HALMSTAD UNIVERSITY

Quantitative methods and modeling: Computer Simulation Part II

Urban Bilstrup (E327)

Urban.Bilstrup@hh.se

140923

Elements of simulation analysis – model verification and validation

- Verification focuses on the process of determining whether the operational logic of the model (the computer program) corresponds to the logic flow cart (are there any errors in the program ????).

- Validation focuses on the process of determining if the model is meaningful and accurate representation of the real system.

Elements of simulation analysis – model verification and validation

- Model verification and validation is concerned with three models.
  - Conceptual model
  - Logic model
  - Computer model
- Validating the ***conceptual model*** is the process of establishing the relevance of our abstraction of the real system, and what questions the simulation model is expected to answer.
- Validating the conceptual model can be thought of as a binding process, in which the simulation analysts, decision makers, and managers, agree upon which aspects of the real system which should be included in the simulation model and what information that the model will provide as output.

Elements of simulation analysis – model verification and validation

- The ***logical model***, the flow chart, serves as a bridge from the conceptual model to the computer model.

- Validating the ***logical model*** is easy if the conceptual model has been well constructed.

- Three basic questions can be used:
  - Are the events within the model processed correctly.
  - Are the mathematical formulas and relationship in the model valid.
  - Are the statistics and measures of performance calculated correctly.

Elements of simulation analysis – model verification and validation

- A common method for verifying and validating the event processing within the logical model is a ***structured walk-through*** (a form of review meeting).
- In a structured walk-through the developer of the logical model must explain the detailed logic of the model to other members of the simulation project team.
- Structured walk-through is not unique for simulation project it is applicable on almost all software development projects.

Elements of simulation analysis – model verification and validation

- The method of detecting errors in mathematical calculations or formulas is not much different than detecting errors in semantics or spelling in natural languages. That is, careful scrutiny by the analysts and proofing by someone else.

- A common error in simulation modeling is to fail to update all relevant statistics and measures when an event occurs.

- One method of verifying that the statistics and the measures of performance are updated correctly is the use of an event graph. In this each event state is associated with a complete list of all statistics and measure of performance that will change when the event occurs.

Elements of simulation analysis – model verification and validation

- Verifying the computer program for a simulation model is not much different from verifying any software.

- It often requires imagination and ingenuity of the programmer, it is one of the few things that only can be conducted by the persons with programming skills in a simulation study team.

Elements of simulation analysis – model verification and validation

- Six general approaches to verify a computer program are:
  - Structured programming methods
  - Tracing the simulation during run time
  - Program testing
  - Logical relationship checks
  - Comparison to analytical models
  - Graphics

Elements of simulation analysis – model verification and validation

- Some rules of the thumb for designing well-structured computer programs are:
  - **Top-down design**: The program is designed starting with the highest level processes which are then decomposed into subsidiary modules which themselves can be further decomposed.
  - **Modularity**: Each subsidiary module is responsible for a single function.
  - **Stepwise refinement**: Each module is developed using a step-by-step refinement of the module's function.
  - **Compact modules**: Modules should be short in length, fifty lines of code is often given as a upper bound.
  - **Structured control**: All control code should be simple to understand and highly structured IF-THEN-ELSE, WHILE, REPEAT-UNTIL, FOR and CASE statements.

Elements of simulation analysis – model verification and validation

It is always easier to verify 20 modules averaging 25 lines of code, than it is to verify one module of 500 lines of code !!!

Elements of simulation analysis – model verification and validation

- When the simulation program is designed, mechanisms for ***tracing*** should be included directly and not patched when erroneous execution is discovered in the simulation program, traces should be possible to turn on and off.

- The trace mechanisms can be based on that a print function is included in all functions, printing all relevant simulation parameters and simulation time.

- However, one should be aware of that the amount of trace output data can be overwhelming.

Elements of simulation analysis – model verification and validation

- The two most basic approaches for ***testing*** a computer program are ***bottom-up*** and ***top-down*** testing.
- In ***bottom-up*** testing the lowest, most basic modules are tested first (sometimes referred to as unit testing).
- After the basic units have been tested, integration tests are performed in which interfaces between two or more modules are tested.
- This continues iteratively until the whole system can be tested as one instance.

