

Simulative Evaluation of Internet Protocol Functions

Introduction

- ❑ Course Objectives & Introduction
- ❑ Performance Evaluation & Simulation
- ❑ A Manual Simulation Example
- ❑ Resources

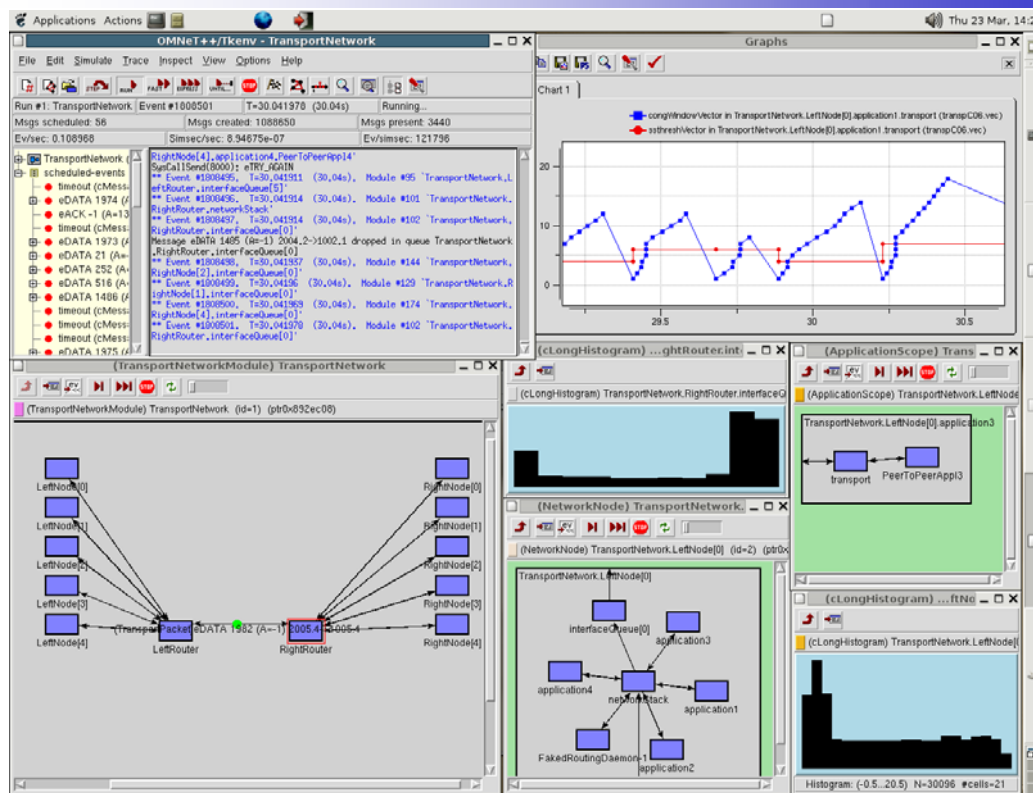
<http://www.tu-ilmenau.de/fakia/simpro.html>



You are here to get a “hands-on” experience...

- ❑ This is a complementary course to our lectures Telematics 1 and Performance Evaluation:
 - ❑ Can also be taken in parallel to / without performance evaluation course
 - ❑ You should be ready to read some introductory material before the first lab
- ❑ This practically oriented course is designed to give you a “hands-on” experience with network protocol functions and simulation studies:
 - ❑ Introduces a simulation environment and lets you add protocol functionality
 - ❑ Studied protocol functions: forwarding, routing, (interface queues), connection setup, error-, flow- and congestion control
 - ❑ Requires good programming skills
 - ❑ Knowledge of C++ is an asset (but not a pre-requisite)
 - ❑ Allows you to obtain in-depth knowledge of topics covered in Telematics I and the techniques and art of simulation studies – because afterwards “you did it!” :o)





- ❑ In this course, you will basically learn about two subject areas:
 - ❑ What are protocol functions and how do they work?
 - We will explain and experiment with just some of them, but almost all of our examples are actually deployed in the Internet
 - ❑ How can one evaluate, if one method realizing a protocol function is good, or, comparatively speaking, better than another one?
 - In fact, we deal with one specific approach to answer such questions, namely *evaluation by simulation*
- ❑ The good thing about our “simulative approach” to teaching protocol functions is that you can:
 - ❑ Literally watch the protocol functions “at work”, being able to examine every single step of their operation
 - ❑ Experiment with protocol functions in “whole networks”, using just one single (even “disconnected”) PC running either Linux or Windows XP

