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. Lossy Networks
16. Reliable Data Transfer Challenge — MAR 5 / MAR 12

= 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. 

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

3. Open the **week17** 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?

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

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

```c
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!

```c
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);
```

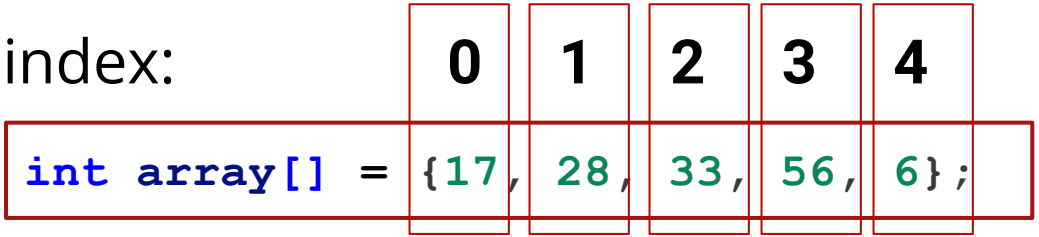
With a **list**:

```
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.

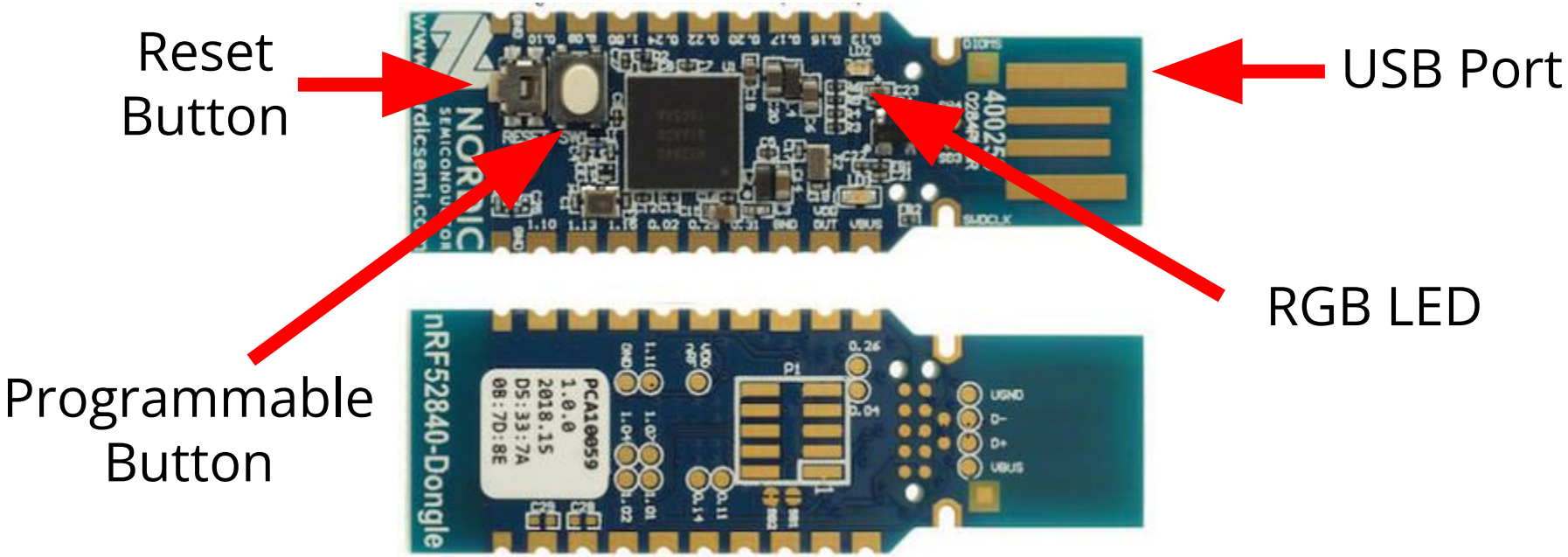
NOTE: List indices start from **0**




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

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

Recap: What is the nRF52840?



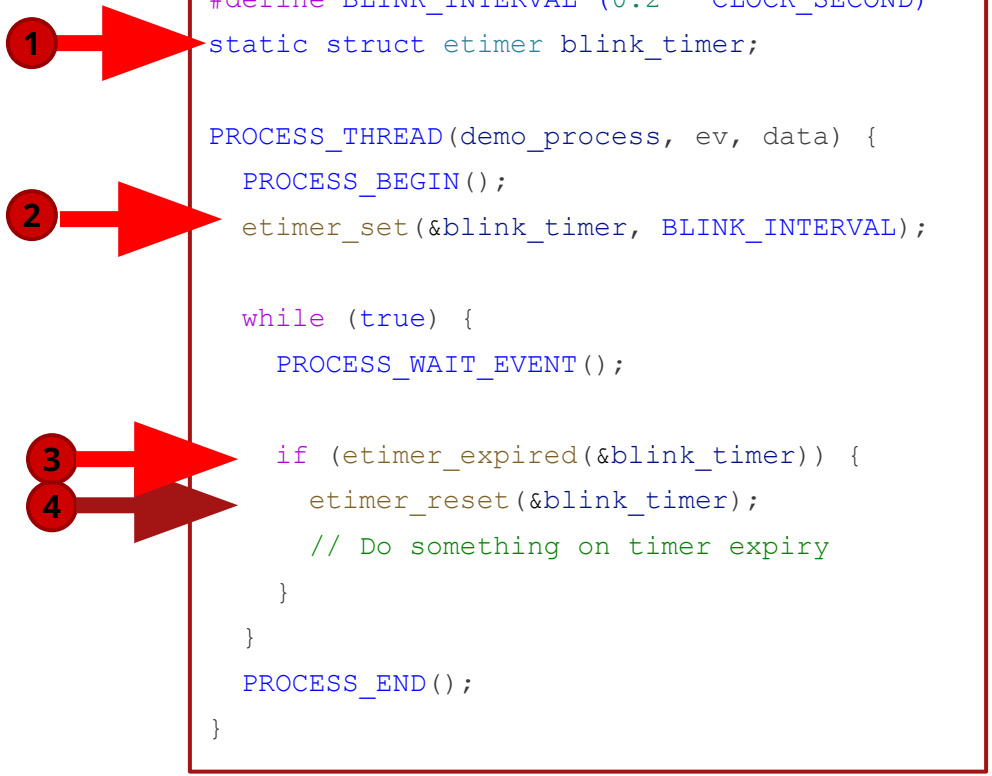
Recap: The LED Library

```
#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)
```

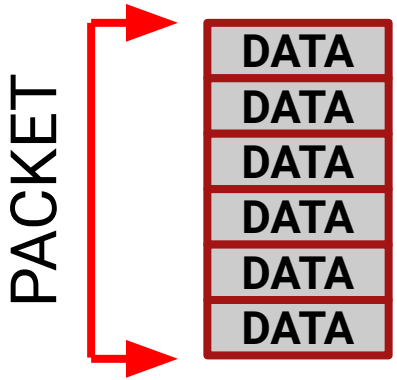
Recap: Using an E-Timer




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

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

  while (true) {
    PROCESS_WAIT_EVENT();

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

Recap: Networking Packets

- Networking involves **transmitting data** over a medium.
- Information is transmitted in **packets**: small chunks of **data** (of arbitrary length) sent over the network.



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: 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: 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: Header Files

Header files allow you to define types, structures, or **common code** once and **reuse** it in multiple source files

file.c

```c
#include "config.h"

printf("Team ID: %c", TEAM_ID);
```

config.h

```c
#ifndef CONFIG_H
#define CONFIG_H

/* Message Configuration */
typedef struct
{
    char team_id;
    int data;
} message_t;

#define TEAM_ID 'Z'

#endif // CONFIG_H
```

Recap: 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.

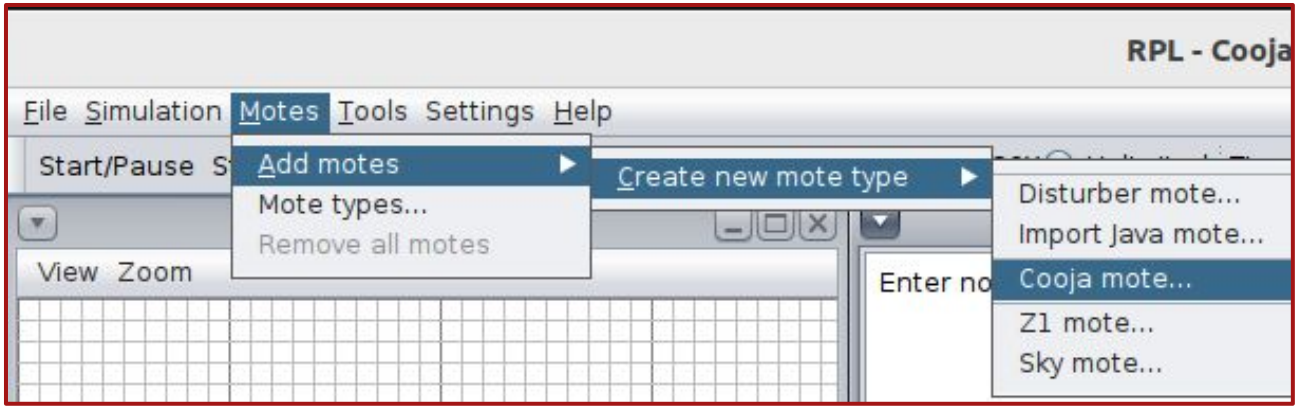
1. From the Terminal run the **make cooja** command:

```
make cooja
```

