Discrete Event Simulation

Modelação e Desempenho de Redes e Serviços

Prof. Amaro de Sousa (asou@ua.pt)

DETI-UA, 2024/2025

Discrete event simulation

A discrete event simulation models the operation of a system whose state changes with events that happen in discrete time instants:

- each event forces a change of either a system state value and/or of a variable value;
- between consecutive events, the system remains in the same state;
- thus, the simulation can directly jump in time from one event to the next event.

Elements of a discrete event simulator:

(1) State variables: variables that describe the state of the system at any time instant

(2) Statistical counters: variables that store the statistical data required to determine the performance of the system

(3) Simulation clock: variable indicating the current simulated time instant (simulated time $\neq$ computation time)

(4) Events: types of occurrences that change either the system state and/or the statistical counters

(5) Event list: list of future events, their time instants and associated parameters

Besides these elements, additional supporting variables might be required.

Basic structure of a discrete event simulator

A discrete event simulator is mainly composed by the following steps:

(1) Initialization of the state variables, the statistical counters and the event list with the first event(s).

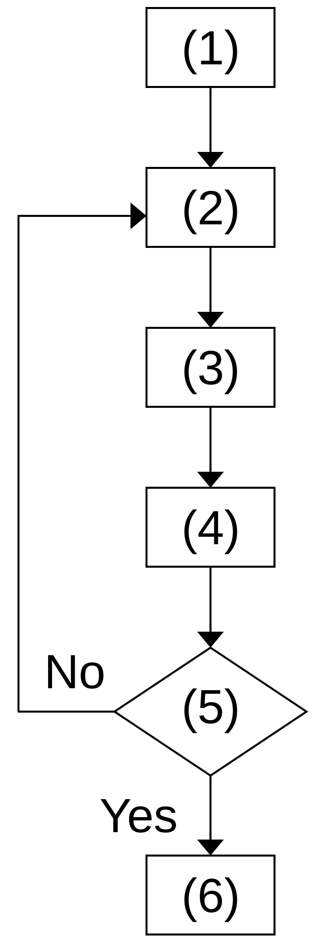
(2) Determination of the next event from the event list.

(3) Update of the simulation clock to the time instant of the next event and removal of the event from the event list.

(4) Execution of all actions associated to the event (generation of new events, update of state variables and/or statistical counters).

(5) Check if the simulation must end; if not, return to Step (2).

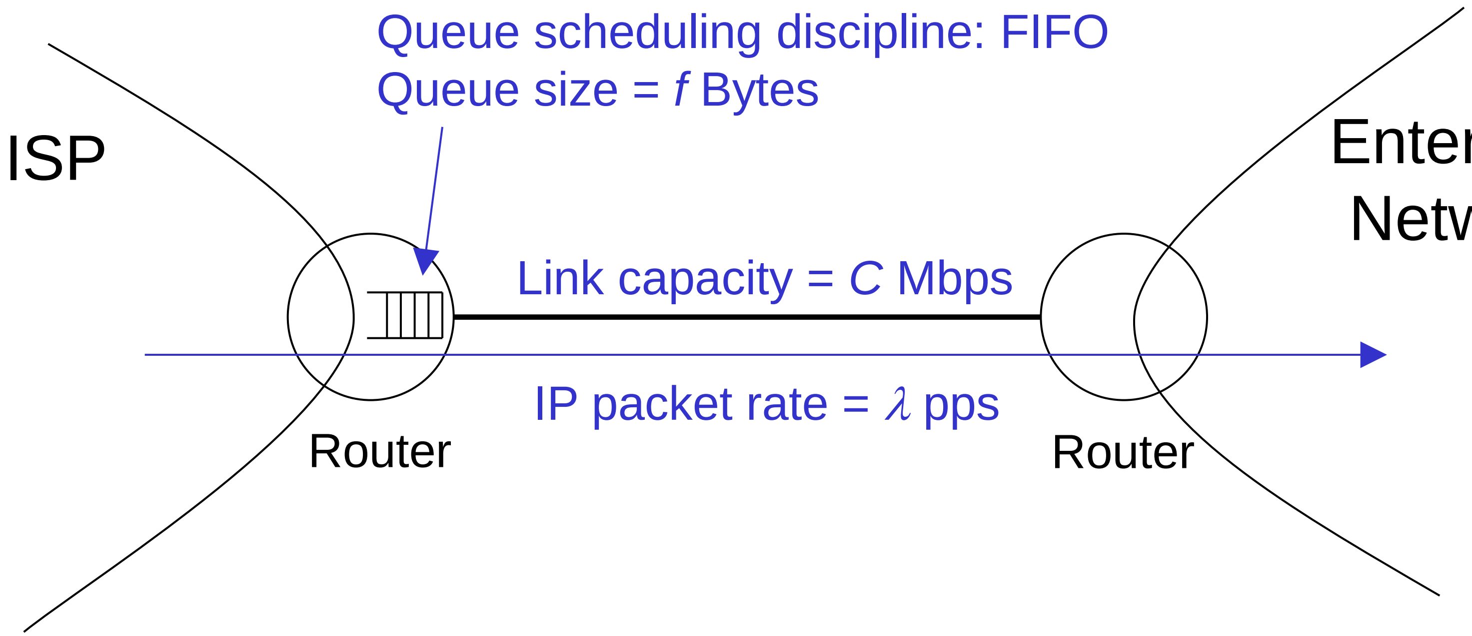
(6) Update of the statistical counters (if needed) and determination of the performance parameters.



