# ECS 132 Fall 2020, Project

December 1, 2020

**Your name: Nicholas Treynor**

**Partner's name: James Lemkin**

**Any other collaborator(s):**

## Instructions

1. Please refer to "ECS_132_Project.pdf" for details.
2. You may in no circumstances upload your project to private tutoring websites such as CourseHero or Chegg. Remember all material related to this course is a property of the University of California and posting them is a violation of the copyright laws.
3. If you refer to a source (either a book or the internet), you must cite it.
4. You may work in groups of size of at most 2. If you collaborate with others, you must list their names.
5. Write your answers in R Markdown and submit the knitted pdf on Gradescope; for due date and other details see the Homework Policy and Schedule.

## 3 Design

### 3.1

Alice and Bob decide to use the following modulation scheme to map the bits to the inter-packet delay. A delay of 0.25 is used to encode a bit 0 and delay of 0.75 is used to encode a bit 1. Write a short R code that will generate the modified packet stream that contains the secret message.

charToRaw(x) - converts characters to hexadecimal value rawToBits(x) - converts hexadecimal value to bit values. The bits are in reverse order per decimal value, so the letter t corresponds to 74 in hexadecimal -> 01110100 in reality, but will return as 00101110 when using the function.

### Answer

We want to output "this is a secret message". To do so, we need to convert the message into binary, and then send along the individual bits that make up the binary using the encoding delays.

We need to loop over the generated strings carefully, stepping 16 forward then looping backwards over 8 integers.

```
#  your R code here
message = "this is a secret message"
rawMessage = charToRaw(message)
bits = rawToBits(rawMessage)

bitLength = (nchar(message) * 8)
sentPacketsTime = numeric(bitLength + 1)
sentPacketsStream = numeric(bitLength)
sentPacketsDelays = numeric(bitLength)
currTime = 0
index = 2
for (i in 1:nchar(message)) {
  for (x in 0:7) {
    if (bits[(i*8) - x] == 00){
      sentPacketsTime[index] = currTime + 0.25
      currTime = currTime + 0.25
      sentPacketsStream[index-1] = 0
      sentPacketsDelays[index-1] = 0.25
    } else {
      sentPacketsTime[index] = currTime + 0.75
      currTime = currTime + 0.75
      sentPacketsStream[index-1] = 1
      sentPacketsDelays[index-1] = 0.75
    }
    index = index + 1
  }
}

print(sentPacketsStream)
```

```
##   [1] 0 1 1 1 0 1 0 0 0 1 1 0 1 0 0 0 0 1 1 0 1 0 0 1 0 1 1 1 0 0 1 1 0 0 1 0 0
##  [38] 0 0 0 0 1 1 0 1 0 0 1 0 1 1 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0
##  [75] 1 0 0 0 0 0 0 0 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 0 1 1 0 0 0 1 1 0 1 1 1 0 0 1
## [112] 0 0 1 1 0 0 1 0 1 0 1 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 1 1 0 1 0 1 1 0
## [149] 0 1 0 1 0 1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 0 1 0 1 1 0 0 1 1 1 0
## [186] 1 1 0 0 1 0 1
```

```
print(sentPacketsDelays)
```

```
##   [1] 0.25 0.75 0.75 0.75 0.25 0.75 0.25 0.25 0.25 0.75 0.75 0.25 0.75 0.25 0.25
##  [16] 0.25 0.25 0.75 0.75 0.25 0.75 0.25 0.25 0.75 0.25 0.75 0.75 0.75 0.25 0.25
##  [31] 0.75 0.75 0.25 0.25 0.75 0.25 0.25 0.25 0.25 0.25 0.25 0.75 0.75 0.25 0.75
##  [46] 0.25 0.25 0.75 0.25 0.75 0.75 0.75 0.25 0.25 0.75 0.75 0.25 0.25 0.75 0.25
##  [61] 0.25 0.25 0.25 0.25 0.25 0.75 0.75 0.25 0.25 0.25 0.25 0.75 0.25 0.25 0.75
##  [76] 0.25 0.25 0.25 0.25 0.25 0.25 0.75 0.75 0.75 0.25 0.25 0.75 0.75 0.25 0.75
##  [91] 0.75 0.25 0.25 0.75 0.25 0.75 0.25 0.75 0.75 0.25 0.25 0.25 0.75 0.75 0.25
## [106] 0.75 0.75 0.75 0.25 0.25 0.75 0.25 0.25 0.75 0.75 0.25 0.25 0.75 0.25 0.75
## [121] 0.25 0.75 0.75 0.75 0.25 0.75 0.25 0.25 0.25 0.25 0.75 0.25 0.25 0.25 0.25
## [136] 0.25 0.25 0.75 0.75 0.25 0.75 0.75 0.25 0.75 0.25 0.75 0.75 0.25 0.25 0.75
## [151] 0.25 0.75 0.25 0.75 0.75 0.75 0.25 0.25 0.75 0.75 0.25 0.75 0.75 0.75 0.25
## [166] 0.25 0.75 0.75 0.25 0.75 0.75 0.25 0.25 0.25 0.25 0.75 0.25 0.75 0.75 0.25
## [181] 0.25 0.75 0.75 0.75 0.25 0.75 0.75 0.25 0.25 0.75 0.25 0.75
```

```
print(sentPacketsTime)
```

```
##   [1]  0.00  0.25  1.00  1.75  2.50  2.75  3.50  3.75  4.00  4.25  5.00  5.75
##  [13]  6.00  6.75  7.00  7.25  7.50  7.75  8.50  9.25  9.50 10.25 10.50 10.75
##  [25] 11.50 11.75 12.50 13.25 14.00 14.25 14.50 15.25 16.00 16.25 16.50 17.25
##  [37] 17.50 17.75 18.00 18.25 18.50 18.75 19.50 20.25 20.50 21.25 21.50 21.75
##  [49] 22.50 22.75 23.50 24.25 25.00 25.25 25.50 26.25 27.00 27.25 27.50 28.25
##  [61] 28.50 28.75 29.00 29.25 29.50 29.75 30.50 31.25 31.50 31.75 32.00 32.25
##  [73] 33.00 33.25 33.50 34.25 34.50 34.75 35.00 35.25 35.50 35.75 36.50 37.25
##  [85] 38.00 38.25 38.50 39.25 40.00 40.25 41.00 41.75 42.00 42.25 43.00 43.25
##  [97] 44.00 44.25 45.00 45.75 46.00 46.25 46.50 47.25 48.00 48.25 49.00 49.75
## [109] 50.50 50.75 51.00 51.75 52.00 52.25 53.00 53.75 54.00 54.25 55.00 55.25
## [121] 56.00 56.25 57.00 57.75 58.50 58.75 59.50 59.75 60.00 60.25 60.50 61.25
## [133] 61.50 61.75 62.00 62.25 62.50 62.75 63.50 64.25 64.50 65.25 66.00 66.25
## [145] 67.00 67.25 68.00 68.75 69.00 69.25 70.00 70.25 71.00 71.25 72.00 72.75
## [157] 73.50 73.75 74.00 74.75 75.50 75.75 76.50 77.25 78.00 78.25 78.50 79.25
## [169] 80.00 80.25 81.00 81.75 82.00 82.25 82.50 82.75 83.50 83.75 84.50 85.25
## [181] 85.50 85.75 86.50 87.25 88.00 88.25 89.00 89.75 90.00 90.25 91.00 91.25
## [193] 92.00
```

