Computer Science Masters Project

Traffic Generator for an On-line Simulator

By

Murat YUKSEL

Project Advisor **:** Prof. K. VASTOLA

Sponsoring Faculty Member **:** Prof. B. SZYMANSKI

Summer 99
Rensselaer Polytechnic Institute

ACKNOWLEDGEMENT

This work was supported in part by DARPA contract number F19628-98-C-0057.

TABLE OF CONTENTS

1. INTRODUCTION

Simulation of the Internet is a challenging problem. Rapid growth in size, difficulty in characterization of topology, various link properties, different protocols, difficulty of traffic generation are stated to be the basic reasons for the difficulty of simulating the Internet. [11]

Traffic generation is mostly dependent on characterization of the traffic, also called traffic modeling, for the kinds of applications taking place in the network. In the case of the Internet, the main applications are WWW, Telnet, ftp, SMTP, and NMTP. [9] Generation of the accurate traffic in a pre-defined topology includes also the true randomness of source, destination, and on-off times of each session for the applications. So, for a candidate simulator of the Internet, there is a need for a traffic generator which uses the correct traffic models for each kind of application and randomizes source, destination, and on-off times of sessions. This report contains the information about a project, the aim of which is to construct a traffic generator as mentioned above.

In Section 2, the general description of the project is included. In Section 3, the necessary information about using the traffic generator as a tool is given. Next in Section 4, the underlying techniques are described, and finally in Section 5, the results of the tests of the traffic generator are given.

2. OVERVIEW OF THE PROJECT

2.1. Problem Statement

One of the basic steps in simulating a real network is to generate the traffic that is able to capture the behavior of the traffic in the real network. Accuracy of the generated traffic in simulation depends on the usage of correct distributions for packet arrivals of applications, such as ftp, WWW, Telnet, and also the true randomization of source, destination, and on-off times of each session. With these facts in mind, it is very important to have a traffic generator that can be adjusted with the most important parameters for each kind of application, such as mean rate, variance, mean number of sessions, mean inter-arrival times of sessions, and mean duration times of sessions.

There is a need for such a traffic generator in the project "Network Management and Control Using On-line Collaborative Simulation", which is being supervised by three professors of the ECSE and CS Departments, and is supported by DARPA. Detailed information about the project can be found at the Web site http://networks.ecse.rpi.edu/~olsim.

2.2. Project Description

The objective of this CS Masters Project is to generate the necessary ns and C++ code for constructing the traffic generator described in the previous section. ns is a simulation language specifically developed for network simulation. It was developed in CS Department of UC-Berkeley, by using C++. Detailed information about ns can be found at Web site http://mash.cs.berkeley.edu/ns/ns.html.

The traffic generator must be able to generate the traffic for applications like ftp, Telnet, WWW, SMTP, which contributes the main part of the Internet traffic. The most accurate distributions for session arrivals and the most accurate traffic models for each application's traffic data should be used. Some introductory information about the conventional traffic models is available in Appendix D, and some helpful publications about the traffic models are [1], [6], and [2].

The traffic generator should take as input the mean number of sessions, the mean inter-arrival time, and the mean duration time for each application. It should also take parameters for the behavior of each application's traffic data, such as mean rate, variance, packet size, burst time, and Hurst parameter.

The traffic generator will be used in the simulator of the project "Network Management and Control Using On-line Collaborative Simulation". The implemented and tested ns Tcl script of the traffic generator will be inserted into the simulator's ns Tcl script, and parameters of it will be adjusted by the other teams in the mentioned project.

3. USING THE TRAFFIC GENERATOR

3.1. How to Run

The traffic generator is implemented in ns Tcl script. So, it must be run in the ns environment. Before using the generator one must make sure that "`$ns expand-port-field-bits 23`" is available just after "`set ns [New Simulator]`" in the ns Tcl script. This command maximizes the possible number of agents that can be attached to a single node. 23 is the maximum possible value for the number of port-field-bits, but smaller values can also be used for efficiency purposes depending on the size and structure of the simulation.

All necessary directions for running the traffic generator is available in file "paramforgenerator.tcl", listed in Appendix A.1.

3.2. Parameters

Although all possible parameters are listed in Figure 3-1, we need to give more detailed information about some of them.

`NODES` defines the number of nodes in the topology. It is a very important parameter for randomization of sources and destinations. In the generator, randomization of sources and destinations is done depending on that variable. So, one must number his/her nodes from 1 to `NODES`; so that the traffic generator won't try to assign a non-existing host as source or destination. All items representing a host in the topology must be within `[1, NODES]`.

`PER_TCP` represents the percentage of TCP traffic in the whole traffic. One must know that the generator will assign the rest of the traffic as UDP traffic.

`SPD` represents the smallest possible value that can be recognized by ns as the difference between the times of two consecutive events in the simulation. Actually in ns, it is not possible to schedule two events with less than 0.1 seconds between them. Thus, it is safer to use a value greater than or equal to 0.1 for `SPD`.

`M_SMTP`, `M_FTP`, `M_WWW`, `M_TELNET`, and `M_X11` represent the average number of sessions available in the topology at any instant. If one wants not to generate traffic for a specific application, he/she must assign the mean number of sessions of that application as zero. For example, if one wants not to generate Telnet traffic, he/she must assign 0 to `M_TELNET`.

All inter-arrival times (`MIAT_*`) and duration times (`MDT_*`) must be given in seconds and must not have ns's own units (e.g. ms, s) at the end.

`MR_SMTP` must not have ns's own units (e.g. k, b, B) at the end, and must be given in "Bytes per second".

MPS_X11, MPS_WWW must not have ns's own units at the end, and must be given in "Bytes".

MR_X11, MR_WWW must not have ns's own units at the end, and must be given in "Kbits".

Parameter	Meaning	Range	Default	Type
NODES	Number of nodes in the topology	>1	-	Integer
PER_TCP	Percentage of TCP traffic	[0,1]	-	Real
SPD	Smallest possible delay	>=10^(-15)	-	Real
M_SMTP	Mean number of SMTP sessions	>=0	-	Integer
MNS_SMTP	Mean number of sinks per SMTP session	(0,NODES)	-	Integer
MIAT_SMTP	Mean inter-arrival time for SMTP sessions	(0,STOP_TIME-START_TIME)	-	Real
MNB_SMTP	Mean number of Bytes for SMTP sessions	>0	-	Integer
VAR_MNB_SMTP	Variance for MNB_SMTP	>0	-	Real
MPS_SMTP	Mean packet size for SMTP sessions	>0	-	Integer
MBS_SMTP	Mean burst size for SMTP sessions	(0,MNB_SMTP/MR_SMTP)	-	Real
MIT_SMTP	Mean idle time for SMTP sessions	(0,MNB_SMTP/MR_SMTP)	500ms	Real
MR_SMTP	Mean rate for SMTP sessions	>0	-	Real
M_X11	Mean number of X11 sessions	>=0	-	Integer
MIAT_X11	Mean inter-arrival time for X11 sessions	(0,STOP_TIME-START_TIME)	-	Real
MDT_X11	Mean duration time for X11 sessions	>0	-	Real
MR_X11	Mean rate for X11 sessions	>0	-	Real
MPS_X11	Mean packet size for X11 sessions	>0	-	Integer
H_X11	Hurst parameter for X11 traffic	(0,1)	0.92	Real
M_FTP	Mean number of FTP sessions	>=0	-	Integer
MIAT_FTP	Mean inter-arrival time for FTP sessions	(0,STOP_TIME-START_TIME)	-	Real
MDT_FTP	Mean duration time for FTP sessions	>0	-	Real
MPS_FTP	Mean packet size for FTP sessions	>0	-	Integer
MBS_FTP	Mean burst size for FTP sessions	(0,MDT_FTP)	-	Real
MIT_FTP	Mean idle time for FTP sessions	(0,MDT_FTP)	500ms	Real
MR_FTP	Mean rate for FTP sessions	>0	-	Real
M_WWW	Mean number of WWW sessions	>=0	-	Integer
MIAT_WWW	Mean inter-arrival time for WWW sessions	(0,STOP_TIME-START_TIME)	-	Real
MDT_WWW	Mean duration time for WWW sessions	>0	-	Real
MR_WWW	Mean rate for WWW sessions	>0	-	Real
MPS_WWW	Mean packet size for WWW sessions	>0	-	Integer
H_WWW	Hurst parameter for WWW traffic	(0,1)	0.92	Real
M_TELNET	Mean number of TELNET sessions	>=0	-	Integer
MIAT_TELNET	Mean inter-arrival time for TELNET sessions	(0,STOP_TIME-START_TIME)	-	Real
MDT_TELNET	Mean duration time for TELNET sessions	>0	-	Real
VAR_TELNET	Variance for MDT_TELNET	>0	-	Real
MPS_TELNET	Mean packet size for TELNET sessions	>0	-	Integer
MBS_TELNET	Mean burst size for TELNET sessions	(0,MDT_TELNET)	-	Real
MIT_TELNET	Mean idle time for TELNET sessions	(0,MDT_TELNET)	500ms	Real
MR_TELNET	Mean rate for TELNET sessions	>0	-	Real
START_TIME	Starting time of the simulation	>=0	-	Real
STOP_TIME	Stopping time of the simulation	>START_TIME	-	Real

Figure 3-1: Description of parameters to the traffic generator.

4. TECHNIQUES

4.1. Method for Generating Traffic

The generator includes several randomization operations performed by using random variables available in ns. Sources and destinations are randomized by two different uniform random variables that are `rvuSource` and `rvuSink`. In order to select whether to create a TCP agent or a UDP agent another uniform random variable, `rvuTCPUDP`, is used. This random variable returns a real value between 0 and 1, and then decision is made depending on the relationship (greater or less) between the TCP percentage (given as parameter `PER_TCP`) and the returned value from `rvuTCPUDP`.

During the generation of the traffic there was need for a sorted linked list data structure for maintaining the ending times of the sessions for each application. Thus, we embedded a sorted linked list object into ns as "Application/Traffic/LList". Each element of the linked list stores ending time of a session, and an additional integer type field for practical purposes. The linked list itself is in ascending order upon ending the times. This new object can be called from ns Tcl as "`set list [new Application/Traffic/LList]`". A new element can be inserted into the linked list by "`$list Insert 0.54 3`", where 0.54 corresponds to the "ending time" and 3 corresponds to the "integer field". In order to get the values of the first element of the linked list, one can use "`$list TimeOfFirst`" for the ending time, and "`$list NoOfFirst`" for the integer field of it. In order to delete the first element of the linked list, one must use "`$list DeleteFirst`". The other option for deleting an element is to use the integer field if it is unique. For example, "`$list Delete 3`" will delete the particular element with integer field 3.

The method for generating the traces for each application was almost the same except SMTP, because of its different characteristics. The algorithm for Telnet, WWW, ftp, and X11 traces is shown in Figure 4-1. The basic idea is to create sessions at the mean number of sessions (given as parameter) and insert them into a sorted linked list depending on their ending times. Next, until there is no element left in the list, take and delete the first element of the list, which is the earliest stopping session, and create its next recurrence and insert into linked list again if the new ending time is within the simulation time. The number of the next recurrences can be 0, or 1, or 2 mostly depending on the number of currently available sessions in the list. The next recurrences are generated in a two step process. In the first step, with a probability $\left(1-\left(\frac{\text{Current Number of Sessions in the List}}{2\bullet(\text{Required Mean Number of Sessions})}\right)\right)$ a new session is generated. Next in the second step, with a probability 0.5 a new session is generated. This two step process causes number of active sessions to be varying, which is more realistic than staying the same. The first step limits the number of active sessions with an upper bound $(2\bullet(\text{Required Mean Number of Sessions}))$, and a lower bound 1. The second step causes the number of active sessions to be varying, whereas the first step tries to make it closer to the required mean number of sessions.

When the mean number of sessions is large, the sorted linked list brings a very good efficiency. Instead of comparing every session's ending time with the stopping time of the simulation, we can directly take the first element of the linked list for the generation of the next session. However, if the mean number of sessions is very few, then it might slow down the process a little bit. Assuming that the traffic generator will be run for big topologies, using a sorted linked list is a better manner.

SMTP traces have different characteristics. SMTP traffic occurs from one source to a group of destinations at the same time. So, traces must be generated depending on this characteristic of SMTP traffic. Figure 4-2 lists the algorithm for generating SMTP traces. The only difference from the others is to generate multiple destinations for each session. This is done with an exponential random variable with an average value given as parameter `MNS_SMTP`.

1. Set up the random variables for generating inter-arrival times and duration times of sessions.
2. Create a linked list for maintaining the sessions.
3. For each session do the followings:
- *3.1. Generate a random source.*
- *3.2. Generate a random destination different from the source.*
- *3.3. Make selection whether to use TCP or UDP depending on the given TCP percentage.*
- *3.4. Identify the source and sink agents depending on the selection made in the previous step.*
- *3.5. Randomize the starting time and the duration time.*
- *3.6. Insert the session into linked list with its ending time.*
- *3.7. Schedule the time for attaching the source agent to source node, and the sink agent to destination node.*
- *3.8. Schedule the time for attaching the traffic source to the source agent.*
- *3.9. Schedule the time for connecting the source and the sink agents to each other.*
- *3.11. Schedule the starting and stopping times for the traffic source.*
- *3.12. Schedule the detachment of agents from the source and destination nodes.*

4. Get the ending time and session number of the first element in the list and delete it.
5. Until there is no element left in the linked list, do the followings:
- *5.1. Generate a random source.*
- *5.2. Generate a random destination different from the source.*
- *5.3. Make selection whether to use TCP or UDP depending on the given TCP percentage.*
- *5.4. Identify the source and sink agents depending on the selection made in the previous step.*
- *5.5. Randomize the starting time and the duration time.*
- *5.6. Create the next recurrences of the session, and insert them into linked list with their ending times, if their ending times are within the simulation time.*
- *5.7. Schedule the time for attaching the source agent to source node, and the sink agent to destination node.*
- *5.8. Schedule the time for attaching the traffic source to the source agent.*
- *5.9. Schedule the time for connecting the source and the sink agents to each other.*
- *5.11. Schedule the starting and stopping times for the traffic source.*
- *5.12. Schedule the detachment of agents from the source and destination nodes.*
- *5.13. Get the ending time and session number of the first element in the list and delete it.*

Figure 4-1: Algorithm for generating the traces of Telnet, WWW, ftp, and X11.

SMTP traffic occurs when an email is sent to an email list. Since email lists automatically send a newly arrived email to all its members, the generated traffic will be like traffic going from one source to multiple destinations all at the same time. In order to simulate

SMTP traffic, new source and sink agents are created at the number of previously and randomly assigned destinations. All source agents are attached to the same node, which is the source of the session. Sink agents are attached to their corresponding nodes. All source agents are started and stopped at the same time. Thus, the general view of the generated trace is similar to SMTP traffic.

During the generation of traces, we used several extra linked lists for managing garbage collection, and storing destinations (or sinks) of SMTP sessions temporarily.

1. Set up the random variables for generating inter-arrival times and number of sinks of sessions.
2. Create two linked lists (`SMTPList`, `SinksList`), one for maintaining the sessions, one for temporarily storing the sinks of each session.
3. For each session do the followings:
- *3.1. Generate a random source.*
- *3.2. Randomly generate the number of sinks for that session.*
- *3.3. Randomly generate destinations different from the source.*
- *3.4. Randomize the starting time and the duration time.*
- *3.5. Insert the session into linked list with its ending time.*
- *3.6. Make selection whether to use TCP or UDP depending on the given TCP percentage.*
- *3.7. Identify the source and sink agents depending on the selection made in the previous step.*
- *3.8. Schedule the time for attaching the source agent to source node, and the sink agent to destination node.*
- *3.9. Schedule the time for attaching the traffic source to the source agent.*
- *3.10. Schedule the time for connecting the source and the sink agents to each other.*
- *3.11. Schedule the starting and stopping times for the traffic source.*
- *3.12. Schedule the detachment of agents from the source and destination nodes.*

4. Get the ending time and session number of the first element in the list and delete it.
5. Until there is no element left in the linked list, do the followings:
- *5.1. Generate a random source.*
- *5.2. Randomly generate the number of sinks for that session.*
- *5.3. Randomly generate destinations different from the source.*
- *5.4. Randomize the starting time and the duration time.*
- *5.5. Create the next recurrences of the session, and insert them into linked list with their ending times, if their ending times are within the simulation time.*
- *5.6. Make selection whether to use TCP or UDP depending on the given TCP percentage.*
- *5.7. Identify the source and sink agents depending on the selection made in the previous step.*
- *5.8. Schedule the time for attaching the source agent to source node, and the sink agent to destination node.*
- *5.9. Schedule the time for attaching the traffic source to the source agent.*
- *5.10. Schedule the time for connecting the source and the sink agents to each other.*
- *5.11. Schedule the starting and stopping times for the traffic source.*
- *5.12. Schedule the detachment of agents from the source and destination nodes.*
- *5.13. Get the ending time and session number of the first element in the list and delete it.*

Figure 4-2: Algorithm for generating the traces of SMTP.

4.2. Distributions

Several applications in the Internet have not been surely identified to have a specific traffic behavior so far. Although there has been substantial research on that area, there is

no guarantee to have the same traffic behavior in the future. So, only available information about this subject is used in the traffic generator.

Application	Inter-arrival Times of Sessions	Duration Times of Sessions	Data in Sessions
Telnet	Exponential	Log-normal [9]	Pareto [7]
WWW	Exponential	Exponential	Self-Similar [5]
ftp	Exponential [10]	Exponential	Pareto
X11	Exponential [10]	Exponential	Self-Similar [10]
SMTP	Exponential	Log-normal [9]	Pareto

Figure 4-3: Used distributions for sessions of each application.

Figure 4-3 lists the distributions for inter-arrival times and duration times of sessions, and packet arrivals in sessions. Inter-arrival times of Telnet, SMTP, and WWW are assumed to be Exponential. Similarly, duration times of WWW, ftp, and X11 are assumed to be Exponential. Packet arrivals of ftp and SMTP are said to be bursty and not Poisson [10] respectively, so we assumed Pareto is able to capture them. Similarly, packet arrivals of X11 are said to be not Poisson [10], so we assumed self-similar behavior is able to capture it. In fact, these assumptions can be changed in the traffic generator very easily. One can always change the corresponding code of the generator for this purpose.