Recap: Adding a Cooja Mote

1. Let's add a new Mote:

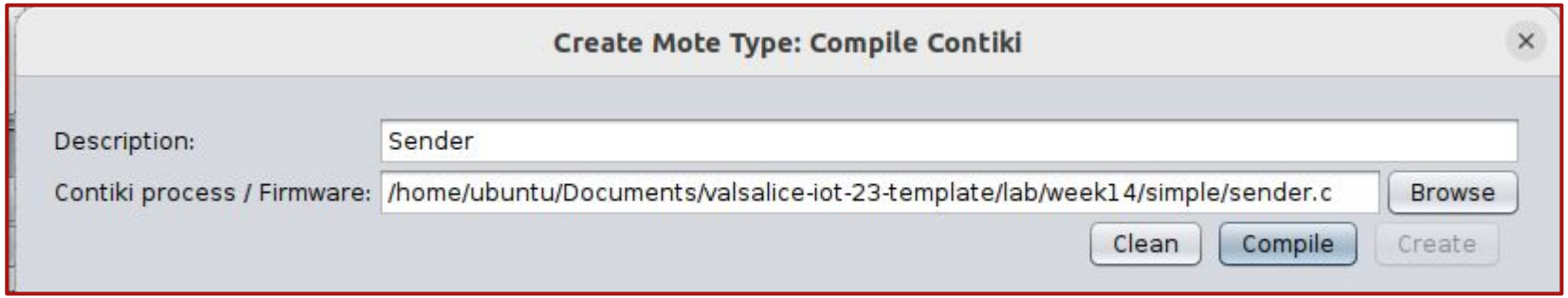
Motes > Add motes > Create new > Cooja mote



Recap: Adding a Cooja Mote

2. Put “**Sender**” in the description
3. Use the firmware under (you can also use “Browse”):

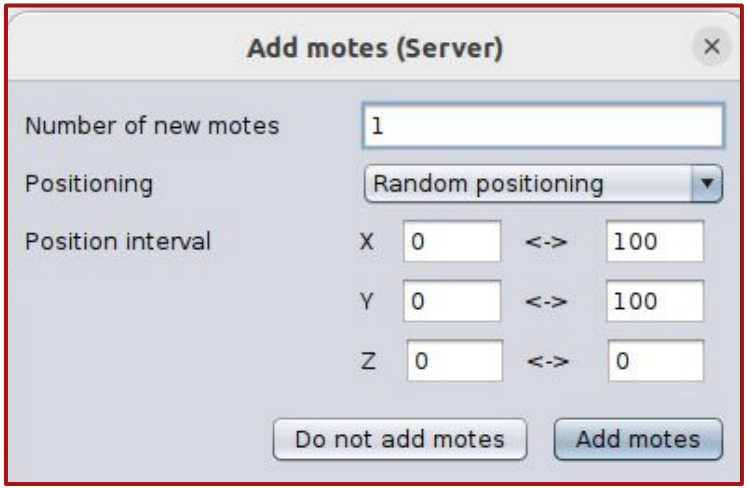
`/home/ubuntu/Documents/valsalice-iot-23/lab/week14/simple/sender.c`



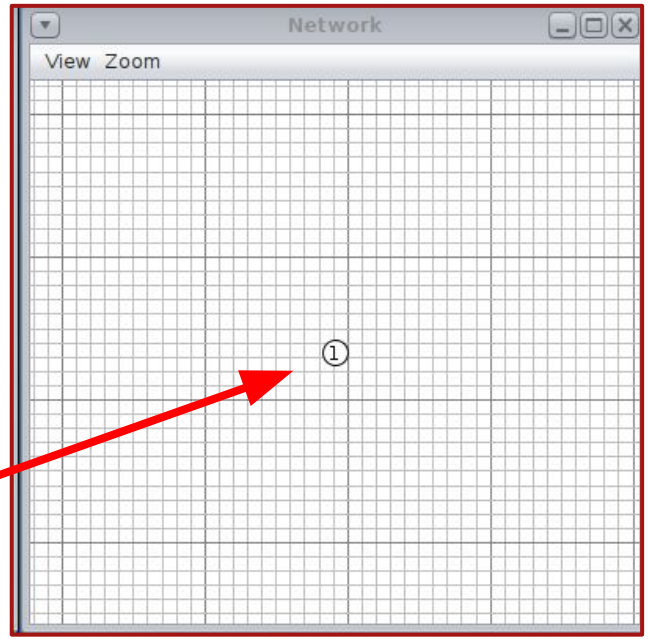
4. Click “**Compile**”
5. Click “**Create**”

