Network Simulator SCNSL

Emad Ebeid
PhD
Department of Computer Science
University of Verona
Italy

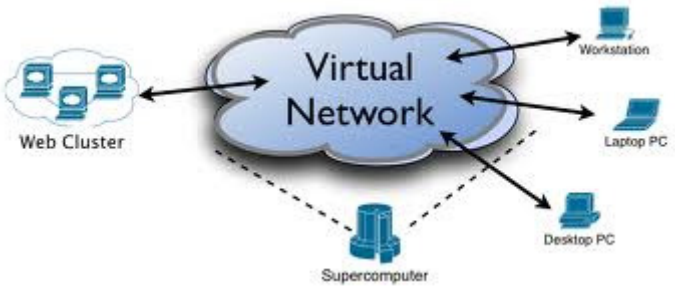
Davide Quaglia
Assistant Professor
Department of Computer Science
University of Verona
Italy

Outline

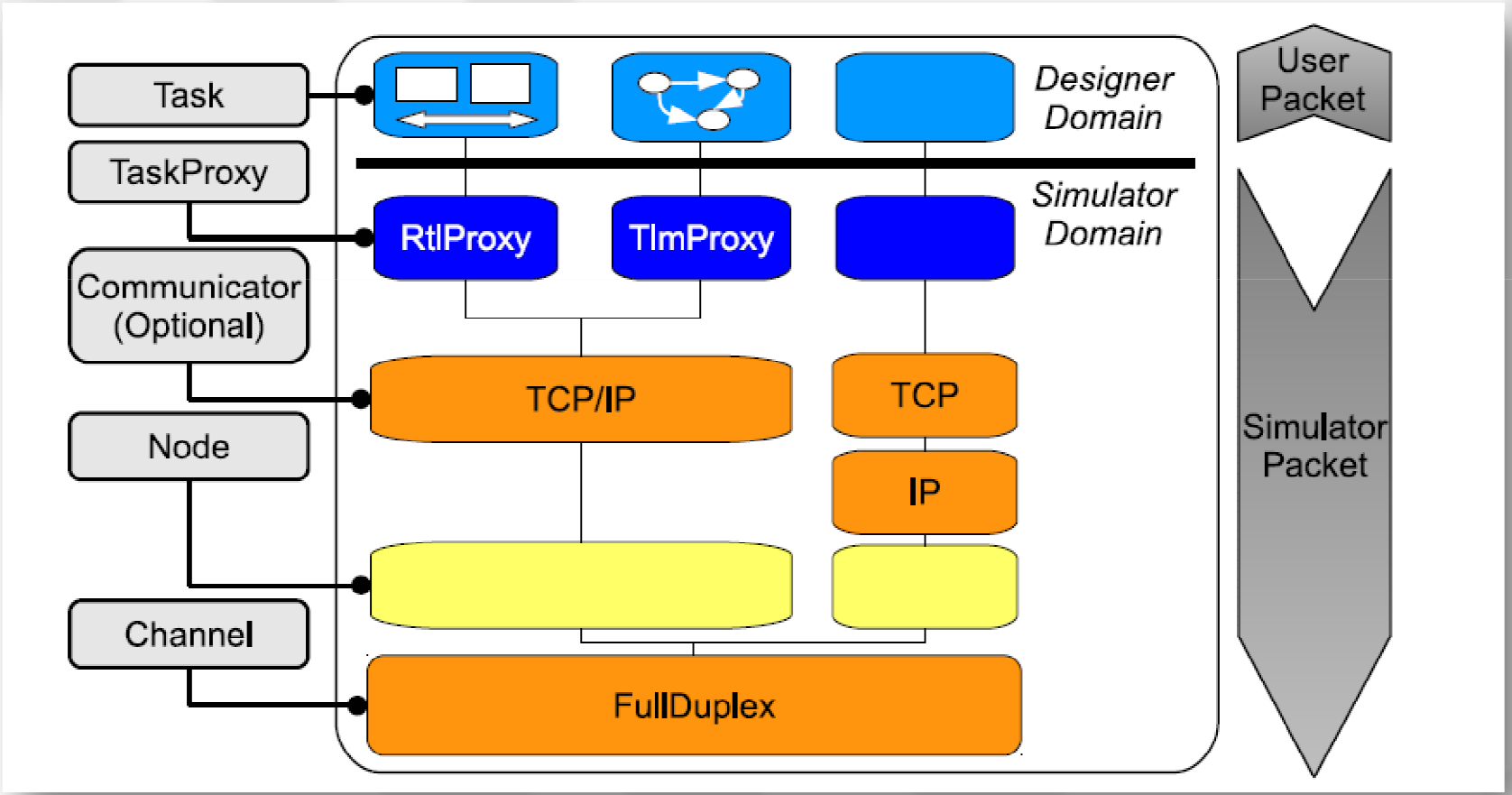
- Introduction
- How to setup SCNSL
- How it works
- Examples and Exercises

