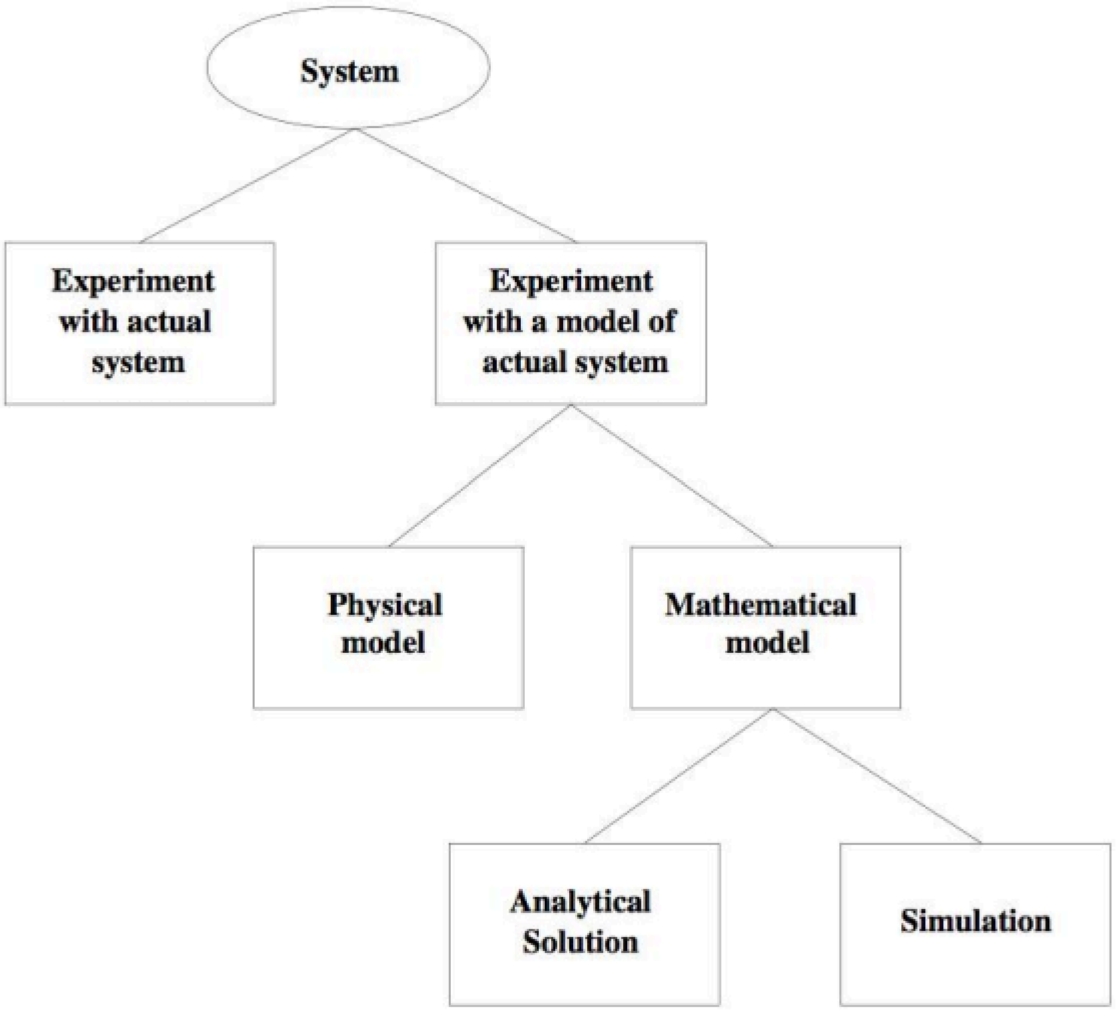
11.Simulating a Distributed System

We can emulate and simulate a distributed system.

Why we simulate/study a system

When an engineer start to study a system there are two possibilities:

- Study the real system(testing)
- Study a model of the system(physical or mathematical). Used to discover wrong assumptions of the system


System
Experiment with actual system
Experiment with a model of actual system
Physical model
Mathematical model
Analytical Solution
Simulation

A physical model is a small realization of the system meanwhile a mathematical model is the description of the system characteristics. A mathematical model is divided in:

- Analytical solution: say if something respect some characteristics

- Simulation: creating a model that behave like or similar to the expected system to permit you to study the system
  Another type of simulation is emulation.

Network emulation

In between experimenting with actual system and using a model of the system

- Use the system on a “simulated” environment/network
- Addresses the difficulty of testing distributed systems in large scale/complex deployments
  Emulation differs from simulation in that a network emulator appears to be a network to the OS
  It can be accomplished by introducing a component/device that alters packet flow in a way
  that imitates the behaviour of the environment being emulated (e.g., a WAN or a wireless network)
- Often coupled with a node emulation software (i.e., a virtual machine like UML, VMWare, VirtualBox)
  Network emulation in pratice:
- Mininet
- FreeBSD Dummynet (used in Emulab), Linux NetEm
  May introduce packet delay, loss, duplication, reordering
- Other more user friendly tools like marionnet, netkit,...
  The network can influence the results of your simulation so you emulate also the network to also have control on it. The application is the system and the deployment is done in a system that is simulated. The emulators permit the simulation of various situations to permit the test of the application on different situations similar to real settings and let you to tune the system.

Analyzing before building

Often important to “analyze before build”. Analysis requires a model of the system
Various classes of models:

- Analytical vs. operational
- Discrete vs. continuous
- Deterministic vs. stochastic
  Simulation is a special form of analysis in which a history of system execution is

obtained and analyzed. We have an operational model and we describe the system and the variable and consider deterministic behaviour.