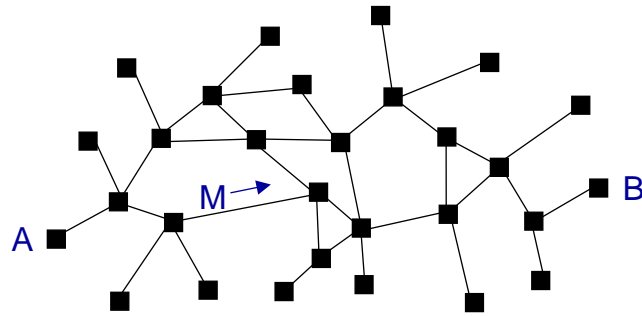


- ❑ There are a couple of things you need to know about in order to be able to start with your experiments:
 - ❑ Simple understanding of a network (see lecture Telematics 1)
 - ❑ Performance evaluation methods, especially simulation (today; see also quick-start material on web page)
 - ❑ Some basic knowledge about the C++ programming language (your task; see also quick-start material)
 - ❑ Introduction to the OMNet++ discrete event simulation system (also covered in quick-start material)
 - ❑ Overview of the protocol function simulation framework used in this course (see quick-start material...)



- ❑ For our purposes:
 - ❑ A network is a (connected) graph consisting of *nodes* and edges (the latter called *links*)
 - ❑ The nodes can send and receive *messages* (“communicate”) over the links directly attached to them
 - ❑ We prefer to deal with connected graphs:
 - Unconnected graphs can be seen as a collection of connected graphs which can not communicate with each other
 - Considering them would allow little additional insight...
 - ❑ Sometimes, we distinguish between;
 - *End systems* that are the (ultimate) source and/or destination of a message
 - *Intermediate systems*, that relay messages between nodes, that do not share a common link





- The above (randomly generated) figure, showing a network with two (randomly) designated nodes *A* and *B*, allows us to raise some (not randomly generated :o) questions:
 - How can a message *M*, that *A* wants to send to *B*, make its way through the network? (→ forwarding, routing)
 - What can we do, if the *M* gets accidentally falsified by an error on its way? (→ error detection and handling)
 - How can we avoid that *A* overloads *B* with its messages? (→ flow control)
 - How can we avoid overload in intermediate nodes? (→ congestion control)



- And this already introduced the protocol functions, we will study in this course:
 - *Forwarding*: an intermediate node receives a message on one link and sends it on another link (which link is determined using local information)
 - *Routing*: determining the forwarding information required in nodes
 - *Error detection*: realizing that a message has been (accidentally) altered while traversing the network (or a link)
 - *Error handling*: the measures taken in reaction to a detected error
 - *Flow control*: avoiding overload of a slow receiver by a fast sender
 - *Congestion control*: avoiding overload in intermediate nodes (= having to forward more messages than the node is able to)



Protocol Functions Studied in this Course (2)

- ❑ Why will we need a whole winter term to study these protocol functions that can be motivated and defined on only two slides?
 - ❑ Well, we want you to actually *understand*:
 - with what “methods” they can be implemented, and
 - how these “methods” work!
 - ❑ This requires you:
 - to program them on your own,
 - to experiment with them and watch them at work, and
 - to evaluate different *parameter settings* (more on this below) and obtain and interpret results on certain *performance metrics* (also more on this below) of different methods
 - ❑ And last but not least:
 - we want you to “have some fun” during this whole exercise, and
 - fun takes its time at well... :o)



Performance Evaluation (1)

- ❑ Given a system, how do you evaluate its performance?
- ❑ Three classic methods:
 - ❑ Experiments: Use a concrete example of a system and try to measure its performance
 - ❑ Analysis: Construct a mathematical abstraction of the system and derive equations describing the system’s performance
 - ❑ Simulation: Build a model (a representation) of the system, along with its operations, and use this model to numerically evaluate system performance – usually with the help of computers
- ❑ In this course, we will use simulation

(Acknowledgement: material on the next slides originally authored by H. Karl)



- ❑ Some questions related to this:
 - ❑ What is a system?
 - ❑ What are operations of a system?
 - ❑ What is performance?
 - ❑ On what does performance depend?
 - ❑ What is a model?
 - ❑ How to build a model?
 - ❑ How to numerically evaluate it?
 - ❑ How to interpret the results of such an evaluation?



- ❑ At the end of this course, you should furthermore:
 - ❑ know about simulation principles (in particular, the so-called discrete event simulation)
 - ❑ design and implement simple discrete event simulation programs
 - ❑ have some experience with a modern simulation tool

- ❑ Emphasis is on practical aspects and questions, not on theory.



What is a System?

