CS425: Computer Networks
IIT Kanpur
Project 5: Internet Measurements
Date: Mon, Nov 14

Name: Neeraj Kumar
Roll: 13427
Email: neerajkr@iitk.ac.in
Dept: EE

Q 1.1

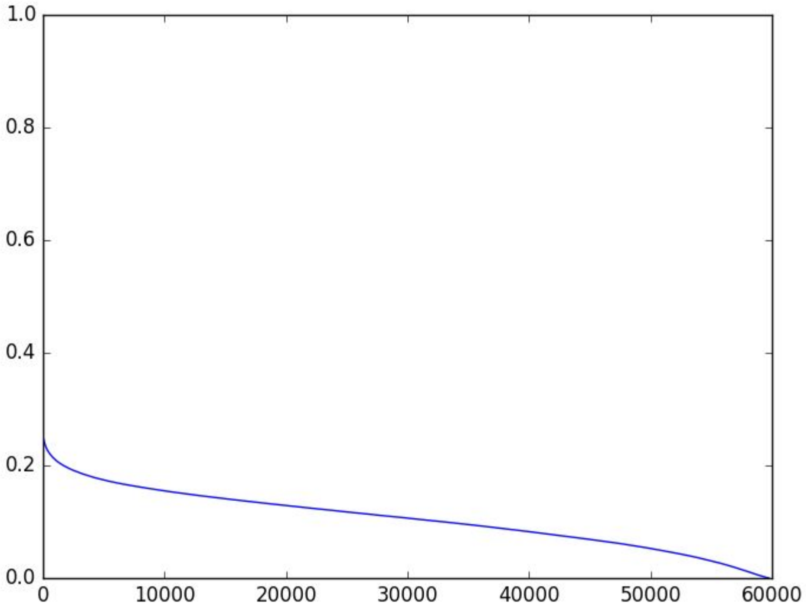
Avg. Packet Size : **768.180860115 bytes/packet**

Firstly, i calculated the total number of packets in all the traffic and then total number of bytes in the traffic. I divided total number of packets by total number of bytes to get Avg. Packet size. I wrote a python code for this.

