System Performance Evaluation

Georgia Koutsandria

Internet of Things A.A. 18-19

Prof. Chiara Petrioli
Dept. of Computer Science
Sapienza University of Rome

System Performance Evaluation

System Performance Evaluation

- Allows to obtain the highest performance at the lowest cost.
- Allows performance comparison of a number of alternative designs/solutions to find the best one.
- Gives good insights on how well a system is performing and whether any improvements need to be made.
    - Useful at any stage of the system's life cycle, i.e., design, manufacturing, use, upgrade, etc..

A systematic aproach

- Most performance problems are unique.
- Evaluation techniques used for one problem generally cannot be used for a different problem.

- **Steps common to all performance evaluation projects:**

1. State goals and define the system under evaluation
    - Define the boundaries of the system.

A systematic aproach

2. List services and outcomes
   - Each system provides a set of services
   - E.g., A computer network allows its users to send packets to specified destinations
   - A list of services and possible outcomes is useful in selecting the right metrics and workloads.

3. Select metrics
   - Select the criteria(metrics) to compare the performance
   - E.g., delay, accuracy, speed etc..

A systematic aproach

4. List of parameters that affect the performance
    - System parameters (hardware and software)
    - Workload parameters (depend on users' requests)

5. Select factors to study
    - Some parameters will be varied during the simulation (factors) and will get different values (levels)

6. Select evaluation technique
    - Analytical modeling
    - Simulation
    - Real test-bed

A systematic aproach

7. Select Workload
   - A list of service requests to the system
   - Analytical modeling: A probability of various requests
   - Simulation: Trace of requests measured on a real system
   - Test-bed: Scripts to be executed on the system.

8. Design Experiments
   - Decide on a sequence of experiments that offer maximum information with minimal effort
     - Varying number of factors and levels to determine their relative effect.

A systematic aproach

9. Analyze and Interpret Data
   - The analysis procudes results (not conclusions)
   - Each repetition of an experiment gives has a different outcome.

10. Present results
    - They should be presented in a manner that is easily understood, e.g., in a graph form
      - If it is needed, redefine system boundaries, included other factors and performance metrics…(several cycles).

Selecting an evaluation technique

Criterion	Analytical modeling	Simulation	Measurement
Stage	Any	Any	Postprototype
Time required	Small	Medium	Varies
Tools	Analysts	Computer Languages	Instrumentation
Accuracy[a]	Low	Moderate	Varies
Trade-off evaluation	Easy	Moderate	Difficult
Cost	Small	Medium	High
Scalability	Low	Medium	High

a In all cases, result may be misleading or wrong.

Selecting an evaluation technique

- Three rules of validation:
  1. Do not trust the results of a simulation model until they have been validated by analytical modeling or measurements.

  2. Do not trust the results of an analytical model until they have been validated by a simulation model or measurements.

  3. Do not trust the results of a measurement until they have been validated by simulation or analytical modeling.

Selecting performance metrics

- For each performance study, a set of performance criteria must be chosen.
- Time to execute a task
    - Execution time, response time, latency
- Number of tasks per day, hour, sec, ns, etc.
    - Throughput, bandwidth.

Aircraft	DC to Paris	Speed (mph)	Passengers	Throughput (pmph)
Boeing 747	6.5 hours	610	470	286,700
Concorde	3 hours	1350	132	178,200

Selecting performance metrics

- Flight time of Concorde vs. Boeing 747?
  - Concorde: 1350 mph / 610 mph = 2.2 times faster
    = 6.5 hours / 3 hours

- Concorde is 2.2 times («120%») faster in terms of flight time.

Aircraft	DC to Paris	Speed (mph)	Passengers	Throughput (pmph)
Boeing 747	6.5 hours	610	470	286,700
Concorde	3 hours	1350	132	178,200

Selecting performance metrics

- Flight time of Concorde vs. Boeing 747:
  - Concorde: 1350 mph / 610 mph = 2.2 times faster
    = 6.5 hours / 3 hours
  - Concorde is 2.2 times (120%) faster in terms of flight time.

- Throughput = profit per passenger = speed per passenger (pmph)
  - Boeing 747 = 286,700 pmph
  - Concorde = 178,200 pmph
    - Boeing 747 procudes 286,700 pmph / 178,200 pmph = 1.6 times (60%) more profit in terms of throughput.

Selecting performance metrics

- Global metrics: Reflect the systemwide utility
  - Resource utilization, reliability, availability.

- Individual metrics: Reflect the utility of each single user
  - Response time, throughput.

- There are cases when the decision that optimizes individual metrics is different from the one that optimizes the system metric.

Selecting performance metrics

- E.g.: Total vs. per node throughput
  - Keep the system throughput constant while increasing the number of packets from one source may lead to increasing its throughput, but it may also decrease someone's else throughput.

  - Using only the system throughput or the individual throughput may lead to unfair situations.