In ns there is no random variable that is able to generate values with Log-normal distribution. Thus, we wrote a procedure, "`Gaussian`", for generating values with Gaussian, also called "normal", distribution, and made the logarithmic conversions for generating Log-normal distribution. `Gaussian` takes "mean" and "variance" as parameters, and returns a value with Gaussian distribution. In other words, it makes the conversion from uniform distribution, which is available in ns, to Gaussian distribution. $\log_2$ of the given mean and variance is given to `Gaussian`, and 2 to the power of the returning value is calculated, which is the required Log-normal value.

A self-similar traffic generator is also not available in ns. So, we implemented and embedded a new traffic generator, SupFRP, into ns (see Section 4.3). This generator is called from ns for generating self-similar WWW and X11 packet arrivals.

4.3. SupFRP

4.3.1. Using SupFRP in ns

In Section 4.2 we stated that there is a need for self-similar traffic for some applications. In order to generate self-similar traffic we created a new TclObject, and embedded into ns's C++ code (see Appendix B). This new TclObject is called from "Application/Traffic/SupFRP" class in ns Tcl with parameters: `rate_`, `FRPs_`, `packet_size_`, and `hurst_`.

The meaning of each parameter and their default values are listed in Figure 4-4. `rate_` must be given in "Kbits per second". `packet_size_` must be given in "Bytes". `FRPs_` is used for defining the number of Fractal Renewal Processes (FRPs) that are

going to be superposed for the traffic. `hurst_` is used for defining the Hurst parameter, which defines the degree of self-similarity (see Section 4.3.2).

Parameter	Meaning	Default	Range	Type
`rate_`	mean transmission rate	5 Kb	`>0`	Real
`packet_size_`	mean packet size	5 B	`>0`	Integer
`FRPs_`	number of FRPs	5	`>0`	Integer
`hurst_`	degree of self-similarity	0.92	`(0,1)`	Real

Figure 4-4: List of parameters in SupFRP

4.3.2. Analysis of SupFRP

Since generation of fractal traffic requires computationally complex formulas to be calculated, it has been a challenge to find a way of generating fractal traffic in a tractable way. B. K. Ryu proposed an algorithm for generating fractal traffic in his PhD dissertation in 1996, and called it as "SupFRP". [12]

The algorithm makes the traffic generation on a renewal-based method. There are M Fractal Renewal Processes (FRP), superposition of which is the total traffic that is being generated. The algorithm takes only three parameters: mean packet arrival rate (λ), Hurst parameter (H), and the number of FRPs (M). By using these parameters it is possible to generate several kinds of traffic.

λ is calculated by the formula λ=`(1000*rate_)/(8*packet_size_)`, with the assumption that `rate_` will be given in "Kbits per second" and `packet_size_` will be given in "Bytes". This results in "packets/second" as the mean packet arrival rate.

The Hurst parameter, H, is the basic parameter to justify the behavior of the resulting traffic. H can have a value in (0,1), and when it gets closer to 1, the generated traffic will have more fractal behavior, i.e. will be more "long-range dependent" and will have more "slowly decaying variance" property. When H gets closer to 0, the traffic will be more "short-range dependent", and will have "geometrically decaying variance" property.

The parameter M corresponds to the number of traffic sources. M is used to control burstiness of the generated traffic. For fixed λ and H, smaller M gives rise to burstier traffic, therefore greater variance.

In order to understand whether SupFRP is able to generate self-similar traffic we need to look at auto-correlation function (ACF) and index of dispersion for counts (IDC) (see Appendix D). Figure 4-5 and Figure 4-6 represent ACF and IDC of the traffic generated by SupFRP for three different values of Hurst parameter when slot length is equal to 0.01 sec and the following values are given as the parameters to the algorithm:

Mean packet arrival rate (λ): 20,000 packets/second
Number of FRPs (M): 5
Simulation time: 1,300,000 packet arrivals

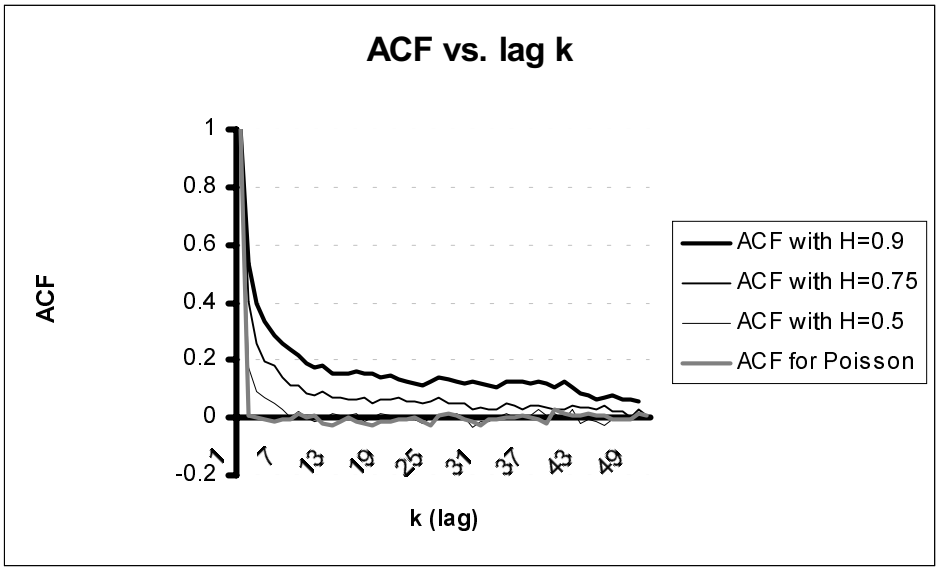

Figure 4-5: ACF of the traffic generated by SupFRP and Poisson.
Obtained by averaging the ACFs of 6 different seeds.

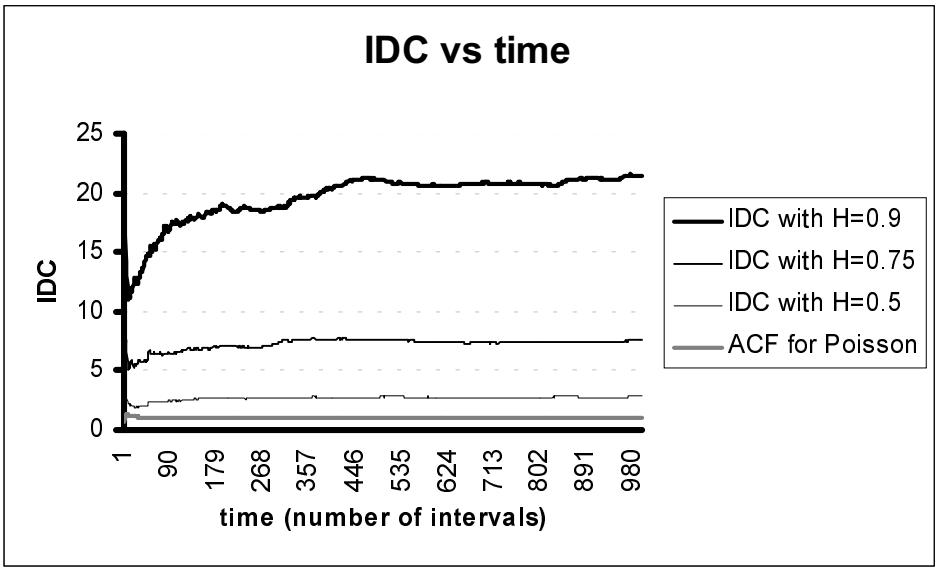

Figure 4-6: IDC of the traffic generated by SupFRP and Poisson.
Obtained by averaging the IDCs of 6 different seeds.

Figure 4-5 (ACF vs. lag k) is obtained by calculating the value of ACF when lag k, which is the number of intervening time slots, increases. We can see that when H is 0.9, the generated traffic is almost scalable (see Appendix D.2.2.3) with respect to ACF. Moreover, ACF is not tending to zero when lag k is increased. So we can also say that ACF is not summable when k goes to infinity, which is the first rule of power-law (see

Appendix D.2.2.3). In addition, when we decrease the Hurst parameter, we can see that ACF is converging to zero in a short amount of time. Thus, we can say that the algorithm is generating a long-range dependent traffic when *H* is close to 1.

Figure 4-6 (IDC vs. time) is obtained by calculating the value of IDC when time increases. Power-law states that fractal traffic has a scalable IDC, which means having "slowly decaying variance" property (see Appendix D.2.2.3). We can see this property for the generated traffic when *H* is 0.9 again. When *H* is 0.9, IDC is converging to a constant in a longer amount of time than the other IDCs obtained for *H*=0.75 and *H*=0.5. When *H* decreases, IDC of the generated traffic converges to a constant in shorter amount of time. Since IDC is calculated by dividing variance with the mean, it is not so hard to figure out that the traffic generated with *H*=0.9 has more "slowly decaying variance" property. Having an IDC converging to a constant in a short amount of time is a behavior of traditional traffic models such as Poisson, MMPP. This property is also defined as "geometrically decaying variance" property.

Finally, we can say that when *H* gets closer to 1, the traffic generated by SupFRP is getting more "long-range dependent", and is having more "slowly decaying variance" property. Since in order to be self-similar at least two of the rules of power-law must be satisfied (see Appendix D.2.2.3), the self-similarity of the generated traffic is getting more when *H* gets closer to 1. Depending on these results, we can use SupFRP as a self-similar traffic generator by just giving a Hurst parameter closer to 1.

In Figure 4-5 and Figure 4-6, we can also see ACF and IDC of Poisson process, which gives us an opportunity of comparing SupFRP and Poisson. We can see that ACF of Poisson is tending to zero earlier than SupFRP, and IDC of Poisson is converging to a constant earlier than SupFRP. Thus, we can easily conclude that SupFRP is generating a traffic with more "long-range dependence" and more "slowly decaying variance" property than Poisson.

5. TESTING RESULTS

Since the aim of this project is to generate sources, destinations, and on-off times of sessions randomly, we tested the generator to make sure that it is generating the required randomization while obeying the given parameters. The other important issue to find out is to identify the minimum simulation time for the stabilized statistical results to be obtained. We ran the generator for from 1 hour to 17 hours of simulation times with the following important parameters given:

```
M_SMTP = M_X11 = M_FTP = M_WWW = M_TELNET =10
MIAT_SMTP = 400
MNB_SMTP = 1000
VAR_MNB_SMTP = 1.0
MR_SMTP = 100
MIAT_X11 = MIAT_FTP = MIAT_WWW = MIAT_TELNET = 150
MDT_X11 = MDT_FTP = MDT_WWW = MDT_TELNET = 10
VAR_TELNET = 1.0
```

Figure 5-1 represents the number of scheduled sessions during the simulation varying according to the simulation time in tabular form. Since the inter-arrival time of SMTP is 400sec, which is greater than the others' inter-arrival times, the number of scheduled sessions for SMTP is growing slower. The number of scheduled sessions depends on the mean duration time, the mean inter-arrival time, the mean number of sessions of the application, and the simulation time.

	Number of Sessions				
Simulation Time (hour)	**Telnet**	**WWW**	**ftp**	**SMTP**	**X11**
1	247	243	265	100	246
2	488	490	439	200	447
3	701	660	675	287	675
4	928	884	893	371	902
5	1137	1132	1124	455	1130
6	1331	1354	1328	545	1325
7	1559	1562	1579	643	1572
8	1772	1787	1825	743	1767
9	1997	2019	2019	839	2003
10	2219	2197	2250	910	2239
11	2461	2425	2500	989	2462
12	2689	2696	2699	1072	2690
13	2875	2912	2919	1183	2912
14	3107	3097	3198	1258	3143
15	3318	3306	3425	1336	3393
16	3532	3593	3732	1419	3586
17	3688	3795	3929	1480	3732

Figure 5-1: Number of scheduled sessions for each application during the simulation.

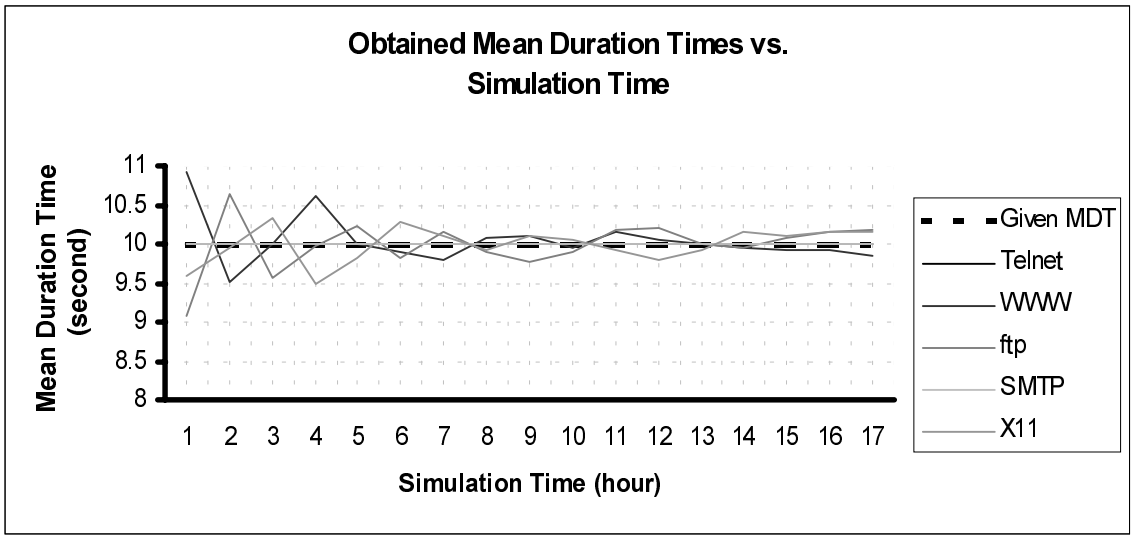

Figure 5-2: Comparison of obtained mean duration times and the required (given) mean duration time, while simulation time is varying.

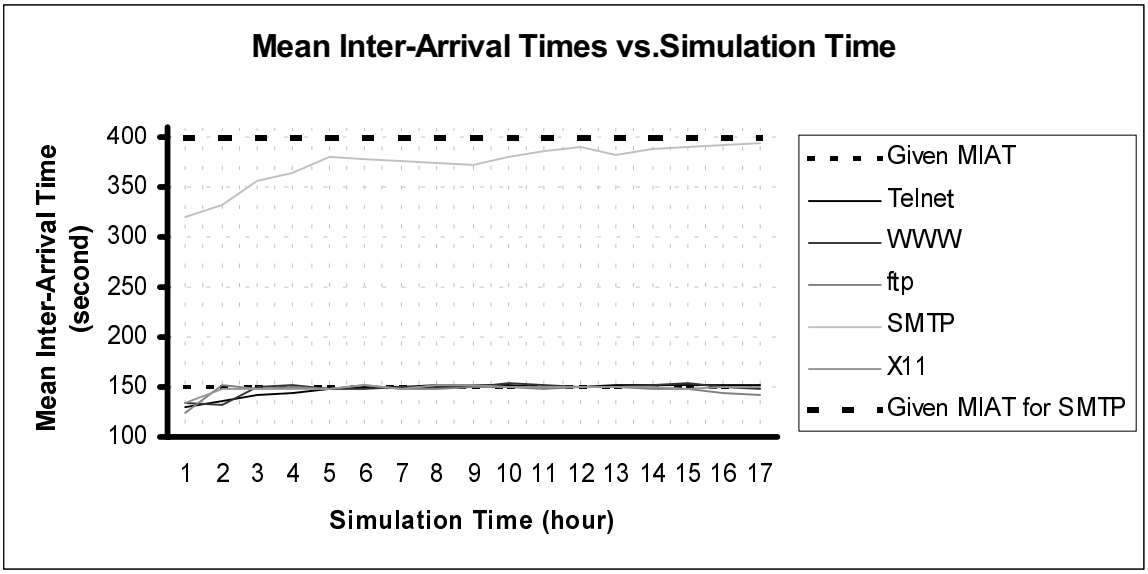

Figure 5-3: Comparison of obtained mean inter-arrival times and the required (given) mean inter-arrival times, while simulation time is varying.

We can write the formula for calculating the expected number of scheduled sessions for an application as follows:

$$\text{Number scheduled} = \frac{(\text{Mean Number of Sessions}) \bullet (\text{Simulation Time})}{(\text{Mean InterArrival Time}) + (\text{Mean Duration Time})}$$

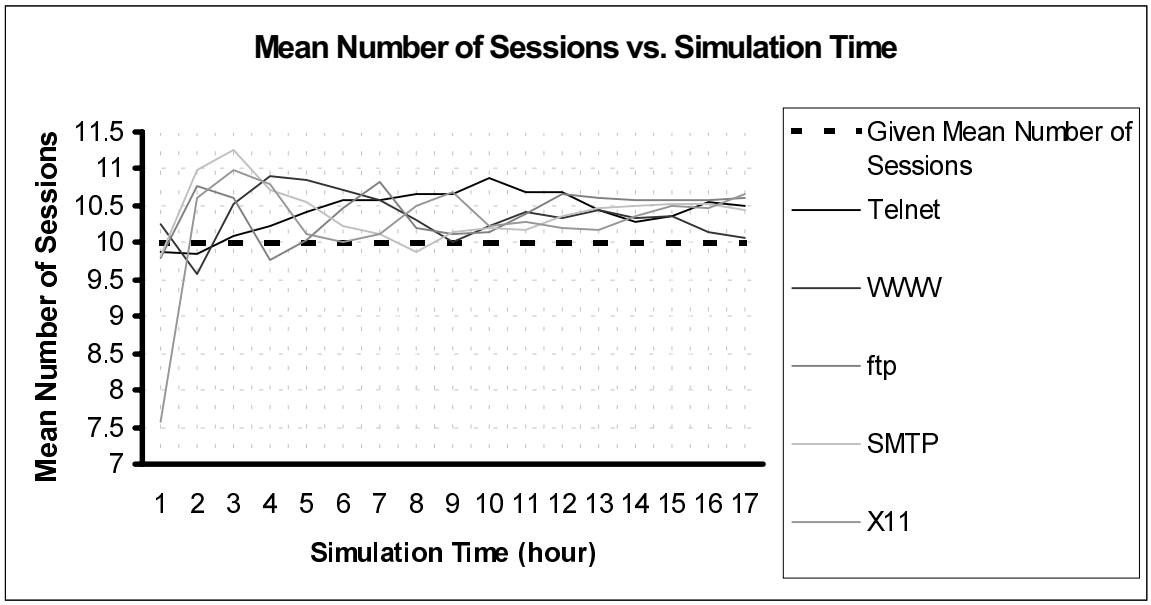

Figure 5-4: Comparison of obtained mean number of sessions and the required (given) mean number of sessions, while simulation time is varying.

