Computer Communication Networks
CS 6/75202
G1 Project
Source Traffic Modeling and Generation

Author: Xiao Zou

Abstract
Source Traffic Modeling and Generation is a subproject of project - “Research and Development of IP Packet Processors”. Its objective is to simulate various Internet traffic and generate IP packet flow at runtime in order to test the functions and performance of a router application, which is going to be implemented by other components of main project. This technical report will discuss the characteristics of Internet traffic, some important distributions used to describe different aspects of traffic, and several effective ways to simulate Internet traffic. The algorithms of traffic generation are given in report.

1. Introduction
Source traffic modeling and generation is a very important aspect of research about network communication. It is a starting point and somehow a basis of network traffic processing.

First, an accurate estimation to the network traffic is the basis to handle/control the traffic. This is the task of traffic modeling. A good estimation to network traffic (PDF, CDF, mean, variance, persistence of clustering, heavy tail, etc) helps to solve the questions in the design of router’s algorithm. For example, which parameter is crucial and is easy to get, how to handle short-term data burst? It is also an important factor to the resource configuration of routers and switches. For example, how large the buffer should be for each communication channel?

Second, traffic generation provides a powerful and flexible tool to test all kinds of service designed for network communication. Developing such a tool is the primary goal of this subproject.

Many works have been done in this area and the approaches cover every aspects of Internet traffic. For examples, San-gi Li and his group developed SMAQ for traffic modeling and queuing analysis [8]. P. Tran-Gia discussed traffic modeling of Wireless application [10]. While IPv6 has not yet been widely used, Liren Zhang and others have shown their interest on IPv6 traffic and its QoS performance [3].

Although it is hard to dig deep into most popular questions about traffic modeling and generation with this in-course project, we still do our best to discuss some characteristics of Internet traffic, some important distributions used to describe different aspects of traffic, and several effective approaches to generate Internet traffic in following part of this technical report.

2. Traffic Modeling
The purpose of traffic modeling is to find out the statistical characteristics of various Internet traffics. Many relative works have been done. Some results and conclusions are shown in papers [1][2][11];

According to the requirement of our main project, traffic modeling is less emphasized than traffic generation. But it is still important because:
- We need find out how close the generated synthetic traffic is to real traffic.
- The data collected from Internet for traffic analysis can also be used to generate traffic flow and test the performance of IP packet processor.
- The analysis result can be used in subproject G10 – Statistics.

To perform traffic modeling, we use TCPDUMP developed at Lawrence Berkeley National Laboratory. It can be downloaded through URL address (ftp://ftp.ee.lbl.gov/tcpdump.tar.Z). TCPDUMP is a powerful tool that allows us to collect network packets and make some statistical analysis out of those dumps. Dumped data are saved into flat files for later usage.

With the data collected by TCPDUMP, some further analysis can be done using TCP-Reduce, which is a collection of shell scripts for reducing a TCPDUMP trace file to a summary of the corresponding TCP connections. TCP-Reduce gives a list of identified TCP/IP connections together with the related data. In particular, the following data are collected:

- Connection start time
- Connection duration
- Higher-level protocol carried
- Port of connection
- Number of bytes transferred between server and client
- IP Server & Client addresses
- State the connection ended to

Another software tool TCPTRACE, which is downloadable from Ohio University, can also be used to analyze TCPDUMP files and report information such as elapsed time, bytes and segments sent and received, retransmissions, round trip times, window advertisements, throughput, and more. It can also produce a number of graphs for further analysis.

3. Traffic Generation

Traffic generation and traffic modeling are really two sides of a coin. Many important works have been done. Mark W. Garrett gives analysis, modeling and generation of self-similar VBR video traffic [5]. Kai Below discussed the Internet traffic generation for large simulation scenarios [6]. E. Casilari talked about how to use neural network in the generation of ATM video traffic [7].

Out traffic generation will simulate Internet traffic by randomly generating traffic flow for different applications. Each flow will contain a train of packets. The flow could either come from real network traffic log or is generated synthetically from given probability density functions. In following sections, we will discuss several different approaches of Internet traffic generation.

