Internet Measurements
CS425- Computer Networks

Vaibhav Nagar (14785)
Email: vaibhavn@iitk.ac.in

November 14, 2016

1. Traffic Measurement

1.1 What is the average packet size, across all traffic in the trace? Describe how you computed this number.

Average Packet Size = Total sum of all bytes(doctets) / Total sum of number of packets(dpkts) in each flow

$$Average\ Packet\ Size = 768.18$$

1.2 Plot the Complementary Cumulative Probability Distribution (CCDF) of flow durations (i.e., the finish time minus the start time) and of flow sizes (i.e., number of bytes, and number of packets). First plot each graph with a linear scale on each axis, and then a second time with a logarithmic scale on each axis. What are the main features of the graphs? What artifacts of Netflow and of network protocols could be responsible for these features? Why is it useful to plot on a logarithmic scale?

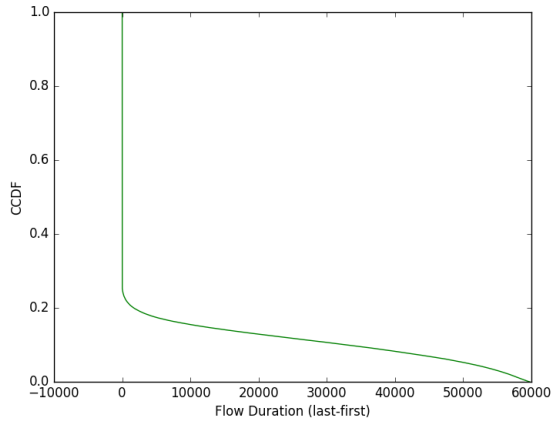

Figure 1: Complementary cumulative distribution of flow durations on linear scale

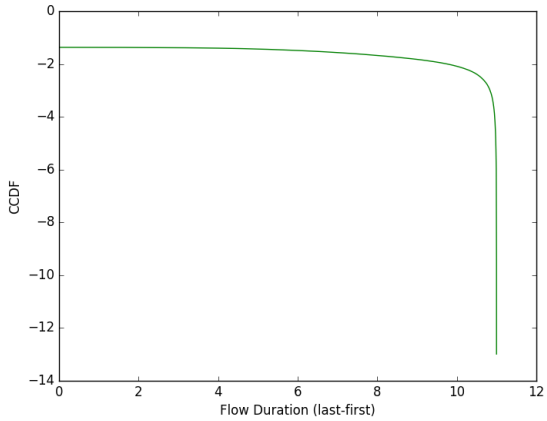

Figure 2: Complementary cumulative distribution of flow durations on logarithmic scale

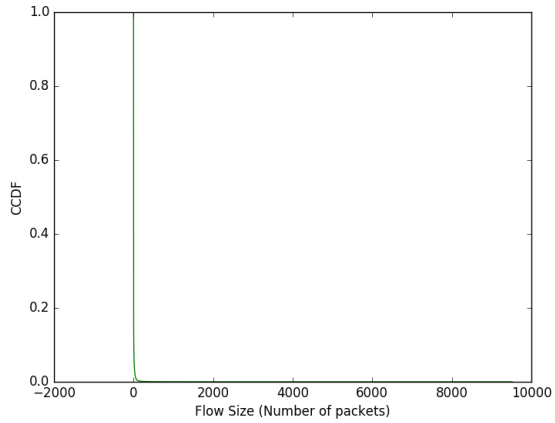

Figure 3: Complementary cumulative distribution of flow size(number of packets) on linear scale

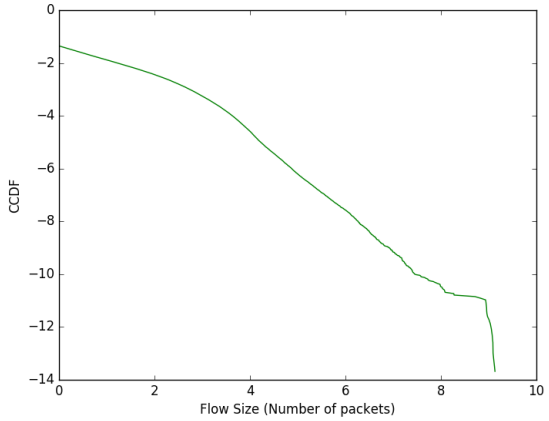

Figure 4: Complementary cumulative distribution of flow size(number of packets) on logarithmic scale

