System Performance Evaluation

Georgia Koutsandria

Internet of Things A.Y. 19-20

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 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	Post-prototype
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, results 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

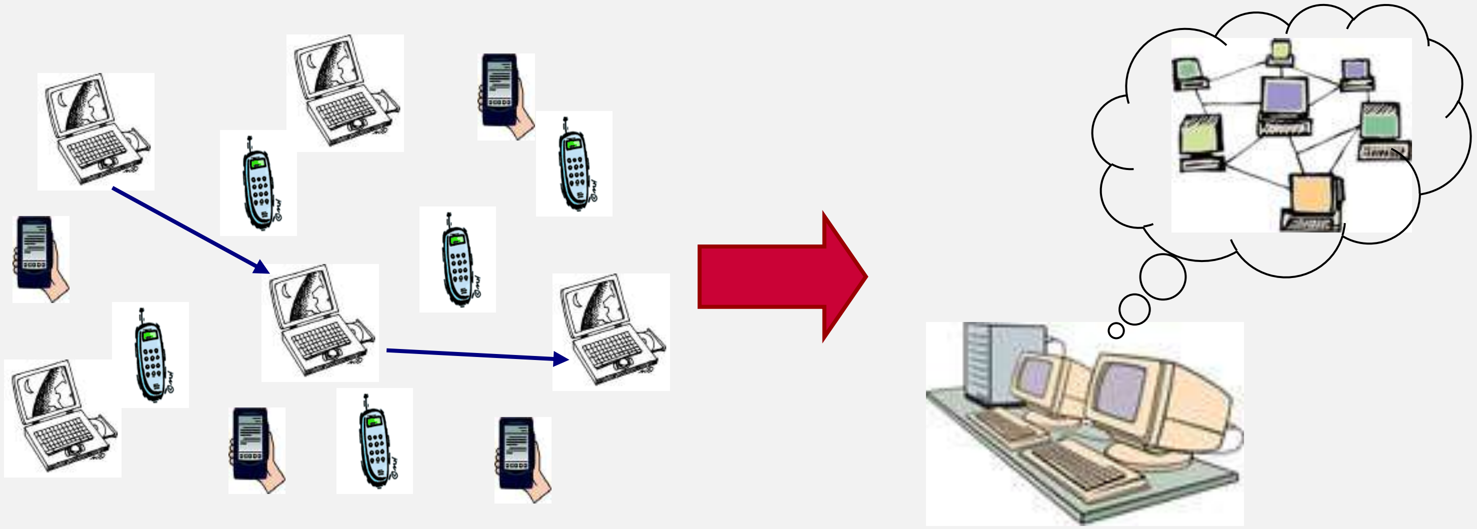
1. Low variability: Reduce the number of repetitions required to obtain a given level of statistical confidence.

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.

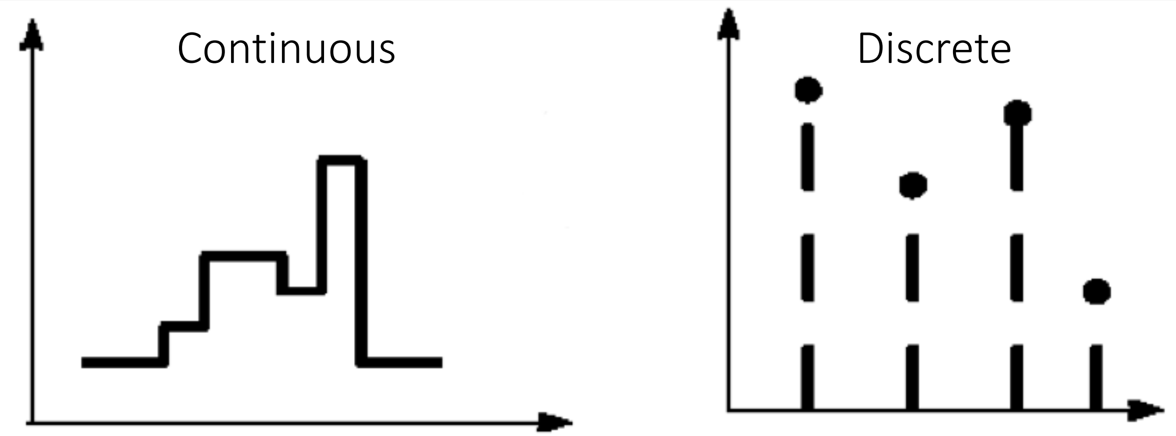
Simulation: Terminology

- 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..