Selecting performance metrics

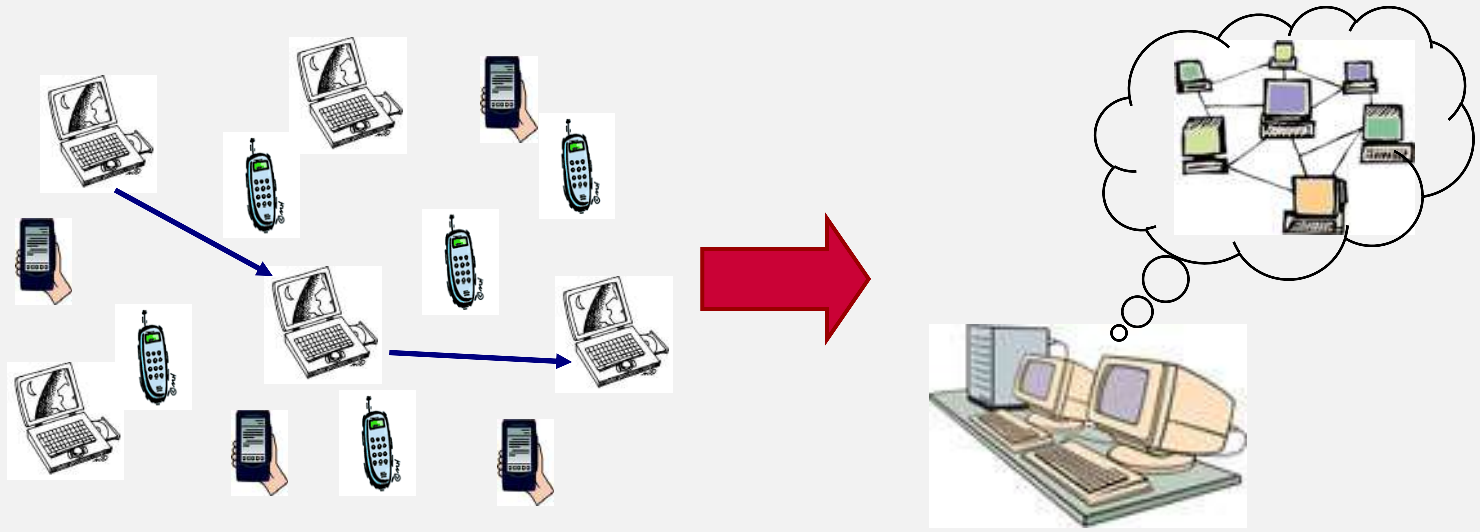
1. Low variability: Reduce the number of repetitions required to obtain a given level of statistical confidence.

2. Nonredudancy: Similar metrics should be avoided.

3. Completeness: All possible outcomes should be reflected in the set of performance metrics.

Introduction to Simulation

Introduction to Simulation



- What is a network simulator?
  - A software for modeling network applications and protocols (wired and wireless).
    - What is it used for?
      - Rebuilding a system that evolves like the real system according to certain aspects, based on a model.

Simulation: When to use it

- Study and experimentation of the internal interactions of a complex system.
- System performance evaluation before the prototype.
- Verify analytical solutions.
- Common approach in research:
    - Design of new protocols
    - Comparison of protocols
    - Traffic analysis

Simulation: Why to use it

- Only one workstation is enough to run simulations.
- Allows the study of a wide range of scenarios in a relatively short time.
- Allows realization of complex and expensive networks to be implemented in a real test-bed.
- Easy to test/check the impact of changes in a simulated solution.

Simulation: Pros & Cons

- Pros
  - System verification before the production of a prototype
  - Easy debugging of the simulated protocol
  - Possibility to analyze the system’s scalability
  - Identification of system vulnerabilities
  - Flexibility on studying the behavior of the system.

- Cons
  - The design/implementation of a model and its validation require the understanding of the simulation tool.
  - It is not always possible to capture the various aspects of the simulated system.

Simulation: Terminology

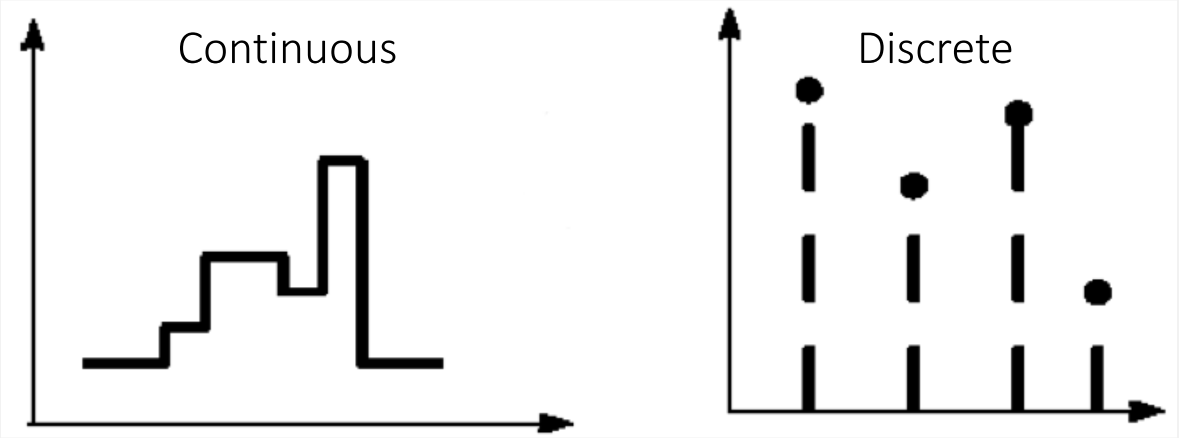
- State variables:
  - The variables whose values define the state of the system
  - Network simulation: number of nodes, packet queue, mac and routing protocols used etc..