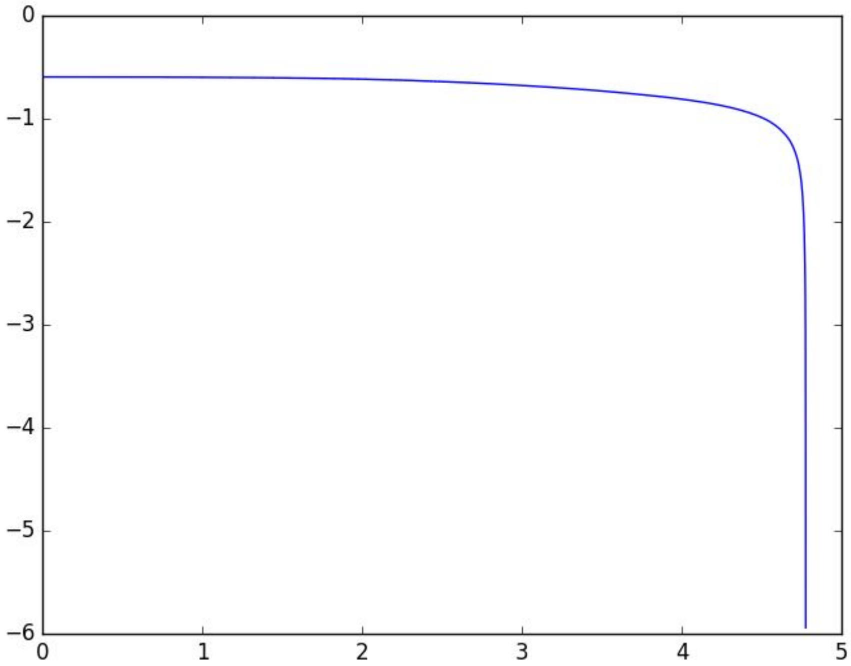
Q 1.2

Complementary Cumulative Probability Distribution (CCDF) of flow durations

1.Linear Scale

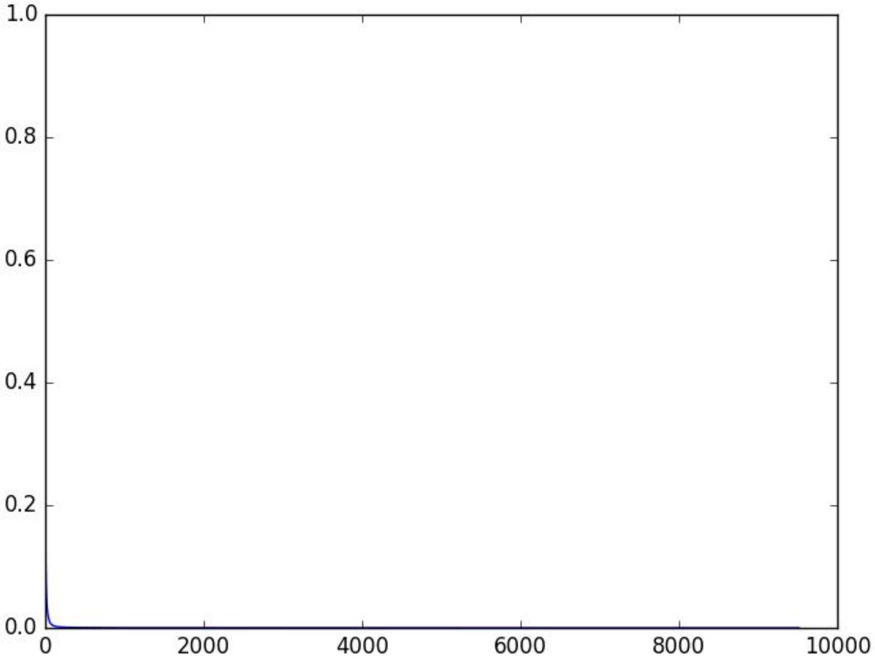


2. Logarithmic Scale

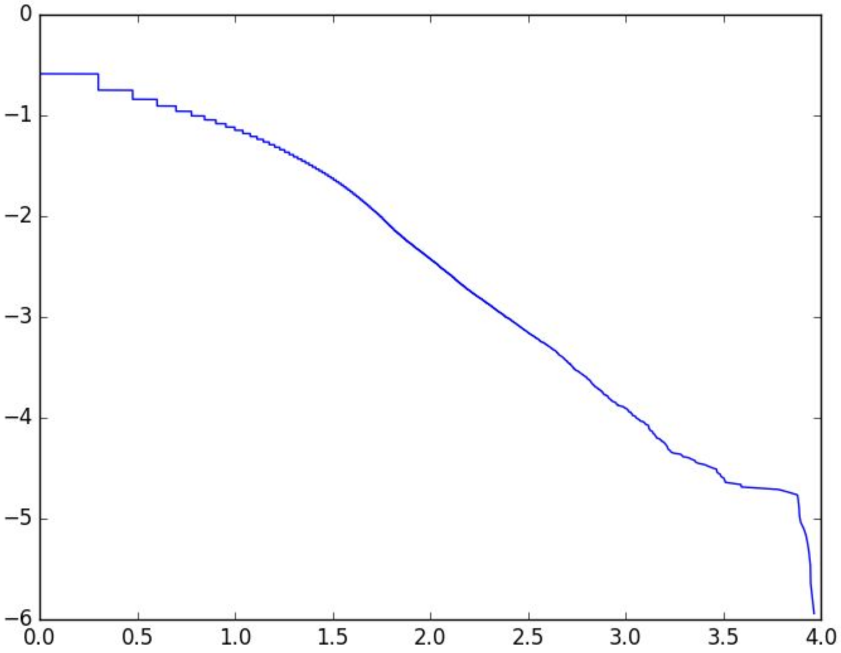


Complementary Cumulative Probability Distribution (CCDF) of Number Of packets

1. Linear scale

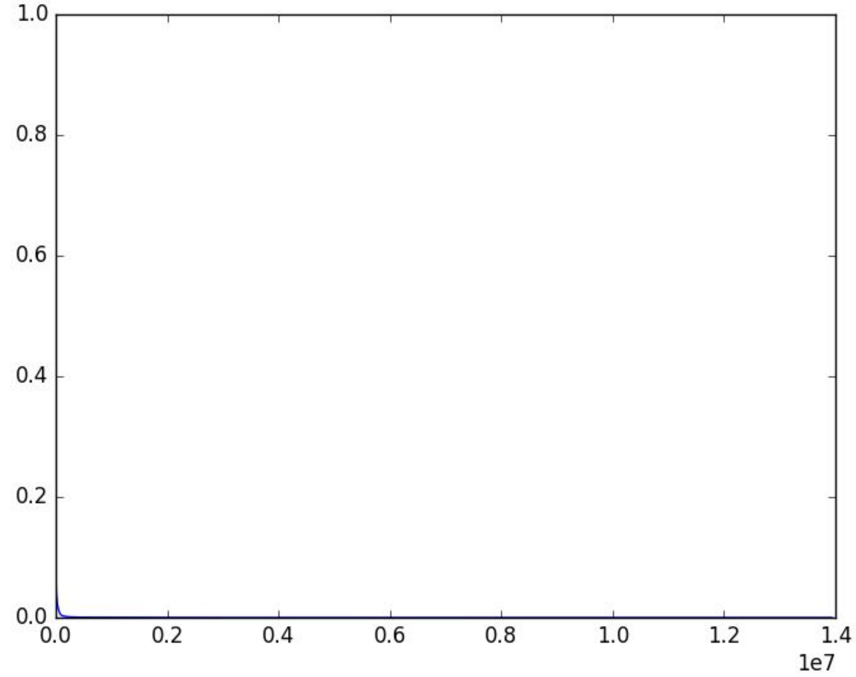


2. Logarithmic Scale

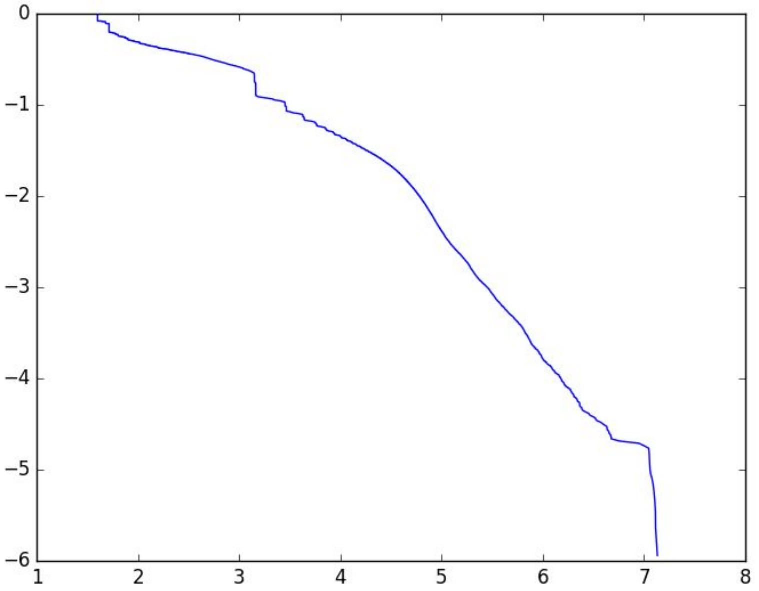


Complementary Cumulative Probability Distribution (CCDF) of Number of bytes:

1. Linear Scale



2. Logarithmic scale:



- On the linear scale, the CCDF for number of packets and bytes shows that the y-value comes to zero very sharply. This is because almost all of the packet length is very small, close to 1. Similar is the case for number of bytes in the traffic.
- While, the CCDF for flow duration shows that the flow duration of many packets are small but there are some packets having larger length as well.
- It is helpful to plot on the logarithmic scale because most of the packets and byte lengths are very small and because of this it is difficult to visualise on the plot. So, plotting on logarithmic scale will be helpful in visualising
- The reason behind these kind of plots is;- people often communicate between many server/sites and packets transferred between them is close to 1. E.g. we use only few sites like fb.com, google.com very often but we open very other sites only when needed i.e. transfeering few bytes/packets between theses sites.