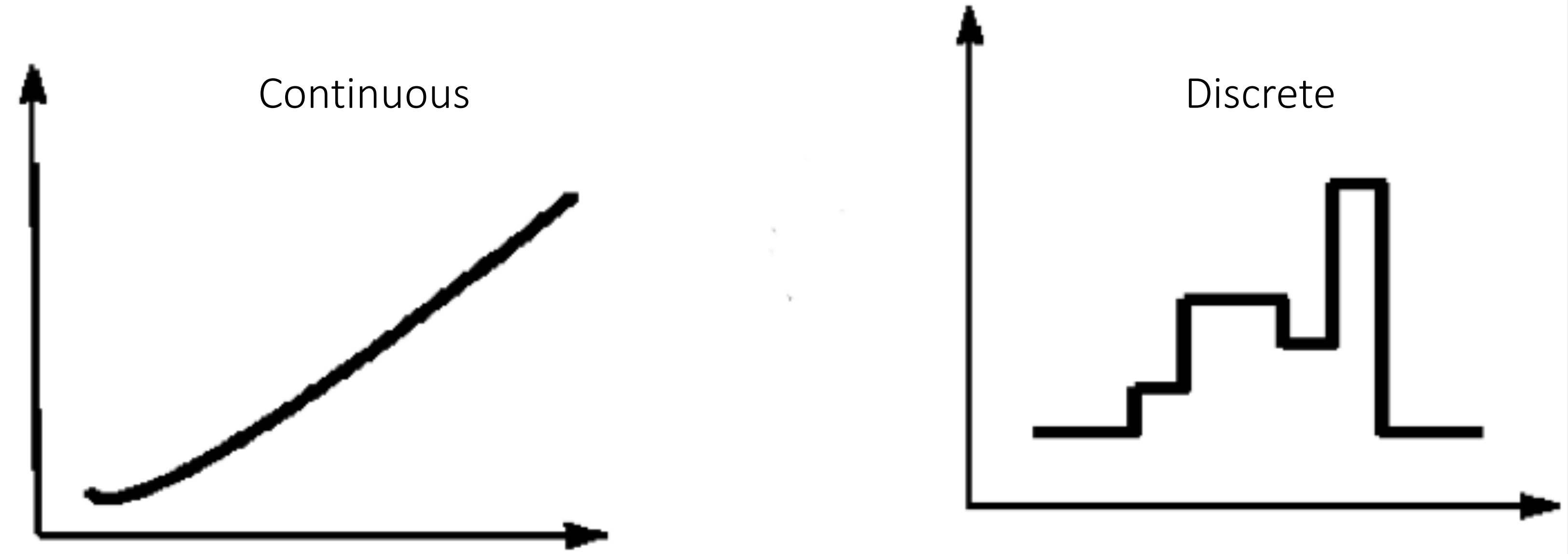
Simulation: Terminology

- 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!

