

Introduction to IoT

School Year 2023-2024

Valsalice



Course Structure

1	Introduction and Basics
2	Basic Data Types and Operators
3	Control Structures Pt. 1
4	Control Structures Pt. 2
5	Functions and Scope
6	Arrays
10	Advanced String Usage
11	Custom Data Types

7	Introduction to Contiki-NG and nRF52840
8	Sensing and Actuating with Contiki-NG
9	Timers and Concurrency
12	Basic Communication and Networking
13	Advanced Communication and Networking
14	Introduction to Cooja
15	Contiki MAC Layer
16	Reliable Data Transfer Challenge



= Core Topics



= Optional Topics

Open your Virtual Machines

1. Turn on your Laptops
2. Login to Windows using "User"
3. Open the **Virtual Box** program
4. Select the **nRF52840LAB** Virtual Machine & click **Start**
5. Log-in using credentials:

Username: **ubuntu**


Password: **ubuntu**
6. Open **Visual Studio Code** (use the App bar on the left)




Prepare the Coding Environment

- 1 From the Terminal:

```
make setup
```

- 2  valsalice-iot-23 git:(master) make setup
Enter your username:



```
✓ Repository setup complete!
```



If you see any (yellow) errors input the credentials again

- 3 Open the **week14** folder in the terminal
- 4 Right click on the left + **“Open in Integrated terminal”**

Recap: Data Types

C has a number of primitive data types:

int

42

1200

1_200

-3

float

3.14

0.00001

-2.1

char

'A'

'@'

'\n'

bool

true

false

Strings are *NOT* a primitive data type, and have special syntax.

strings

"Hello"

"A"

"I am a full sentence!"

Recap: Variables

A variable is a named container that stores data or values.

```
int x = 42;  
float y = -0.12;  
char w = 'A';  
char z[50] = "Full sentence";
```

Booleans require a custom include statement:

```
#include <stdbool.h>  
bool hello = true;
```

Recap: Boolean Operators

Greater than	>
Greater or equal than	>=
Less than	<
Less or equal than	<=
Equals	==
Not equals	!=
Not	!

Recap: Chaining Comparisons

- **and** (both must be true)

```
true && false
```

```
(5 < 6) && (5 < 10)
```

- **or** (either must be true)

```
true || false
```

```
(5 < 3) || (5 < 10)
```

- **not** (negation)

```
!true
```

```
!(5 < 3)
```


Recap: If-Statement chaining

You can chain multiple conditions with **else if**.

What is the difference between these two snippets of code?

```
int num;
scanf("%d", &num);

if (num < 3) {
    printf("Small number\n");
} else if (num < 10) {
    printf("Medium number\n");
}
```

```
int num;
scanf("%d", &num);

if (num < 3) {
    printf("Small number\n");
}
if (num < 10) {
    printf("Medium number\n");
}
```

Recap: While-Loops

Repeat parts of
your code!

```
int num;  
printf("Input a number greater than 100: ");  
scanf("%d", &num);  
  
while (num <= 100) {  
    printf("Wrong number, try again: ");  
    scanf("%d", &num);  
}  
  
printf("Well done!\n");
```

Recap: For-Loops

Repeat a **specific** amount of times!

```
int x;  
  
for (x = 1; x <= 5; x++) {  
    printf("Hello %d\n", x);  
}
```

```
int x = 0;  
  
while (x < 5) {  
    x += 1;  
    printf("Hello %d\n", x);  
}
```

Recap: Arrays

Modifiable containers for data.

With **variables**:

```
int num1 = 42;
int num2 = 100;
int num3 = 10;

printf("%d\n", num1);
printf("%d\n", num2);
printf("%d\n", num3);
```

With a **list**:

```
int array[] = {42, 100,
10};

for(int i = 0; i < 3; i++)
{
    printf("%d\n",
array[i]);
}
```

Recap: Accessing Array Elements

To access array elements you can use the **[index]** operator.