3.1 Trace Traffic

The traffic flow is generated from TCPDUMP trace file, which record the real Internet traffic in a long enough time interval.

According to the requirement of main project, a flow is described by 6 important elements:

- The number of IP packets in the flow
- The number of bytes in the flow
- Staring time of the flow (Unix time in ms)
- Ending time of the flow (Unix time in ms)
- Transport protocol: TCP (6), UDP (17), Others (47, ...)
- Port of transport protocol (Optional)

A sample list of flows is as follows:

<table>
<thead>
<tr>
<th>Packets</th>
<th>Bytes</th>
<th>Start</th>
<th>End</th>
<th>Protocol</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1500</td>
<td>3913644961</td>
<td>3913644961</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>40</td>
<td>3913644961</td>
<td>3913644961</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>40</td>
<td>3913644961</td>
<td>3913644961</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>40</td>
<td>3913644961</td>
<td>3913644961</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>40</td>
<td>3913644961</td>
<td>3913644961</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>48</td>
<td>3913644961</td>
<td>3913644961</td>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>132</td>
<td>3913644961</td>
<td>3913682958</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
<td>3913644961</td>
<td>3913684145</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>9202</td>
<td>3913644961</td>
<td>3913686568</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>172</td>
<td>3913644961</td>
<td>3913692399</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>1200</td>
<td>3913644961</td>
<td>3913702043</td>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>33080</td>
<td>3913644961</td>
<td>3913702459</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>2922</td>
<td>3913644961</td>
<td>3913704792</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>88</td>
<td>3913644962</td>
<td>3913644962</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>94</td>
<td>3913644962</td>
<td>3913644962</td>
<td>17</td>
<td>0</td>
</tr>
</tbody>
</table>

Table-1

Obviously, we can easily generate synthetic traffic flows like above table using TCP-Reduce on TCPDUMP trace files. The IP packets for each flow can be obtained separately from same group of TCPDUMP trace files. This work can be done by either UNIX shell script or specially designed C/C++ code.

For test purpose of IP packet processor and reducing the memory size of Queue in project G7, only header information of IP packet is generated. The format of each packet follows the IPv4 header definition and it is delivered as follows:

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>IHL</td>
<td>Type of Service</td>
<td>Total Length</td>
<td></td>
</tr>
<tr>
<td>Identification</td>
<td>Flags</td>
<td>Fragment Offset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time to Live</td>
<td>Protocol</td>
<td>Header checksum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To generate special flow for major applications like TCP/UDP, FTP, HTTP, Email, etc, we can change the command line parameter and filter expression of TCPDUMP to get particular flows and packets of flow required by our project.

3.2 Synthetic Traffic

Synthetic traffic is generated from given Probability Density Functions (PDF). The generation of traffic is divided into two parts: generation of IP packets, and synthetic flow.

Before we discuss the detail of traffic generation, we list all of random distributions and their generation algorithms used in this project. The distributions and algorithms that
3.2.1 Random Distributions and Generation Algorithms

**Binomial Distribution**

Usage: Model the number of resource available; number of packets that reach the destination without loss; number of bits in error in a packet.

PDF: ‘x’ is the number of successes in a sequence of n Bernoulli trials

\[
f(x) = \binom{n}{x} p^x (1-p)^{n-x}
\]

Mean: np  
Variance: np(1-p)

Generation Algorithm:

Algorithm Binomial(n, p)
Begin
C = 0
For (i=0; i<n; i++)
    S = rand() / RAND_MAX
    If S \leq p
        C = C+1;
    Return C;
End

**Poisson Distribution**

Usage: Model call arrivals, number of tasks in the system, number of requests to a server, number of failed components, and message length;

PDF: ‘x’ is the number of successes in a sequence of Bernoulli trials’

\[
P(x = k) = \frac{\lambda^k}{k!} e^{-\lambda}, \ k = 0, 1, ...
\]

Mean: \(\lambda\)  
Variance: \(\lambda\)

Generation Algorithm:

Algorithm Poisson (\(\lambda\) )
Begin
R = e^{-\lambda}
C = 0
Initialize array U
U[0] = rand() / RAND_MAX
DO
    U[i] = U[n-1] * rand() / RAND_MAX
    If U[n] \leq R < U[n-1]
        Break;
While (1)
Return C;
End

**Geometric Distribution**
Usage: Model the number of attempts between successive failure (or successes), the number of packets in a message.