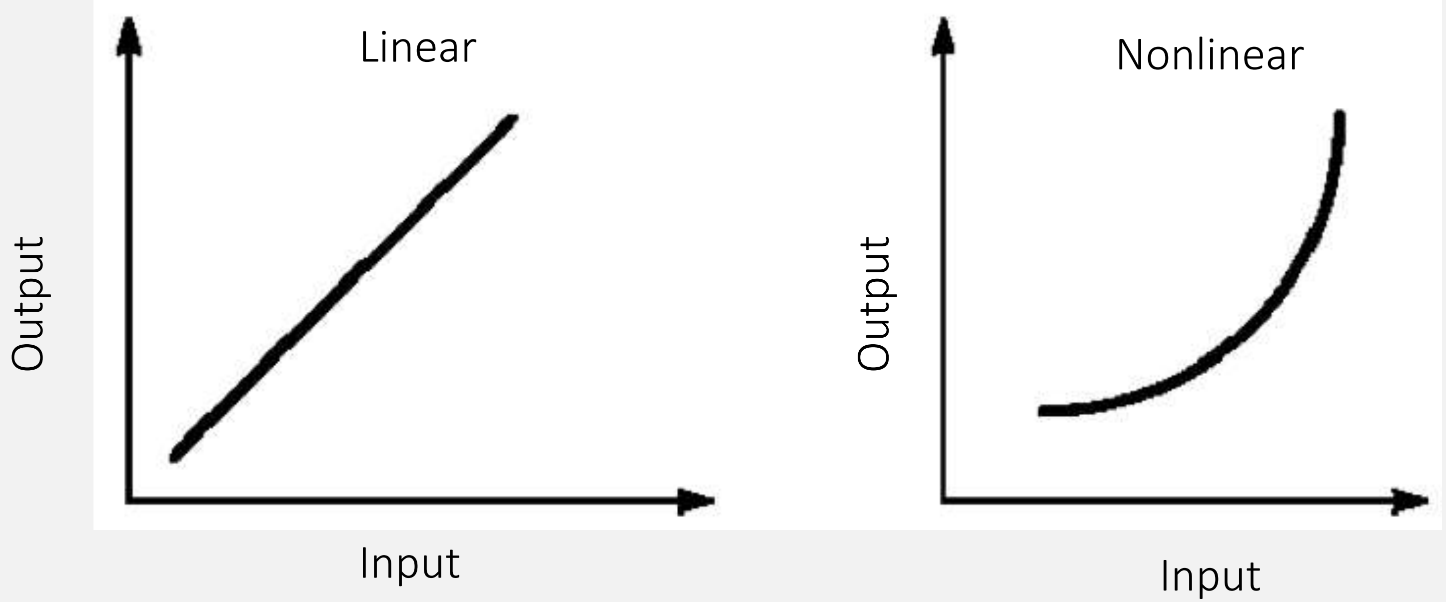
- Four possible combinations:
  1. Continuous state/continuous time
  2. Discrete state/discrete time
  3. Continuous state/discrtete time
  4. Discrete state/continuous time

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

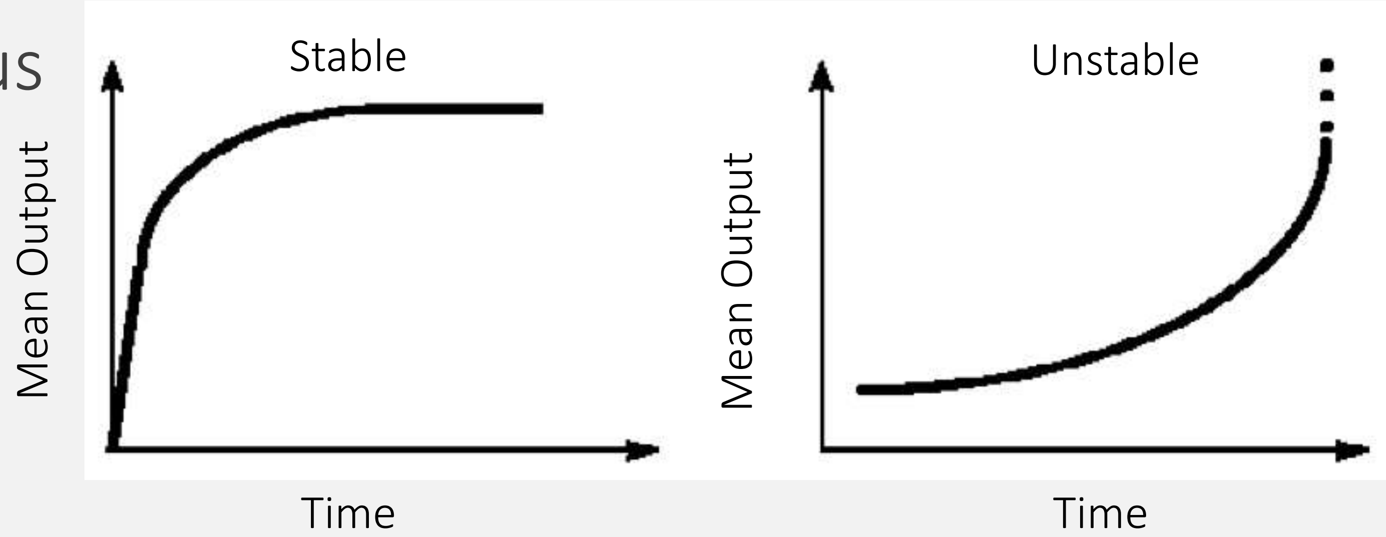
- Linear and Nonlinear models:
  - *Linear:* The output parameters are a linear function of the input parameter.
  - *Nonlinear:* Otherwise.