Example: performance of a downstream link from an ISP to a client enterprise

Consider the event driven simulation of a point-to-point IP link between an enterprise router and its Internet Service Provider (ISP).

Let us consider the downstream direction, *i.e.*, from ISP to the company (usually, the direction with highest traffic load).



Example: performance of a downstream link from an ISP to a client enterprise

Input parameters of simulation:

λ – packet rate, in packets per second (pps)

C – connection capacity, in Mbps

f – queue size, in Bytes

P – total number of transmitted packets of a simulation run

Stopping criteria of simulation:

Time instant when the link finishes the transmission of the P^{th} packet

Performance parameters to be estimated by the simulation:

PL – Packet Loss (%)

APD – Average Packet Delay (milliseconds)

MPD – Maximum Packet Delay (milliseconds)

TT – Transmitted Throughput (Mbps)

Example: performance of a downstream link from an ISP to a client enterprise

Events:

ARRIVAL – the arrival of a packet

DEPARTURE – the end of transmission of a packet

State variables:

STATE – binary variable indicating if the connection is free (i.e., not being used) or busy with the transmission of a packet

QUEUEOCCUPATION – occupation of the queue, in number of bytes, with the queued packets

QUEUE – matrix with (i) a number of rows equal to the number of queued packets and (ii) 2 columns where each column has the size and the arriving time instant of each packet in the queue

Example: performance of a downstream link from an ISP to a client enterprise

Statistical Counters:

TOTALPACKETS – number of packets arrived to the system

LOSTPACKETS – number of packets dropped due to buffer overflow

TRANSPACKETS – number of transmitted packets

TRANSBYTES – sum of the bytes of the transmitted packets

DELAYS – sum of the delays of the transmitted packets

MAXDELAY – maximum delay among all transmitted packets

Performance parameters (at the end of the simulation):