Figure 5-2 represents the change in the mean duration times of each application when the simulation time is increased. According to the given parameters, the required mean duration time for all applications is 10sec. From Figure 5-2, we can conclude that the mean duration times of all application are stabilized after at least a 5 hours of simulation time, which corresponds to 1000 sessions to be scheduled approximately. The interesting result available in Figure 5-2 is to have the mean duration times of SMTP and Telnet being stabilized even for 1 hour of simulation time. The reason is to have small variances. Parameters affecting the mean duration time of SMTP are `MNB_SMTP`, `VAR_MNB_SMTP`, `MR_SMTP`. At first, by using `MNB_SMTP` and `VAR_MNB_SMTP`, the number of bytes to be transmitted in the SMTP session is identified. Next, the duration time is found by dividing the number of bytes by `MR_SMTP`. So, the `VAR_MNB_SMTP` is a very important parameter for the duration time of the SMTP sessions. Since `VAR_MNB_SMTP`, used for generating the results in Figure 5-2, is very small compared to `MNB_SMTP`, the mean duration time of SMTP sessions is stabilized very early. Similar case is also available for Telnet sessions' mean duration time. However, one must take care of the variance, if the variance is large compared to the mean.

In Figure 5-3, which represents the changes in the mean inter-arrival times, we can see the importance of the number of scheduled sessions. The mean inter-arrival time for SMTP stabilizes after 12 hours of simulation time, whereas the other applications stabilize after 5 hours of simulation time. From Figure 5-1, we can see that both of them correspond to the same number of scheduled sessions, approximately 1000.

Figure 5-4 represents the changes in the mean number of sessions for the applications. We can see the same difference between SMTP and the other applications as in the mean inter-arrival times. All of the applications except SMTP have mean number of sessions

between 10 and 11 after 5 hours of simulation time. Thus SMTP is stabilizing after 12 hours of simulation time again, whereas the others are stabilizing after 5 hours of simulation time.

We can conclude that it is not safe to run the traffic generator for fewer than 1000 sessions for each application. However, if there is smaller error tolerance, then more detailed analysis of the results must be done in order to find out the minimum number of sessions each application should be run.

6. FUTURE WORK

The first thing that can be done for improvement of this traffic generator is to implement the approximations in SupFRP, proposed by B. K. Ryu. He claims about the computational complexity of the pure algorithm of SupFRP, and proposes some approximation techniques in his PhD dissertation. [12] However, one can also implement a better self-similar traffic generator, which is more tractable and flexible.

Another beneficial thing to do is to analyze the behavior of aggregate traffic generated by the traffic generator. This can be performed with tracking packet arrivals on some particular links in the topology. This work will give a chance of comparing the generated traffic and the real traffic.

Appendix A. Tcl SCRIPT OF THE TRAFFIC GENERATOR

Tcl code of the traffic generator is divided into two files: paramforgenerator.tcl and trafgenerator.tcl. paramforgenerator.tcl includes the tcl code for adjusting the parameters to the traffic generator. trafgenerator.tcl is the traffic generator itself.

Appendix A.1. Tcl Script of Parameters

"paramforgenerator.tcl"

```
##############################################################################
# This file contains directions for the necessary adjustments for using the #
# traffic generator. The traffic generator is available as tcl code in      #
# "trafgenerator.tcl" file.                                                 #
##############################################################################

### 1 ###
# At first, if you want to define your own simulator set up options, then
# you must change the tcl code in "SETTING UP THE SIMULATION" part of
# this file.

### 2 ###
# Next, necessary tcl code for defining the topology must also be added to the
# end of this file.

### 3 ###
# Before using trafgenerator.tcl, this file (paramforgenerator.tcl) must be set
# up according to the explanations for each parameter, and merged to the
# beginning of trafgenerator.tcl.
# Some of the parameters must be defined, otherwise the generator will give
# error. However, the rest have default values. The ones without a default value
# are marked with "NEEDS VALUE". A brief info about the units of the parameters
# is also added. For more detailed information about the parameters, look at
# "project.ps" file.

### 4 ###
# After setting up the parameters, you can run the generator in the ns
# environment.

################# SETTING UP THE SIMULATION ################
set ns [new Simulator]

set nf [open /space/tmp/out.nam w]
$ns namtrace-all $nf
$ns expand-port-field-bits 23

proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec nam /space/tmp/out.nam &
        exit 0
   }

# NEEDS VALUE # integer
# Number of nodes in the topology. You must number your nodes from 1 to $NODES.
set NODES 32

# NEEDS VALUE # real
# percetage of TCP traffic.
set PER_TCP 0.8

# NEEDS VALUE # real
# smallest possible delay (smallest possible value is 0.000000000000001=10^-15)
set SPD 0.1
```

```
############## SMTP PARAMETERS ###############
# NEEDS VALUE # integer
# mean # of SMTP sessions
set M_SMTP 10

# NEEDS VALUE # integer
# Mean # of sinks per each SMTP session
set MNS_SMTP 5

# NEEDS VALUE # real
# Mean Inter-Arrival Time for SMTP (must be in seconds)
set MIAT_SMTP 400

# NEEDS VALUE # real
# Mean Number of Bytes for each SMTP session
set MNB_SMTP 1000

# NEEDS VALUE # real
# Variance for the Mean Number of Bytes for each SMTP session
set VAR_MNB_SMTP 100.0

# NEEDS VALUE # integer
# Mean Packet Size for SMTP (must be in bytes)
set MPS_SMTP 500

# NEEDS VALUE # real
# Mean Burst Size for SMTP (must be in seconds)
set MBS_SMTP 0.05

# Mean Idle Time for SMTP (must be in seconds or with ns' units)
set MIT_SMTP 10ms

# NEEDS VALUE # real
# Mean Rate of SMTP sessions (don't use ns' units like k, b, B, MB)
set MR_SMTP 100

############## X11 PARAMETERS ###############
# NEEDS VALUE # integer
# mean # of X11 sessions
set M_X11 10

# NEEDS VALUE # real
# Mean Inter-Arrival Time for X11 (must be in seconds)
set MIAT_X11 150

# NEEDS VALUE # real
# Mean Time for duration of each X11 connection (must be in seconds)
set MDT_X11 10

# NEEDS VALUE # real
# Mean Rate of X11 sessions (must be in Kbits)
set MR_X11 0.1

# NEEDS VALUE # integer
# Mean Packet Size for X11 sessions (must be in bytes)
set MPS_X11 500

# Hurst paremeter for X11 sessions (must be in (0,1))
set H_X11 0.92

############## FTP PARAMETERS ###############
# NEEDS VALUE # integer
# mean # of ftp sessions
set M_FTP 10

# NEEDS VALUE # real
# Mean Inter-Arrival Time for ftp (must be in seconds)
set MIAT_FTP 150

# NEEDS VALUE # real
# Mean Time for duration of each ftp connection (must be in seconds)
```

```
set MDT_FTP 10

# NEEDS VALUE # integer
# Mean Packet Size for FTP (must be in bytes)
set MPS_FTP 500

# NEEDS VALUE # real
# Mean Burst Size for FTP (must be in seconds)
set MBS_FTP 0.05

# Mean Idle Time for FTP (must be in seconds or with ns' units)
set MIT_FTP 10ms

# NEEDS VALUE # real
# Mean Rate of FTP sessions (must be in bytes)
set MR_FTP 100

############## WWW PARAMETERS ###############
# NEEDS VALUE # integer
# mean # of WWW sessions
set M_WWW 10

# NEEDS VALUE # real
# Mean Inter-Arrival Time for WWW (must be in seconds)
set MIAT_WWW 150

# NEEDS VALUE # real
# Mean Time for duration of each WWW connection (must be in seconds)
set MDT_WWW 10

# NEEDS VALUE # real
# Mean Rate of WWW sessions (must be in Kbits)
set MR_WWW 0.1

# NEEDS VALUE # integer
# Mean Packet Size for WWW sessions (must be in bytes)
set MPS_WWW 500

# Hurst paremeter for WWW sessions (must be in (0,1))
set H_WWW 0.92

############## TELNET PARAMETERS ###############
# NEEDS VALUE # integer
# mean # of Telnet sessions
set M_TELNET 10

# NEEDS VALUE # real
# Mean Inter-Arrival Time for Telnet (must be in seconds)
set MIAT_TELNET 150

# NEEDS VALUE # real
# Mean Time for duration of each Telnet connection (must be in seconds)
set MDT_TELNET 10

# NEEDS VALUE # real
# Variance for duration of each Telnet connection (must be in seconds)
set VAR_TELNET 2.5

# Mean Packet Size for Telnet (must be in Bytes)
set MPS_TELNET 500

# Mean Burst Size for Telnet(must be in seconds)
set MBS_TELNET 0.05

# Mean Idle Time for Telnet (must be in seconds or with ns' units)
set MIT_TELNET 10ms

# Mean Rate of TELNET sessions (must be in Bytes)
set MR_TELNET 100

# NEEDS VALUE # real
```

```
# starting time of the simulation
set START_TIME 0.0

# NEEDS VALUE # real
# ending time of the simulation
set STOP_TIME 10800.0
```