Q 1.3

Top-ten port numbers by sender traffic volume

Port Number	Traffic(In number of bytes)	Percentage of total traffic
80	1309585549	43.93922921
33001	219443373	7.362766542
1935	109209645	3.664203249
22	64623818	2.168259075
443	51432480	1.725663153
55000	48388885	1.623544419
388	39899296	1.338701632
16402	22714732	0.7621249459
20	20021646	0.6717664939
0	18939605	0.635461841965

Top-ten port numbers by Receiver traffic volume

Port Number	Traffic(In number of bytes)	Percentage of total traffic

33002	119957708	4.024822381
80	87250983	2.927445972
49385	62341592	2.091685802
62269	36640981	1.22937861
443	23074506	0.7741960873
43132	22743259	0.7630820842
16402	22140759	0.7428669973
22	19324558	0.6483777894
5500	19306744	0.6477800939
0	18207368	0.610893817829

- From the sender side we see that, port 80 (http server) comprises of more than 40 % of total traffic. This shows that most of the traffic come from server i.e. response from server**.**
- port 1935, Macromedia Flash Communications Server MX, also contributes approx 4 % of sender traffic volume, showing that this port number has also responded many times.
- Port 33001 and 33002 have also contributed significantly in the traffic and most probably this is some client server because there is no description about this ip port number on IANA

Q 1.4

The fraction of the total traffic that comes from the most popular (by number of bytes) 0.1% of source IP prefixes = 1698471776/2980447251= 0.569871443096

The fraction of the total traffic that comes from the most popular (by number of bytes) 1% of source IP prefixes = 2357652915/2980447251= 0.791039973685

The fraction of the total traffic that comes from the most popular (by number of bytes) 10%of source IP prefixes = 2897521207/2980447251= 0.972176644303

The fraction of traffic (by bytes) that has a source mask of 0 = (2357652915-1691108868)/2357652915= 0.28271508616

After Excluding the traffic that has source mask zero:

The fraction of the total traffic that comes from the most popular (by number of bytes) 0.1% of source IP prefixes = 1691108868/478874491= 0.283171888021

The fraction of the total traffic that comes from the most popular (by number of bytes) 1% of source IP prefixes = 1078422671/1691108868= 0.637701505448

The fraction of the total traffic that comes from the most popular (by number of bytes) 10%of source IP prefixes = 1608576402/1691108868= 0.95119624315

Q 1.5

What fraction of the traffic (by bytes) in the trace is sent by Princeton?
= 20890679/2980447251= 0.00700924298962
What fraction of the traffic (by bytes) in the trace is sent to Princeton?
= 65318259/2980447251 =0.0219155896747

What fraction of the traffic (by packets) in the trace is sent by Princeton?
= 39352 /3879877= 0.0101425895718
What fraction of the traffic (by packets) in the trace is sent to Princeton?
= 56974 /3879877= 0.0146844861319

Q 2.1

I ignored the last update file. So, there were total of 8 files comprising of 120 minutes. I counted the total number of updates for each session and divided the count by 120.

Session 1:

How many BGP updates per minute does the session handle, on average? = 623.666666667

Session 2:

How many BGP updates per minute does the session handle, on average? = 1020.20833333

Session 3:

How many BGP updates per minute does the session handle, on average? =1959.05

Q 2.2

What fraction of IP prefixes experience no update messages?

Session 1:

NumOfUpdates: 487503
NumOfZeroUpdate: 482151
Percentage of Zero Update: 0.989021606021

Session 2:

NumOfUpdates: 500325
NumOfZeroUpdate: 478105
Percentage of Zero Update: 0.955588867236

Session 3:

NumOfUpdates: 511812
NumOfZeroUpdate: 448763
Percentage of Zero Update: 0.876812188851

Q2.3:

What prefix (or prefixes) experiences the most updates, and how frequent are they?

For each session, below are the top nine prefixes having most updates and there count/frequency is alongside

Session 1:

```
121.52.150.0/24 1020
121.52.144.0/24 1020
121.52.149.0/24 1020
121.52.145.0/24 1020
85.249.160.0/20 561
109.161.64.0/20 534
70.32.130.0/24 525
70.32.133.0/24 525
70.32.132.0/24 525
```

Session 2:

```
85.249.160.0/20 1113
89.221.206.0/24 1070
70.32.130.0/24 1036
70.32.133.0/24 1036
70.32.132.0/24 1036
165.193.245.0/24 1029
109.161.64.0/20 1027
121.52.150.0/24 1020
121.52.144.0/24 1020
```

Session 3:

```
109.161.64.0/20 1755
89.221.206.0/24 1640
70.32.130.0/24 1550
70.32.133.0/24 1548
70.32.132.0/24 1548
165.193.245.0/24 1547
192.58.232.0/24 1132
85.249.160.0/20 1113
121.52.149.0/24 1034
```

Q2.4:

What fraction of all update messages come from the most unstable 0.1% of prefixes? The most unstable 1% of prefixes? The most unstable 10% of prefixes?