PL = 100 × LOSTPACKETS / TOTALPACKETS

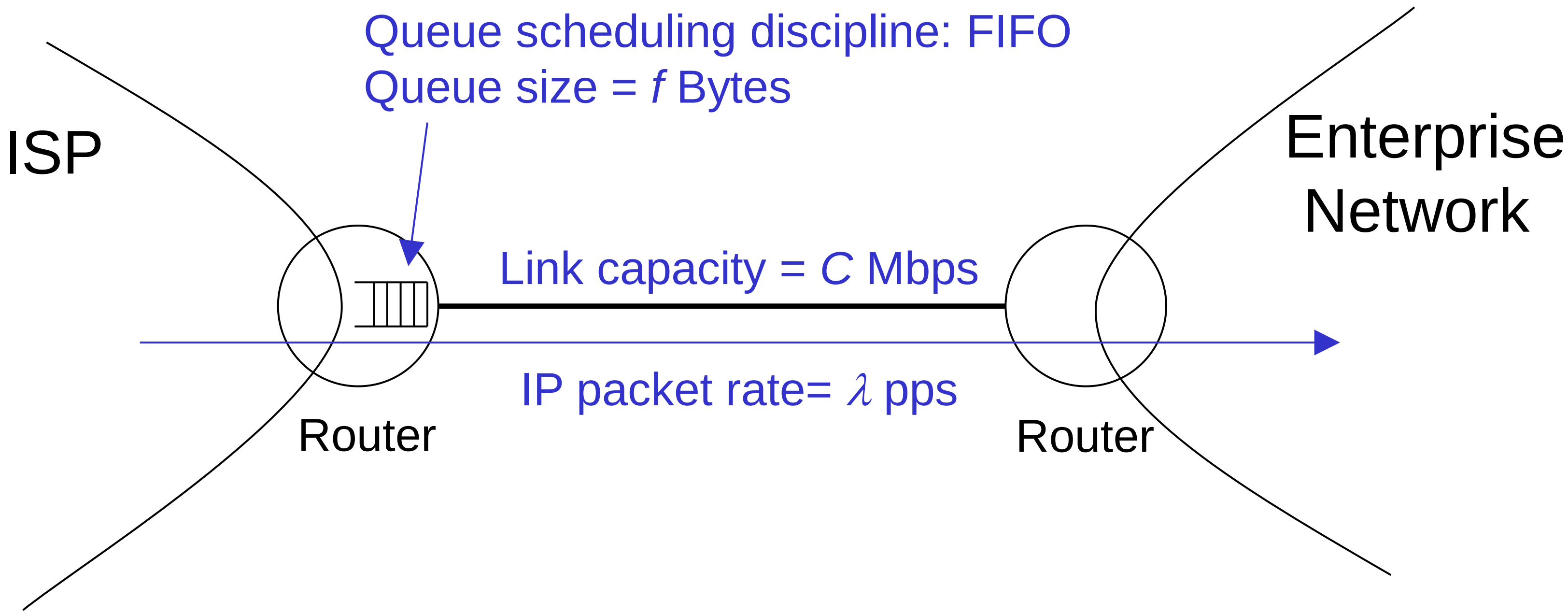
APD = 1000 × DELAYS / TRANSPACKETS

MPD = 1000 × MAXDELAY

TT = 10^{-6} × TRANSBYTES × 8 / total simulated time

Example: performance of a downstream link from an ISP to a client enterprise

Illustration of a simulation run in MATLAB



Generation of random numbers with a uniform distribution between 0 and 1

A Linear Congruential Generator (LCG) is an algorithm that yields a sequence of randomized numbers calculated with a linear equation.

The method represents one of the oldest and best-known pseudorandom number generator algorithms.

Generation method:

(1) Generate integer values Z_1, Z_2, … with the following recursive expression:

$$Z_i = (aZ_{i-1} + c)(\operatorname{mod} m)$$

where m, a, c and Z_0 are non-negative integer parameters;

(2) Compute $U_i = Z_i / m$.

Seed of the generator

The values U_i seem to be real values uniformly distributed on the interval [0,1]

Example

Example: $Z_i = (5Z_{i-1} + 3)(\text{mod } 16)$

$Z_0 = 7$

i	Z_i	U_i	***i***	Z_i	U_i
0	7	----	10	9	0.563
1	6	0.375	11	0	0.000
2	1	0.063	12	3	0.188
3	8	0.500	13	2	0.125
4	11	0.688	14	13	0.813
5	10	0.625	15	4	0.250
6	5	0.313	16	7	0.438
7	12	0.750	17	6	0.375
8	15	0.938	18	1	0.063
9	14	0.875	19	8	0.500

- In this example, m = 16 and the algorithm repeats the generated numbers after 16 iterations (we say the generator has a period of 16).
- The random generator (function *rand*) of MATLAB has a period of 2^{31}-1.

Generation of random numbers with other distributions

Discrete variables:

Consider the generation of a random variable that can be $X_1, X_2, \ldots, X_n$.

Consider the probability of value X_i as $P(X = X_i) = f_i$.

