Simulation of data communications networks

T-110.6130 Systems Engineering in Data Communications Software
Zhonghong Ou
Post-doc researcher
Data Communications Software Lab,
School of Science and Technology, Aalto University
Agenda

- **What**
- **Why**
- **Where**
- **When**
- **Who**
- **How**
What is simulation?

• Simulation is the imitation of some real thing, state of affairs, or process. (wikipedia)
  

• Simulation is an imitation of a real work process or system over time.

• Simulation is the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system.

  --R.E.Shannon
Example

Human-in-the-loop simulation of outer space. (wikipedia)
Key characteristics

• Acquisition of valid source information about the relevant selection of key characteristics and behaviors.

• Use of simplifying approximations and assumptions within the simulation.
  – the assumptions usually take the form of mathematical or logical relations.

• Fidelity and validity of the simulation outcomes.
Computer simulation

- A computer program, or network of computers, that attempts to simulate an abstract model of a particular system.
- Been used in:
  - Natural systems
    - Physics (computational physics),
    - Astrophysics,
    - Chemistry,
    - Biology.
  - Human systems
    - Economics,
    - Psychology,
    - Social science,
    - Engineering.
Classification (coarse-grained)

• **Physical simulation** refers to simulation in which the real thing is substituted for physical objects, which are often chosen because they are smaller or cheaper than the actual object or system.

• **Interactive simulation** is a special kind of physical simulation, often referred to as a human in the loop simulation, in which physical simulations include human operators, such as in a flight simulator or a driving simulator.
  – Human in the loop simulations can include a computer simulation as a so-called synthetic environment;
  – Hardware in the loop simulation;
  – Semi-physical simulation;
  – Half-object simulation.
Classification (fine-grained)

- **Determinacy:**
  - **Stochastic (random)**
    - There exists some indeterminacy in its future evolution described by probability distributions;
    - Even if the initial condition (or starting point) is known, there are many possibilities the process might go to, but some paths may be more probable and others less so.
    - E.g. stock market, exchange rate.
  - **Deterministic**
    - Given a particular input, it will always produce the same output;
    - E.g. mathematical function.

- **Steadiness:**
  - **Static (steady)**
    - Relating to a given instant of time, time not considered;
    - Used for estimation of quantities from a given distribution;
  - **Dynamic**
    - Development of the system based on time;
    - Output changes in a system in response to (usually changing) input signals.
Classification (fine-grained, cont.)

• Continuity:
  – Continuous
    • In mathematics, a continuous function is a function for which, intuitively, small changes in the input result in small changes in the output.
    • Considering a function \( h(t) \) which describes the height of a growing flower with respect to \( t \).
    • “Analog” system.
  – Discrete
    • The objects studied in discrete simulation—such as integers, graphs, and statements in logic—do not vary smoothly, but have distinct, separated values.
    • “Digital” system.
    • Special case: discrete event simulation (DES)
      • Managing events in time. The simulator maintains a queue of events sorted by the simulated time they should occur, reads the queue and triggers new events as each event is processed. It is not important to execute the simulation in real time, but rather to be able to access the data produced by the simulation, to discover logic defects in the design etc. Most computer, logic-test and fault-tree simulations are of this type.

• Locality:
  – Distributed
    • Using a network of interconnected computers to accomplish a common objective or task;
    • Simulations dispersed across multiple host computers.
  – Local
    • Using a single computer to conduct the simulation.
Classification (fine-grained, cont.)

Simulation

Determinacy
- Deterministic
- Stochastic

Steadiness
- Static
- Dynamic

Continuity
- Continuous
- Discrete

Locality
- Distributed
- Local
When to use simulation?

- **Modeling** of natural systems or human systems in order to gain insight into their functioning.

- **Imitating** technology for performance optimization, safety engineering, testing, training and education, **exploring** and **gaining** new insights into new technology.

- **Showing** the eventual real effects of alternative conditions and courses of action.

