DISCRETE EVENT SIMULATION WITH APPLICATION TO COMPUTER COMMUNICATION SYSTEMS PERFORMANCE

Introduction to Simulation

Helena Szczerbicka[1], Kishor S. Trivedi[2] and Pawan K. Choudhary[2]
[1]University of Hannover, Germany ; [2]Duke University, Durham, NC

Abstract: As complexity of computer and communication systems increases, it becomes hard to analyze the system via analytic models. Measurement based system evaluation may be too expensive. In this tutorial, discrete event simulation as a model based technique is introduced. This is widely used for the performance/availability assessment of complex stochastic systems. Importance of applying a systematic methodology for building correct, problem dependent, and credible simulation models is discussed. These will be made evident by relevant experiments for different real-life problems and interpreting their results. The tutorial starts providing motivation for using simulation as a methodology for solving problems, different types of simulation (steady state vs. terminating simulation) and pros and cons of analytic versus simulative solution of a model. This also includes different classes of simulation tools existing today. Methods of random deviate generation to drive simulations are discussed. Output analysis, involving statistical concepts like point estimate, interval estimate, confidence interval and methods for generating it, is also covered. Variance reduction and speed-up techniques like importance sampling, importance splitting and regenerative simulation are also mentioned. The tutorial discusses some of the most widely used simulation packages like *OPNET MODELER* and *ns-2.* Finally the tutorial provides several networking examples covering TCP/IP, FTP and RED.

Key words: Simulation, Statistical Analysis, random variate, TCP/IP, OPNET MODELER and ns-2

In many fields of engineering and science, we can use a computer to simulate natural or man-made phenomena rather than to experiment with the real system. Examples of such computer experiments are simulation studies of congestion control in a network and competition for resources in a computer operating system. A simulation is an experiment to determine characteristics of a system empirically. It is a modeling method that mimics or emulates the behavior of a system over time. It involves generation and observation of artificial history of the system under study, which leads to drawing inferences concerning the dynamic behavior of the real system.

A *computer simulation* is a discipline of designing a model of an actual or theoretical system, executing the model (an experiment) on a digital computer, and statistically analyzing the execution output (see Fig. 1). The current state of the physical system is represented by state variables (program variables). Simulation program modifies state variables to reproduce the evolution of the physical system over time.

This tutorial provides an introductory treatment of various concepts related to simulation. In Section 1 we discuss the basic notion of going from the system description to its simulation model. In Section 2, we provide a broad classification of simulation models followed by a classification of simulation modeling tools/languages in Section 3. In Section 4 we discuss the role of probability and statistics in simulation while in Section 5 we develop several networking applications using the simulation tools OPNET MODELER and ns-2. Finally, we conclude in Section 6.

1. FROM SYSTEM TO MODEL

System can be viewed as a set of objects with their attributes and functions that are joined together in some regular interaction toward the accomplishment of some goal. Model is an abstract representation of a system under study. Some commonly used model types are:

1. Analytical Models. These employ mathematical formal descriptions like algebraic equations, differential equations or stochastic processes and associated solution procedures to solve the model. For example continuous time Markov chains, discrete time Markov chains, semi-Markov and Markov regenerative models have been used extensively for studying reliability/availability/performance and performability of computer and communication systems [1].
    1. Closed form Solutions: Underlying equations describing the dynamic behavior of such models can sometimes be solved in closed form if the model is small in size (either by hand or by such packages as Mathematica) or if the model is highly structured such as the Markov chain underlying a product-form queuing network [1].
    2. Numerical Methods: When the solution of an analytic model cannot be obtained in a closed form, then computational procedures are used to

numerically solve analytical models using packages such as SHARPE [2] or SPNP [3]

2. Simulation models: Employ methods to "run" the model so as to mimic the underlying system behavior; no attempt is made to solve the equations describing system behavior as such equations may be either too complex or not possible to formulate. An artificial history of the system under study is generated based on model assumptions. Observations are collected and analyzed to estimate the dynamic behavior of the system being simulated. Note that simulation provides a model-based evaluation method of system behavior but it shares its experimental nature with measurement-based evaluation and as such needs the statistical analysis of its outputs.

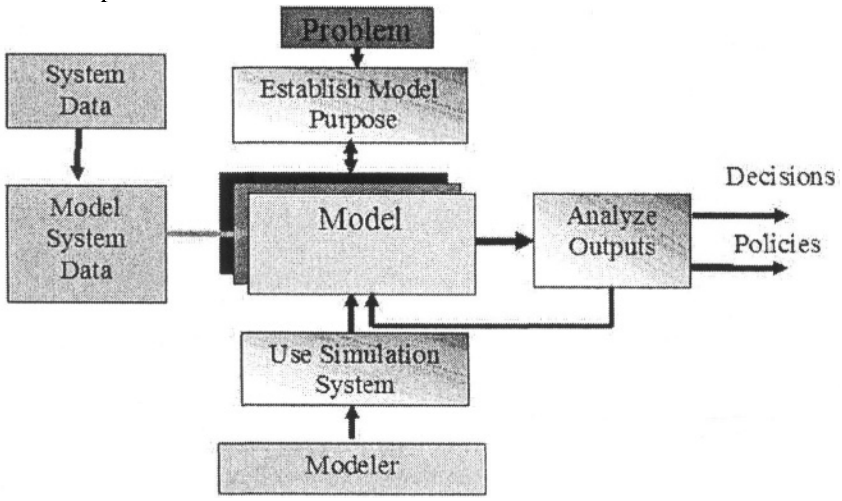


Figure 1. Simulation based problem solving

Simulation or analytic models are useful in many scenarios. As the real system becomes more complex and computing power becomes faster and cheaper, modeling is being used increasingly for the following reasons [4]:

1. If the system is unavailable for measurement the only option available for its evaluation is to use a model. This can be the case if system is being designed or it is too expensive to experiment with the real system
2. Evaluation of system under wide variety of workloads and network types (or protocols).

3. Suggesting improvement in the system under investigation based on knowledge gained during modeling.
4. Gaining insight into which variables are most important and how variables interact.
5. New polices, decision rules, information flows can be explored without disrupting ongoing operations of the real system.
6. New hardware architectures, scheduling algorithms, routing protocols, reconfiguration strategies can be tested without committing resources for their acquisition/implementation.

While modeling has proved to be a viable and reliable alternative to measurements on the real system, the choice between analytical and simulation is still a matter of importance For large and complex systems, analytic model formulation and/or solution may require making unrealistic assumptions and approximations. For such systems simulation models can be easily created and solved to study the whole system more accurately. Nevertheless, many users often employ simulation where a faster analytic model would have served the purpose.

Some of difficulties in application of simulation are:

1. Model building requires special training. Frequently, simulation languages like Simula [5], Simscript [6], Automod [7], Csim [8], etc are used. Users need some programming expertise before using these languages.
2. Simulation results are difficult to interpret, since most simulation outputs are samples of random variables. However most of the recent simulation packages have inbuilt output analysis capabilities to statistically analyze the outputs of simulation experiments.
3. Though the proper use of these tools requires a deep understanding statistical methods and necessary assumptions to assert the credibility of obtained results. Due to a lack of understanding of statistical techniques frequently simulation results can be wrongly interpreted [9].
4. Simulation modeling and analysis are time consuming and expensive. With availability of faster machines, development in parallel and distributed simulation [10, 11] and in variance reduction techniques such as importance sampling [12, 13, 14], importance splitting [15, 16, 17] and regenerative simulation [18], this difficulty is being alleviated.

In spite of some of the difficulties, simulation is widely used in practice and the use of simulation will surely increase manifold as experimenting with real systems gets increasingly difficult due to cost and other reasons. Hence it is important for every computer engineer (in fact, any engineer) to be familiar with the basics of simulation.

2. CLASSIFICATION OF SIMULATION MODELS

Simulation models can be classified according to several criteria [19]:

1. *Continuous vs. Discrete:* Depending upon the way in which state variables of the modeled system change over time. For example concentration of a substance in a chemical reactor changes in a smooth, continuous fashion like in a fluid flow whereas changes in the length of a queue in a packet switching network can be tracked at discrete points in time. In a *discrete event simulation* changes in the modeled state variable are triggered by scheduled events [20].
2. *Deterministic vs. stochastic*
   This classification refers to type of variables used in the model being simulated. The choice of stochastic simulation makes it experimental in nature and hence necessitates statistical analysis of results.
3. *Terminating vs. Steady state:* A terminating simulation is used to study the behavior of a system over a well-defined period of time, for example for the reliability analysis of a flight control system over a designated mission time. This corresponds to transient analysis put in the context of analytic models. Whereas steady state simulation corresponds to the steady state analysis in the context of analytic models. As such, we have to wait for the simulation system output variables to reach steady state values. For example, the performance evaluation of a computer or networking system is normally (but not always) is done using steady state simulation. Likewise, the availability analysis is typically carried out for steady state behavior.
4. *Synthetic or distribution driven vs. Trace driven:* A time-stamped sequence of input events is required to drive a simulation model. Such an event trace may already be available to drive the simulation hence making it a trace driven simulation. Examples are cache simulations for which many traces are available. Similarly, traces of packet arrival events (packet size, etc.) are first captured by using a performance measurement tool such as tcpdump. Then these traces are used as input traffic to the simulation. Lots of traces are freely available on Web. One of Internet traces archive is http://ita.ee.lbl.gov. Alternatively, event traces can be synthetically generated. For the synthetic generation, distributions of all inter-event times are assumed to be known or given and then random deviates of the corresponding distributions are used as the time to next event of that type. We will show how to generate random deviates of important distributions such as the exponential, the Weibull and the Pareto distribution. The distribution needed to drive such distribution

driven simulations may have been obtained by statistical inference based on real measurement data.