PDF: 'x' represents the number of trials up to including the first success in a sequence of Bernoulli trials. 'p' denotes the probability of success, '1-p' is the probability of failure, \(0 \leq p \leq 1\), and \(x = 1, 2, \ldots, \infty\)

\[
f(x) = (1 - p)^{x-1} p \quad x = 1, 2, \ldots, \infty
\]

Mean: \(\frac{1}{p}\)  
Variance: \(\frac{1-p}{p^2}\)

**Generation Algorithm:**

Algorithm Geometric( p )
Begin
\[ u = \frac{\text{rand}()}{\text{RAND\_MAX}} \]
Return \[ \left\lfloor \frac{\ln(u)}{\ln(1-p)} \right\rfloor \]
End

**Normal Distribution**
Usage: model the sum of a large number of independent random variables according to central limit theorem

PDF: \[
f(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}
\]

Mean: \(\mu\)  
Variance: \(\sigma^2\)

**Generation Algorithm:**

Algorithm Normal(n, \(\mu\), \(\sigma\))
Begin
\[ U = \frac{\text{rand}()}{\text{RAND\_MAX}} \]
For (i=1; i<n; i++) Do
\[ U = U + \frac{\text{rand}()}{\text{RAND\_MAX}} \]
\[ T = (U - n/2) / \sqrt{n/12} \]
Return \(\mu + \sigma T\)
End

**Uniform Distribution**
Usage: model the distance between source and destination

PDF:
\[
f(x) = \begin{cases} 
1/(b-a) & \text{for } a \leq x \leq b \\
0 & \text{otherwise}
\end{cases}
\]

Mean: \((a + b)/2\)  
Variance: \((b-a)^2/12\)

**Generation Algorithm:**

Algorithm Uniform(a, b)
Begin
\[ U = \frac{\text{rand}()}{\text{RAND\_MAX}} \]
Return \(a + (b - a)U\)
End
**Exponential Distribution**
Usage: Model the inter-arrival times between successive events
PDF: ‘λ’ is mean arrival rate of events.
\[
f(x) = \begin{cases} 
0 & x < 0 \\
\lambda e^{-\lambda x} & 0 \leq x < \infty 
\end{cases}
\]
Mean: \(1/\lambda\)  
Variance: \(1/\lambda^2\)
Generation Algorithm:

```
Algorithm Exponential(\(\lambda\))
Begin
    U = rand() / RAND_MAX
    Return \(-\ln(U) / \lambda\)
End
```

**Pareto Distribution**
Usage: Model the bursty/fractal/self-similar traffic, for example, packet interval time and burst length
PDF: ‘a’ – determine the mean and variance of random variables.
\[
f(x) = \begin{cases} 
0 & x < 1 \\
a x^{(a+1)} & 1 \leq x < \infty 
\end{cases}
\]
Mean and Variance: \(a/(a-1), a > 1\)  
\[a/[(a-1)^2(a-2)], a > 2\]
Generation Algorithm:

```
Algorithm Pareto( a )
Begin
    U = rand() / RAND_MAX
    Return \(1 / U^{1/a}\)
End
```

3.2.2 Generation of IP Packets
Before generating traffic data, the caller of traffic generation module should specify the value of several basic parameters, which include the starting time of traffic, the duration of traffic, and the type of traffic.