- **Estimating** the performance of systems too complex for analytical solutions. The real system **cannot be engaged** for reasons, e.g. it may not be accessible, it may be dangerous or unacceptable to engage, or it may simply not exist.
Agenda

- What
- Why
- Where
- When
- Who
- How
Analysis versus simulation

- Traditionally, the formal modeling of systems has been via a mathematical model, which attempts to find analytical solutions enabling the prediction of the behavior of the system from a set of parameters and initial conditions.

- Computer simulation is often used as an adjunct to, or substitution for, modeling systems for which simple closed form analytic solutions are not possible.
  - Common feature: attempting to generate a sample of representative scenarios for a model in which a complete enumeration of all possible states would be prohibitive or impossible.
Analysis versus simulation (cont.)

• Analysis (two steps):
  – Modeling of the system as a time-dependent stochastic process;
  – Analytical solution of the model.

• Simulation (four steps):
  – Modeling of the system as a dynamic stochastic process;
  – Generating realizations of the process;
  – Collecting data (measurement);
  – Statistically analyzing the data and drawing conclusions.

• Common features:
  – Modeling is common
    • Differing with respect to details;
    • Mathematical analysis usually utilizing restrictive assumptions.
Mathematical analysis

• Pros:
  – Results obtained quickly;
  – Results exact;
  – Giving insight of the system;
  – Allowing optimization.

• Cons:
  – Requiring restrictive assumptions;
  – Analysis complex;
  – Results (might be) limited to equilibrium state, or average values.
Simulation

• Pros:
  – No constraints in the model building;
  – Enabling complex system;
  – Modeling usually straightforward.

• Cons:
  – Time- and energy-consuming;
  – Results imprecise (precision could be improved by multiple iterations)
  – Getting insight more difficult;
  – Optimization more difficult (maybe limited to the trial of a few parameter combinations).
Agenda

- What
- Why
- Where
- When
- Who
- How

19.9.2011
Conductor of simulation

- You
- Other co-operators
Agenda

What

When

Why

Who

Where

How
Place for simulation

- Single computer.
- Cluster of computers.
Agenda

- What
- Why
- Where
- When
- Who
- How
How to build simulation model

• Calibration.
• Verification.
• Validation
Model calibration

- Can be achieved by adjusting any available parameters in order to adjust how the model operates and simulates the process.

- For example, in the simulation of a peer-to-peer (P2P) network, the typical parameters include joining rate, leaving rate, (churn rate), exchanging rate, publishing rate, lookup rate etc.

- These parameters influence the behaviors of the P2P network, for instance, lookup success ratio, average traffic load (bytes), and average number of messages.
Model verification

• Can be achieved by obtaining output data from the model and comparing it to what is expected from the input data.

• For example, in the simulation of a peer-to-peer (P2P) network, there are certain expectations with regard to the lookup success ratio, given the churn rate, exchanging rate etc.

• In some cases the system or some part of it can be analyzed under simplified assumptions.

• The simulation can be run under the same assumptions; at least the results should match with each other.

• In real-life, this is usually not so realistic as it is not easy to get the theoretical results given the input data.
Model validation

- Can be achieved by comparing the results with what’s expected based on historical data from the study area.

- It is the best and most reliable method.

- Ideally, the model should produce similar results to what has happened historically.

- If model output values are drastically different than historical values, it probably means there’s an error in the model.

- Often difficult to apply, e.g. there is no real system, or measurements are too expensive to conduct.
What if…

• No simplified mathematical system available, neither no historical measurements existing…

• Expert intuition
  – a common and practical method;
  – “brain storming” with people who know the system in order to define sensible assumptions and input data;
  – an expert can easily recognize “impossible” results;
  – not so reliable.
Selection of languages

General purpose languages (C/C++, Java…)
- Most users have knowledge of at least one language;
- Available on most computers;
- Code easily transported;
- Low cost of the programs;
- Code running faster;
- Flexible.
- Requires a lot of programming work;
- Susceptible to errors

Simulation languages (general purpose, GASP, GPSS, SIMSCRIPT, Simula)
- Supporting many features needed in the programming of a simulation model;
- Shorter development time;
- Lower cost
- Programming with the aid of the modeling constructs of the language.