5. *Sequential vs. Distributed simulation:* Sequential simulation processes events in a non-decreasing time order. In distributed simulation a primary model is distributed over heterogeneous computers, which independently perform simulations locally. The challenge is to produce such a final overall order of events, which is identical with the order that would be generated when simulating the primary model on a single computer, sequentially. There is extensive research in parallel and distributed simulation [10,11].

The rest of this tutorial is concerned with sequential, distribution driven discrete event simulation.

3. CLASSIFICATION OF SIMULATION TOOLS

Simulation tools can be broadly divided into three basic categories:

1. *General Purpose Programming Language (GPPL):* - C, C++, Java are some of the languages which have the advantage of being readily available. These also provide a total control over software development process. But the disadvantage is that model construction takes considerable time. Also it doesn't have support for any control of a simulation process. Furthermore, generation of random deviates for various needed distributions and the statistical analysis of output will have to be learned and programmed.
2. *Plain Simulation Language (PSL)* - SIMULA, SIMSCRIPT II.5 [6], SIMAN, GPSS, JSIM, SILK are some of the examples. Almost all of them have basic support for discrete event simulation. One drawback is that they are not readily available. There is also the need for programming expertise in a new language.
3. *Simulation Packages (SPs)-* like OPNET MODELER [21], ns-2 [22], CSIM [8], COMMNET III, Arena [23], Automod [7], SPNP [3] etc. They have a big advantage of being user-friendly, with some of them having graphical user interface. They provide basic support for discrete event simulation (DES) and statistical analysis as well as several application domains like TCP/IP networks. This ensures that model construction time is shorter. Some simulation tools like OPNET MODELER also provide user an option of doing analytical modeling of the network. The negative side is that they are generally expensive, although most of them have free academic version for research. Like PSL, SPs require some expertise in new language/environment, and they tend to be less flexible than the PSLs.

Information about a variety of available simulation tools can be found at: http://www.idsia.ch/~andrea/simtools.html

4. THE ROLE OF STATISTICS IN SIMULATION

There are two different uses of statistical methods and one use of probabilistic methods in distribution driven simulations. First, the distributions of input random variables such as inter-arrival times, times to failure, service times, times to repair, etc. need to be estimated from real measurement data. Statistical inference techniques for parameter estimation and fitting distributions are covered in [1] and will be reviewed in the tutorial. Using random number generators, probabilistic methods of generating random deviates are then used to obtain inter-event times and drive the simulation. Once again this topic is covered in [1] and will be reviewed. Simulation runs are performed as computer experiments in order to determine the characteristics of its output random variables. A single simulation run produces sample of values of an output variable over time. Statistical techniques are employed here to examine the data and to get meaningful output from the experiment. Also they are used to define the necessary length of simulation (the size of the sample), characteristics of output variables like mean value and some assessments regarding an accuracy of results. Two principal methods, independent replication and the method of batch means, will be discussed.

In the following subsections we discuss random variate generation methods and the statistical analysis of simulation output.

4.1 Random Variate generation

In this section we describe methods of generating random deviates of any arbitrary distribution, assuming a routine to generate uniformly distributed random numbers is available. The distribution can be either continuous or discrete. Most of the simulation packages like OPNET MODELER, ns -2 and CSIM have built-in routines for generating random variates. But still knowledge of random variate generation is necessary to more accurately model the real world problem especially when available built-in generators in simulation packages do not support the needed distribution. Some of the popular methods for generating variates are [1,4]:

1. *Inverse Transform:* In this method the following property is used: if X is a continuous random variable with the CDF F, than the new random variable $Y=F(X)$ is uniformly distributed over the interval (0, 1). Thus to

generate a random deviate x of X first a random number u from a uniform distribution over (0, 1) is generated and then the F is inverted. $F^{-1}(u)$ gives the required value of x. This can be used to sample from exponential, uniform, Weibull, triangular, as well as empirical and discrete distributions. It is most useful when the inverse of the CDF, *F(.)* can be easily computed. Taking the example of exponential distribution (see Eq.1) given u drawn from *U(0,1)*, generate x drawn from *exponential distribution* (see Eq. 2).

$$F(x) = 1 - e^{-\lambda x}, \quad x \geq 0 \tag{1}$$

$$x = -\frac{\ln(1-u)}{\lambda} \tag{2}$$

Some distribution which can be easily inverted are exponential, Weibull, Pareto and log-logistic.

For Weibull distribution whose distribution is given by Eq. (3).

$$F_X(x) = 1 - e^{-\lambda x^{\alpha}} \tag{3}$$

The random variate is generated using Eq. (4)

$$x = \left(\frac{-\ln(1-u)}{\lambda}\right)^{\frac{1}{\alpha}} \tag{4}$$

Similarly Pareto distribution is given by Eq. (5)

$$\begin{aligned} F_X(x) &= 1 - \left(\frac{k}{x}\right)^{\alpha} \quad && x \geq k \\ &= 0 && x \leq k \end{aligned} \tag{5}$$

The random variate is generated using Eq. (6)

$$x = \frac{k}{(1-u)^{\frac{1}{\alpha}}} \tag{6}$$

For Rayleigh distribution given by

$$\begin{aligned} F_X(x) &= 1 - e^{-x^2/2\sigma^2} \quad & x \geq 0 \\ &= 0 & \textit{otherwise} \end{aligned} \tag{7}$$

The random variate can be generated using:

$$x = \sqrt{-2\sigma^2 \ln(1-u)} \tag{8}$$

Similarly for Log-Logistic Distribution given by

$$F_X(x) = 1 - \frac{1}{1+(\lambda t)^{\kappa}} \tag{9}$$

The random deviate is generated using

$$x = \frac{1}{\lambda}\left(\frac{u}{1-u}\right)^{\kappa} \tag{10}$$

Random variate of (discrete) Bernoulli distribution with parameter (1-*q*) can also be generated by the inverse transform technique. The CDF is given by

$$\begin{aligned} F_X(x) &= 0 \quad & x < 0 \\ &= q & 0 \leq x < 1 \\ &= 1 & x \geq 1 \end{aligned} \tag{11}$$

The inverse function for Bernoulli distribution becomes

$$\begin{aligned} F^{-1}(u) &= 0 \quad & 0 < u \leq q \\ &= 1 & q < u \leq 1 \end{aligned} \tag{12}$$

Now by generating u between (0, 1) we can obtain a random deviate of the Bernoulli distribution by Eq. (12).

For *Hyperexponential distribution* given by

$$F(t) = \sum_i \alpha_i (1 - e^{-\lambda_i t}) \quad t \geq 0 \tag{13}$$

Random variate for Hyperexponential can be generated in two steps. Consider for example a three stage Hyperexponential distribution with parameters α_1, α_2, α_3 and λ_1, λ_2 λ_3. First a uniform random number u is generated and like Eq. (12) the following inverse function is generated:

$$\begin{aligned} F^{-1}(u) &= 1 \quad 0 < u \le \alpha_1 \\ &= 2 \quad \alpha_1 < u \le \alpha_1 + \alpha_2 \\ &= 3 \quad \alpha_1 + \alpha_2 < u \le 1 \end{aligned} \tag{14}$$

Now if $u < \alpha_1$, $F^{-1}(u) = 1$, and the variate is then generated from exponential distribution which occur with probability α_1.

$$x = -\frac{\ln(1-u)}{\lambda_1} \tag{15}$$

Similarly if $F^{-1}(u)=2$, Hyperexponential variate is given as

$$x = -\frac{\ln(1-u)}{\lambda_2} \tag{16}$$

Similarly depending upon the output of Bernoulli variate, Hyperexponential variate can be generated. Note that this example was for $k=3$, but it can be easily extended to $k=n$ stages.