The IP packet header fields influenced by traffic generation algorithm are:
- Type of Service (TOS)
- Total Length
- Identification

Other fields are designated as follows:
- Version ← 4
- IHL (Internet Head Length) ← 5
- Flags ← Don’t care
- Fragment Offset ← Don’t care
- Time to Live ← 255
- Protocol ← Specified by caller
- HeaderChecksum ← Don’t care
- Source Address ← Specified by caller
- Destination Address ← Specified by caller

Another property not showing up in IP packet header is the interarrival times between packets, which is also generated by our algorithm.

We have several different approaches considering different situation of Internet communication. In following discussion, all approaches of traffic generation use above configurations.

**No Error with Short Term Clustering**
This type of traffic considers high quality, error free communication network. It is supposed there is no data error in packet during transmission and no packet is dropped between two communicating entities due to network congestion or inappropriate configuration of Internet route information.

Within the IP packet header, the value of TOS is always generated through a random variable with Uniform Distribution. IPv4 define 5 possible values for TOS. So we always call Uniform(1, 5) to set TOS.

The length of IP packet usually follows Poisson distribution. The call to algorithm Poisson(λ) will generate length of packet in bytes – Total Length in IP packet header. Usually, we use same λ to generate all IP packets for one flow. But packets generation for different flows will probably use different λ-value. 'λ' denotes the average length of IP packet.

Identification is a number generated in sequence.

If the clustering of data packet occurs in short term (small time scale), we can assume the arrival pattern of IP packet is also in Poisson distribution. Then, the interarrival time between consecutive packets follows Exponential distribution. We call Exponential(λ) to generate the interarrival time between packets and we apply the time value to IP packet in following way:

Before any IP packet of a flow can be delivered to IP packet processor, all packets as well as their starting times are generated first and saved in a priority queue. Packets in queue are sorted by their starting times. Given starting time – $T_p$ of one packet, its successor’s starting time $T_s$ is calculated as:

$$T_s = T_p + \text{Exponential}(\lambda)$$

When traffic generation function is called to serve traffics, it can

1) Deliver whole priority queue to caller and let caller decide what to do next; OR
2) Deliver packets to an output stream specified by caller one by one. The pace and rate of delivery is controlled by the starting time of each packet.

**Error with Short-Term Clustering**
This type of traffic considers the error/loss of data packet in unreliable network communication. The loss of data packet may be caused by either network congestion or the quality of communication channel. Packet could also be rejected by destination due to the error in data packet.
We assume the error/loss of packets may not happen at same location where our IP packet processor runs, but our IP packet processor will observe the phenomenon that same packet is transmitted repeatedly.

Traffic generation in this circumstance is similar to error-free network communication in many aspects except that a packet may be transmitted several times. Two different approaches are used to simulate the packet transmission with respect to errors.

1. Generate IP packets just like what we do in previous error-free network. The number of packets that can reach destination without loss or rejection is described by Binomial distribution.

Suppose 'n' is the number of packets we previously generated for a flow. 'p' is the probability reflecting the quality of communication. We calculate the total number of repeated packets as follows:

\[
\text{Algorithm RepeatPacketsEST}(n, p) \\
\text{Begin} \\
\quad A[0] = n \\
\quad M = 0 \\
\quad \text{While Binomial}(A[M], p) > 1, \text{Do} \\
\quad \quad M = M + 1 \\
\quad \quad A[M] = (A[M-1] - \lceil \text{Binomial}(A[M-1], p) \rceil) \\
\text{Return } \sum_{i=1}^{M} A[i] \\
\text{End}
\]

The total number of packets that should be transmitted is:

\[
(n + \text{RepeatPacketsEST}(n, p))
\]

2. Generate IP packets just like what we do in previous error-free network. For each packet, the number of attempts to transmit successfully is described by Geometric distribution.

