**Wireless Energy Harvesting for the Internet of Things**

**Software:** NetSim Standard v13.1 (64-bit), Visual Studio 2019

**Project Download Link:**

Follow the instructions specified in the following link to download and setup the Project in NetSim:

<https://support.tetcos.com/en/support/solutions/articles/14000128666-downloading-and-setting-up-netsim-file-exchange-projects>

**Introduction**

Among different energy harvesting methods, such as vibration, light, and thermal energy extraction, wireless energy harvesting (WEH) has proven to be one of the most promising solutions by virtue of its simplicity, ease of implementation, and availability. This recent technology trend in energy harvesting provides a fundamental method to prolong battery longevity. While harvesting from the environmental sources is dependent on the presence of the corresponding energy source, RF energy harvesting provides key benefits in terms of being wireless, readily available in the form of transmitted energy (TV/radio broadcasters, mobile base stations, and handheld radios), low cost, and small form factor implementation.

A WEH-enabled sensor device usually consists of an antenna, a transceiver, a WEH unit, a power management unit (PMU), a sensor/processor unit, and possibly an onboard battery. The available harvested power, *PH*, is given by a Friis equation and is directly proportional to the transmitted power, *PT*, path loss, *PL*, transmitter antenna gain, *GT*, receiver antenna gain, *GR*, power conversion efficiency of the converter, *PCEH*, and the square of the wavelength, l, and is inversely proportional to the square of the communication distance, *r*, between the source and the device.

The communication energy consists of *ELS* (listening energy), *ERX* (receiver energy), and *ETX* (transmitter energy). The computation energy includes *EPR* (processing energy) and *ESN* (sensing energy). To capture the energy distribution among the a forementioned energy consumers, weighting coefficients aLS> a*TX* > a*RX* > a*PR* > a*SN* are assigned to them. The total average energy consumption *ED* = a*LS ELS* + a*TX ETX* + a*RX ERX* + a*PR* E*PR* + a*SN ESN*. *EB* is the total energy stored in the battery, and *EH* is the available harvested energy per active-duty cycle. We assume constant energy consumptions for receiver, processor, and sensor. However, the energy consumption of the transmitter (*ETX*) is directly proportional to *rij* 2, where *rij* is the distance between the originating device *j* and the sink node *i* (in ring topology) or the sink node/sensor device (in multihop topology). The harvested energy *EH* is inversely proportional to *rij* 2 (here *j* is the sink node and *rij* = *rji*).

# IEEE Ref Paper:

Wireless Energy Harvesting for the Internet of Things

P. Kamalinejad C. Mahapatra ; Z. Sheng ; S. Mirabbasi ; V. C. M. Leung ; Y. L. Guan

*IEEE* COMMUNICATIONS MAGAZINE · JUNE 2015

**Steps to simulate**

1. Open the Source codes in Visual Studio by going to Your work-> Source code and Clicking on Open code button in NetSim Home Screen window.
2. Right click on the solution in the solution explorer and select Rebuild.