Elements of simulation analysis – model verification and validation

- In ***top-down testing***, the testing begins with the main module and incrementally moves down to lower modules.
- In top-down testing, stubs or dummy routines are required to simulate the function of lower-level modules.
- An advantage of the top-down approach is that testing can be conducted parallel with the development of the software.
- Programmers and management often feel more comfortable with top-down testing because it gives the ***appearance*** that progress is being made.

Elements of simulation analysis – model verification and validation

- Checking the logic expressions in IF-THEN-ELSE statements are crucial.
- Avoid a depth of more than 7 IF statements.
- Typically, logic errors are not randomly or uniformly distributed throughout a computer program, they tend to cluster into colonies of bugs. ☺
- Queues and lists generate runtime errors, memory leakage, do not use recursive functions or be extremely careful when you use it.

Elements of simulation analysis – model verification and validation

- When the computer program has been verified, assumed to be correct, the next step is to validate whether the simulation program generates output that is a valid representation of the real system or not.

- ***Program validation*** is to judge whether the simulation program have enough confidence in the result to use it as part of the decision making process.

- The most straightforward method is to compare simulation result with an existing system. This is however only possible if necessary data is available from a real system.

Elements of simulation analysis – model experimentation and optimization

- There are some aspects in the interpretation of simulation models outputs that are unique to simulation. The main issue is that a simulation models only yield estimates of measures that are subject to sampling errors.

  - As an example; an analytical evaluation provide ***mean*** and ***variance*** and exact ***probability distributions*** for the measures of performance.

  - Simulations on the other hand provide the ***sample mean*** and ***sample variance*** and ***sample distributions***, from these we must ***estimate population parameters*** and ***distributions***.

Elements of simulation analysis – model experimentation and optimization

- The major issue in obtaining useful estimates from samples are:
  - that the samples are representative for the typical system behavior
  - that the sample size is large enough to provide some stated level of precision for the performance measure estimates.

- ***Sample size*** is especially important since the ***precision*** of estimates is dependent upon the variance of the sample mean and the variance changes inversely with the sample size

Elements of simulation analysis – model experimentation and optimization

- We may now turn to the ultimate purpose of model experimentation: to derive information about the system behavior.
- This information should be in a format which is helpful for decision making.
- When considering system performance we may whish to know how well a system behaves in an ***absolute sense***, ***comparatively***, or ***in contrast to one or several system configurations***.

Elements of simulation analysis – model experimentation and optimization

- Even for the modest experimental design, enumeration of all possible solutions is not recommended nor feasible in searching for the best or very good configurations of the system under consideration.

- Consequently we need a more directed, structured approach for seeking worthwhile solutions.

- Two different approaches are:
  - predetermined sets of experiments
  - optimum seeking techniques.

Elements of simulation analysis – model experimentation and optimization

- The ***predetermined sets of experiments*** approach identifies factors which would affect some output measure.
- This is done by performing experiments with a limited set of factors, set at a limited set of values and combination of these.
- Statistical techniques, often referred to as analysis of variance (ANOVA) is then applied to discern whether a specific factor have any large impact on the output measure of performance.
- If for example two factors are considered, at three different levels, nine runs are necessary.

Elements of simulation analysis – model experimentation and optimization

- The use of predetermined set of model is effective in identifying important factors.
- Unfortunately can the technique not lead one toward the overall optimal solution (a global optimum).
- It can not even guarantee one of several very good solutions (a local optimum).

