ECS 132 Fall 2020, Project

December 1, 2020

Your name: Han Nguyen #917278789

Partner's name: Zibo Zhang #914913319

Any other collaborator(s): N/A

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.

Answer

```r
## Read data from .csv file
data = read.csv("Traffic_data_orig.csv", header=TRUE)

## Get message information
secret_message = "this is a secret message"
message_length = nchar(secret_message)
message_bits = as.integer(rawToBits((charToRaw(secret_message))))

## Get lists for delay times, inter packet delays (in total) and covert packet stream
delay_times = numeric(message_length * 8)
inter_packet_delays = numeric(length(data$Time) - 1)
```

```r
covert_packet_stream = numeric(length(data$Time) - 1)

## Get delay times by message bits
curr_index = 1

for (i in 0:(message_length-1))
{
  for (j in 1:8)
  {
    if (message_bits[i*8+j] == 0)
    {
      delay_times[curr_index] = 0.25
    }
    else
    {
      delay_times[curr_index] = 0.75
    }

    curr_index = curr_index + 1
    j = j - 1
  }
}
# print(delay_times)

## Get inter packet delay times
for (i in (1:(length(data$Time) - 1)))
{
  inter_packet_delays[i] = data$Time[i+1] - data$Time[i]
}
# print(inter_packet_delays)

## Get covert packet stream
covert_packet_stream = inter_packet_delays

for (i in 1:(message_length*8))
{
  covert_packet_stream[i] = delay_times[i]
}

## DISPLAY THE COVERT PACKET STREAM
print(covert_packet_stream)
```

```
##   [1] 0.25000 0.25000 0.75000 0.25000 0.75000 0.75000 0.75000 0.25000 0.25000
##  [10] 0.25000 0.25000 0.75000 0.25000 0.75000 0.75000 0.25000 0.75000 0.25000
##  [19] 0.25000 0.75000 0.25000 0.75000 0.75000 0.25000 0.75000 0.75000 0.25000
##  [28] 0.25000 0.75000 0.75000 0.75000 0.25000 0.25000 0.25000 0.25000 0.25000
##  [37] 0.25000 0.75000 0.25000 0.25000 0.75000 0.25000 0.25000 0.75000 0.25000
##  [46] 0.75000 0.75000 0.25000 0.75000 0.75000 0.25000 0.25000 0.75000 0.75000
##  [55] 0.75000 0.25000 0.25000 0.25000 0.25000 0.25000 0.25000 0.75000 0.25000
##  [64] 0.25000 0.75000 0.25000 0.25000 0.25000 0.25000 0.75000 0.75000 0.25000
##  [73] 0.25000 0.25000 0.25000 0.25000 0.25000 0.75000 0.25000 0.25000 0.75000
##  [82] 0.75000 0.25000 0.25000 0.75000 0.75000 0.75000 0.25000 0.75000 0.25000
##  [91] 0.75000 0.25000 0.25000 0.75000 0.75000 0.25000 0.75000 0.75000 0.25000
## [100] 0.25000 0.25000 0.75000 0.75000 0.25000 0.25000 0.75000 0.25000 0.25000
## [109] 0.75000 0.75000 0.75000 0.25000 0.75000 0.25000 0.75000 0.25000 0.25000
## [118] 0.75000 0.75000 0.25000 0.25000 0.25000 0.75000 0.25000 0.75000 0.75000
## [127] 0.75000 0.25000 0.25000 0.25000 0.25000 0.25000 0.25000 0.75000 0.25000
## [136] 0.25000 0.75000 0.25000 0.75000 0.75000 0.25000 0.75000 0.75000 0.25000
## [145] 0.75000 0.25000 0.75000 0.25000 0.25000 0.75000 0.75000 0.25000 0.75000
## [154] 0.75000 0.25000 0.25000 0.75000 0.75000 0.75000 0.25000 0.75000 0.75000
## [163] 0.25000 0.25000 0.75000 0.75000 0.75000 0.25000 0.75000 0.25000 0.25000
## [172] 0.25000 0.25000 0.75000 0.75000 0.25000 0.75000 0.75000 0.75000 0.25000
## [181] 0.25000 0.75000 0.75000 0.25000 0.75000 0.25000 0.75000 0.25000 0.25000
## [190] 0.75000 0.75000 0.25000 0.18733 0.10734 0.07915 0.20875 0.02338 0.05134
## [199] 0.02258 0.08589 0.01190 0.01008 0.05376 0.06232 0.02347 0.13929 0.02377
## [208] 0.10184 0.34431 0.20390 0.03399 0.09466 0.09033 0.15325 0.18373 0.05694
## [217] 0.07798 0.11209 0.33524 0.08969 0.10415 0.23894 0.45649 0.07024 0.01816
## [226] 0.23676 0.01751 0.09907 0.01797 0.09051 0.05191 0.18864 0.18671 0.29542
## [235] 0.05702 0.20033 0.06585 0.05368 0.00998 0.15592 0.25081 0.05664 0.01566
## [244] 0.04058 0.02204 0.05062 0.17345 0.09403 0.00332 0.27011 0.11581 0.18259
## [253] 0.24585 0.02950 0.08774 0.02559 0.05476 0.02030 0.09343 0.03410 0.06354
## [262] 0.15096 0.13921 0.18122 0.34107 0.19335 0.21425 0.22392 0.04745 0.10721
## [271] 0.42241 0.12020 0.11808 0.06222 0.25003 0.04774 0.09118 0.02157 0.01481
## [280] 0.09310 0.06577 0.01472 0.33141 0.10038 0.11018 0.01916 0.00316 0.02074
## [289] 0.00864 0.02445 0.00495 0.10464 0.54102 0.04747 0.21455 0.10024 0.02851
## [298] 0.06676 0.20493 0.06035 0.23721 0.00851 0.03790 0.00305 0.08028 0.03177
## [307] 0.03019 0.02668 0.01796 0.28584 0.02134 0.09817 0.02782 0.01659 0.09798
## [316] 0.09067 0.11290 0.09715 0.27404 0.43303 0.02315 0.05561 0.06000 0.05859
## [325] 0.11338 0.05683 0.08452 0.09727 0.28833 0.00383 0.10281 0.05003 0.02247
## [334] 0.04711 0.02793 0.17033 0.18120 0.03078 0.10812 0.07985 0.09859 0.17335
## [343] 0.04323 0.06017 0.10110 0.15213 0.03614 0.13735 0.13197 0.03931 0.01792
## [352] 0.00714 0.01441 0.06564 0.01648 0.22812 0.26932 0.11449 0.01190 0.10282
## [361] 0.00883 0.06992 0.31102 0.01931 0.07322 0.17210 0.11521 0.37747 0.01603
## [370] 0.03494 0.08583 0.18543 0.11302 0.08568 0.14288 0.13033 0.07702 0.05826
## [379] 0.01174 0.37553 0.06113 0.21108 0.05302 0.11302 0.00134 0.00570 0.04084
## [388] 0.73199 0.08744 0.09660 0.03690 0.02583 0.15000 0.10889 0.00586 0.29323
## [397] 0.00187 0.00596 0.13894 0.04815 0.06398 0.12894 0.16018 0.07674 0.03978
## [406] 0.15976 0.05329 0.01020 0.10582 0.13168 0.12974 0.14872 0.16253 0.13452
## [415] 0.06955 0.02290 0.00403 0.00432 0.04731 0.00467 0.00728 0.04801 0.02859
## [424] 0.19388 0.09521 0.00001 0.08719 0.24778 0.02204 0.04427 0.03977 0.31302
## [433] 0.03622 0.01412 0.02994 0.03579 0.00661 0.05183 0.06476 0.16709 0.00662
## [442] 0.16947 0.03694 0.01992 0.17916 0.07338 0.01332 0.12988 0.03067 0.05224
## [451] 0.00076 0.08836 0.16457 0.00685 0.01641 0.14291 0.11131 0.03446 0.01599
## [460] 0.02937 0.13478 0.12469 0.09888 0.01620 0.07928 0.03448 0.27688 0.26850
## [469] 0.26066 0.00406 0.01869 0.00892 0.09416 0.03929 0.03962 0.20953 0.16678
## [478] 0.01824 0.05712 0.14752 0.02016 0.06816 0.07195 0.16133 0.07946 0.01101
## [487] 0.05371 0.02669 0.06633 0.01387 0.12383 0.06757 0.00742 0.17985 0.16448
## [496] 0.08271 0.01440 0.07497 0.07727 0.01254
```