2. *Convolution Method:* This is very helpful in such cases when the random variable Y can be expressed as a sum of other random variables that are independent and easier to generate than Y. Let

$$Y = X_1 + X_2 + \ldots + X_k \tag{17}$$

Taking an example of Hypoexponential case, random variable X with parameters $(\mu_1, \ldots \mu_k)$ is sum of k independent exponential RV's Y_i with mean $1/\mu_i$. For example, a 2-stage hypoexponential distribution is given by

$$F(t) = 1 - \frac{\lambda_2}{\lambda_2 - \lambda_1} e^{-\lambda_1 t} + \frac{\lambda_1}{\lambda_2 - \lambda_1} e^{-\lambda_2 t} \tag{18}$$

From the inverse transform technique, each Y_i is generated using Eq. (2) and their sum is the required result. Note that Erlang is a special case of the

Hypoexponential distribution when all the k sequential phases have identical distribution. Random variate for hypoexponential distribution is given as Eq. (19).

$$x = \sum_{i=1}^{k} -\frac{1}{\mu_i}\ln(1-u_i) \tag{19}$$

Binomial random variable is known to be the sum of n independent and identically distributed Bernoulli random variables hence generating n Bernoulli random variates and adding, this sum will result in a random variate of the Binomial. If $(x_1, x_2, .. x_n)$ are the Bernoulli random variates given by Eq.(12) and let y be Binomial random variate then,

$$y = \sum_{i=1}^{n} x_i \tag{20}$$

3. Direct Transform of Normal Distribution: Since inverse of a normal distribution cannot be expressed in closed form we cannot apply inverse transform method. The CDF is given by:

$$\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt \tag{21}$$

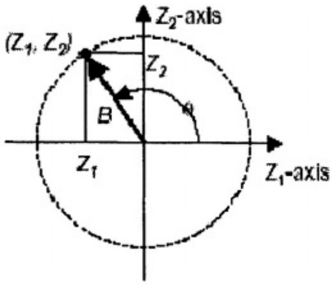


Figure 2. Polar representation

In order to derive a method of generating a random deviate of this distribution, we use a property of the normal distribution that relates it to the Rayleigh distribution. Assume that Z_1 and Z_2 are independent standard normal random variables. Then the square root of their sum $Z_1^2 + Z_2^2$ is known to have the Rayleigh distribution [1] for which we know how to generate its random deviate.

Now in polar coordinates, the original normal random variable can be written as:

$$Z_1 = B\cos\theta \qquad B^2 = Z_1^2 + Z_2^2$$
$$Z_2 = B\sin\theta \tag{22}$$

Using the inverse transform technique (see Eq. 8) we have:

$$b = \sqrt{-2\ln(1-u)} \tag{23}$$

Next we generate a random value of θ to finally get two random deviates of the standard normal:

$$\begin{aligned} z_1 &= \sqrt{-2\ln(1-u_1)}\cos(2\pi u_2) \\ z_2 &= \sqrt{-2\ln(1-u_1)}\sin(2\pi u_2) \end{aligned} \tag{24}$$

4.2 Output Analysis

Discrete-event simulation takes random numbers as inputs that result in each set of study to produce different set of outputs. Output analysis is done to examine data generated by a simulation. It can be used to predict the performance/reliability/availability of a system or compare attributes of different systems. While estimating some measure of the system, say θ, simulation will generate an estimator $\hat{\Theta}$ of θ due to presence of random variability. The precision of the estimator $\hat{\Theta}$ will depend upon its variance. Output analysis will help in estimating this variance and also in determining number of observations needed to achieve a desired accuracy. Phenomenon like *sampling error* and *systematic error* influence how well an estimate $\hat{\theta}$ will approximate θ. Sampling error is introduced due to random inputs and dependence or correlation among observations. Systematic errors occur due to dependence of the observations on initially chosen state and initial condition of the system.

4.2.1 Point and Interval Estimates

Estimation of parameter by a single number from the output of a simulation is called point estimate. Let random variables $\{Y_1, Y_2 .. Y_n\}$ are set of observations obtained after simulation. Then a common point estimator for parameter θ is given by Eq. (25).

$$\hat{\Theta} = \frac{1}{n}\sum_{i=1}^{n} Y_i \tag{25}$$

The point estimator $\hat{\Theta}$ is also a random variable and called unbiased if its expected value is θ i.e.

$$E[\hat{\Theta}] = \theta \tag{26}$$

If $E[\hat{\Theta}] = \theta + b$, then b is called bias of the point estimator.

The confidence interval provides an interval or range of values around the point estimate [1]. Confidence interval is defined as

$$P(A < \hat{\Theta} < B) = 1 - \alpha \textit{ where A and B are functions of parameter } \theta \tag{27}$$

For a single parameter, such as the mean, the standard deviation, or probability level, the most common intervals are two sided (i.e., the statistic is between the lower and upper limit) and one sided (i.e., the statistic is smaller or larger than the end point). For the simultaneous estimation of two or more parameters, a confidence region, the generalization of a confidence interval, can take on arbitrary shapes [24, 25].

4.2.2 Terminating vs. Steady State simulation

Output analysis is discussed here for two classes of simulations: terminating simulation and steady state simulation.

Terminating simulation: This applies to the situation wherein we are interested in the transient value of some measure, e.g., channel utilization after 10 minutes of system operation or the transient availability of the system after 10 hours of operation. In these cases each simulation run is conducted until the required simulated time and from each run a single sample value of the measure is collected. By making m independent simulation runs, point and interval estimates of the required measure are obtained using standard statistical techniques. In both the cited examples, each simulation run will provide a binary value of the measure and hence we use the inference procedure based on sampling from the Bernoulli random variable [1]. Yet another situation for terminating simulation arises when the system being modeled has some absorbing states. For instance we are interested in estimating the mean time to failure of a system then form each simulation run a single value is obtained and multiple independent runs are used to get the required estimate. In this case, we could use inference procedure assuming sampling from the exponential or the Weibull distribution [1].

Steady-State Simulation: In this case, we can in principle make independent runs but since the transient phase needs to be thrown away and since it can be long, this approach is wasteful. Attempt is therefore made to get the required statistics from a long single run. The first problem encountered then is to estimate the length of the transient phase. The second problem is the dependence in the resulting sequence. [1] talks about how to estimate the correlation in the sequence first using independent runs. Instead of using independent runs, we can divide a single sequence into first the transient phase and then a batch of steady state runs. Then there are dependencies not only within a batch but also across batches.

The estimator random variable of the mean measure, θ to be estimated is given by Eq. (28) where n is number of observations.

$$\Theta = \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} Y_i \tag{28}$$

This value should be independent of the initial conditions. But in real system, simulation is stopped after some number of observations n have been collected. The simulation run length is decided on the basis of how large the bias in the point estimator is, the precision desired or resource constraint for computing.

4.2.3 Initialization Bias

Initial conditions may be artificial or unrealistic. There are methods that reduce the point-estimator bias in steady state simulation. One method is called *intelligent initialization* that involves initialization of simulation in a state that is more representative of long-run conditions. But if the system doesn't exist or it is very difficult to obtain data directly from the system, any data on similar systems or simplified model is collected.

The second method involves dividing the simulation into two phases. One of them is called the *initialization phase* from time *0* to T_0 and the other is called the *data-collection phase* from T_0 to $T_0 + T_E$.

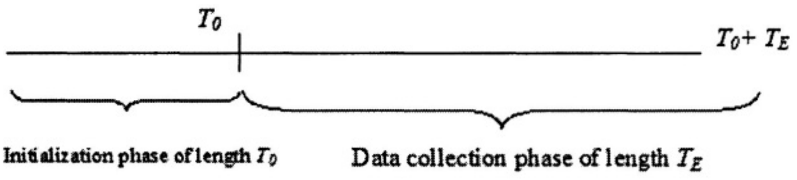

T_0
T_0+T_E
Initialization phase of length T_0
Data collection phase of length T_E

Figure 3. Initialization and Data Collection phase

The choice of T_0 [26, 27, 28] is important as system state at time T_0 will be more representative of steady state behavior than at the time of original initial conditions (i.e., at time $t=0$). Generally T_0 is taken to be more than five times T_E [29].