Simulation

- SystemC Network Simulation Library (SCNSL): is an extension of SystemC to allow modeling packet-based networks such as wireless networks, Ethernet, and fieldbus. As done by basic SystemC for signals on the bus, SCNSL provides primitives to model packet transmission, reception, contention on the channel and wireless path loss. The use of SCNSL together with **SystemC** allows the easy and complete modeling of distributed applications of networked embedded systems such as wireless sensor networks, routers, and distributed plant controllers



SCNSL components



SCNSL components

- ***Tasks*** are the system functionality which is under development. Thus, tasks shall be implemented by designers either at RTL or TLM level. From the point of view of a network simulator, a task is just the producer or consumer of packets and therefore its implementation is not important. However, for the system designer, task implementation is crucial and many operations are connected to its modeling, i.e., change of abstraction level, validation, fault injection, HW/SW partitioning, mapping to an available platform, synthesis, and so forth. For this reason the class TaskProxy_if_t has been introduced, which decouples task implementation from the backend which simulates the network.
- Each Task instance is connected to one or more **TaskProxy** instances and, from the perspective of the network simulation kernel, the TaskProxy instance is the alter-ego of the task. Viceversa, from the point of view of the application, each TaskProxy can represent a sort of socket interface, since it provides the primitives for network communication.

SCNSL components

Tasks are hosted on ***Nodes***, which are the abstraction of physical devices. Thus, tasks deployed on different nodes shall communicate by using the API provided by SCNSL for the network communication, while tasks deployed on the same node shall communicate by using standard SystemC communication primitives.

Setup (1)

- Prerequisite:
  - Linux operating system
  - SystemC 2.0 or newer and TLM 2.0 must installed in your system (check http://www.systemc.org)
  - CMake
  - C++ compiler and linker
- Create an empty directory in your home (e.g. work/) and enter it
- Download SCNSL from Sourceforge Bazaar repository
  - $ bzr checkout bzr://scnsl.bzr.sourceforge.net/bzrroot/scnsl/trunk
- Uncompress the provided zip archive inside work/ folder
- Modify trunk/scripts/env-setup.sh
  - Change SYSTEMC_DIR and TLM_DIR to point to your SystemC and TLM folders; on Lab Alfa you have to modify the lines as follows:
    - SYSTEMC_DIR=/usr/local/systemc-2.3.0
    - TLM_DIR=/usr/local/systemc-2.3.0

Setup (2)

Run script env-setup.sh with the CPU type (32 or 64 bits):

- $. ../scripts/. env-setup.sh 32|64
  Please note the space between “’.” and “env-setup.sh”
  On Lab Alfa you have to specify “64”.

Create obj_linux_64/ folder inside SCNSL_lesson/:

- $ mkdir obj_linux_64; cd obj_linux_64; cmake ..
- $ make –j12
- $ make doc
- $ make install

To compile original examples

- $ cd ..
- $ cd /tests
- $ mkdir obj ; cd obj; cmake ..
- $ make
- $ make install

How to use it (1)

- Nodes creation:

```
Scnsl::Core::Node_t * n0 = scnsl->createNode();
```

- Channel setup:

```
CoreChannelSetup_t ccs;
ccs.extensionId = "core";
ccs.channel_type = CoreChannelSetup_t::SHARED;
ccs.name = "SharedChannel";
ccs.alpha = 0.1;
ccs.delay = sc_core::sc_time( 1.0, sc_core::SC_US );
ccs.nodes_number = 2;
```

- Environment setup:

```
Scnsl::Utils::DefaultEnvironment_t::createInstance(ccs.
alpha);
```

How to use it (2)

- Channel creation:

```
Scnsl::Core::Channel_if_t * ch1 = scnsl->createChannel( ccs );
```

- Task ID:

```
const Scnsl::Core::task_id_t id3 = 3;
```

- Task Proxy:

```
const Scnsl::Core::size_t ONEPROXY = 1;
```

- Task Creation:

```
MyTask_t t3( "Task3", id3, n3, ONEPROXY );
```

How to use it (3)