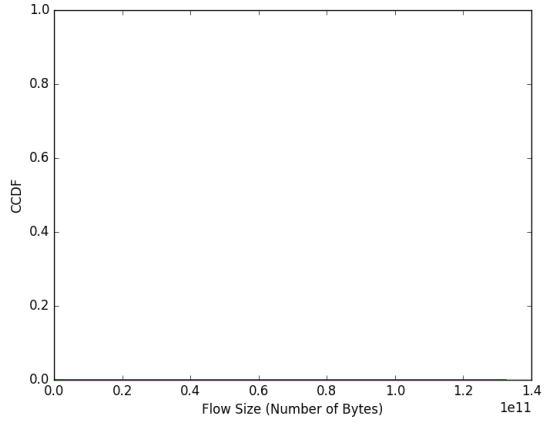

Figure 5: Complementary cumulative distribution of flow size(number of bytes) on linear scale

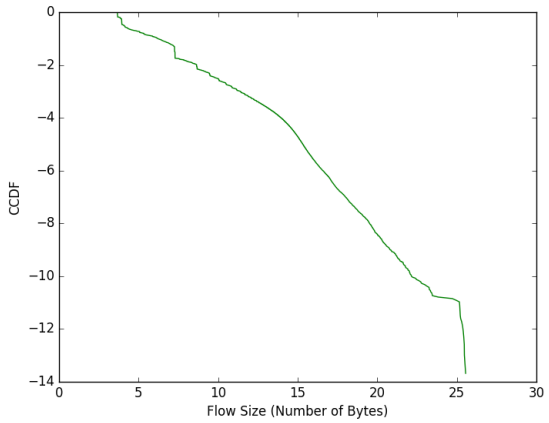

Figure 6: Complementary cumulative distribution of flow size(number of bytes) on logarithmic scale

- Graph of CCDF of flow durations shows that there are huge number of flows whose duration is zero and then after $x = 0$ it gradually decreases. The logarithmic graph of ccdf of flow duration shows clearly that after $e^10duration$ there is significant fall.
- CCDF graph of flow size (both number of packets and number of bytes) doesn't reveal much as linear scale is too large and large number of packets are concentrated at smaller range. So the graph rapidly falls to 0 just after $x = 0$.
- Their logarithmic graphs clearly shows the packet density falls after certain point.
  The logarithmic graph of ccdf of flow size in bytes start with $y = 0$ upto almost $x = 4$ which is due to minimum acceptable size of packet is at least 8 bytes (UDP) and 20 bytes(TCP) and thus drastically decreases after $x = 25$ due to maximum acceptable size of packets.
- Plotting in logarithmic scale shrinks the x-axis. In linear scale when some values are too large and most of the data are concentrated in small region then the graph doesn't render much info. Logarithmic graphs perfectly responds to skewness towards large values and shows percent change or multiplicative factors.

1.3 Summarize the traffic by which TCP/UDP port numbers are used. Create two tables, listing the top-ten port numbers by sender traffic volume (i.e., by source port number) and by receiver traffic volume (i.e., by destination port number), including the percentage of traffic (by bytes) they contribute. Where possible, explain what applications are likely responsible for this traffic. Explain any significant differences between the results for sender vs. receiver port numbers

Sender traffic at port 80 is maximum as expected. Since port 80 is well-known and registered port for HTTP protocol and thus used by global servers. Port 22 and 443 also contributes for high traffic value as port 22 is used for SSH and 443 for HTTPS globally.

Port Numbers	Percentage of traffic (in bytes)
80	43.94
33001	7.36
1935	3.66
22	2.17
443	1.72
55000	1.62
388	1.34
16402	0.76
20	0.671
873	0.58

Table 1: Top-ten port numbers by *sender* traffic volume

Port Numbers	Percentage of traffic (in bytes)
33002	4.02
80	2.98
49385	2.09
62269	1.23
443	0.77
43132	0.76
16402	0.74
22	0.65
5500	0.64
57493	0.35

Table 2: Top-ten port numbers by *receiver* traffic volume

Servers send large number of bytes to receivers whose port may not necessarily be the well-known or registered port, when requested by clients. Thus percentage of receiver traffic is not as much high as senders.

1.4 Aggregate the traffic volumes based on the source IP prefix. What fraction of the total traffic comes from the most popular (by number of bytes) 0.1% of source IP prefixes? The most popular 1% of source IP prefixes? The most popular 10% of source IP prefixes? Some flows will have a source mask length of 0. Report the fraction of traffic (by bytes) that has a source mask of 0, and then exclude this traffic from the rest of the analysis. That is, report the top 0.1%, 1%, and 10% of source prefixes that have positive mask lengths.