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

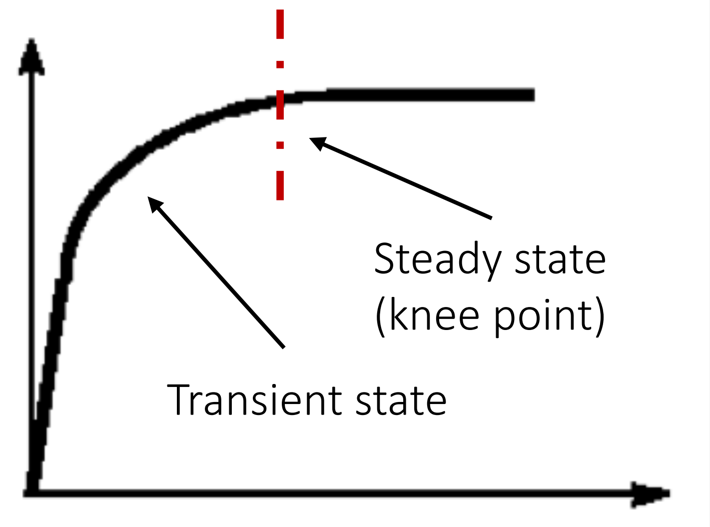
5. Improperly handled initial conditions
    - Generally not representative of the system behavior in a steady state.
6. Too short simulations
7. Poor random-number generators
8. Improper selection of seeds
    - The seed for different random-number streams should be carefully chosen to maintain independence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

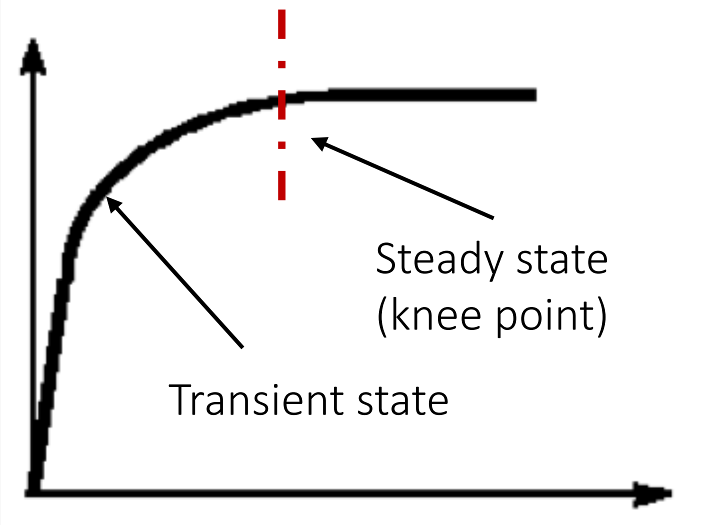
- In most simulations, only the **steady-state** performance is of interest!
- Results of the initial part of the simulation should not be included in the final computations.



- **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.**

Simulators for IoT Systems

What is a Simulator?

- A tool/software that realistically imitates/models the behavior of IoT systems.

- Different types of simulators; Most commonly used:
  - Trace-Driven Simulators
  - Discrete-Event Simulators

Why do we use Simulators?

- The most common approach to delelop and test new protocols/applications.
- Evulate the performance of new solutions.
- Consider a large-scale IoT network:
  - Low cost
  - Easy(?) to implement
  - Practical

Simulators for IoT Systems

- Several simulators exist:
  - ns-3/ns-2
  - OMNeT
  - Castalia
  - GreenCastalia
  - SUNSET
  - COOJA
  - Avrora
  - …

GreenCastalia: An energy harvesting-enabled simulator for IoT

What is GreenCastalia?