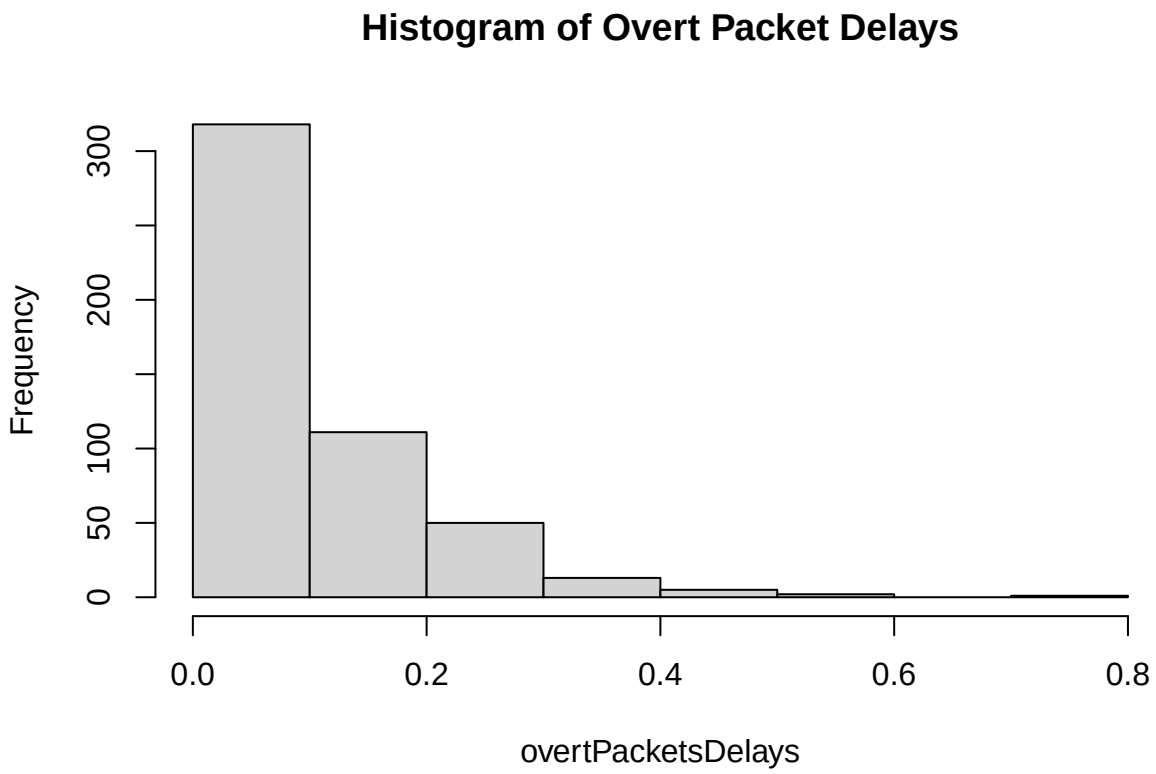
### 3.2

Plot the histogram of the inter-packet delays of the overt packet stream. Plot the histogram of the covert packet stream. Will Eve be suspicious?