- Event:
  - A change in the system state.
    - Network simulation: packet transmission, packet reception etc..

Introduction to Simulation

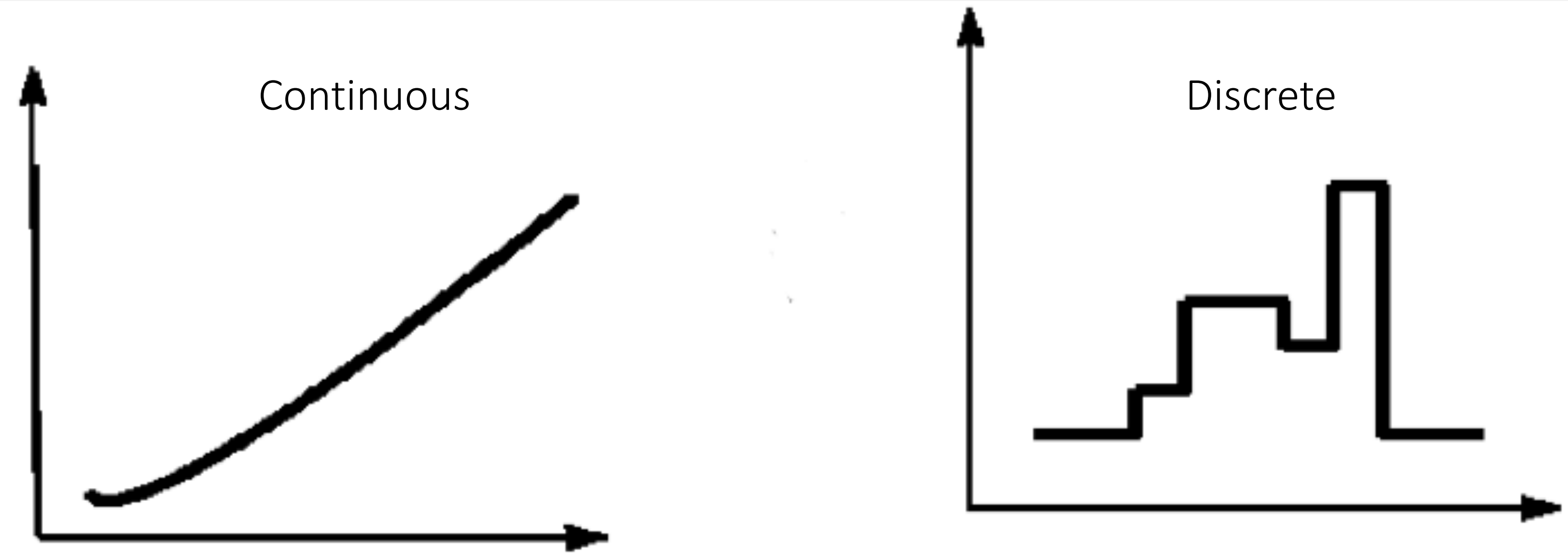
Simulation: Terminology

- Continuous-Time and Discrete-Time models:
  - *Continuous time model:* A model in which the system state is defined at all times.
    - Network simulation: number of nodes, communication among nodes is defined at any time.
  - *Discrete-Time model:* The system state is defined only at particular instants in time.
    - Classes: weekly



Simulation: Terminology

- Continuous-State and Discrete-State models:
  - *Continuous:* State variables are continuous.
  - *Discrete:* State variables are discrete.
    - Network simulation: number of nodes, packet queue length.



Simulation: Terminology

- Continuous-state models = Continuous-event models
- Discrete-state models = Disctete-event models

- Continuity of time does not imply continuity of state and vice versa!