- An extension of the Castalia simulator.
- Allows to model and simulate networks of IoT devices, i.e., embedded devices, with energy harvesting capabilities.
- Castalia: Am OMNeT++ based simulator for WSNs, BANs, and networks of low-power embedded devices.
  - A realistic framework for fisrt order validation.
  - Not platform(device) specific.
  - Highly parametric.

How to install GC

- You will first need to install OMNET++
  - OMNET++ (recommended version 4.6): https://omnetpp.org
  - Castalia: https://github.com/boulis/Castalia

- Complete instructions:
  - http://senseslab.di.uniroma1.it/greencastaliav01d

How to install GC

- We strongly recommend that you use a Unix-based machine.

- Alternative Option: Download the VM (available link on twiki) with the GC simulator already installed on it (pwd: iot2018)
  - You will first need to install the VirtualBox software
    - https://www.virtualbox.org/wiki/Downloads

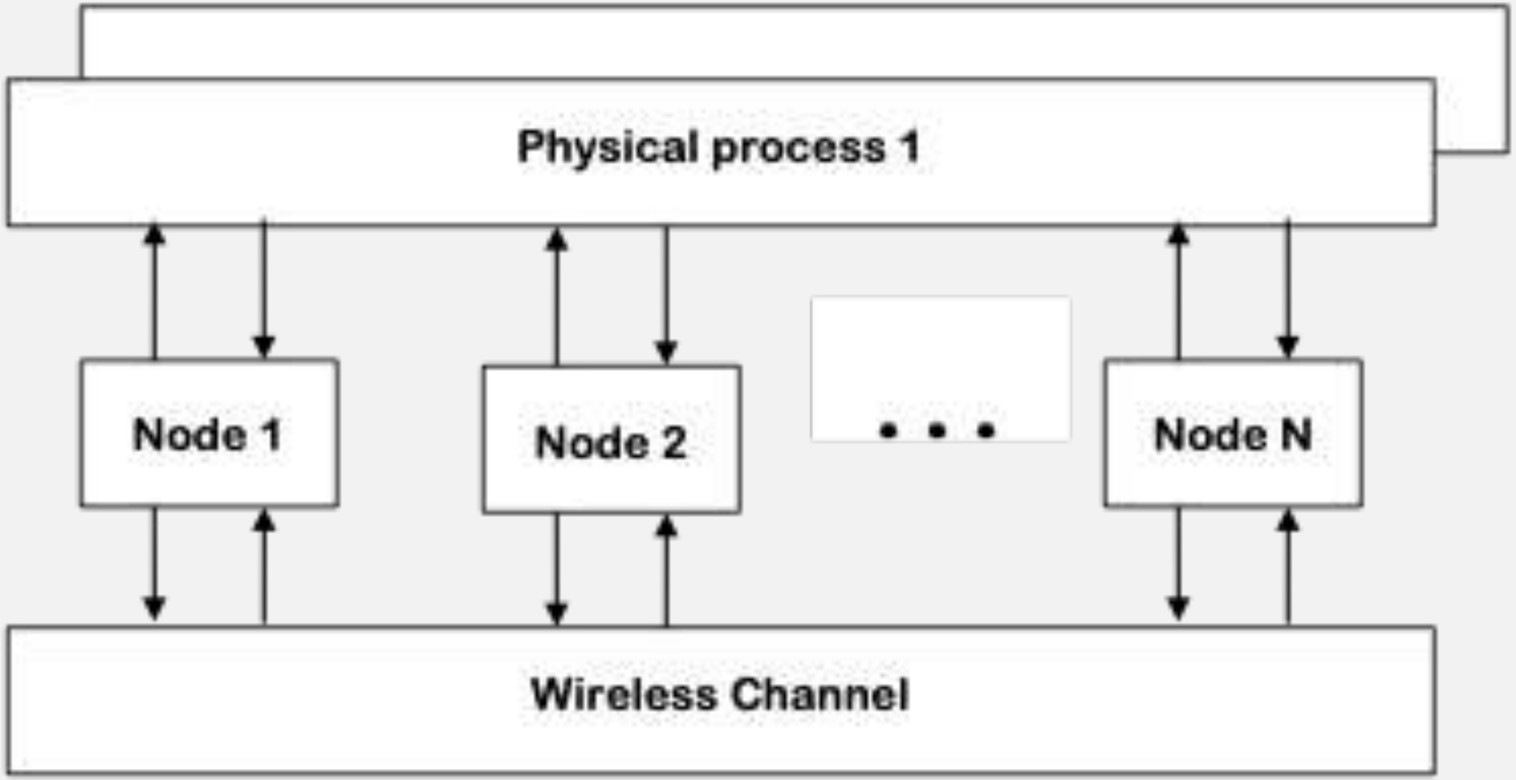
GreenCastalia: Main features

- Inherited by the Castalia simulator:
  - Channel model based on empirically measured data.
  - Radio model based on real traces for low-power communication.
  - Sensing modelling provisions.
  - MAC and routing protocols available.
  - Designed for adaption and expansion.