Suppose 'n' is the number of packets we previously generated for a flow. 'p' is the probability that a packet can reach destination with no error. We calculate the total number of packets that should be transmitted as:

\[
\text{Algorithm TotalPacketsEST}(n, p) \\
\text{Begin} \\
\quad C = 0; \\
\quad \text{For (i=0; i<n; i++) Do} \\
\quad \quad C = C + \text{Geometric}(p) \\
\text{Return } \lceil C \rceil \\
\text{End}
\]

**Error with Self-Similarity**

This type of traffic considers unreliable network communication. Different with above two situations, the packet arrival pattern is no longer Poisson distribution. The arrival time of packets show self-similarity. This will change the distribution of interarrival times of packets from Exponential distribution to Pareto distribution.

The packet generation is similar to 'Error with Short-Term Clustering'. But the interarrival time between consecutive packets is calculated using: Pareto(a)
Caller of traffic generation algorithm can specify different value of ‘a’ to control the distribution of interarrival times of packets. If $a \leq 2$, the distribution has infinite variance, and if $a \leq 1$, the distribution has infinite mean and variance. So, caller can generate traffic with Heavy-Tail though controlling the value of ‘a’.

### 3.2.3 Synthetic Flow

As traffic flows in Table-1, each flow is described by 1) the number of IP packets; 2) the number of bytes; 3) starting time; 4) ending time; 5) transport protocol; and 5) port (optional). These values should be given as properties of Event object in IP Packet Processor.

The basic process of generating one traffic flow is as follows:

1. Given the starting time, duration ($T_D$), source address, destination address, protocol, number of packets, mean arrival rate of packets ($\lambda$), and type of traffic of flow
2. According to the traffic type, algorithm generates IP packets for the flow as we discussed in section 3.2.2.
3. Algorithm initializes an Event object with required information and fire the Event to IP Packet Processor
4. According to IP Packet Processor’s choice, algorithm delivers the packets to processor

Traffic generation algorithm provides several different ways to generate a bunch of flows automatically. This is about generation of flow list like the one shown in Table-1. Algorithm requires three parameters: the very beginning time ($T_s$) of earliest flow; the ending time ($T_E$) of last flow; and the maximum packet arrival rate $R$.

Traffic generation algorithm will create a list of flows as follows:

1. Call $\text{Poisson}( (T_s - T_E)R )$ to generate the total number of flows
2. For each flow, generate the values for each flow in following way
   - Starting time $\to$ generated by calling $\text{Uniform}(T_s, T_E)$
   - Number of packets $\to$ generated by calling $\text{Geometric}(1/\text{Uniform}(1, R))$
   - Protocol $\to$ generated by calling $\text{Uniform}(1, 20)$
   - Port $\to$ generated by calling $\text{Uniform}(1, 100)$

   Generate all packets for the flow
   - Ending time $\leftarrow$ Starting time + $\sum$ (all time intervals)
   - Number of bytes $\leftarrow$ $\sum$ (all packet’s length)
3. Create Event queue for generated flow list
4. Fire all Events to IP Packet Processor and wait for further instruction to deliver IP packets of all flows

The second way to generate flows is using flow list generated by TCP-Reduce. That means using real traffic log to generate synthetic traffic flows. The only difference with above approach is the just flow list. Flow is generated synthetically as before.

The third way is, given maximum number of flows caller want, the $\mu$ and $\sigma$ of number of flows in system, traffic generation algorithm call $\text{Normal}(n, \mu, \sigma)$ to generate the number of flows in list. All other works are same as first approach. The convenience is caller can create a series of flow list that each contains a bunch of flows. This is useful to test performance of IP Packet Processor with long-term traffic and heavy congestion.
4. C/C++ Code

The C/C++ code can be found in files: sourcetraffic.tar.gz. It is written under Linux and compiled using 'gcc'.

5. Conclusion

The source traffic modeling and generation is still ongoing research. We use some convenient tools like TCPDUMP, TCP-Reduce to try to catch some essential characteristic of Internet traffic. With this knowledge and experience, we create \( \alpha \)-version of traffic generation module and discussed the detail of our solution in this technical report. Apparently, our approaches are not enough to simulate all kinds of Internet traffic. But we will keep moving forward and add more traffic generation algorithms in the future.

Reference: