SenScript

SenScript is the script used to program sensor nodes of the CupCarbon simulator. It is a script where variables are not declared and without types, but they can be initialized (**set** command). It is possible to use the instruction **function** to add complex and additional functions programmed in Java (in a source code mode only).

Example:

SenScript: **set x "abcd"** → Java: **x = "abcd";**
SenScript: **set y x** → Java: **y = x;**
SenScript: **set z x+y** → Java: **z = x+y;**

List of Commands

Language	Sensor	Radio Module	Embedded Card	Messages Data	Mobile	Visualization
inc / dec conc hash int max / min smax / smin and / or xor / not band / bor bxor / bnot for end if [else] end while end goto function math rand / randb rgauss set tab / vec tget / tset vget / vset angle / angle2 sadd / spop / nth charat / length	areadsensor dreadsensor getpos getpos2 getinfo rotate	atch atid atpl atmy atnd atnid atget drssi distance	loop stop buffer cbuffer delay led mark print printfile battery time kill script rscript	data rdata vdata send receive wait read pick	move rmove route	Simulation cprint edge

and (logic)

```
and x a b
```

→ x = a & b

angle / angle2

```
angle a x1#y1 x2#y2 x3#y3
```

→ a = angle formed by the edge {(x2, y2),(x1,y1)} and {(x2, y2),(x3,y3)}

```
angle2 a x1 y1 x2 y2 x3 y3
```

→ a = angle formed by the edge {(x2, y2),(x1,y1)} and {(x2, y2),(x3,y3)}

areadsensor (Analog Read Sensor)

```
areadsensor s
```

→ s=X or s=S#y#v, where X means that there is no sensed value (or event), S means that a sensed value has been read from the event having id=y, where the read value is equal to v (use the function rdata function to read these values). If there is many events then s=S#s_1#v1#s_2#v2#S_3#v3… (use rdata function to read these values). Note that, s_2, s_3, … can be null if there is no many events.

atch

```
atch 12
```

→ Change the channel of the current sensor to 12

atid

```
atid 1111
```

→ Change the identifier of the sensor to 1111

atpl

```
atpl 60
```

→ Change the signal power (power level) of the radio of the sensor. In this example, the sensor will use only 60% of its radio range.

atmy

```
atmy 4
```

→ Change the my of the sensor to 4.

atnd

```
atnd n
```

→ n = the number of the neighbors of the sensor.

```
atnd n v
```

→ n = the number of the neighbors of the sensor and v the vector of the identifiers of each neighbor (use vget to get the values of v)

atnid

```
atnid 2222
```

→ Change the network address

atget

```
atget id x
atget my x
atget ch x
```

→ x = id, my or ch of the sensor node

band

```
band x 1010 1100
```

→ x = 1000

battery

```
battery x
```

→ x = the level of the battery in Joules

```
battery set x
```

→ The level of the battery will be set to x

bnot

```
band x 1010
```

→ x = 0101

bor

```
bor x 1010 1100
```

→ x = 1110

buffer

```
buffer x
```

→ x = the size of the buffer in bits

bxor

```
band x 1010 1100
```

→ x = 0110

cbuffer

```
cbuffer
```

→ clear the buffer

charat
```
charat c hello 1
```
→ c = 'e'
the character situated in the index 1 of hello

conc
```
conc x a b
```
→ concatenate a and b and assign it to x (=ab). In java this command is written as `x = "a"+"b";`

```
conc x a b
```
→ concatenate the value of the variable a with the value of the variable b

cprint
```
cprint "hello" x
```
→ The same function as print where the result is displayed in the console

data
```
data p 1 4 6
```
→ p = 1#4#6

dec
```
dec x
```
→ x = x - 1

delay
```
delay 1000
```
→ wait 1 second (1000 ms) before the next instruction

distance
```
distance x 2
```
→ x = the Euclidean distance between the current sensor and a communicating sensor node having an id=2.

dreadsensor
```
dreadsensor x
```
→ x=1 if the sensor detects an event (mobile), x=0, otherwise

drssi
```
drssi x
```
→ x = the rssi value transformed to a Euclidean distance between the current sensor the last sending sensor node.

egde
```
edge a x
```
→ to mark (if a=1) or unmark (if a=0) the edge (communication link) between the current sensor node and the node having the id x.

for end
```
for i 0 10
print "i = " i
delay 1000
end
```
→ the same as the following for on C: for(int i=0; i<10; i+=1)

```
for i 0 10 2
print "i = " i
delay 1000
end
```
→ the same as the following for on C: for(int i=0; i<10; i+=2)

function
The function command can be used only with the source code of CupCarbon.

```
function x myf 1,2,v
```
→ execute the function myf with the arguments 1, 2 and the value of v. The obtained result is returned in x.

To create your own function (example: myf(args)):

1. Add the following script in the method function of the class ScriptFunction (package script_functions):

```
if(function.equals("myf")) {
return Function_Calc.myf(args);
}
```

2. Add the methode myf the class Functions as follows:

```
public static String myf(String [] args) {
    String valToReturn = "";
    // TODO
    // Your program here
    return valToReturn;
}
```

getinfo

```
getinfo p
```
→ if a mobile is detected by the sensing unit, it returns in p information formed by the id and the coordinates of the mobile in a format: id#longitude#latitude.

getpos

```
getpos pos
```
→ pos = the position of the sensor longitude#latitude

→ pos = "-5.489438533782959#48.390415333407574"; // example

```
getpos pos
rdata pos x y
```
→ x = longitude; // example: -5.489438533782959

→ y = latitude; // example: 48.390415333407574

getpos2

```
getpos2 x y
```
→ x = longitude and y = latitude
→ x = -5.489438533782959; and y = 48.390415333407574; // example

goto

```
goto 5
```
→ Go to to the line of the code number 5

```
goto B
```
→ Go to to the line labeled by B

To label a line by B:

```
B:set x 12
```

hash

```
hash x hello
```
→ assign to x the hash code of hello

if [else] end

```
if (x==1)
    mark 1
else
    mark 0
end

if ((x==1) && ((y>0) || (y<5)))
    mark 1
```

```
else                                    end
    mark 0
```

- In the current version of CupCarbon, it is not possible to write: `if (x==1 && y>0) ...`
- One must write: `if ((x==1) && (y>0)) ...`

inc

```
inc x
```

→ x = x + 1

int

```
int x a
```

→ assign to x the integer value of a (if a=3.2 then x=3)

kill

```
kill 0.3
```

→ the current node will be killed (i.e., with empty battery) with a probability of 30%

led

```
led 1 2
```

→ set the color number 2 to the sensor (1 is not important here, it represents the pin number in the read card)

length

```
length v hello
```

→ v = 5
the length of the string “hello”

loop

```
loop
```

→ Starting the loop section

mark

```
mark 1
```

→ Mark or unmark (`mark 0`) the sensor

math

```
math f y x
```

→ y = f(x) where f = "sqrt, sin, cos, tan, asin, acos, atan, abs, pow, log, log10, exp".

Example:

```
math sin y 3.14
```

→ y=sin(3.14)

max

```
max x 4 3
```

→ assign to x the maximum value between 4 and 3 (4 and 3 can be replaced with variables)

min

```
min x 4 3
```

→ assign to x the minimum value between 4 and 3 (4 and 3 can be replaced with variables)

move

```
move x y z t
```

→ The sensor node will go to the position latitude (x), longitude (y), and elevation (z) with the speed t meters/second.

not (logic)

```
not x a
```

→ x = ~a

nth
```
nth v i t
```
→ v will be equal to the i^{th} value of t (t is in a format of a string with the separator &)
→ Example: if t=4&9&2&7 then nth v 1 t will put in v the value 9 (v=9) and t remains the same.

or (logic)
```
or x a b
```
→ x = a | b

pick
```
pick x
```
→ same as read x without removing the read data from the buffer

printfile
```
printfile "hello"
```
→ add the string “hello” in a file generated during the simulation in the directory *results* with the name of the sensor node

print
```
print "hello"
```
→ display hello

```
print x
```
→ display the value of x

```
print "i = " i
```
→ display "i = (value of i)"

```
print ""
```
→ display nothing

radio
```
radio [name of a radio module]
radio radio2
```
→ Select the radio module having the name radio2. This means that any **send** command that will be used after this instruction will consider the radio2 as the radio module which will send the message. The considered standard is the one of the considered radio module.

rand
```
rand [variable]
rand x
```
→ x = rand

randb
```
reandb [varialbe][inf][sup]
reandb x 2 6
```
→ x = random values between 2 (included) and 6 (included)

rdata
```
rdata p a b
```
→ We assume that p is a message formed using the command **data**. It is possible to form manually p which is a string with # separator (example: p=hello#4). In this example, **rdata p a b** will lead to an a="hello” and b=4

read
```
read x
```
→ Assign the value of the buffer to x

receive
```
receive x
```

→ Wait until receiving data in the buffer and assign it to x. This is a blocking function, if there is not data in the buffer then it remains blocked on this instruction.

```
receive x t
```
→ Wait until receiving data in the buffer. If there is no data received after t milliseconds then x will receive an empty message (x="") and the execution of the script will be continued.

rgauss
```
rgauss x
```
→ x = Standard Gaussian random value

rmove
```
rmove t
```
→ Go to the next point of the route created by the markers. A route must be assigned before simulation to the sensor node. The next instruction of the script will be executed after t milliseconds. Note that the mobility check box in the simulation parameters view must be activated.

rotate
```
rotate x t
```
→ Rotate with x (double) units the Directional sensor node. The next instruction of the script will be executed after t milliseconds.

route
```
route route1
```
→ The sensor node will change its route to route having the name route1 after intersecting with it.

rscript
```
rscript flooding
```
→ Load the script floodint.csc while reinitializing the variables of the environment (see script function)

sadd
```
sadd x t
```
→ add the value of x to the stack t which is in a format of string separated by the symbol &. You can use the command spop to get the first element of the stack t.

→ Example: if t = 4&6&7, sadd 56 t will lead to: t = 56&4&6&7

script
```
script flooding
```
→ Load the script flooding.csc without reinitializing the variables of the environment (see rscript function)

send
```
send hello 2
```
→ Sends hello to the sensor having an id=2

```
send p 2
```
→ Sends the value of p to the sensor having an id=2

```
send p
```
→ Sends the value of p in a broadcast mode

```
send p *
```
→ Sends the value of p in a broadcast mode(the same as send p)

```
send p * 3
```
→ Sends the value of p in broadcast except the sensor having an id=3

```
send p 0 4
```
→ Sends the value of p to sensors having a MY address equal to 4

```
send p 0 0 5
```

→ Direct sending of the value p to sensors having an id equal to 5 (like in a GPRS/3G/4G mode)

```
send !color 1
```
→ Change the color of the send link on the IHM. It is practical to differentiate the type of the communication links(of the send instruction) between sensors.

set

```
set x 5
```
→ x = 5

```
set s hello
```
→ x = "hello";

```
set x ""
```
→ x = ""

```
set z x*5
set z (x*y)/3
set z x%y
set z (x+y)%2
```

simulation

```
simulation 50 200
simulation sspeed 50
simulation aspeed 200
```
→ just for the visualization and it is not considered in the simulation process. It allows to change the simulation speed during the simulation process.

smin

```
smin m x y
```
→ x and y must be in a format a#b. If x=a#b and y=c#d then m=x if b<d or m=y if d<=b

smax

```
smax m x y
```
→ x and y must be in a format a#b. If x=a#b and y=c#d then m=x if b>d or m=y if d>=b

spop

```
spop v t
```
→ retrieve the first value from the stack t (in a format of a string with the separator &) and put it in the variable v.
→ Example: if t = 4&6&7, sad v t will put in v the value 4 and t becomes t = 6&7

stop

```
stop
```
→ Stop the execution of the script

tab

```
tab t 2 5
```
→ create a table t with 2 rows and 5 columns (t[2][5])

tget

```
tget x t 0 2
```
→ x = t[0][2]

time

```
time x
```
→ assign to x the current simulation time

tset

```
tset 55 t 0 2
```
→ t[0][2] = 55

vdata

```
vdata v 14#54#2
```
Transform tokens that are separated with # to a vector, with the name v, of these tokens.
→ v[0]=14, v[1]=54 and v[2]=2

vec

```
vec v 5
```

→ create a vector v with 5 elements (t[5])

vget / vset

```
vget x v 2
```

→ x = v[2]

```
vset 55 v 1
```

→ v[1] = 55

wait

```
wait (deprecated → use receive)
```

→ Wait until receiving data in the buffer

```
wait 1000 (deprecated → use receive)
```

→ Wait until receiving data in the buffer. If there is no data received after 1 second (=1000 milliseconds) then the execution of the script will be continued.

while end

```
while (1)
```

→ The same as "while (true)" in C.

Example:

```
set i 0
while ((i<5) && (i>=0))
    print "i = " i
    delay 1000
    set i i+1              // → inc i
end
```

xor (logic)

```
xor x a b      →  x = a ^ b
```

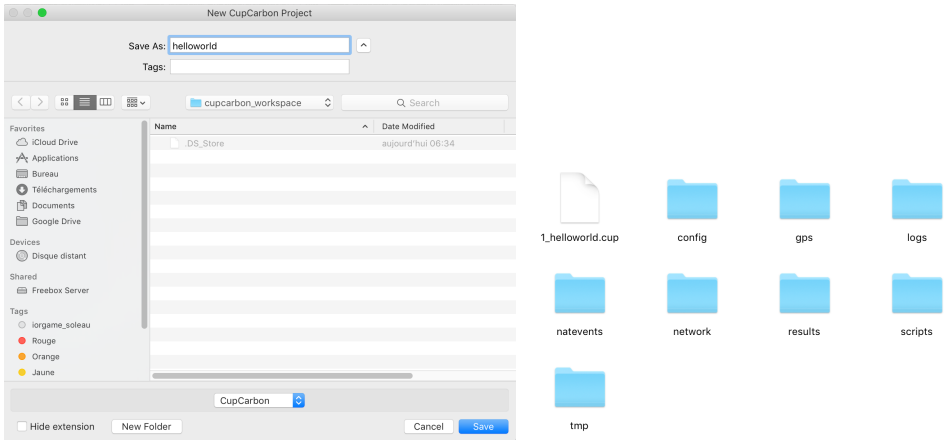
Basic examples

In this section we will present some basic examples that can help to understand how to use the environment of CupCarbon and also how to simulate different scenarios. The different examples can be downloaded from the official web site of CupCarbon (www.cupcarbon.com) and they are included in the source code and the executable.

Example 1: Hello World

This example shows how a sensor can display a message “Hello World”. The following steps show how to do this:

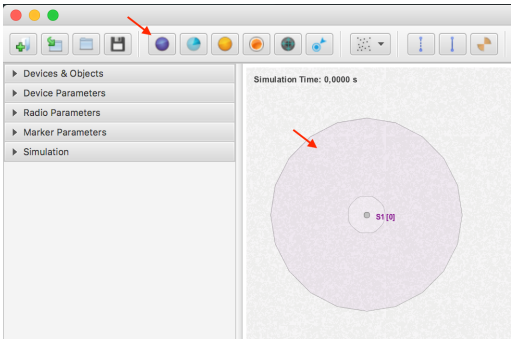
Step 1. Create a new project: this can be done either by clicking on the “New project” icon of the toolbar or on the menu Project → New project. Choose the name (example: helloworld) and the place where you want to save your project. The project will create a new folder with the given project name.

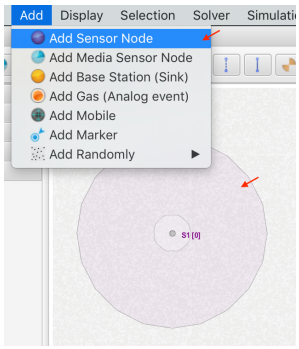


Inside this folder, 1 file (helloworld.cup) and 8 other directories will be created. The content of each directory is given in the following:

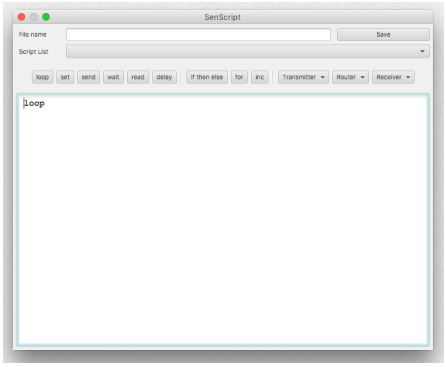
a. config: it contains the simulation parameters file, the building list file, the marker list file and two other directories (sensor and sensor_radios) that contains the list of sensor nodes (one file by sensor node) and the list of the radio modules of each sensor.
b. gps: it contains the list of routes
c. logs: it contains the log file
d. results: it contains the simulation results (a csv file)
e. scripts: it contains the SenScript files of the project
f. netevents: it contains the natural event files
g. tmp, network: are used by the simulator

Step 2. Add a new sensor node on the map: click either on the Add Sensor icon of the toolbar or from the menu bar (Add → Add Sensor Node). Then, click on the map where you want to add the sensor node. Another click will lead to another new sensor node and so on. To stop adding sensor nodes, just click on the right button of the mouse. You can also click on the icon of the toolbar, or by typing on the escape [esc] button of the keyboard.





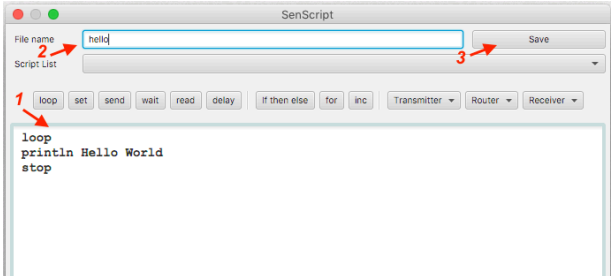
Step 3. Open the SenScript Window: the SenScript window can be opened by clicking on the icon  of the toolbar or from the menu Simulation → SenScript Window.



Step 4. Write the script: add the following script (1) in the text area part of the SenScript window:

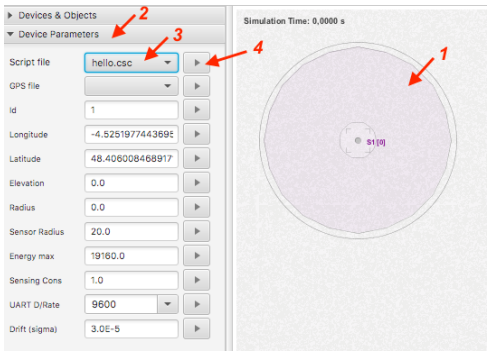
```
loop
print "Hello World"
stop
```

Add the name **hello** of this script in the File name field (2), then click on the Save button (3) just in the left part of this field. This will create a file **hello.csc** in the directory **scripts**.

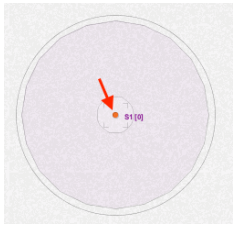


Finally, close the SenScript Window.

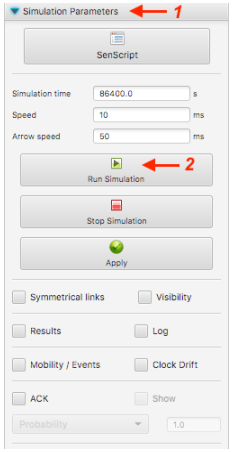
Step 5. Assign the SenScript file to the sensor node: Select the sensor node on the map (1). Go to Device Parameters in the left part of the main window (2). Then, select the hello.csc file in the field Script file (3). And then, click on the apply button just in the right (4).



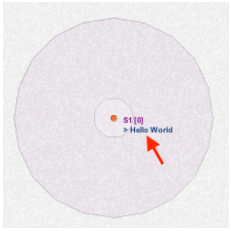
Note that once the script is assigned to a sensor, the center will be colored in orange. This can help to detect graphically sensors without scripts.



Step 6. Run the simulation: For this example, there is no need to parameterize the simulation, just click on the run simulation button in the toolbar or in the simulation parameters menu in the left.



Step 7. Simulation results: in this example the simulation results shows a *Hello World* message displayed by the sensor.



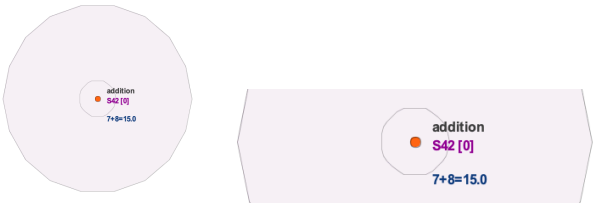
Step 8. Save the project: Click on the icon to save the project.

Example 2: Calculate a+b

This example shows how to calculate the sum of two variables a and b. Repeat all the steps of Example 1 where only the script must be changed as follows:

```
loop
set a 7
set b 8
set x a+b
print a "+" b "=" x
stop
```

The simulation result will display 7 + 8 = 15.



Test with the following script (with a simulation speed of 1000):


```
set a 7
loop
for b 0 11
    set x a*b
    print a " X " b " = " x
    delay 1000
end
```

Example 3: Calculate the sum of a vector

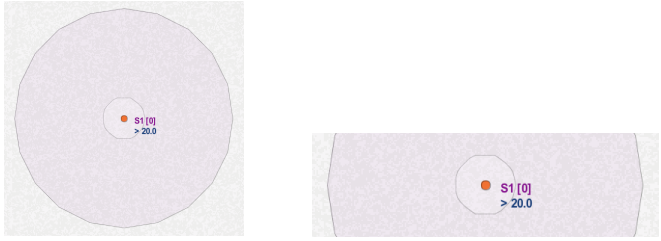
This example shows how to create tables and how to access their elements. As an example, we will try to create a table with 5 rows and 1 column. Then, we will display the sum of its elements.

Repeat all the steps of the Example 1 where only the script must be changed as follows:

```
tab t 5 1
tset 3 t 0 0
tset 5 t 1 0
tset 2 t 2 0
tset 6 t 3 0
tset 4 t 4 0
set s 0
```

```
loop
for i 0 5
  tget x t i 0
  set s s+x
end
print s
stop
```

The simulation result will display 20.



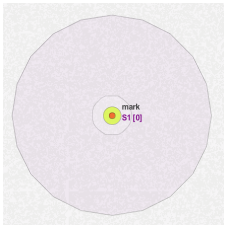
Example 4: Marking nodes

A marker node represents in the reality a sensor node with a switched on led. It helps to do a visible action rather than displaying messages. Marking a node is done by the SenScript command **mark 1**. Unmarking sensor node is done by the command **mark 0**.

To mark a node, use the same steps of Example 1 with the following script:

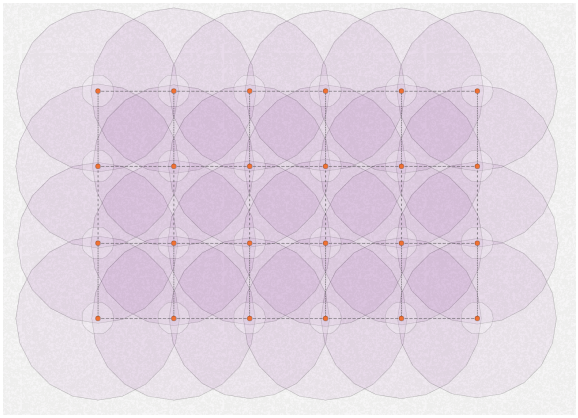
```
loop
mark 1
stop
```

The simulation result will shows a sensor marked with a yellow green center.



Example 5: Marking randomly (game of light)

Add many sensor nodes on the map or generate a randomly network.

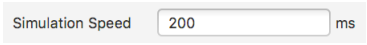


Then, use the same steps of Example 1, with the following script:

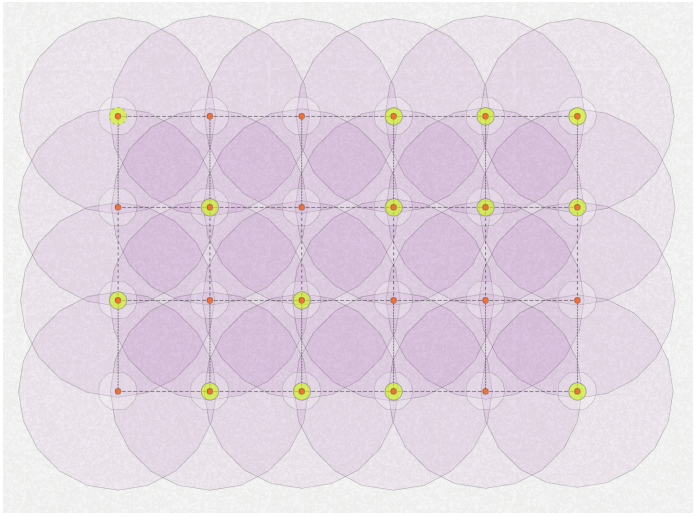
```
loop
rand x
if(x<0.5)
    mark 1
```

```
else
    mark 0
end
delay 1000
```

The simulation speed must be equal to 200:



The simulation results show marked and unmarked sensor nodes that change the marking state during the simulation.



In this example, the simulation will stop only if it reaches the simulation time. To stop manually the simulation, just click on the button .

Example 6: Blinking and LEDs

Blinking

Use Example 3 with the following script:

```
loop
mark 1
delay 1000
mark 0
delay 1000
```

Simulate with the simulation speed equal to 1000.

LEDs

Use Example 3 with the following script:

```
loop
for i 0 15
    led 13 i
    delay 1000
end
```

Simulate with the simulation speed equal to 1000.

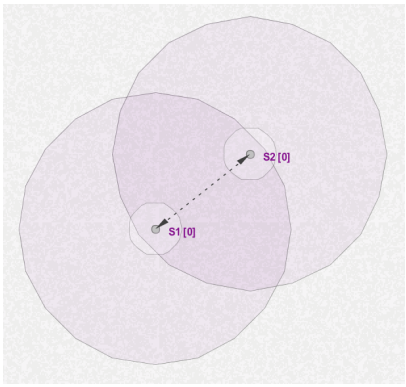
Note that the command **led 13 0** is the same as **mark 0** and the **command led 13 1** is the same as **mark 1**. The other colors are between the values 2 and 14. The number 13 means that this LED will be connected to the pin 13 of the considered embedded card. It is not considered in the simulation.

Example 7: Sending and Receiving messages

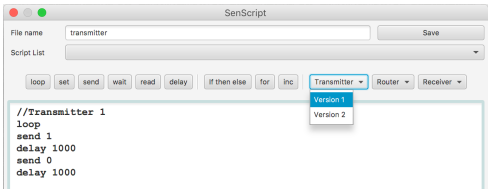
The example of sending and receiving messages is very important since the main actions of the simulation under CupCarbon are based on sending/receiving messages. The following example will show a sensor node that will send each second 0 and 1 to another sensor node. The receiver will be marked if it receives 1 (mark 1) and unmarked if it receives 0 (mark 0).

Step 1. Create a new project: this can be done either by clicking on the "New project" icon of the toolbar or on the menu Project → New project.

Step 2. Add two sensor nodes on the map: click either on the Add Sensor icon of the toolbar or from the menu bar (Add → Add Sensor Node). Then, click on the map where you want to add the sensor nodes so that they can communicate (i.e., There exists a link between the two sensor nodes).



Step 3. Write the SenScript of the transmitter (sensor node 1): The SenScript of the transmitter can be obtained directly from the SenScript window by clicking on the menu button Transmitter (Version 1) which must be completed by adding the id of the receiver (i.e., 2) in the command **send**. Save the file with the name **transmitter**.

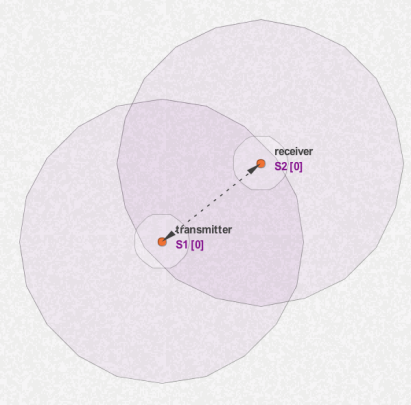



```
loop
send 1 2
delay 1000
send 0 2
delay 1000
```

Step 4. Write the SenScript of the receiver (sensor node 2): As for the first script, The SenScript of the receiver can be obtained directly from the SenScript window by clicking on the menu button Receiver (Version 1). Save the file with the name **receiver**.

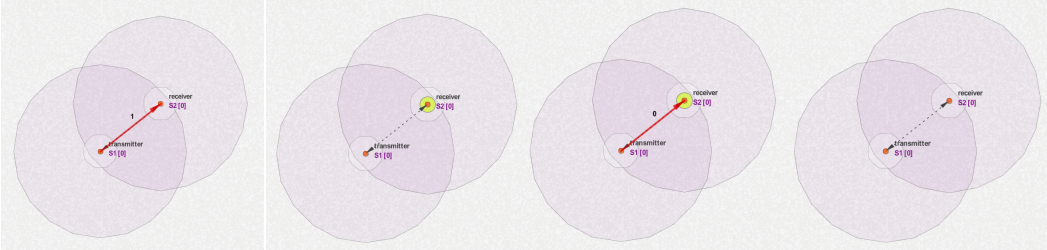

```
loop
receive v
mark v
```

Step 5. Assign the SenScript file to the sensor nodes: Select the sensor node 1 on the map (1). Go to Device Parameters in the left part of the main window (2). Then, select the transmitter.csc file in the field Script file (3). And then, click on the apply button just in the right (4). Do the same procedure for the second sensor node by choosing the SenScript file receiver.csc. After doing this, the center of each sensor node will be colored in orange and the name of the assigned SenScript file will be displayed on the sensor node in gray color.



Step 6. Configure the simulation parameters: To visualize the result of the simulation, in this example one assign to the simulation speed the value 500 ms (1/2 second) and to the arrow speed the value 1000 ms (1 second). The arrows correspond to the send messages.

Step 7. Run the simulation: Just click on Run Simulation button



The red arrow shows the sent messages. The value in the middle of the arrow represents the sent message. If the transmitter is sending A and B instead of 1 and 0, then the scripts must be rewritten as follows:

Transmitter 2:

```
loop
send "A" 2
delay 1000
send "B" 2
delay 1000
```

Receiver 2:

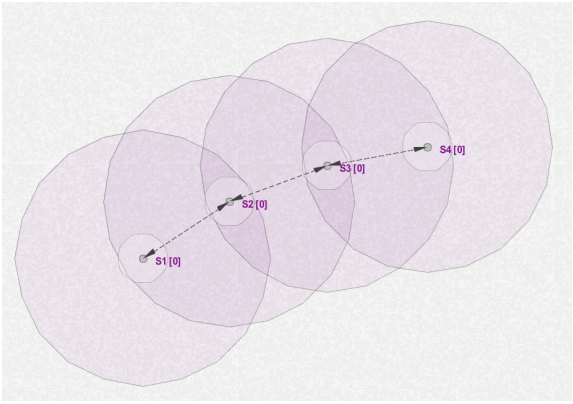
```
loop
receive v
if(v=="A")
    mark 1
else
    mark 0
end
```

To stop manually the simulation, just click on the button .

Remark: Be sure that the id of the transmitter is 1 and the one of the receiver is 2. It is possible to change the value 2 by * for the broadcast sending (send "A" *).

Example 8: Routing messages

The previous Example 7 can be completed by adding routers between the transmitter and the receiver. Let create 4 sensor nodes as follows.



In the same way as in the Example 7, we will create the same SenScript codes for the transmitter (sensor node 1) and the receiver (here, the sensor node 4). We will add two routers, the first one, the sensor node 2, will route the received messages to the sensor node 3, and the second router is the sensor node 3, which will route the received messages to the sensor node 4 (the receiver). The codes of the routers are given as follows:

Router 1 (routing to sensor node 3)

```
loop
receive v
send v 3
```

Router 2 (routing to sensor node 4)

```
loop
receive v
send v 4
```

Simulate …

Using this method requires to create a new script for each router. It is possible to use another method based on the command `send m * x` which allows to send the message **m** to all the neighbors except the neighbor **x**. The advantage of this command is to use the same script file for all the routers. Then, the scripts of the previous example can be rewritten as follows:

The transmitter:

```
atget id id          → id = the identifier of the sensor node (in the example id=1)
loop
data p id "A"        → p = "1#A"
send p               → send "1#A" in a broadcast mode
delay 1000           → wait for 1 second
data p id "B"        → p = "1#B"
send p               → send "1#B"
delay 1000           → wait 1 second
```

The router, this script will be assigned to the sensor node 2 and 3:

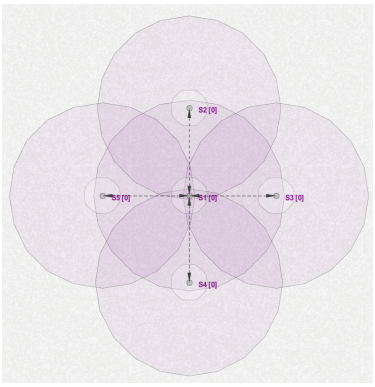
```
atget id id          → id = the identifier of the sensor node (id of the router 2 or 3)
loop
receive rp           → the received message will be assigned to rp (eg. rp = 1#A or 2#A)
rdata rp rid v       → rid=1 and v=A (sensor node 2), rid=2 and v=A (sensor node 3)
data p id v          → p = 2#A
send p * rid         → send 2#A to all the neighbors except the neighbor rid (ie. 1 or 2)
```

The receiver:

```
loop
receive rp          → read the received message and assign it to rp
rdata rp rid v      → rid = 3 and v=A
if(v=="A")          → if v==A
  mark 1            → the sensor node will be marked
else                → otherwise,
  mark 0            → it will be unmarked
end
```

Example 9: Sending Broadcast messages

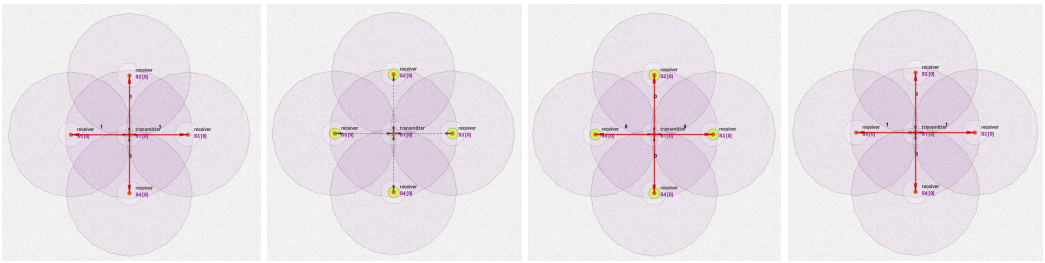
To send a message in a broadcast mode, one needs just to remove the id of the receiver in the **send** command. One can also replace it by *. Let consider the same Example 7 with one transmitter and 4 receivers, as follows:



The codes of the receivers are same. That is to say, one will assign the same SenScript file for each receiver. No need to create different scripts for each sensor node since they have the same script. We need just to remove in the script of the transmitter the id of the receiver. The script will be as follows:

```
loop
send "A" 2
delay 1000
send "B" 2
delay 1000
```

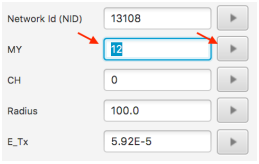
Simulate …



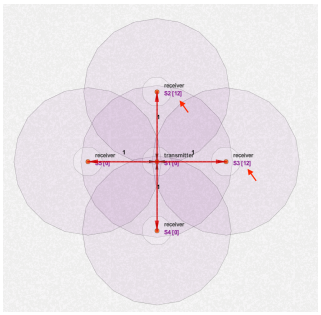
Example 10: Sending messages to a group

This example is the same as the Example 9 with the current changes:

1. Change the MY addresses of the sensor nodes 2 and 3 to the value 12. To do this, select each sensor node and open the Radio Parameters view in the left of the CupCarbon environment. Modify the value of the field MY to 12 and then click on the apply button in the right of this filed.



Once the button apply is pressed, one remark, for the corresponding sensor nodes, that between the brackets situated in the right of the name in the center, the value is changed from 0 to 12. This value represents the MY address.

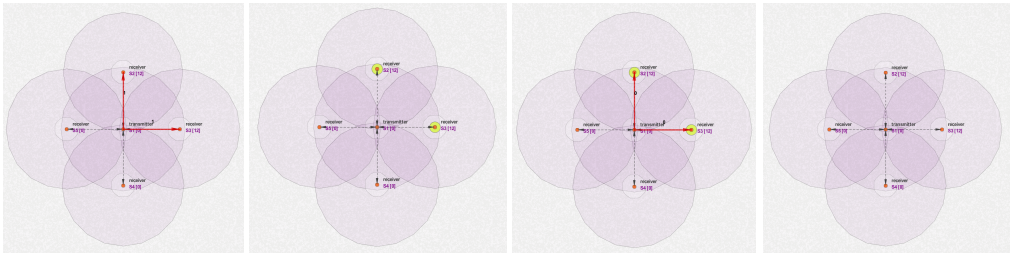


2. To send messages to a group of sensor nodes having the same MY address, we modify slightly the code of the transmitter, of the previous example, as follows:

```
loop
send "A" 0 12
delay 1000
send "B" 0 12
delay 1000
```

The value 0 of this script means that the message will be sent for sensor nodes having the MY address given in the followed number. In this example, this number is equal to 12.

The simulation will lead to the following result:

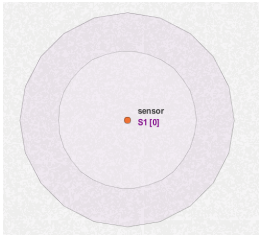


Example 11: Reading digital sensor values

In this example, we will add two sensor nodes and one mobile. The first sensor node will send a message to its neighbor (the second sensor node) about the state of its sensing unit each 100 milliseconds (0 if no

detection and 1 if an event is detected). The second sensor node will print the received message and will be marked if it receives 1 and unmarked if it receives 0. A mobile will be added to pass near to the first sensor node, exactly near to its sensing unit, in order to be detected.

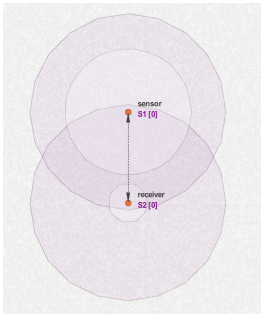
Let first create the first sensor node that sends each 100 milliseconds the value of its sensing unit. Let call its script sensor. Let increase its sensing unit so that to obtain the following result:



Script of the sensor node 1

```
loop
dreadsensor s
send s
delay 100
```

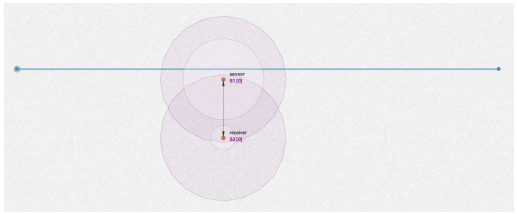
Then, the second sensor node will just await for messages received from the first one and be marked or not according to the received message.

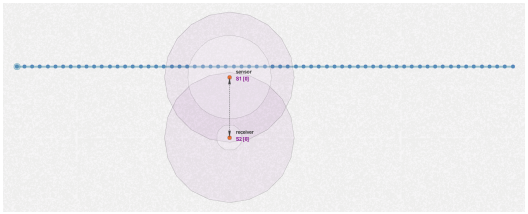


Script of the sensor node 2

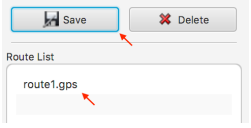
```
loop
receive v
mark v
```

Before adding a mobile, first, we start with adding a route. To do this, we will add two markers and then select both of them. Then we will type many times on the key ‘u’ of the keyboard until to obtain the following result:



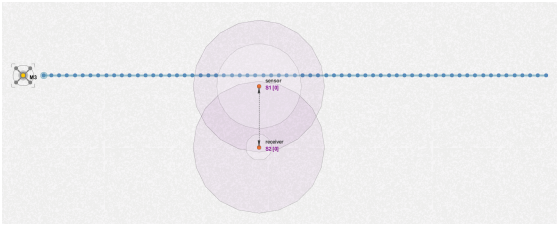


The route must be saved using the button **Save** in the Marker Parameters view.

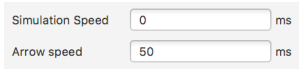


Now, we can add a mobile and assign it the previously created route. This is done using the field **GPS file** in the Device Parameters view.

Once done, the mobile will have a center with the orange color. Then, the project is ready for simulation.



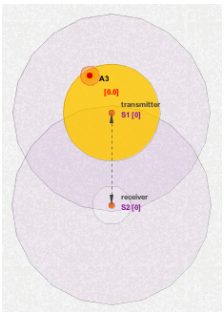
For the simulation, change the value of the simulation speed to 0 ms and the Arrow Speed to 50 ms.

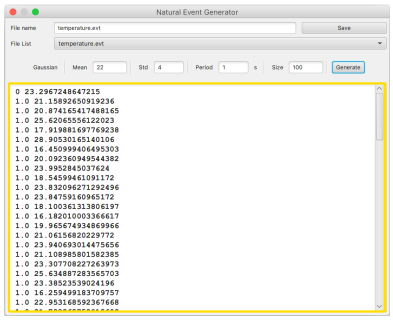


The simulation results must be as follows:

Example 12: Reading analog sensor values

This example is the same as the previous one (Example 11) except that there is no mobility and the mobile is replaced by a natural event (a gas) that will generate random values following the Gaussian distribution (mean = 22 and std = 4). The transmitter will read each 100 milliseconds the value of its sensor unit and then send it to the receiver. The receiver will be marked if it receives a value that is greater than 20. The interface will look like the following one:





To create a file with 100 natural events generated each second from a Gaussian distribution (mean=22 and std=4), we will use the Natural Event Generator. The script of the transmitter and the receiver are given as follows:

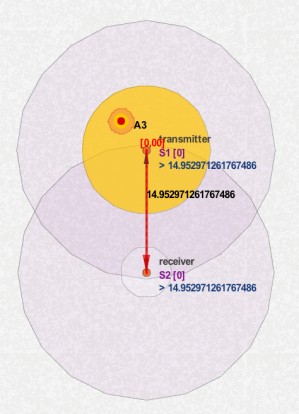
Transmitter:

```
loop
areadsensor v
if(v!="X")
    print v
    rdata v a b c
    send c 2
end
delay 1000
```

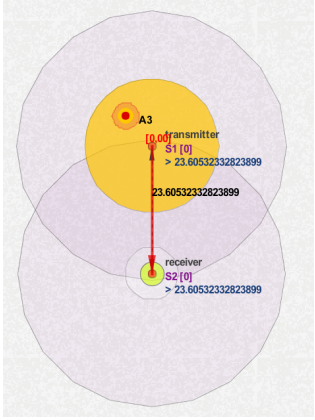
Receiver:

```
loop
receive y
print y
if(y>20)
    mark 1
else
    mark 0
end
```

The simulation results will show something like the following figures where the sensor node is unmarked after receiving a value less than 20 and it is marker after receiving the value 23 which is greater than 20.



Unmarked sensor node after receiving the value 14<20



Marked sensor node after receiving the value 23>20

It is also possible to write the received values in a file during the simulation using the command **printfile** as follows:

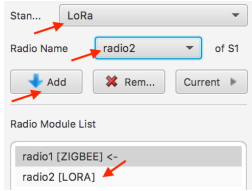
```
loop
receive y
print y
time t          → get the current simulation time
printfile t y   → print in a file the current simulation time and the received value x
if(y>20)
    mark 1
else
    mark 0
end
```

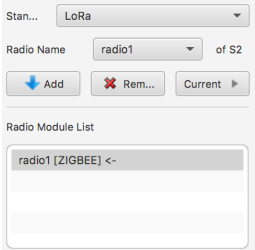
The obtained file has the same name as the executing sensor node and it is located in the directory results.

Example 13: Using many radio modules and standards

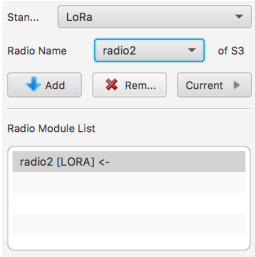
This example shows how to used many radio modules in the same sensor node and switch between them. As an example, we will consider 3 sensor nodes. One transmitter with two radio modules (ZigBee and WiFi), and two receivers having each one of them one radio module. The standard of the first one is ZigBee and the standard of the second one is WiFi. The transmitter will send 1 and 0 each second using its first radio module and will switch to the second radio module while sending 1 and 0 each second. The scenario is created as follows:

1. Create a new project,
2. Add a new sensor node,
3. Open the radio parameters view: we can see that one radio module with the standard ZigBee is already added automatically,
4. Add a second radio module by changing the standard to **LoRa** and the name to **radio2** and then by clicking on **Add** button. In the radio module list we will find two radio modules radio1 [ZIGBEE] and radio2 [LoRa]. It is possible to change the current radio module to the second one (LoRa) by selecting it and by clicking on the button **Current**. It is not possible to delete all the radio modules as well as the current radio module.
5. Add another sensor node without making any changes.

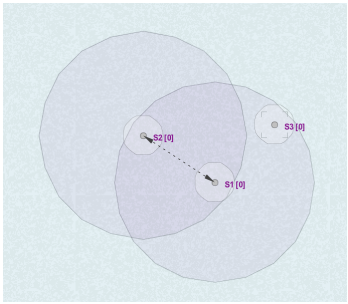


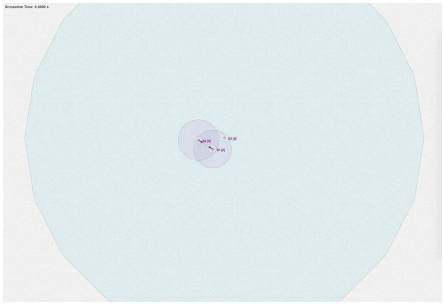


6. Add a third sensor node by adding a radio module LoRa, choose it as the current radio module and then delete the first (ZigBee) radio module.



7. The scene will look like in the following Figure:





8. Add the following scripts for each sensor node:
   a. Sensor Node S1 (with two radio modules):

```
loop
radio "radio1"
send 1
delay 1000
send 0
delay 1000
radio "radio2"
send 1
delay 1000
send 0
delay 1000
```

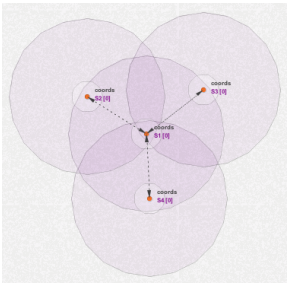
   b. The same script for the other sensor odes (S2 and S3):

```
loop
receive v
mark v
```

9. Simulate …

Example 14: My coordinates and my neighbors

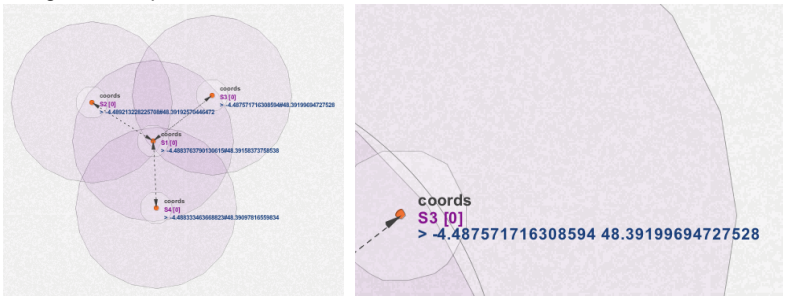
In this example, we will add 3 sensor nodes as follows and we will display they GPS coordinates (longitude and latitude).



1. **Display the Coordinates of each sensor node:** the script that allows to display the coordinates of a sensor node is given by:

```
loop
getpos x
print x
stop
```

Assign this script to each sensor node and simulate. The result will be as follows:



To recuperate each coordinate separately, use the command **rdata**. One can use also the command getpos2 that allows to recuperate the coordinates into 2 separated variable as follows:

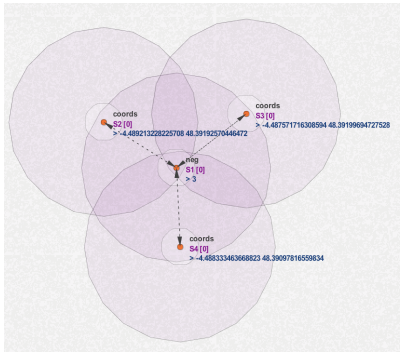
```
loop
getpos2 x y
print x y
stop
```

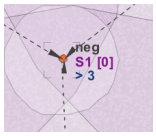
2. **Obtaining the list of the neighbors:** To obtain the list of the neighbors of the sensor S1 for example, we can use two commands. The first one is **atnd x** and the second one is **atnd x y**. The first command allows to assign to x the number of neighbors and the second one allows to assign to x the number of neighbors and to y the list of the identifiers of the neighbors separated by #. In the following we present two scripts using each of them each of these commands and the results of their simulation.

Script 1 (**atnd x**): this will display the number of the neighbors of S1.

```
loop
atnd x
print x
stop
```

Simulation result:

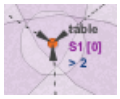




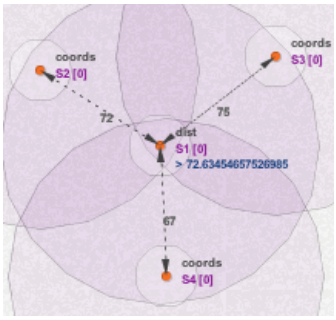
Script 2 (**atnd x y**): this will display each second the identifier of each neighbor (for good visualization, you must set the value of the simulation speed to 1000):

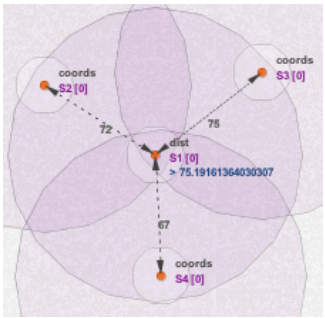
```
loop
atnd n v
for i 0 n
    vget x v i
    print x
    delay 1000
end
stop
```

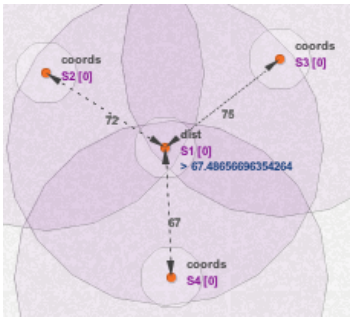
Simulation result:



In the following, we will display each second the distance with each neighbor:







Example 15: Working with radio parameters

In this example, we will use the same procedure as Example 1. Instead of displaying Hellow World we will display each second:

1. The Network ID
2. The Channel
3. The MY address

```
loop
atget nid v1
atget ch v2
atget my v3
print v1
delay 1000
print v2
delay 1000
print v3
stop
```

Change the simulation speed to 1000 and simulate …

Example 16: Transmission power

In the following, in order to show how to change the power of transmitted signal, let consider the following example with three sensor nodes:

The first sensor node will send A and B each second and the other sensor nodes will display the received message (i.e., A and B). After each transmission of the couple A and B we will change the percentage of the power of the sending message. In the beginning we will put 100% and then we switch to 60%. The script of this situation is written as follows:

The script of the transmitter:

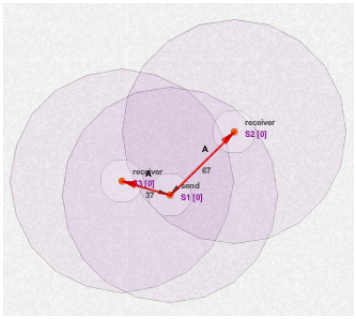
```
loop
atpl 100
send "A"
delay 1000
send "B"
delay 1000
atpl 60
send "A"
delay 1000
send "B"
delay 1000
```

The script of the receiver:

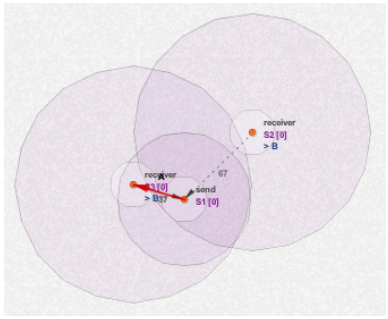
```
loop
receive x
print x
```

Simulate with the following simulation parameters:

The results are close to:



100%



60%

Example 17: Interferences and Acknowledgments

Use Example 8 with the following scripts:

Router 1 (routing to sensor node 3)

```
loop
receive v
send v 3
```

Router 2 (routing to sensor node 4)

```
loop
receive v
send v 4
```

Before simulating, activate the ACK and the Show boxes. Run simulation …

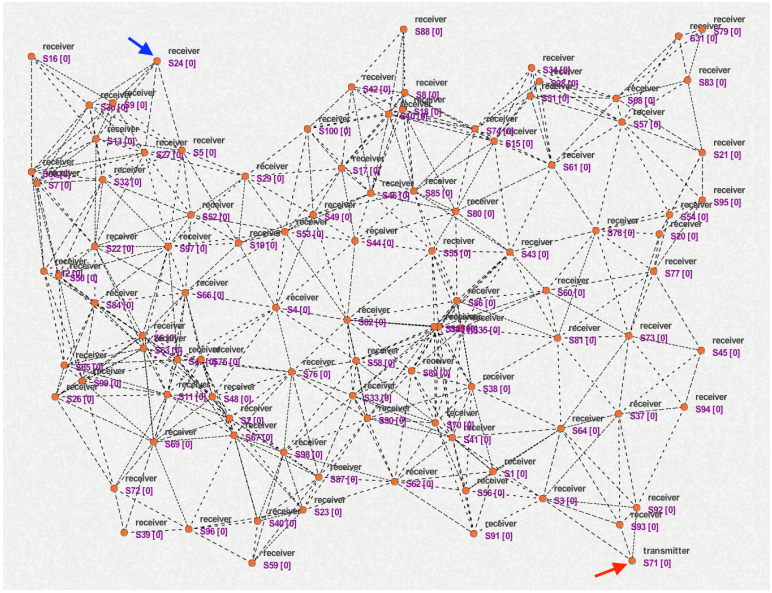
Note that there are no acknowledgment messages when sending in a broadcast mode (i.e., in the example, there is no ACK message for the message sent by S1).

Advanced Examples

Tutorial 1: Send me your coordinates please

In this example a specified sensor node will send a message to ask a specific sensor node in the network for its coordinates. We will see, in this example, that there is no need of neighboring detection process and saving the table of the neighbors locally. This is just an example and not a recommended routing protocol. Because, in a real network, doing this process for each action (asking for coordinates) will be, indeed, very energy consuming. Note that the CupCarbon simulation and the SenScript is executed at the application level.

Create a new project and add 100 sensor nodes randomly.



To obtain this visualization, click on the button **Connections** in the state bar. Let take any sensor node, for example S71, the starting node situated in the extreme bottom right of the network, as shown in red arrow in the Figure above. This sensor node will ask for the coordinate of the sensor node S24 situated in the top left of the network, as shown by the blue arrow of the figure above. These ids (24 and 71) must be adapted and changed in your example. Then, replace these values in your script by the ones of your example.

First, we will start with a script that allows just to detect and mark the concerned sensor node (i.e., S24). To do this, S71 will send in a broadcast the message formed by A and 24 which means “I am searching for the sensor node having the id 24”. The other sensor nodes, which will receive this message, will do the same thing **once** if they are not the sensor node S24. Otherwise, they will be marked.

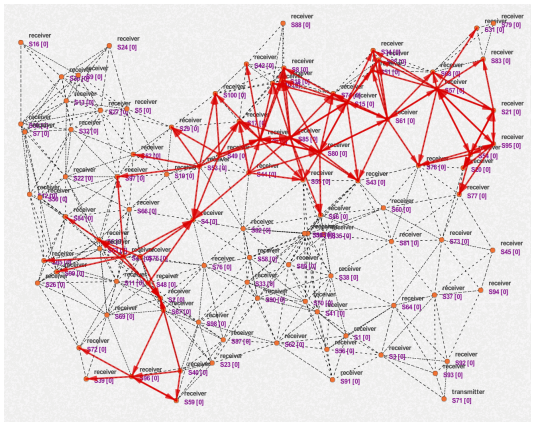
The script of the starting node (S71) is given as follows:

```
atget id id
data d id "A" 24
send d
loop
stop
```

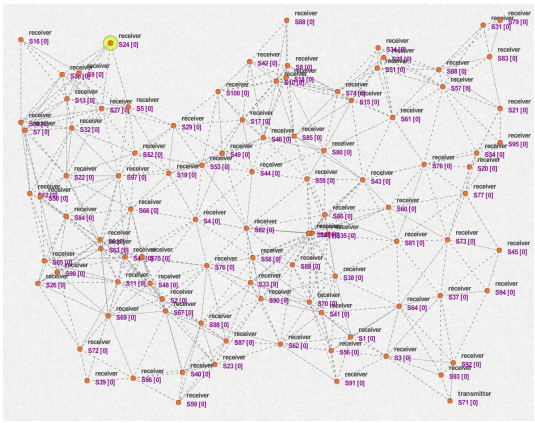
The script of the receivers (other sensor nodes) is given as follows:

```
atget id id
set recA 0
loop
receive m
rdata m rid type info
if((type=="A") && (recA==0))
    set recA 1
    if(info==id)
        mark 1
    else
        data d id "A" info
        send d
    end
end
```

The variable id represents the current id of the sensor. The variable recA means “already received the message A” in order to send the message A once or to be marked once if its id is the one which is researched (here, 24). The info represents here 24, which is the researched id. We call it info because in the following next step this info will become the coordinates of the sensor node 24 which will be sent to the sensor node S71. Simulate with Simulation speed = 0 and Simulation Arrow speed = 500. The simulation will give the following result:



During simulation



End of simulation

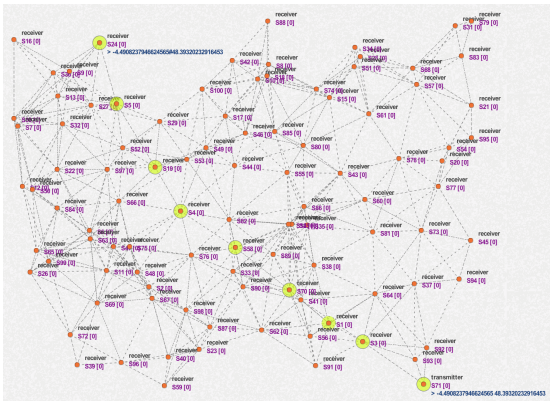
Now, the marked sensor node will display its coordinates and will send them as a message B to the previous sensor node (here, S5), which sent to it the message A. This previous sensor node (S5) will send again the B to its previous sensor node, and so on until reaching the starting sensor node S71. This last one will be marked and display the received coordinates! The previous scripts will be completed as follows. The script of the starting node (S71) will be as follows:

```
atget id id
data d id "A" 24
send d
loop
receive m
rdata m rid type x y
if (type=="B")
    mark 1
    print x " " y
    stop
end
```

The script of the other sensor nodes will be as follows:

```
getpos p
atget id id
set recA 0
loop
receive m
rdata m rid type info info2
if((type=="A") && (recA==0))
    edge 1 rid
    set recA 1
    if(info==id)
        mark 1
        print p
        data d id "B" p
        send d rid
    else
        set prev rid
        data d id "A" info
        send d
    end
end
if(type=="B")
    mark 1
    data d id "B" info info2
    send d prev
end
```

The simulation will give the following result:



Simulate with Simulation speed = 0 and Simulation Arrow speed = 500.

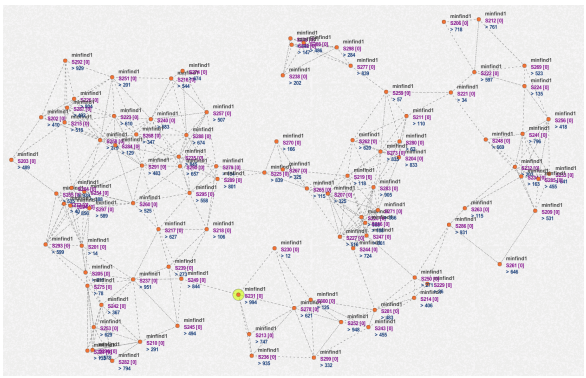
Tutorial 2: Find the extreme left node

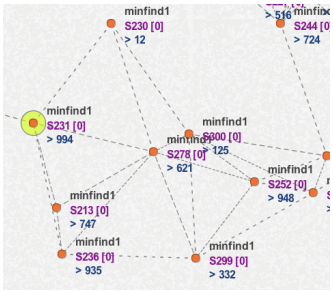
This example is inspired from the example of finding the leader in the network. First, each sensor node will generate a random value between 0 and 1000 and then marks the sensor node having the maximum generated value.

The script of all the sensor nodes is the same and it is given as follows:

```
randb x 0 1000
print x
mark 1
set vmax x
send vmax
loop
receive v
if (v > vmax)
    mark 0
    set vmax v
    send v
end
delay 1
```

The result of the simulation can be as follows. Test with Simulation Speed=0 and for Arrows Speed =0, 100 and 500. If there are many sensor nodes having the same maximum then they will all be marked.

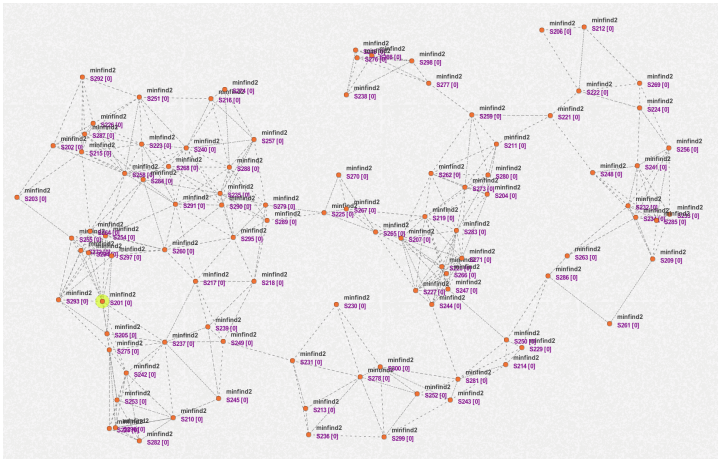




This algorithm can be used also to find the sensor node having the minimum id. In this case, only one sensor node will be marked. The script is almost like the previous one, only the variable **x** will be replaced by the id of the sensor node instead of a random value, and we don't need to print it as it represents the id itself, which is already displayed on the map for each sensor node. Also, as we are searching for the min instead of the max, we will change the if condition to **<**.

```
atget id vmin
mark 1
send vmin
loop
receive v
if (v < vmin)
    mark 0
    set vmin v
    send v
end
delay 1
```