Elements of simulation analysis – model experimentation and optimization

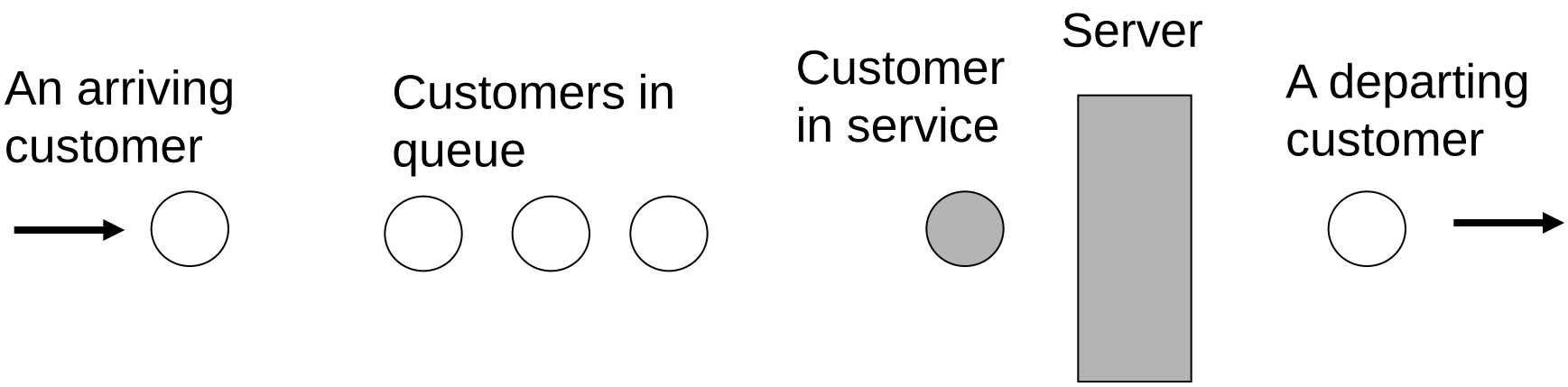
- One optimum seeking technique is simply to start generating output to form a response surface, a two factor experiment defines a surface in three dimensions, i.e., compare with a topographic map.
- By using different strategies one can reach high points or even the summit.
- One often used way is ***steepest ascent***, it requires that sufficient simulations runs are made to determine which direction that seems to yield the greatest increase in altitude.
- The decision variables are changed accordingly, and the process is continued until one can not go higher (a local or global optima is reached).

An implementation example of a discrete event simulator.

An implementation and execution example of a discrete event simulator.

- Consider a s service facility with a single server, e.g. an information desk.
- We want to estimate the average delay in the queue, average number of customers in the queue and expected utilization of the service desk.

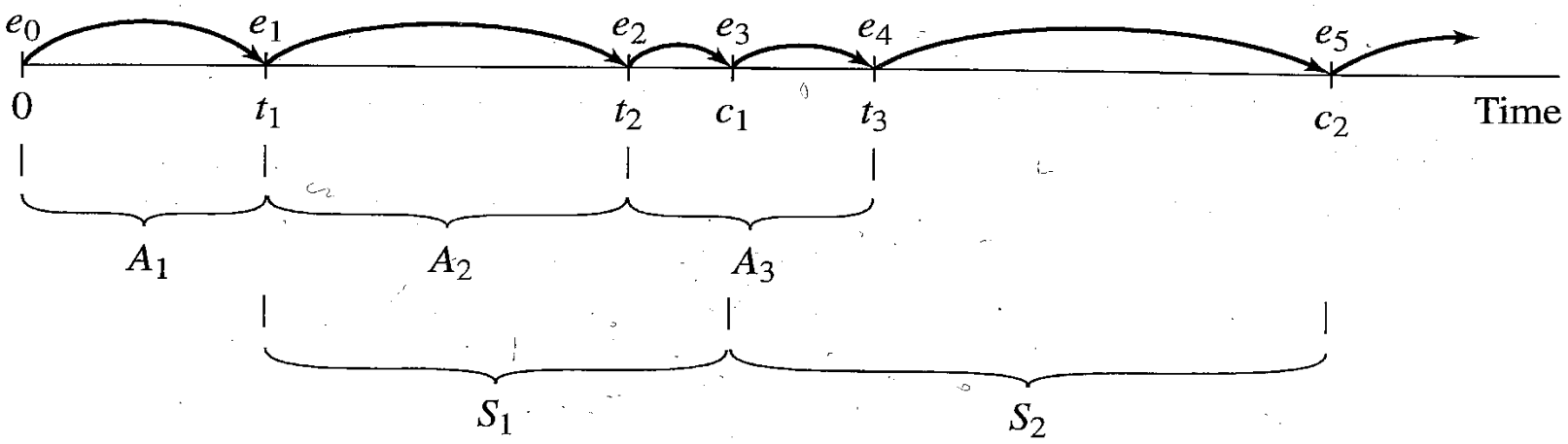
Try to visualize the problem formulation !!!!!



An implementation and execution example of a discrete event simulator.

