BitSimulator, a C++ wireless nanonetwork simulator for routing and transport levels

Introduction

BitSimulator is dedicated to wireless nanonetworks simulations. Nanonetwork's nodes are of micrometric dimensions. As such, they have drastic constraints on memory, energy and CPU. Those constrains along with the possibility of extremely high neighborhood densities call for specific network protocols.

BitSimulator has been designed to allow simulation of application or routing protocols while keeping a relatively detailed model for the MAC and physical levels. As such, it enables exploration and understanding of the effects of low level coding and channel access contention. For its Medium Access Control, BitSimilator uses the TS-OOK modulation proposed by Josep Jornet.

BitSimulator differs from other network simulators as it is completely dedicated to wireless and potentially very dense nanonetworks. Due to fine memory management and clever optimizations, simulations of up to hundreds of thousands of nodes are possible on a laptop. Of course, running time greatly varies with the complexity of the simulated scenario and with neighborhood density.

We actively develop the simulator, feel free to use it and contact us for any remark.

Be sure to have also read the NanoCom conference paper describing it.

Examples of BitSimulator usage

We use the simulator to discover behaviours of communications in naonnetworks and to validate our algorithms and protocols. You might be interested by the following articles which use the simulator in the evaluation part:

- overview of the simulator and some examples of use
- density estimator
- backoff flooding
- deviation routing and code to reproduce the results.
- sleeping mechanism and code to reproduce the results.
- sleeping mechanism on heterogeneous networks and info on how to reproduce the results.
- ring-based forwarding and code to reproduce the results.

Features

- Propagation delay: packet arrival time on a node depends on its distance from the sender (extremely important considering the duration of TS-OOK pulses).
- Collisions: computation of collisions uses the TS-OOK model by checking the actual bit value of each packet currently being received at each node.
- Being very focused on nanocommunications, its design is kept simple and efficient. It allows it to scale up to hundred of thousands of simulated nodes.

- A simple infrastructure helps to build new routing protocols of applications (C++ classes to derive from).
- It implements several protocols: SLR routing protocol, probabilistic flooding, backoff flooding, SLR backoff flooding, and others
- It comes with a visualisation tool, VisualTracer.
- And much more.

The manual below gives more information about features and limitations.

Tutorial / Get started with BitSimulator

First steps with the simulator

Create a directory, e.g. `example`. Then create inside an XML file named `scenario.xml` containing the following lines:

```
<world sizeX_nm="6000000" sizeY_nm="0" sizeZ_nm="6000000">
  <genericNodes count="1000" positionRNGSeed="1"/>
</world>

<modulation>
  <ts-ook pulseDuration_fs="100" defaultBeta="1000"
        defaultCommRange_nm="500000" maxConcurrentReceptions="10"
        minIntervalBetweenSends="1000" minIntervalBetweenReceiveAndSend="1000
</modulation>

<routing defaultBackoffWindow="10000" backoffRNGSeed="1">
  <PureFloodingRouting/>
</routing>

<applications>
  <cbr flowId="0" srcId="3" dstId="10" port="3001" packetSize="1000"
      repetitions="3" interval_ns="300000" startTime_ns="6000000"/>
</applications>

<log/>
</scenario>
```

Then start the simulation with the above scenario with the following command line:

```
./bitsimulator -D example
```

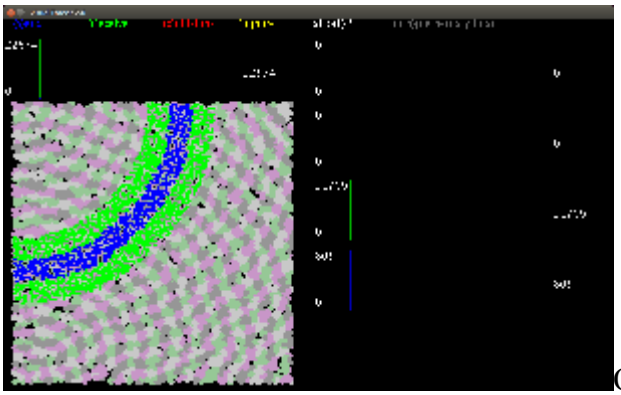
The simulation generates several log files in the directory, the most important being `events.log` file, which traces all basic events in the network, such as packet receptions and emissions.

Note that a lot of parameters from the configuration file can be overwritten by options given in command line. This is especially useful when doing batch runs from shell scripts. For more information on the available options use:

```
./bitsimulator -h
```

BitSimulator has a built-in graphic mode which appears when started with the option `-g`. It opens an SDL window displaying the propagation of packets during the simulation. It is very useful when debugging your own protocols and applications to detect and prematurely end useless simulations.

First steps with VisualTracer

 Once the simulation ended, the simulation progress from the beginning can be visualized step by step with VisualTracer using the following command (in `visualtracer` directory):

```
./visualtracer -D ../example
```

Documentation

The user's and developer's manual, with tutorials too.

The API documentation can be generated with doxygen.

Download

The simulator works on GNU/Linux and macOS.

We currently use a private git repository for the simulator. Until it gets public, the simulator can be downloaded as a tar.gz file (v0.9.4+, as of 8 May 2021). It is licensed as GPL. Note that this is the stable version, which might be outdated in some respects compared to the information in this web page.

Installation

This is done through the usual `./configure && make && make install` triplet, and optionally `make check`.

`configure` checks for required dependencies: pkg-config, tinyxml2, sdl2, sdl2_gfx, sdl2_ttf, tclap. For macOS you could install these dependencies using `homebrew` for example (note that `macports` does not have `tinyxml2` for example).

`make install` is optional, it just copies the two binary files `bitsimulator` and `visualtracer` to the installation directory, but you can execute them from the source directory directly if you prefer.

You might also execute `make check`, which executes a short simulation and tests its result.

Contact

Dominique Dhoutaut, associate professor at University of Franche-Comté, France.