Discrete event simulation

Simple method to analyze the performance of a system using a discrete, deterministic, operational model
Involves different elements:

- A list of events (timestamped objects)
- A simulation clock
- A set of state variable and performance indicators
- An event processing function
  - Takes an event as a parameter and process it by updating the value of state variables and performance indicators and creating new events
  Operates as follow:
  forever
- keep the first event from the list and remove it
- advance the clock to the timestamp of the event
- pass the event to the event processing function
  A software that implements the loop above by simplifying the job of writing discrete event simulations. Usually provides a library of basic, general purpose elements(E.g., random number generators, containers, etc.)
  Plus a library of existing models(E.g., hosts, routers, switch, etc.)
  Examples (relevant for networks/distributed systems)
- OpNet, QualNet, JiST/Swans, Parsec/Glomosim, J-Sim, Ns2,OMNeT++
  The simulator describe on the time as not all time is considered bit it skip to the time there are events to consider and analyze them.

OMNeT++ Simulator

Provides a library that describe the various natural elements that compose a distributed system.
It is opensource, well documented, continuously updated and used in the most common commercial simulator. It provides the loop and the set of components and the components model. It says that a system need to be described by a single function, everything need to be in this function. OMNET give a way to take the function and export it in a set of modules that are similar to an object oriented

langiuage objects.
Provides:

- A component model to easily and effectively structure complex simulations through reusable components
- A C++ class library including the simulation kernel and utility classes (for random number generation, statistics collection, topology discovery etc) to build such components
- An infrastructure to assemble simulations from these components and configure them (NED language, ini files)
- Runtime environments for simulations (Tkenv, Cmdenv)
- An Eclipse-based simulation IDE for designing, running and evaluating simulations

The OMNeT++ component model

We model the distributed system introducing models(simple and compound).
A module is something that as a set of gates(input/output) through gates we can connect modules between them. Through gates we send messages. Simple modules have a behaviour that is described by their C++ class. Instead a compound module is a module that is composed of various simple module. There is a language to describe the compound module and how the modules are attached between them. A system is described as a compound module that contains all the modules of the system(highest level module). Modules have parameters whose values can be given into the omnetpp.ini file and change at each run
A simple module (NED interface)

```
//
// Ethernet CSMA/CD MAC
//
simple EtherMAC {
        parameters:
                string address; // others omitted for brevity
        gates:
                input phyIn;// to physical layer
                output phyOut;// to physical layer
                input llcIn;// to EtherLLC or higher layer
                output llcOut;// to EtherLLC or higher layer
}
```

For each simple module you have to write a C++ class with the module’s
name, which extends the library class cSimpleModule

- You can redefine several methods, main ones are initialize(), finish(), and handleMessage(cMessage *msg)(invoked every time a packet is entering/exiting the module)
- initialize() and handleMessage(cMessage msg) always need redefinition
  You have also to register the class via the Define_Module(module_name) macro
  NED parameters can be read using the par(const char *paramName) method
  You can send messages to other (connected) modules using the
  send(cMessage* msg, char *outGateName) methodFor each simple module you have to write a C++ class with the module’s
  name, which extends the library class cSimpleModule

A compound module (in NED)

```
//
// Host with an Ethernet interface
//
module EtherStation {
        parameters: ...
        gates: ...
                input in;     // for connecting to switch/hub, etc
                output out;
        submodules:
                app: EtherTrafficGen;
                llc: EtherLLC;
                mac: EtherMAC;
        connections:
                app.out --> llc.hlIn;
                app.in <-- llc.hlOut;
                llc.macIn <-- mac.llcOut;
                llc.macOout --> mac.llcIn;
                mac.phyIn <-- in;
                mac.phyOut --> out;
}
```

A message can represent a real packet in the system or a signal of the system as some links between modules can represent invokations to some functions in the modules. The cMessage represent packets or operations invoketions, so they represent events in the system that have to be managed by the destination link of the gate we are sending the package to. This let us breaking the monolithic function permitting the invokation of operations in different modules.

Message classes

They are subclasses of the cMessage library class
You can define them using a special, very simple language. E.g.,

```
message NetworkPacket {
        fields:
                int srcAddr;
                int destAddr;
}
```

Collecting results

While the simulation is going on you collect a log of the operations you are doing to understand better the behaviour of the system.
The simulation results are recorded into output vector (.vec) and output scalar (.sca) files

- This is done by simple modules in C++
  Output vectors capture behavior over time
- An output vector file contains several output vectors, each being a named series of (timestamp, value) pairs
- You can configure output vectors from omnetpp.ini(You can enable or disable recording individual output vectors, or limit recording to a certain simulation time interval)
  Output scalar files contain summary statistics
- E.g., number of packets sent, average hop-to-hop delay
  The OMNeT++ library also includes classes to record statistics and organize them. E.g., cLongHistogram

Random numbers

Often required to generate data randomly
OMNeT++ provides a configurable number of RNG instances (that is, streams), which can be freely mapped to individual simple modules in omnetpp.ini.

- This means that you can set up a simulation model so that all traffic generators use global stream 0, all MACs use global stream 1 for backoff calculation, and physical layer uses global stream 2 and global stream 3 for radio channel modelling
- Seeding can be automatic or manual, manual seeds also come from the ini file
  Several distributions are supported and they are available from both NED and C++
  Non-const module parameters can be assigned random variates like exponential(0.2), which means that the C++ code will get a different number each time it reads the parameter
  During a simulation you use random number generators to simulate the external agents of the system or if you want to simulate some situations where the system does things wrong.