1. Fraction of the total traffic comes from the most popular (by number of bytes):

- 0.1% of source IP prefixes = 0.58944
- 1% of source IP prefixes = 0.82253
- 10% of source IP prefixes = 0.98383

2. The fraction of traffic (by bytes) that has a source mask of 0 = 0.43259
3. Now after excluding the traffic that has source mask 0, fraction of the total traffic comes from the most popular (by number of bytes):

   - 0.1% of source IP prefixes = 0.27005
   - 1% of source IP prefixes = 0.45622
   - 10% of source IP prefixes = 0.56383

1.5 Princeton has the 128.112.0.0/16 address block. What fraction of the traffic (by bytes and by packets) in the trace is sent by Princeton? To Princeton?

1. Fraction of the traffic in the trace is sent by Princeton:

   - In terms of Bytes = 0.00701368
   - In terms of Packets = 0.02191742

2. Fraction of the traffic in the trace is sent to Princeton:

   - In terms of Bytes = 0.010148
   - In terms of Packets = 0.014689

2. BGP Measurement

2.1 How many BGP updates per minute does the session handle, on average? Count each announcement or withdrawal for a single prefix as a BGP update, even if multiple prefixes appear in a single update message or a single prefix has multiple announcement or withdrawal messages in a short period of time. Describe how you computed this result.

Average BGP updates per minute is calculated by counting the number of BGP update messages in the text file for all intervals of a session. Update messages for IPv6 are excluded and all prefixes are counted in a single update message. Then updates per minute is averaged by dividing total count by number of minutes in a session.

- **Session 1:** RouteViews on January 3, 2014 between 12pm and 2pm from Sydney, Australia

$$Average\ BGP\ updates\ per\ minute = 623.66$$

- **Session 2:** RouteViews on February 3, 2014 between 12pm and 2pm from Sydney, Australia

$$Average\ BGP\ updates\ per\ minute = 1020.21$$

- **Session 3:** RouteViews on March 3, 2014 between 12pm and 2pm from Sydney, Australia

$$Average\ BGP\ updates\ per\ minute = 1959.05$$

2.2 What fraction of IP prefixes experience no update messages? (Count each prefix equally, independently of what fraction of address space they cover or whether one prefix is contained inside another.)

- **Session 1:** RouteViews on January 3, 2014 between 12pm and 2pm from Sydney, Australia
  - Fraction of IP prefixes experience no update messages: 0.95736
- **Session 2:** RouteViews on February 3, 2014 between 12pm and 2pm from Sydney, Australia
  - Fraction of IP prefixes experience no update messages: 0.93444
- **Session 3:** RouteViews on March 3, 2014 between 12pm and 2pm from Sydney, Australia
  - Fraction of IP prefixes experience no update messages: 0.88125

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

Prefixes that experiences the most updates are:

- **Session 1:** RouteViews on January 3, 2014 between 12pm and 2pm from Sydney, Australia
  - 121.52.149.0/24 = 1020 updates in two hour session
  - 121.52.144.0/24 = 1020 updates in two hour session
  - 121.52.145.0/24 = 1020 updates in two hour session
  - 121.52.150.0/24 = 1020 updates in two hour session
- **Session 2:** RouteViews on February 3, 2014 between 12pm and 2pm from Sydney, Australia
  - 89.221.206.0/24 = 712 updates in two hour session
- **Session 3:** RouteViews on March 3, 2014 between 12pm and 2pm from Sydney, Australia
  - 109.161.64.0/20 = 728 updates in two hour session

2.4 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?

Fraction of all update messages come from the most unstable:

Session	0.1% of prefixes	1% of prefixes	10% of prefixes
Session-1	0.3876	0.8443	0.8504
Session-2	0.2791	0.7026	0.8800
Session-3	0.0718	0.2278	0.5727

Table 3: Fraction of all update messages come from unstable prefixes

2.5 Briefly summarize your results and what you learned about BGP stability from them.

After looking at the fraction of prefixes which is close to 1, this indicates that very less number of prefixes announce or withdraw. Still the average updates per minute is quite high in all three sessions which indicates BGP instability. Thus due to large number of updates per minute BGP packets consumes significant amount of bandwidth and CPU resources on routers and disrupt the delivery of data traffic.