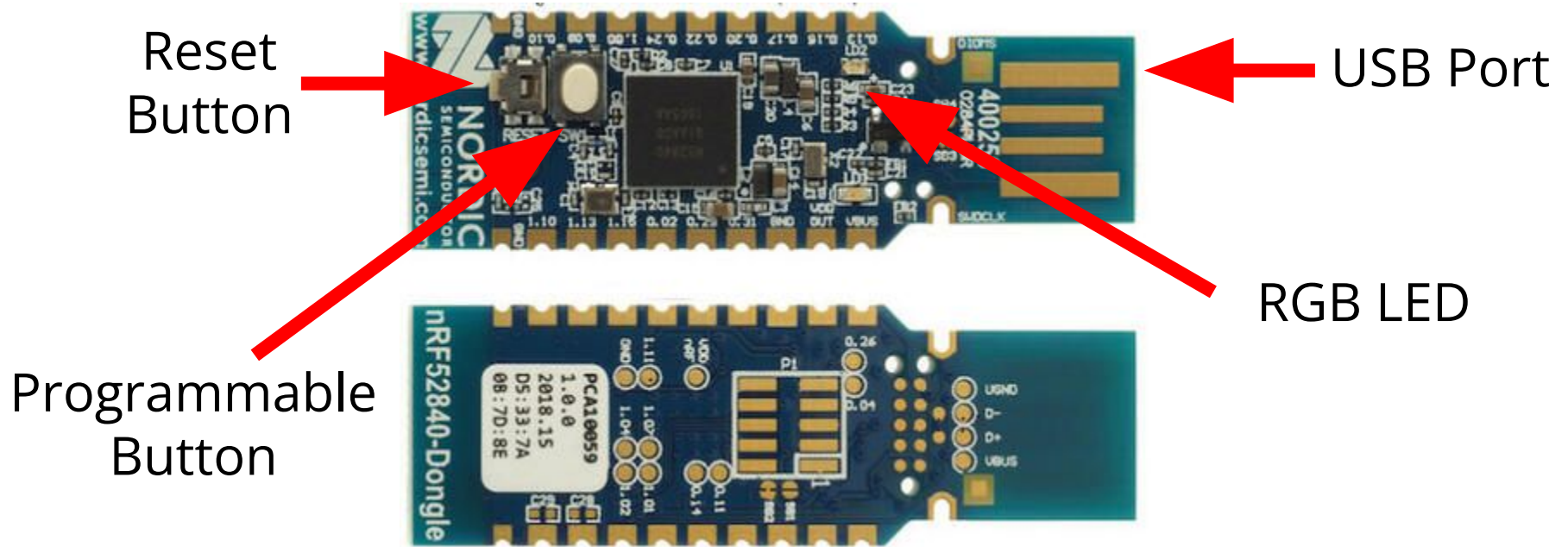
NOTE: List indices start from **0**

index:	0	1	2	3	4
<code>int array[] =</code>	<code>{17,</code>	<code>28,</code>	<code>33,</code>	<code>56,</code>	<code>6};</code>

```
printf("%d\n", array[0]);
```

```
printf("%d\n", array[3]);
```

Recap: What is the nRF52840?



Recap: The LED Library

```
#define RGB_LED_RED      1
#define RGB_LED_GREEN   2
#define RGB_LED_BLUE    4
#define RGB_LED_MAGENTA (RGB_LED_RED | RGB_LED_BLUE)
#define RGB_LED_YELLOW  (RGB_LED_RED | RGB_LED_GREEN)
#define RGB_LED_CYAN    (RGB_LED_GREEN | RGB_LED_BLUE )
#define RGB_LED_WHITE   (RGB_LED_RED | RGB_LED_GREEN | RGB_LED_BLUE)
/*-----*/
void rgb_led_off(void);
void rgb_led_set(uint8_t colour);
```

Recap: The E-Timer Library

```
/* Event generated when a timer expires */
#define PROCESS_EVENT_TIMER          0x88
/*-----*/
/* Set the amount of time on the timer. Also start the timer */
void etimer_set(struct etimer *et, clock_time_t interval);
/* Restart the timer with the previously set amount of time */
void etimer_restart(struct etimer *et);
void etimer_stop(struct etimer *et);
/*-----*/
/* Check if the timer has completed */
bool etimer_expired(struct etimer *et)
```


Recap: Using an E-Timer

```
1 → #define BLINK_INTERVAL (0.2 * CLOCK_SECOND)
    static struct etimer blink_timer;

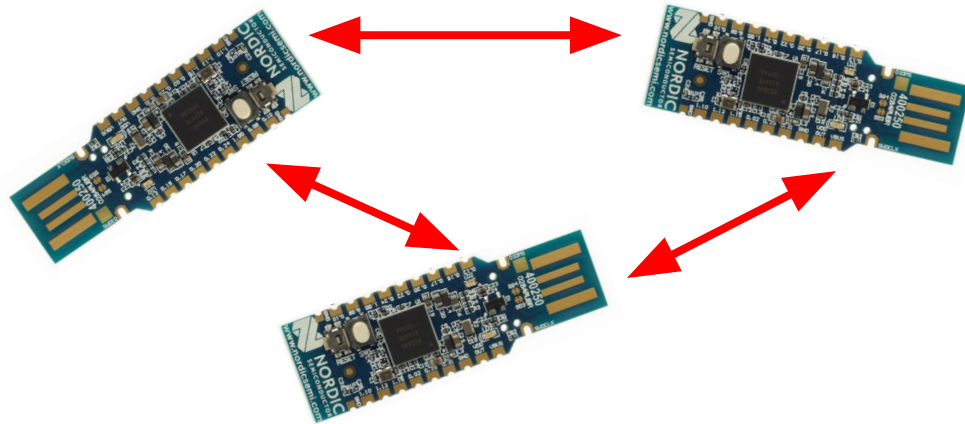
    PROCESS_THREAD(demo_process, ev, data) {
2 →     PROCESS_BEGIN();
        etimer_set(&blink_timer, BLINK_INTERVAL);

        while (true) {
            PROCESS_WAIT_EVENT();

3 →         if (etimer_expired(&blink_timer)) {
4 →             etimer_reset(&blink_timer);
                // Do something on timer expiry
            }
        }
        PROCESS_END();
    }
```

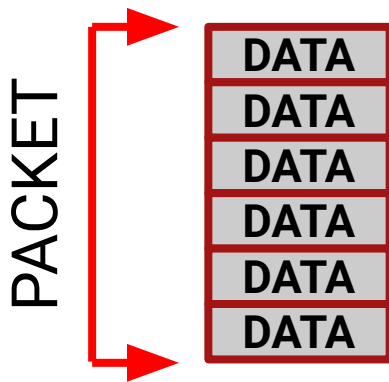
Recap: Networking

- **Networking** is the practice of connecting computers and other devices to **share information**.
- Involves **transmitting data** over various types of media, like wireless signals.



Recap: Networking Packets

- **Packets** are small chunks of **data** sent over the network.
- Packets can have **arbitrary length** (up to a maximum)



Recap: Receiving Packets

We provide a helper function to simplify **receiving packets** over nullnet:

receive_nullnet_data

You can add functionality **inside the function body**.

```
/* Helper function to receive data over nullnet */
void receive_nullnet_data (
    const void *bytes,
    uint16_t len,
    const linkaddr_t *src,
    const linkaddr_t *dest)
{
    int data;
    memcpy(&data, bytes, len);

    printf("Data received: %d\n", data);
}
```

Recap: Disambiguating Packets

To know where packets are coming from we add a **team_id** field to packets.

To use this field in the next exercise you **MUST** set the **TEAM_ID** macro at the top of the file.

```
typedef struct {  
    char team_id;  
    int command;  
    int data;  
} message_t;
```

```
// IMPORTANT!  
// Change the `TEAM_ID`!  
#define TEAM_ID 'Z'
```

Recap: Sending Packets

We provide a helper function to simplify **sending packets** over nullnet:

send_nullnet_data

You can **call** this function but you **should NOT edit** it.

```
/* Helper function to send data over nullnet */  
void send_nullnet_data (int data) {  
    printf("Sending data: %d\n", data);  
    nullnet_buf = (uint8_t *)&data;  
    nullnet_len = sizeof(data);  
  
    NETSTACK_NETWORK.output(NULL);  
}
```

```
send_nullnet_data (200);  
  
variable = 42;  
send_nullnet_data (variable);
```

Recap Exercise

To flash:

```
make counter.dfu-upload
```

For console:

```
make login
```

⚠ Change the **TEAM_ID** at the top of **counter.c**!

Flash the code in (**counter.c**) to the nRF52840.

Use **make login** to connect to serial output.

1) What is the code doing?

2) Where in the code can you find this behavior specified?

Recap Exercise

To flash: `make counter.dfu-upload`
For console: `make login`

Extend the code in **counter.c**



Change the **TEAM_ID** at the top!

- 1) If you receive **command 3**: LED GREEN.
- 2) If you receive **command 4**: LED RED.

3) If you receive **command 0**:
Reply with **data**.

4) If you receive **command 1**:
Reply with **data + 1**.

4) If you receive **command 1**:
Reply with **data - 1**.

```
typedef struct {  
    char team_id;  
    int command;  
    int data;  
} message_t;  
  
void receive_nullnet_data (...) {  
    message_t message;  
    memcpy(&message, bytes, len);  
  
    char team_id = message.team_id;  
    int command = message.command;  
    int data = message.data;  
}
```



Save remotely your Changes

1

```
make save
```

2

```
|Password
```

```
Git: https://aspina@git.spina.me (Press 'Enter' to confirm or  
'Escape' to cancel)
```

3

```
✓ Changes committed and pushed. All done!
```

Cooja

Cooja is a **Simulator** for the Contiki-NG Operating System.

It allows for Contiki-NG programs to be compiled and executed on virtual **simulated test-beds**.

The simulated motes will behave similarly to the real world.

- 1 From the Terminal run the **make cooja** command:

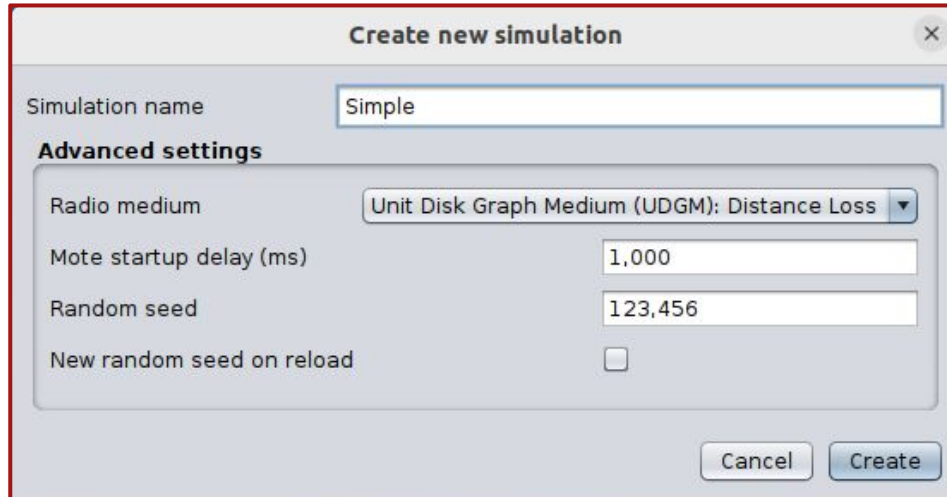
```
make cooja
```

Cooja

2 Create a New Simulation:

File > New Simulation

3 Call the Simulation “Simple”, then click “**Create**”



Simulation name: Simple

Advanced settings

Radio medium: Unit Disk Graph Medium (UDGM): Distance Loss

Mote startup delay (ms): 1,000

Random seed: 123,456

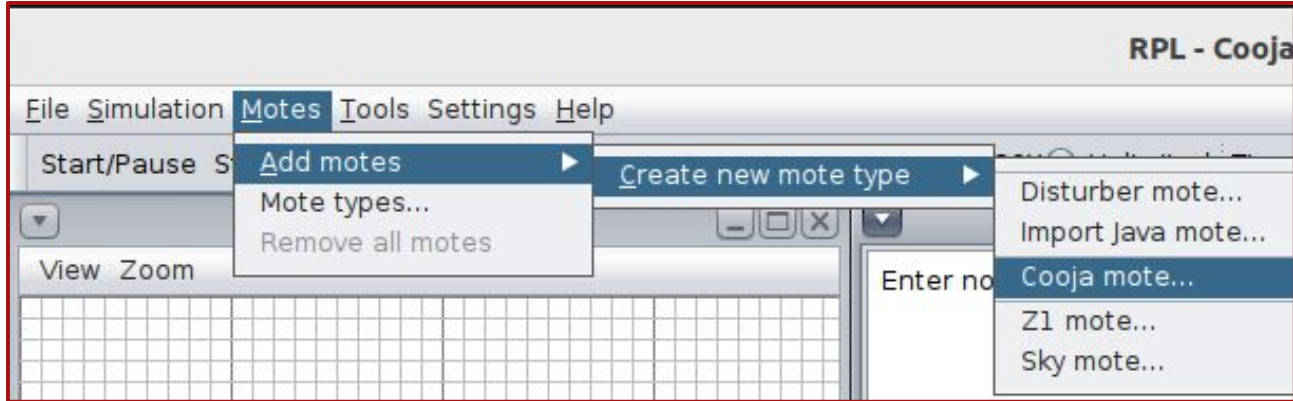
New random seed on reload: ☐

Buttons: Cancel, Create

Cooja

4 Let's add a new Mote:

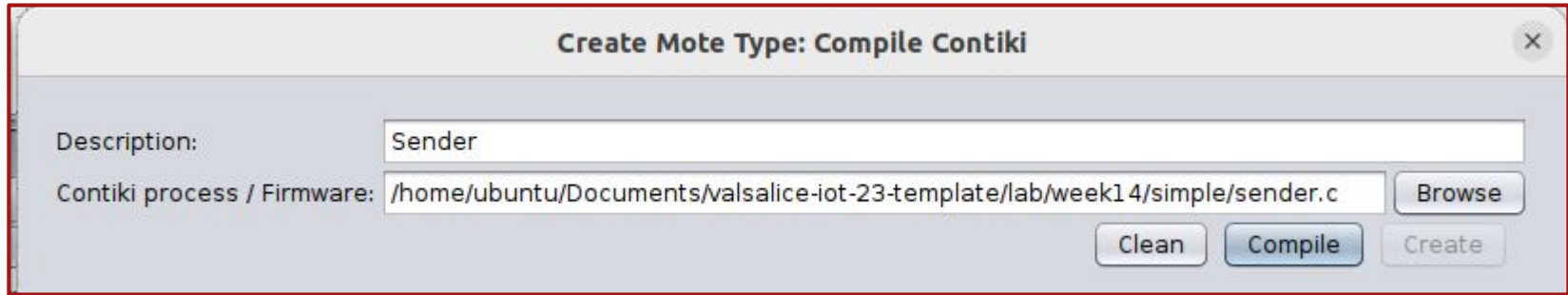
Motes > Add motes > Create new > Cooja mote



Cooja

- 5 Put "**Sender**" in the description
- 6 Use the firmware under (you can also use "Browse"):

`/home/ubuntu/Documents/valsalice-iot-23-template/lab/week14/simple/sender.c`



Create Mote Type: Compile Contiki

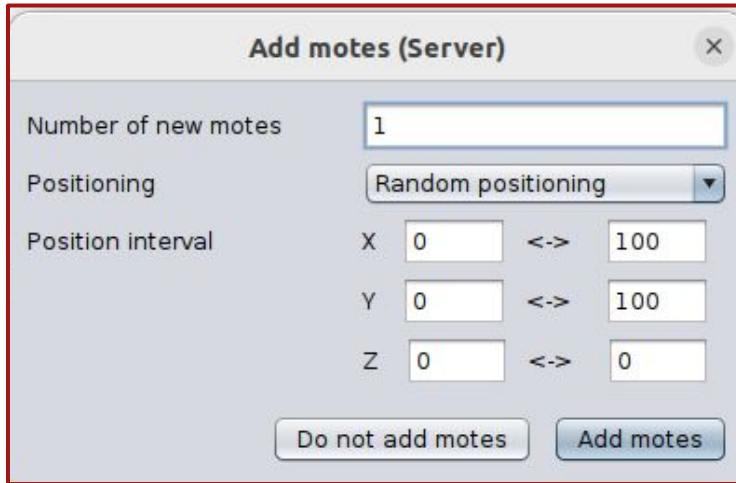
Description:

Contiki process / Firmware:

- 7 Click "**Compile**"
- 8 Click "**Create**"

Cooja

- 9 Put “1” in the “Number of new motes” field



Dialog box titled "Add motes (Server)".

Number of new motes: 1

Positioning: Random positioning

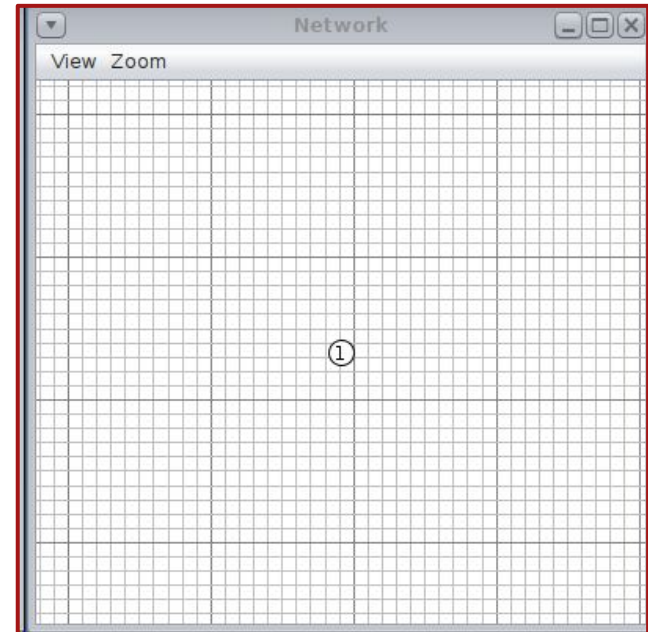
Position interval:

- X: 0 <-> 100
- Y: 0 <-> 100
- Z: 0 <-> 0

Buttons: Do not add motes, Add motes

- 10 Click “**Add motes**”

- 11 You should get this —————>



Exercise

- 1 Create ONE new “Sender” mote:

```
/home/ubuntu/Documents/valsalice-iot-23-template/lab/week14/simple/sender.c
```

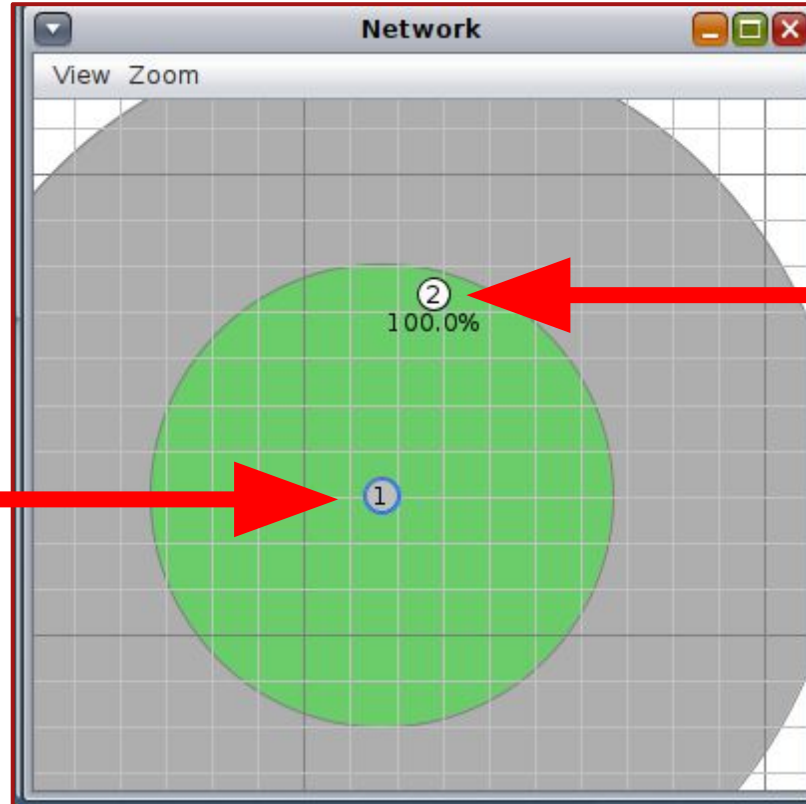
- 2 Create ONE new “Receiver” mote:

```
/home/ubuntu/Documents/valsalice-iot-23-template/lab/week14/simple/receiver.c
```

Cooja

Target topology:

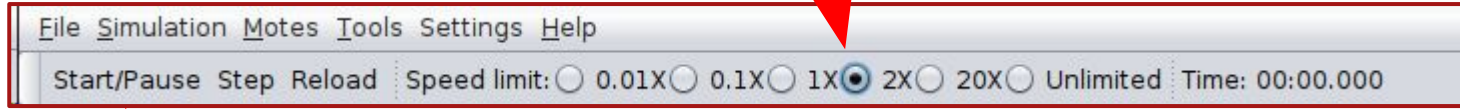
Sender



Receiver

Cooja

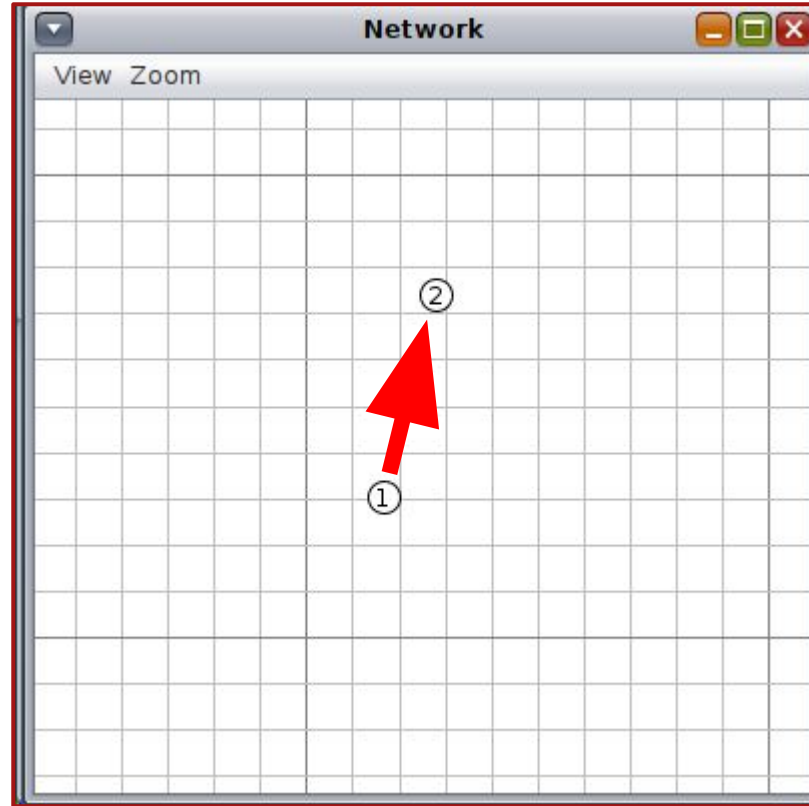
- 12 Set the simulation speed to **2X**



- 13 **Start** the simulation

Cooja

Packet Route:



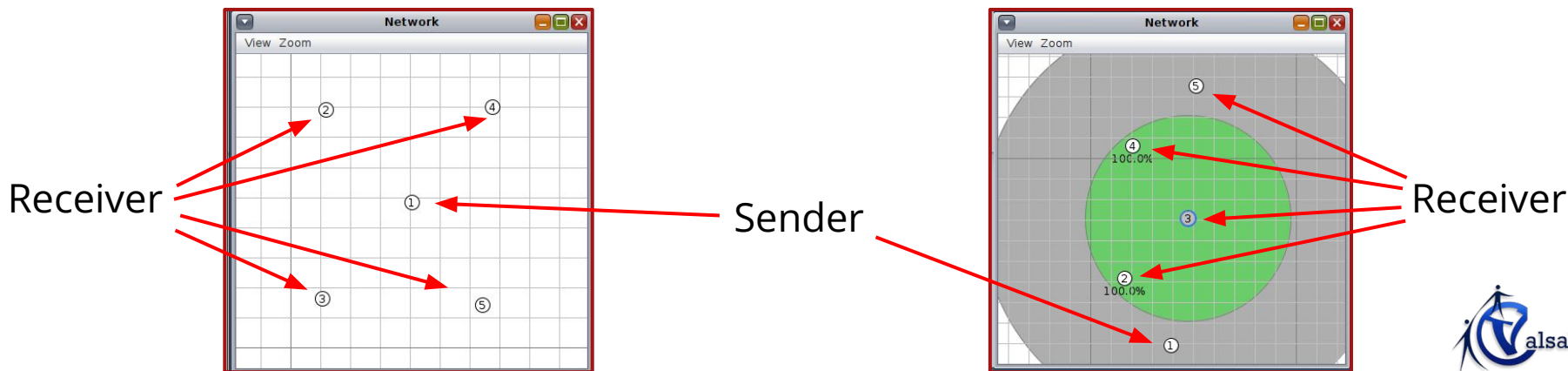
Exercise

- 1 Create **ONE** new “Sender” mote:

`/home/ubuntu/Documents/valsalice-iot-23-template/lab/week14/simple/sender.c`

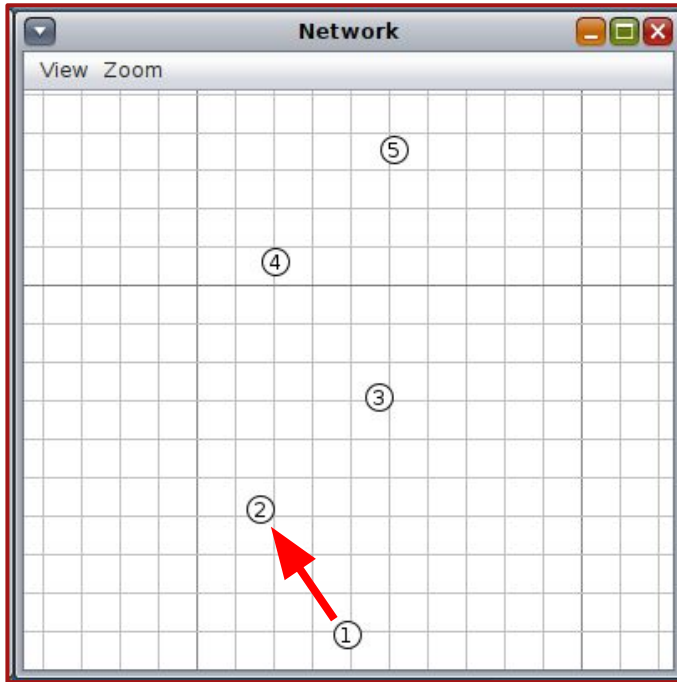
- 2 Create **FOUR** new “Receiver” motes:

`/home/ubuntu/Documents/valsalice-iot-23-template/lab/week14/simple/receiver.c`

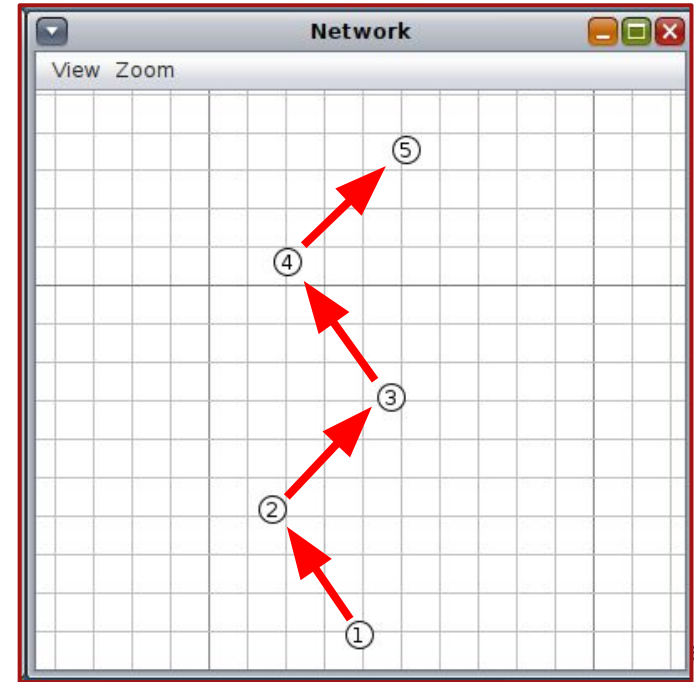


Cooja

Packet Route:



Expected Route:



Exercise



Directory: **week14/multi**

1

Create **ONE** new “Sender” mote:

`/home/ubuntu/Documents/valsalice-iot-23-template/lab/week14/multi/sender.c`

2

Create **ONE** new “Receiver” mote:

`/home/ubuntu/Documents/valsalice-iot-23-template/lab/week14/multi/receiver.c`

3

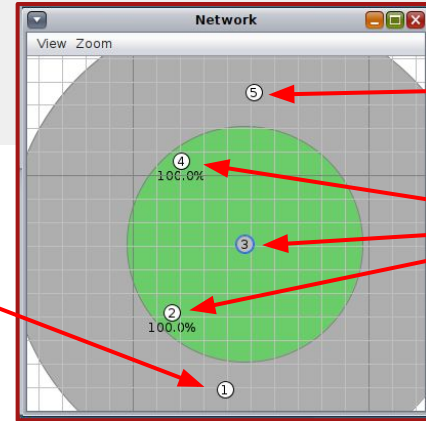
Create **THREE** new “Relay” mote:

`/home/ubuntu/Documents/valsalice-iot-23-template/lab/week14/multi/relay.c`

Sender

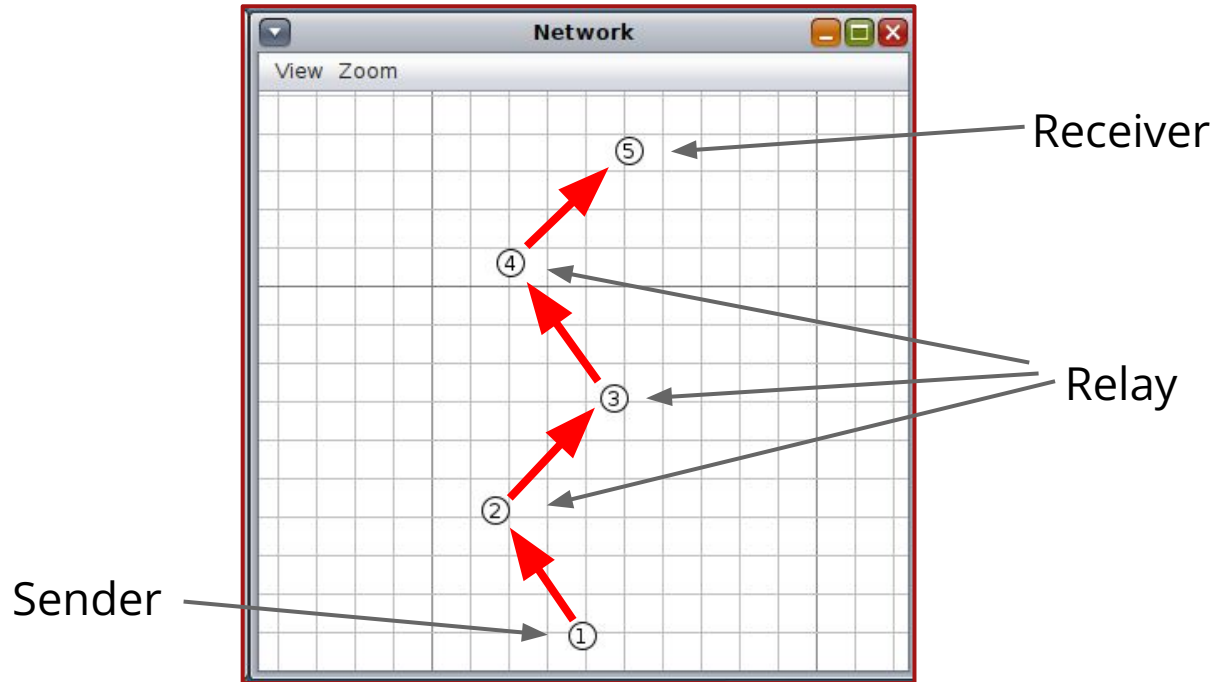
Receiver

Relay



Cooja

Packet Route:



Save remotely your Changes

1

```
make save
```

2

```
|Password
```

```
Git: https://aspina@git.spina.me (Press 'Enter' to confirm or  
'Escape' to cancel)
```

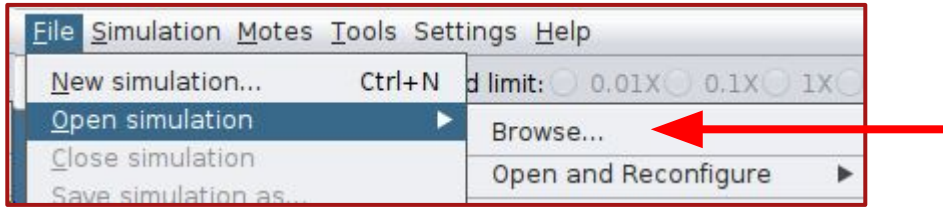
3

```
✓ Changes committed and pushed. All done!
```

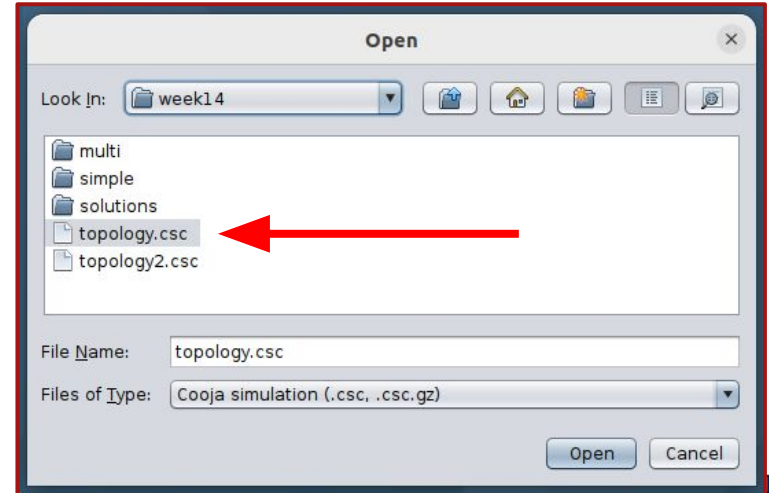
Cooja

Open an existing Cooja Simulation

File > Open simulation > Browse...