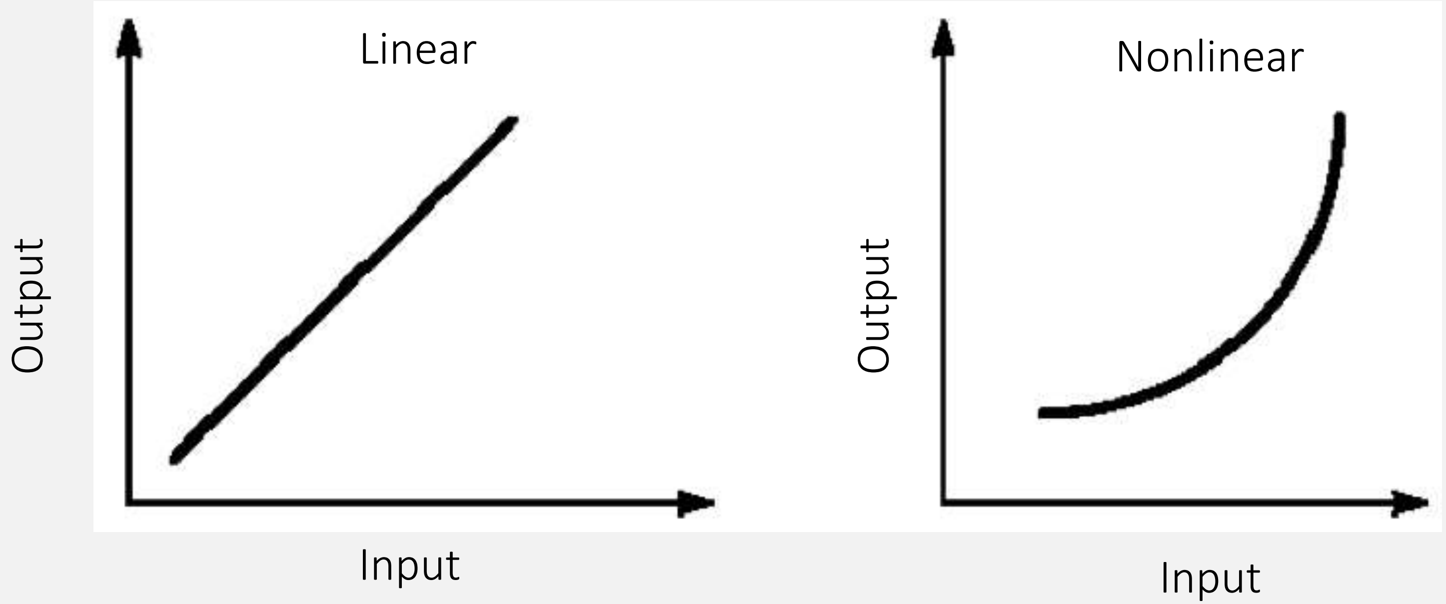
- Four possible combinations:
  1. Continuous state/continuous time
  2. Discrete state/discrete time
  3. Continuous state/discrtete time
  4. Discrete state/continuous time

Simulation: Terminology

- Deterministic and Probabilistic models:
  - *Deterministic:* The output (results) of a model can be predicted with certainty.
  - *Probabilistic:* For the same set on input parameters, each repetition gives a different output.
- Static and Dynamic models:
  - *Static:* Time is not a variable.
  - *Dynamic:* The system state changes with time.
- Open and Closed models:
  - *Open*: The input is external to the model and is independent of it.
  - *Closed:* No external input.

Simulation: Terminology

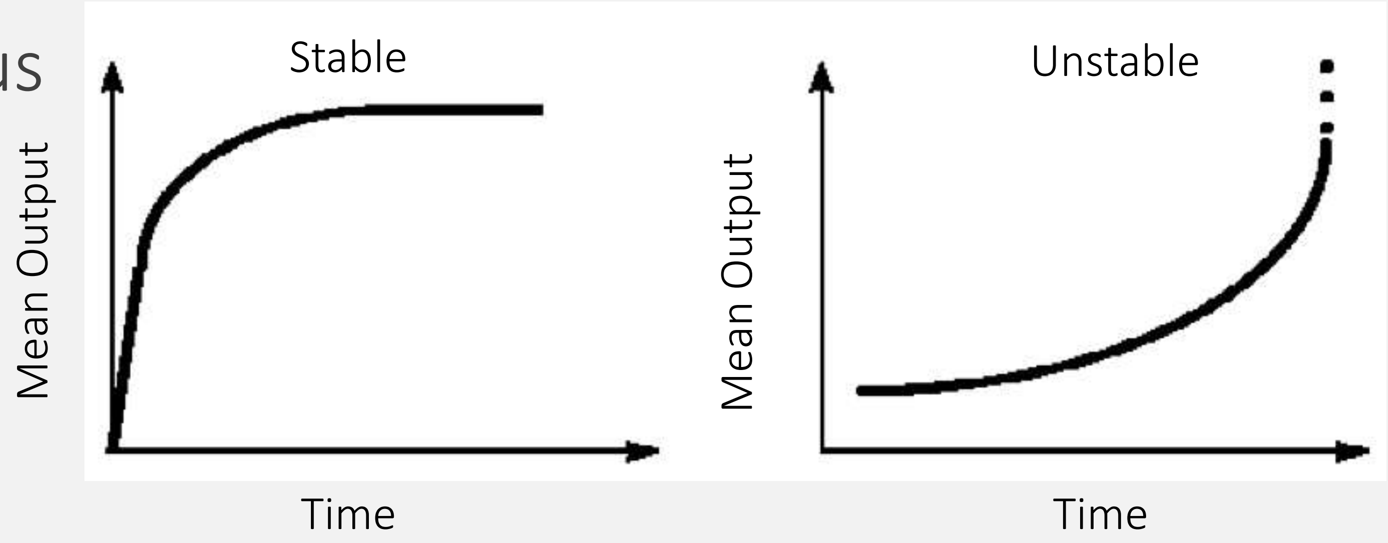
- Linear and Nonlinear models:
  - *Linear:* The output parameters are a linear function of the input parameter.
  - *Nonlinear:* Otherwise.



Computer system models: tempo continuo, stato discreto, probabilistico, dinamico e non lineare. Aperti o chiusi, stabili o instabili.

Simulation: Terminology

- Stable and Unstable models:
  - *Stable:* The dynamic behavior of the model settles down to a steady state.
  - *Unstable:* The bevavior of the model is continuous changing.



Computer system models: tempo continuo, stato discreto, probabilistico, dinamico e non lineare. Aperti o chiusi, stabili o instabili.