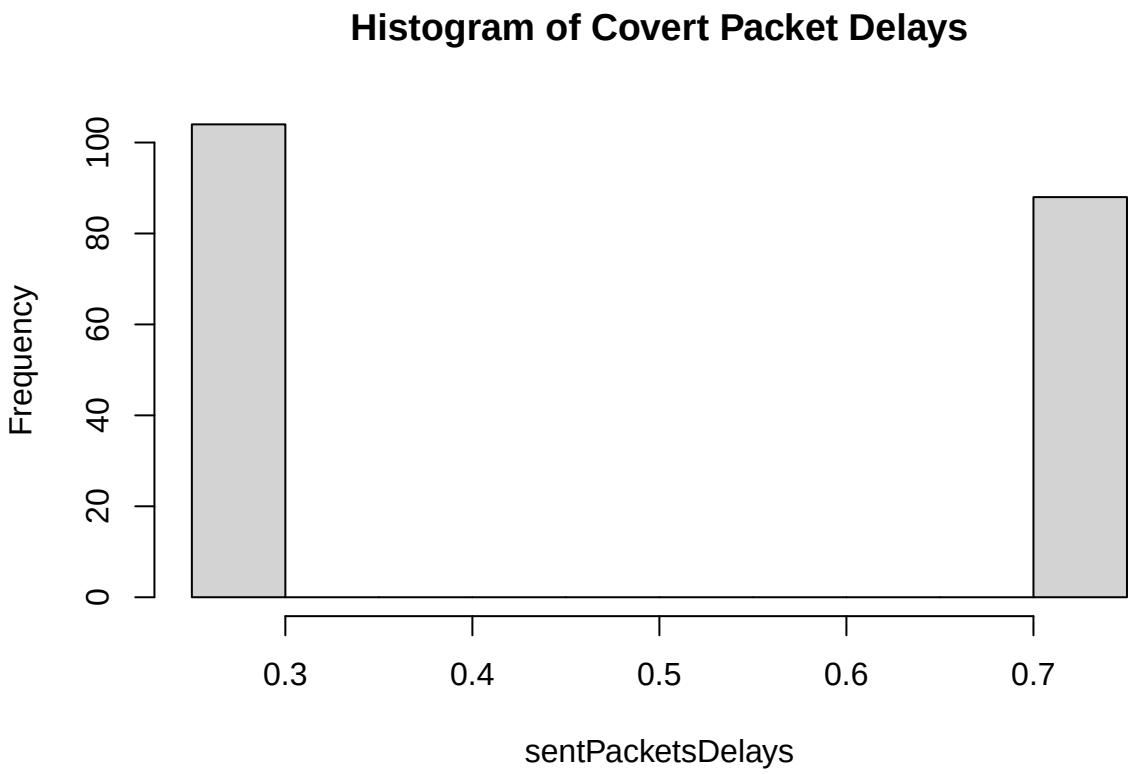
### Answer

```
#  your R code here
data <- read.csv("Traffic_data_orig.csv", header=TRUE)
overtPacketStream = data[,2]
x = length(overtPacketStream)
overtPacketsDelays = numeric(x-1)
currTime = 0
index = 2
for (i in 2:x-1){
  overtPacketsDelays[i-1] = overtPacketStream[i]-overtPacketStream[i-1]
}

hist(overtPacketsDelays, main = "Histogram of Overt Packet Delays")
```




```
hist(sentPacketsDelays, main = "Histogram of Covert Packet Delays")
```




There is a very clear difference in the two histograms, indicating that some sort of unusual data transmission is occurring. Eve would be suspicious if she were monitoring our packet delays.

### 3.3

Alice and Bob decide to use the following modulation scheme. Let $m$, $min$, and $max$ denote the median, min, and max of the inter-packet delay of the overt packet stream. If Alice needs to send a 0 she randomly generates a delay between $min$ and $m$. If she want to send a 1 she randomly generates a delay between $m$ and $max$. First, compute $m$, $min$, and $max$. Next, modify the code in Question 1, to generate the packet stream that contains the secret message.