- ❑ Definition of a system:
 - ❑ American Heritage Dictionary: 1. A group of interacting, interrelated, or interdependent elements forming a complex whole, 2d. A group of interacting mechanical or electrical components
 - ❑ Characterized by its parts and their interactions
 - ❑ System as a whole is also defined by its purpose or function
 - ❑ Systems can be part of the real world or virtual entity
- ❑ System notion depends on abstraction level:
 - ❑ On a higher level, system can be part of another system
 - ❑ On a lower level, parts of a system usually can be regarded as systems as well (subsystems)
- ❑ Systems are recursively structured:
 - ❑ In general, no atomic parts
 - ❑ In practice, atomic parts are chosen depending on intentions of usage
 - ❑ Choice of “system” abstraction depends on objective or purpose of usage



System State

- ❑ In most systems, different modes or conditions of being can be distinguished:

The system can be in different *states*
- ❑ Because of recursive nature of system, parts also (usually) have a state
- ❑ State can be expressed by the values of a set of variables
- ❑ System state is the aggregation of state of the parts
- ❑ State can be *continuous* or *discrete*



System State Changes (1)

- ❑ Interesting (dynamic) systems change their state over time:
 - ❑ Static systems do not change their state, passage of time plays no role
- ❑ In *(time-)continuous systems*, the state of the system is a continuous (in the mathematical sense) function of time:
 - ❑ Examples: airplanes moving, planets turning, cars crashing, etc.
 - ❑ In such systems, state is continuous as well
 - ❑ Discrete state values make little sense for time-continuous systems
- ❑ In *(time-)discrete systems*, the state of the system changes instantaneously at separate points in time:
 - ❑ Examples: a cashier line in a supermarket, computer systems, communication networks
 - ❑ State can be continuous or discrete



System State Changes (2)

- ❑ Which rules govern the process of state changes?
 - ❑ Philosophical questions as well as question of abstraction level
- ❑ Newtonian physics/mechanics results in precise description of the development of the world and its objects:
 - ❑ Results in a deterministic concept of a system
 - ❑ Often appropriate
- ❑ Quantum mechanics:
 - ❑ Everything is a result of random behavior of small particles
 - ❑ Results in a stochastic concept of a system
 - ❑ Often appropriate
- ❑ Which level to choose? How to describe a system?



- ❑ Sometimes, the actual system is available for investigation (at reasonable cost and effort)
- ❑ Sometimes, only a model of a system can be used – and usually has to be constructed explicitly:
 - ❑ A model of a system is also a system
 - ❑ Built to capture some relevant properties of the original system
 - ❑ Simplified with respect to original system, reduced complexity
- ❑ Models appear in many different kinds, with many different characteristics



- ❑ Physical models:
 - ❑ Scale representation of system
 - ❑ Example: build a little airplane and a wind tunnel to study aerodynamics with experiments
- ❑ Mathematical models:
 - ❑ Represent system with appropriate mathematical or logical formalism
 - ❑ An airplane is described by the laws of aerodynamics
 - ❑ Manipulate this representation, e.g., introduce external stimuli
 - ❑ Move the rudders of the airplane = use different laws to represent it
 - ❑ Try to deduce how the real system would react (provided model is valid)
 - ❑ Would such an airplane turn left or right?
- ❑ We will only consider mathematical models!



- ❑ *Static model:*
 - ❑ A model where only a certain, fixed state of a system is considered, state changes are not taken into account
 - ❑ Evidently appropriate for static systems (one without state changes)
 - ❑ Sometimes appropriate even for dynamic systems, if only system properties in certain, fixed states are of interest
- ❑ *Dynamic model:*
 - ❑ Reflect the system's state changes as it evolves over time
 - ❑ How to handle continuous or discrete systems?



- ❑ *(Time-)Continuous models:*
 - ❑ A continuous model describes the system such that the state variables are a continuous function of time
 - ❑ Typical description: differential equation(s), describing relationships between interdependence of the rate of change of certain state variables with each other and with time
- ❑ *(Time-)Discrete models:*
 - ❑ Change of state only happens at discrete, well separated instances of time (the set of points in time where the state changes is at most countably infinite)
 - ❑ In between such times, all state variables maintain their values, the state does not change
 - ❑ At such points in times, events of the model occur, i.e., the state of the systems can only change when an event occurs (but need not necessarily change at every event)



Types of Models: Continuous vs. Discrete (2)

- ❑ Remark:
 - ❑ The type of system and does not automatically determine the type of an appropriate model of the system
- ❑ For continuous systems, continuous models are not necessarily used:
 - ❑ E.g., voice data is a continuous system, but is – for purposes of transmission – modeled as a discrete system (quantization, sampling)
- ❑ For discrete systems, discrete models are not necessarily used:
 - ❑ E.g., traffic flow on a highway – with discrete events of cars entering and leaving the highway – can be modeled continuously if only the behavior of large numbers is interesting
- ❑ Choice depends on intentions, objectives, feasibility