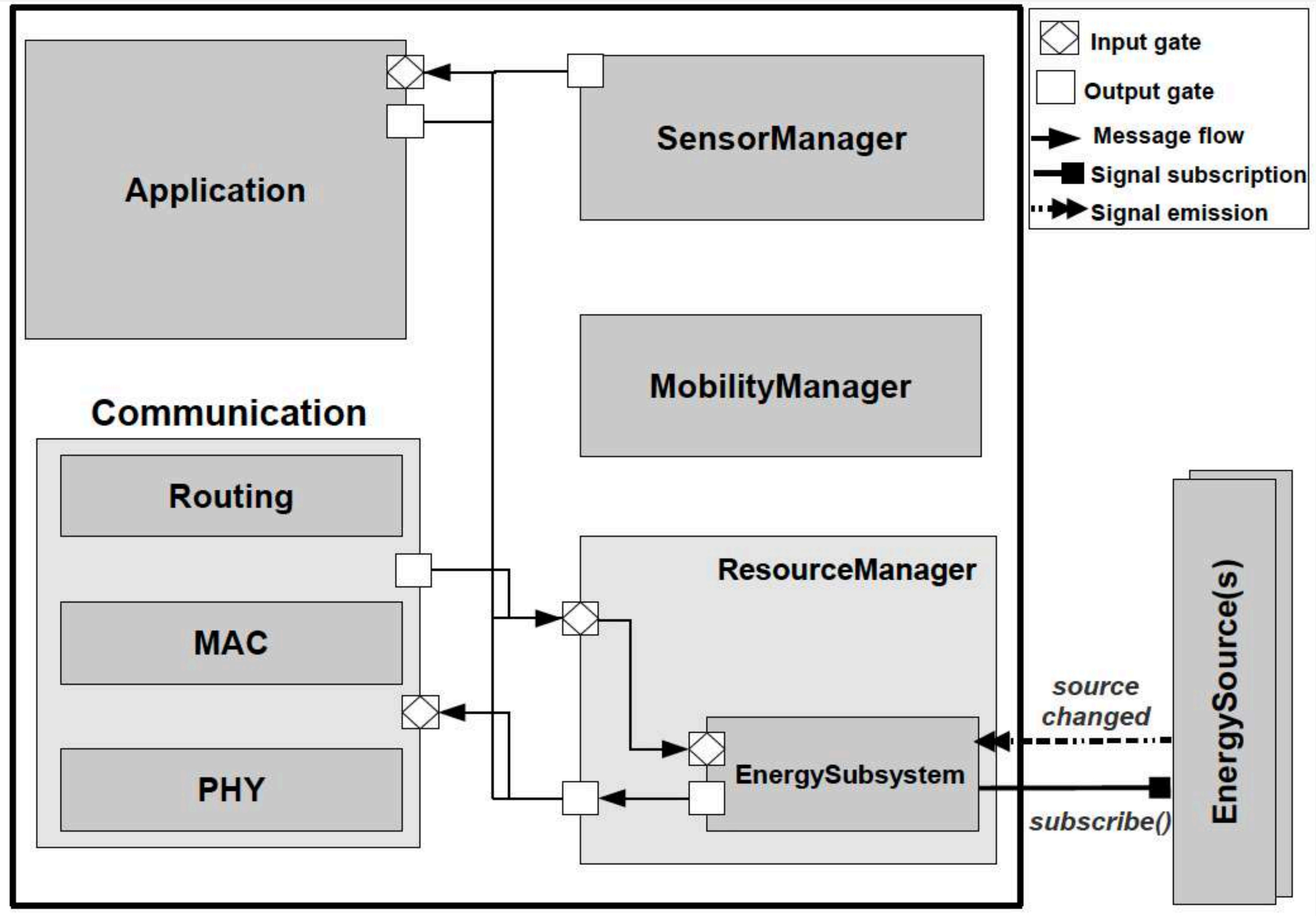
GreenCastalia: Main features

- GC-specific:
  - Multiple energy sources and multi-source harvesters.
  - Networks of embedded devices with heterogeneous harvesting and storage capabilities.
  - Multi-storage architectures (batteries, supercaps, rechargeable batteries).
  - Non-ideal battery models based on empirical discharge patterns, and supercaps leakage models.
  - Energy prediction models.

GreenCastalia Structure



SensorNode module



GreenCastalia

GC Organization

- Each module or submodule has its corrsponding directory.
- All reside in the directory `~/Castalia/src/`
- E.g.: Module node resides in the directory:
  `~/Castalia/src/node/`
  Module communication resides in the directory:
  `~/Castalia/src/node/communication/`
  Submodule routing resides in the directory:
  `~/Castalia/src/node/communication/routing`

GreenCastalia

GC Organization

- In the GC directory there is a folder named *Simulations*
  `~/Castalia/Simulations/`
- This folder includes:
  - Existing simulation examples with their simulation configuration files.
  - A subfolder named *Parameters*
    - Includes specially fromatted files with parameters that define the basic operational properties of specific modules (MAC, Radio, WirelessChannel, SensorDevice, PhysicalProcess).

Building GreenCastalia

- (Re)Build GC by using the following commands at the top-most GC directory `~/Castalia/`

```
make clean
./makemake
make
```

- After the creation of new files or any modifications in existing ones, rebuild GC using the same commands.

Using GreenCastalia

- Files with the suffice «.ned» contain NED language code
  - Define a module’s name and interfaces (gates in/out)
  - Define parameters
- Module directories always contain a «.ned» file defining them
- Simple modules include C++ code (.cc and .h files) defining their behavior