- We assume the next event time advance, visualize the time domain operation of the simulator and define some related variables.

t_i = time of arrival of the ith customer ($t_0 = 0$)
$A_i = t_i - t_{i-1}$ = interarrival time between $(i-1)$st and ith arrivals of customers
S_i = time that server actually spends serving ith customer (exclusive of customer's delay in queue)
D_i = delay in queue of ith customer
$c_i = t_i + D_i + S_i$ = time that ith customer completes service and departs
e_i = time of occurrence of ith event of any type (ith value the simulation clock takes on, excluding the value $e_0 = 0$)



An implementation and execution example of a discrete event simulator.

- Data structures
  - **System state variables**: The collection of state variables necessary to describe the state of a system at a particular time.
  - **Simulation clock**: A variable given the current value of simulated time.
  - **Event list**: A list containing the next time an event will occur.
  - **Statistical counters**: Variables used for storing statistical information about the system performance.

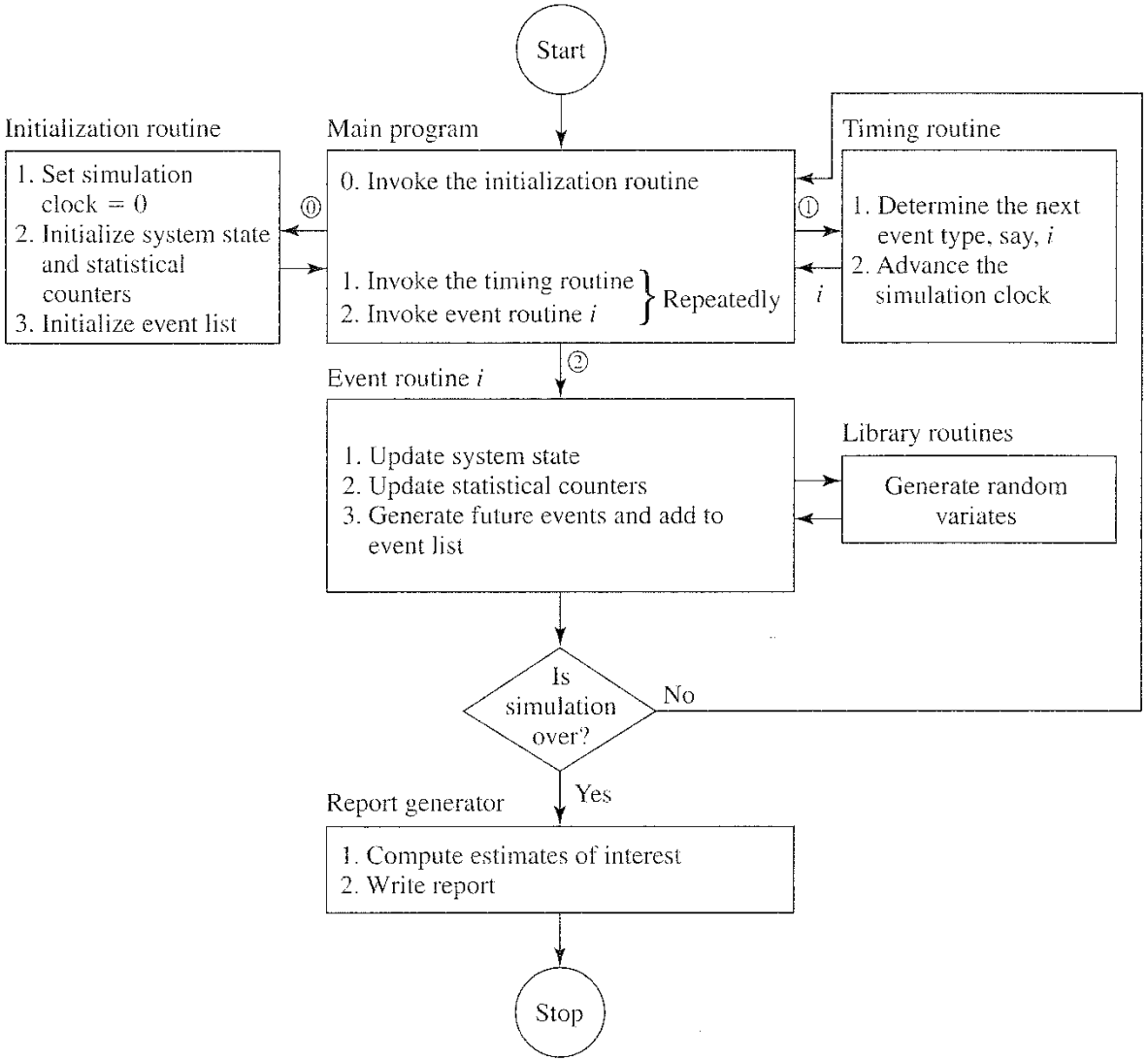
An implementation and execution example of a discrete event simulator.