## Answer

```
#  your R code here
med = median(overtPacketsDelays)
min = min(overtPacketsDelays)
max = max(overtPacketsDelays)
mean = mean(overtPacketsDelays)

message = "this is a secret message"
rawMessage = charToRaw(message)
bits = rawToBits(rawMessage)

bitLength = (nchar(message) * 8)
```

```
sentPacketsTime = numeric(bitLength + 1)
sentPacketsStream = numeric(bitLength)
sentPacketsDelays = numeric(bitLength)
currTime = 0
index = 2
for (i in 1:nchar(message)) {
  for (x in 0:7) {
    if (bits[(i*8) - x] == 00){
      delay = rexp(1, 1/mean)
      while (delay > med || delay < min) {
        delay = rexp(1, 1/mean)
      }
      sentPacketsTime[index] = currTime + delay
      currTime = currTime + delay
      sentPacketsStream[index-1] = 0
      sentPacketsDelays[index-1] = delay
    } else {
      delay = rexp(1, 1/mean)
      while (delay <= med || delay > max) {
        delay = rexp(1, 1/mean)
      }
      sentPacketsTime[index] = currTime + delay
      currTime = currTime + delay
      sentPacketsStream[index-1] = 1
      sentPacketsDelays[index-1] = delay
    }
    index = index + 1
  }
}

print(sentPacketsStream)
```

```
##   [1] 0 1 1 1 0 1 0 0 0 1 1 0 1 0 0 0 0 1 1 0 1 0 0 1 0 1 1 1 0 0 1 1 0 0 1 0 0
##  [38] 0 0 0 0 1 1 0 1 0 0 1 0 1 1 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0
##  [75] 1 0 0 0 0 0 0 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 0 1 1 0 0 0 1 1 0 1 1 1 0 0 1
## [112] 0 0 1 1 0 0 1 0 1 0 1 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 1 0 1 0 1 1 0
## [149] 0 1 0 1 0 1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 0 1 0 1 1 0 0 1 1 1 0
## [186] 1 1 0 0 1 0 1
```

```
print(sentPacketsDelays)
```

```
##   [1] 0.0045695826 0.0915528792 0.1149359445 0.1063418433 0.0141477767
##   [6] 0.0775911009 0.0415433565 0.0224055174 0.0117084705 0.1337640807
##  [11] 0.1087017037 0.0406249034 0.1033520955 0.0622886613 0.0533820489
##  [16] 0.0158296370 0.0084045682 0.1225451858 0.1125703497 0.0252315996
##  [21] 0.0965616464 0.0267727538 0.0217587009 0.1150030347 0.0424217204
##  [26] 0.1680568837 0.1504054281 0.1393127409 0.0242198773 0.0435754515
##  [31] 0.0860641424 0.2228305853 0.0049969130 0.0633342127 0.2455158612
##  [36] 0.0260668261 0.0605478015 0.0528750813 0.0246982181 0.0097673585
##  [41] 0.0393526658 0.2016842573 0.1131541698 0.0513357237 0.1788595996
##  [46] 0.0730065202 0.0657509884 0.1433548140 0.0029502239 0.1054588572
##  [51] 0.1364469183 0.2620452485 0.0312104438 0.0525349372 0.1339095468
```

```
##  [56] 0.1059822739 0.0586424442 0.0567389485 0.3549422380 0.0513509183
##  [61] 0.0397073635 0.0278906028 0.0138225596 0.0645366919 0.0073363469
##  [66] 0.1563622713 0.2003282096 0.0131050966 0.0119056188 0.0370108996
##  [71] 0.0247825620 0.1287038789 0.0480617422 0.0078264249 0.1309650496
##  [76] 0.0059431674 0.0007709205 0.0198564145 0.0016998747 0.0192396756
##  [81] 0.0006591645 0.1737249911 0.0949235298 0.3027268532 0.0534697436
##  [86] 0.0383208617 0.1675151474 0.3260545433 0.0592604429 0.1525485767
##  [91] 0.2579428524 0.0097244919 0.0034272315 0.1119265328 0.0272755318
##  [96] 0.1071625223 0.0120362135 0.1466145872 0.0993179407 0.0680223793
## [101] 0.0229384401 0.0684140000 0.1496849556 0.3637682495 0.0572033836
## [106] 0.1956676092 0.2471248992 0.1427241222 0.0022485109 0.0219683867
## [111] 0.1263536128 0.0738047737 0.0170081152 0.1367759134 0.0946648381
## [116] 0.0625134083 0.0248121395 0.1736951692 0.0739886083 0.0782425944
## [121] 0.0249083460 0.2795420526 0.0969459108 0.0883980790 0.0093837234
## [126] 0.1337059888 0.0728299331 0.0639885174 0.0558067922 0.0118548022
## [131] 0.3458978744 0.0751424796 0.0756497197 0.0524608038 0.0426423424
## [136] 0.0550463953 0.0292850018 