Most unstable 10% of prefixes

Session 1:

MostFrequentUpdate: 1.0

Session 2:

MostFrequentUpdate: 1.0

Session 3:

MostFrequentUpdate: 0.954786854692

Most unstable 1% of prefixes

Session 1:

MostFrequentUpdate: 0.992890995261

Session 2:

MostFrequentUpdate: 0.734166582491

Session 3:

MostFrequentUpdate: 0.429906430969

Most unstable 0.1% of prefixes

Session 1:
MostFrequentUpdate: 0.455781096307

Session 2:
MostFrequentUpdate: 0.29338273994

Session 3:
MostFrequentUpdate: 0.164382910339

Q 2.5:

Conclusion:

- Number and frequency of updates is lowest in session 1 and highest in session 3
- We also conclude that most of the ip prefixes do not go through updates. This number is highest in the case of session 1 and lowest in case of session 3
- Top 1 % updates comprises of almost more than 75 % of total updates
- We also conclude that most of the prefixes are stable

Appendix:

Code for part 1:

```
import csv
import numpy as np
from pylab import *
from scipy.stats import norm
import matplotlib.pyplot as plt
from bisect import bisect_left
```

```
# i=0
# Header=[]
# row1=[]
# temp=[]
# NumberOfPackets=[]
# NumberOfBytes=[]
# FlowDuration=[]

# with open('ft-v05.2010-09-29.235501+0000.csv', 'rb') as f:
#    reader = csv.reader(f)
#        if i==0:
#                # print "Header\n"
#                Header=row;
#                # print row
#        else:
#                temp=row
#                NumberOfPackets.append(temp[4])
#                NumberOfBytes.append(temp[5])
#                FlowDuration.append(str(int(temp[7])-int(temp[6])))
#                # FlowDuration
#                if i==1:
#                        row1=row

#        i=i+1

# print i
# # print Header
# # print row1

#        # for i in range(len(Header)):
#        #        # print i
#        #        # print Header[i]+ " = "+ row1[i]
#        #        # print row1[i]

# print "Number Of data = " + str(len(NumberOfPackets))



# ############################### 1.1 ################################
# NumberOfBytes = list(map(int, NumberOfBytes))
# NumberOfPackets = list(map(int, NumberOfPackets))
# FlowDuration=list(map(int, FlowDuration))
# # print FlowDuration

# TotalNumPackets=0
# TotalNumBytes=0
# for i in range(len(NumberOfPackets)):
#        TotalNumPackets=TotalNumPackets + NumberOfPackets[i]
#            TotalNumBytes=TotalNumBytes + NumberOfBytes[i]

# print TotalNumPackets
# print TotalNumBytes
# print "Q1.1: Avg Packet Size= "+ str((TotalNumBytes*1.0)/TotalNumPackets)
# AvgNumPackets=(TotalNumPackets*1.0)/(len(NumberOfPackets))
# AvgNumBytes=(TotalNumBytes*1.0)/(len(NumberOfBytes))



# print "AvgNumPackets= "+str(AvgNumPackets)
# print "AvgNumBytes= "+str(AvgNumBytes)

# ############################ 1.2 #####################################


# # plt.show()
# a=FlowDuration
# sorted = np.sort(a)
# yvals = np.arange(1,len(sorted)+1)/float(len(sorted))
# plt.plot((sorted), (1-yvals))
# plt.show()
# plt.plot(np.log10(sorted), np.log10(1-yvals))
# plt.show()

# a=NumberOfPackets
# sorted = np.sort(a)
# yvals = np.arange(1,len(sorted)+1)/float(len(sorted))
# plt.plot(sorted, 1-yvals)
# plt.show()
# plt.plot(np.log10(sorted), np.log10(1-yvals))
# plt.show()

# # a=NumberOfBytes
# # sorted = np.sort(a)
# # yvals = np.arange(1,len(sorted)+1)/float(len(sorted))
# # plt.plot(sorted, 1-yvals)
# # plt.show()

# # print sorted(FlowDuration)
# a=NumberOfBytes
# sorted = np.sort(a)
# yvals = np.arange(1,len(sorted)+1)/float(len(sorted))
#http://stackoverflow.com/questions/31147893/logarithmic-plot-of-a-cumulative-distribution-function-in-matplotlib
# plt.plot(sorted, 1-yvals)
# plt.show()
# plt.plot(np.log10(sorted), np.log10(1-yvals))
# plt.show()
# ############################## 1.3
############################################################

