to fruition. Even when the attacker is using very simple methods of masking his attack (here he is not even attempting to replicate the original signals clock cycle) the offset calculations are too noisy to begin with for us to detect the manipulation attack. We see this because both the upper and lower control limits are unchanged once the attack begins. This will of course correspond to a very low (unchanged in this case) MSI and we will therefore not detect the attack.

On the other hand, NTP offset calculation shows strength here as it clearly has the ability to recognize and display this new offset. This shows itself in the dramatic increasing of the upper control limit. This of course would change the MSI number allowing for us to easily detect the attack. It seems that in the case of this attack (at least) NTP offset calculation is superior as it allows you to more easily see modifications done to the signal.

Task 6: Simulate the cloaking attack in Scenario 2. In this simulation, we first feed 1000 batches of arrival timestamps of message 0x184 to the IDS, and then feed 1000 batches of arrival timestamps of message 0x180 to it (batch size is 20). Plot control limits as function of number of batches for the state-of-the-art and the NTP-based IDSs. Which IDS can detect the cloaking attack? Why? Comparing masquerade and cloaking attacks, comment on the limitations of a clock-skew based IDS.





With regards to the SOTA offset calculation we see exactly what we expected when tasked with an attacker who is even more intelligent and powerful than our previous attacker it will also fail to detect changes in the signal. This speaks to the inconsistencies and ineffectiveness of this offset calculation method. We see this once again through the lack of change in the upper and lower control limits. This will of course echo into the MSI calculation showing no change.

What is more interesting here is the NTP offset calculation method. While we are not able to detect the attack nearly as effectively as before however there is some irregularities in the lower control limits. This is likely due to the dependence of the NTP offsets on the previous batch offset and the current last batch offset. At specific batch numbers we see that these numbers end up detecting the attacks due to the end points of the batches likely lining up with the attack point. The end result of this is that by changing batch numbers repeatedly we can detect that this attack happened due to the changing control bounds.

Task 7: Read references [2] and [3], and answer the following questions:

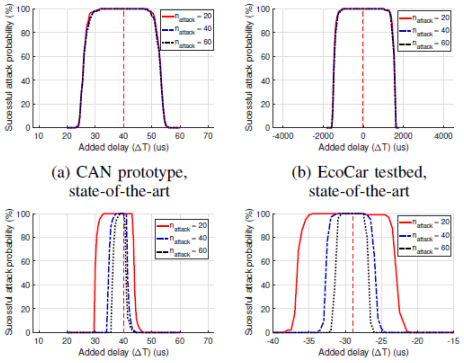
1. Briefly explain how the adversary chooses DELTA T for the cloaking attack on the clock skew detector.

$$\Delta T = \frac{(S_A - S_B)}{1 + S_B} \cdot T$$

In other words the adversary needs to choose the DELTA T so that it is based on an estimate of the period of the target message (measured by its local clock of course).

2. What is Maximum Slackness Index (MSI), and what does it measure? Based on Fig. 8 of [3], briefly comment on the performance of cloaking attack on an IDS in terms of MSI.

In reality no attacker will be able to perfectly replicate DELTA T (due to hardware and software limitations) so we use MSI as a measure of how sure we are that the irregularities detected in clock and message periods are actual attacks and not simply normal errors or delays. What we see in figure 8 is that using this we can detect attacks better when it is measured in NTP-based offset estimation. In fact it seems that the difference is night and day on the detection capabilities using NTP vs SOTA offset calculations



(a) CAN prototype, state-of-the-art

(b) EcoCar testbed, state-of-the-art

(c) CAN prototype, NTP-based

(d) EcoCar testbed, NTP-based

Fig. 8: Successful attack probability on the state-of-the-art IDS and the NTP-based IDS on the CAN bus prototype and EcoCar testbed with message period 100ms. For the value of $\Delta T = 40\mu s$ achieved in our hardware experiments (red dashed line), the attack was successful in all test cases. The width of each curve is equal to the ϵ-MSI for the given detector.

3. Based on [2], explain under what circumstances, two messages are likely to be highly correlated. Based on the analysis in Section IV-C and Fig. 10 in [3], explain under what circumstances, two messages are likely to be highly correlated.

Two messages are very likely to be correlated when they are sent from the same transmitter since the offset is likely to be very similar.

Two messages are even more likely to be correlated when they are both received from the same ECU and received consecutively. Also, with SOTA Offset calculation the same ECU is much more likely to by correlated.

4. Based on [3], describe how to launch the cloaking attack on the correlation detector, and briefly explain why it works.

To launch a cloaking attack on the correlation detector you want to first find two ECU's that are already highly correlated. Then you want to impersonate one of them using the other that way you can mask the messages for one with another highly correlated message.