0.2819043611 0.1200908670 0.0758053432
## [141] 0.0836366721 0.0784561670 0.0436638092 0.3104033773 0.0083254214
## [146] 0.1781657794 0.1768796919 0.0023261629 0.0019486851 0.1216637176
## [151] 0.0328157838 0.1789848979 0.0147388925 0.1629866161 0.2598359147
## [156] 0.1586442176 0.0316323126 0.0300996334 0.2242065097 0.1111351355
## [161] 0.0238007530 0.1039165341 0.0924208784 0.2684467538 0.0292611995
## [166] 0.0601060665 0.1146291690 0.2895861876 0.0436529610 0.1415654189
## [171] 0.1750561486 0.0670523186 0.0418707630 0.0299413935 0.0125427778
## [176] 0.0927930396 0.0206294710 0.3803147570 0.1284149978 0.0627687691
## [181] 0.0532012633 0.2008652765 0.0781958719 0.1315123161 0.0690536499
## [186] 0.1283177177 0.1633069655 0.0573097221 0.0046636231 0.1340855411
## [191] 0.0106884961 0.1114980721
```

```
print(sentPacketsTime)
```

```
##   [1]  0.000000000  0.004569583  0.096122462  0.211058406  0.317400250
##   [6]  0.331548026  0.409139127  0.450682484  0.473088001  0.484796472
##  [11]  0.618560553  0.727262256  0.767887160  0.871239255  0.933527916
##  [16]  0.986909965  1.002739602  1.011144171  1.133689356  1.246259706
##  [21]  1.271491306  1.368052952  1.394825706  1.416584407  1.531587441
##  [26]  1.574009162  1.742066045  1.892471474  2.031784214  2.056004092
##  [31]  2.099579543  2.185643686  2.408474271  2.413471184  2.476805397
##  [36]  2.722321258  2.748388084  2.808935886  2.861810967  2.886509185
##  [41]  2.896276543  2.935629209  3.137313467  3.250467636  3.301803360
##  [46]  3.480662960  3.553669480  3.619420468  3.762775282  3.765725506
##  [51]  3.871184363  4.007631282  4.269676530  4.300886974  4.353421911
##  [56]  4.487331458  4.593313732  4.651956176  4.708695125  5.063637363
##  [61]  5.114988281  5.154695645  5.182586247  5.196408807  5.260945499
##  [66]  5.268281846  5.424644117  5.624972327  5.638077423  5.649983042
##  [71]  5.686993942  5.711776504  5.840480382  5.888542125  5.896368550
##  [76]  6.027333599  6.033276767  6.034047687  6.053904102  6.055603976
##  [81]  6.074843652  6.075502816  6.249227807  6.344151337  6.646878190
##  [86]  6.700347934  6.738668796  6.906183943  7.232238486  7.291498929
##  [91]  7.444047506  7.701990358  7.711714850  7.715142082  7.827068615
##  [96]  7.854344146  7.961506669  7.973542882  8.120157469  8.219475410
## [101]  8.287497789  8.310436230  8.378850229  8.528535185  8.892303435
## [106]  8.949506818  9.145174427  9.392299327  9.535023449  9.537271960
## [111]  9.559240346  9.685593959  9.759398733  9.776406848  9.913182761
```

```
## [116] 10.007847599 10.070361008 10.095173147 10.268868316 10.342856925
## [121] 10.421099519 10.446007865 10.725549918 10.822495828 10.910893907
## [126] 10.920277631 11.053983620 11.126813553 11.190802070 11.246608862
## [131] 11.258463664 11.604361539 11.679504018 11.755153738 11.807614542
## [136] 11.850256884 11.905303280 11.934588281 12.216492642 12.336583509
## [141] 12.412388853 12.496025525 12.574481692 12.618145501 12.928548878
## [146] 12.936874300 13.115040079 13.291919771 13.294245934 13.296194619
## [151] 13.417858337 13.450674121 13.629659018 13.644397911 13.807384527
## [156] 14.067220442 14.225864659 14.257496972 14.287596605 14.511803115
## [161] 14.622938251 14.646739004 14.750655538 14.843076416 15.111523170
## [166] 15.140784369 15.200890436 15.315519605 15.605105792 15.648758754
## [171] 15.790324172 15.965380321 16.032432640 16.074303403 16.104244796
## [176] 16.116787574 16.209580614 16.230210085 16.610524842 16.738939839
## [181] 16.801708608 16.854909872 17.055775148 17.133971020 17.265483336
## [186] 17.334536986 17.462854704 17.626161669 17.683471391 17.688135015
## [191] 17.822220556 17.832909052 17.944407124
```