Method:

- Split the interval [0,1] in n intervals proportional to f_i, $i = 1 \ldots n$
- Generate a uniformly distributed random value U in [0,1]
- Return X_i if U falls into the i-th interval

For example, the Bernoulli variable X with $p(0) = 1/4$ and $p(1) = 3/4$ can be randomly generated as:

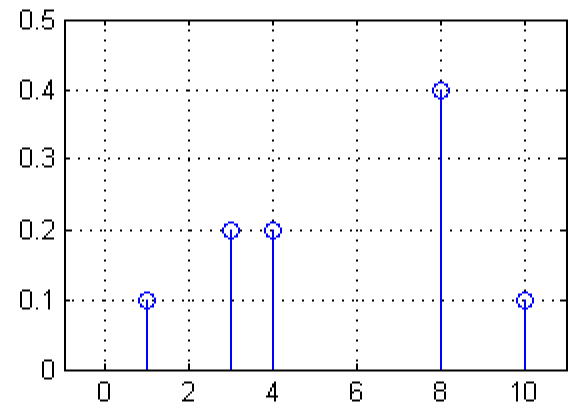
(1) Generate $U \sim U(0,1)$

(2) If $U \leq 1/4$, return $X = 0$; otherwise, return $X = 1$

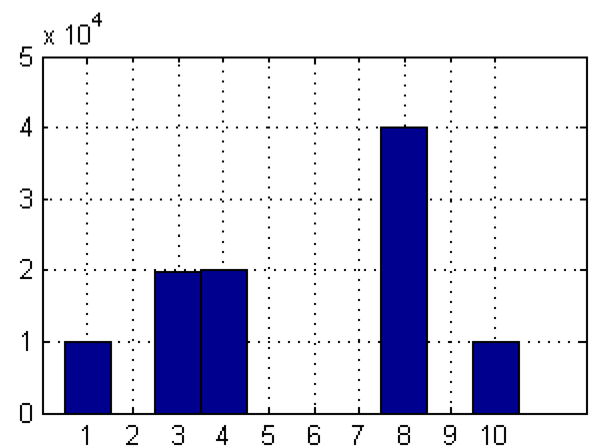
Generation of random numbers with other distributions

__Discrete variables__ (MATLAB example)

```
x= [1 3 4 8 10];
f= [0.1 0.2 0.2 0.4 0.1];

figure(1)
stem(x,f)
axis([-1 11 0 0.5])
grid on

f_cum= [0 cumsum(f)]

a= zeros(1,100000);
for it= 1:100000
    a(it)= x(sum(rand()>f_cum));
end

figure(2)
hist(a,1:10)
grid on
```




```
f_cum =

    0.0   0.1   0.3   0.5   0.9   1.0
```



Generation of random numbers with other distributions

Generate a random packet size between 64 and 1518 bytes with the probabilities: 19% for 64 bytes, 23% for 110 bytes, 17% for 1518 bytes and an equal probability for all other values (i.e., from 65 to 109 and from 111 to 1517).

Custom MATLAB function:

```
function out= GeneratePacketSize()
    aux= rand();
    aux2= [65:109 111:1517];
    if aux <= 0.19
        out= 64;
    elseif aux <= 0.19 + 0.23
        out= 110;
    elseif aux <= 0.19 + 0.23 + 0.17
        out= 1518;
    else
        out = aux2(randi(length(aux2)));
    end
end
```

Generation of random numbers with other distributions

<u>*Continuous variables*</u>:

The most popular methods are based on the inverse of the cumulative distribution function (cdf).

Consider $F(X)$ as the cdf of a continuous random variable and $F^{-1}(U)$ as its inverse function.

Method:

(1) Generate U~U(0,1)

(2) Return $X = F^{-1}(U)$

For example, an exponential distributed random variable with average $1/\lambda$:

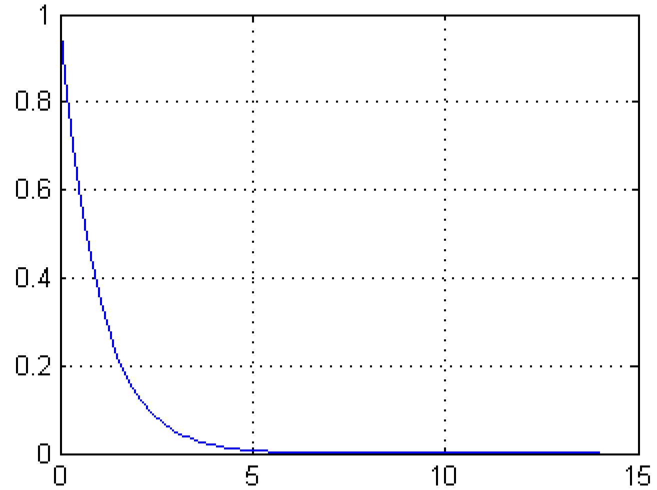
$$F(x) = \begin{cases} 1 - e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases} \qquad F^{-1}(U) = -\frac{1}{\lambda}\ln(U)$$