Graphical user interface, text, application, Word

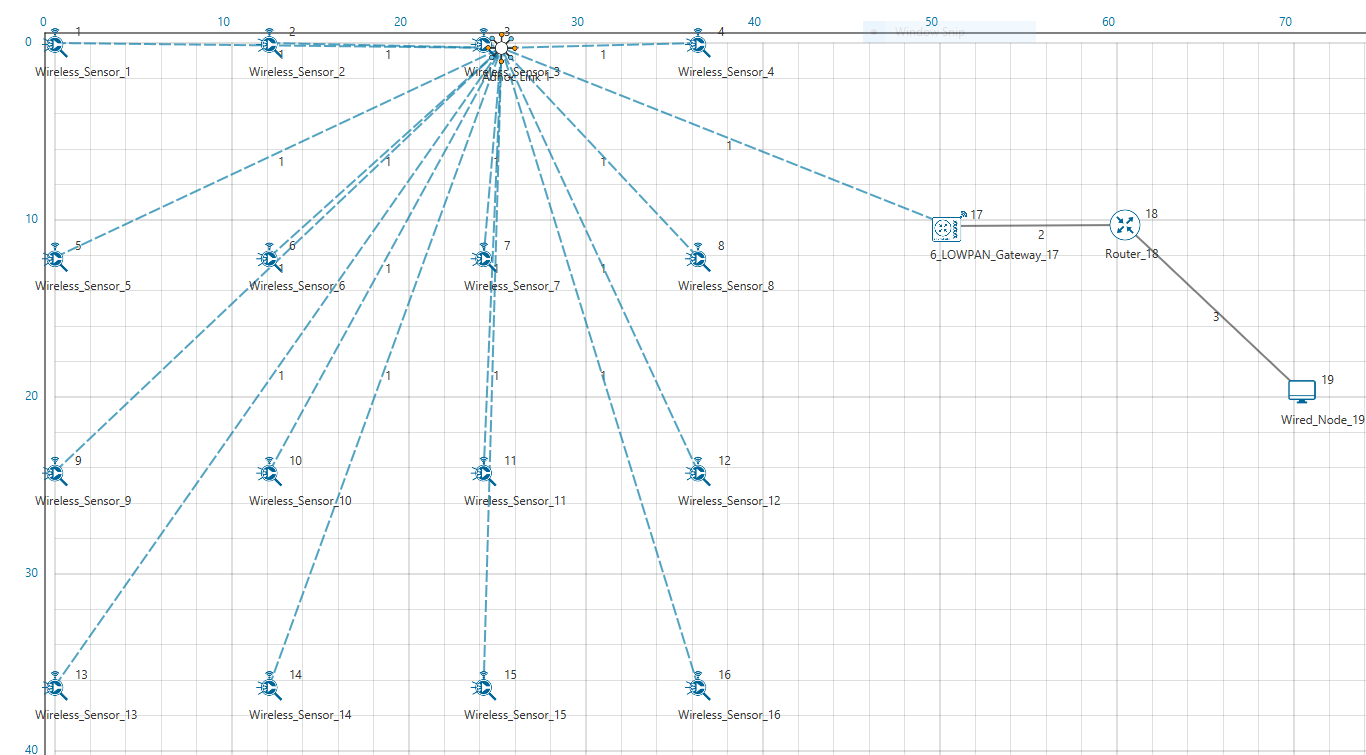
Description automatically generated

**Figure 1:** Screen shot of NetSim project source code in Visual Studio

1. Upon successful build modified libZigBee.dll and BatteryModel.dll file gets automatically updated in the directory containing NetSim binaries.