Simulation languages (data communications network oriented, OPNET, QualNet, NS2, OMNET++)
- Containing network building blocks;
- Developed specifically for the simulation of data communications networks.
Selection of simulators

- ns-2;
- ns-3;
- GloMoSim;
- OPNet;
- QualNet;
- OMNet++;
- NAB;
- J-Sim;
- SimPy;
- NetHawk EAST;
- SENSE;
- Sidh;
- TOSSIM;
- ATEMU;
- Arvora;
- EmStar;
- MATLAB.
ns-2

- The de facto standard for network simulation. Its behavior is highly trusted within the networking community.

- It was developed at ISI/USC (Information Science Institute, University of Southern California), and was supported by the DARPA (Defense Advanced Research Projects Agency) and NSF (National Science Foundation).

- ns-2 is a discrete-event simulator organized according to the OSI model and primarily designed to simulate wired networks.

- The support for wireless networking had been brought by several extensions.

- Simulations are based on a combination of C++ and Otcl (object oriented extension of Tcl created by David Wetherall at MIT). In general, C++ is used for implementing protocols and extending the ns-2 library. Otcl is used to create and control the simulation environment itself, including the selection of output data. Simulation is run at the packet level, allowing for detailed results.

- This design choice was originally made to avoid unnecessary recompilations if changes are made to the simulation set-up. The design of ns-2 trades off simulation performance for the saving of recompilations, which is questionable if one is interested in conducting scalable network simulations. Therefore, ns-2 is currently undergoing a major redesign, one of the main development goals of its successor, ns-3, is the improvement of simulation performance.
ns-2 (cont.)

- Weaknesses:
  - Lack of modularity;
  - Inherent complexity (ns-2 was candidate to be the basis for the Qualnet simulator but got eventually rejected);
  - High consumption of computational resources. A harmful consequence is that ns-2 lacks scalability, which impedes the simulation of large networks (ns-2 is typically used for simulations consisting of no more than a few hundreds nodes).
ns-3

• Like its predecessor, ns-3 relies on C++ for the implementation of the simulation models.

• However, ns-3 no longer uses oTcl scripts to control the simulation, thus avoiding the problems which were introduced by the combination of C++ and oTcl in ns-2.

• Instead, network simulations in ns-3 can be implemented in pure C++, while parts of the simulation optionally can be realized using Python as well.

• Moreover, ns-3 integrates architectural concepts and code from GTNetS, a simulator with good scalability characteristics.

• These design decisions were made at expense of compatibility. In fact, ns-2 models need to be ported to ns-3 in a manual way.

• Besides performance improvements, the feature set of the simulator is also about to be extended. For example, ns-3 is slated to support the integration of real implementations’ code by providing standard APIs, such as Berkeley sockets or PÖIX threads, which are transparently mapped to the simulation.
GloMoSim

- GloMoSim was developed in 1998 at UCLA (University of California, Los Angeles) for mobile wireless networks.

- It is written in Parsec, which is an extension of C for parallel programming, and benefits from Parsec’s ability to run on shared-memory symmetric processor (SMP) computers. New protocols and modules must be written in Parsec too.

- Respects OSI model. GloMoSim is designed to be extensible, with all protocols implemented as modules in the GloMoSim library.

- Capable of supporting parallel environment. The network is split into different sub-networks, each of them being simulated by distinct processors. The network is partitioned in such a way that the number of nodes simulated by each partition is homogeneous.

- GloMoSim uses an object-oriented approach. However, the designers realized that a purely object-oriented approach would not be scalable. Instead, GloMoSim partitions the nodes, and each object is responsible for running one layer in the protocol stack of every node for its given partition. This helps to ease the overhead of a large network.

- Weaknesses:
  - While effective for simulating IP networks, it is not capable of simulating any other type of network.
  - Lack of a good and in-depth documentation. GloMoSim stopped releasing updates in 2000. Instead, it was chosen as the core of the commercial QualNet simulator, and is now updated as QualNet.
GloMoSim (cont.)