3. Python Scripts (Python-3.5.2)

3.1 Plot CCDF of traffic flow for Q1.1 and Q1.2

```
import pandas as pd
import sys
import numpy as np
import matplotlib.pyplot as plt


def plot_ccdf(dataset, xstring, ystring, img, fig_num, log_scale):
  dataset.sort()
  xvals = np.array(list(set(dataset)))
  xvals.sort()

  counts = []
  c = 0
  d_prev = dataset[0]

  for d in dataset:
    if d_prev != d:
      d_prev = d
      counts = counts + [c]
      c = 1
    else:
      c += 1
  counts = counts + [c]

  assert len(counts) == len(xvals)

  cdf = np.cumsum(counts)

  assert len(dataset) == cdf[len(cdf)-1]

  yvals = 1 - cdf/float(len(dataset))

  if log_scale:
    xvals = np.log(xvals)
    yvals = np.log(yvals)
  else:
    xvals = np.append([xvals[0]-1], xvals)
    yvals = np.append([1],yvals)
    xvals = np.append([xvals[0]-1], xvals)
    yvals = np.append([1],yvals)


  assert len(xvals) == len(yvals)

  plt.figure(fig_num)
  plt.plot(xvals,yvals, color="green")
  plt.xlabel(xstring)
  plt.ylabel(ystring)
```

```
# plt.show()
  plt.savefig(img)


try:
  data = pd.read_csv("ft-v05.2010-09-29.235501+0000.csv")
except:
  print("Unable to read csv file")
  sys.exit(1)

# extract number of packets in the flow <dpkts>
dpkts = data.dpkts

# extract bytes in the flow <doctets>
doctets = data.doctets

assert len(dpkts) == len(doctets)

packet_list = [int(pk) for pk in dpkts]
octets_list = [int(oc) for oc in doctets]

avg = sum(octets_list)/sum(packet_list)

print("Total Size: ", sum(octets_list))
print("Number of Samples: ", len(doctets))
print("Average Packet Size: ", avg)

finish = data['last']
start = data['first']

assert len(finish) == len(start)

durations = [f-s for f,s in zip(finish, start)]

plot_ccdf(durations, "Flow Duration (last-first)", "CCDF", "traffic_q1.2.1.png", 1, False)
plot_ccdf(durations, "Flow Duration (last-first)", "CCDF", "traffic_q1.2.1_log.png", 2,
    True)

plot_ccdf(packet_list, "Flow Size (Number of packets)", "CCDF", "traffic_q1.2.2.png", 3,
    False)
plot_ccdf(packet_list, "Flow Size (Number of packets)", "CCDF", "traffic_q1.2.2_log.png",
    4, True)

plot_ccdf(octets_list, "Flow Size (Number of Bytes)", "CCDF", "traffic_q1.2.3.png", 5,
    False)
plot_ccdf(octets_list, "Flow Size (Number of Bytes)", "CCDF", "traffic_q1.2.3_log.png", 6,
     True)
```

3.2 Fractions for Q1.3 and Q1.4 and Q1.5

```
import pandas as pd
import sys
import numpy as np
import matplotlib.pyplot as plt
import operator

def top_ten(port_list, octet_list, proto_list):
  _traffic ={}

  for port in list(set(port_list)):
    _traffic[int(port)] = 0

  print(len(_traffic))
  total_volume = 0
```

```
  for port, vol, proto in zip(port_list, octet_list, proto_list):
    if  (int(proto) == 6) or (int(proto) == 17):
      _traffic[int(port)] += int(vol)
    total_volume += int(vol)

  for key in _traffic:
    _traffic[key] = (_traffic[key]/total_volume)*100

  return sorted(_traffic.items(), key=operator.itemgetter(1), reverse=True)[:10]


def ip_traffic_fraction(ipaddr, octet_list, mask_list, masked, fractions):
  ip_traffic = {}
  for ip in list(set(ipaddr)):
    ip_traffic[ip] = 0

  total_traffic = 0
  for ip, octets, mask in zip(ipaddr, octet_list, mask_list):
    if masked:
      if mask != 0:
        ip_traffic[ip] += octets
    else:
      ip_traffic[ip] += octets
    total_traffic += octets

  for f in fractions:
    f = f/100
    a = dict(sorted(ip_traffic.items(), key=operator.itemgetter(1), reverse=True)[:int(len
    (ip_traffic)*f)])
    print("Fraction of the total traffic comes from the most popular ", f*100," percent of
     source IP prefixes: ", sum(a.values())/total_traffic)
  return


def ip_traffic_zero_mask(octet_list, mask_list):
  res = 0
  tot = 0
  for oc, mask in zip(octet_list,mask_list):
    if mask == 0:
      res += oc
    tot += oc
  print(tot)
  print(res)
  print("Fraction of traffic (by bytes) that has a source mask of 0", res/tot)


def specific_ip_traffic(ipaddr, _list, ip_string, addr_block):
  total_traffic = 0
  res = 0
  for ip, octets in zip(ipaddr, _list):
    if ip_string in ip[:addr_block]:
      res += octets
    total_traffic += octets
  print(total_traffic)
  print(res)

  return res/total_traffic


try:
  data = pd.read_csv("ft-v05.2010-09-29.235501+0000.csv")
except:
  print("Unable to read csv file")
```

```
  sys.exit(1)

sender = top_ten(data.srcport, data.doctets, data.prot)
reciever = top_ten(data.dstport, data.doctets, data.prot)
print("Top Ten Senders: ")
print(sender)
print("Top Ten Recievers: ")
print(reciever)

ip_traffic_fraction(data.srcaddr, data.doctets, data.src_mask, False, [0.1, 1, 10])

ip_traffic_zero_mask(data.doctets, data.src_mask)

print("Non zero mask traffic:")

ip_traffic_fraction(data.srcaddr, data.doctets, data.src_mask, True, [0.1, 1, 10])

p1 = specific_ip_traffic(data.srcaddr, data.doctets, "128.112", 16)
p2 = specific_ip_traffic(data.srcaddr, data.dpkts, "128.112", 16)
p3 = specific_ip_traffic(data.dstaddr, data.doctets, "128.112", 16)
p4 = specific_ip_traffic(data.dstaddr, data.dpkts, "128.112", 16)

print("fraction of the traffic (by bytes) in the trace is sent by Princeton: ", p1)
print("fraction of the traffic (by packets) in the trace is sent by Princeton: ", p3)
print("fraction of the traffic (by bytes) in the trace is to by Princeton: ", p2)
print("fraction of the traffic (by packets) in the trace is to by Princeton: ", p4)
```