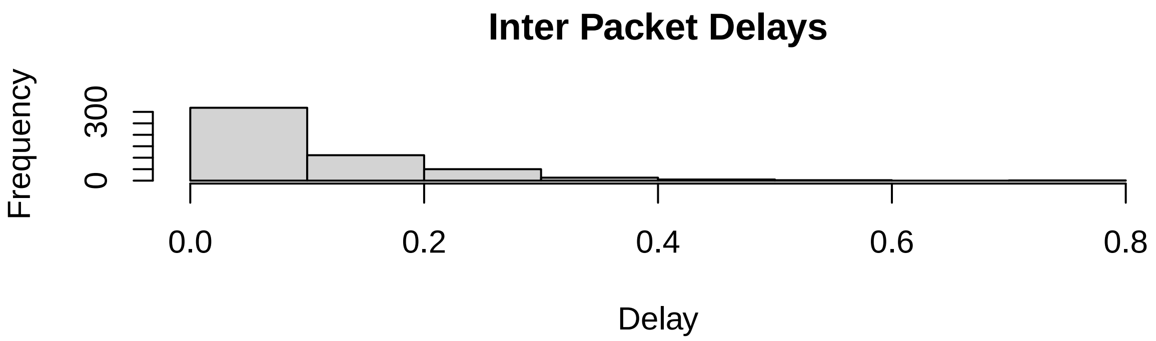
3.2

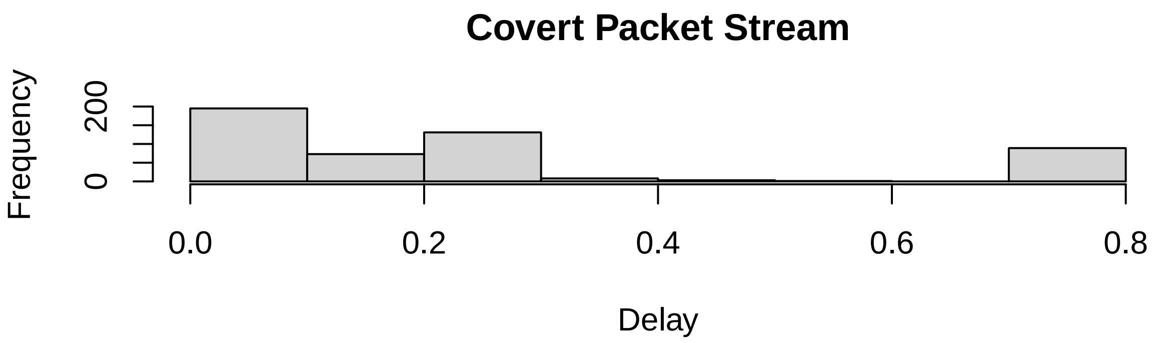
Plot the histogram of the inter-packet delays of the overt packet stream. Plot the histogram of the covert packet stream. Will Eve be suspicious?