Fig. GloMoSim architecture
OPNet

- Optimized Network Engineering Tools (OPNet) is a discrete-event network simulator first proposed by MIT in 1986, written in C++. It is a well-established and widely used commercial suite for network simulation.

- It uses a hierarchical model to define each aspect of the system.
  - The top level consists of the network model, where topology is designed;
  - The next level is the node level, where data flow models are defined;
  - A third level is the process editor, which handles control flow models.
  - Finally, a parameter editor is included to support the three higher levels.

- The hierarchical model results in an event queue for the discrete event simulation engine and a set of entities representing the nodes that will be handling the events. Each entity in the system consists of a finite state machine which processes the events during simulation.
OPNet (cont.)

• Pros:
  – Capable of executing and monitoring several scenarios in a concurrent manner;
  – Supporting the use of modeling different sensor-specific hardware, such as physical-link transceivers and antennas;
  – Supporting custom packet formats;
  – Graphical interface to develop models, also can be used to model, graph, and animate the resulting output.

• Cons:
  – Suffering from the same object-oriented scalability problems as ns-2.
  – Not supporting as many protocols as ns-2 because of its commercial feature.
QualNet

- A commercial ad hoc network simulator based on the GloMoSim core.

- It extends the GloMoSim offering by bringing support, a decent documentation, a complete set of user-friendly tools for building scenarios and analyzing simulation output.

- QualNet also extends the set of models and protocols supported by the initial GloMoSim distribution.

- As it is built on top of GloMoSim, QualNet is written in Parsec.
QualNet (interface)
OMNeT++

• In contrast to ns-2 and ns-3, OMNeT++ is not a network simulator by definition, but a general purpose discrete event-based simulation framework, based on object-oriented design.

• It is mostly applied to the domain of network simulation, given the fact that with its INET package it provides a comprehensive collection of Internet protocol models.
  – The INET Framework contains models for several wired and wireless networking protocols, including UDP, TCP, SCTP, IP, IPv6, Ethernet, PPP, 802.11, MPLS, OSPF, and many others.

• In addition, other model packages such as the OMNeT++ Mobility Framework and Castalia facilitate the simulation of mobile ad hoc networks or wireless sensor networks.

• OMNeT++ simulations consist of so-called simple modules (C++) which realize the atomic behavior of a model, e.g. a particular protocol. Multiple simple modules can be linked together and form a compound module.

• OMNeT++ utilizes NED (NEtwork Description) language to combine the simple modules into compound modules and define the network topologies. NED is transparently rendered into C++ code when the simulation is compiled as a whole.
NAB

- Network in A Box (NAB) is a discrete event simulator developed at EPFL (Lausanne, Switzerland).

- NAB is dedicated to MANETs simulation.

- NAB is focusing on scalability and visualization and features a very realistic mobility model (a constrained waypoint based on city maps).

- NAB’s design is node-oriented (and object-oriented); that is each node is represented by an object. It is written in OCaml. It is open source.
J-Sim

- J-Sim, developed at University of Illinois at Urbana-Champaign, is a general purpose Java-based simulator modeled after ns-2.

- Unlike ns-2, J-Sim uses the concept of components, replacing the notion that each node should be represented as an object.

- J-Sim uses three top level components, each component is broken into different parts and modeled differently within the simulator:
  - the target node (which produces stimuli);
  - the sensor node (that reacts to the stimuli);
  - the sink node (the ultimate destination for stimuli reporting).

- Pros:
  - Component-based architecture scales better than the object oriented model used by ns-2 and other simulators.
  - Applications may be simulated, and there is support for the connection of real hardware sensors to the simulator.

- Cons:
  - Relatively complicated to use;
  - Faces its share of inefficiencies. Java, in general, is arguably less efficient than many other languages.
JiST

- Java in Simulation Time (JiST) allows the implementation of network simulations in standard Java. It is mostly used in conjunction with SWANS, a simulator for mobile ad hoc networks built on top of JiST.

- Network simulations in JiST are made up of entities which represent the network elements, for example nodes, with simulation events being formed by method invocations among those entities.

- The entities advance the simulation time independently by notifying the simulation core.