- Composite modules, e.g., node, include subdirectories to define the submodules.

Simulation Configuration File

- All simulation examples/tests reside in the directory `~/Castalia/Simulations`
- Configuration file typically named `omnetpp.ini`
  - Assigns values to parameters; Defines the simulation scenario.
  - The following file should be always included in the configuration file
    - `include ../Parameters/Castalia.ini`
    - It containes basic parameter assignment.
  - Defines the simulation time
  - Parameters always start with `SN` (sensor network: the top-most composite module)

GreenCastalia

Simulation Configuration File


```
[General]

include ../Parameters/Castalia.ini

sim-time-limit = 100s

SN.field_x = 200 #meters
SN.field_y = 200 #meters
```

Simulation Configuration File

- Defining the area of deployment using the parameter
  `SN.deployment`

- Several options:
  - uniform: random uniform distribution
  - NxM: nodes are placed in a NxM grid area
  - NxMxK: 3D dimension; nodes are placed in a NxMxK grid area
  - randomized_NxM: nodes are randomly places to NxM grid
  - Randomized_NxMxK: nodes are randomly places to NxMxK grid
  - center: nodes are placed in the center of the deployment area

GreenCastalia

Simulation Configuration File

- The sensor network compound module (`SN`) contains many Node sub-modules.

- Sub-modules are addressed in the form of an array.

- Assigning values to multiple nodes:
  - `[*]`: all indexes
  - `[3..5]`: indexes 3,4,5
  - `[..4]`: indexes 1, 2, 3, 4
  - `[5..]`: indexes 5 till last one

Running a simulation

- How to use the Castalia input script
  - `../../bin/Castalia -h`
- Available configurations
  - `../../bin/Castalia`
- Run a simulation using a specific configuration
  `../../bin/Castalia -c General`
- Two files created in the directory
  1. `YYMMDD-HHMMSS.txt:` Output file which includes results.
  2. `Castalia-Trace.txt:` Contains traces of all events requested.