Answer

```r
## Read data from .csv file
data = read.csv("Traffic_data_orig.csv", header=TRUE)

## Get message information
secret_message = "this is a secret message"
message_length = nchar(secret_message)
message_bits = as.integer(rawToBits((charToRaw(secret_message))))

## Get lists for delay times, inter packet delays (in total) and covert packet stream
delay_times = numeric(message_length * 8)
inter_packet_delays = numeric(length(data$Time) - 1)
covert_packet_stream = numeric(length(data$Time) - 1)

## Get inter packet delay times
for (i in (1:(length(data$Time) - 1)))
{
  inter_packet_delays[i] = data$Time[i+1] - data$Time[i]
}

## Get delay times by message bits
curr_index = 1

for (i in 0:(message_length-1))
{
  for (j in 1:8)
  {
    if (message_bits[i*8+j] == 0)
    {
      delay_times[curr_index] = 0.25
    }
    else
    {
      delay_times[curr_index] = 0.75
    }

    curr_index = curr_index + 1
    j = j - 1
  }
}
# print(delay_times)

## Get covert packet stream
covert_packet_stream = inter_packet_delays

for (i in 1:(message_length*8))
{
  covert_packet_stream[i] = delay_times[i]
}

## PLOT HISTORGRAMS OF INTER PACKET DELAYS & COVERT PACKET STREAM
par(mfrow=c(2,1))
```

```r
hist(inter_packet_delays, main="Inter Packet Delays", xlab="Delay")
hist(covert_packet_stream, main="Covert Packet Stream", xlab="Delay")
```





From the two histograms, we think that Eve will be suspicious since the trend of the distribution between two histograms is NOT similar with each other. Also, compared to the original overt packet stream, the covert packet stream has produced higher packet number within a unit of delay time.