Protocol 802.15.4 creation:

```
CoreCommunicatorSetup_t ccoms;
ccoms.extensionId = "core";
ccoms.ack_required = true;
ccoms.short_addresses = true;
ccoms.type =    CoreCommunicatorSetup_t::MAC_802_15_4;
```

MAC creation:

```
ccoms.name = "Mac0";
ccoms.node = n0;
Scnsl::Core::Communicator_if_t * mac0 =
                scnsl->createCommunicator( ccoms );
```

How to use it (4)

Bind setup

```
BindSetup_base_t bsb1;
bsb1.extensionId = "core";
bsb1.destinationNode = n2
bsb1.node_binding.x = 1;
bsb1.node_binding.y = 1;
bsb1.node_binding.z = 1;
bsb1.node_binding.bitrate =
     Scnsl::Protocols::Mac_802_15_4::BITRATE;
bsb1.node_binding.transmission_power = 1000;
bsb1.node_binding.receiving_threshold = 1;
```

How to use it (5)

```
scnsl->bind( n1, ch, bsb1 );
scnsl->bind( & t1, & t3, ch, bsb1, mac1 );
scnsl->bind( n2, ch2, bsb2 );
scnsl->bind( & t2, & t0, ch, bsb2, mac2 );
```

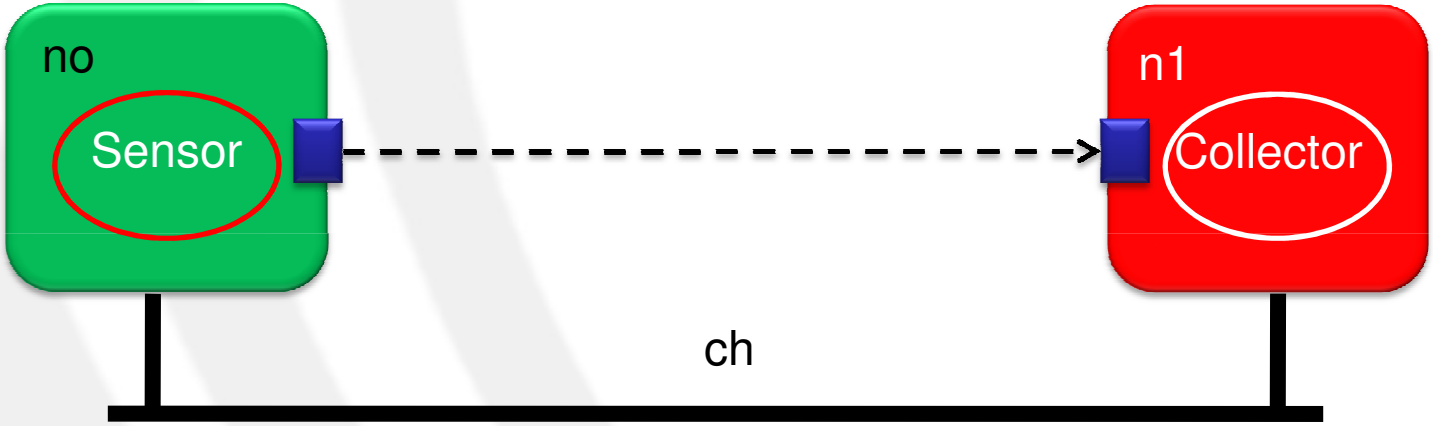
Note: Node binding should be done from both sides

n1 -> ch -> n2 and n2 -> ch -> n1

Tutorial examples

NOTE: to compile tutorial examples, copy and paste tests/ folder from the zip archive into trunk/ overwriting the original one