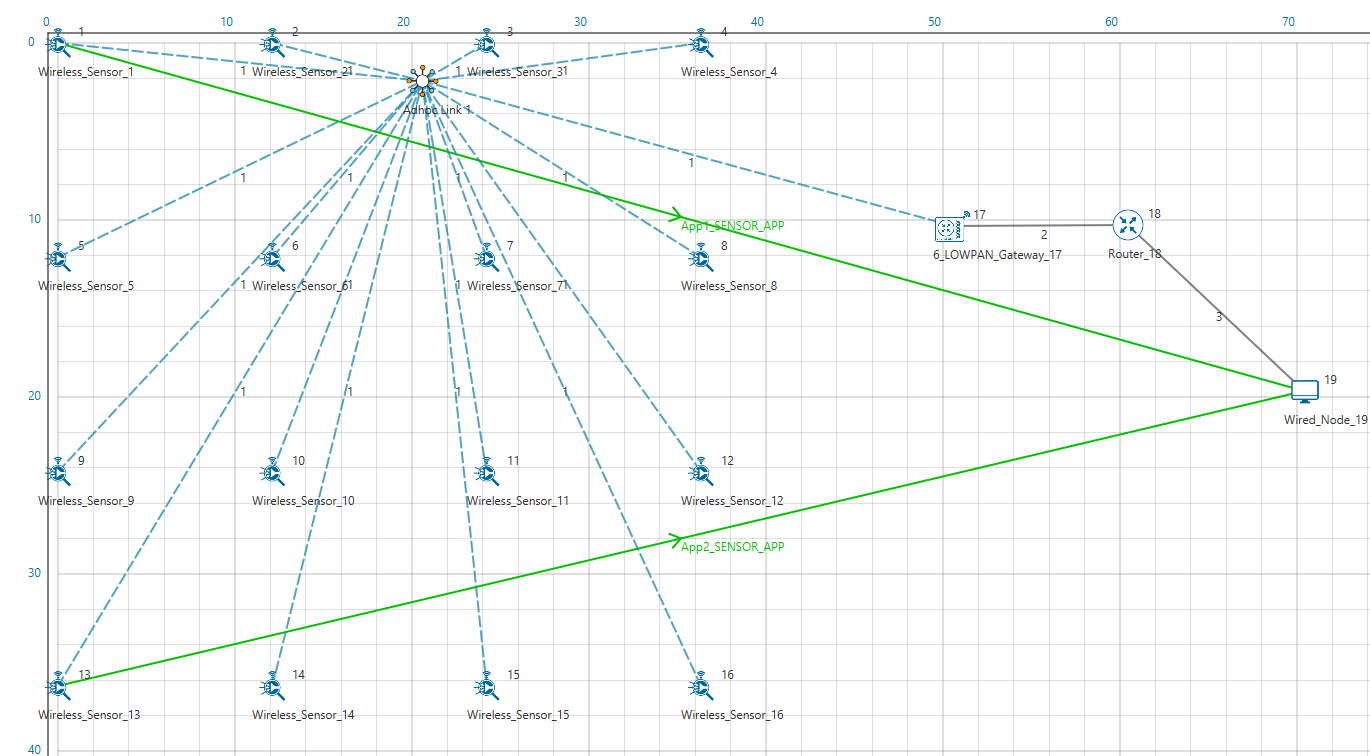
**COMPARATIVE ANALYSIS:**

1. The **WSN\_Energy\_Harvesting\_Workspace** comes with a sample network configuration that are already saved. To open this example, go to Your work in the Home screen of NetSim and click on the **WSN\_Energy\_Harvesting\_Example** from the list of experiments.
2. Create a network scenario in IoT with say 16 sensors, a 6LoWPAN Gateway, a Router and a Wired Node as shown below.



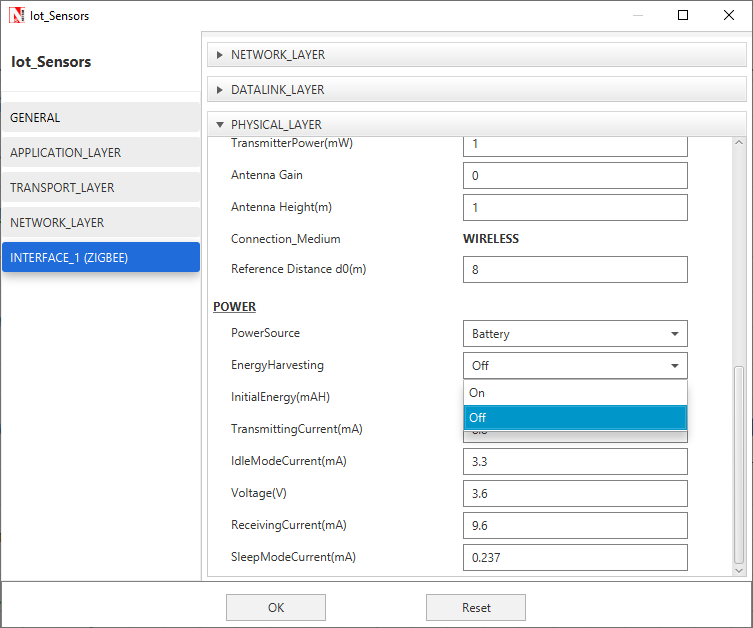
**Figure 2:** Energy Harvesting Network Topology

1. Configure traffic in the network by setting a few applications between some of the sensor nodes to the Wired Node using the Application Icon, as shown below.



**Figure 3:**Configured traffic wireless sensor to Cloud server

1. Disable Energy Harvesting in all Sensor nodes by setting the EnergyHarvesting parameter to OFF in the Interface (ZigBee) Properties of the sensor nodes as shown below:



**Figure 4:** Disable Energy Harvesting in all Sensor nodes

1. Run Simulation for say 100 seconds and save the simulation results. In NetSim Simulation Results Window, the Battery model table provides detailed metrics related to energy consumed by each sensor node. The column Remaining energy can be used to compare simulations with and without energy harvesting code modification.

**WITH ENERGY HARVESTNG**

1. Now re-run the network simulation for 100 seconds and save the simulation results. You can use the table remaining energy column present in the Battery model table which is part of NetSim Simulation Results window to compare simulations with and without wireless energy harvesting.

Table

Description automatically generated

**Figure 5:** Battery model table from result dashboard

Now on comparing the custom IOT metrics we can observe that Energy Harvesting increases sensors’ working capability. Simulations can be performed for different values of EH Fraction which may vary as per the efficiency of the Energy Harvesting unit.