4.2.4 Dealing with Dependency [1]

Successive values of variables monitored from a simulation run exhibit dependencies, such as high correlation between the response times of consecutive requests to a file server. Assume that the observed quantities are dependent random variables, $(Y_1, Y_2 \ldots Y_m)$ having index invariant mean μ and variance σ^2. The sample mean is given by

$$\bar{Y} = \sum_{i=1}^{m} \frac{Y_i}{m} \qquad (29)$$

Sample mean is unbiased point estimator of population mean but variance of sample mean is not equal to σ^2/n. Taking sequence to be wide sense stationary the variance is given by Eq.(30)

$$Var[\overline{Y}] = \frac{\sigma^2}{m} + \frac{2}{m}\sum_{i=1}^{m-1}\left(1 - \frac{i}{m}\right)K_i \tag{30}$$

$$So\ mVar[\bar{Y}] = \sigma^2 a \quad where\ a = 1 + 2\sum_{i=1}^{\infty}\frac{K_i}{\sigma^2} \tag{31}$$

The statistic $\frac{\bar{Y}-\mu}{\sigma\sqrt{a/m}}$ approaches standard normal distribution as *m* approaches infinity. Therefore an approximate $100(1-\alpha)\%$ confidence interval becomes

$$\overline{Y} \pm \sigma z_{\alpha/2}\sqrt{\frac{a}{m}} \tag{32}$$

The need to estimate $\sigma^2 a$ can be avoided using *Replication method.* It is used to estimate point-estimator variability. In this method, simulation experiment is replicated *m* times with *n* observations each. If initial state is chosen randomly for all *m* observations, the result will be independent of each other. But the *n* observations within each experiment will be dependent. Let the sample mean and sample variance of the j^{th} experiment be given by $\overline{\overline{Y}}(j)$ *and* $S^2(j)$ respectively. From individual sample means, point estimator of population mean is given by

$$\overline{Y} = \frac{1}{m}\sum_{j=1}^{m}\overline{\overline{Y}}(j) = \frac{1}{mn}\sum_{j=1}^{m}\sum_{i=1}^{n}Y_i(j) \tag{33}$$

All $\bar{Y}(1), \bar{Y}(2)..\bar{Y}(m)$, are independent and identically distributed (*i.i.d*) random variables. Assume that the common variance of $\bar{Y}(j)$, is denoted by v^2. The estimator of the variance v^2 is given by

$$\overline{V^2} = \frac{1}{m-1}\sum_{j=1}^{m}[\bar{Y}(j) - \bar{Y}]^2 \tag{34}$$

And $100(1\text{-}\alpha)\%$ confidence interval for θ, is approximately given by

$$\bar{y} - t_{\alpha/2,m-1}\frac{v}{\sqrt{m}} \le \theta \le \bar{y} + t_{\alpha/2,m-1}\frac{v}{\sqrt{m}} \tag{35}$$

where '*t*' represents *t-student* distribution with *(m-1)* degree of freedom.

4.2.5 Method of Batch Means

One major disadvantage of the replication method is that initialization phase data from each replication is wasted. To address the issue, we use a design based on a single, long simulation run divided into contiguous segments (or batches), each having length *n*. The sample mean Y_r of r^{th} segment is then treated as an individual observation. This method called the *method of batch means,* reduces the unproductive portion of simulation time to just one initial stabilization period. But the disadvantage is the set of sample means are not statistically independent and usually the estimator is biased. Estimation of the confidence interval for a single run method can be done following the same procedure as done for replication method. We just replace r^{th} replication in independent replication by the r^{th} batch. *Method of batch means* is also called *single run method.*

4.2.6 Variance Reduction Techniques

Variance reduction techniques help in obtaining greater precision of simulation results (smaller confidence interval) for the same number of simulation runs, or in reducing the number of runs required for the desired precision. They are used to improve the efficiency and accuracy of the simulation process.

One of frequently used technique is importance sampling [12, 13, 14]. In this approach the stochastic behavior of the system is modified in such a way that some events occur more often. This helps in dealing with rare events scenarios. But this modification causes model to be biased, which can be removed using the likelihood ratio function. If carefully done, the variance of the estimator of the simulated variable is smaller than the original implying reduction in the size of the confidence interval. Other techniques include importance splitting [15, 16, 17] and regenerative simulation [18].

Some of the other methods that are used to speed up of simulations are parallel and distributed simulation [10, 11].

To summarize, before generating any sound conclusions on the basis of the simulation-generated output data, a proper statistical analysis is required. The simulation experiment helps in estimating different measures of the system. The statistical analysis helps in acquiring some assurance that these estimates are sufficiently precise for the proposed use of the model. Depending on the initial conditions and choice of run length terminating simulations or steady-state simulations can be performed. Standard error or a confidence interval can be used to measure the precision of point estimators.

5. SOME APPLICATIONS

In this section we discuss some of the simulation packages like OPNET MODELER [21] and ns-2 [22]. We also discuss Network Animator (NAM) [30] which generates graphs and animation in ns-2. OPNET MODELER and ns-2 are application oriented simulation packages. While OPNET MODELER uses GUI extensively for configuring network, ns-2 is OTcl Interpreter and uses code in OTCL and C++ to connect network.

5.1 OPNET MODELER

This simulation package uses an object oriented approach in formulating the simulation model. One of the powers of OPNET MODELER comes from its simplicity that is due to its menu-driven graphical user interface. Some of the application areas where OPNET can be used are:

1. For network (LAN/WAN) planning. It has built-in libraries for all the standard TCP/IP protocol and applications including IP Quality of Service (QoS), Resource Reservation Protocol (RSVP) etc.
2. It supports wireless and satellite communication schemes and protocols.
3. It can be used for microwave and fiber-optic based network management.

4. Can be used for evaluating new routing algorithms for routers, switches and other connecting devices, before plugging them physically in the network.

Features of OPNET MODELER that make it a comprehensive tool for simulation are:

1. It uses hierarchical model structure. The model can be nested within layers.
2. Multiple scenarios can be simulated simultaneously and results can be compared. This is very useful when deciding the amount of resource needed for a network configuration. This also helps in pinpointing which system parameter is affecting the system output most.
3. OPNET MODELER gives an option of importing traffic patterns from an external source.
4. It has many of built-in graphing tools that make the output analysis easier.
5. It has the capability of automatically generating models with live network information (topobgy, device configurations, traffic flows, network management data repositories, etc.).
6. OPNET MODELER has animation capabilities that can help in understanding and debugging the network.

5.1.1 Construction of Model in OPNET MODELER [19]

OPNET MODELER allows to model network topologies using three hierarchical levels:

1. *Network Level:* It is the highest level of modeling in OPNET MODELER. Topologies are modeled using network level components like routers, hosts and links. These network models can be dragged and dropped from object palette, can be chosen from OPNET MODELER menu which contain numerous topologies like star, bus, ring, mesh etc. or can be imported from a real network by collecting network topology information. (See Fig. 4)
2. *Node level:* It is used to model internal structure of a network level component. It captures the architecture of a network device or system by depicting the interactions between functional elements called modules. Modules have the capability of generating, sending and receiving packets from other modules to perform their functions within the node. They typically represent applications, protocol layers and physical resources ports, buses and buffers. Modules are connected by “streams” that can be a packet stream, a statistic stream or an association stream. As the name suggests packet stream represents packet flows between modules, a

statistic stream is used to convey statistics of the between modules. An association stream is used for logically associating different modules and it does not carry any information. (See Fig. 5)

3. *Process Level:* It uses a Finite State Machine (FSM) description to support specification at any level of detail of protocols, resources, applications, algorithms and queuing policies. States and transitions graphically define the evolution of a process in response to events. Each state of the process model contains C/C++ code, supported by an extensive library for protocol programming. Actions taken in a state are divided into enter executives and exit executives which are described by Proto-C (See Fig. 6).

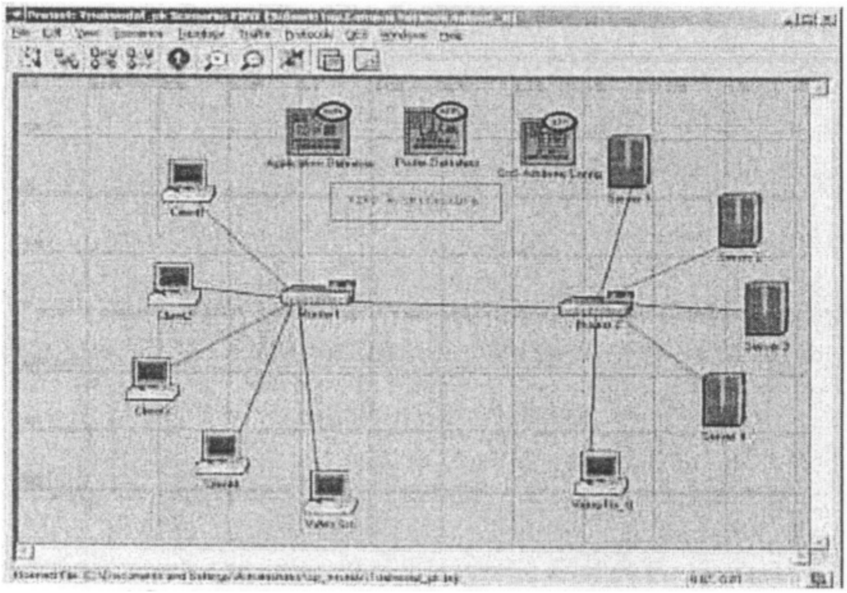


Figure 4. Screen Shot for Network level Modeling. Detail of an FIFO architecture.