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

```r
## Read data from .csv file
data = read.csv("Traffic_data_orig.csv", header=TRUE)

## Get message information
secret_message = "this is a secret message"
message_length = nchar(secret_message)
message_bits = as.integer(rawToBits((charToRaw(secret_message))))

## Get lists for delay times, inter packet delays (in total) and covert packet stream
delay_times = numeric(message_length * 8)
```

```r
inter_packet_delays = numeric(length(data$Time) - 1)
covert_packet_stream = numeric(length(data$Time) - 1)

## Get inter packet delay times
for (i in (1:(length(data$Time) - 1)))
{
  inter_packet_delays[i] = data$Time[i+1] - data$Time[i]
}
# print(inter_packet_delays)

## Get median, min, max of the inter packet delays
min = min(inter_packet_delays)
m = median(inter_packet_delays)
max = max(inter_packet_delays)

## Get delay times by message bits
curr_index = 1

for (i in 0:(message_length-1))
{
  for (j in 1:8)
  {
    if (message_bits[i*8+j] == 0)
    {
      delay_times[curr_index] = runif(1, min, m)
    }
    else
    {
      delay_times[curr_index] = runif(1, m, max)
    }

    curr_index = curr_index + 1
    j = j - 1
  }
}
# print(delay_times)

## Get covert packet stream
updated_covert_packet_stream = inter_packet_delays

for (i in 1:(message_length*8))
{
  updated_covert_packet_stream[i] = delay_times[i]
}

## DISPLAY THE COVERT PACKET STREAM
print(updated_covert_packet_stream)
```

```
##    [1] 0.0286169639 0.0062214594 0.3775618290 0.0378096815 0.2212087047
##    [6] 0.0954506677 0.5109358635 0.0684366045 0.0692961359 0.0745214978
##   [11] 0.0750522072 0.5471833688 0.0413756575 0.5960479809 0.1347216150
##   [16] 0.0052124869 0.2752668098 0.0348546103 0.0562271045 0.3187960671
##   [21] 0.0665496390 0.5600095669 0.3519347498 0.0159623160 0.0954617599
##   [26] 0.2898727830 0.0485881259 0.0227535115 0.3510320209 0.4851989304
##  [31] 0.5734945962 0.0396147256 0.0125387631 0.0284369744 0.0454578950
##  [36] 0.0144313736 0.0726272291 0.6480992408 0.0100853436 0.0374978703
##  [41] 0.4881248465 0.0065314096 0.0141626652 0.2084741145 0.0155108645
##  [46] 0.3201883308 0.6725485916 0.0236342025 0.1950640155 0.1101424837
##  [51] 0.0516977098 0.0271041753 0.4547606896 0.4159567519 0.5242272059
##  [56] 0.0309265421 0.0359879035 0.0385958863 0.0249499420 0.0442446026
##  [61] 0.0003524907 0.1653450301 0.0660565438 0.0585941885 0.3386804839
##  [66] 0.0234444512 0.0610110837 0.0768554517 0.0253545854 0.5026863997
##  [71] 0.2399357568 0.0110909048 0.0266609840 0.0048752116 0.0175211773
##  [76] 0.0064721360 0.0292336691 0.5487340016 0.0058078749 0.0140550200
##  [81] 0.1017213220 0.4083961674 0.0342068936 0.0427755867 0.4323208766
##  [86] 0.1370155818 0.6608678853 0.0205979159 0.4544818106 0.0325137159
##  [91] 0.1440584897 0.0486980872 0.0124649435 0.4712082931 0.0907634064
##  [96] 0.0076477580 0.5729348755 0.5047597137 0.0529598202 0.0160383910
## [101] 0.0220721935 0.1707988587 0.6874857645 0.0342695544 0.0375095771
## [106] 0.2747496620 0.0338675573 0.0266613103 0.5567689145 0.6160101254
## [111] 0.4522165551 0.0100229866 0.5486202315 0.0195773437 0.1341823370
## [116] 0.0038176160 0.0677502012 0.5718024394 0.6099792697 0.0621036382
## [121] 0.0553721330 0.0681049526 0.0921285472 0.0036475868 0.4250818235
## [126] 0.5295690986 0.6477475495 0.0251061447 0.0752574840 0.0045987664
## [131] 0.0547628284 0.0591155679 0.0410754216 0.2601033733 0.0019115013
## [136] 0.0414222920 0.1414275389 0.0671718773 0.1032800931 0.5560921324
## [141] 0.0282540652 0.6759607509 0.1389801313 0.0401107261 0.4394494655
## [146] 0.0561596007 0.5435834252 0.0367477252 0.0730614552 0.1604278859
## [151] 0.6314047929 0.0031750284 0.1368863166 0.1533258794 0.0717872044
## [156] 0.0741452660 0.6049926348 0.6547875446 0.3021018423 0.0573431673
## [161] 0.6305736063 0.5198367657 0.0392653907 0.0154111785 0.3416090671
## [166] 0.3220776278 0.3674427713 0.0018408977 0.7139714257 0.0470011975
## [171] 0.0406910969 0.0077184265 0.0255008440 0.3192824112 0.1396234262
## [176] 0.0084527507 0.1582643435 0.6810242003 0.2601047181 0.0078428532
## [181] 0.0756895612 0.5388939298 0.7079532123 0.0252669301 0.6648358577
## [186] 0.0579510302 0.2626119235 0.0498754253 0.0582487486 0.4241512088
## [191] 0.2109885781 0.0122523267 0.1873300000 0.1073400000 0.0791500000
## [196] 0.2087500000 0.0233800000 0.0513400000 0.0225800000 0.0858900000
## [201] 0.0119000000 0.0100800000 0.0537600000 0.0623200000 0.0234700000
## [206] 0.1392900000 0.0237700000 0.1018400000 0.3443100000 0.2039000000
## [211] 0.0339900000 0.0946600000 0.0903300000 0.1532500000 0.1837300000
## [216] 0.0569400000 0.0779800000 0.1120900000 0.3352400000 0.0896900000
## [221] 0.1041500000 0.2389400000 0.4564900000 0.0702400000 0.0181600000
## [226] 0.2367600000 0.0175100000 0.0990700000 0.0179700000 0.0905100000
## [231] 0.0519100000 0.1886400000 0.1867100000 0.2954200000 0.0570200000
## [236] 0.2003300000 0.0658500000 0.0536800000 0.0099800000 0.1559200000
## [241] 0.2508100000 0.0566400000 0.0156600000 0.0405800000 0.0220400000
## [246] 0.0506200000 0.1734500000 0.0940300000 0.0033200000 0.2701100000
## [251] 0.1158100000 0.1825900000 0.2458500000 0.0295000000 0.0877400000
## [256] 0.0255900000 0.0547600000 0.0203000000 0.0934300000 0.0341000000
## [261] 0.0635400000 0.1509600000 0.1392100000 0.1812200000 0.3410700000
## [266] 0.1933500000 0.2142500000 0.2239200000 0.0474500000 0.1072100000
## [271] 0.4224100000 0.1202000000 0.1180800000 0.0622200000 0.2500300000
## [276] 0.0477400000 0.0911800000 0.0215700000 0.0148100000 0.0931000000
## [281] 0.0657700000 0.0147200000 0.3314100000 0.1003800000 0.1101800000
## [286] 0.0191600000 0.0031600000 0.0207400000 0.0086400000 0.0244500000
## [291] 0.0049500000 0.1046400000 0.5410200000 0.0474700000 0.2145500000
## [296] 0.1002400000 0.0285100000 0.0667600000 0.2049300000 0.0603500000
## [301] 0.2372100000 0.0085100000 0.0379000000 0.0030500000 0.0802800000
## [306] 0.0317700000 0.0301900000 0.0266800000 0.0179600000 0.2858400000
## [311] 0.0213400000 0.0981700000 0.0278200000 0.0165900000 0.0979800000
## [316] 0.0906700000 0.1129000000 0.0971500000 0.2740400000 0.4330300000
## [321] 0.0231500000 0.0556100000 0.0600000000 0.0585900000 0.1133800000
## [326] 0.0568300000 0.0845200000 0.0972700000 0.2883300000 0.0038300000
## [331] 0.1028100000 0.0500300000 0.0224700000 0.0471100000 0.0279300000
## [336] 0.1703300000 0.1812000000 0.0307800000 0.1081200000 0.0798500000
## [341] 0.0985900000 0.1733500000 0.0432300000 0.0601700000 0.1011000000
## [346] 0.1521300000 0.0361400000 0.1373500000 0.1319700000 0.0393100000
## [351] 0.0179200000 0.0071400000 0.0144100000 0.0656400000 0.0164800000
## [356] 0.2281200000 0.2693200000 0.1144900000 0.0119000000 0.1028200000
## [361] 0.0088300000 0.0699200000 0.3110200000 0.0193100000 0.0732200000
## [366] 0.1721000000 0.1152100000 0.3774700000 0.0160300000 0.0349400000
## [371] 0.0858300000 0.1854300000 0.1130200000 0.0856800000 0.1428800000
## [376] 0.1303300000 0.0770200000 0.0582600000 0.0117400000 0.3755300000
## [381] 0.0611300000 0.2110800000 0.0530200000 0.1130200000 0.0013400000
## [386] 0.0057000000 0.0408400000 0.7319900000 0.0874400000 0.0966000000
## [391] 0.0369000000 0.0258300000 0.1500000000 0.1088900000 0.0058600000
## [396] 0.2932300000 0.0018700000 0.0059600000 0.1389400000 0.0481500000
## [401] 0.0639800000 0.1289400000 0.1601800000 0.0767400000 0.0397800000
## [406] 0.1597600000 0.0532900000 0.0102000000 0.1058200000 0.1316800000
## [411] 0.1297400000 0.1487200000 0.1625300000 0.1345200000 0.0695500000
## [416] 0.0229000000 0.0040300000 0.0043200000 0.0473100000 0.0046700000
## [421] 0.0072800000 0.0480100000 0.0285900000 0.1938800000 0.0952100000
## [426] 0.0000100000 0.0871900000 0.2477800000 0.0220400000 0.0442700000
## [431] 0.0397700000 0.3130200000 0.0362200000 0.0141200000 0.0299400000
## [436] 0.0357900000 0.0066100000 0.0518300000 0.0647600000 0.1670900000
## [441] 0.0066200000 0.1694700000 0.0369400000 0.0199200000 0.1791600000
## [446] 0.0733800000 0.0133200000 0.1298800000 0.0306700000 0.0522400000
## [451] 0.0007600000 0.0883600000 0.1645700000 0.0068500000 0.0164100000
## [456] 0.1429100000 0.1113100000 0.0344600000 0.0159900000 0.0293700000
## [461] 0.1347800000 0.1246900000 0.0988800000 0.0162000000 0.0792800000
## [466] 0.0344800000 0.2768800000 0.2685000000 0.2606600000 0.0040600000
## [471] 0.0186900000 0.0089200000 0.0941600000 0.0392900000 0.0396200000
## [476] 0.2095300000 0.1667800000 0.0182400000 0.0571200000 0.1475200000
## [481] 0.0201600000 0.0681600000 0.0719500000 0.1613300000 0.0794600000
## [486] 0.0110100000 0.0537100000 0.0266900000 0.0663300000 0.0138700000
## [491] 0.1238300000 0.0675700000 0.0074200000 0.1798500000 0.1644800000
## [496] 0.0827100000 0.0144000000 0.0749700000 0.0772700000 0.0125400000
```