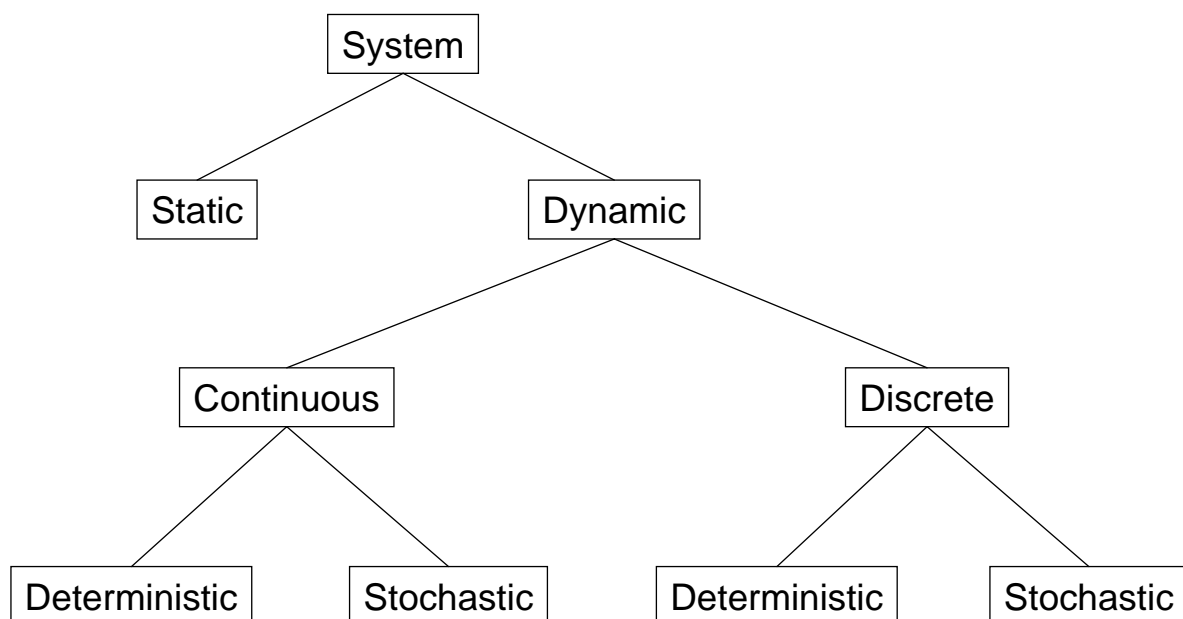


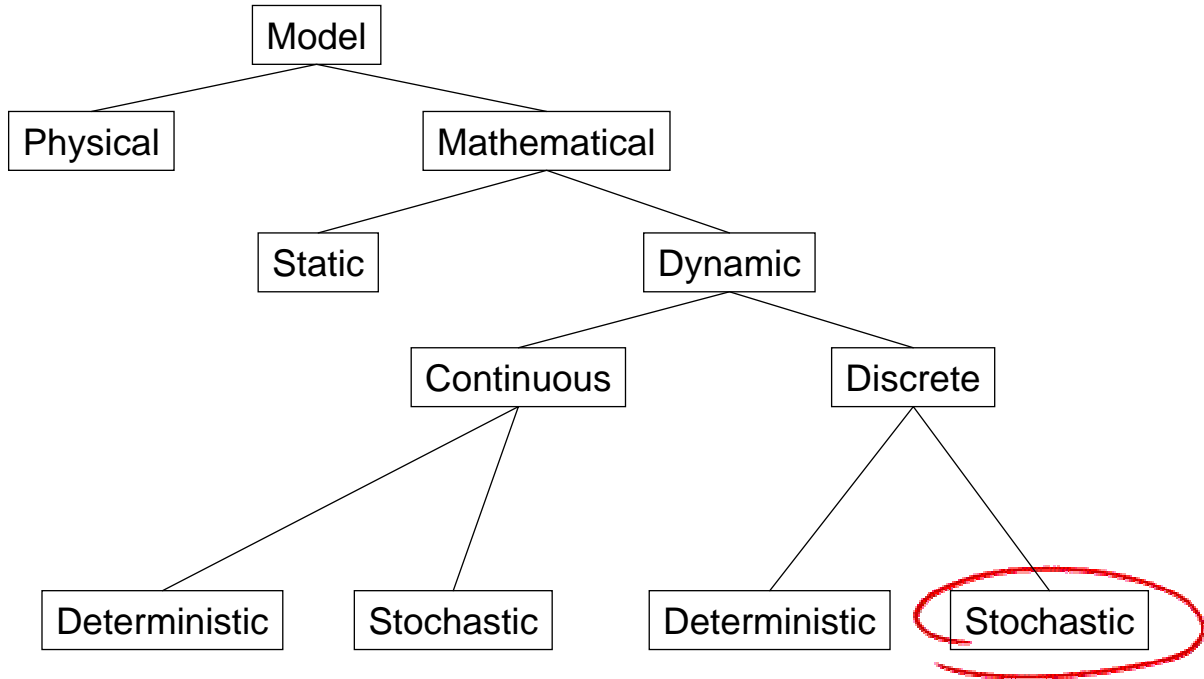
Types of Models: Deterministic vs. Stochastic