In MATLAB, use function *exprnd().*

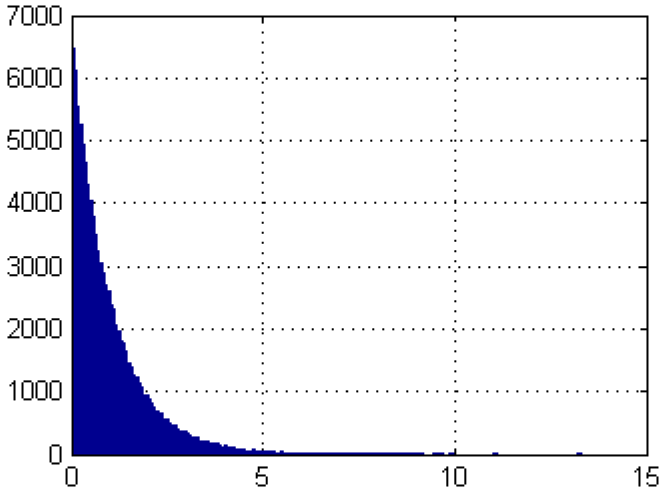
Generation of random numbers with other distributions

Exponential variable (MATLAB example)

```
lambda= 1
x= 0:0.1:14;
f=exppdf(x,1/lambda)
figure(1)
plot(x,f)
grid on
```




```
a=exprnd(1/lambda,1,100000);
figure(2)
hist(a,200)
grid on
```



Analysis of the results of a simulation

Consider X_1, X_2, …, X_n as the observations of independent and identically distributed (IID) random variables with average μ and finite variance σ^2 (for example, the results of different simulations of a given system).

The <u>sample mean</u> defined by

$$\overline{X}(n) = \frac{\sum_{i=1}^{n} X_i}{n}$$

is an estimator for average μ.

The <u>sample variance</u> defined by

$$S^2(n) = \frac{\sum_{i=1}^{n} \left(X_i - \overline{X}(n)\right)^2}{n-1}$$

is an estimator for variance σ^2.

The analysis of the results of a simulation is, usually, based on the <u>Central Limit Theorem</u>.

Analysis of the results of a simulation

Consider Z_n as a random variable given by: $$Z_n = \frac{\overline{X}(n) - \mu}{\sqrt{\sigma^2 / n}}$$

Consider $F_n(z)$ the cumulative distribution function of Z_n for a sample of size n.

The *Central Limit Theorem* states that

$$\lim_{n \to +\infty} F_n(z) = \Phi(z)$$

where $\Phi(z)$ is the cumulative distribution function of a standard Gaussian random variable (i.e., a Gaussian distribution with mean 0 and variance 1).

Given that $\lim_{n \to +\infty} S^2(n) = \sigma^2$ than, the random variable $$\frac{\overline{X}(n) - \mu}{\sqrt{S^2(n) / n}}$$

has approximately a standard Gaussian distribution.

Analysis of the results of a simulation

For a sufficiently high value of *n*,

$$P\left(-z_{1-\alpha/2} \leq \frac{\overline{X}(n)-\mu}{\sqrt{S^2(n)/n}} \leq z_{1-\alpha/2}\right) =$$

$$P\left(\overline{X}(n)-z_{1-\alpha/2}\sqrt{S^2(n)/n} \leq \mu \leq \overline{X}(n)+z_{1-\alpha/2}\sqrt{S^2(n)/n}\right) \approx 1-\alpha$$

where $z_{1-\alpha/2}$ is the critical value of the standard Gaussian distribution ($z_{1-\alpha/2}$ is the value z such that $P(x \leq z) = 1 - \alpha/2$ where x is a random variable with a standard Gaussian distribution).