Simulation: Types

- Model Carlo method:
  - Static simulation without a time axis.
  - Model probabilistic phenomena that do not change characteristics with time.
- Trace-driven:
  - The simulation uses a trace as its input (a time-ordered record of events on a real system.)
- Discrete-event:
  - Discrete-state model of system.
    - Network simulation: number of packets in the queue.
    - Discrete- or continuous-time values.

Discrete-event Simulation

- Components:
  1. *Event scheduler*: It keeps a linked list of events waiting to happen.
  2. *Simulation clock*: Each simulation has a global variable representing simulated time.
     - The scheduler is responsible for advancing this time.
       - ***Unit time:*** Increments time by small increment and then checks to see if there are any events that can occur.
       - ***Event-driven:*** Increments the time automatically to the time of the next earliest occurring time.

Discrete-event Simulation

- Components:
  3. *Event routine:* Each event is simulated by its routine.
  4. *Input routines:* Get the model parameters.
  5. *Initialization routines:* Set the initial state of the system.
  6. *Trace routines:* Print out intermediate variables as the simulation proceeds; Useful on debugging.
  7. *Report generator:* Output routines executed at the end of the simulation; Calculate the final result.
  8. *Main program:* It brings all the routines together.

Common mistakes

1. Inappropriate level of detail
   - More details => Longer simulations => More bugs => More computations => More parameters ≠ Higher accuracy
2. Inappropriate experimental design
   - Too much generic => longer simulations and less accurate
3. Unverified models
   - Bugs in the code
4. Invalid models
   - Non realistic results

Common mistakes

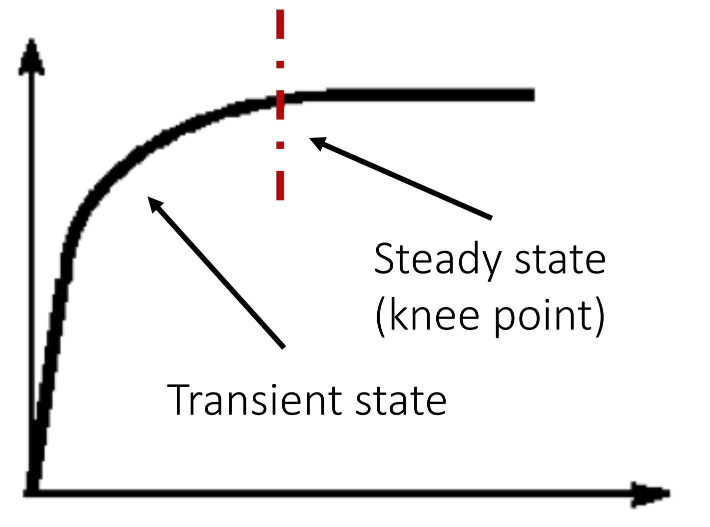
5. Improperly handled initial conditions
    - Generally not representative of the system behavior in a steady state.
6. Too short simulations
7. Poor random-number generators
8. Improper selection of seeds
    - The seed for different random-number streams should be carefully chosen to maintain independence among the streams.

Model Verification and Validation

1. **Antibugging:** Include additional checks and output in the program that will point out the bugs (if any).
    - E.g. 1: Check if the probabilities for certain events add up to 1.
    - E.g. 2: Packets received = pkts generated – pkts lost/dropped.
2. **Structured walk-through:** Explain the code to another person or a group. (It works even when the others do not understand the model!).
3. **Run simplified cases:** Easy to analyze them.
4. **Consistenty test:** Check that the model produces similar results for input parameter values that have similar effects.
5. **Degeneracy test**: Check that the model works for extreme values of system configuration or workload parameters.

Simulation Results Analysis

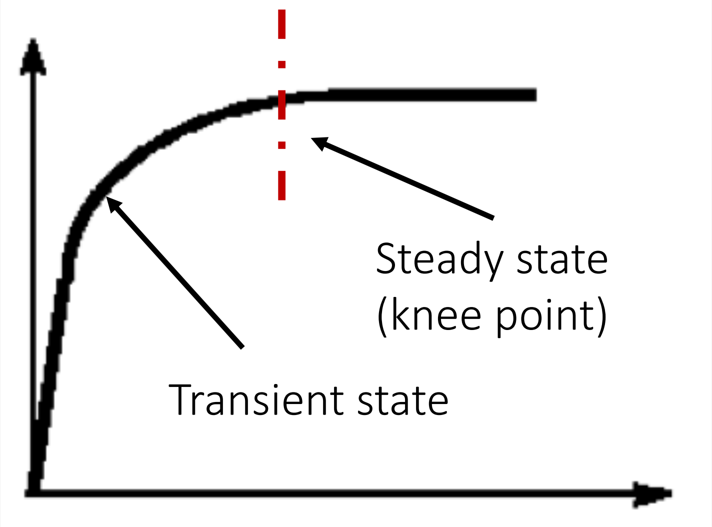
- In most simulations, only the **steady-state** performance is of interest!
- Results of the initial part of the simulation should not be included in the final computations.



- **Transient removal:** Identify the end of the transient state.
  - It is not possible to define exactly what consistutes the transient state and when the transient state ends.
    - All methods for transient removal are heuristic.