- ❑ A model where the evolution of state is completely described such that it only depends on the initial state is a *deterministic model*:
 - ❑ E.g., a set of differential equations describing concentration of different substances in a chemical reaction
- ❑ A model where the evolution of state depends on random events (random in both time of occurrence or nature) is a *stochastic model*:
 - ❑ E.g., model of a highway where the times when cars enter the highways are described by a random variable
 - ❑ Output/results for such models do not only depend on initial state, but also on the values of random variables -> no fixed or single result for such models
- ❑ Again note difference between system and its model:
 - ❑ Sometimes, stochastic systems are modeled deterministically
 - ❑ Example: chemical processes are actually random by their very nature (quantum mechanics), yet they are usually modeled deterministically (appropriate because of the large number of particles involved)



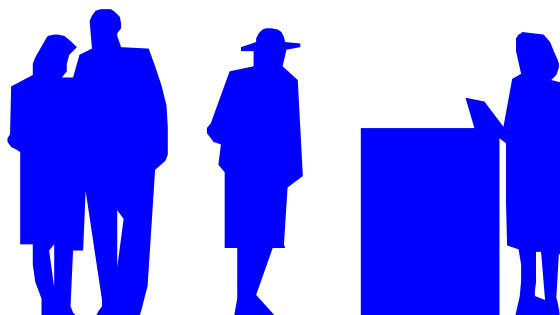
- ❑ Good models should be:
 - ❑ Appropriate representation of the system (depending on purpose of investigation)
 - ❑ As simple as possible without impeding appropriateness
 - ❑ Reusable for similar systems / models, as a part in other models
 - ❑ Parameterizable
 - ❑ Amenable to appropriate investigation method:
 - In acceptable time, with acceptable effort, with desired accuracy
 - Method also depends on desired results





A Simple Example: Problem Statement (1)

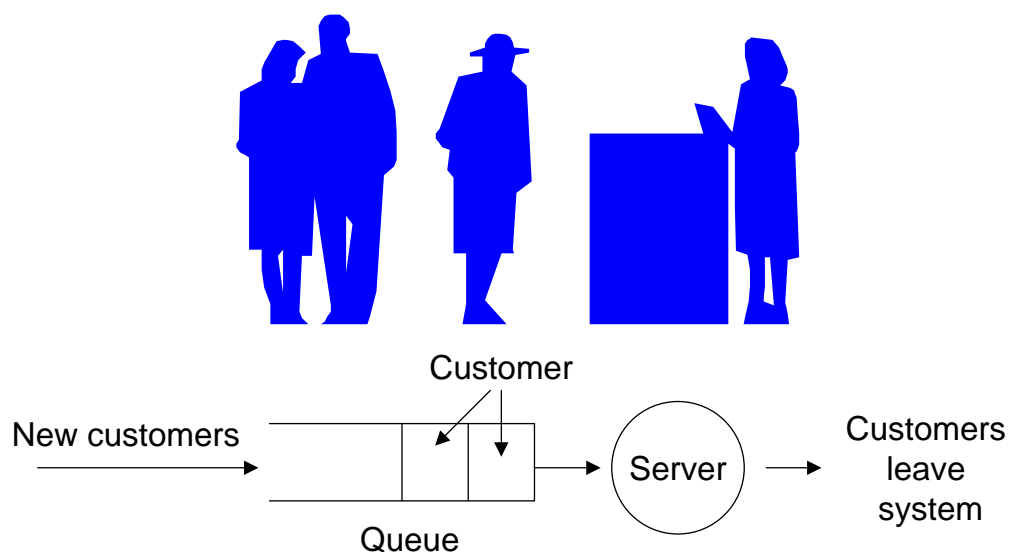
- ❑ Perform a performance evaluation of a checkout counter in a store
- ❑ Objectives – find answers to the following questions:
 - ❑ How long do customers have to wait?
 - ❑ How many customers are in line?
 - ❑ How much of the time is the cashier busy?



- ❑ What are relevant parts?
 - ❑ The person at the counter
 - ❑ People waiting in line
- ❑ What are relevant parameters?
 - ❑ How do customers arrive at the queue?
 - ❑ How long does it take to serve a customer?
 - ❑ How is the queue organized?
- ❑ Metrics = objectives in this example



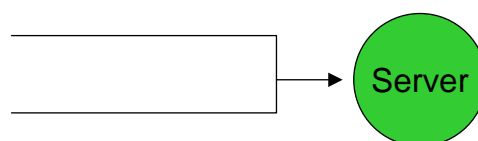
- ❑ Map the real-world system parts to abstract representations