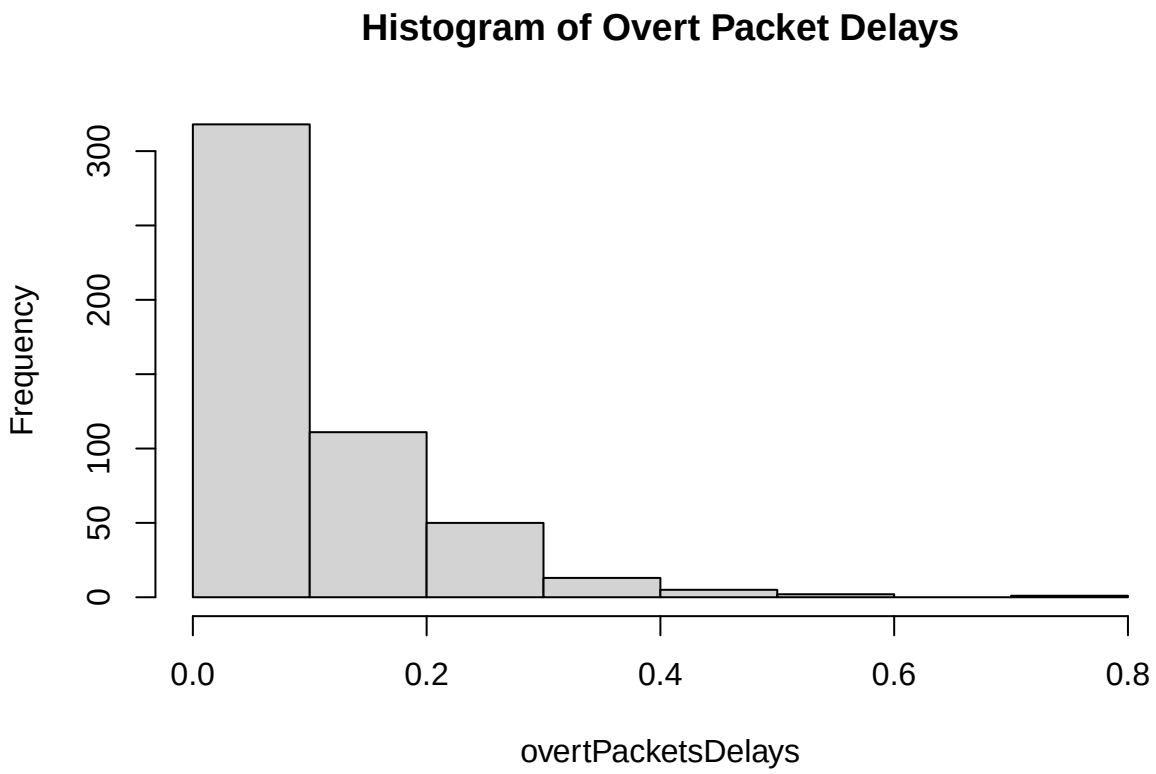
### 3.4

Plot the histogram of the inter-packet delays of the overt packet stream and that of the new covert packet stream. Do you think Eve will be suspicious?

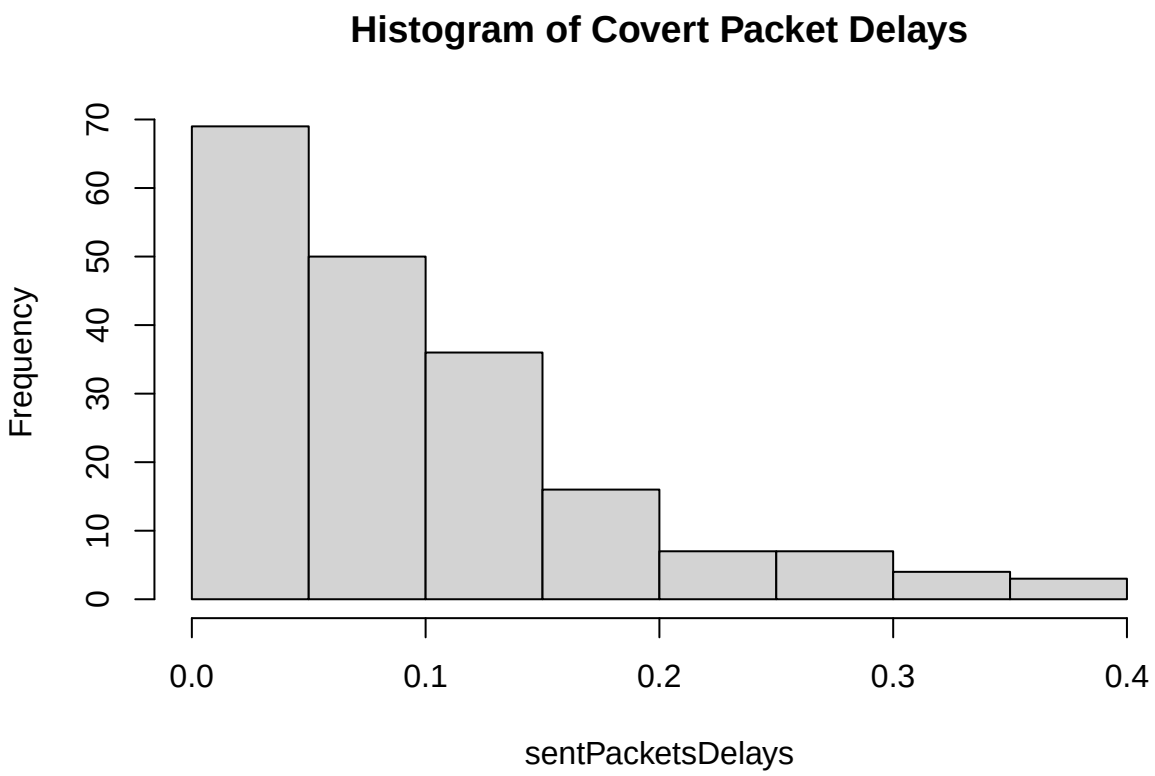
### Answer

```r
#  your R code here
data <- read.csv("Traffic_data_orig.csv", header=TRUE)
overtPacketStream = data[,2]
x = length(overtPacketStream)
overtPacketsDelays = numeric(x-1)
currTime = 0
index = 2
for (i in 2:x-1){
  overtPacketsDelays[i-1] = overtPacketStream[i]-overtPacketStream[i-1]
}

hist(overtPacketsDelays, main = "Histogram of Overt Packet Delays", breaks = 8)
```




```
hist(sentPacketsDelays, main = "Histogram of Covert Packet Delays", breaks = 8)
```




Eve will most likely not be suspicious because the distributions of the overt and covert packet delays are nearly indistinguishable from one another. The overt distribution looks like an exponential distribution so we sampled an exponential distribution with the same rate parameter as the overt distribution and bounded it in order to generate our delays.

### 3.5

Answer the following questions briefly (in 1 or 2 sentences)

1. How can you improve upon the method in Question 3?

   **Answer:** If we had access to more data we could potentially compute an even more precise rate parameter. Potentially another way we could approach this problem is by taking samples from the overt distribution and adding some noise to it. With this approach we wouldn't need to make assumptions about the underlying distribution.