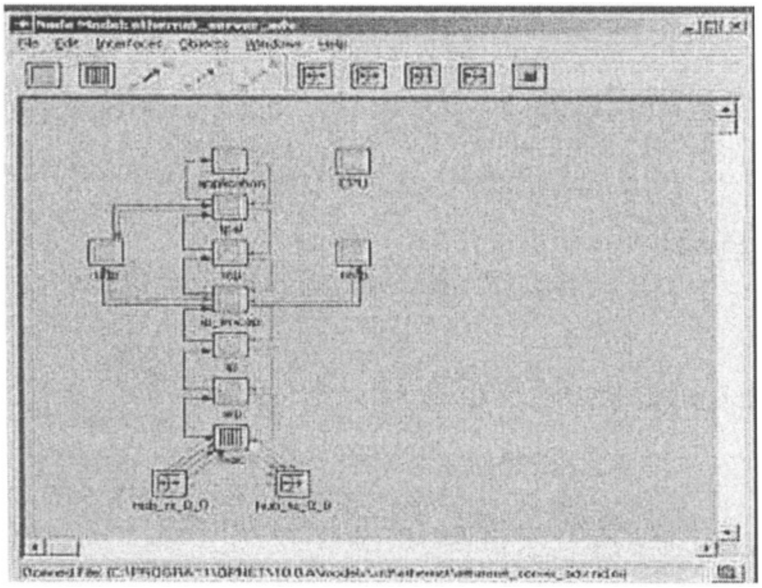


Figure 5. Screen Shot for Node level Modeling. Detail of server using Ethernet link.

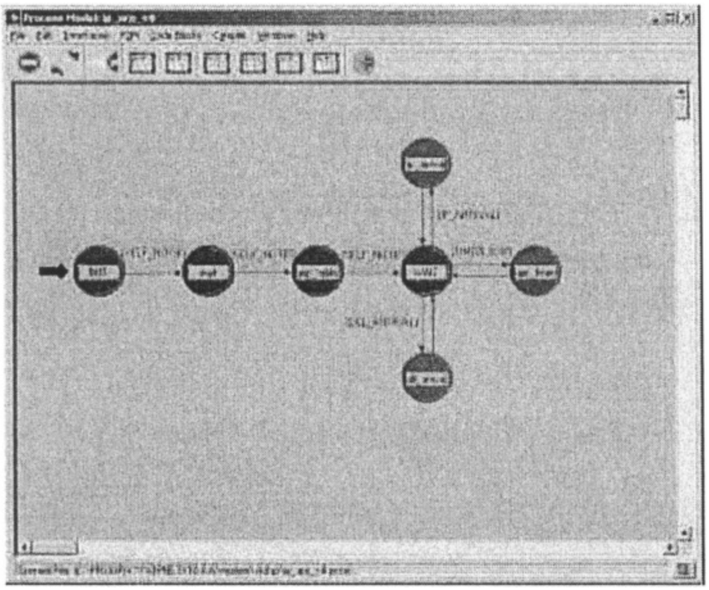


Figure 6. Screen Shot for Process Level Modeling. Details of an IP Node

5.1.2 Example- Comparison of RED vs. FIFO with Tail-drop

The normal behavior of router queues on the Internet is called tail-drop. Tail-drop works by queuing the incoming messages up to a certain queue length and then dropping all traffic that comes when the queue is full. This could be unfair, and may lead to many retransmissions. The sudden burst of drops from a router that has reached its buffer size will cause a delayed burst of retransmits, which will over fill the congested router again.

RED (Random Early Detection) [31] is an active queue management scheme proposed for IP routers. It is a router based congestion avoidance mechanism. RED is effective in preventing congestion collapse when TCP window size is configured to exceed network storage capacity. It reduces congestion and end-to-end delay by controlling the average queue size. It drops packets randomly with certain probability even before the queue gets full (see Fig. 7).

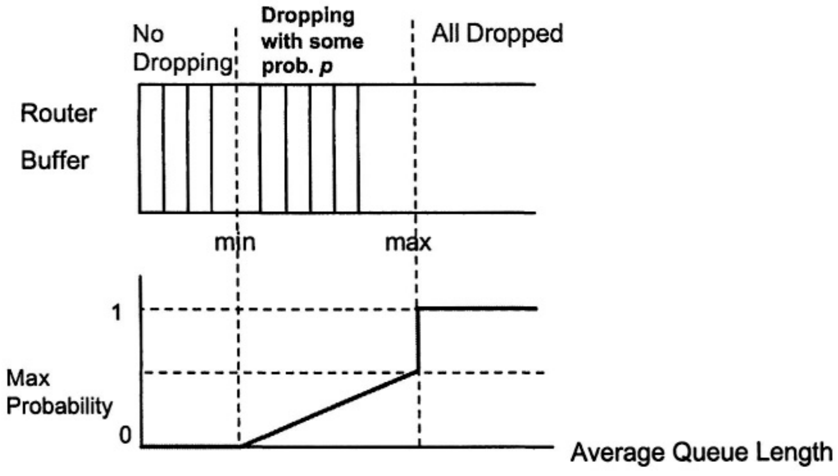


Figure 7. Active Queue Management by RED

In this example we compare the performance of RED and FIFO with Tail Drop. The network for the example consists of two routers and five clients with their corresponding servers. The capacity of link between two routers is taken to be 2.048Mbps. All other links have capacity of 100Mbps fast Ethernet. Clearly the link between Router 1 and Router 2 is the bottleneck. Our goal is the buffer occupancy at Router 1 for the two schemes.

Model is constructed using network level editor of OPNET MODELER. Hosts and servers are joined together with the help of routers and switches that are simply dragged and dropped from object palette. Attributes are assigned for various components. Configuration parameters are assigned with the help of utility objects. Some of the utility objects like Application configuration, Profile configuration and QoS configuration are shown in following screen shots. The application chosen is video conferencing with each of the clients having different parameters set- Heavy, Streaming Multimedia, Best Effort, Standard and with Background Traffic. Incoming and outgoing frame sizes are set to 1500 bytes.

All the screen shots from (Fig.8-11) were for the FIFO scheme. OPNET MODELER has a facility for generating duplicate scenario using which we generate the model for the RED scheme. The applications and profile configuration for RED remains the same as in the FIFO case. Only the QoS attributes configuration needs to be changed (See Fig. 12). RED parameters are set as in Table 1. After this, discrete event simulation is run and different statistics like buffer size for Router 1 are collected. All five clients are sending video packets having length 1500 bytes with interarrival time and service time derived from *constant* distribution.

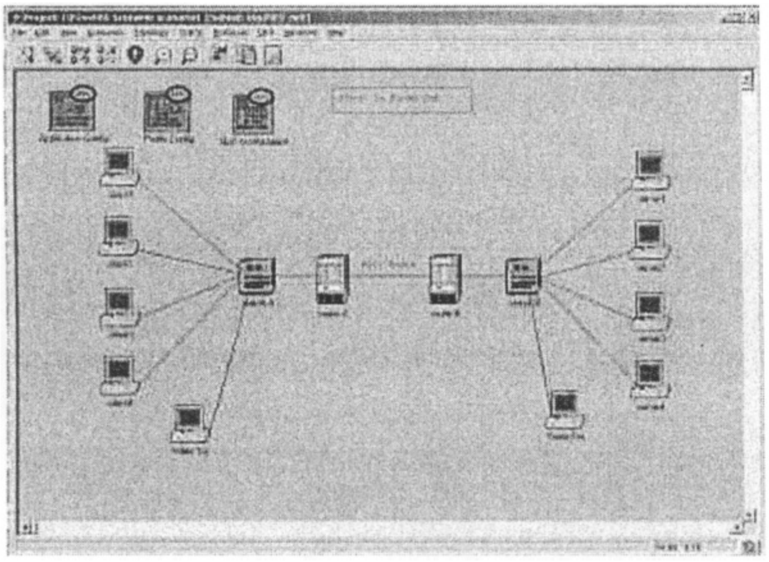
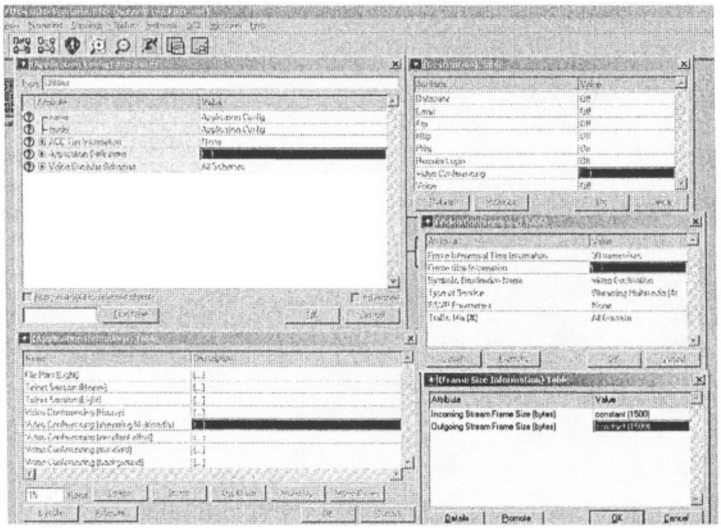