- Routines
  - **Initialization routine**: A sub program to initialize the simulation model at t=0.
  - **Timing routine:** A subprogram that determines the next event from the event list and then advances the clock to the time when the event occurs.
  - **Event routine:** A subprogram that updates the system state when a particular type of event occurs.
  - **Library routines:** A set of subprograms used to generate random observations from probability distributions that were predetermined as part of the simulation model during problem formulation and data collection.

An implementation and execution example of a discrete event simulator

- **Report routine:** A subprogram that computes estimates of the desired measure of performance and produces a report when the simulation ends.
- **Main program:** The main program invoke the timing routine to determine the next event and then transfer control to the corresponding event routine to update the system appropriately. The main program may also check for termination of the simulation and invoke the report generator when the simulation is over.

An implementation and execution example of a discrete event simulator



An implementation and execution example of a discrete event simulator

- The simulation will begin with in the state, i.e., no customer is present and the server is idle.
- At time 0, we will begin waiting for the arrival of the first customer, which will occur after the first inter-arrival time A_1.
- We wish to simulate this system until a fixed number, n, of customers have completed their delays in the queue.
- Note that the time the simulation will stop is thus a random variable, depending on the observed values of the inter-arrival and the service time random variables.

Some measures

- To measure the performance of this system, we will look at the estimates of the three measures of performance, customer delay, queue length and utilization.
- First the estimate of average customer delay, ***d*(*n*)** of the ***n*** customers.
- One interpretation of this is that ***d*(*n*),** is the average of the number of ***n*** customers delay.
- From a single run of the simulation resulting in customer delays: D_1, D_2, D_3…….D_n, and obvious estimator of ***d*(*n*)** is:

$$\hat{d}(n) = \frac{\sum_{i=1}^{n} D_i}{n}$$

Some measures

- The average number of customers in the queue, q(n), this is a different kind of average than the previous because it is taken over continuous time, rather than over customers.
- Let ***Q*(*t*)** denote the number of customers in the queue at time ***t*** for any ***t*** $\geq$ ***0***, and let ***T*(*n*)** be the time required to observe our ***n*** delays in the queue.
- For any time ***t*** between ***0*** and ***T*(*n*)**, **Q(*t*)** is nonnegative integer.
- Further if we let p_i be the expected proportion of the time ***Q*(*t*)** is equal to ***i***, then a reasonable estimate of ***q*(*n*)** would be:

$$\hat{q}(n) = \sum_{i=0}^{\infty} i p_i$$

q(n) is a weighted average of the possible values of I for the queue length Q(t), with the weights being the expected proportion of time the queue spends at each of its possible position

Some measures

- Computationally it is however easier rewrite $q(n)$ using some geometric considerations. If we let T_i be the total time during the simulation that the queue is of length i, then $T(n)=T_0+T_1+T_2\ldots$ and