3.4

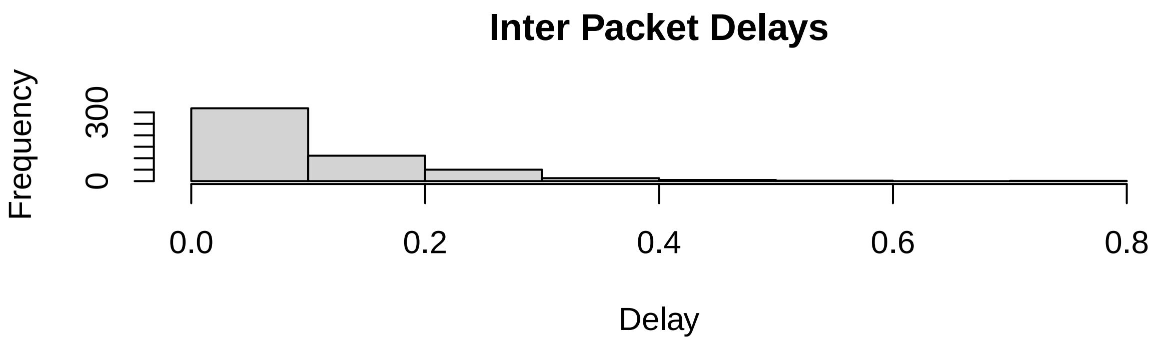
Plot the histogram of the inter-packet delays of the overt packet stream and that of the new covert packet stream. Do you think Eve will be suspicious?