- While the code inside an entity is executed like any arbitrary Java program, only the interactions between the individual entities are carried out in simulation time.

- The interactions between entities correspond to synchronization points and facilitate the parallel execution of code at different entities, resulting in a potential performance gain.

- The official development of JiST has stalled, as it is no longer maintained by its original author, Rimon Barr.
SimPy

- SimPy is a process-oriented discrete-event simulator;

- Unlike the other simulators, no public available network models exist for SimPy. Instead, it is a bare simulation API written in Python.

- In SimPy, the basic simulation entities are processes. They are executed in parallel and may exchange Python objects among each other.

- Most processes include an infinite loop in which the main actions of the process are performed.

- Besides abstractions for processes and the related exchange of objects, SimPy provides instructions for the synchronization of simulation processes and commands for the monitoring of simulation data.
NetHawk EAST

- Environment for Automated Systems Testing (EAST) is a **test automation and traffic generation** tool that allows users to emulate/simulate one or more network elements in the telecommunications network.

- EAST can be used for **feature** testing as well as **load** testing.

- EAST provides an easy to use, intuitive **GUI** front-end that is consistent across all protocols.

- The EAST GUI is JAVA based and hence platform independent.

- NetHawk was originally based in **Oulu**, and was acquired by EXFO (Canadian company) on March 12, 2010.
NetHawk EAST (cont.)
Matlab

• MATLAB (matrix laboratory) is a **numerical computing** environment and fourth-generation programming language.

• It was originally designed to solve linear algebra type problems using matrices.

• Developed by MathWorks, MATLAB allows:
  – matrix manipulations;
  – plotting of functions and data;
  – implementation of algorithms;
  – creation of user interfaces, and interfacing with programs written in other languages, including C/C++, Java, SQL, and Fortran etc.

• Commercial product, but widely used in industry and academia.
  – Many algorithms and toolboxes freely available
# Granularity and Mobility

<table>
<thead>
<tr>
<th>Name</th>
<th>Granularity</th>
<th>Metropolitan mobility</th>
</tr>
</thead>
<tbody>
<tr>
<td>ns-2</td>
<td>Finest</td>
<td>Support</td>
</tr>
<tr>
<td>DIANEmu</td>
<td>Application-level</td>
<td>No</td>
</tr>
<tr>
<td>Glomosim</td>
<td>Fine</td>
<td>Support</td>
</tr>
<tr>
<td>GTNets</td>
<td>Fine</td>
<td>No</td>
</tr>
<tr>
<td>J-Sim</td>
<td>Fine</td>
<td>Support</td>
</tr>
<tr>
<td>Jane</td>
<td>Application-level</td>
<td>Native</td>
</tr>
<tr>
<td>NAB</td>
<td>Medium</td>
<td>Native</td>
</tr>
<tr>
<td>OMNet++</td>
<td>Medium</td>
<td>No</td>
</tr>
<tr>
<td>OPNet</td>
<td>Fine</td>
<td>Support</td>
</tr>
<tr>
<td>QualNet</td>
<td>Finer</td>
<td>Support</td>
</tr>
<tr>
<td>SWANS</td>
<td>Medium</td>
<td>-</td>
</tr>
</tbody>
</table>
## Parallelism and interface

<table>
<thead>
<tr>
<th>Name</th>
<th>Parallelism</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>ns-2</td>
<td>No</td>
<td>C++/OTCL</td>
</tr>
<tr>
<td>DIANEmu</td>
<td>No</td>
<td>Java</td>
</tr>
<tr>
<td>Glomosim</td>
<td>SMP/beowulf</td>
<td>Parsec (C-based)</td>
</tr>
<tr>
<td>GTNets</td>
<td>SMP/beowulf</td>
<td>C++</td>
</tr>
<tr>
<td>J-Sim</td>
<td>RMI-based</td>
<td>Java</td>
</tr>
<tr>
<td>Jane</td>
<td>No</td>
<td>Java</td>
</tr>
<tr>
<td>NAB</td>
<td>No</td>
<td>OCaml</td>
</tr>
<tr>
<td>OMNet++</td>
<td>MPI/PVM</td>
<td>C++</td>
</tr>
<tr>
<td>OPNet</td>
<td>Yes</td>
<td>C</td>
</tr>
<tr>
<td>pdns</td>
<td>beowulf</td>
<td>C++/OTCL</td>
</tr>
<tr>
<td>QualNet</td>
<td>SMP/beowulf</td>
<td>Parsec (C-based)</td>
</tr>
<tr>
<td>SWANS</td>
<td>No</td>
<td>Java</td>
</tr>
</tbody>
</table>
### Popularity and license