- ❑ How does such a checkout queue work?
- ❑ Server is idle (no customer present)
 - ❑ If customer arrives, customer is serviced immediately, until completion
 - ❑ Server is then busy until customer is completely served
- ❑ Server is busy
 - ❑ If customer arrives, the customer joins the end of the queue
 - ❑ When server becomes idle (a customer has been finished), server checks the queue
 - If one or more customer waits in queue, the first customer leaves the queue and is now serviced
 - If no customer in queue, the server becomes idle

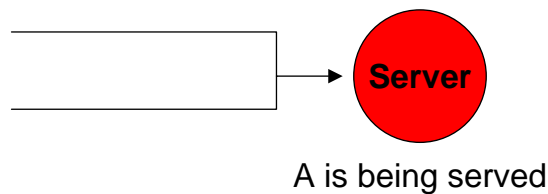


- ❑ Let us start the simulation at some arbitrary time, say $t=0$
- ❑ Initially, the server is idle, no customers are waiting
- ❑ Observe the model in its operation as customers enter and leave the model, as the model changes its state



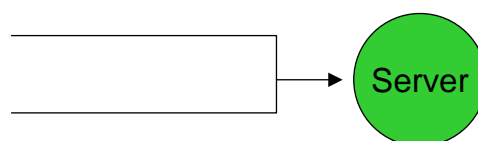
Performing a Simulation: Manual Simulation (2)

- ❑ At some point in time, a customer A arrives, say $t=2.1$
- ❑ The server is now busy
- ❑ Assume this customer needs 1.2 time units to be served



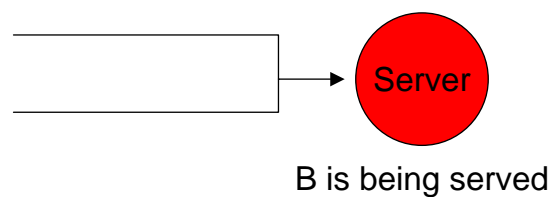
Performing a Simulation: Manual Simulation (3)

- ❑ At $t=3.3$, the customer leaves the system
- ❑ The queue is empty, the server becomes idle again



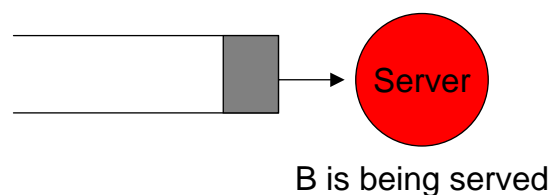
Performing a Simulation: Manual Simulation (4)

- ❑ At $t=4.5$, the next customer B arrives
- ❑ The server is idle, the customer begins service immediately, the server is then busy
- ❑ Assume this customer needs 3.7 time units for service



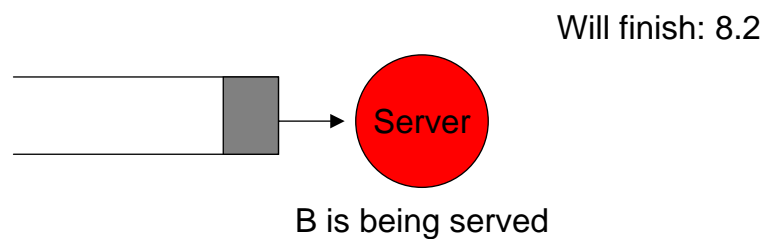
Performing a Simulation: Manual Simulation (5)

- ❑ At $t=4.9$, the next customer C arrives at the counter
- ❑ Server is busy, customer joins the queue
- ❑ Assume this customer will need 1.8 time units for service



Performing a Simulation: Manual Simulation (6)

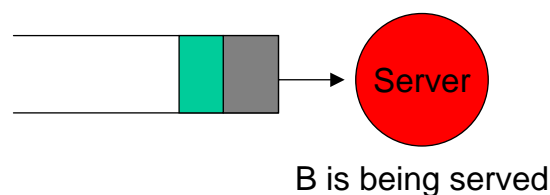
- ❑ When does this third customer start service?
- ❑ Wait a second: When did the other guy finish?
 - ❑ Oh, it arrived at 4.5, it will take 3.7 time units, so it will leave at $t=8.2$
 - ❑ Maybe it would be more convenient to write down the time the currently serviced customer will finish (= time the server can accept the next customer)!



Performing a Simulation: Manual Simulation (7)

- ❑ Did I mention that customer D will arrive at $t=5.6$ (requiring 3.5 units of service time)?
- ❑ So at that time, a new customer enters the queue
 - ❑ I guess the server was busy at that time?
- ❑ Maybe it would be a good idea to write down the time which is represented by these figures!