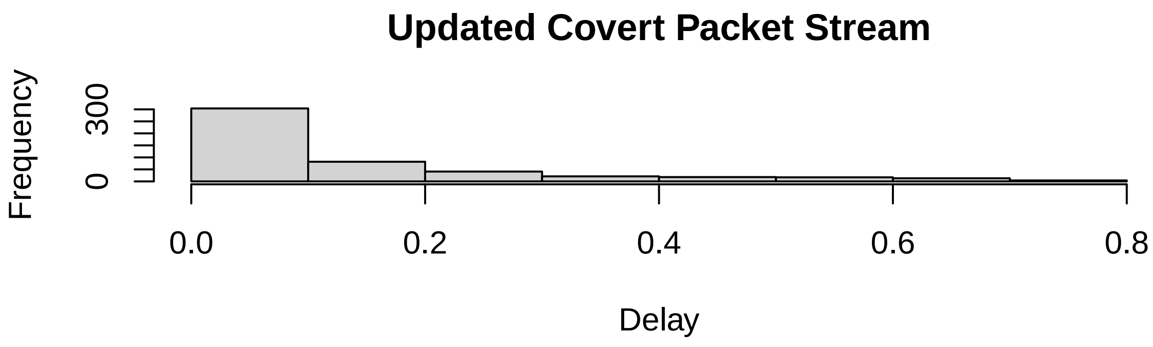
Answer

```r
## Read data from .csv file
data = read.csv("Traffic_data_orig.csv", header=TRUE)

## Get message information
```

```r
secret_message = "this is a secret message"
message_length = nchar(secret_message)
message_bits = as.integer(rawToBits((charToRaw(secret_message))))

## Get lists for delay times, inter packet delays (in total) and covert packet stream
delay_times = numeric(message_length * 8)
inter_packet_delays = numeric(length(data$Time) - 1)
covert_packet_stream = numeric(length(data$Time) - 1)

## Get inter packet delay times
for (i in (1:(length(data$Time) - 1)))
{
  inter_packet_delays[i] = data$Time[i+1] - data$Time[i]
}
# print(inter_packet_delays)

## Get median, min, max of the inter packet delays
min = min(inter_packet_delays)
m = median(inter_packet_delays)
max = max(inter_packet_delays)

## Get delay times by message bits
curr_index = 1

for (i in 0:(message_length-1))
{
  for (j in 1:8)
  {
    if (message_bits[i*8+j] == 0)
    {
      delay_times[curr_index] = runif(1, min, m)
    }
    else
    {
      delay_times[curr_index] = runif(1, m, max)
    }

    curr_index = curr_index + 1
    j = j - 1
  }
}
# print(delay_times)

## Get covert packet stream
updated_covert_packet_stream = inter_packet_delays

for (i in 1:(message_length*8))
{
  updated_covert_packet_stream[i] = delay_times[i]
}
# print(updated_covert_packet_stream)

par(mfrow=c(2,1))
```

```r
hist(inter_packet_delays, main="Inter Packet Delays", xlab="Delay")
hist(updated_covert_packet_stream, main="Updated Covert Packet Stream", xlab="Delay")
```





At this time, we believe that Eve might not be suspicious anymore, since the trend of the distribution of the two histograms is VERY similar with each other. The number of packets by each unit of delay time is also similar as well.

3.5

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

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

   Answer: In my opinion, by using `runif()`, we should reach the most prominent method to generate a new packet stream via finding the median, min and max of the inter-packet delays. At first, I would like to use `sample()`, but `sample()` is a method of generating the data inclusively, and it does not collect data from non-discrete (continuous) data sets.

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

   Answer: By buffering up the packets, Alice needs to ensure that the packets would be transmitted at the same rate that they arrive. However, since the packet stream is not following a constant rate (as shown in printing the data set or histograms), it will be unrealistic for Alice to do so (Source: Stanford University, link below).

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: If the network altered the inter-packet delays, the output data of covert packet stream should be totally different from the original dataset, making the detection of a third person like Eve will be more difficult since the differences made by such constant alterations would always exist. We can address the problem by designing the network upon the average inter-packet delay everytime when the delay times get altered to ensure the adaptation of the data every time they change.