<table>
<thead>
<tr>
<th>Name</th>
<th>Popularity</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>ns-2</td>
<td>88.8%</td>
<td>Open source</td>
</tr>
<tr>
<td>GloMoSim</td>
<td>4%</td>
<td>Open source</td>
</tr>
<tr>
<td>OPNet</td>
<td>2.61%</td>
<td>Commercial</td>
</tr>
<tr>
<td>QualNet</td>
<td>2.49%</td>
<td>Commercial</td>
</tr>
<tr>
<td>OMNet++</td>
<td>1.04%</td>
<td>Free for academic and educational use</td>
</tr>
<tr>
<td>NAB</td>
<td>0.48%</td>
<td>Open source</td>
</tr>
<tr>
<td>J-Sim</td>
<td>0.45</td>
<td>Open source</td>
</tr>
<tr>
<td>SWANS</td>
<td>0.3%</td>
<td>Open source</td>
</tr>
<tr>
<td>GTNots</td>
<td>0.13</td>
<td>Open source</td>
</tr>
<tr>
<td>pdus</td>
<td>&lt; 0.1%</td>
<td>Open source</td>
</tr>
<tr>
<td>DIANEmu</td>
<td>&lt; 0.1%</td>
<td>Free</td>
</tr>
<tr>
<td>Jane</td>
<td>&lt; 0.1%</td>
<td>Free</td>
</tr>
</tbody>
</table>
Popularity

Simulation result of 2000-2005 proceedings of the MobiHoc conference
Pitfalls of simulation

• Simulation setup
  – Simulation type:
    • Dynamic vs. steady-state (execute one type of simulation and report results on the other type of simulation).
  – Model validation & verification:
    • Execute simulations with a model that has not been validated in the specific environment.
  – PRNG (Pseudo Random Number Generator) validation & verification:
    • Someone [9] estimates that the NS-2 PRNG is only valid for several thousand numbers.
  – Variable definition:
    • There are 674 variables defined in the ns-default.tcl file of NS-2.27.
  – Scenario development:
    • Lack of independence between parameters;
    • Lack of rigorous scenarios, no benchmark scenarios.
Simulation execution

• Setting the PRNG seed:
  – Not setting properly, NS-2 uses a default seed of 12345 for each simulation run;

• Scenario initialization:
  – Most simulations start with empty caches, queues, and tables, determining and reaching the steady-state level of activity is part of the initialization.

• Metric collection:
  – Output needs to be in line with the required granularity.
Output analysis

• Single set of data:
  – Taking the first set of results from a simulation and accepting the results as “truth”.

• Statistical analysis:
  – Not using the correct statistical formulas with the different forms of output.

• Confidence intervals:
  – A culmination of several of the previous analysis issues.
  – Confidence intervals account for the randomness and varied output from a stochastic simulation.

• Publishing:
  – The lack of consistency in publishing simulation based study results directly impacts the trustworthiness of these studies.
  – A new researcher cannot repeat the studies to start his or her own follow-on research.
Conclusion

• Each of the pitfalls discussed takes away from the goals of making the research:
  – repeatable,
  – unbiased,
  – rigorous,
  – statistically sound.

• It is still a long way to go!!

http://www.youtube.com/watch?v=cddlgb1nGJ8
References

Contact Information

• Course web page:
  • https://noppa.tkk.fi/noppa/kurssi/t-110.6130/
• Contact email:
  • zhonghong.ou@aalto.fi
• Office hour:
  • Fri 10-11 room A 109
• Questions & Suggestions?

Zhonghong Ou