3.3 Average BGP updates for Q2.1

Change session value accordingly.

```
import pandas as pd
import sys
import numpy as np

session = "1"

intervals = [1200, 1215, 1230, 1245, 1300, 1315, 1330, 1345]

updates = []

for interval in intervals:
  filename = "./updates-asst4/updates.20140" + session + "03." + str(interval) + ".txt"

  try:
    print(filename)
    cols = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O"]
    data = pd.read_csv(filename , sep="|", header=None, names=cols)
  except:
    #raise
    print("Unable to read file")
    sys.exit(1)

  count = 0
  for aw, entry in zip(data["C"], data["F"]):     # index "F" is the sixth column
    if aw == "A" or aw == "W":
      ent = entry.split()     # multiple prefixes in one update entry
      for e in ent:
        if ':' not in e:  # not ipv6
          count += 1
  updates += [count]

print(updates)
print(sum(updates))
```

```
print("Average BGP updates per minute: ", sum(updates)/120)
```

3.4 Fractions for Q2.2, Q2.3 and Q2.4

Change session value accordingly.

```
import pandas as pd
import sys
import numpy as np
import operator

try:
  cols = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O"]
  rib = pd.read_csv("./rib/rib.20140103.1200.txt" , sep="|", header=None, names=cols)
except:
  raise
  print("Unable to read file")
  sys.exit(1)

rib_data = list(set(rib["F"]))

session = "3"

intervals = [1200, 1215, 1230, 1245, 1300, 1315, 1330, 1345]

updates_data = []
update_messages = 0

for interval in intervals:
  filename = "./updates-asst4/updates.20140" + session + "03." + str(interval) + ".txt"

  try:
    print(filename)
    cols = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O"]
    data = pd.read_csv(filename , sep="|", header=None, names=cols)
  except:
    #raise
    print("Unable to read file")
    sys.exit(1)

  count = 0
  update_messages += len(data["F"])
  for entry in data["F"]:       # index "F" is the sixth column
    ent = entry.split()       # multiple prefixes in one update entry
    for e in ent:
      if ':' not in e:      # not ipv6
        updates_data += [e]


print("This may take some time...")

ip_updates = {}
count = 0
for prefix in rib_data:
  if ":" not in prefix:
    ip_updates[prefix] = updates_data.count(prefix)
    if ip_updates[prefix] == 0:
      count += 1

print("count: ",count)
print("rib size: ", len(rib["F"]))
print("rib_data size(non-repititive: ",len(rib_data))
print("Total Update Messages: ", update_messages)
```

```
58 print("Fraction of IP prefixes experience no update messages: ", count/len(rib_data))
59
60 b = dict(sorted(ip_updates.items(), key=operator.itemgetter(1), reverse=True)[:10])
61 print(b)
62
63 fractions = [0.1, 1, 10]
64 for f in fractions:
65   f = f/100
66   a = dict(sorted(ip_updates.items(), key=operator.itemgetter(1), reverse=True)[:int(len(
       ip_updates)*f)])
67   print("Sum of update messages for fraction ", f*100, ": ", sum(a.values()))
68   print("Fraction of all update messages come from the most popular ", f*100," percent of
       prefixes: ", sum(a.values())/update_messages)
```