# DictSenderPort={}
# DictRecvPort={}

# Header=1

# TotalNumBytes=0
# with open('ft-v05.2010-09-29.235501+0000.csv', 'rb') as f:
#    reader = csv.reader(f)
#    for row in reader:
#       if Header==1:
#               Header=0
#       else:
#                       TotalNumBytes=TotalNumBytes+int(row[5])
#                       if row[15] in DictSenderPort:
#                               DictSenderPort[row[15]]=DictSenderPort[row[15]]+int(row[5])
#                       else:
#                               DictSenderPort[row[15]]=int(row[5])

#                       if row[16] in DictRecvPort:
#                               DictRecvPort[row[16]]=DictRecvPort[row[16]]+int(row[5])
#                       else:
#                               DictRecvPort[row[16]]=int(row[5])


# d=DictRecvPort
# i=0

# print "Receiver Port"
# for w in sorted(d, key=d.get, reverse=True):
#       i=i+1
#       if i<(11):
#               print w, d[w],(d[w]*100.0)/TotalNumBytes

# print "Sender Port"
# d=DictSenderPort
# i=0
# for w in sorted(d, key=d.get, reverse=True):
#       i=i+1
#       if i<(11):
#               print w, d[w], (d[w]*100.0)/TotalNumBytes

# print TotalNumBytes


######################################## 1.4  ##########################
```

```
import socket, struct

def ip2long(ip):
    """
    Convert an IP string to long
    """
    packedIP = socket.inet_aton(ip)
    return struct.unpack("!L", packedIP)[0]


def Mask(len):
              ans=0
              for x in xrange(1,len+1):
                            ans=ans+pow(2,32-x)
              return ans



DictSourceIPTraffic={}

TotalTraffic=0;
TotalPackets=0
Header=1
pricetonTraffic1=0
pricetonTraffic2=0
pricetonTraffic1P=0
pricetonTraffic2P=0
with open('ft-v05.2010-09-29.235501+0000.csv', 'rb') as f:
    reader = csv.reader(f)
    for row in reader:
              if Header==1:
                            Header=0
              else:
                                          # if int(row[20])==0:
                                          #             continue
                                          TotalTraffic=TotalTraffic+int(row[5])
                                          TotalPackets=TotalPackets+ int(row[4])
                                          src_mask=Mask(int(row[20]))  # source mask length is at 20th index
                                          row[10]=ip2long(row[10])   # src address at 10th index
                                          row[10]=row[10]&src_mask   #masking


                                          des_mask=Mask(int(row[21]))  # source mask length is at 20th index
                                          row[11]=ip2long(row[11])   # src address at 10th index
                                          row[11]=row[11]&des_mask   #masking


                                          # row[10]=socket.inet_ntoa(struct.pack('!L', row[10]))  # again back to ip
                                          princetonIPPrefix=(ip2long("128.112.0.0")&Mask(16))
```

```
            if row[10]==princetonIPPrefix:
                # print "Matched"
                pricetonTraffic1=pricetonTraffic1+int(row[5])
                pricetonTraffic1P=pricetonTraffic1P+int(row[4])

            if row[11]==princetonIPPrefix:
                # print "Matched"
                pricetonTraffic2=pricetonTraffic2+int(row[5])
                pricetonTraffic2P=pricetonTraffic2P+int(row[4])



            # if row[10] in DictSourceIPTraffic:
            #   DictSourceIPTraffic[row[10]]=DictSourceIPTraffic[row[10]]+int(row[5])
            # else:
            #   DictSourceIPTraffic[row[10]]=int(row[5])




# d=DictSourceIPTraffic

# LenDict=len(d)
# i=0

# Sum=0
# for w in sorted(d, key=d.get, reverse=True):
#   i=i+1
#   if i<(0.001*LenDict):
#       print w, d[w]
#       Sum=Sum+d[w]

# print TotalTraffic
# print Sum
# print Sum*1.0/TotalTraffic

print pricetonTraffic1, TotalTraffic, pricetonTraffic1*1.0/TotalTraffic

print pricetonTraffic2, TotalTraffic, pricetonTraffic2*1.0/TotalTraffic

print pricetonTraffic1P, TotalPackets, pricetonTraffic1P*1.0/TotalPackets

print pricetonTraffic2P, TotalPackets, pricetonTraffic2P*1.0/TotalPackets
```