Therefore, the approximate confidence interval of $100(1-\alpha)\%$ for the average μ is given as

$$\overline{X}(n) \pm z_{1-\alpha/2}\sqrt{S^2(n)/n}$$

Analysis of the results of a simulation

The approximate confidence interval of 100(1-α)% for the average μ is

$$\overline{X}(n) \pm z_{1-\alpha/2}\sqrt{S^2(n)/n}$$

In MATLAB:

```
N = 20;               %number of simulations
per1= zeros(1,N); %vector with N simulation values
per2= zeros(1,N); %vector with N simulation values
for it= 1:N
     [per1(it),per2(it)]= simulator();
end

alfa= 0.1; %90% confidence interval%
media = mean(per1);
term = norminv(1-alfa/2)*sqrt(var(per1)/N);
fprintf('per1 = %.2e +- %.2e\n',media,term)
media = mean(per2);
term = norminv(1-alfa/2)*sqrt(var(per2)/N);
fprintf('per2 = %.2e +- %.2e\n',media,term)
```

Analysis of the results of a simulation

The central limit theorem requires variables X_1, X_2, …, X_n to be independent and identically distributed (IID).

- One way to guarantee this independence is to run different simulations guaranteeing that the random values are different on the different runs.
- This is done by using different seeds on the random generators.

The confidence interval provides a measure on how close the estimated performance value might be from its real value:

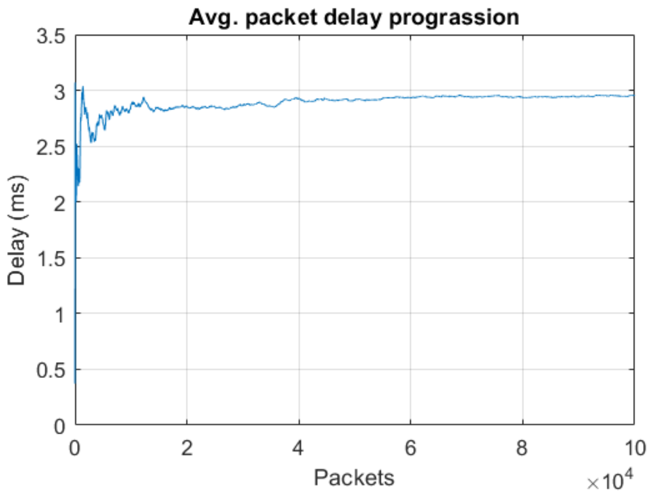
- If it is small, the estimated value should be very close to the real value.
- If it is large, we don't know the quality of the estimated value.

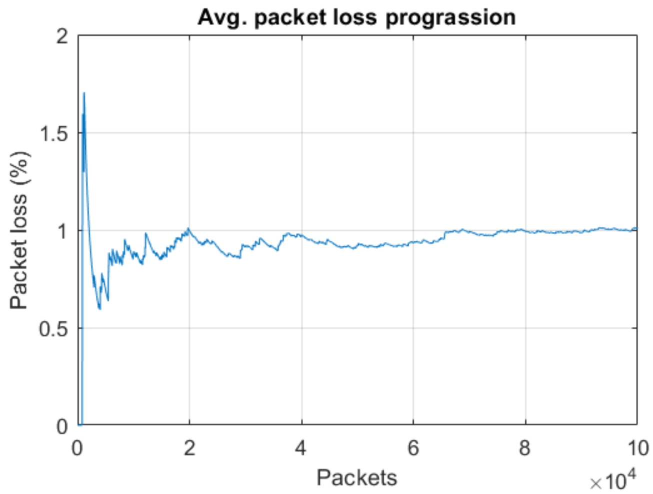
For a given set of simulations, if the confidence interval is large, it can be made small by:

- running a larger number of simulations, and/or
- running longer simulations.

Analysis of the results of a simulation

In general, the stochastic processes have initial transient states (which are dependent on the initial conditions) before reaching the stationary state.





- In order to guarantee that the performance estimations are correct, the simulation must first warm-up to let the transient states vanish.
- If the simulated time is much higher than the warm-up time, statistical counters can be initialized at the beginning of the simulation.
- Otherwise, the statistical counters must be initialized only after the warm-up time (this time must be estimated though).