GreenCastalia

The `CastaliaResults` script

- Directory:
  `~/Castalia/bin/CastaliaResults/`
- `CastaliaResults`
  - Full list of Castalia output files with information about the configurations and the creation date.
  - Number of repetitions is indicated in the parenthesis.
  - `CastaliaResults -i YYMMDD-HHMMSS.txt`
    - Parses the given file and finds out what output was recorded by the different modules.

GreenCastalia

The `CastaliaResults` script

- Use the -s switch to select among outputs, e.g., packets;
  Results are the average of all modules and indices.

```
../../bin/CastaliaResults -i YYMMDD-HHMMSS.txt -s packets
```

- Get the sum of all nodes

```
../../bin/CastaliaResults -i YYMMDD-HHMMSS.txt -s packets -sum
```

- Get per node results

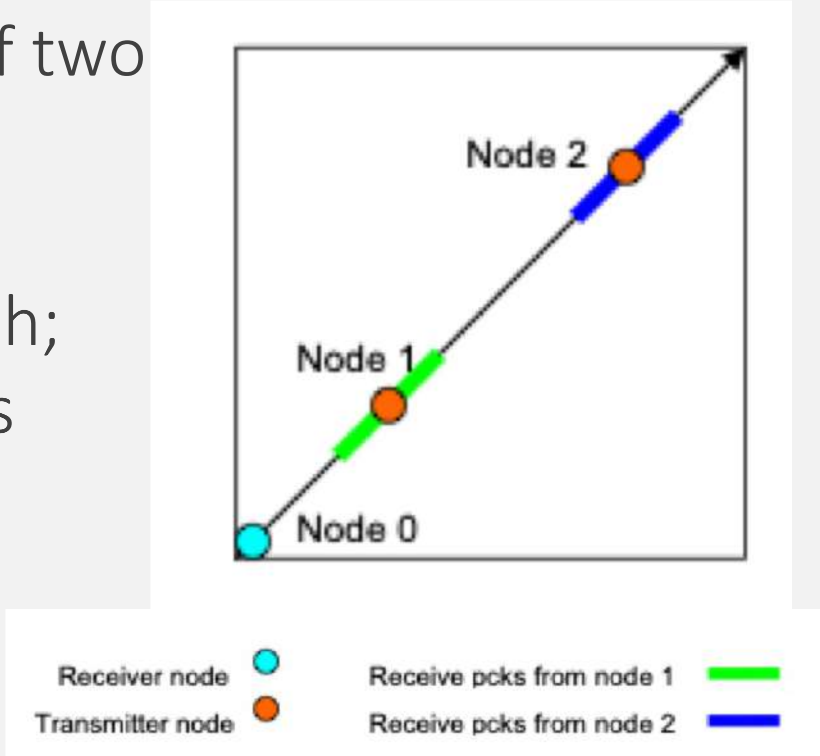
```
../../bin/CastaliaResults -i YYMMDD-HHMMSS.txt -s packets -n
```

GreenCastalia

Simulation: An Example

- Go to `~/Castalia/Simulations/radioTest`

- Scenario: General (Tests reception)
  - A receiver (node 0) moves through the area of two transmitters (nodes 1 and 2).
  - No interference between transmitters.
  - Receiver moves in a straight line back and forth;
  - The receiver should receive packets when it is close to each of the two transmitters.



GreenCastalia

Simulation: An Example

- Type the following commands:
  1. `rm 1*.txt`
  2. `rm Castalia-Trace.txt`

- Run a simulation using the default configuration

```
../../bin/Castalia -c General
```

- Two files created in the directory
  1. `YYMMDD-HHMMSS.txt:` Output file which includes results.
  2. `Castalia-Trace.txt:` Contains traces of all events requested.

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

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

GreenCastalia

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.`"

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.

GreenCastalia

Node mobility

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

- No mobility
- `SN.wirelessChannel.onlyStaticNodes`

GreenCastalia

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`

GreenCastalia

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.

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.

GreenCastalia

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.

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 *)`

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?