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

```r
#  your R simulation code here
i = c(2, 6, 10, 14, 18)
m = c(16, 32)
B = 20
sample_num = 500
lambda = 1

## Get random bits size of m for the secret message
get_random_secret_message_bits = function(m)
{
  message_bits = numeric(m)

  for (i in 1:m)
  {
    message_bits[i] = sample(c(0,1), 1)
  }

  return(message_bits)
}

## Get time for each packet source generation using IPD distribution
get_packet_times = function(idp, worst_case)
{
  packet_times = numeric(worst_case)

  for (i in 1:worst_case)
  {
    if (i == 1)
    {
      packet_times[i] = 0
    }

    else
    {
      packet_times[i] = packet_times[i-1] + idp[i-1]
    }
  }

  return(packet_times)
}

## Get overflow and underflow of the packet stream under IPD distribution
get_overflow_underflow = function(m, i)
{
  curr_buffer = i
  message_bits = get_random_secret_message_bits(m)

  worst_case = 49

  idp_expo = rexp(worst_case, lambda)
  packet_times = get_packet_times(idp_expo, worst_case+1)
  next_sent_time = packet_times[i]

  sender_delay = rexp(worst_case, lambda)
  med = median(sender_delay)
  min = min(sender_delay)
  max = max(sender_delay)
  message_bit_delay = rexp(worst_case, lambda)

  underflow = 0
  overflow = 0

  next_buffer = i+1

  for (j in 1:m)
  {
    if (message_bits[j] == 0)
    {
      delay = runif(1, min, med)
      message_bit_delay[j] = delay
    }

    else if (message_bits[j] == 1)
    {
      delay = runif(1, med, max)
      message_bit_delay[j] = delay
    }

    next_sent_time = next_sent_time + delay

    # Update buffer
    if (next_sent_time <= packet_times[next_buffer])
    {
      curr_buffer = curr_buffer - 1

      if (curr_buffer < 1)
      {
        underflow = 1
        break
      }
    }

    else
    {
      curr_buffer = curr_buffer + 1

      if (curr_buffer > B)
      {
        overflow = 1
        break
      }
    }

    next_buffer = next_buffer + 1
  }

  return(c(overflow, underflow))
}

## Get results of overflow-underflow probabilities
get_prob_results = function(m, i)
{
  overflow_probs = numeric(sample_num)
  underflow_probs = numeric(sample_num)

  sample_count = 1

  for (j in 1:sample_num)
  {
    overflow = 0
    underflow = 0
    over_under = get_overflow_underflow(m, i)

    # Update overflow and underflow
    overflow = overflow + over_under[1]
    underflow = underflow + over_under[2]

    overflow_probs[j] = overflow/sample_count
    underflow_probs[j] = underflow/sample_count

    sample_count = sample_count + 1
  }

  return(c(mean(overflow_probs), mean(underflow_probs)))
}

# tabulate results and compute the probabilities

## GET PROBABILITIES OF OVERFLOW-UNDERFLOW of EACH BIT AND BUFFER
prob_16_2 = get_prob_results(16,2)
prob_16_6 = get_prob_results(16,6)
prob_16_10 = get_prob_results(16,10)
```

```r
prob_16_14 = get_prob_results(16,14)
prob_16_18 = get_prob_results(16,18)
prob_32_2 = get_prob_results(32,2)
prob_32_6 = get_prob_results(32,6)
prob_32_10 = get_prob_results(32,10)
prob_32_14 = get_prob_results(32,14)
prob_32_18 = get_prob_results(32,18)

## GET TABLE OF OVERFLOW PROBABILITIES
table_overflow = matrix(list(), nrow = 2, ncol = 5)
table_overflow[1, 1] = prob_16_2[1]
table_overflow[1, 2] = prob_16_6[1]
table_overflow[1, 3] = prob_16_10[1]
table_overflow[1, 4] = prob_16_14[1]
table_overflow[1, 5] = prob_16_18[1]
table_overflow[2, 1] = prob_32_2[1]
table_overflow[2, 2] = prob_32_6[1]
table_overflow[2, 3] = prob_32_10[1]
table_overflow[2, 4] = prob_32_14[1]
table_overflow[2, 5] = prob_32_18[1]
print(table_overflow)
```

```
##      [,1]        [,2]        [,3]        [,4]        [,5]
## [1,] 0           0.003032588 0.005873976 0.008602839 0.01169957
## [2,] 0.006899393 0.008949689 0.01050235  0.01229741  0.01261919
```

```r
## GET TABLE OF UNDERFLOW PROBABILITIES
table_underflow = matrix(list(), nrow = 2, ncol = 5)
table_underflow[1, 1] = prob_16_2[2]
table_underflow[1, 2] = prob_16_6[2]
table_underflow[1, 3] = prob_16_10[2]
table_underflow[1, 4] = prob_16_14[2]
table_underflow[1, 5] = prob_16_18[2]
table_underflow[2, 1] = prob_32_2[2]
table_underflow[2, 2] = prob_32_6[2]
table_underflow[2, 3] = prob_32_10[2]
table_underflow[2, 4] = prob_32_14[2]
table_underflow[2, 5] = prob_32_18[2]
print(table_underflow)
```

```
##      [,1]        [,2]        [,3]        [,4]         [,5]
## [1,] 0.005165171 0.002212651 0.001368778 0.0005622234 0
## [2,] 0.006344494 0.003753414 0.002817457 0.001069359  0.0007892789
```