Appendix A.2. Tcl Script of the Traffic Generator

"trafgenerator.tcl"

```
##############################################################################
# This file contains tcl code of a traffic generator which randomizes the   #
# source, destination, and on-off times of the sessions according to the    #
# given parameters. Directions for setting up the parameters are available  #
# in paramforgenerator.tcl file.                                             #
##############################################################################

######################## TRAFFIC GENERATOR ####################
# percetage of UDP
set PER_UDP [expr 1 - $PER_TCP]

# RV for constructing Gaussian distribution...
set rvuGauss [new RandomVariable/Uniform]
$rvuGauss set min_ 0
$rvuGauss set max_ 1

# returns a value with a Gaussian distribution of "mean" and "variance"
proc Gaussian { mean variance } {
        global rvuGauss
        set p1 [$rvuGauss value]
        set p2 [$rvuGauss value]
        set gau [expr $mean + $variance * sqrt(-2.0*log($p1)) * cos($p2*2.0*3.1415927)]
        return $gau
}


# RV for creating the next recurrences of each session
set rvuSession [new RandomVariable/Uniform]
$rvuSession set min_ 0
$rvuSession set max_ 1

# RV for TCP/UDP selection
set rvuTCPUDP [new RandomVariable/Uniform]
$rvuTCPUDP set min_ 0
$rvuTCPUDP set max_ 1

# RV for generating sources
set rvuSource [new RandomVariable/Uniform]
$rvuSource set min_ 1
$rvuSource set max_ $NODES

# RV for generating sinks
set rvuSink [new RandomVariable/Uniform]
$rvuSink set min_ 1
$rvuSink set max_ $NODES

#######################################################################
# Generating SMTP traffic...
puts "Generating SMTP traffic..."

set SMTPCOUNTER 0
set SINKCOUNTER 0

# RV for generating MIAT for SMTP sessions (Poisson SMTP arrivals)
set rveMIATSMTP [new RandomVariable/Exponential]
$rveMIATSMTP set avg_ $MIAT_SMTP

# RV for generating Number of Sinks of SMTP sessions
set rveSINKSMTP [new RandomVariable/Exponential]
$rveSINKSMTP set avg_ $MNS_SMTP

set TotalSMTPTCP 0
set TotalSMTPUDP 0
set TotalSMTPTCPAgent 0
set TotalSMTPUDPAgent 0
```

```
set CurrSMTPTCP(0) 0
set CurrSMTPUDP(0) 0
set CurrSMTPTCPAgent(0) 0
set CurrSMTPUDPAgent(0) 0

set SMTPList [new Application/Traffic/LList]
set SinksList [new Application/Traffic/LList]
set SMTPFreeTCPList [new Application/Traffic/LList]
set SMTPFreeUDPList [new Application/Traffic/LList]
set SMTPFreeTCPAgentList [new Application/Traffic/LList]
set SMTPFreeUDPAgentList [new Application/Traffic/LList]

set SMTPsrc(0) [new Agent/TCP]
set SMTPsink(0) [new Agent/TCPSink]

set SMTPTCPsrc(0) [new Agent/TCP]
set SMTPTCPsink(0) [new Agent/TCPSink]
set SMTPUDPsrc(0) [new Agent/UDP]
set SMTPUDPsink(0) [new Agent/UDP]
set SMTPTCPAgent(0) [new Application/FTP]
set SMTPUDPAgent(0) [new Application/Traffic/Pareto]

## Finds an idle agent, an idle source, and an idle sink or creates new ones
## For TCP session that is going to be generated
proc SMTPProc1TCP { SINKCOUNTER } {
 global CurrSMTPTCP SMTPFreeTCPList TotalSMTPTCP SMTPsrc SMTPsink SMTPTCPsrc SMTPTCPsink
 global CurrSMTPTCPAgent SMTPFreeTCPAgentList TotalSMTPTCPAgent SMTPTCPAgent
 global MPS_SMTP MBS_SMTP MR_SMTP

 set CurrSMTPTCPAgent($SINKCOUNTER) [$SMTPFreeTCPAgentList NoOfFirst]
 $SMTPFreeTCPAgentList DeleteFirst
 if {$CurrSMTPTCPAgent($SINKCOUNTER) == 0} {
  incr TotalSMTPTCPAgent
  set CurrSMTPTCPAgent($SINKCOUNTER) $TotalSMTPTCPAgent
  set SMTPTCPAgent($CurrSMTPTCPAgent($SINKCOUNTER)) [new Application/FTP]
 }

 set CurrSMTPTCP($SINKCOUNTER) [$SMTPFreeTCPList NoOfFirst]
 $SMTPFreeTCPList DeleteFirst
 if {$CurrSMTPTCP($SINKCOUNTER) == 0} {
  incr TotalSMTPTCP
  set CurrSMTPTCP($SINKCOUNTER) $TotalSMTPTCP
  set SMTPTCPsrc($CurrSMTPTCP($SINKCOUNTER)) [new Agent/TCP]
  set SMTPTCPsink($CurrSMTPTCP($SINKCOUNTER)) [new Agent/TCPSink]
 }
 set SMTPsrc($SINKCOUNTER) $SMTPTCPsrc($CurrSMTPTCP($SINKCOUNTER))
 set SMTPsink($SINKCOUNTER) $SMTPTCPsink($CurrSMTPTCP($SINKCOUNTER))

 $SMTPsrc($SINKCOUNTER) set packetSize_ $MPS_SMTP
 $SMTPsrc($SINKCOUNTER) set maxburst_ [expr round($MBS_SMTP*$MR_SMTP/$MPS_SMTP)]
 ## Mean idle time is ignored for TCP Telnet sessions...
}

## Finds an idle agent, an idle source, and an idle sink or creates new ones
## For UDP session that is going to be generated
proc SMTPProc1UDP {SINKCOUNTER} {
 global CurrSMTPUDP SMTPFreeUDPList TotalSMTPUDP SMTPUDPsrc SMTPUDPsink SMTPsrc SMTPsink
 global CurrSMTPUDPAgent SMTPFreeUDPAgentList TotalSMTPUDPAgent SMTPUDPAgent
 global MPS_SMTP MBS_SMTP MIT_SMTP MR_SMTP

 set CurrSMTPUDPAgent($SINKCOUNTER) [$SMTPFreeUDPAgentList NoOfFirst]
 $SMTPFreeUDPAgentList DeleteFirst
 if {$CurrSMTPUDPAgent($SINKCOUNTER) == 0} {
  incr TotalSMTPUDPAgent
  set CurrSMTPUDPAgent($SINKCOUNTER) $TotalSMTPUDPAgent
  set SMTPUDPAgent($CurrSMTPUDPAgent($SINKCOUNTER)) [new Application/Traffic/Pareto]
  $SMTPUDPAgent($CurrSMTPUDPAgent($SINKCOUNTER)) set packet_size_ $MPS_SMTP
  $SMTPUDPAgent($CurrSMTPUDPAgent($SINKCOUNTER)) set burst_size_ $MBS_SMTP
  $SMTPUDPAgent($CurrSMTPUDPAgent($SINKCOUNTER)) set idle_time_ $MIT_SMTP
  $SMTPUDPAgent($CurrSMTPUDPAgent($SINKCOUNTER)) set rate_ $MR_SMTP
```

```
  $SMTPUDPAgent($CurrSMTPUDPAgent($SINKCOUNTER)) set shape_ 1.5
 }

 set CurrSMTPUDP($SINKCOUNTER) [$SMTPFreeUDPList NoOfFirst]
 $SMTPFreeUDPList DeleteFirst
 if {$CurrSMTPUDP($SINKCOUNTER) == 0} {
  incr TotalSMTPUDP
  set CurrSMTPUDP($SINKCOUNTER) $TotalSMTPUDP
  set SMTPUDPsrc($CurrSMTPUDP($SINKCOUNTER)) [new Agent/UDP]
  set SMTPUDPsink($CurrSMTPUDP($SINKCOUNTER)) [new Agent/UDP]
 }
 set SMTPsrc($SINKCOUNTER) $SMTPUDPsrc($CurrSMTPUDP($SINKCOUNTER))
 set SMTPsink($SINKCOUNTER) $SMTPUDPsink($CurrSMTPUDP($SINKCOUNTER))
}

## Sets up the connection between previously identified source & destination
## For TCP session that is going to be generated
proc SMTPProc2TCP {SINKCOUNTER node1 node2} {
 global SMTPsrc SMTPsink ns SMTPTCPAgent CurrSMTPTCPAgent n

 $ns attach-agent $n($node1) $SMTPsrc($SINKCOUNTER)
 $ns attach-agent $n($node2) $SMTPsink($SINKCOUNTER)
 $ns connect $SMTPsrc($SINKCOUNTER) $SMTPsink($SINKCOUNTER)
 $SMTPTCPAgent($CurrSMTPTCPAgent($SINKCOUNTER)) attach-agent $SMTPsrc($SINKCOUNTER)
}

## Sets up the connection between previously identified source & destination
## For UDP session that is going to be generated
proc SMTPProc2UDP {SINKCOUNTER node1 node2} {
 global SMTPsrc SMTPsink ns SMTPUDPAgent CurrSMTPUDPAgent n

 $ns attach-agent $n($node1) $SMTPsrc($SINKCOUNTER)
 $ns attach-agent $n($node2) $SMTPsink($SINKCOUNTER)
 $ns connect $SMTPsrc($SINKCOUNTER) $SMTPsink($SINKCOUNTER)
 $SMTPUDPAgent($CurrSMTPUDPAgent($SINKCOUNTER)) attach-agent $SMTPsrc($SINKCOUNTER)
}

## Starts the TCP traffic source
proc SMTPProc3TCP {SINKCOUNTER BytesToSend} {
 global SMTPsrc

 $SMTPsrc($SINKCOUNTER) send $BytesToSend
}

## Starts the UDP traffic source
proc SMTPProc3UDP {SINKCOUNTER} {
 global SMTPUDPAgent CurrSMTPUDPAgent

 $SMTPUDPAgent($CurrSMTPUDPAgent($SINKCOUNTER)) start
}

## Stops the TCP traffic source
proc SMTPProc4TCP {SINKCOUNTER} {
 global SMTPTCPAgent CurrSMTPTCPAgent

 $SMTPTCPAgent($CurrSMTPTCPAgent($SINKCOUNTER)) stop
}

## Stops the UDP traffic source
proc SMTPProc4UDP {SINKCOUNTER} {
 global SMTPUDPAgent CurrSMTPUDPAgent

 $SMTPUDPAgent($CurrSMTPUDPAgent($SINKCOUNTER)) stop
}

## Detaches source & sink agents from their corrsponding nodes, and hanles free list
operations
## For TCP connection that was generated
proc SMTPProc5TCP {SINKCOUNTER node1 node2} {
 global ns SMTPsrc SMTPsink SMTPFreeTCPAgentList CurrSMTPTCPAgent n
 global SMTPFreeTCPList CurrSMTPTCP
```

```
 $SMTPFreeTCPList Insert $CurrSMTPTCP($SINKCOUNTER) $CurrSMTPTCP($SINKCOUNTER)

 $ns detach-agent $n($node1) $SMTPsrc($SINKCOUNTER)
 $ns detach-agent $n($node2) $SMTPsink($SINKCOUNTER)
 $SMTPFreeTCPAgentList Insert $CurrSMTPTCPAgent($SINKCOUNTER)
$CurrSMTPTCPAgent($SINKCOUNTER)
}

## Detaches source & sink agents from their corrsponding nodes, and hanles free list
operations
## For UDP connection that was generated
proc SMTPProc5UDP {SINKCOUNTER node1 node2} {
 global ns SMTPsrc SMTPsink SMTPFreeUDPAgentList CurrSMTPUDPAgent n
 global SMTPFreeUDPList CurrSMTPUDP

 $SMTPFreeUDPList Insert $CurrSMTPUDP($SINKCOUNTER) $CurrSMTPUDP($SINKCOUNTER)

 $ns detach-agent $n($node1) $SMTPsrc($SINKCOUNTER)
 $ns detach-agent $n($node2) $SMTPsink($SINKCOUNTER)
 $SMTPFreeUDPAgentList Insert $CurrSMTPUDPAgent($SINKCOUNTER)
$CurrSMTPUDPAgent($SINKCOUNTER)
}

# Insert SMTP sessions into list
for {set i 1} {$i <= $M_SMTP} {incr i} {
 set node1 [expr round([$rvuSource value])]
 set SinkCount [expr round([$rveSINKSMTP value])]
 while {$SinkCount <= 0 || $SinkCount >= $NODES} {
  set SinkCount [expr round([$rveSINKSMTP value])]
 }

 for {set k 1} {$k <= $SinkCount} {incr k} {
  set node2 [expr round([$rvuSink value])]
  while { $node1 == $node2 } {
   set node2 [expr round([$rvuSink value])]
  }
  $SinksList Insert $node2 $node2
 }

 set IAT [$rveMIATSMTP value]
 while {$IAT < 4.0*$SPD} {
  set IAT [$rveMIATSMTP value]
 }

 set Starting_Time [expr $START_TIME + $IAT]
 set Duration_Time [expr (pow(2.0, ["Gaussian" log([expr $MNB_SMTP-300])/log(2.0)
log($VAR_MNB_SMTP)/log(2.0)]) + 300) / $MR_SMTP]
 set lag [expr $Starting_Time - $START_TIME]

 if {[$rvuTCPUDP value] <= $PER_TCP} {
  set node2 [$SinksList NoOfFirst]
  while {$node2 != 0} {
   $ns at [expr $Starting_Time-0.5*$lag] "SMTPProc1TCP $SINKCOUNTER"
   $ns at [expr $Starting_Time-0.25*$lag] "SMTPProc2TCP $SINKCOUNTER $node1 $node2"
   $ns at $Starting_Time "SMTPProc3TCP $SINKCOUNTER [expr $MR_TELNET*$Duration_Time]"
   $ns at [expr $Starting_Time+$Duration_Time] "SMTPProc4TCP $SINKCOUNTER"
   $ns at [expr $Starting_Time+$Duration_Time+$SPD] "SMTPProc5TCP $SINKCOUNTER $node1
$node2"

   $SinksList DeleteFirst
   set node2 [$SinksList NoOfFirst]
   incr SINKCOUNTER
 }
 } else {
  set node2 [$SinksList NoOfFirst]
  while {$node2 != 0} {
   $ns at [expr $Starting_Time-0.5*$lag] "SMTPProc1UDP $SINKCOUNTER"
   $ns at [expr $Starting_Time-0.25*$lag] "SMTPProc2UDP $SINKCOUNTER $node1 $node2"
   $ns at $Starting_Time "SMTPProc3UDP $SINKCOUNTER"
   $ns at [expr $Starting_Time+$Duration_Time] "SMTPProc4UDP $SINKCOUNTER"
```

```
   $ns at [expr $Starting_Time+$Duration_Time+$SPD] "SMTPProc5UDP $SINKCOUNTER $node1
$node2"

   $SinksList DeleteFirst
   set node2 [$SinksList NoOfFirst]
   incr SINKCOUNTER
  }
 }

 $SMTPList Insert [expr $Starting_Time+$Duration_Time] 1

 incr SMTPCOUNTER
}

set Ending_Session [$SMTPList TimeOfFirst]
set Session_No [$SMTPList NoOfFirst]
$SMTPList DeleteFirst

while {$Session_No != 0} {
 set node1 [expr round([$rvuSource value])]
 set SinkCount [expr round([$rveSINKSMTP value])]
 while {$SinkCount <= 0 || $SinkCount >= $NODES} {
  set SinkCount [expr round([$rveSINKSMTP value])]
 }

 for {set k 1} {$k <= $SinkCount} {incr k} {
  set node2 [expr round([$rvuSink value])]
  while { $node1 == $node2 } {
   set node2 [expr round([$rvuSink value])]
  }
  $SinksList Insert $node2 $node2
 }
 set IAT [$rveMIATSMTP value]
 while {$IAT < 4.0*$SPD} {
  set IAT [$rveMIATSMTP value]
 }

 set Starting_Time [expr $Ending_Session + $IAT]
 set Duration_Time [expr (pow(2.0,["Gaussian" log([expr $MNB_SMTP-300])/log(2.0)
log($VAR_MNB_SMTP)/log(2.0)])+300) / $MR_SMTP]
 set lag [expr $Starting_Time - $Ending_Session]

 if {[$rvuTCPUDP value] <= $PER_TCP} {
  set node2 [$SinksList NoOfFirst]
  while {$node2 != 0} {
   $ns at [expr $Starting_Time-0.5*$lag] "SMTPProc1TCP $SINKCOUNTER"
   $ns at [expr $Starting_Time-0.25*$lag] "SMTPProc2TCP $SINKCOUNTER $node1 $node2"
   $ns at $Starting_Time "SMTPProc3TCP $SINKCOUNTER [expr $MR_TELNET*$Duration_Time]"
   $ns at [expr $Starting_Time+$Duration_Time] "SMTPProc4TCP $SINKCOUNTER"
   $ns at [expr $Starting_Time+$Duration_Time+$SPD] "SMTPProc5TCP $SINKCOUNTER $node1
$node2"

   $SinksList DeleteFirst
   set node2 [$SinksList NoOfFirst]
   incr SINKCOUNTER
  }
 } else {
  set node2 [$SinksList NoOfFirst]
  while {$node2 != 0} {
   $ns at [expr $Starting_Time-0.5*$lag] "SMTPProc1UDP $SINKCOUNTER"
   $ns at [expr $Starting_Time-0.25*$lag] "SMTPProc2UDP $SINKCOUNTER $node1 $node2"
   $ns at $Starting_Time "SMTPProc3UDP $SINKCOUNTER"
   $ns at [expr $Starting_Time+$Duration_Time] "SMTPProc4UDP $SINKCOUNTER"
   $ns at [expr $Starting_Time+$Duration_Time+$SPD] "SMTPProc5UDP $SINKCOUNTER $node1
$node2"

   $SinksList DeleteFirst
   set node2 [$SinksList NoOfFirst]
   incr SINKCOUNTER
  }
 }
```

```
 if {[expr $Starting_Time + $Duration_Time] <= $STOP_TIME} {
  set CurrCount [$SMTPList Count]
  set P1 [expr 1.0-$CurrCount/(2.0*$M_SMTP)]
  set P2 0.5
  if {[$rvuSession value]<=$P1} {$SMTPList Insert [expr $Starting_Time+$Duration_Time] 1}
  if {[$rvuSession value]<=$P2} {$SMTPList Insert [expr $Starting_Time+$Duration_Time] 1}
 }

 set Ending_Session [$SMTPList TimeOfFirst]
 set Session_No [$SMTPList NoOfFirst]
 $SMTPList DeleteFirst

 incr SMTPCOUNTER
}

#######################################################################
# Generating Telnet traffic...
puts "Generating Telnet traffic..."

set TELNETCOUNTER 0

# RV for generating MIAT for Telnet sessions (Poisson Telnet arrivals)
set rveMIATTelnet [new RandomVariable/Exponential]
$rveMIATTelnet set avg_ $MIAT_TELNET

# RV for generating mean duration time for Telnet sessions (Poisson Telnet session)
set rveMDTTelnet [new RandomVariable/Exponential]
$rveMDTTelnet set avg_ $MDT_TELNET

set TotalTelnetTCP 0
set TotalTelnetUDP 0
set TotalTelnetTCPAgent 0
set TotalTelnetUDPAgent 0

set CurrTelnetTCP(0) 0
set CurrTelnetUDP(0) 0
set CurrTelnetTCPAgent(0) 0
set CurrTelnetUDPAgent(0) 0

set TelnetList [new Application/Traffic/LList]
set TelnetFreeTCPList [new Application/Traffic/LList]
set TelnetFreeUDPList [new Application/Traffic/LList]
set TelnetFreeTCPAgentList [new Application/Traffic/LList]
set TelnetFreeUDPAgentList [new Application/Traffic/LList]

set Telnetsrc(0) [new Agent/TCP]
set Telnetsink(0) [new Agent/TCPSink]

set TelnetTCPsrc(0) [new Agent/TCP]
set TelnetTCPsink(0) [new Agent/TCPSink]
set TelnetUDPsrc(0) [new Agent/UDP]
set TelnetUDPsink(0) [new Agent/UDP]
set TelnetTCPAgent(0) [new Application/FTP]
set TelnetUDPAgent(0) [new Application/Traffic/Pareto]

## Finds an idle agent, an idle source, and an idle sink or creates new ones
## For TCP session that is going to be generated
proc TelnetProc1TCP { TELNETCOUNTER } {
 global CurrTelnetTCP TelnetFreeTCPList TotalTelnetTCP Telnetsrc Telnetsink TelnetTCPsrc
TelnetTCPsink
 global CurrTelnetTCPAgent TelnetFreeTCPAgentList TotalTelnetTCPAgent TelnetTCPAgent
 global MPS_TELNET MBS_TELNET MR_TELNET

 set CurrTelnetTCPAgent($TELNETCOUNTER) [$TelnetFreeTCPAgentList NoOfFirst]
 $TelnetFreeTCPAgentList DeleteFirst
 if {$CurrTelnetTCPAgent($TELNETCOUNTER) == 0} {
  incr TotalTelnetTCPAgent
  set CurrTelnetTCPAgent($TELNETCOUNTER) $TotalTelnetTCPAgent
  set TelnetTCPAgent($CurrTelnetTCPAgent($TELNETCOUNTER)) [new Application/FTP]
 }
```

```
 set CurrTelnetTCP($TELNETCOUNTER) [$TelnetFreeTCPList NoOfFirst]
 $TelnetFreeTCPList DeleteFirst
 if {$CurrTelnetTCP($TELNETCOUNTER) == 0} {
  incr TotalTelnetTCP
  set CurrTelnetTCP($TELNETCOUNTER) $TotalTelnetTCP
  set TelnetTCPsrc($CurrTelnetTCP($TELNETCOUNTER)) [new Agent/TCP]
  set TelnetTCPsink($CurrTelnetTCP($TELNETCOUNTER)) [new Agent/TCPSink]
 }
 set Telnetsrc($TELNETCOUNTER) $TelnetTCPsrc($CurrTelnetTCP($TELNETCOUNTER))
 set Telnetsink($TELNETCOUNTER) $TelnetTCPsink($CurrTelnetTCP($TELNETCOUNTER))

 $Telnetsrc($TELNETCOUNTER) set packetSize_ $MPS_TELNET
 $Telnetsrc($TELNETCOUNTER) set maxburst_ [expr
round($MBS_TELNET*$MR_TELNET/$MPS_TELNET)]
 ## Mean idle time is ignored for TCP Telnet sessions...
}

## Finds an idle agent, an idle source, and an idle sink or creating new ones
## For UDP session that is going to be generated
proc TelnetProc1UDP {TELNETCOUNTER} {
 global CurrTelnetUDP TelnetFreeUDPList TotalTelnetUDP TelnetUDPsrc TelnetUDPsink
Telnetsrc Telnetsink
 global CurrTelnetUDPAgent TelnetFreeUDPAgentList TotalTelnetUDPAgent TelnetUDPAgent
 global MPS_TELNET MBS_TELNET MIT_TELNET MR_TELNET

 set CurrTelnetUDPAgent($TELNETCOUNTER) [$TelnetFreeUDPAgentList NoOfFirst]
 $TelnetFreeUDPAgentList DeleteFirst
 if {$CurrTelnetUDPAgent($TELNETCOUNTER) == 0} {
  incr TotalTelnetUDPAgent
  set CurrTelnetUDPAgent($TELNETCOUNTER) $TotalTelnetUDPAgent
  set TelnetUDPAgent($CurrTelnetUDPAgent($TELNETCOUNTER)) [new
Application/Traffic/Pareto]
  $TelnetUDPAgent($CurrTelnetUDPAgent($TELNETCOUNTER)) set packet_size_ $MPS_TELNET
  $TelnetUDPAgent($CurrTelnetUDPAgent($TELNETCOUNTER)) set burst_size_ $MBS_TELNET
  $TelnetUDPAgent($CurrTelnetUDPAgent($TELNETCOUNTER)) set idle_time_ $MIT_TELNET
  $TelnetUDPAgent($CurrTelnetUDPAgent($TELNETCOUNTER)) set rate_ $MR_TELNET
  $TelnetUDPAgent($CurrTelnetUDPAgent($TELNETCOUNTER)) set shape_ 1.5
 }

 set CurrTelnetUDP($TELNETCOUNTER) [$TelnetFreeUDPList NoOfFirst]
 $TelnetFreeUDPList DeleteFirst
 if {$CurrTelnetUDP($TELNETCOUNTER) == 0} {
  incr TotalTelnetUDP
  set CurrTelnetUDP($TELNETCOUNTER) $TotalTelnetUDP
  set TelnetUDPsrc($CurrTelnetUDP($TELNETCOUNTER)) [new Agent/UDP]
  set TelnetUDPsink($CurrTelnetUDP($TELNETCOUNTER)) [new Agent/UDP]
 }
 set Telnetsrc($TELNETCOUNTER) $TelnetUDPsrc($CurrTelnetUDP($TELNETCOUNTER))
 set Telnetsink($TELNETCOUNTER) $TelnetUDPsink($CurrTelnetUDP($TELNETCOUNTER))
}

## Sets up the connection between previously identified source & destination
## For TCP session that is going to be generated
proc TelnetProc2TCP {TELNETCOUNTER node1 node2} {
 global Telnetsrc Telnetsink ns TelnetTCPAgent CurrTelnetTCPAgent n