Example # 1
Two Nodes



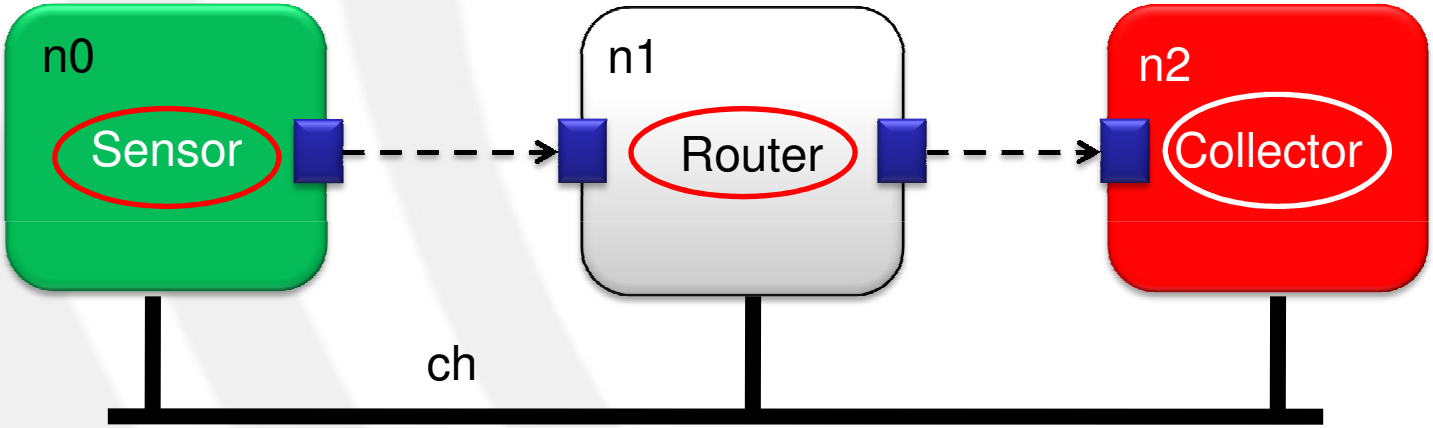
Proxy number: 1 1

EXERCISE

1. Calculate the minimum transmitting power.

Suggestion : If the transmitting power is **lower than** the minimum transmitting power, **no** packets will arrive to the receiver. (Packet Loss Rate (PLR)=100%)

Example # 2
Three nodes with router



Proxy number: 1 2 1

Binding mechanism

```
scnsl->bind( n0, ch, bsb0 );
scnsl->bind( & sensor, & router, ch, bsb0, mac0 );

scnsl->bind( n1, ch, bsb1 );
scnsl->bind( & router, & collector, ch, bsb1, mac1 );
scnsl->bind( & router, NULL , ch, bsb1, mac1 );


scnsl->bind( n2, ch, bsb2 );
scnsl->bind( & collector , NULL, ch, bsb2, mac2);
```

EXERCISES

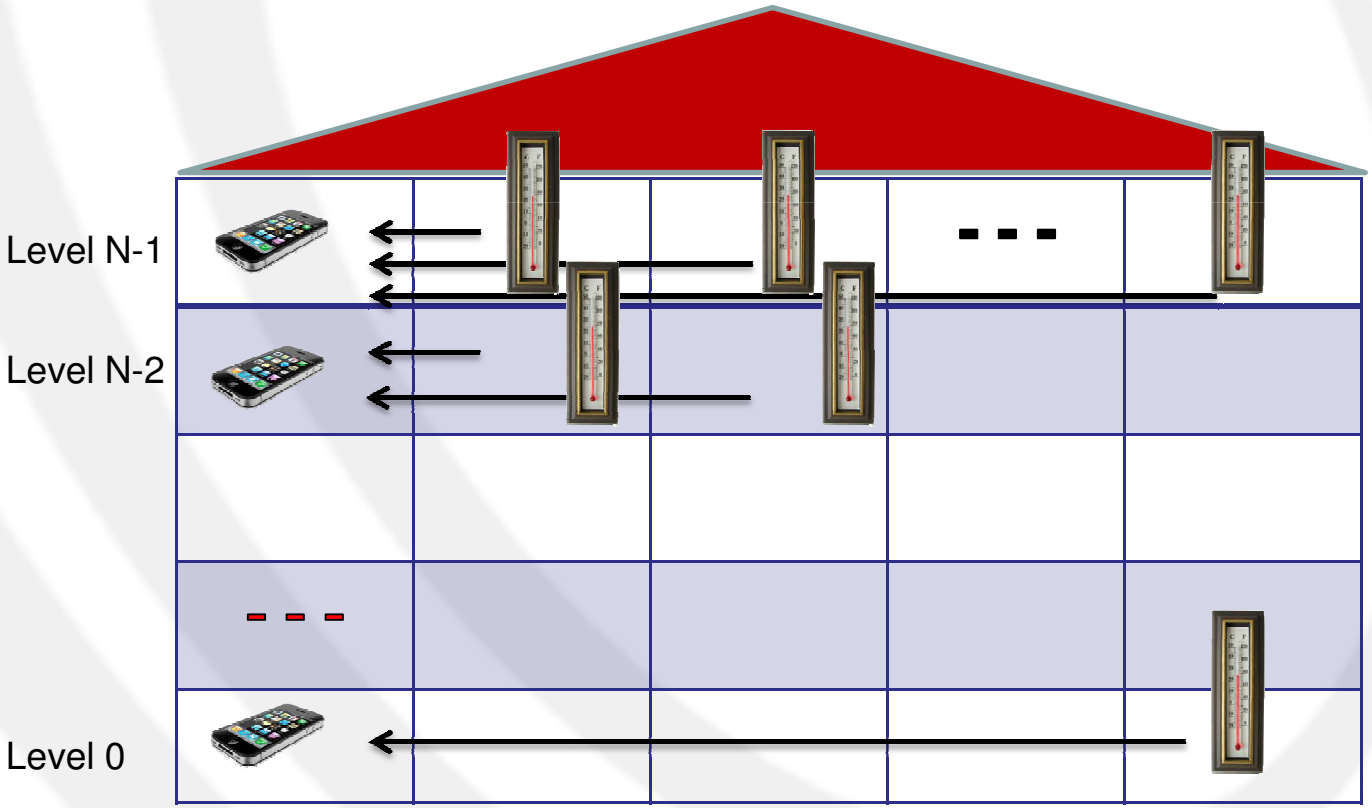
1. Calculate the end-to-end delay
2. Calculate the single-hop delay
3. Calculate Packet Loss Rate
   - Use “grep” and “wc –l” BASH commands to count sent and received packets.
4. Change the transmitting power to the minimum and calculate the new PLR