Code for question 2:

```
DictIP={}

def DictInit(FileName):
    global DictIP
    fileHandle=open(FileName, 'r')
    for line in fileHandle:
        fields=line.split('|')
        fields[5]="".join(fields[5].split())
        # fields[5].replace(" ", "")
        if '.' in fields[5]:
            if fields[5] not in DictIP:
                DictIP[fields[5]]=0
    fileHandle.close()


def DictIPUpdate(FileName):
    fileHandle=open(FileName, 'r')
    i=0
    for line in fileHandle:
        i=i+1
        fields=line.split('|')
        # fields[5].replace(" ", "")
        fields[5]="".join(fields[5].split())
        if '.' in fields[5]:
            if fields[5] in DictIP:
                DictIP[fields[5]]=DictIP[fields[5]]+1
            # continue

            # if i%100==0:
            #   print str(i)
            #   # print DictIP[fields[5]]
            #   print DictIP[fields[5]]

        # for fields[5] in DictIP:
        #   DictIP[fields[5]]=DictIP[fields[5]]+1


    fileHandle.close()


################################ 2.1 ############################################
def UpdateCount(FileName):
    fileHandle = open(FileName, 'r')
    i=0
    for line in fileHandle:
        fields = line.split('|')
        fields[5]="".join(fields[5].split())
        if '.' in fields[5]:                #IPv4
```

```
            i=i+1

    fileHandle.close()
    return i

CountJan=UpdateCount("updates.20140103.1200.txt")+UpdateCount("updates.20140103.1215.txt")+UpdateCount("updates.20140103.1230.txt")+UpdateCount("updates.20140103.1245.txt")+UpdateCount("updates.20140103.1300.txt")+UpdateCount("updates.20140103.1315.txt")+UpdateCount("updates.20140103.1330.txt")+UpdateCount("updates.20140103.1345.txt")
CountFeb=UpdateCount("updates.20140203.1200.txt")+UpdateCount("updates.20140203.1215.txt")+UpdateCount("updates.20140203.1230.txt")+UpdateCount("updates.20140203.1245.txt")+UpdateCount("updates.20140203.1300.txt")+UpdateCount("updates.20140203.1315.txt")+UpdateCount("updates.20140203.1330.txt")+UpdateCount("updates.20140203.1345.txt")
CountMar=UpdateCount("updates.20140303.1200.txt")+UpdateCount("updates.20140303.1215.txt")+UpdateCount("updates.20140303.1230.txt")+UpdateCount("updates.20140303.1245.txt")+UpdateCount("updates.20140303.1300.txt")+UpdateCount("updates.20140303.1315.txt")+UpdateCount("updates.20140303.1330.txt")+UpdateCount("updates.20140303.1345.txt")

print CountJan
print CountFeb
print CountMar

print "Total Count: " + str(CountJan+CountFeb+CountMar)
AvgNumUpdateJan=(CountJan)/120.0
print "AvgNumUpdateJan: "+str(AvgNumUpdateJan)
AvgNumUpdateFeb=(CountFeb)/120.0
print "AvgNumUpdateFeb: "+str(AvgNumUpdateFeb)
AvgNumUpdateMar=(CountMar)/120.0
print "AvgNumUpdateMar: "+str(AvgNumUpdateMar)


##################################### 2.2 ###############################################
DictInit("../rib/rib.20140103.1200.txt")
print "Dictionary Initialised"
DictIPUpdate("updates.20140103.1200.txt")