 $ns attach-agent $n($node1) $Telnetsrc($TELNETCOUNTER)
 $ns attach-agent $n($node2) $Telnetsink($TELNETCOUNTER)
 $ns connect $Telnetsrc($TELNETCOUNTER) $Telnetsink($TELNETCOUNTER)
 $TelnetTCPAgent($CurrTelnetTCPAgent($TELNETCOUNTER)) attach-agent
$Telnetsrc($TELNETCOUNTER)
}

## Sets up the connection between previously identified source & destination
## For UDP session that is going to be generated
proc TelnetProc2UDP {TELNETCOUNTER node1 node2} {
 global Telnetsrc Telnetsink ns TelnetUDPAgent CurrTelnetUDPAgent n

 $ns attach-agent $n($node1) $Telnetsrc($TELNETCOUNTER)
 $ns attach-agent $n($node2) $Telnetsink($TELNETCOUNTER)
```

```
 $ns connect $Telnetsrc($TELNETCOUNTER) $Telnetsink($TELNETCOUNTER)
 $TelnetUDPAgent($CurrTelnetUDPAgent($TELNETCOUNTER)) attach-agent
$Telnetsrc($TELNETCOUNTER)
}

## Starts the TCP traffic source
proc TelnetProc3TCP {TELNETCOUNTER BytesToSend} {
 global Telnetsrc

 $Telnetsrc($TELNETCOUNTER) send $BytesToSend
}

## Starts the UDP traffic source
proc TelnetProc3UDP {TELNETCOUNTER} {
 global TelnetUDPAgent CurrTelnetUDPAgent

 $TelnetUDPAgent($CurrTelnetUDPAgent($TELNETCOUNTER)) start
}

## Starts the TCP traffic source
proc TelnetProc4TCP {TELNETCOUNTER} {
 global TelnetTCPAgent CurrTelnetTCPAgent

 $TelnetTCPAgent($CurrTelnetTCPAgent($TELNETCOUNTER)) stop
}

## Starts the UDP traffic source
proc TelnetProc4UDP {TELNETCOUNTER} {
 global TelnetUDPAgent CurrTelnetUDPAgent

 $TelnetUDPAgent($CurrTelnetUDPAgent($TELNETCOUNTER)) stop
}

## Detaches source & sink agents from their corrsponding nodes, and hanles free list
operations
## For TCP connection that was generated
proc TelnetProc5TCP {TELNETCOUNTER node1 node2} {
 global ns Telnetsrc Telnetsink TelnetFreeTCPAgentList CurrTelnetTCPAgent n
 global TelnetFreeTCPList CurrTelnetTCP

 $TelnetFreeTCPList Insert $CurrTelnetTCP($TELNETCOUNTER) $CurrTelnetTCP($TELNETCOUNTER)

 $ns detach-agent $n($node1) $Telnetsrc($TELNETCOUNTER)
 $ns detach-agent $n($node2) $Telnetsink($TELNETCOUNTER)
 $TelnetFreeTCPAgentList Insert $CurrTelnetTCPAgent($TELNETCOUNTER)
$CurrTelnetTCPAgent($TELNETCOUNTER)
}

## Detaches source & sink agents from their corrsponding nodes, and hanles free list
operations
## For UDP connection that was generated
proc TelnetProc5UDP {TELNETCOUNTER node1 node2} {
 global ns Telnetsrc Telnetsink TelnetFreeUDPAgentList CurrTelnetUDPAgent n
 global TelnetFreeUDPList CurrTelnetUDP

 $TelnetFreeUDPList Insert $CurrTelnetUDP($TELNETCOUNTER) $CurrTelnetUDP($TELNETCOUNTER)

 $ns detach-agent $n($node1) $Telnetsrc($TELNETCOUNTER)
 $ns detach-agent $n($node2) $Telnetsink($TELNETCOUNTER)
 $TelnetFreeUDPAgentList Insert $CurrTelnetUDPAgent($TELNETCOUNTER)
$CurrTelnetUDPAgent($TELNETCOUNTER)
}

# Insert Telnet sessions into list
for {set i 1} {$i <= $M_TELNET} {incr i} {
 set node1 [expr round([$rvuSource value])]
 set node2 [expr round([$rvuSink value])]
 while { $node1 == $node2 } {
  set node2 [expr round([$rvuSink value])]
 }
```

```
 set IAT [$rveMIATTelnet value]
 while {$IAT < 4.0*$SPD} {
  set IAT [$rveMIATTelnet value]
 }

 set Starting_Time [expr $START_TIME + $IAT]
 set Duration_Time [expr pow(2.0, ["Gaussian" log($MDT_TELNET)/log(2.0)
log($VAR_TELNET)/log(2.0)])]
 set lag [expr $Starting_Time - $START_TIME]

 if {[$rvuTCPUDP value] <= $PER_TCP} {
  $ns at [expr $Starting_Time-0.5*$lag] "TelnetProc1TCP $TELNETCOUNTER"
  $ns at [expr $Starting_Time-0.25*$lag] "TelnetProc2TCP $TELNETCOUNTER $node1 $node2"
  $ns at $Starting_Time "TelnetProc3TCP $TELNETCOUNTER [expr $MR_TELNET*$Duration_Time]"
  $ns at [expr $Starting_Time+$Duration_Time] "TelnetProc4TCP $TELNETCOUNTER"
  $ns at [expr $Starting_Time+$Duration_Time+$SPD] "TelnetProc5TCP $TELNETCOUNTER $node1
$node2"
 } else {
  $ns at [expr $Starting_Time-0.5*$lag] "TelnetProc1UDP $TELNETCOUNTER"
  $ns at [expr $Starting_Time-0.25*$lag] "TelnetProc2UDP $TELNETCOUNTER $node1 $node2"
  $ns at $Starting_Time "TelnetProc3UDP $TELNETCOUNTER"
  $ns at [expr $Starting_Time+$Duration_Time] "TelnetProc4UDP $TELNETCOUNTER"
  $ns at [expr $Starting_Time+$Duration_Time+$SPD] "TelnetProc5UDP $TELNETCOUNTER $node1
$node2"
 }

 $TelnetList Insert [expr $Starting_Time+$Duration_Time] 1

 incr TELNETCOUNTER
}

set Ending_Session [$TelnetList TimeOfFirst]
set Session_No [$TelnetList NoOfFirst]
$TelnetList DeleteFirst

while {$Session_No != 0} {
 set node1 [expr round([$rvuSource value])]
 set node2 [expr round([$rvuSink value])]
 while {$node1 == $node2} {
  set node2 [expr round([$rvuSink value])]
 }

 set IAT [$rveMIATTelnet value]
 while {$IAT < 4.0*$SPD} {
  set IAT [$rveMIATTelnet value]
 }

 set Starting_Time [expr $Ending_Session+$IAT]
 set Duration_Time [expr pow(2.0, ["Gaussian" log($MDT_TELNET)/log(2.0)
log($VAR_TELNET)/log(2.0)])]
 set lag [expr $Starting_Time - $Ending_Session]

 if {[$rvuTCPUDP value] <= $PER_TCP} {
  $ns at [expr $Starting_Time-0.5*$lag] "TelnetProc1TCP $TELNETCOUNTER"
  $ns at [expr $Starting_Time-0.25*$lag] "TelnetProc2TCP $TELNETCOUNTER $node1 $node2"
  $ns at $Starting_Time "TelnetProc3TCP $TELNETCOUNTER [expr $MR_TELNET*$Duration_Time]"
  $ns at [expr $Starting_Time+$Duration_Time] "TelnetProc4TCP $TELNETCOUNTER"
  $ns at [expr $Starting_Time+$Duration_Time+$SPD] "TelnetProc5TCP $TELNETCOUNTER $node1
$node2"
 } else {
  $ns at [expr $Starting_Time-0.5*$lag] "TelnetProc1UDP $TELNETCOUNTER"
  $ns at [expr $Starting_Time-0.25*$lag] "TelnetProc2UDP $TELNETCOUNTER $node1 $node2"
  $ns at $Starting_Time "TelnetProc3UDP $TELNETCOUNTER"
  $ns at [expr $Starting_Time+$Duration_Time] "TelnetProc4UDP $TELNETCOUNTER"
  $ns at [expr $Starting_Time+$Duration_Time+$SPD] "TelnetProc5UDP $TELNETCOUNTER $node1
$node2"
 }

 if {[expr $Starting_Time + $Duration_Time] <= $STOP_TIME} {
  set CurrCount [$TelnetList Count]
  set P1 [expr 1.0-$CurrCount/(2.0*$M_TELNET)]
```

```
  set P2 0.5
  if {[$rvuSession value]<=$P1} {$TelnetList Insert [expr $Starting_Time+$Duration_Time]
1}
  if {[$rvuSession value]<=$P2} {$TelnetList Insert [expr $Starting_Time+$Duration_Time]
1}
 }

 set Ending_Session [$TelnetList TimeOfFirst]
 set Session_No [$TelnetList NoOfFirst]
 $TelnetList DeleteFirst

 incr TELNETCOUNTER
}

#######################################################################
# Generating WWW traffic...
puts "Generating WWW traffic..."

set WWWCOUNTER 0

# RV for generating MIAT for WWW sessions (Poisson WWW arrivals)
set rveMIATWWW [new RandomVariable/Exponential]
$rveMIATWWW set avg_ $MIAT_WWW

# RV for generating mean duration time for WWW sessions (Poisson WWW session)
set rveMDTWWW [new RandomVariable/Exponential]
$rveMDTWWW set avg_ $MDT_WWW

set TotalWWWTCP 0
set TotalWWWUDP 0
set TotalWWWTCPAgent 0
set TotalWWWUDPAgent 0

set CurrWWWTCP(0) 0
set CurrWWWUDP(0) 0
set CurrWWWTCPAgent(0) 0
set CurrWWWUDPAgent(0) 0

set WWWList [new Application/Traffic/LList]
set WWWFreeTCPList [new Application/Traffic/LList]
set WWWFreeUDPList [new Application/Traffic/LList]
set WWWFreeTCPAgentList [new Application/Traffic/LList]
set WWWFreeUDPAgentList [new Application/Traffic/LList]

set WWWsrc(0) [new Agent/TCP]
set WWWsink(0) [new Agent/TCPSink]

set WWWTCPsrc(0) [new Agent/TCP]
set WWWTCPsink(0) [new Agent/TCPSink]
set WWWUDPsrc(0) [new Agent/UDP]
set WWWUDPsink(0) [new Agent/UDP]
set WWWTCPAgent(0) [new Application/FTP]
set WWWUDPAgent(0) [new Application/Traffic/SupFRP]

## Finds an idle agent, an idle source, and an idle sink or creates new ones
## For TCP session that is going to be generated
proc WWWProc1TCP { WWWCOUNTER } {
 global CurrWWWTCP WWWFreeTCPList TotalWWWTCP WWWsrc WWWsink WWWTCPsrc WWWTCPsink
 global CurrWWWTCPAgent WWWFreeTCPAgentList TotalWWWTCPAgent WWWTCPAgent
 global MPS_WWW

 set CurrWWWTCPAgent($WWWCOUNTER) [$WWWFreeTCPAgentList NoOfFirst]
 $WWWFreeTCPAgentList DeleteFirst
 if {$CurrWWWTCPAgent($WWWCOUNTER) == 0} {
  incr TotalWWWTCPAgent
  set CurrWWWTCPAgent($WWWCOUNTER) $TotalWWWTCPAgent
  set WWWTCPAgent($CurrWWWTCPAgent($WWWCOUNTER)) [new Application/FTP]
 }

 set CurrWWWTCP($WWWCOUNTER) [$WWWFreeTCPList NoOfFirst]
 $WWWFreeTCPList DeleteFirst
```

```
 if {$CurrWWWTCP($WWWCOUNTER) == 0} {
  incr TotalWWWTCP
  set CurrWWWTCP($WWWCOUNTER) $TotalWWWTCP
  set WWWTCPsrc($CurrWWWTCP($WWWCOUNTER)) [new Agent/TCP]
  set WWWTCPsink($CurrWWWTCP($WWWCOUNTER)) [new Agent/TCPSink]
 }
 set WWWsrc($WWWCOUNTER) $WWWTCPsrc($CurrWWWTCP($WWWCOUNTER))
 set WWWsink($WWWCOUNTER) $WWWTCPsink($CurrWWWTCP($WWWCOUNTER))

 $WWWsrc($WWWCOUNTER) set packetSize_ $MPS_WWW
}

## Finds an idle agent, an idle source, and an idle sink or creating new ones
## For UDP session that is going to be generated
proc WWWProc1UDP {WWWCOUNTER} {
 global CurrWWWUDP WWWFreeUDPList TotalWWWUDP WWWUDPsrc WWWUDPsink WWWsrc WWWsink
 global CurrWWWUDPAgent WWWFreeUDPAgentList TotalWWWUDPAgent WWWUDPAgent
 global MPS_WWW MBS_WWW MIT_WWW MR_WWW

 set CurrWWWUDPAgent($WWWCOUNTER) [$WWWFreeUDPAgentList NoOfFirst]
 $WWWFreeUDPAgentList DeleteFirst
 if {$CurrWWWUDPAgent($WWWCOUNTER) == 0} {
  incr TotalWWWUDPAgent
  set CurrWWWUDPAgent($WWWCOUNTER) $TotalWWWUDPAgent
  set WWWUDPAgent($CurrWWWUDPAgent($WWWCOUNTER)) [new Application/Traffic/SupFRP]
  $WWWUDPAgent($CurrWWWUDPAgent($WWWCOUNTER)) set packet_size_ $MPS_WWW
  $WWWUDPAgent($CurrWWWUDPAgent($WWWCOUNTER)) set rate_ $MR_WWW
  $WWWUDPAgent($CurrWWWUDPAgent($WWWCOUNTER)) set hurst_ 0.92
 }

 set CurrWWWUDP($WWWCOUNTER) [$WWWFreeUDPList NoOfFirst]
 $WWWFreeUDPList DeleteFirst
 if {$CurrWWWUDP($WWWCOUNTER) == 0} {
  incr TotalWWWUDP
  set CurrWWWUDP($WWWCOUNTER) $TotalWWWUDP
  set WWWUDPsrc($CurrWWWUDP($WWWCOUNTER)) [new Agent/UDP]
  set WWWUDPsink($CurrWWWUDP($WWWCOUNTER)) [new Agent/UDP]
 }
 set WWWsrc($WWWCOUNTER) $WWWUDPsrc($CurrWWWUDP($WWWCOUNTER))
 set WWWsink($WWWCOUNTER) $WWWUDPsink($CurrWWWUDP($WWWCOUNTER))
}

## Sets up the connection between previously identified source & destination
## For TCP session that is going to be generated
proc WWWProc2TCP {WWWCOUNTER node1 node2} {
 global WWWsrc WWWsink ns WWWTCPAgent CurrWWWTCPAgent n

 $ns attach-agent $n($node1) $WWWsrc($WWWCOUNTER)
 $ns attach-agent $n($node2) $WWWsink($WWWCOUNTER)
 $ns connect $WWWsrc($WWWCOUNTER) $WWWsink($WWWCOUNTER)
 $WWWTCPAgent($CurrWWWTCPAgent($WWWCOUNTER)) attach-agent $WWWsrc($WWWCOUNTER)
}

## Sets up the connection between previously identified source & destination
## For TCP session that is going to be generated
proc WWWProc2TCP {WWWCOUNTER node1 node2} {
 global WWWsrc WWWsink ns WWWTCPAgent CurrWWWTCPAgent n

 $ns attach-agent $n($node1) $WWWsrc($WWWCOUNTER)
 $ns attach-agent $n($node2) $WWWsink($WWWCOUNTER)
 $ns connect $WWWsrc($WWWCOUNTER) $WWWsink($WWWCOUNTER)
 $WWWTCPAgent($CurrWWWTCPAgent($WWWCOUNTER)) attach-agent $WWWsrc($WWWCOUNTER)
}

## Sets up the connection between previously identified source & destination
## For UDP session that is going to be generated
proc WWWProc2UDP {WWWCOUNTER node1 node2} {
 global WWWsrc WWWsink ns WWWUDPAgent CurrWWWUDPAgent n

 $ns attach-agent $n($node1) $WWWsrc($WWWCOUNTER)
 $ns attach-agent $n($node2) $WWWsink($WWWCOUNTER)
```

```
 $ns connect $WWWsrc($WWWCOUNTER) $WWWsink($WWWCOUNTER)
 $WWWUDPAgent($CurrWWWUDPAgent($WWWCOUNTER)) attach-agent $WWWsrc($WWWCOUNTER)
}

## Starts the TCP traffic source
proc WWWProc3TCP {WWWCOUNTER BytesToSend} {
 global WWWsrc

 $WWWsrc($WWWCOUNTER) send $BytesToSend
}

## Starts the UDP traffic source
proc WWWProc3UDP {WWWCOUNTER} {
 global WWWUDPAgent CurrWWWUDPAgent

 $WWWUDPAgent($CurrWWWUDPAgent($WWWCOUNTER)) start
}

## Starts the TCP traffic source
proc WWWProc4TCP {WWWCOUNTER} {
 global WWWTCPAgent CurrWWWTCPAgent

 $WWWTCPAgent($CurrWWWTCPAgent($WWWCOUNTER)) stop
}

## Starts the UDP traffic source
proc WWWProc4UDP {WWWCOUNTER} {
 global WWWUDPAgent CurrWWWUDPAgent

 $WWWUDPAgent($CurrWWWUDPAgent($WWWCOUNTER)) stop
}

## Detaches source & sink agents from their corrsponding nodes, and hanles free list
operations
## For TCP connection that was generated
proc WWWProc5TCP {WWWCOUNTER node1 node2} {
 global ns WWWsrc WWWsink WWWFreeTCPAgentList CurrWWWTCPAgent n
 global WWWFreeTCPList CurrWWWTCP

 $WWWFreeTCPList Insert $CurrWWWTCP($WWWCOUNTER) $CurrWWWTCP($WWWCOUNTER)

 $ns detach-agent $n($node1) $WWWsrc($WWWCOUNTER)
 $ns detach-agent $n($node2) $WWWsink($WWWCOUNTER)
 $WWWFreeTCPAgentList Insert $CurrWWWTCPAgent($WWWCOUNTER) $CurrWWWTCPAgent($WWWCOUNTER)
}

## Detaches source & sink agents from their corrsponding nodes, and hanles free list
operations
## For UDP connection that was generated
proc WWWProc5UDP {WWWCOUNTER node1 node2} {
 global ns WWWsrc WWWsink WWWFreeUDPAgentList CurrWWWUDPAgent n
 global WWWFreeUDPList CurrWWWUDP

 $WWWFreeUDPList Insert $CurrWWWUDP($WWWCOUNTER) $CurrWWWUDP($WWWCOUNTER)

 $ns detach-agent $n($node1) $WWWsrc($WWWCOUNTER)
 $ns detach-agent $n($node2) $WWWsink($WWWCOUNTER)
 $WWWFreeUDPAgentList Insert $CurrWWWUDPAgent($WWWCOUNTER) $CurrWWWUDPAgent($WWWCOUNTER)
}

# Insert WWW sessions into list
for {set i 1} {$i <= $M_WWW} {incr i} {
 set node1 [expr round([$rvuSource value])]
 set node2 [expr round([$rvuSink value])]
 while { $node1 == $node2 } {
  set node2 [expr round([$rvuSink value])]
 }

 set IAT [$rveMIATWWW value]
 while {$IAT < 4.0*$SPD} {
  set IAT [$rveMIATWWW value]
```

```
 }

 set Starting_Time [expr $START_TIME + $IAT]
 set Duration_Time [expr [$rveMDTWWW value]]
 set lag [expr $Starting_Time - $START_TIME]

 if {[$rvuTCPUDP value] <= $PER_TCP} {
  $ns at [expr $Starting_Time-0.5*$lag] "WWWProc1TCP $WWWCOUNTER"
  $ns at [expr $Starting_Time-0.25*$lag] "WWWProc2TCP $WWWCOUNTER $node1 $node2"
  $ns at $Starting_Time "WWWProc3TCP $WWWCOUNTER [expr
((1000.0*$MR_WWW)/8.0)*$Duration_Time]"
  $ns at [expr $Starting_Time+$Duration_Time] "WWWProc4TCP $WWWCOUNTER"
  $ns at [expr $Starting_Time+$Duration_Time+$SPD] "WWWProc5TCP $WWWCOUNTER $node1
$node2"
 } else {
  $ns at [expr $Starting_Time-0.5*$lag] "WWWProc1UDP $WWWCOUNTER"
  $ns at [expr $Starting_Time-0.25*$lag] "WWWProc2UDP $WWWCOUNTER $node1 $node2"
  $ns at $Starting_Time "WWWProc3UDP $WWWCOUNTER"
  $ns at [expr $Starting_Time+$Duration_Time] "WWWProc4UDP $WWWCOUNTER"
  $ns at [expr $Starting_Time+$Duration_Time+$SPD] "WWWProc5UDP $WWWCOUNTER $node1
$node2"
 }

 $WWWList Insert [expr $Starting_Time+$Duration_Time] 1

 incr WWWCOUNTER
}

set Ending_Session [$WWWList TimeOfFirst]
set Session_No [$WWWList NoOfFirst]
$WWWList DeleteFirst

while {$Session_No != 0} {
 set node1 [expr round([$rvuSource value])]
 set node2 [expr round([$rvuSink value])]
 while {$node1 == $node2} {
  set node2 [expr round([$rvuSink value])]
 }

 set IAT [$rveMIATWWW value]
 while {$IAT < 4.0*$SPD} {
  set IAT [$rveMIATWWW value]
 }

 set Starting_Time [expr $Ending_Session+$IAT]
 set Duration_Time [expr [$rveMDTWWW value]]
 set lag [expr $Starting_Time - $Ending_Session]

 if {[$rvuTCPUDP value] <= $PER_TCP} {
  $ns at [expr $Starting_Time-0.5*$lag] "WWWProc1TCP $WWWCOUNTER"
  $ns at [expr $Starting_Time-0.25*$lag] "WWWProc2TCP $WWWCOUNTER $node1 $node2"
  $ns at $Starting_Time "WWWProc3TCP $WWWCOUNTER [expr
((1000.0*$MR_WWW)/8.0)*$Duration_Time]"
  $ns at [expr $Starting_Time+$Duration_Time] "WWWProc4TCP $WWWCOUNTER"
  $ns at [expr $Starting_Time+$Duration_Time+$SPD] "WWWProc5TCP $WWWCOUNTER $node1
$node2"
 } else {
  $ns at [expr $Starting_Time-0.5*$lag] "WWWProc1UDP $WWWCOUNTER"
  $ns at [expr $Starting_Time-0.25*$lag] "WWWProc2UDP $WWWCOUNTER $node1 $node2"
  $ns at $Starting_Time "WWWProc3UDP $WWWCOUNTER"
  $ns at [expr $Starting_Time+$Duration_Time] "WWWProc4UDP $WWWCOUNTER"
  $ns at [expr $Starting_Time+$Duration_Time+$SPD] "WWWProc5UDP $WWWCOUNTER $node1
$node2"
 }

 if {[expr $Starting_Time + $Duration_Time] <= $STOP_TIME} {
  set CurrCount [$WWWList Count]
  set P1 [expr 1.0-$CurrCount/(2.0*$M_WWW)]
  set P2 0.5
  if {[$rvuSession value]<=$P1} {$WWWList Insert [expr $Starting_Time+$Duration_Time] 1}
  if {[$rvuSession value]<=$P2} {$WWWList Insert [expr $Starting_Time+$Duration_Time] 1}
```

```
 }

 set Ending_Session [$WWWList TimeOfFirst]
 set Session_No [$WWWList NoOfFirst]
 $WWWList DeleteFirst

 incr WWWCOUNTER
}

#######################################################################
# Generating X11 traffic...
puts "Generating X11 traffic..."

set X11COUNTER 0

# RV for generating MIAT for X11 sessions (Poisson X11 arrivals)
set rveMIATX11 [new RandomVariable/Exponential]
$rveMIATX11 set avg_ $MIAT_X11

# RV for generating mean duration time for X11 sessions (Poisson X11 session)
set rveMDTX11 [new RandomVariable/Exponential]
$rveMDTX11 set avg_ $MDT_X11

set TotalX11TCP 0
set TotalX11UDP 0
set TotalX11TCPAgent 0
set TotalX11UDPAgent 0

set CurrX11TCP(0) 0
set CurrX11UDP(0) 0
set CurrX11TCPAgent(0) 0
set CurrX11UDPAgent(0) 0

set X11List [new Application/Traffic/LList]
set X11FreeTCPList [new Application/Traffic/LList]
set X11FreeUDPList [new Application/Traffic/LList]
set X11FreeTCPAgentList [new Application/Traffic/LList]
set X11FreeUDPAgentList [new Application/Traffic/LList]

set X11src(0) [new Agent/TCP]
set X11sink(0) [new Agent/TCPSink]

set X11TCPsrc(0) [new Agent/TCP]
set X11TCPsink(0) [new Agent/TCPSink]
set X11UDPsrc(0) [new Agent/UDP]
set X11UDPsink(0) [new Agent/UDP]
set X11TCPAgent(0) [new Application/FTP]
set X11UDPAgent(0) [new Application/Traffic/SupFRP]

## Finds an idle agent, an idle source, and an idle sink or creates new ones
## For TCP session that is going to be generated
proc X11Proc1TCP { X11COUNTER } {
 global CurrX11TCP X11FreeTCPList TotalX11TCP X11src X11sink X11TCPsrc X11TCPsink
 global CurrX11TCPAgent X11FreeTCPAgentList TotalX11TCPAgent X11TCPAgent
 global MPS_X11