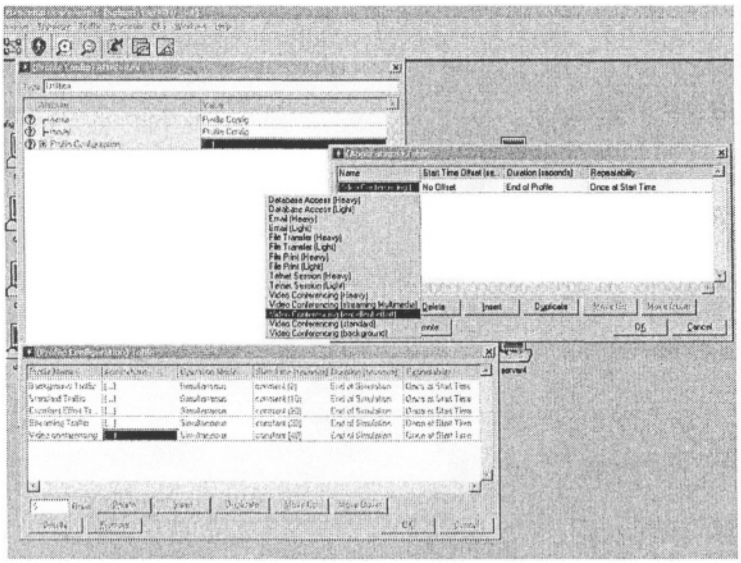
Figure 8. Network level modeling for FIFO arrangement. 5 clients are connected to 2 switches and 2 routers. They are connected with 5 servers.



(Frame Size Information) Table

Attribute	Value
Incoming Stream Frame Size (bytes)	constant (1500)
Outgoing Stream Frame Size (bytes)	constant (1500)

Figure 9. Application Configuration- Different window showing assignment of parameter to video conferencing (streaming Multimedia)



Name	Start Time Offset (se...	Duration (seconds)	Repeatability
Video Conferencing (...)	No Offset	End of Profile	Once at Start Time

Database Access (Heavy)
Database Access (Light)
Email (Heavy)
Email (Light)
File Transfer (Heavy)
File Transfer (Light)
File Print (Heavy)
File Print (Light)
Telnet Session (Heavy)
Telnet Session (Light)
Video Conferencing (Heavy)
Video Conferencing (streaming Multimedia)
Video Conferencing (excellent effort)
Video Conferencing (standard)
Video Conferencing (background)

Figure 10. Profile Configuration -Different Screen shot for entering Video conferencing (various modes) to each of the client.

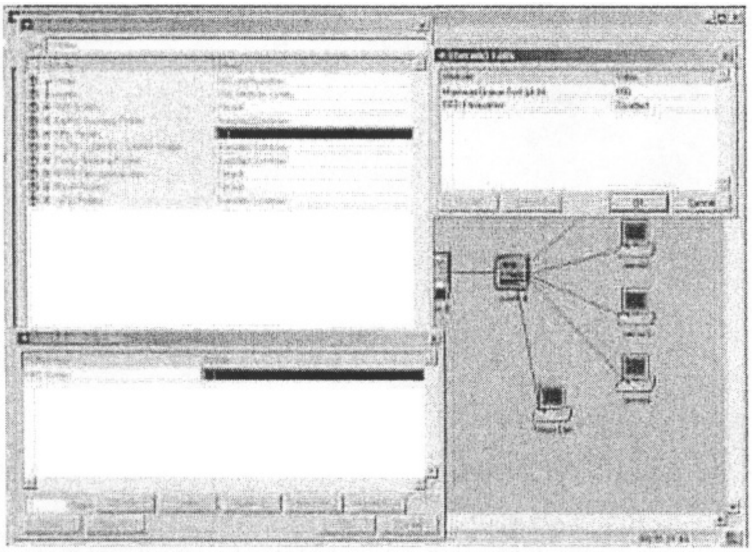
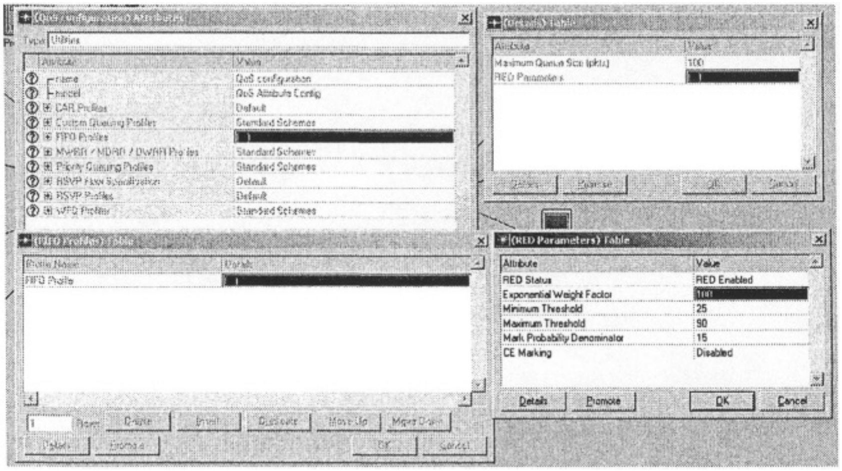


Figure 11 QoS Attribute Configuration- This shows that FIFO is selected with queue size of 100 and RED is disabled.

Figure 13 shows the result of simulation where the buffer sizes for the two cases are plotted as a function of time. Notice that both buffers using RED and FIFO taildrop behave similarly when link utilization is low. After 40 seconds, when utilization jumps to almost 100 %, congestion starts to build at router buffer that uses FIFO taildrop. In case of active queue management (RED case), the buffer occupancy remains low and it never saturates. In fact buffer occupancy is much smaller than that of FIFO during the congestion period.

***Table 1*. RED parameters**

Parameters	Values
min_{th}	25
max_{th}	90
max_p	0.15
w_q	0.01



Type: Utilities

Attribute	Value
name	QoS configuration
model	QoS Attribute Config
CAR Profiles	Default
Custom Queuing Profiles	Standard Schemes
FIFO Profiles	
MWRR / MDRR / DWRR Profiles	Standard Schemes
Priority Queuing Profiles	Standard Schemes
RSVP Flow Specification	Default
RSVP Profiles	Default
WFQ Profiles	Standard Schemes

(FIFO Profiles) Table

Profile Name	Details
FIFO Profile	

Attribute	Value
Maximum Queue Size (pkts)	100
RED Parameters	

(RED Parameters) Table

Attribute	Value
RED Status	RED Enabled
Exponential Weight Factor	100
Minimum Threshold	25
Maximum Threshold	90
Mark Probability Denominator	15
CE Marking	Disabled

Figure 12. QoS Attribute configuration for RED case. Application and Profile configuration remains same as FIFO

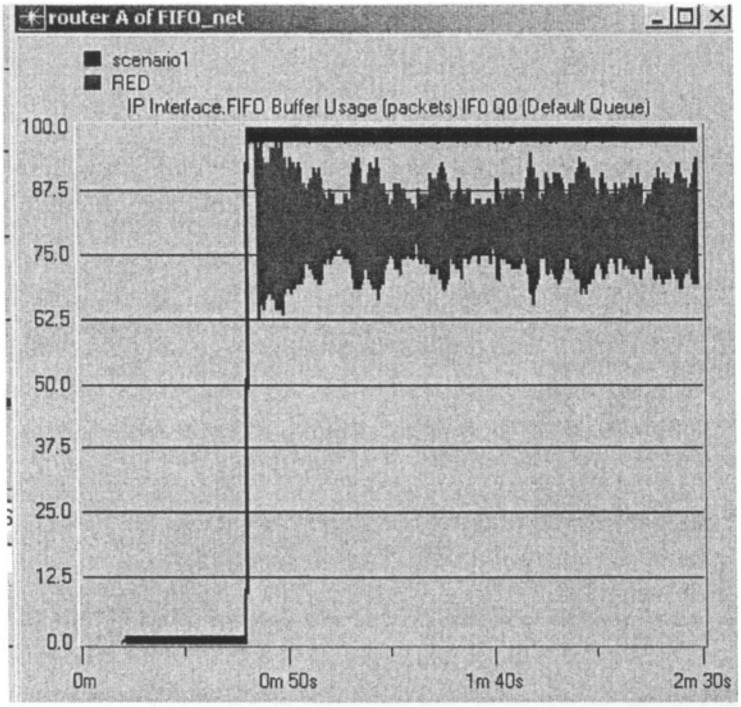


Figure 13. RED vs. FIFO for buffer occupancy

5.2 ns-2 and NAM

Network Simulator (ns) started as a variant of REAL network simulator [32] with the support of DARPA and several companies/universities. It has evolved and is now known as ns-2. It is a public domain simulation package in contrast to OPNET MODELER which is a commercial package. Like OPNET MODELER, it also uses an object oriented approach towards problem solving. It is written in C++ and object oriented TCL [33]. All network components and characteristics are represented by classes. ns-2 provides a substantial support for simulation of TCP, routing and multicast protocols over wired and wireless networks. Details about ns -2can be found from http://www.isi.edu/nsnam/ns/.