2. We assumed the Alice will buffer up the packets and we mentioned that it was unrealistic. Why?

   **Answer:** In the case where we're using Skype and transmitting data, altering the delays of packets by buffering up a large number of them before sending them out would prevent the call from behaving normally – it may cause problems with the application or rouse suspicion if someone is watching. As a result, it could be problematic.

3. We have assumed that the network does not alter the inter-packet delays. What would be the problem if it did? Can you suggest methods to mitigate the effect of the changes of the inter-packet delay (noise)?

**Answer:** The problem could be that the timing delays could flip a 0 bit to a 1. There are multiple methods that we could use such as using a parity bit, re-sending the message multiple times, or increasing the odds of a correct transmission by moving the max thresholds for emitting a 0 and the minimum threshold for emitting a 1 away from the median to reduce the chance noise flips a bit. To evade suspicion, we would need to be careful with how much we adjust the min and max however, lest our distribution appear bimodal.

# 4 Implementation

## 4.1

For buffer size $B = 20$ we want to find out the probability of overflow and underflow, when the IPD follows the Exponential with $\lambda = 1$ pkts/sec and $i = 2, 6, 10, 14, 18$. Use message size $m = 16, 32$ bits. Tabulate the results. Remember that to determine the probability you need to run multiple (say 500) experiments for each parameter, i.e., for $B = 20, m = 16, i = 2$ run 500 experiments and determine the probability of overflow and underflow. Similarly for other values of $i$ and $m$.

### Answer

```
#  your R simulation code here
ms = c(16, 32)
t = 500
B = 20
i = c(2, 6, 10, 14, 18)
for (m in ms) {
  OverflowProbability = numeric(5)
  UnderflowProbability = numeric(5)
  SuccessProbability = numeric(5)
  index = 0
  for (is in i) {
    index = index + 1
    overflows = numeric(t)
    underflows = numeric(t)
    for (x in 1:t) {
      message = sample(c(0, 1), size = m, replace = TRUE)
      CB = is

      packet_gen_delays = rexp(m, rate = 1)
      packet_send_delays = numeric(m)
      for (bit_i in 1:length(message)) {
        if (message[bit_i] == 0) {
          delay = rexp(1, 1)
          #while (delay > med || delay < min) {
          while (delay > qexp(0.5, 1) || delay < 0) {
            #print(delay)
            delay = rexp(1, 1)
          }
          packet_send_delays[bit_i] = delay
        } else {
          delay = rexp(1, 1)
```

```
      #while (delay <= med || delay > max) {
      while (delay <= qexp(0.5, 1)) {
        #print(delay)
        delay = rexp(1, 1)
      }
      packet_send_delays[bit_i] = delay
    }
  }

  gen_i = is + 1
  t_next_gen = packet_gen_delays[gen_i]
  send_i = 1
  t_next_send = packet_send_delays[send_i]

  outcome = "success"

  while (gen_i < m) {
    if (t_next_gen < t_next_send) {
      # If  a packet will be added to the buffer before the next packet is sent
      t_next_send = t_next_send - t_next_gen

      gen_i = gen_i + 1
      t_next_gen = packet_gen_delays[gen_i]
      CB = CB + 1

      if (CB > B) {
        outcome = "overflow"
        break
      }
    } else if (t_next_send < t_next_gen) {
      # If we send the next packet before we generate the next packet
      t_next_gen = t_next_gen - t_next_send

      send_i = send_i + 1
      t_next_send = packet_send_delays[send_i]
      CB = CB - 1

      if (CB < 0) {
        outcome = "underflow"
        break
      }
    } else {
      # If we send a packet as we generate it
      gen_i = gen_i + 1
      t_next_gen = packet_gen_delays[gen_i]
      send_i = send_i + 1
      t_next_send = packet_send_delays[send_i]
    }
  }

  if (outcome == "overflow") {
    overflows[x] = 1
    underflows[x] = 0
```

```
      } else if (outcome == "underflow") {
        overflows[x] = 0
        underflows[x] = 1
      } else {
        overflows[x] = 0
        underflows[x] = 0
      }
    }



    z = mean(underflows) + mean(overflows)

    OverflowProbability[index] = mean(overflows)
    UnderflowProbability[index] = mean(underflows)
    SuccessProbability[index] = 1 - z
  }

  print(paste("Exponential Distribution"))
  print(paste("Message Size: ", m))
  df = data.frame(i, OverflowProbability, UnderflowProbability, SuccessProbability)
  print(df)
  table(df)
}
```

```
## [1] "Exponential Distribution"
## [1] "Message Size:  16"
##    i OverflowProbability UnderflowProbability SuccessProbability
## 1  2                    0                0.556              0.444
## 2  6                    0                0.146              0.854
## 3 10                    0                0.010              0.990
## 4 14                    0                0.000              1.000
## 5 18                    0                0.000              1.000
## [1] "Exponential Distribution"
## [1] "Message Size:  32"
##    i OverflowProbability UnderflowProbability SuccessProbability
## 1  2                0.002                0.670              0.328
## 2  6                0.014                0.352              0.634
## 3 10                0.048                0.106              0.846
## 4 14                0.198                0.034              0.768
## 5 18                0.540                0.002              0.458
```

```
# tabulate results and compute the probabilities
```

## 4.2

For buffer size $B = 20$ we want to find out the probability of overflow and underflow, when the IPD follows the Uniform distribution in the range (0,1) and $i = 2, 6, 10, 14, 18$. Use message size $m = 16, 32$ bits. Tabulate the results. Remember that to determine the probability you need to run multiple (say 500) experiments for each parameter, i.e., for $B = 20, m = 16, i = 2$ run 500 experiments and determine the probability of overflow and underflow. Similarly for other values of $i$ and $m$.