DictIPUpdate("updates.20140103.1215.txt")
DictIPUpdate("updates.20140103.1230.txt")
DictIPUpdate("updates.20140103.1245.txt")
DictIPUpdate("updates.20140103.1300.txt")
DictIPUpdate("updates.20140103.1315.txt")
DictIPUpdate("updates.20140103.1330.txt")
DictIPUpdate("updates.20140103.1345.txt")

d= DictIP
i=0
LenDict=len(DictIP)

TotalUpdateCount=0
```

```
NumOfUpdates=0
NumOfZeroUpdate=0
UpdateCount=0
for w in sorted(d, key=d.get, reverse=True):
    i=i+1
    NumOfUpdates=NumOfUpdates+1
    TotalUpdateCount=TotalUpdateCount+d[w]
    if d[w]==0:
        NumOfZeroUpdate=NumOfZeroUpdate+1
    if i<(10):
        print w, d[w]
    if i<(LenDict*0.001):
        UpdateCount=UpdateCount+d[w]

print "NumOfUpdates: " ,NumOfUpdates
print "NumOfZeroUpdate:",NumOfZeroUpdate
print "Percentage of Zero Update: ", NumOfZeroUpdate*1.0/NumOfUpdates

print "MostFrequentUpdate: ",UpdateCount*1.0/TotalUpdateCount


DictInit("../rib/rib.20140203.1200.txt")
print "Dictionary Initialised"
DictIPUpdate("updates.20140203.1200.txt")
DictIPUpdate("updates.20140203.1215.txt")
DictIPUpdate("updates.20140203.1230.txt")
DictIPUpdate("updates.20140203.1245.txt")
DictIPUpdate("updates.20140203.1300.txt")
DictIPUpdate("updates.20140203.1315.txt")
DictIPUpdate("updates.20140203.1330.txt")
DictIPUpdate("updates.20140203.1345.txt")


TotalUpdateCount=0
NumOfUpdates=0
NumOfZeroUpdate=0
d= DictIP
i=0
UpdateCount=0
LenDict=len(DictIP)
for w in sorted(d, key=d.get, reverse=True):
    i=i+1
    TotalUpdateCount=TotalUpdateCount+d[w]
    NumOfUpdates=NumOfUpdates+1
    if d[w]==0:
        NumOfZeroUpdate=NumOfZeroUpdate+1
    if i<(10):
        print w, d[w]
    if i<(LenDict*0.001):
        UpdateCount=UpdateCount+d[w]

print "NumOfUpdates: " ,NumOfUpdates
print "NumOfZeroUpdate:",NumOfZeroUpdate
print "Percentage of Zero Update: ", NumOfZeroUpdate*1.0/NumOfUpdates
print "MostFrequentUpdate: ",UpdateCount*1.0/TotalUpdateCount

DictInit("../rib/rib.20140303.1200.txt")
print "Dictionary Initialised"
DictIPUpdate("updates.20140303.1200.txt")
DictIPUpdate("updates.20140303.1215.txt")
DictIPUpdate("updates.20140303.1230.txt")
DictIPUpdate("updates.20140303.1245.txt")
DictIPUpdate("updates.20140303.1300.txt")
DictIPUpdate("updates.20140303.1315.txt")
DictIPUpdate("updates.20140303.1330.txt")
DictIPUpdate("updates.20140303.1345.txt")




TotalUpdateCount=0
NumOfUpdates=0
NumOfZeroUpdate=0
d= DictIP
i=0
UpdateCount=0
for w in sorted(d, key=d.get, reverse=True):
    i=i+1
    NumOfUpdates=NumOfUpdates+1
    TotalUpdateCount=TotalUpdateCount+d[w]
    if d[w]==0:
        NumOfZeroUpdate=NumOfZeroUpdate+1
    if i<(10):
        print w, d[w]

    if i<(LenDict*0.001):
        UpdateCount=UpdateCount+d[w]
print "NumOfUpdates: " ,NumOfUpdates
print "NumOfZeroUpdate:",NumOfZeroUpdate
print "Percentage of Zero Update: ", NumOfZeroUpdate*1.0/NumOfUpdates

print "MostFrequentUpdate: ",UpdateCount*1.0/TotalUpdateCount
```