**Appendix: NetSim source code modifications**

**Changes to Battery Model.h within Battery Model project**

/\* We implemented the Batter Model\*/

#ifndef \_NETSIM\_BATTERY\_MODEL\_H\_

#define \_NETSIM\_BATTERY\_MODEL\_H\_

#ifdef \_\_cplusplus

extern "C" {

#endif

#ifndef \_BATTERY\_MODEL\_CODE\_

#pragma comment(lib,"BatteryModel.lib")

typedef void\* ptrBATTERY;

#endif

\_declspec(dllexport) ptrBATTERY battery\_find(NETSIM\_ID d,

NETSIM\_ID in);

\_declspec(dllexport) void battery\_add\_new\_mode(ptrBATTERY battery,

int mode,

double current,

char\* heading);

\_declspec(dllexport) ptrBATTERY battery\_init\_new(NETSIM\_ID deviceId,

NETSIM\_ID interfaceId,

double initialEnergy,

double voltage,

double dRechargingCurrent);

\_declspec(dllexport) bool battery\_set\_mode(ptrBATTERY battery,

int mode,

double time);

\_declspec(dllexport) void battery\_animation();

\_declspec(dllexport) void battery\_metrics(PMETRICSWRITER metricsWriter);

\_declspec(dllexport) double battery\_get\_remaining\_energy(ptrBATTERY battery);

\_declspec(dllexport) int battery\_energy\_harvesting(ptrBATTERY battery, double eh\_energy);

\_declspec(dllexport) double battery\_get\_consumed\_energy(ptrBATTERY battery, int mode);

#ifdef \_\_cplusplus

}

#endif

#endif //\_NETSIM\_BATTERY\_MODEL\_H\_

**Changes to double battery\_get\_remaining\_energy (), Battery Model.c within Battery Model project**

\_declspec(dllexport) double battery\_get\_remaining\_energy(ptrBATTERY battery)

{

return battery->remainingEnergy;

}

\_declspec(dllexport) int battery\_energy\_harvesting(ptrBATTERY battery, double eh\_energy) {

double eh\_energy\_mJ = eh\_energy \* ((pstruEventDetails->dEventTime - battery->modeChangedTime) / 1000000);

battery->remainingEnergy += eh\_energy\_mJ;

}

**Changes code to ChangeRadioState.c, within Zigbee project at the end of the file**

#define EH\_FRACTION 0.1

// EH\_FRACTION is the fraction of the received signal energy that can be

// captured and harvested by the sensor.

int calculate\_eh(NETSIM\_ID dev1, NETSIM\_ID dev2)

{

double rx\_pwr = GET\_RX\_POWER\_mw(dev1, dev2, pstruEventDetails->dEventTime);

double eh\_energy = EH\_FRACTION \* rx\_pwr;

ptrBATTERY battery = WSN\_PHY(dev2)->battery;

if (battery)

battery\_energy\_harvesting(battery, eh\_energy);

}

**Changes code to int fn\_NetSim\_Zigbee\_Run(),** 802\_15\_4.c file**, within Zigbee project**

case UPDATE\_MEDIUM:

{

double dtime=pstruEventDetails->dEventTime;

NETSIM\_ID nLink\_Id, nConnectionID, nConnectionPortID, nLoop;

NETSIM\_ID nTransmitterID;

nTransmitterID = pstruEventDetails->nDeviceId;

ZIGBEE\_CHANGERADIOSTATE(nTransmitterID, WSN\_PHY(nTransmitterID)->nRadioState, RX\_ON\_IDLE);

if(WSN\_PHY(nTransmitterID)->nRadioState != RX\_OFF)

WSN\_MAC(nTransmitterID)->nNodeStatus = IDLE;

nLink\_Id = fn\_NetSim\_Stack\_GetConnectedDevice(pstruEventDetails->nDeviceId,pstruEventDetails->nInterfaceId,&nConnectionID,&nConnectionPortID);

for(nLoop=1; nLoop<=NETWORK->ppstruNetSimLinks[nLink\_Id-1]->puniDevList.pstruMP2MP.nConnectedDeviceCount; nLoop++)

{

NETSIM\_ID ncon = NETWORK->ppstruNetSimLinks[nLink\_Id-1]->puniDevList.pstruMP2MP.anDevIds[nLoop-1];

if(ncon != pstruEventDetails->nDeviceId)

{

calculate\_eh(nTransmitterID, nLoop);

WSN\_PHY(ncon)->dTotalReceivedPower -= GET\_RX\_POWER\_mw(nTransmitterID,ncon,pstruEventDetails->dEventTime);

if(WSN\_PHY(ncon)->dTotalReceivedPower < WSN\_PHY(ncon)->dReceiverSensivity)

WSN\_PHY(ncon)->dTotalReceivedPower = 0;

}

}

This completes the code modifications for energy harvesting.