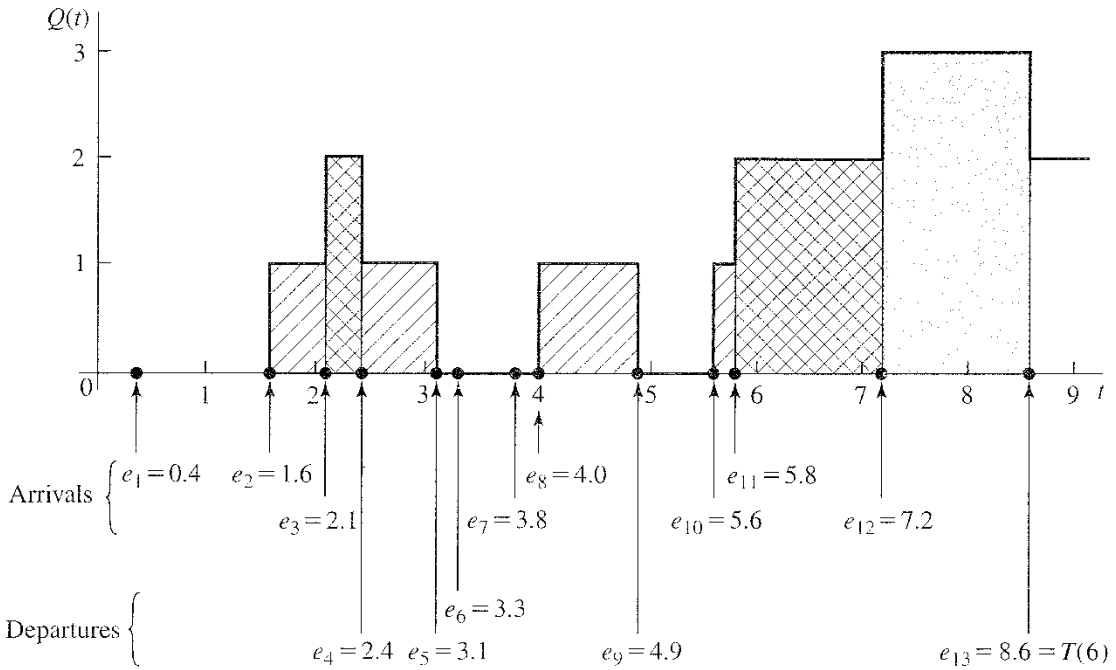
$$p_i=T_i/T(n)$$

so that we can rewrite it to

$$q(n) = \frac{\sum_{1=0}^{\infty} iT_i}{T(n)}$$

Easier understood by a visualization !!!!!

Some measures



T_0 = (1.6 - 0.0) + (4.0 - 3.1) + (5.6 - 4.9) = 3.2

T_1 = (2.1 - 1.6) + (3.1 - 2.4) + (4.9 - 4.0) + (5.8 - 5.6) = 2.3

T_2 = (2.4 - 2.1) + (7.2 - 5.8) = 1.7

T_3 = (8.6 - 7.2) = 1.4

Some measures

- The numerator is then:

$$\sum_{i=0}^{\infty} iT_i = 0x3.2 + 1x2.3 + 2x1.7 + 3x1.4 = 9.9$$

and our estimate of the time average number in the queue from this particular simulation run is **9.9 / 8.6 = 1.15**.

Note that each of the non-zero terms corresponds to one of the shaded areas in the figure.

In other words, the summation is just the ***area under the Q(t) curve*** between the beginning and end of the simulation, i.e.,

$$\hat{q}(n) = \frac{\int_0^{T(n)} Q(t)dt}{T(n)}$$

Some measures

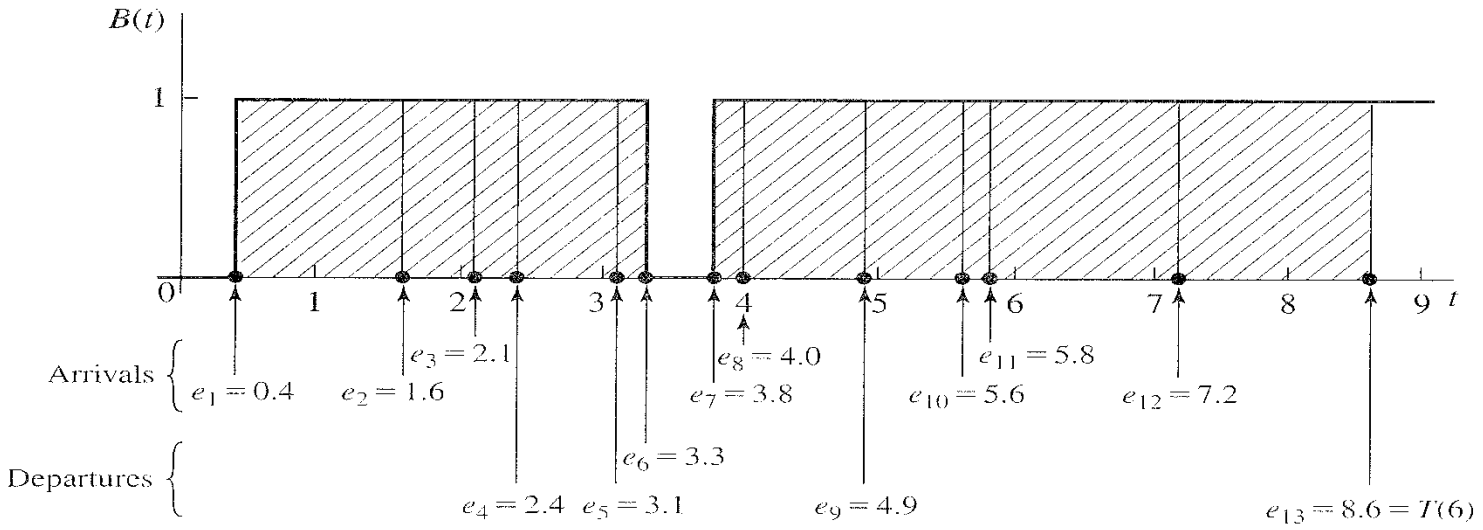
- The expected utilization of the server is the expected portion of time during the simulation that the server is busy.
- It is easy to look at this quantity as a continuous time average, by defining a busy function, ***B*(*t*)**.