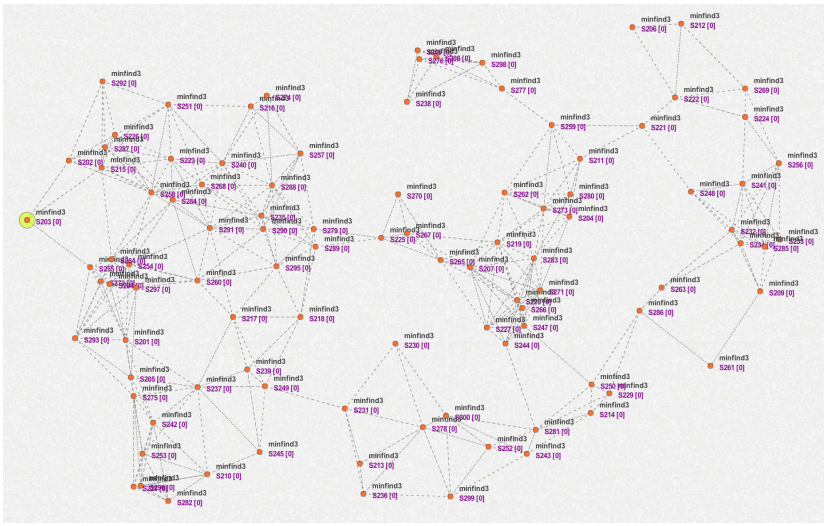
The simulation result will give (here the id max is 201):



This same script can be used now to find the sensor node, which is in the extreme left of the network. It is simply, the sensor node having the minimum x-coordinate.

```
getx vmin
mark 1
send vmin
loop
receive v
if (v < vmin)
    mark 0
    set vmin v
    send v
end
delay 1
```

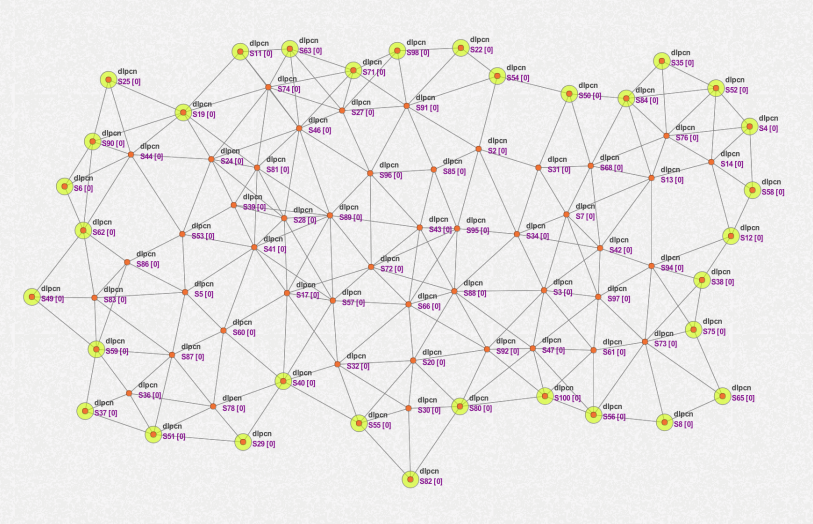
The simulation result gives:



Tutorial 3: Simulate the D-LPCN algorithm (version 1)

The D-LPCN algorithm allows to find the boundary sensor nodes of a network. It starts from any boundary sensor node (for example, the one in the extreme left). Then, each node calculates the angles formed by the previously found boundary node and each one of its neighbors; and then it chooses the node that forms the smallest angle as the next boundary node. To write this algorithm, we will use 3 types of messages. A message *AC* to ask the neighbors to send their coordinates, a message *CS* for sending the coordinates and a message *SN* to inform a sensor that it is a boundary node. The script of the DLPCN algorithm is given as bellow. First, (lines 1 to 3), each sensor node will assign to the variable *cid* its current *id* and it calculates its current coordinates *cx* and *cy*. Since this script is executed by all the sensor nodes of the network, we will add the variable *first* (here it is equal to 49, which represents the extreme left sensor node) to designate the starting node. Some parts of the script will be executed only by the first node. Then, (lines 5 to 10), only the first node will execute this part. It will execute it once. That is why in line 6, the variable *first* is set to 0. To calculate the angles formed by the previous boundary node and its neighbors. In lines 5 to 8, we will consider that there is a virtual sensor node, which is situated in the left part of the starting node, which have the coordinates (*px=cx-1* and *py=cy*). Then, a message *SN* will be considered since the starting node is also a boundary node. The line 10 (*SN* message) will be followed directly by line 22. The block 23 to 26 represents the stop condition, which represents visiting twice the starting node. In the first time, the variable *first* is equal to 0, then the stop condition is not verified, and the variable *first* will be set to -1 (line 26). However, if we come back to this block a second time, the condition *(first == –1)* is verified, then the *stop* command will be called. If the stop condition is not verified, then, as the considered block is executed after receiving an *SN* message, which means that the receiver is a boundary sensor node. The line 27 will mark this sensor, and then, from line 28 to 45 the process of asking and receiving the coordinates of the

neighbors will be started. For each received message (coordinates) the angle will be calculated and compared with the previously calculated angle in order to determine the smallest one. Line 28 allows to get the value of the coordinates *px* and *py* of the previous boundary node. Line 29 forms a message with an angle equal to 10 radian. Line 30 is used to get the number of neighbors and their identifiers. Lines 31 to 34 are used to send to each neighbor node an *AC* message to ask it to send its coordinates. Lines 35 to 45 are used to wait for the answer of the neighbors. If a *CS* message is received, then the coordinates are sent and can be used to calculate the angle (line 40) in order to select the minimum one (line 42). The function *smin* allows to find the minimum value between two values extracted from messages formed by the *id* of a sensor node and its corresponding variable. Once the minimum angle is found as well as its corresponding sensor node (*id*), then a message *SN* will be sent (lines 46 and 47) to inform the obtained sensor node that it is a boundary sensor node. Note that, the lines 17 to 20 are executed by the sensor nodes that receive the message *AC* in order to send their coordinates (with message *CS*: line 18). For simplicity, we consider as stop condition when the first starting node is visited twice, which is not the real stop condition. For more information about this algorithm, please refer to the paper [1]. The execution of the script given bellow will result to the following situation:



```
1:  atget id cid
2:  getpos2 cx cy
3:  set first 49
4:  loop
5:  if (cid==first)
6:     set first 0
7:     set px cx-1
8:     set py cy
9:     data p 0 "SN" px py
10:    set type "SN"
11: else
12:
13:    receive p
14:    rdata p id type
15: end
16:
17: if (type=="AC")
18:    data p cid "CS" cx cy
19:    send p id
20: end
21:
22: if (type=="SN")
23:    if (first==-1)
24:       stop
25:    end
26:    set first -1
```

```
27:     mark 1
28:     rdata p id type px py
29:     data m cid 10
30:     atnd n tneg
31:     for i 0 n
32:         vget neg tneg i
33:         data p cid "AC"
34:         send p neg
35:
36:         receive p
37:         rdata p id type
38:         if (type=="CS")
39:             rdata p id type x y
40:             angle2 a px py cx cy x y
41:             data p id a
42:             smin m m p
43:             rdata m id a
44:         end
45:     end
46:     data p cid "SN" cx cy
47:     send p id
48: end
```

This script can be written as follows, where we consider broadcast messages to the neighbors.

```
1  : atget id cid
2  : getpos2 cx cy
3  : set first 49
4  : loop
5  : if (cid==first)
6  :    set first 0
7  :    set px cx-1
8  :    set py cy
9  :    data p 0 "SN" px py
10 :    set type "SN"
11 : else
12 :
13 :    receive p
14 :    rdata p id type
15 : end
16 :
17 : if (type=="AC")
18 :    data p cid "CS" cx cy
19 :    send p id
20 : end
21 : if (type=="CS")
22 :    rdata p id type x y
23 :    angle2 a px py cx cy x y
24 :    data p id a
25 :    smin m m p
26 :    rdata m id a
27 :    inc i
28 :    if(i==n)
29 :        data p cid "SN" cx cy
30 :        send p id
31 :    end
32 : end
33 : if (type=="SN")
34 :    if(first==-1)
35 :        stop
36 :    end
37 :    set first -1
38 :    rdata p id type px py
39 :    mark 1
40 :    data m cid 10
41 :    atnd n
```

```
42 :    set i 0
43 :    data p cid "AC"
44 :    send p
45 : end
```

Tutorial 4: Simulate the D-LPCN algorithm (version 2)

In this example we will show how we can merge many SenScripts in one script. For example, the script given previously in Example 3 starts from a given sensor node. It is the sensor node that is in the extreme left of the network. Example 2 shows another script allowing to find this extreme left sensor node. Therefore, in the following, we will show how to write a code that, first, finds the extreme left sensor node, using the script of Example 2, and then starts the D-LPCN algorithm form that one, using Example 3. To do this, we will create a new variable ***step***. This variable will be equal to 1 and then to 2 in order to determine which algorithm will be executed. Let call the code of Example 2 by **Code1** and the one of Example 3 by **Code2**. Then the final script will have almost the following structure:

```
set step 1
if(step == 1)
    if (condition)
         set step 2
    end
    Code1
end
if(step == 2)
    Code2
end
```

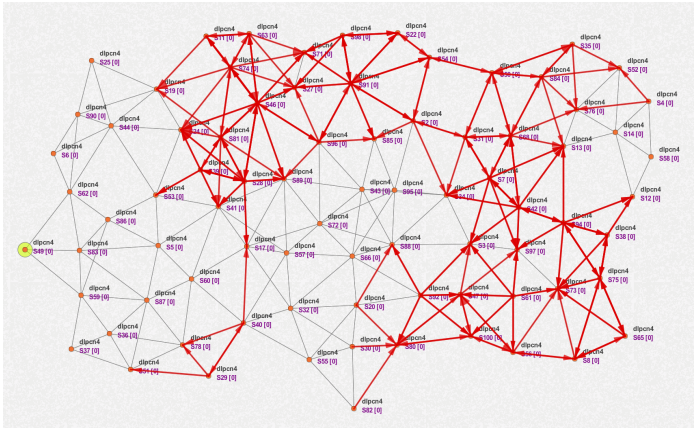
Code1 will allows to determine the sensor node which is in the extreme left. The identifier of this sensor node will be the value of the variable ***first*** which is used in Code2 for the starting node. In this new situation, in the script that finds the extreme left sensor node, the ***wait*** command will lead to a blocking situation. No sensor node can continue execution. Then, it is not possible to go to the second step. To overcome this limitation, we will use a command ***wait t***, which will wait for receiving messages during a certain time ***t*** and continue the execution after this time. In our case, this time represents the required time to finish the first process of finding the extreme left node. If after this time there are no read messages (empty message), then we start the second step of determining the boundary nodes using the D-LPCN algorithm.