 set CurrX11TCPAgent($X11COUNTER) [$X11FreeTCPAgentList NoOfFirst]
 $X11FreeTCPAgentList DeleteFirst
 if {$CurrX11TCPAgent($X11COUNTER) == 0} {
  incr TotalX11TCPAgent
  set CurrX11TCPAgent($X11COUNTER) $TotalX11TCPAgent
  set X11TCPAgent($CurrX11TCPAgent($X11COUNTER)) [new Application/FTP]
 }

 set CurrX11TCP($X11COUNTER) [$X11FreeTCPList NoOfFirst]
 $X11FreeTCPList DeleteFirst
 if {$CurrX11TCP($X11COUNTER) == 0} {
  incr TotalX11TCP
  set CurrX11TCP($X11COUNTER) $TotalX11TCP
  set X11TCPsrc($CurrX11TCP($X11COUNTER)) [new Agent/TCP]
  set X11TCPsink($CurrX11TCP($X11COUNTER)) [new Agent/TCPSink]
```

```
 }
 set X11src($X11COUNTER) $X11TCPsrc($CurrX11TCP($X11COUNTER))
 set X11sink($X11COUNTER) $X11TCPsink($CurrX11TCP($X11COUNTER))

 $X11src($X11COUNTER) set packetSize_ $MPS_X11
}

## Finds an idle agent, an idle source, and an idle sink or creating new ones
## For UDP session that is going to be generated
proc X11Proc1UDP {X11COUNTER} {
 global CurrX11UDP X11FreeUDPList TotalX11UDP X11UDPsrc X11UDPsink X11src X11sink
 global CurrX11UDPAgent X11FreeUDPAgentList TotalX11UDPAgent X11UDPAgent
 global MPS_X11 MBS_X11 MIT_X11 MR_X11

 set CurrX11UDPAgent($X11COUNTER) [$X11FreeUDPAgentList NoOfFirst]
 $X11FreeUDPAgentList DeleteFirst
 if {$CurrX11UDPAgent($X11COUNTER) == 0} {
  incr TotalX11UDPAgent
  set CurrX11UDPAgent($X11COUNTER) $TotalX11UDPAgent
  set X11UDPAgent($CurrX11UDPAgent($X11COUNTER)) [new Application/Traffic/SupFRP]
  $X11UDPAgent($CurrX11UDPAgent($X11COUNTER)) set packet_size_ $MPS_X11
  $X11UDPAgent($CurrX11UDPAgent($X11COUNTER)) set rate_ $MR_X11
  $X11UDPAgent($CurrX11UDPAgent($X11COUNTER)) set hurst_ 0.92
 }

 set CurrX11UDP($X11COUNTER) [$X11FreeUDPList NoOfFirst]
 $X11FreeUDPList DeleteFirst
 if {$CurrX11UDP($X11COUNTER) == 0} {
  incr TotalX11UDP
  set CurrX11UDP($X11COUNTER) $TotalX11UDP
  set X11UDPsrc($CurrX11UDP($X11COUNTER)) [new Agent/UDP]
  set X11UDPsink($CurrX11UDP($X11COUNTER)) [new Agent/UDP]
 }
 set X11src($X11COUNTER) $X11UDPsrc($CurrX11UDP($X11COUNTER))
 set X11sink($X11COUNTER) $X11UDPsink($CurrX11UDP($X11COUNTER))
}

## Sets up the connection between previously identified source & destination
## For TCP session that is going to be generated
proc X11Proc2TCP {X11COUNTER node1 node2} {
 global X11src X11sink ns X11TCPAgent CurrX11TCPAgent n

 $ns attach-agent $n($node1) $X11src($X11COUNTER)
 $ns attach-agent $n($node2) $X11sink($X11COUNTER)
 $ns connect $X11src($X11COUNTER) $X11sink($X11COUNTER)
 $X11TCPAgent($CurrX11TCPAgent($X11COUNTER)) attach-agent $X11src($X11COUNTER)
}

## Sets up the connection between previously identified source & destination
## For UDP session that is going to be generated
proc X11Proc2UDP {X11COUNTER node1 node2} {
 global X11src X11sink ns X11UDPAgent CurrX11UDPAgent n

 $ns attach-agent $n($node1) $X11src($X11COUNTER)
 $ns attach-agent $n($node2) $X11sink($X11COUNTER)
 $ns connect $X11src($X11COUNTER) $X11sink($X11COUNTER)
 $X11UDPAgent($CurrX11UDPAgent($X11COUNTER)) attach-agent $X11src($X11COUNTER)
}

## Starts the TCP traffic source
proc X11Proc3TCP {X11COUNTER BytesToSend} {
 global X11src

 $X11src($X11COUNTER) send $BytesToSend
}

## Starts the UDP traffic source
proc X11Proc3UDP {X11COUNTER} {
 global X11UDPAgent CurrX11UDPAgent

 $X11UDPAgent($CurrX11UDPAgent($X11COUNTER)) start
```

```
}

## Starts the TCP traffic source
proc X11Proc4TCP {X11COUNTER} {
 global X11TCPAgent CurrX11TCPAgent

 $X11TCPAgent($CurrX11TCPAgent($X11COUNTER)) stop
}

## Starts the UDP traffic source
proc X11Proc4UDP {X11COUNTER} {
 global X11UDPAgent CurrX11UDPAgent

 $X11UDPAgent($CurrX11UDPAgent($X11COUNTER)) stop
}

## Detaches source & sink agents from their corrsponding nodes, and hanles free list
operations
## For TCP connection that was generated
proc X11Proc5TCP {X11COUNTER node1 node2} {
 global ns X11src X11sink X11FreeTCPAgentList CurrX11TCPAgent n
 global X11FreeTCPList CurrX11TCP

 $X11FreeTCPList Insert $CurrX11TCP($X11COUNTER) $CurrX11TCP($X11COUNTER)

 $ns detach-agent $n($node1) $X11src($X11COUNTER)
 $ns detach-agent $n($node2) $X11sink($X11COUNTER)
 $X11FreeTCPAgentList Insert $CurrX11TCPAgent($X11COUNTER) $CurrX11TCPAgent($X11COUNTER)
}

## Detaches source & sink agents from their corrsponding nodes, and hanles free list
operations
## For UDP connection that was generated
proc X11Proc5UDP {X11COUNTER node1 node2} {
 global ns X11src X11sink X11FreeUDPAgentList CurrX11UDPAgent n
 global X11FreeUDPList CurrX11UDP

 $X11FreeUDPList Insert $CurrX11UDP($X11COUNTER) $CurrX11UDP($X11COUNTER)

 $ns detach-agent $n($node1) $X11src($X11COUNTER)
 $ns detach-agent $n($node2) $X11sink($X11COUNTER)
 $X11FreeUDPAgentList Insert $CurrX11UDPAgent($X11COUNTER) $CurrX11UDPAgent($X11COUNTER)
}

# Insert X11 sessions into list
for {set i 1} {$i <= $M_X11} {incr i} {
 set node1 [expr round([$rvuSource value])]
 set node2 [expr round([$rvuSink value])]
 while { $node1 == $node2 } {
  set node2 [expr round([$rvuSink value])]
 }

 set IAT [$rveMIATX11 value]
 while {$IAT < 4.0*$SPD} {
  set IAT [$rveMIATX11 value]
 }

 set Starting_Time [expr $START_TIME + $IAT]
 set Duration_Time [expr [$rveMDTX11 value]]
 set lag [expr $Starting_Time - $START_TIME]

 if {[$rvuTCPUDP value] <= $PER_TCP} {
  $ns at [expr $Starting_Time-0.5*$lag] "X11Proc1TCP $X11COUNTER"
  $ns at [expr $Starting_Time-0.25*$lag] "X11Proc2TCP $X11COUNTER $node1 $node2"
  $ns at $Starting_Time "X11Proc3TCP $X11COUNTER [expr
((1000.0*$MR_X11)/8.0)*$Duration_Time]"
  $ns at [expr $Starting_Time+$Duration_Time] "X11Proc4TCP $X11COUNTER"
  $ns at [expr $Starting_Time+$Duration_Time+$SPD] "X11Proc5TCP $X11COUNTER $node1
$node2"
 } else {
  $ns at [expr $Starting_Time-0.5*$lag] "X11Proc1UDP $X11COUNTER"
```

```
  $ns at [expr $Starting_Time-0.25*$lag] "X11Proc2UDP $X11COUNTER $node1 $node2"
  $ns at $Starting_Time "X11Proc3UDP $X11COUNTER"
  $ns at [expr $Starting_Time+$Duration_Time] "X11Proc4UDP $X11COUNTER"
  $ns at [expr $Starting_Time+$Duration_Time+$SPD] "X11Proc5UDP $X11COUNTER $node1
$node2"
 }

 $X11List Insert [expr $Starting_Time+$Duration_Time] 1

 incr X11COUNTER
}

set Ending_Session [$X11List TimeOfFirst]
set Session_No [$X11List NoOfFirst]
$X11List DeleteFirst

while {$Session_No != 0} {
 set node1 [expr round([$rvuSource value])]
 set node2 [expr round([$rvuSink value])]
 while {$node1 == $node2} {
  set node2 [expr round([$rvuSink value])]
 }

 set IAT [$rveMIATX11 value]
 while {$IAT < 4.0*$SPD} {
  set IAT [$rveMIATX11 value]
 }

 set Starting_Time [expr $Ending_Session+$IAT]
 set Duration_Time [expr [$rveMDTX11 value]]
 set lag [expr $Starting_Time - $Ending_Session]

 if {[$rvuTCPUDP value] <= $PER_TCP} {
  $ns at [expr $Starting_Time-0.5*$lag] "X11Proc1TCP $X11COUNTER"
  $ns at [expr $Starting_Time-0.25*$lag] "X11Proc2TCP $X11COUNTER $node1 $node2"
  $ns at $Starting_Time "X11Proc3TCP $X11COUNTER [expr
((1000.0*$MR_X11)/8.0)*$Duration_Time]"
  $ns at [expr $Starting_Time+$Duration_Time] "X11Proc4TCP $X11COUNTER"
  $ns at [expr $Starting_Time+$Duration_Time+$SPD] "X11Proc5TCP $X11COUNTER $node1
$node2"
 } else {
  $ns at [expr $Starting_Time-0.5*$lag] "X11Proc1UDP $X11COUNTER"
  $ns at [expr $Starting_Time-0.25*$lag] "X11Proc2UDP $X11COUNTER $node1 $node2"
  $ns at $Starting_Time "X11Proc3UDP $X11COUNTER"
  $ns at [expr $Starting_Time+$Duration_Time] "X11Proc4UDP $X11COUNTER"
  $ns at [expr $Starting_Time+$Duration_Time+$SPD] "X11Proc5UDP $X11COUNTER $node1
$node2"
 }

 if {[expr $Starting_Time + $Duration_Time] <= $STOP_TIME} {
  set CurrCount [$X11List Count]
  set P1 [expr 1.0-$CurrCount/(2.0*$M_X11)]
  set P2 0.5
  if {[$rvuSession value]<=$P1} {$X11List Insert [expr $Starting_Time+$Duration_Time] 1}
  if {[$rvuSession value]<=$P2} {$X11List Insert [expr $Starting_Time+$Duration_Time] 1}
 }

 set Ending_Session [$X11List TimeOfFirst]
 set Session_No [$X11List NoOfFirst]
 $X11List DeleteFirst

 incr X11COUNTER
}

#######################################################################
# Generating FTP traffic...
puts "Generating FTP traffic..."

set FTPCOUNTER 0

# RV for generating MIAT for FTP sessions (Poisson FTP arrivals)
```

```
set rveMIATFTP [new RandomVariable/Exponential]
$rveMIATFTP set avg_ $MIAT_FTP

# RV for generating mean duration time for FTP sessions (Poisson FTP session)
set rveMDTFTP [new RandomVariable/Exponential]
$rveMDTFTP set avg_ $MDT_FTP

set TotalFTPTCP 0
set TotalFTPUDP 0
set TotalFTPTCPAgent 0
set TotalFTPUDPAgent 0

set CurrFTPTCP(0) 0
set CurrFTPUDP(0) 0
set CurrFTPTCPAgent(0) 0
set CurrFTPUDPAgent(0) 0

set FTPList [new Application/Traffic/LList]
set FTPFreeTCPList [new Application/Traffic/LList]
set FTPFreeUDPList [new Application/Traffic/LList]
set FTPFreeTCPAgentList [new Application/Traffic/LList]
set FTPFreeUDPAgentList [new Application/Traffic/LList]

set FTPsrc(0) [new Agent/TCP]
set FTPsink(0) [new Agent/TCPSink]

set FTPTCPsrc(0) [new Agent/TCP]
set FTPTCPsink(0) [new Agent/TCPSink]
set FTPUDPsrc(0) [new Agent/UDP]
set FTPUDPsink(0) [new Agent/UDP]
set FTPTCPAgent(0) [new Application/FTP]
set FTPUDPAgent(0) [new Application/Traffic/Pareto]

## Finds an idle agent, an idle source, and an idle sink or creates new ones
## For TCP session that is going to be generated
proc FTPProc1TCP { FTPCOUNTER } {
 global CurrFTPTCP FTPFreeTCPList TotalFTPTCP FTPsrc FTPsink FTPTCPsrc FTPTCPsink
 global CurrFTPTCPAgent FTPFreeTCPAgentList TotalFTPTCPAgent FTPTCPAgent
 global MPS_FTP MBS_FTP MR_FTP

 set CurrFTPTCPAgent($FTPCOUNTER) [$FTPFreeTCPAgentList NoOfFirst]
 $FTPFreeTCPAgentList DeleteFirst
 if {$CurrFTPTCPAgent($FTPCOUNTER) == 0} {
  incr TotalFTPTCPAgent
  set CurrFTPTCPAgent($FTPCOUNTER) $TotalFTPTCPAgent
  set FTPTCPAgent($CurrFTPTCPAgent($FTPCOUNTER)) [new Application/FTP]
 }

 set CurrFTPTCP($FTPCOUNTER) [$FTPFreeTCPList NoOfFirst]
 $FTPFreeTCPList DeleteFirst
 if {$CurrFTPTCP($FTPCOUNTER) == 0} {
  incr TotalFTPTCP
  set CurrFTPTCP($FTPCOUNTER) $TotalFTPTCP
  set FTPTCPsrc($CurrFTPTCP($FTPCOUNTER)) [new Agent/TCP]
  set FTPTCPsink($CurrFTPTCP($FTPCOUNTER)) [new Agent/TCPSink]
 }
 set FTPsrc($FTPCOUNTER) $FTPTCPsrc($CurrFTPTCP($FTPCOUNTER))
 set FTPsink($FTPCOUNTER) $FTPTCPsink($CurrFTPTCP($FTPCOUNTER))

 $FTPsrc($FTPCOUNTER) set packetSize_ $MPS_FTP
 $FTPsrc($FTPCOUNTER) set maxburst_ [expr round($MBS_FTP*$MR_FTP/$MPS_FTP)]
 ## Mean idle time is ignored for TCP FTP sessions...
}

## Finds an idle agent, an idle source, and an idle sink or creating new ones
## For UDP session that is going to be generated
proc FTPProc1UDP {FTPCOUNTER} {
 global CurrFTPUDP FTPFreeUDPList TotalFTPUDP FTPUDPsrc FTPUDPsink FTPsrc FTPsink
 global CurrFTPUDPAgent FTPFreeUDPAgentList TotalFTPUDPAgent FTPUDPAgent
 global MPS_FTP MBS_FTP MIT_FTP MR_FTP
```

```
 set CurrFTPUDPAgent($FTPCOUNTER) [$FTPFreeUDPAgentList NoOfFirst]
 $FTPFreeUDPAgentList DeleteFirst
 if {$CurrFTPUDPAgent($FTPCOUNTER) == 0} {
  incr TotalFTPUDPAgent
  set CurrFTPUDPAgent($FTPCOUNTER) $TotalFTPUDPAgent
  set FTPUDPAgent($CurrFTPUDPAgent($FTPCOUNTER)) [new Application/Traffic/Pareto]
  $FTPUDPAgent($CurrFTPUDPAgent($FTPCOUNTER)) set packet_size_ $MPS_FTP
  $FTPUDPAgent($CurrFTPUDPAgent($FTPCOUNTER)) set burst_size_ $MBS_FTP
  $FTPUDPAgent($CurrFTPUDPAgent($FTPCOUNTER)) set idle_time_ $MIT_FTP
  $FTPUDPAgent($CurrFTPUDPAgent($FTPCOUNTER)) set rate_ $MR_FTP
  $FTPUDPAgent($CurrFTPUDPAgent($FTPCOUNTER)) set shape_ 1.5
 }

 set CurrFTPUDP($FTPCOUNTER) [$FTPFreeUDPList NoOfFirst]
 $FTPFreeUDPList DeleteFirst
 if {$CurrFTPUDP($FTPCOUNTER) == 0} {
  incr TotalFTPUDP
  set CurrFTPUDP($FTPCOUNTER) $TotalFTPUDP
  set FTPUDPsrc($CurrFTPUDP($FTPCOUNTER)) [new Agent/UDP]
  set FTPUDPsink($CurrFTPUDP($FTPCOUNTER)) [new Agent/UDP]
 }
 set FTPsrc($FTPCOUNTER) $FTPUDPsrc($CurrFTPUDP($FTPCOUNTER))
 set FTPsink($FTPCOUNTER) $FTPUDPsink($CurrFTPUDP($FTPCOUNTER))
}

## Sets up the connection between previously identified source & destination
## For TCP session that is going to be generated
proc FTPProc2TCP {FTPCOUNTER node1 node2} {
 global FTPsrc FTPsink ns FTPTCPAgent CurrFTPTCPAgent n

 $ns attach-agent $n($node1) $FTPsrc($FTPCOUNTER)
 $ns attach-agent $n($node2) $FTPsink($FTPCOUNTER)
 $ns connect $FTPsrc($FTPCOUNTER) $FTPsink($FTPCOUNTER)
 $FTPTCPAgent($CurrFTPTCPAgent($FTPCOUNTER)) attach-agent $FTPsrc($FTPCOUNTER)
}

## Sets up the connection between previously identified source & destination
## For UDP session that is going to be generated
proc FTPProc2UDP {FTPCOUNTER node1 node2} {
 global FTPsrc FTPsink ns FTPUDPAgent CurrFTPUDPAgent n

 $ns attach-agent $n($node1) $FTPsrc($FTPCOUNTER)
 $ns attach-agent $n($node2) $FTPsink($FTPCOUNTER)
 $ns connect $FTPsrc($FTPCOUNTER) $FTPsink($FTPCOUNTER)
 $FTPUDPAgent($CurrFTPUDPAgent($FTPCOUNTER)) attach-agent $FTPsrc($FTPCOUNTER)
}

## Starts the TCP traffic source
proc FTPProc3TCP {FTPCOUNTER BytesToSend} {
 global FTPsrc

 $FTPsrc($FTPCOUNTER) send $BytesToSend
}

## Starts the UDP traffic source
proc FTPProc3UDP {FTPCOUNTER} {
 global FTPUDPAgent CurrFTPUDPAgent

 $FTPUDPAgent($CurrFTPUDPAgent($FTPCOUNTER)) start
}

## Starts the TCP traffic source
proc FTPProc4TCP {FTPCOUNTER} {
 global FTPTCPAgent CurrFTPTCPAgent

 $FTPTCPAgent($CurrFTPTCPAgent($FTPCOUNTER)) stop
}

## Starts the UDP traffic source
proc FTPProc4UDP {FTPCOUNTER} {
 global FTPUDPAgent CurrFTPUDPAgent
```

```
 $FTPUDPAgent($CurrFTPUDPAgent($FTPCOUNTER)) stop
}

## Detaches source & sink agents from their corrsponding nodes, and hanles free list
operations
## For TCP connection that was generated
proc FTPProc5TCP {FTPCOUNTER node1 node2} {
 global ns FTPsrc FTPsink FTPFreeTCPAgentList CurrFTPTCPAgent n
 global FTPFreeTCPList CurrFTPTCP

 $FTPFreeTCPList Insert $CurrFTPTCP($FTPCOUNTER) $CurrFTPTCP($FTPCOUNTER)

 $ns detach-agent $n($node1) $FTPsrc($FTPCOUNTER)
 $ns detach-agent $n($node2) $FTPsink($FTPCOUNTER)
 $FTPFreeTCPAgentList Insert $CurrFTPTCPAgent($FTPCOUNTER) $CurrFTPTCPAgent($FTPCOUNTER)
}

## Detaches source & sink agents from their corrsponding nodes, and hanles free list
operations
## For UDP connection that was generated
proc FTPProc5UDP {FTPCOUNTER node1 node2} {
 global ns FTPsrc FTPsink FTPFreeUDPAgentList CurrFTPUDPAgent n
 global FTPFreeUDPList CurrFTPUDP

 $FTPFreeUDPList Insert $CurrFTPUDP($FTPCOUNTER) $CurrFTPUDP($FTPCOUNTER)

 $ns detach-agent $n($node1) $FTPsrc($FTPCOUNTER)
 $ns detach-agent $n($node2) $FTPsink($FTPCOUNTER)
 $FTPFreeUDPAgentList Insert $CurrFTPUDPAgent($FTPCOUNTER) $CurrFTPUDPAgent($FTPCOUNTER)
}

# Insert FTP sessions into list
for {set i 1} {$i <= $M_FTP} {incr i} {
 set node1 [expr round([$rvuSource value])]
 set node2 [expr round([$rvuSink value])]
 while { $node1 == $node2 } {
  set node2 [expr round([$rvuSink value])]
 }

 set IAT [$rveMIATFTP value]
 while {$IAT < 4.0*$SPD} {
  set IAT [$rveMIATFTP value]
 }

 set Starting_Time [expr $START_TIME + $IAT]
 set Duration_Time [expr [$rveMDTFTP value]]
 set lag [expr $Starting_Time - $START_TIME]

 if {[$rvuTCPUDP value] <= $PER_TCP} {
  $ns at [expr $Starting_Time-0.5*$lag] "FTPProc1TCP $FTPCOUNTER"
  $ns at [expr $Starting_Time-0.25*$lag] "FTPProc2TCP $FTPCOUNTER $node1 $node2"
  $ns at $Starting_Time "FTPProc3TCP $FTPCOUNTER [expr $MR_FTP*$Duration_Time]"
  $ns at [expr $Starting_Time+$Duration_Time] "FTPProc4TCP $FTPCOUNTER"
  $ns at [expr $Starting_Time+$Duration_Time+$SPD] "FTPProc5TCP $FTPCOUNTER $node1
$node2"
 } else {
  $ns at [expr $Starting_Time-0.5*$lag] "FTPProc1UDP $FTPCOUNTER"
  $ns at [expr $Starting_Time-0.25*$lag] "FTPProc2UDP $FTPCOUNTER $node1 $node2"
  $ns at $Starting_Time "FTPProc3UDP $FTPCOUNTER"
  $ns at [expr $Starting_Time+$Duration_Time] "FTPProc4UDP $FTPCOUNTER"
  $ns at [expr $Starting_Time+$Duration_Time+$SPD] "FTPProc5UDP $FTPCOUNTER $node1
$node2"
 }

 $FTPList Insert [expr $Starting_Time+$Duration_Time] 1

 incr FTPCOUNTER
}

set Ending_Session [$FTPList TimeOfFirst]
```

```
set Session_No [$FTPList NoOfFirst]
$FTPList DeleteFirst

while {$Session_No != 0} {
 set node1 [expr round([$rvuSource value])]
 set node2 [expr round([$rvuSink value])]
 while {$node1 == $node2} {
  set node2 [expr round([$rvuSink value])]
 }

 set IAT [$rveMIATFTP value]
 while {$IAT < 4.0*$SPD} {
  set IAT [$rveMIATFTP value]
 }

 set Starting_Time [expr $Ending_Session+$IAT]
 set Duration_Time [expr [$rveMDTFTP value]]
 set lag [expr $Starting_Time - $Ending_Session]

 if {[$rvuTCPUDP value] <= $PER_TCP} {
  $ns at [expr $Starting_Time-0.5*$lag] "FTPProc1TCP $FTPCOUNTER"
  $ns at [expr $Starting_Time-0.25*$lag] "FTPProc2TCP $FTPCOUNTER $node1 $node2"
  $ns at $Starting_Time "FTPProc3TCP $FTPCOUNTER [expr $MR_FTP*$Duration_Time]"
  $ns at [expr $Starting_Time+$Duration_Time] "FTPProc4TCP $FTPCOUNTER"
  $ns at [expr $Starting_Time+$Duration_Time+$SPD] "FTPProc5TCP $FTPCOUNTER $node1
$node2"
 } else {
  $ns at [expr $Starting_Time-0.5*$lag] "FTPProc1UDP $FTPCOUNTER"
  $ns at [expr $Starting_Time-0.25*$lag] "FTPProc2UDP $FTPCOUNTER $node1 $node2"
  $ns at $Starting_Time "FTPProc3UDP $FTPCOUNTER"
  $ns at [expr $Starting_Time+$Duration_Time] "FTPProc4UDP $FTPCOUNTER"
  $ns at [expr $Starting_Time+$Duration_Time+$SPD] "FTPProc5UDP $FTPCOUNTER $node1
$node2"
 }

 if {[expr $Starting_Time + $Duration_Time] <= $STOP_TIME} {
  set CurrCount [$FTPList Count]
  set P1 [expr 1.0-$CurrCount/(2.0*$M_FTP)]
  set P2 0.5
  if {[$rvuSession value]<=$P1} {$FTPList Insert [expr $Starting_Time+$Duration_Time] 1}
  if {[$rvuSession value]<=$P2} {$FTPList Insert [expr $Starting_Time+$Duration_Time] 1}
 }

 set Ending_Session [$FTPList TimeOfFirst]
 set Session_No [$FTPList NoOfFirst]
 $FTPList DeleteFirst

 incr FTPCOUNTER
}

$ns at $STOP_TIME "finish"

$ns run
```