Simulation Results Analysis

- Six methods for transient removal:
  1. Long runs
  2. Proper initialization
  3. Truncation
  4. Initial data deletion
  5. Moving average of independent replications
  6. Batch means



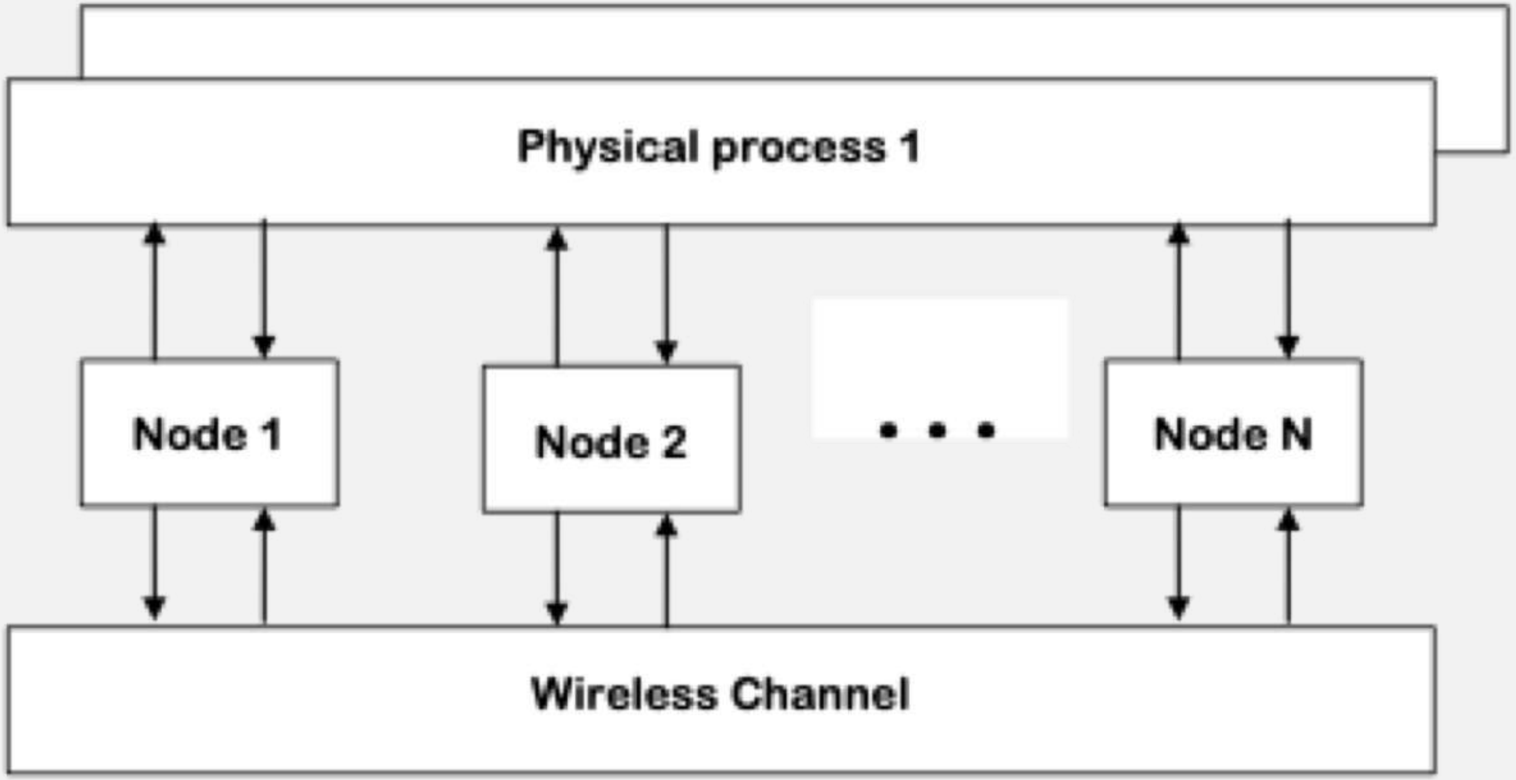
Terminating Simulations

- Short simulations => low degree confidence
- Long simulations => waste of resources
- There are systems that never reach a steady-state performance.
  - These systems always operate under transient conditions.
  - Such simulations are called **terminating simulations**; they do not require transient removal.
  - E.g.: A system shuts down at 5pm every day.

- **To increase data confidence take the average over several independent repetitions.**