Example # 3
Temperature monitoring for Building automation

There are N floors with N rooms for each floor.

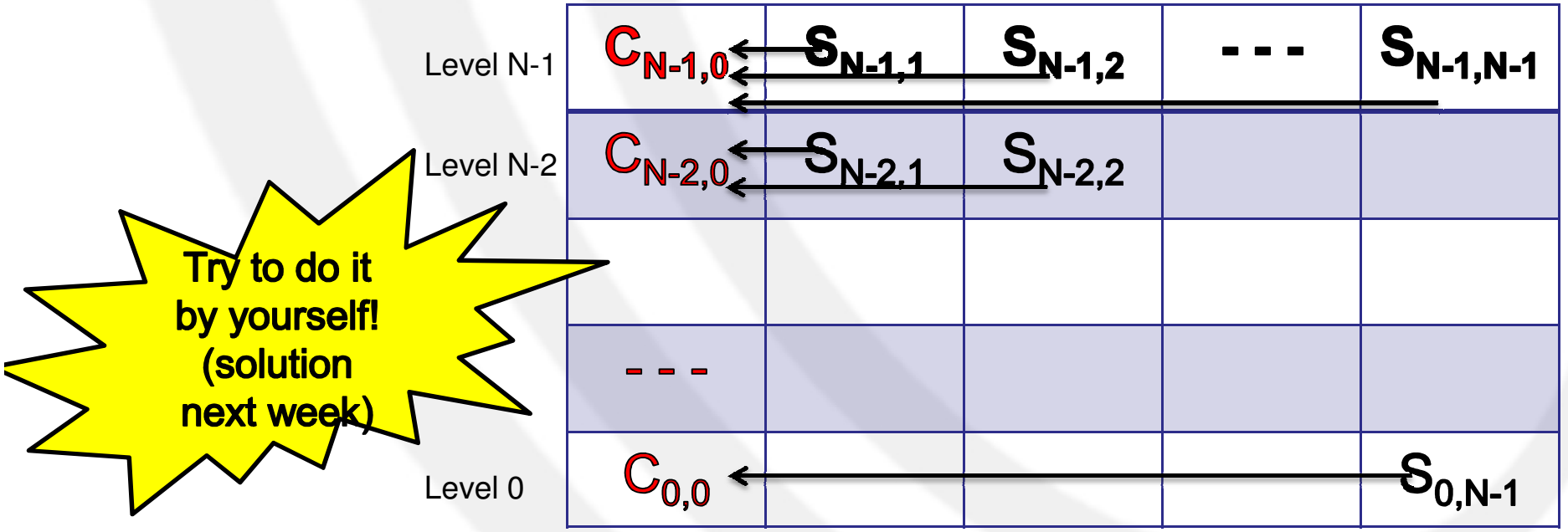
In each floor there is one sensor for each room which sends data to a controller.

There is a separate controller for each floor



Example # 3
Simulation scenario

Idea: Network consists of a NxN matrix.
Nodes in the first column will work as a collector node (RX only) while other nodes as sensor nodes (TX only)



EXERCISES

1. Write a simplified version of the scenario
   - 1 floor with 3 nodes (1 collector and 2 sensors)
2. Calculate Packet Loss Rate
   - Use “grep” and “wc –l” BASH commands to count sent and received packets.
3. Change the transmitting power to the minimum needed by each sensor and calculate the new PLR