Open the **topology.csc** file
Inside the **week14** directory



Exercise



Directory: **week14**

1

Open the file:

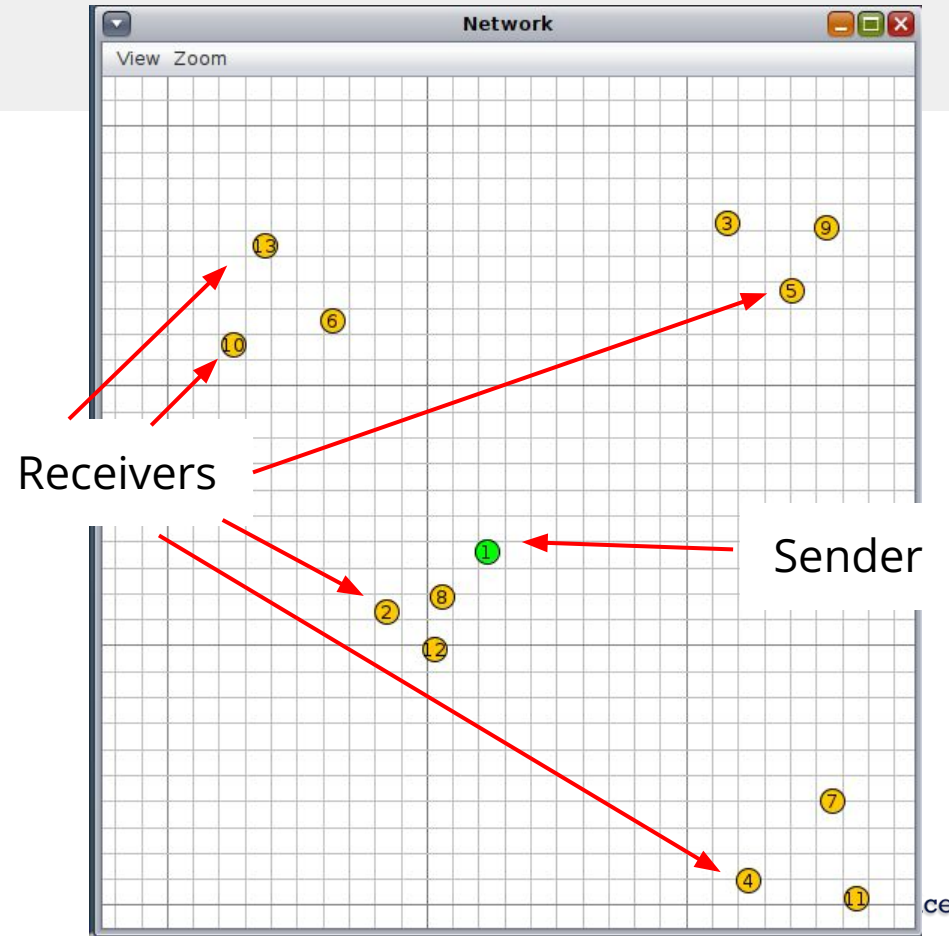
topology.csc

2

Add Motes to ensure all **Receiver** motes can be reached by the **Sender**



DO NOT move any motes



Save remotely your Changes

1

```
make save
```

2

```
|Password
```

```
Git: https://aspina@git.spina.me (Press 'Enter' to confirm or  
'Escape' to cancel)
```

3

```
✓ Changes committed and pushed. All done!
```

Exercise



Directory: **week14**

1

Open the file:

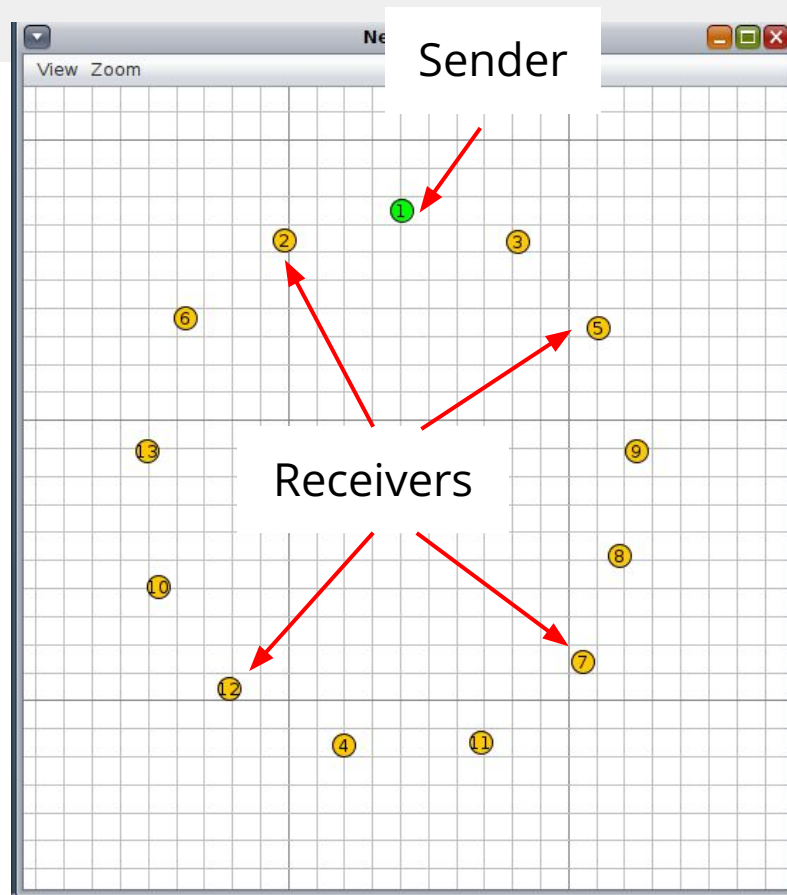
topology2.csc

2

Add Motes to ensure all **Receiver** motes can be reached by the **Sender**



DO NOT move any motes



Save remotely your Changes

1

```
make save
```

2

```
|Password
```

```
Git: https://aspina@git.spina.me (Press 'Enter' to confirm or  
'Escape' to cancel)
```

3

```
✓ Changes committed and pushed. All done!
```

Quiz Time!

ahaslides.com/SWIRN

End of Class

See you all next week!