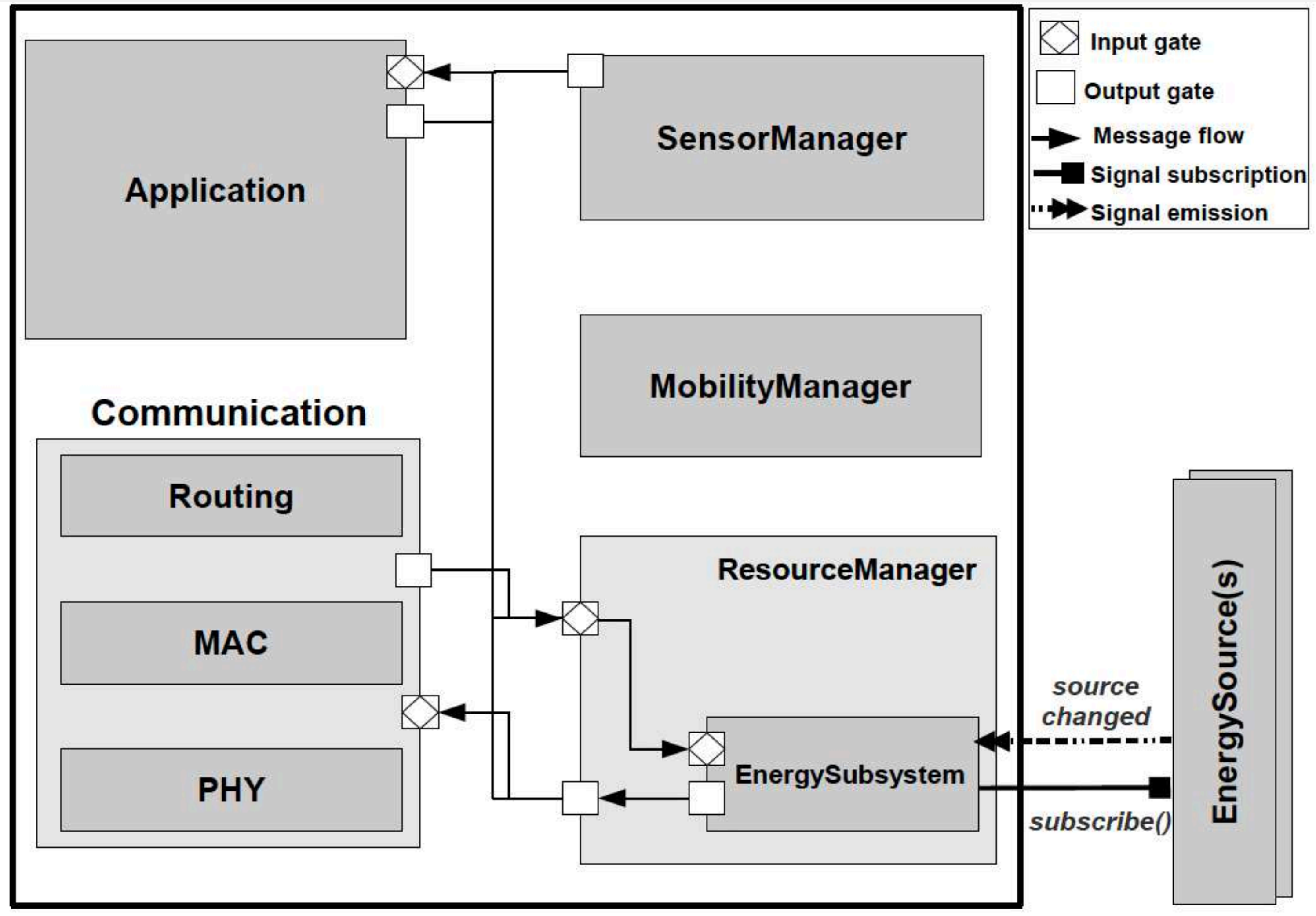
GreenCastalia: An energy harvesting-enabled simulator for IoT (Cont.)

GreenCastalia Structure



GreenCastalia

SensorNode module



GreenCastalia

Modeling in GC

- Different aspects of a wireless sensor network from communications to physical processes.
- Single common point for all non-composite modules:
  - A parameter called `collectTraceInfo`
  - Is set by default to false; If set to true, then the module will produce trace information
  - Traces will be written in the `Castalia-Trace.txt` file (in the directory of the simulation scenario folder)

The wireless channel

- GC implements a realistic wireless channel for IoT systems, such as WSNs and BANs.

- Average path loss modeling
  - Lognormal shadowing model has been shown to give accurate estimates.

- Path loss in dB as a function of the distance between two nodes

$$PL(d) = PL(d_0) + 10 \cdot \eta \cdot \log\left(\frac{d}{d_0}\right) + X_\sigma$$

The wireless channel

- Directory:
  `~/Castalia/src/wirelessChannel/defaultChannel/`
- File `WirelessChannel.ned`:
  - Defines parameters related to the wireless channel
  - `pathLossExponent, PLd0, d0, sigma`

- To access these parameters in the .ini file you have to prefix their name with “`SN.wirelessChannel.`”

GreenCastalia

Node mobility

- Taking the path loss between two nodes is not enough.
- Keep state about the path losses between points in the space.
- Space is broken in discrete cells and calculate the path losses from each cell to each other cell.
- Use cell locations and cell IDs instead of specific node locations and node IDs.

Node mobility

- Parameters that set the cell size (default=5m)
- `SN.wirelessChannel.xCellSize`
- `SN.wirelessChannel.yCellSize`
- `SN.wirelessChannel.zCellSize`

- No mobility
- `SN.wirelessChannel.onlyStaticNodes`

The radio module

- Captures many features for real low-power radios.
- Main features include:
  - Multiple states: Transmit, receive, listen, configurable sleep states.
  - Transition delays from one state to another.
  - Different power consumption for the different states and Tx levels used.
  - …
  - GC includes 3 already defined radios.