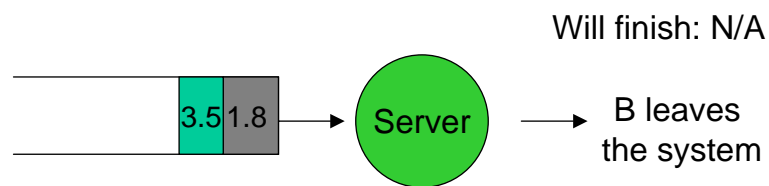
Clock: 5.6



Performing a Simulation: Manual Simulation (8)

- ❑ By the way, the next customer E will arrive at $t=9.3$
- ❑ But that is only after the second customer has already finished its service
 - ❑ So have a look first what happens at $t=8.2$, then consider the next event
 - ❑ At 8.2, customer leaves system, next one is taken into service, and that will finish after (oh, what was it) 1.8 time units
 - ❑ Let's write down the required time units as well

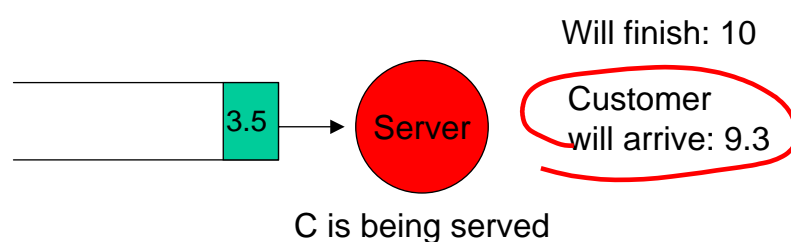
Clock: 8.2



Performing a Simulation: Manual Simulation (9)

- ❑ Start to serve customer C, set the time it will finish
- ❑ Now what was the next thing that will happen?
 - ❑ Add a customer? Finish customer?
 - ❑ Maybe it's a good idea to write down the time the next customer will arrive as well!

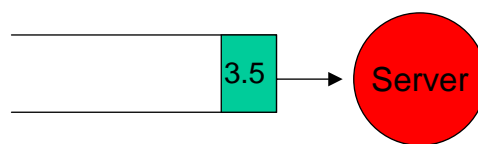
Clock: 8.2



Performing a Simulation: Manual Simulation (10)

- ❑ Compare the time the current customer will finish with the time the next customer will arrive
 - ❑ Next event that changes the state of the model is the arrival of a new customer at $t=9.3$
 - ❑ Set the clock to this time and update the state

Clock: 8.2



Will finish: 10

Customer will arrive: 9.3

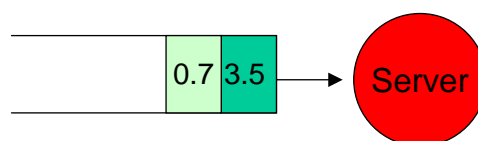
C is being served



Performing a Simulation: Manual Simulation (11)

- ❑ At $t=9.3$, customer E arrives and is put into the queue
 - ❑ E will need, say, 0.7 time units for processing by the server
- ❑ Write down the arrival of the next customer at, say, $t=12.2$
- ❑ Next event to process is the finishing of customer C at $t=10$

Clock: 9.3



Will finish: 10

Customer will arrive: 12.2

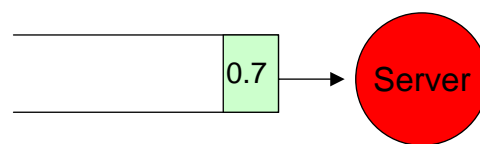
C is being served



Performing a Simulation: Manual Simulation (12)

- ❑ C finishes, remove D from queue and put it on the server
- ❑ Write down the finish time of D, which took 3.5 time units
- ❑ Next event would be at $t=12.2$, arrival of a new customer

Clock: 10



Will finish: 13.5

Customer
will arrive: 12.2

D is being served



Extracting the Structure: What did we do here? (1)

- ❑ You got the idea! Generalize?
- ❑ We observed the model *only* at the points in time when the state of the model changed
- ❑ These points in time were explicitly represented by the *simulation clock* variable
- ❑ The state of the model changed due to two different kinds of events:
 - ❑ Arrival of customers
 - ❑ Customers finishing service and leaving the server



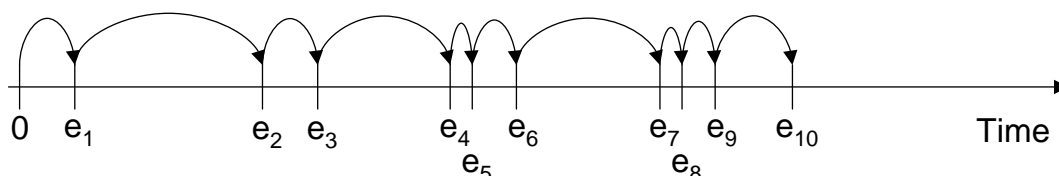
Extracting the Structure: What did we do here? (2)