### Answer

```
#  your R simulation code here

ms = c(16, 32)
t = 500
B = 20
i = c(2, 6, 10, 14, 18)
for (m in ms) {
  OverflowProbability = numeric(5)
  UnderflowProbability = numeric(5)
  SuccessProbability = numeric(5)
  index = 0
  for (is in i) {
    index = index + 1
    overflows = numeric(t)
    underflows = numeric(t)
    for (x in 1:t) {
      message = sample(c(0, 1), size = m, replace = TRUE)
      CB = is

      packet_gen_delays = runif(m, 0, 1)
      packet_send_delays = numeric(m)
      for (bit_i in 1:length(message)) {
        if (message[bit_i] == 0) {
          delay = runif(1, 0, 0.5)
          packet_send_delays[bit_i] = delay
        } else {
          delay = runif(1, 0.5, 1)
          packet_send_delays[bit_i] = delay
        }
      }

      gen_i = is + 1
      t_next_gen = packet_gen_delays[gen_i]
      send_i = 1
      t_next_send = packet_send_delays[send_i]

      outcome = "success"

      while (gen_i < m) {
        if (t_next_gen < t_next_send) {
          # If  a packet will be added to the buffer before the next packet is sent
          t_next_send = t_next_send - t_next_gen

          gen_i = gen_i + 1
          t_next_gen = packet_gen_delays[gen_i]
          CB = CB + 1

          if (CB > B) {
            outcome = "overflow"
            break
          }
```

```
        } else if (t_next_send < t_next_gen) {
          # If we send the next packet before we generate the next packet
          t_next_gen = t_next_gen - t_next_send

          send_i = send_i + 1
          t_next_send = packet_send_delays[send_i]
          CB = CB - 1

          if (CB < 0) {
            outcome = "underflow"
            break
          }
        } else {
          # If we send a packet as we generate it
          gen_i = gen_i + 1
          t_next_gen = packet_gen_delays[gen_i]
          send_i = send_i + 1
          t_next_send = packet_send_delays[send_i]
        }
      }

      if (outcome == "overflow") {
        overflows[x] = 1
        underflows[x] = 0
      } else if (outcome == "underflow") {
        overflows[x] = 0
        underflows[x] = 1
      } else {
        overflows[x] = 0
        underflows[x] = 0
      }
    }



    z = mean(underflows) + mean(overflows)

    OverflowProbability[index] = mean(overflows)
    UnderflowProbability[index] = mean(underflows)
    SuccessProbability[index] = 1 - z
  }
  print(paste("Uniform Distribution"))
  print(paste("Message Size: ", m))
  df = data.frame(i, OverflowProbability, UnderflowProbability, SuccessProbability)
  print(df)
  table(df)
}
```

```
## [1] "Uniform Distribution"
## [1] "Message Size:  16"
##    i OverflowProbability UnderflowProbability SuccessProbability
## 1  2                   0                0.438              0.562
## 2  6                   0                0.030              0.970
```

```
## 3 10                  0               0.000              1.000
## 4 14                  0               0.000              1.000
## 5 18                  0               0.000              1.000
## [1] "Uniform Distribution"
## [1] "Message Size:  32"
##    i OverflowProbability UnderflowProbability SuccessProbability
## 1  2               0.000                0.600              0.400
## 2  6               0.000                0.128              0.872
## 3 10               0.002                0.018              0.980
## 4 14               0.044                0.000              0.956
## 5 18               0.338                0.000              0.662
```

```
# tabulate results and compute the probabilities
```

### 4.3

Propose methods to deal with buffer overflow and underflow.

## Answer

Sending shorter messages, using larger buffers, ensuring that the buffer is half filled with packets when transmission begins, and ensuring that packet delays are generated using the appropriate distribution all ensure that buffer overflow and underflow has a low probability of occurring.

In the event that a buffer overflow or underflow occurs, we believe the best way to handle it would be to send a message to the receiver that a buffer overflow or underflow has occurred. This could be done by spiking the delay of a packet to be longer than usual. If using a uniform distribution, you could set the packet delay to be longer than the maximum normal delay, indicating that the previous packet needs to be resent. If overflow occurs, a similar spiked message could be sent, and then the receiver could know to ignore the next 5 packet delays, as the sender clears out some of the packets from the overflowed buffer at a slightly higher than normal rate. To use this method with an exponential distribution, the sender and receiver would need to place a cap on the length of delays generated by the sender, with the understanding that any delay longer than the agreed upon max indicates a problem.

Both of these methods do contain a slight risk of anomalous behavior being detected by a savvy Eve, however. A better method perhaps would be to transmit the same message multiple times, in the hopes that at least one such message would arrive intact.