The radio module

- Directory: `~/Simulations/Parameters/Radio/`
- Available radios
  - BANRadio.txt: defines the narrowband radio proposed in the IEEE 802.15 Task Group 6 documents
  - CC1000.txt: defines the CC1000 real radio by Texas Instruments
  - CC2420.txt: defines the CC2420 real radio by Texas Instruments

- In the `omnetpp.ini` file the parameter `RadioParametersFile` of the radio module points to one of these files.

The radio module

- Directory
  `~/Simulations/Parameters/Radio/`
- E.g.: `CC2420.txt`
- Radio parameters file
  - RX MODES
  - TX LEVELS
  - DELAY TRANSITION MATRIX
  - POWER TRANSITION MATRIX
  - SLEEP LEVELS

The radio module

- A set of parameters can be specified in the configuration file
- Select the starting RX mode; default empty value means that the first mode listed will be used: `string mode = default ("")`
- Select the starting state once simulation begins; default value is set to listening (receiving) state: `string state = default ("RX")`
- Starting transmission output power level; only levels declared in the radio parameter file can be used; default empty value means that the highest value will be used:
  `string TxOutputPower = default ("")`

GreenCastalia

Creating modules

- Determine the correct location for the new code.

- Create a dedicated directory for the source code of the new module.

- Possible locations:
  - Application: ~/Castalia/src/node/application
  - Routing: ~/Castalia/src/node/communication/routing
  - MAC~/Castalia/src/node/communication/mac
  - Mobility: ~/Castalia/src/node/communication/mobilityManager

Creating modules

- Define the module using the NED language; the `.ned` file is named by the name of the module:
  - E.g.: new module: `newCastaliaModule`

- Then the name of the corresponding `.ned` file will be `NewCastaliaModule.ned`

- The dedicated directory starts with a lower case letter, while the name of the `.ned` file starts with an upper case letter.

GreenCastalia

Creating modules

- In the `.ned` file define the following:
  - The package of the module
  - Obtain the package by taking the current directory path to the Castalia's `src/` directory and by replacing each "/" symbol with "."

- Include all the parameters to be passed to the module at runtime from the simulation configuration; Some parameters are mandatory for all modules.

Creating modules: An example


```
package node.communication.mac.newCastaliaModule;
simple NewCastaliaModule like node.communication.mac.iMac {
parameters:
bool collectTraceInfo = default(false);
int macMaxPacketSize = default(0);
int macBufferSize = default(16);
int macPacketOverhead = default(8);
int newParameter1;
string newParameter2 = default("default value");
bool newParameter3 = default(false);
gates:
output toNetworkModule;
output toRadioModule;
input fromNetworkModule;
input fromRadioModule;
input fromCommModuleResourceMgr;
}
```

Creating modules

- The next step is to include (and write in C++) the actual code of the module.
- The new module has to inherit some "properties" from appropriate base classes that are provided (Virtual classes).
- A .h file and a .cc file have to be created.
- In the source code file (.cc)
  - Include the .h file
  - Register the new creating as an OMNeT module
    - `Define_Module(NewCastaliaModule);`
  - Define the methods that the virtual class implements.

GreenCastalia

Creating modules

- Application:

```
class NewCastaliaModule : public VirtualApplication{
```

- Routing:

```
class NewCastaliaModule : public VirtualRouting{
```

- MAC:

```
class NewCastaliaModule : public VirtualMac{
```

- MobilityManager:

```
class NewCastaliaModule : public VirtualMobilityManager{
```

Defining an application packet

- Default application packets have only one field (double) to carry data.
- Create a new .msg file in the new application directory

```
cplusplus {{
#include "ApplicationPacket_m.h"
}}
class ApplicationPacket;
struct info {
unsigned short nodeID; //the ID of the Node
double locX; // x-coordinate of the node
double locY; // y-coordinate of the node
}
packet MyPacket extends ApplicationPacket {
info extraData;
}
```

Defining a new Routing module

- Directory: `~/src/node/communication/routing`

- The VirtualRouting class defines a set of methods
  - Callback methods: Allow the specific routing protocol to react to certain events.
  - Pre-defined methods that perform generic operations.

GreenCastalia

Defining a new Routing module

Callbacks:

- `void startup()`
- `void finishSpecific()`
- `void fromApplication Layer(cPacket *pkt, const char*dstAddr)`
- `void fromMacLayer(cPacket *pkt, int srcMacAddress, double RSSI, double LQI)`
- `void handleNetworkControlCommand(cMessage *)`
- `void handleMacControlMessage(cMessage *)`
- `void handleRadioControlMessage(cMessage *)`

GreenCastalia

Defining a new Routing module

Methods:

```
• void encapsulatePacket(cPacket *, cPacket *)
• int bufferPacket(cPacket *pkt)
• cPacket *decapsulatePacket(cPacket *)
• void toApplicationLayer(cMessage *msg)
• void toMacLayer(cMessage *msg)
• void toMacLayer(cPacket *pkt, int macAddr)
• int resolveNetworkAddress(const char *)
• bool is NotDuplicatePacket(cPacket *pkt)
```

Additional Resources

- R. Jain, "*The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*“, Wiley-Interscience, New York, NY, April 1991. (Chapters 2, 3, 24)

Questions?