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

```r
#  your R simulation code here
i = c(2, 6, 10, 14, 18)
m = c(16, 32)
B = 20
sample_num = 500
lambda = 1

## Get random bits size of m for the secret message
get_random_secret_message_bits = function(m)
{
  message_bits = numeric(m)

  for (i in 1:m)
  {
    message_bits[i] = sample(c(0,1), 1)
  }

  return(message_bits)
}

## Get time for each packet source generation using IPD distribution
get_packet_times = function(idp, worst_case)
{
  packet_times = numeric(worst_case)

  for (i in 1:worst_case)
  {
    if (i == 1)
    {
      packet_times[i] = 0
    }

    else
    {
      packet_times[i] = packet_times[i-1] + idp[i-1]
    }
  }

  return(packet_times)
}

## Get overflow and underflow of the packet stream under IPD distribution
get_overflow_underflow = function(m, i)
{
  curr_buffer = i
  message_bits = get_random_secret_message_bits(m)

  worst_case = 49

  idp_expo = runif(worst_case, min = 0, max = 1)
  packet_times = get_packet_times(idp_expo, worst_case+1)
```

```
next_sent_time = packet_times[i]

sender_delay = runif(worst_case, min = 0, max = 1)
med = median(sender_delay)
min = min(sender_delay)
max = max(sender_delay)
message_bit_delay = rexp(worst_case, lambda)

underflow = 0
overflow = 0

next_buffer = i+1

for (j in 1:m)
{
  if (message_bits[j] == 0)
  {
    delay = runif(1, min, med)
    message_bit_delay[j] = delay
  }

  else if (message_bits[j] == 1)
  {
    delay = runif(1, med, max)
    message_bit_delay[j] = delay
  }

  next_sent_time = next_sent_time + delay

  # Update buffer
  if (next_sent_time <= packet_times[next_buffer])
  {
    curr_buffer = curr_buffer - 1

    if (curr_buffer < 1)
    {
      underflow = 1
      break
    }
  }

  else
  {
    curr_buffer = curr_buffer + 1

    if (curr_buffer > B)
    {
      overflow = 1
      break
    }
  }

  next_buffer = next_buffer + 1
  }

  return(c(overflow, underflow))
}

## Get results of overflow-underflow probabilities
get_prob_results = function(m, i)
{
  overflow_probs = numeric(sample_num)
  underflow_probs = numeric(sample_num)

  sample_count = 1

  for (j in 1:sample_num)
  {
    overflow = 0
    underflow = 0
    over_under = get_overflow_underflow(m, i)

    # Update overflow and underflow
    overflow = overflow + over_under[1]
    underflow = underflow + over_under[2]

    overflow_probs[j] = overflow/sample_count
    underflow_probs[j] = underflow/sample_count

    sample_count = sample_count + 1
  }

  return(c(mean(overflow_probs), mean(underflow_probs)))
}

# tabulate results and compute the probabilities

## GET PROBABILITIES OF OVERFLOW-UNDERFLOW of EACH BIT AND BUFFER
prob_16_2 = get_prob_results(16,2)
prob_16_6 = get_prob_results(16,6)
prob_16_10 = get_prob_results(16,10)
prob_16_14 = get_prob_results(16,14)
prob_16_18 = get_prob_results(16,18)
prob_32_2 = get_prob_results(32,2)
prob_32_6 = get_prob_results(32,6)
prob_32_10 = get_prob_results(32,10)
prob_32_14 = get_prob_results(32,14)
prob_32_18 = get_prob_results(32,18)

## GET TABLE OF OVERFLOW PROBABILITIES
table_overflow = matrix(list(), nrow = 2, ncol = 5)
table_overflow[1, 1] = prob_16_2[1]
table_overflow[1, 2] = prob_16_6[1]
table_overflow[1, 3] = prob_16_10[1]
table_overflow[1, 4] = prob_16_14[1]
table_overflow[1, 5] = prob_16_18[1]
```

```
table_overflow[2, 1] = prob_32_2[1]
table_overflow[2, 2] = prob_32_6[1]
table_overflow[2, 3] = prob_32_10[1]
table_overflow[2, 4] = prob_32_14[1]
table_overflow[2, 5] = prob_32_18[1]
print(table_overflow)
```

```
##      [,1]       [,2]        [,3]        [,4]        [,5]
## [1,] 0          0.002236672 0.004526182 0.006698252 0.009637963
## [2,] 0.00334637 0.005676068 0.006111403 0.006981795 0.009161537
```

```
## GET TABLE OF UNDERFLOW PROBABILITIES
table_underflow = matrix(list(), nrow = 2, ncol = 5)
table_underflow[1, 1] = prob_16_2[2]
table_underflow[1, 2] = prob_16_6[2]
table_underflow[1, 3] = prob_16_10[2]
table_underflow[1, 4] = prob_16_14[2]
table_underflow[1, 5] = prob_16_18[2]
table_underflow[2, 1] = prob_32_2[2]
table_underflow[2, 2] = prob_32_6[2]
table_underflow[2, 3] = prob_32_10[2]
table_underflow[2, 4] = prob_32_14[2]
table_underflow[2, 5] = prob_32_18[2]
print(table_underflow)
```

```
##      [,1]        [,2]        [,3]        [,4]        [,5]
## [1,] 0.008848133 0.006037272 0.003755708 0.001714236 0
## [2,] 0.009159119 0.007345162 0.004444078 0.006018308 0.003824818
```

4.3

Propose methods to deal with buffer overflow and underflow.

Answer

One of the most common techniques to avoid underflows and overflows in data exchange between two devices is to increase the maximum buffer size. In this case, we should increase the buffer size to a number larger than 20 (Wikipedia). Also, to enhance the effectiveness in dealing with the buffer overflow, instead of restricting the buffer parameters to 5 paramenters in the `i` list, we can increase such number of parameters by incrementing by 1 instead of 2 or 4 (Wikipedia)

5 References

5.1 For Problem 3: Design

- https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/nchar
- https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/rawConversion#:~:text=charToRaw%20converts%20a%20length%2Done,with%20%22%22%20for%200%20)
- http://yuba.stanford.edu/~nbehesht/mythesis-Dec2009/Chapter1-Introduction.pdf

- https://www.geo.fu-berlin.de/en/v/soga/Basics-of-statistics/Continous-Random-Variables/Continuous-Uniform-Distribution/Continuous-Uniform-Distribution-in-R/index.html#:~:text=The%20runif()%20function%20generates,min%20and%20the%20max%20argument.
- https://stats.stackexchange.com/questions/67911/how-to-sample-from-a-discrete-distribution

5.2 For Problem 4: Implementation

- ECS 132 Pizza Post @234
- https://www.geeksforgeeks.org/exponential-distribution-in-r-programming-dexp-pexp-qexp-and-rexp-functions/
- https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/matrix
- https://en.wikipedia.org/wiki/Buffer_underrun
- https://en.wikipedia.org/wiki/Buffer_overflow#Buffer_overflow_protection