Appendix B. C++ CODE OF Application/Traffic/SupFRP

C++ code of SupFRP is available in file “SupFRP.cc”.

“SupFRP.cc”

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "trafgen.h"

int count=0;
double Alfa, Gamma, A,        L, // # of packet arrivals per sec
                      M, // # of FRPs
                      rate_, // transmission rate in Kbps
                      H; // hurst parameter
long Seed;
double S=0.0; // saves the time of the previous arrival...
struct node *saver; // saves the first node of the linked list for the next time...

struct node{
        int     processno;
        double  time;
        struct  node *next;
};
struct node *head=NULL;

class SupFRP_Traffic : public TrafficGenerator {

 public:
        SupFRP_Traffic();
        virtual double next_interval(int&);
        int on() { return on_;}

struct node *Insert(struct node *HList, int newprocess, double newtime)
{
        struct node *save, *temp1, *temp2;

        save=(struct node*)malloc(sizeof(struct node));
        save->time = newtime;
        save->processno = newprocess;

        if(HList==NULL){
                HList=save;
                HList->next=NULL;
        }
        else if (newtime<=HList->time){
                save->next=HList;
                HList=save;
        }
        else{
                temp2=HList;

                while((temp2->time<=newtime)&&(temp2->next!=NULL)){
                        temp1=temp2;
                        temp2=temp2->next;
                }
                if (temp2->time<=newtime){
                        temp1=temp2;
                        temp2=temp2->next;
                }
                temp1->next=save;
                save->next=temp2;
        }
        return(HList);
}

struct node *GetMin(struct node *HList, struct node ** DeletedNode)
```

```
{
        (*DeletedNode)=HList;
        return(HList->next);
}


double GenerateTime2(double U)
{
        if (U>=exp(-Gamma))
          return(-(1/Gamma)*A*log(U));
        else
          return (exp((double)-1)*A*exp((double)((-1/Gamma)*log(U))));
}

double GenerateTime(double U)
{
        double V;

        V=(1+(Gamma-1)*exp((double)Gamma))*U/Gamma;
        if (V>=1)
          return(-(1/Gamma)*A*log(U*(Gamma*V-1)/(Gamma*V-U)));
        else
          return (A*exp((double)((1/(1-Gamma))*log(V))));
}

double SupFRP(void)
{
        int j;
        double Taoj;
        double diff=0.0;

        if (count == 0) {
                for(j=1;j<=M;j++) {
                        Taoj = GenerateTime(drand48());
                        head = Insert(head,j,Taoj);
                }

                head = GetMin(head,&saver);
                diff = saver->time-S;
                S = saver->time;
                count++;
        } else {
                Taoj = GenerateTime2(drand48());
                saver->time += Taoj;
                head = Insert(head,saver->processno,saver->time);
                free(saver); head = GetMin(head,&saver);
                diff = saver->time-S;
                S = saver->time;
        }

        return (diff);
}
        protected:
                void init();
                int on_;

};