5.2.1 Overview and Model construction in ns-2

ns-2 provides canned sub-models for several network protocols like TCP and UDP, router queue management mechanism like Tail Drop, RED, routing algorithms like Dijkstra [34] and traffic source behavior like telnet, FTP, CBR etc. It contains simulation event scheduler and a large number of network objects, such as routers, links etc. which are interconnected to form a network. The user needs to write an OTc1 script that initiates an event scheduler, sets up the network topology using network objects and tells traffic sources when to start and stop transmitting packets through the event scheduler.

5.2.2 Network Components (ns objects)

Objects are built from a hierarchical C++ class structure. As shown in Fig. 14, all objects are derived from class NsObject. It consists of two classes- connectors and classifiers. Connector is an NsObject from which links like queue and delay are derived. Classifiers examine packets and forward them to appropriate destinations. Some of the most frequently used objects are:

1. *Nodes:* This represents clients, hosts, router and switches. For example, a node n1 can be created by using command *set n1 [$ns node].*
2. *Classifiers:* It determines the outgoing interface object based on source address and packet destination address. Some of the classifiers are Address classifier, Multicast classifier, Multipath classifier and Replicators.

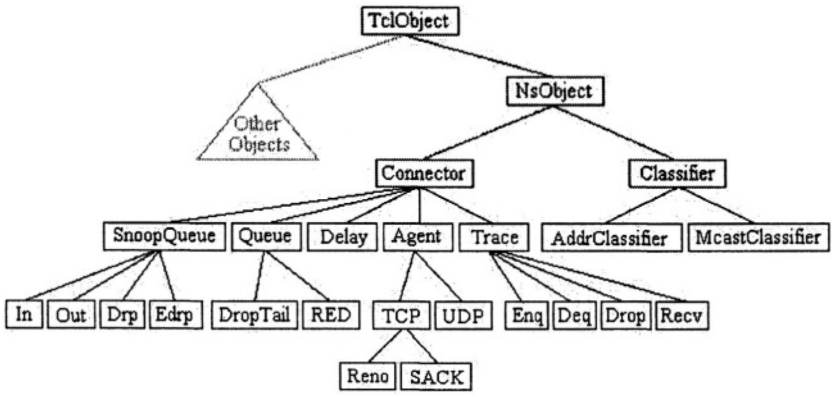


Figure 14. Class Hierarchy (Taken from "NS by example" [35])

3. *Links:* These are used for connection of nodes to form a network topology. A link is defined by its head which becomes its entry point, a reference to main queue element and a queue to process packets dropped at the link. Its format is *$ns <type>-link <nodel> <node2> <bandwidth> <delay> <queue-type>*.
4. *Agents:* these are the transport end-points where packets originate or are destined. Two types of agents are TCP and UDP. ns-2 supports wide variants of TCP and it gives an option for setting ECN bit specification, congestion control mechanism and window settings. For more details about Agent specification see [14]
5. *Application:* The major types of applications that ns-2 supports are traffic generators and simulated applications. *Attach-agent* is used to attach application to transport end-points. Some of the TCP based applications supported by ns-2 are Telnet and FTP.
6. *Traffic generators:* In cases of a distribution driven simulation automated traffic generation with desired shape and pattern is required. Some of traffic generators which ns-2 provide are Poisson, On-OFF, Constant bit rate and Pareto On-OFF.

5.2.3 Event Schedulers

Event scheduler is used by network components that simulate packet-handling delay or components that need timers. The network object that issues an event will handle that event later at a scheduled time. Event scheduler is also used to schedule simulated events, such as when to start a Telnet application, when to finish a simulation, etc. ns -2 has real-time and non-real-time event schedulers. Non-real-time scheduler can be implemented either by a list, heap or a calendar.

5.2.4 Data collection and Execution

ns-2 uses tracing and monitoring for data collection. Events such as a packet arrival, packet departure or a packet drop from a link/queue are recorded by tracing. Since tracing module does not collect data for any specific performance metrics, it is only useful for debugging and verification purposes. The command in ns-2 for activating tracing is *$ns trace-all <tracefile>*.

Monitoring is a better alternative to tracing where we need to monitor a specific link or node. Several trace objects are created which are then inserted into a network topology at desired places. These trace objects collect different performance metrics. Monitoring objects can also be written in C++ (Tracing can written in OTcl only) and inserted into source or sink functions.

After constructing network model and setting different parameters, ns-2 model is executed by using *run* command. *($ns run).*

5.2.5 Network Animator

NAM is an animation tool that is used extensively along with ns -2. It was developed in LBL. It is used for viewing network simulation packet traces and real world packet traces. It supports packet level animation that shows packets flowing through the link, packets being accumulated in the buffer and packets dropping when the buffer is full. It also supports topology layout that can be rearranged to suit user's needs. It has various data inspection tools that help in better understanding of the output. More information about NAM can be found at http://www.isi.edu/nsnam/ns/tutorial/index.html.

5.2.6 Example- RED Analysis

Objective: Studying the dynamics of current and average queue size in a RED queue.

In this example we have taken six nodes. All links are duplex in nature with their speed and delay shown in the Fig.16. In this example FTP application is chosen for both source nodes *n1* and *n3*. Node *n2* is the sink node. The window size for TCP application is taken to be 15. RED buffer can hold a maximum of 30 packets in this example. First FTP application starts from 0 till 12 seconds and second FTP application starts from 4 to 12 seconds. For output data collection, monitoring feature is used. NAM is used to display graph of buffer size vs. time

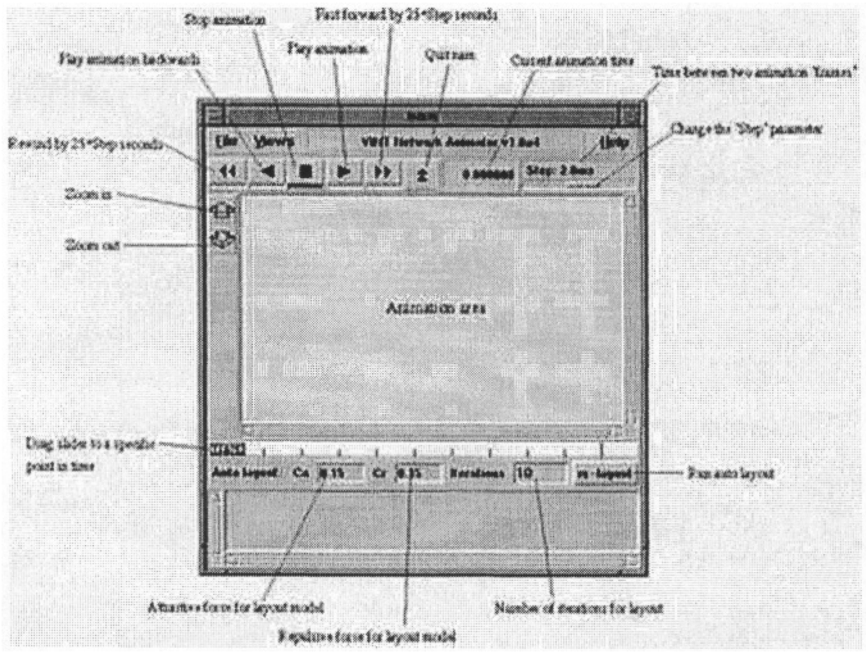

Stop animation
Fast forward by 25*Step seconds
Play animation backwards
Play animation
Quit nam
Current animation time
Time between two animation "frames"
Rewind by 25*Step seconds
Change the 'Step' parameter
Zoom in
Zoom out
Animation area
Drag slider to a specific point in time
Run auto layout
Attractive force for layout model
Number of iterations for layout
Repulsive force for layout model

Figure 15. NAM Window (picture taken from "Marc Greis Tutorial" [36])

In this File Transfer Protocol has been simulated over TCP network. By default FTP is modeled by simulating the transfer of a large file between two endpoints. By large file we mean that FTP keeps on packetizing the file and sending it continuously between the specified start and stop times. The number of packets to be sent between start and stop time can also be specified using *produce* command. Traffic is controlled by TCP which performs the appropriate congestion control and transmits the data reliably. The buffer size is taken to be 14000 packets and router parameters are given in table 2. The output shows the buffer occupancy at router *r1,* for instantaneous and average value case. From the graph it becomes clear that during higher utilization also, RED helps in reducing congestion.