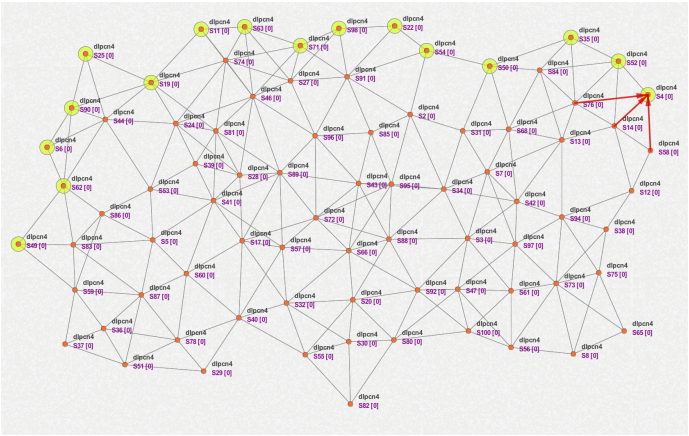
The final script is given as follows:

```
1  : set step 1
2  : set first -2
3  : getpos2 vmin y
4  : set marked 1
5  : send vmin
6  : loop
7  : if(step == 1)
8  :     delay 1
9  :     receive v 2000
10 :     if(v == "")
11 :         atget id cid
12 :         getpos2 cx cy
13 :         set step 2
14 :         if (marked == 1)
15 :             set first cid
16 :         end
17 :     else
18 :         if (v < vmin)
19 :             set marked 0
20 :             set vmin v
21 :             send v
22 :         end
23 :     end
24 : end
25 : if(step == 2)
26 :     if (cid==first)
27 :         set first 0
```

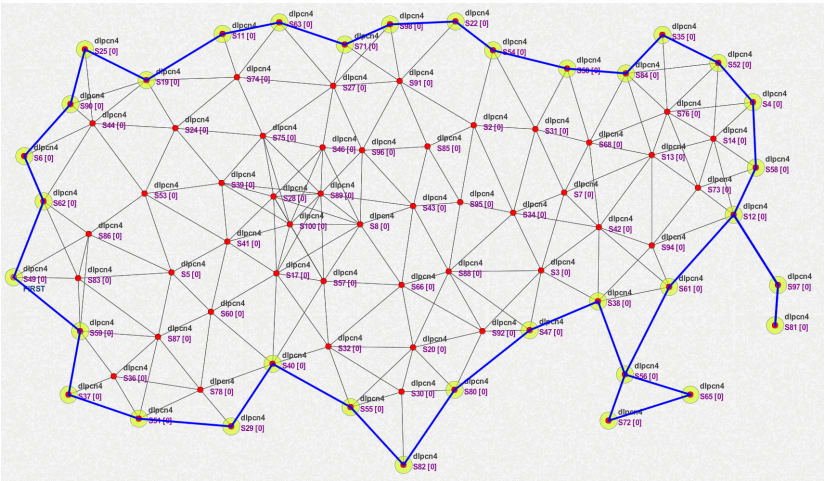
```
28 :         set px cx-1
29 :         set py cy
30 :         data p 0 "SN" px py
31 :         set type "SN"
32 :     else
33 :
34 :         receive p
35 :         rdata p id type
36 :     end
37 :     if (type=="AC")
38 :         data p cid "CS" cx cy
39 :         send p id
40 :     end
41 :     if (type=="CS")
42 :         rdata p id type x y
43 :         angle2 a px py cx cy x y
44 :         data p id a
45 :         smin m m p
46 :         rdata m id a
47 :         inc i
48 :         if(i==n)
49 :             data p cid "SN" cx cy
50 :             send p id
51 :         end
52 :     end
53 :     if (type=="SN")
54 :         if(first==-1)
55 :             stop
56 :         end
57 :         if(first==0)
58 :             set first -1
59 :         end
60 :         set first -1
61 :         rdata p id type px py
62 :         mark 1
63 :         data m cid 10
64 :         atnd n
65 :         set i 0
66 :         data p cid "AC"
67 :         send p
68 :     end
69 : end
```

Simulate with Simulation speed = 0 and Arrow Speed = 100. For the visualization, since the simulation speed is more quick for the first step than for the second one, it is recommended to change the arrow speed parameter for example from 5 for step 1 to 200 for step2. This procedure is difficult to do manually. That is why we recommend to use the command ***simulation aspeed*** to force different values for each step.





If we add the instruction **`edge 1 id`** between the lines 48 and 49 of the last script, we get the following result (with marked boundary edges):



Tutorial 5: Simulate the D-LPCN algorithm (version 3)

In this tutorial we will use the command **script**. First, we can consider the script of the D-LPCN algorithm without specifying the value of the variable **first** representing the id of the starting node. This script must be saved with the name dlpcn.csc. To call it from another script, we use the command **script dlpcn**.

```
1  : atget id cid
2  : getpos2 cx cy
3  : set first 49
4  : loop
5  : if (cid==first)
6  :    set first 0
7  :    set px cx-1
8  :    set py cy
9  :    data p 0 "SN" px py
10 :    set type "SN"
11 : else
12 :
13 :    receive p
14 :    rdata p id type
15 : end
16 :
17 : if (type=="AC")
18 :    data p cid "CS" cx cy
19 :    send p id
20 : end
21 : if (type=="CS")
22 :    rdata p id type x y
23 :    angle2 a px py cx cy x y
24 :    data p id a
25 :    smin m m p
26 :    rdata m id a
27 :    inc i
28 :    if(i==n)
29 :        data p cid "SN" cx cy
30 :        send p id
31 :    end
32 : end
33 : if (type=="SN")
34 :    if(first==-1)
35 :        stop
36 :    end
37 :    if(first==-1)
38 :        set first -1
39 :    end
40 :    rdata p id type px py
41 :    mark 1
42 :    data m cid 10
43 :    atnd n
44 :    set i 0
45 :    data p cid "AC"
46 :    send p
47 : end
```

Then, in the MinFind algorithm, we will call the previous script LPCN once finished using the command script (line 16). Note, that we will add a delay time once the first node is found (line 12). This will allow to wait for the other nodes to finish their MinFind algorithm and to be ready to start the LPCN algorithm. The MinFind algorithm can be rewritten in this case as follows. In this tutorial, only the MinFind algorithm is assigned to the sensor nodes. The LPCN algorithm will be called automatically from this one (line 16).

```
1  : atget id cid
2  : getpos2 vmin y
3  : set marked 1
4  : send vmin
5  : loop
```

```
6  : mark marked
7  :
8  : receive v 2000
9  : if(v == "")
10 :    if (marked == 1)
11 :        set first cid
12 :        delay 1000
13 :    else
14 :        set first -3
15 :    end
16 :    script "dlpcn"
17 : else
18 :    if (v < vmin)
19 :        set marked 0
20 :        set vmin v
21 :        send v
22 :    end
23 : end
24 : delay 1
```

References:

[1] Massinissa Saoudi, Ahcène Bounceur, Farid Lalem, Reinhardt Euler, M-Tahar Kechadi, Abdelkader Laouid, Madani Bezoui, Marc Sevaux, D-LPCN: A Distributed Least Polar-angle Connected Node Algorithm for Finding the Boundary of a Wireless Sensor Network, Ad Hoc Networks Journal, Elsevier, Volume 56, 1 March 2017, Pages 56–71, DOI: 10.1016/j.adhoc.2016.11.010.

[2] Taha Alwajeeh, Pierre Combeau, Rodolphe Vauzelle and Ahcène Bounceur, A High-Speed 2.5D Ray-Tracing Propagation Model for Microcellular Systems, Application: Smart Cities, In the 11th IEEE European Conference on Antennas and Propagation (EuCAP 2017), 19-24 March 2017, Paris, France.

[3] Umber Noreen, Ahcène Bounceur and Laurent Clavier, Modeling Interference for Wireless Sensor Network Simulators, In ACM International conference on Big Data and Advanced Wireless technologies (BDAW'2016). Blagoevgrad, Bulgaria, November 10-11, 2016.

[4] Farid Lalem, Ahcene Bounceur, Rahim Kacimi, Reinhardt Euler and Saoudi Massinissa, Faulty Data Detection in Wireless Sensor Networks Based on Copula Theory, In ACM International conference on Big Data and Advanced Wireless technologies (BDAW'2016). Blagoevgrad, Bulgaria, November 10-11, 2016.

[5] Umber Noreen, Ahcène Bounceur, Laurent Clavier, Rahim Kacimi, Performance Evaluation of IEEE 802.15.4 PHY with Impulsive Network Interference in CupCarbon Simulator, Invited paper, in the 3rd IEEE International Symposium on Networks, Computers and Communications (ISNCC), Hammamet, Tunisia, May 11-13, 2016.

[6] Farid Lalem, Rahim Kacimi, Ahcène Bounceur, Reinhardt Euler, Boundary Node Failure Detection in Wire-less Sensor Networks, The 3rd IEEE International Symposium on Networks, Computers and Communications (ISNCC), Hammamet, Tunisia, May 11-13, 2016.

[7] Umber Noreen, Ahcene Bounceur, Laurent Clavier, Integration of OFDM Based Communication System with Alpha-Stable Interference using CupCarbon Simulator, In the ACM International Conference on Internet of things and Cloud Computing (ICC'2016), University of Cambridge, United Kingdom, March, 22-23, 2016.

[8] Taha Alwajeeh, Pierre Combeau, AhcèneBounceur, Rodolphe Vauzelle, Efficient Method for Associating Radio Propagation Models with Spatial Partitioning for Smart City Applications, In the ACM International Conference on Internet of things and Cloud Computing (ICC'2016), University of Cambridge, United Kingdom, March, 22-23, 2016.

[9] Farid Lalem, Muath Alshaikh, Ahcène Bounceur, Reinhardt Euler, Lamri Laouamer, Laurent Nana, Anca Pascu, Data Authenticity and Integrity in Wireless Sensor Networks Based on a Watermarking Approach, The 29th In-ternational Florida Artificial Intelligence Research Society Conference (FLAIRS), May 16-18, 2016, Key Largo, Florida, USA.

[10] Massinissa Lounis, Ahcène Bounceur, Laga Arezki and Bernard Pottier. GPU-based Parallel Computing of Energy Consumption in Wireless Sensor Networks. European Conference on Networks and Communications (EuCNC), June 2015, Paris, France.

[11] K. Mehdi, M. Lounis, A. Bounceur, and T. Kechadi. Cupcarbon: A multi-agent and discrete event wireless sensor network design and simulation tool. 7th International Conference on Simulation Tools and Techniques (SIMUTools'14), Lisbon, Portugal, March 17-19, 2014.

[12] A. S. Wander, N. Gura, H. Eberle, V. Gupta, S. C. Shantz, Energy analysis of public-key cryptography for wireless sensor networks, in: Third IEEE International Conference on Pervasive Computing and Communications (PerCom), IEEE, 2005, pp. 324–328.

www.cupcarbon.com
contact@cupcarbon.com