SupFRP_Traffic::SupFRP_Traffic()
{
         bind("rate_", &rate_);
         bind("packet_size_", &size_);
         bind("FRPs_", &M);
         bind("hurst_", &H);
}

void SupFRP_Traffic::init()
{
        on_ = 0;
        Seed = 12351;
        srand48(Seed);
```

```
        L = (double)(rate_*1000)/(double)(8*size_); // find packet rate as packets/sec

        Alfa = (2*(double)H-1);
        Gamma = (2-Alfa);
        A = ((M*Gamma)/(L*(1+exp(-Gamma)/(Gamma-1))));

        if (agent_)
                agent_->set_pkttype(PT_SupFRP);
}

double SupFRP_Traffic::next_interval(int& size)
{
        double t = SupFRP();
        on_ = 1;
        size = size_;
        return(t);
}

static class SupFRPTrafficClass : public TclClass {
 public:
        SupFRPTrafficClass() : TclClass("Application/Traffic/SupFRP") {}
        TclObject* create(int, const char*const*) {
                return (new SupFRP_Traffic());
        }
} class_SupFRP_traffic;
```

Appendix C. C++ CODE OF Application/Traffic/LList

C++ code of the sorted linked list TclObject, embedded into ns, is available in file "LList.cc".

"LList.cc"

```
#include <stdio.h>
#include <stdlib.h>
#include "trafgen.h"

class Double_LList : public TrafficGenerator{

        struct node {
                double item;
                int no;
                struct node *next;
        };

        public:
                Double_LList();
                command(int, const char*const*);
                virtual double next_interval(int&);
                struct node* Head;

void Insert(double item, int no)
// inserts the "item" to the linked list
{
        struct node *save, *temp1, *temp2;

        save=(struct node*)malloc(sizeof(struct node));
        save->item = item;
        save->no = no;

        if(Head==NULL){
                Head=save;
                Head->next=NULL;
        }
        else if (item<=Head->item){
                save->next=Head;
                Head=save;
        }
        else{
                temp2=Head;

                while((temp2->item<=item)&&(temp2->next!=NULL)){
                        temp1=temp2;
                        temp2=temp2->next;
                }
                if (temp2->item<=item){
                        temp1=temp2;
                        temp2=temp2->next;
                }
                temp1->next=save;
                save->next=temp2;
        }
}

void Delete(double item)
// removes "item" from the linked list
{
        struct node *save, *temp;

        if(Head==NULL) return;
        else if (item==Head->item){
                save=Head->next;
                free (Head);
```

```
                Head=save;
             }else{
                save=Head;

                while((save->item<item)&&(save->next!=NULL)){
                        temp=save;
                        save=save->next;
                }
                if (save->item==item){
                        temp->next=save->next;
                        free (save);
                }
        }
}

int LookUp (int itemno)
{
        struct node *save;

        save = Head;
        while ((save != NULL) && (save->no != itemno)) {
                save = save->next;
        }
        if ((save != NULL) && (save->no == itemno)) return 1;
        else return 0;
}

void DeleteList()
// removes the first element of the linked list
{
        struct node *save,*temp;

        save = Head;
        while (save != NULL) {
                temp = save;
                save = save->next;
                free(temp);
        }
        Head = NULL;

}

double TimeOfFirst()
// removes the first element of the linked list
{
        if(Head==NULL) return (0.0);
        else return (Head->item);
}

int NoOfFirst()
// removes the first element of the linked list
{
        if(Head==NULL) return (0);
        else return (Head->no);
}

void DeleteFirst()
// removes the first element of the linked list
{
        double item;
        struct node *save;
        if(Head==NULL) return;
        else {
                item = Head->item;
                save=Head->next;
                 free (Head);
                 Head=save;
                return;
        }
}
```

```
void DisplayList()
// prints the elements of the linked list to the screen
{
        struct node* save;
        save=Head;
        while(save!=NULL){
                printf("%f %d, ",save->item, save->no);
                save=save->next;
        }
       printf ("\n");
}
};


static class LinkedListClass : public TclClass {
 public:
        LinkedListClass() : TclClass("Application/Traffic/LList") {}
        TclObject* create(int, const char*const*) {
                return (new Double_LList());
        }
} LList_class;

Double_LList::Double_LList()
{
       Head = NULL;
}

double Double_LList::next_interval(int& size)
{
        double t = 0.0;
        size = size_;
        return(t);
}

int Double_LList::command(int argc, const char*const*argv)
{
       Tcl& tcl = Tcl::instance();
       if (argc >= 2){
               if (strcmp(argv[1], "DisplayList")==0){
                       DisplayList();
                       return TCL_OK;
               } else if (strcmp(argv[1], "DeleteFirst")==0){
                       DeleteFirst();
                       return TCL_OK;
               } else if (strcmp(argv[1], "DeleteList")==0){
                       DeleteList();
                       return TCL_OK;
               } else if (strcmp(argv[1], "LookUp")==0){
                       int itemno = atoi(argv[2]);
                       tcl.resultf("%d",LookUp(itemno));
                       return TCL_OK;
               } else if (strcmp(argv[1], "TimeOfFirst")==0){
                       tcl.resultf("%f",TimeOfFirst());
                       return TCL_OK;
               } else if (strcmp(argv[1], "NoOfFirst")==0){
                       tcl.resultf("%d",NoOfFirst());
                       return TCL_OK;
               } else if (strcmp(argv[1], "Insert")==0){
                       double time = atof(argv[2]);
                       int no = atoi(argv[3]);
                       Insert(time, no);
                       return TCL_OK;
               } else if (strcmp(argv[1], "Delete")==0){
                       double sender = atof(argv[2]);
                       Delete(sender);
                       return TCL_OK;
               }
       }
       return (Double_LList::command(argc,argv));
}
```

Appendix D. TRAFFIC MODELING

D.1. Importance of Traffic Modeling

The basic goal of telecommunications networking is to provide satisfactory service desired by users. The requirements of users can be shortly called as Quality of Service (QoS) requirements. In the design and engineering perspective of networking, there has always been a trade-off between resources and QoS requirements. While limited amount of resources is an obstacle to guarantee QoS, assurance of QoS requires more resources. To guarantee QoS, more optimization of resources must be achieved. Optimization needs accurate modeling. Modeling of a network requires a good traffic model, which can capture behavior of the real traffic. Thus, traffic modeling is the basic step for optimizing resources of a network.

D.2. Overview of Traffic Modeling

Three basic characteristics can be determined as the evaluation criteria for a traffic model. These are:

- **Accuracy:** To what extent the model is able to capture behavior of the traffic to be modeled?

The model should be able to cover the behavior of the real traffic.

- **Analytical tractability:** What is the computational complexity of the generation of traffic by using the model?

Tractability has been the biggest obstacle for creating traffic models. In recent years, there has been several models, proposed as accurate, but not tractable enough for real applications. On the other hand, there are several tractable traffic models but not accurate enough. So, there exists a trade-off between a model's accuracy and its analytical tractability. [12]

- **Flexibility:** To what extent the model is tunable for the generation of different kinds of traffic?

The value of a traffic model increases with respect to its number of adjustable parameters to generate different kinds of traffic.

D.2.1. Mathematical Description of Traffic

D.2.1.1. Point (Stochastic) Processes

The most usual ways to describe network traffic are the two point processes: "counting process" and "inter-arrival time process". [12]

In the counting processes, the network traffic is modeled as the number of packet arrivals during a predefined time slot, while the time between consecutive packet arrivals is used in the inter-arrival time processes. Recently, counting processes have been used more frequently.

What is a "counting process"? Say that *dN(t)* is a stochastic process representing packet arrivals. Define a counting process as $N(t) = \int_0^t dN(s)ds$ that is the number of packet arrivals in the interval $(0,t]$. [12]

How can a counting process describe traffic? Define a time series $X=\{X_n,\ n \in Z^+\}$ as $X_n = N[nT_s] - N[(n-1)T_s]$, where T_s is the duration of the nth interval. X_n refers to the number of packet arrivals during the nth interval. Figure D-1 indicates X_n on the time line. Here, X is assumed to have a mean $\mu=E[X_n]$ and variance $\sigma^2=Var[X_n]$.

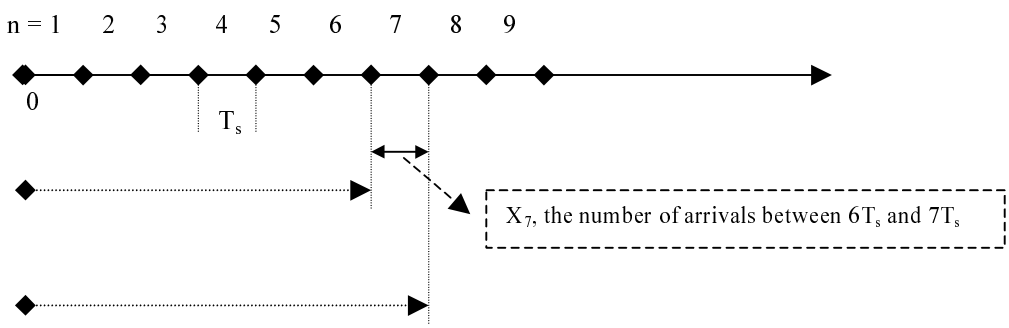

Figure D-1: Representation of the counting process on the time line.

D.2.1.2. Statistical Measures

As described in the previous section, point processes are the underlying basis for most traffic models. So, the characteristics of traffic models are adjusted by the statistical measures of the underlying point process.

Statistical measures of a point process are the first-order statistical measures such as average rate, the probability density function (pdf) of the number of packet arrivals during an interval and of the inter-arrival time. Basic second-order statistical measures are power spectral density (PSD), index of dispersion for counts (IDC), coincidence rate (CR), and covariance function (COV). We can briefly describe the second-order statistical measures as follows:

- **PSD:** Provides a measure of how the power in a process is concentrated in various frequency bands.[8]

- **CR:** Measures the correlation between pairs of arrivals with a specified time delay between them, regardless of the intervening arrivals. [4] Can be formulized as $G(\tau) = E[dN(t)dN(t+\tau)]$, where τ is the pre-mentioned time delay (number of intervening time slots). CR is related to auto-correlation function (ACF). CR and ACF identify the same characteristic of a traffic model. In fact, ACF is the CR, normalized with variance, σ^2. ACF can be formulized as:

$$r(k,T_s) = Co\,\mathrm{var}[X_n, X_{n+k}]/\sigma^2 = Co\,\mathrm{var}[X_n, X_{n+k}]/Co\,\mathrm{var}[X_n, X_n]$$
$$= E[X_n X_{n+k}]/\sigma^2$$

- **IDC:** Measures the variance related characteristics, and defined as the number of arrivals in a specified time window of width T_s divided by the mean number of arrivals. Formulization of it can be done as $F(T_s) = Var[N(T_s)]/E[N(T_s)]$.

- **COV:** Used for measuring the covariance between the number of arrivals in two counting time windows of width T_s and separation kT_s. Can be formulized as $C(k,T_s) = Cov(X_n, X_{n+k})$.

D.2.2. Some Important Types of Traffic Models

The classification of traffic models are mostly being done depending on the underlying point process. Actually, there are several point processes, which have been used for generation of traffic. Here, we are listing some of the important ones.

D.2.2.1. Poisson Process

Poisson models are the oldest traffic models. The basic idea is to generate the next-coming arrival after an independent exponentially distributed inter-arrival time. Poisson models have a geometrically decreasing ACF. ACF of Poisson models converges to zero when the intervening number of slots (also called "lag") goes to infinity. We can represent this mathematically as $r(k,T_s) \to 0$ when $k \to \infty$. This property of Poisson models is also described as having an ACF, which can be summed for all values of it from 0 to infinity. In other words, $\sum_{k=0}^{\infty} r(k,T_s)$ is finite and exists. This second description says that "ACF varies at zero after some point when *k* increases, so that its values can be summed to a finite number.".

This property of Poisson models, like several other traffic models, also represents short-range dependence property. In other words, the correlation between pairs of the number of arrivals in each interval vanishes when the number of intervening intervals between them increases. In long-range dependent models, ACF is not summable and never reaches to zero. This property of Poisson models caused them to be unsuccessful. Recently, Poisson models have been proven to be unsuccessful to capture the network traffic. [10]

D.2.2.2. Markov-Modulated Poisson Process (MMPP)

MMPP models are generated by a *k*-state Markovian system. Each state has a Poisson arrival rate represented with λ_i where $1<i<k$. In Markovian system, the generated traffic can be adjusted by changing the number of states, the mean rates of each state, and the transition probabilities between the states. Interrupted Poisson Process (IPP) model is an example of MMPP models. In IPP, there are two states, one of which has an arrival rate of zero. IPP models have been widely used to model bursty traffic such as VBR video and voice.

Although, MMPP models have a good tractability, they are also not able to catch the behavior of the network traffic accurately. MMPP models are also short-range dependent models.

In addition, superposition of MMPPs has been proposed to capture the behavior of long-range dependence. [3]

D.2.2.3. Fractal Point Process (FPP)

FPPs are the stochastic processes, which exhibit the power-law behavior. Power-law behavior is described as follows:

1. Scalability of ACF: $\lim_{k\to\infty}\left(\frac{r(k,T_s)}{k^{-(1-\alpha)}}\right)$ exists and is finite.
2. Scalability of IDC: $\lim_{T_s\to\infty}\left(\frac{F(T_s)}{T_s^{\alpha}}\right)$ exists and is finite.
3. Scalability of PSD: $\lim_{f\to\infty}\left(\frac{S(f)}{f^{-\alpha}}\right)$ exists and is finite.

For the conditions above, $0<\alpha<1$ must hold for all α. [7] In the above formulas, α is called as the "fractal exponent", and has the relation to the Hurst parameter, *H*, with the equation $H=(\alpha+1)/2$.

Basically, power-law indicates having the property of scalability. In other words, parts of the whole thing resemble to the general view of it. This property can be easily seen on clouds. Parts of the whole cloud, most of the time, resemble to the general view of it. For an FPP model to be fractal, at least two of the relations in Power-law must hold. Usually, ACF and IDC are checked for evaluating a traffic model.

Scalability of ACF means to have *long-range dependence* property, of IDC means to have *slowly decaying variance* property, and of PSD means to have *1/f noise* property.

In fact, ACF and IDC functions have been used to characterize the real network traffic. Several researches have shown that the behavior of the network traffic is scalable for

either ACF or IDC, or both. [12] Since traditional (such as Poisson and MMPP) models do not have scalability of ACF and IDC, they seem to be unsuccessful.

Which properties do the traditional models have? Traditional models have

- IDC of converging to a constant value.
- ACF of decaying geometrically.
- PSD bounded at the origin. [12]

D.3. Fractal Traffic Models

As it was mentioned in Section D.2.2.3, a point process has fractal property if it is satisfying at least two of the power-law rules. We can look at the FPPs closer by mentioning some important concepts related to fractal processes.

D.3.1. Self-Similarity

An FPP is self-similar if it has scalability depending on a parameter H, the Hurst parameter. The cloud-example (mentioned in Section D.2.2.3) is a very good example for explaining self-similarity. We can briefly say that a self-similar traffic has property of long-range dependence, slowly decaying variance, and *1/f* noise all at the same time or two of them.

D.3.2. Long-Range Dependence

In the previous sections, we stated that a long-range dependent traffic has a non-summable ACF, when lag k goes to infinity. This can also be interpreted as "there still exists some correlation between the traffic being generated and the traffic that was generated a very long time ago". In other words, although the traffic is randomly being generated, there is still some kind of relation.

D.3.3. Second-Order Self-Similarity

We can simply say that this is a more powerful self-similarity. In order to identify whether the traffic is second-order self-similar, at first we need to obtain the number of arrivals for each time slot, X_n. Then, we will look at the traffic from a distance, m times further. Mathematically, we can do that by averaging the m adjacent X_n values and constructing a new point process by these averages. If this final point process has the property of non-summable ACF, then we call the original traffic as second-order self-similar.

REFERENCES

1. Adas, A.: "Traffic Models in Broadband Networks", *IEEE Communications Magazine*, pp. 82-89, July 1997.

2. Addie, R. G. and Zukerman, M. and Neame, T. D.: "Broadband Traffic Modeling: Simple Solutions to Hard Problems", *IEEE Communications Magazine*, pp. 88-95, August 1998.

3. Andersen, A. T. and Nielsen, B. F.: "A Markovian Approach for Modeling Packet Traffic with Long Range Dependence", *IEEE Journal on Selected Areas in Communications*, Vol. 16, No. 5, pp. 719-732, June 1998.

4. Cox, D. R. and Lewis, P. A. W.: *The Statistical Analysis of Series of Events*, Methuen, London, 1966.

5. Crovella, M. E. and Bestavros A.: "Self-Similarity in World Wide Web Traffic: evidence and possible causes", *Proceedings of SIGMETRICS'96*, Philadelphia, PA, 1996.

6. Frost, V. S. and Melamed, B.: "Traffic Modeling for Telecommunications Network", *IEEE Communications Magazine*, pp. 70-81, March 1994.

7. Leland, W. E. and Taqqu, M. S. and Willinger, W. and Wilson, D.: "On the Self-Similar Nature of Ethernet Traffic (extended version)", *IEEE/ACM Transactions on Networking*, 2:1-15, 1994.

8. Papoulis, A.: *Probability, Random Variables, and Stochastic Processes*, McGraw Hill, New York, 3rd Edition, 1990.

9. Paxson, V.: "Empirically-Derived Analytic Models of Wide-Area TCP Connections", *IEEE/ACM Transactions on Networking*, 2(4), pp.316-336, August 1994.

10. Paxson, V. and Floyd, S.: "Wide-Area Traffic: The Failure of Poisson Modeling", *IEEE/ACM Transactions on Networking*, 3(3), pp. 226-244, June 1995.

11. Paxson, V. and Floyd, S.: "Why We Don't Know How To Simulate The Internet", *Proceedings of the 1997 Winter Simulation Conference*, Atlanta, GA, 1997.

12. Ryu, B. K.: "Fractal Network Traffic: From Understanding to Implications", PhD dissertation, Columbia University, 1996.