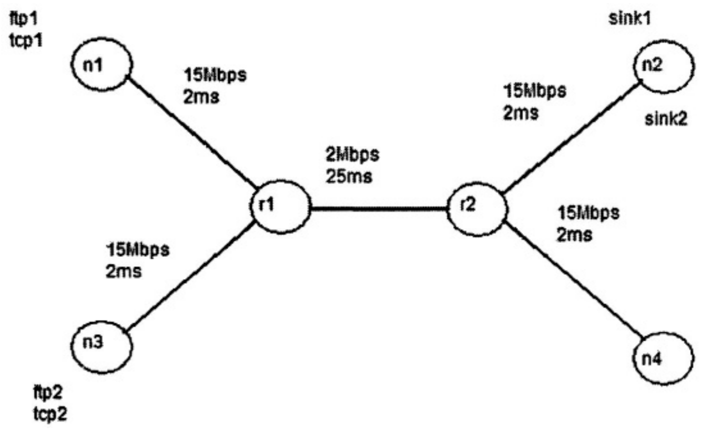


Figure 16. Network connection for an RED configuration

Table 2. RED parameters

Parameters	Values
minth	6000 packets
maxth	12000 packets
max_p	0.15
w_q	0.01

6. SUMMARY

This tutorial discussed simulation modeling basics and some of its applications. Role of statistics in different aspects of simulation was discussed. This includes random variate generation and the statistical analysis of simulation output.

Different classes of simulation were discussed. Simulation packages like OPNET MODELER and ns-2 along with some applications were discussed in the last section. These packages are extensively used in research and industry for real-life applications.

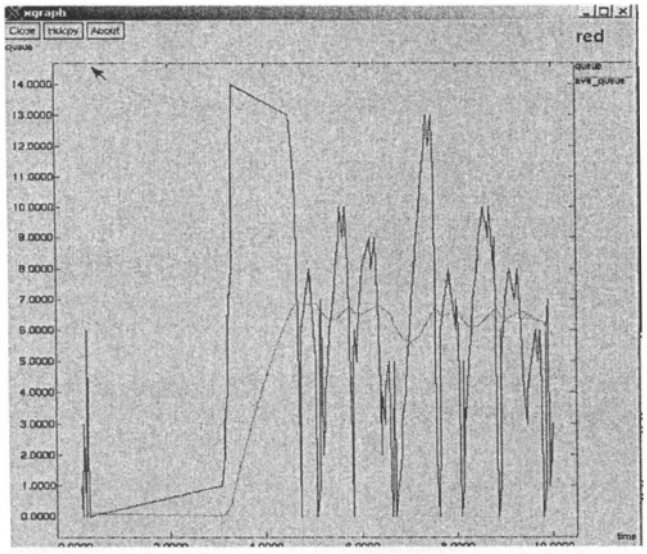


Figure 17. Plot of RED Queue Trace path

REFERENCES

1. Kishor S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications,* (John Wiley and Sons, New York, 2001).
2. Robin A. Sahner, Kishor S. Trivedi, and Antonio Puliafito, *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package,* (Kluwer Academic Publishers, 1996).
3. Kishor S. Trivedi, G. Ciardo, and J. Muppala, SPNP: Stochastic Petri Net Package, *Proc. Third Int. Workshop on Petri Nets and Performance Models (PNPM89),* Kyoto, pp. 142 - 151, 1989.
4. J. Banks, John S. Carson, Barry L. Nelson and David M. Nicol, *Discrete –Event System Simulation,* Third Edition, (Prentice Hall, NJ, 2001).
5. Simula Simulator; http://www.isima.fr/asu/ .
6. Simscript II.5 Simulator; http://www.caciasl.com/ .
7. AUTOMOD Simulator; http://www.autosim.com/.
8. CSIM 19 Simulator; http://www.mesquite.com/.

9. K, Pawlikowski, H. D. Jeong and J. S. Lee, On credibility of simulation studies of telecommunication networks, *IEEE Communication Magazine, 4(1),* 132-139, Jan 2002.
10. H. M. Soliman, A.S. Elmaghraby, M.A. El-sharkawy, Parallel and Distributed Simulation System: an overview, *Proceedings of IEEE Symposium on Computer and Communications,* pp 270-276, 1995.
11. R.M. Fujimoto, Parallel and Distributed Simulation System, Simulation Conference, *Proceeding of winter, Vol. 1,* 9-12 Dec. 2001.
12. B. Tuffin, Kishor S. Trivedi, Importance Sampling for the Simulation of Stochastic Petri Nets and Fluid Stochastic Petri Nets, *Proceeding of High Performance Computing,* Seattle, WA, April 2001.
13. G. S. Fishman, *Concepts Algorithms and Applications,* (Springer-Verlag, 1997).
14. P.W. Glynn and D.L. Iglehart, Importance Sampling for stochastic Simulations, *Management Science, 35(11),* 1367-1392, 1989.
15. P. Glasserman, P. Heidelberger, P. Shahabuddin, and T. Zajic, Splitting for rare event simulation: analysis of simple cases. In *Proceedings of the 1996 Winter Simulation Conference* edited by D.T. Brunner J.M. Charnes, D.J. Morice and J.J. Swain editors, pages 302-308, 1996.
16. P. Glasserman, P. Heidelberger, P. Shahabuddin, and T. Zajic, A look at multilevel splitting. In *Second International conference on Monte-Carlo and Quasi- Monte Carlo Methods in Scientific Computing* edited by G. Larcher, H. Niederreiter, P. Hellekalek and P. Zinterhof, Volume 127 of Lecture Series in Statistics, pages 98-108, (Springer-Verlag, 1997).
17. B. Tuffin, Kishor S. Trivedi, Implementation of Importance Splitting techniques in Stochastic Petri Net Package," in *Computer performance evaluation: Modeling tools and techniques; 11th International Conference; TOOLS 2000, Schaumburg, Il. USA,* edited by B. Haverkort, H. Bohnenkamp, C. Smith, Lecture Notes in Computer Science 1786, (Springer Verlag, 2000).
18. S. Nananukul, Wei-Bo-Gong, A quasi Monte-Carlo Simulation for regenerative simulation, *Proceeding of 34th IEEE conference on Decision and control,* Volume 2, Dec. 1995.
19. M. Hassan, and R. Jain, *High Performance TCP/IP Networking: Concepts, Issues, and Solutions,* (Prentice-Hall, 2003).
20. Bernard Zeigler, T. G. Kim, and Herbert Praehofer, *Theory of Modeling and Simulation,* Second Edition, (Academic Press, New York, 2000).
21. OPNET Technologies Inc.; http://www.opnet.com/.
22. Network Simulator; http://www.isi.edu/nsnam/ns/.
23. Arena Simulator; http://www.arenasimulation.com/.

24. Liang Yin, Marcel A. J. Smith, and K.S. Trivedi, Uncertainty analysis in reliability modeling, In *Proc. of the Annual Reliability and Maintainability Symposium, (RAMS),* Philadelphia, PA, January 2001.
25. Wayne Nelson. *Applied Life Data Analysis* John Wiley and Sons, New York, 1982.
26. L.W. Schruben, Control of initialization Bias in multivariate simulation response, *Communications of the Association for Computing machinery,* 246-252, 1981.
27. A.M. Law and J.M. Carlson, A sequential Procedure for determining the length of steady state simulation, *Operations Research,* Vol. 27, pp-131-143, 1979.
28. Peter P. Welch, *Statistical analysis of simulation result, Computer performance Modeling Handbook,* edited by Stephen S. Lavenberg, Academic Press, 1983
29. W.D. Kelton, Replication Splitting and Variance for simulating Discrete Parameter Stochastic Process, *Operations Research Letters,* Vol.4,pp-275-279, 1986.
30. Network Animator.; http://www.isi.edu/nsnam/nam/.
31. S. Floyd, and V. Jacobson, Random early detection gateways for congestion avoidance, *IEEE/ACM Transactions on Networking,* Volume1, Issue 4 , Aug. 1993 Pages:397– 413.
32. REAL network simulator.; http://www.cs.cornell.edu/skeshav/real/overview.html
33. OTCL-Object TCL extensions.; http://bmrc.berkeley.edu/research/cmt/cmtdoc/otcl/
34. E. W. Dijkstra, A Note on Two Problems in Connection with Graphs. *Numerische Math. 1,* 269-271, 1959.
35. Jay Cheung, Claypool, NS by example.; http://nile.wpi.edu/NS/
36. Marc Greis, Tutorial on ns.; http://www.isi.edu/nsnam/ns/tutorial/