$$B(t) = \begin{cases} 1 \text{ if the server is busy at time } t \\ 0 \text{ if the server is idle at time } t \end{cases}$$

- So *u*(*n*) could be expressed as the portion of time that ***B*(*t*)** is equal to 1 (unity).

Easier understood by a visualization !!!!!

Some measures



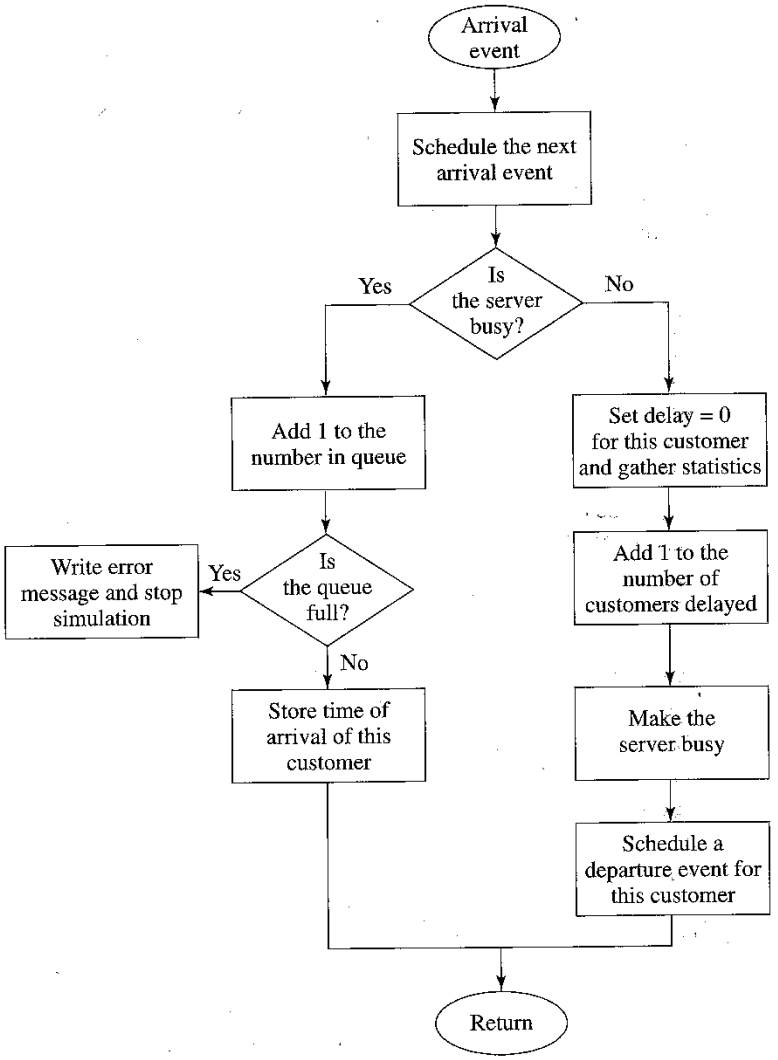
$$\hat{u}(n) = \frac{(3.3-0.4)+(8.6-3.8)}{8.6} = \frac{7.7}{8.6} = 0.90$$

Some measures

- Again, the numerator can be viewed as the area under the ***B*(*t*)** function, i.e.,

$$\hat{u}(n) = \frac{\int_0^{T(n)} B(t)dt}{T(n)}$$

An implementation and execution example of a discrete event simulator



An implementation and execution example of a discrete event simulator