- ❑ Both kinds of events were processed = the state was manipulated according to the algorithms describing the model (see slide 27)
- ❑ Two variables were used to determine which kind of event is the next one (arrival or departure of customer)
- ❑ In a nutshell: we advanced time from one event to the next, checking to see which kind of event would be the next one
- ❑ This is the essence of the “next-event time advance algorithm”
 - ❑ Note: time was incremented as necessary, not in fixed amounts
 - ❑ Fixed-increment algorithms are also possible, yet rarely used
 - ❑ Result: periods of inactivity are skipped with next-event algorithms



Next-Event Time Advance Algorithm

- ❑ Initialize *simulation clock* to 0
- ❑ Determine time of occurrence of future events (might be infinite for some events)
 - ❑ Might be arbitrarily many – stored in the *future event list*
- ❑ As long as there are events to be processed
 - ❑ Increment the simulation clock to the time of the next, most imminent event
 - ❑ Update the *system state* as required by the occurrence of this event (usually done by an *event routine* specific for each kind of event)
 - ❑ Compute times for future events



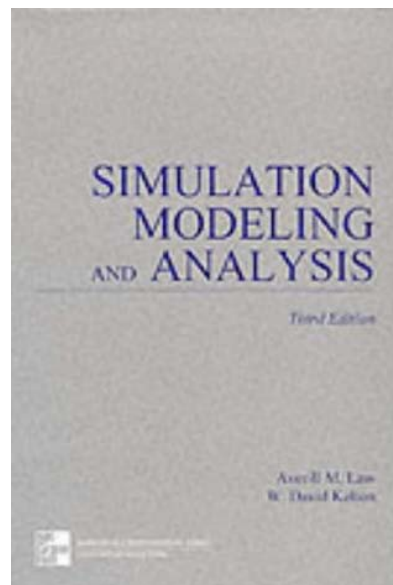
- ❑ Carefully distinguish between
 - ❑ Simulated time:
 - The time as measured by the simulation clock,
 - Virtual time within the simulated system.
 - Units can be chosen arbitrarily
 - ❑ Simulation time:
 - Time that is necessary to run a given simulation
 - Depends on parameters, models, equipment used, accuracy, ...
 - ❑ Ideally, simulation time should be much shorter than simulated time

- ❑ Do not let the common term “simulation clock” confuse you: it is measuring *simulated* time



[LK00] Averill M. Law, W. D. Kelton, *Simulation, Modeling and Analysis*. 784 pp., McGraw-Hill Education, 2000.

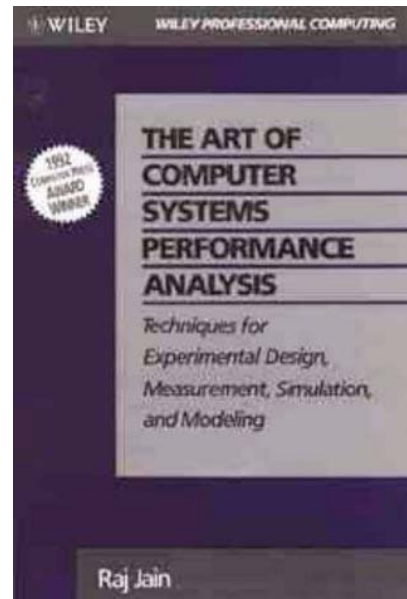
- ❑ General book on simulation, both excellent overview and in-depth treatment. Not oriented towards a single problem area.
- ❑ If you want to buy only one book, get this one. Expensive, though.
- ❑ Main source for the first part of this course!



Some Books Recommendations (2)

[J91] R. Jain, *The Art of Computer Systems Performance Analysis*. Wiley, 1991.

- ❑ Very good book on performance analysis in general
- ❑ Treats simulation as well as mathematical tools (queuing theory) and experimental design
- ❑ Examples heavily focus on computer systems and their architecture.
- ❑ The treatment of simulation is not as thorough as in [LK00], but also very good.
- ❑ Unfortunately, currently out of print



Further References

- ❑ H. Karl. *Praxis der Simulation*. course slides, Universität Paderborn.
Good course on performance evaluation by simulation, material is included in our performance evaluation course with kind permission by H. Karl
- ❑ A. Varga, *OMNeT++: Object-Oriented Discrete Event Simulator*, <http://www.hit.bme.hu/phd/vargaa/omnetpp.htm>
Homepage for the simulation tool recommended in this class. Contains the manual in an online version, also links to other web sites relevant to this simulation tool.
- ❑ B. Stroustrup, *The C++ Programming Language*. Addison-Wesley, 2000.
Good book on C++, yet not for the faint of heart. :o) If you have mastered it, you have probably understood C++ quite well. Make sure you read a recent edition, though – old versions to not match the current language definition.
- ❑ M. A. Weiss. *C++ for Java Programmers*. Addison-Wesley, 2003.
A C++ textbook for readers with knowledge of the Java programming language.
- ❑ CS123 TA Staff. *Java to C++ Transition Tutorial*.
<http://www.cs.brown.edu/courses/cs123/javatoc.shtml>