Recap: Adding a Cooja Mote

6. Put “**1**” in the “Number of new motes” field

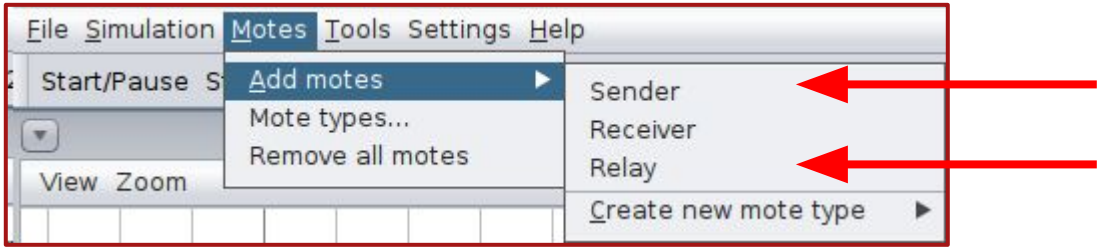


7. Click “**Add motes**”
8. You should get this



Recap: Adding a Cooja Mote

9 After you create the Mote it will be available for **quick access** in the simulation:

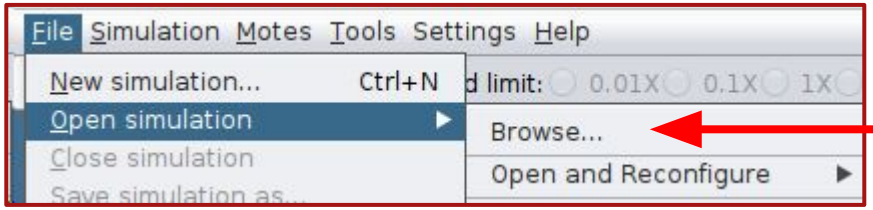


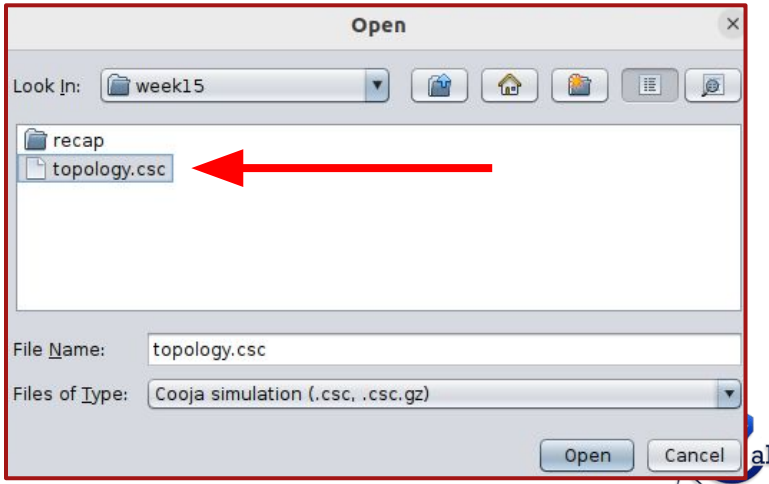
10 Simply click on the **Mote name** to create new motes of that type

Recap: Opening a Cooja Simulation

Open an existing Cooja Simulation

`File > Open simulation > Browse…`

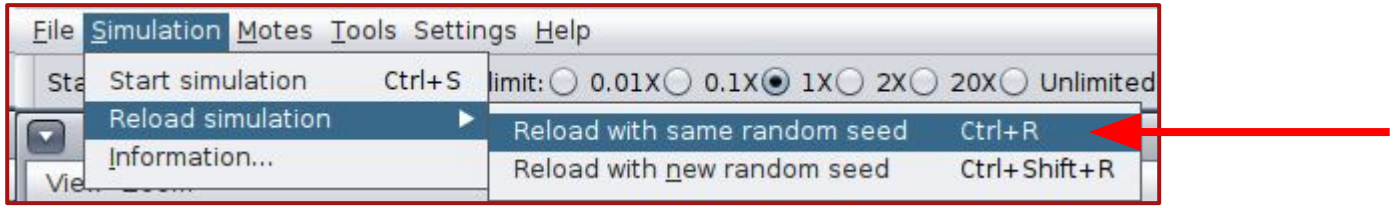




Recap: Recompiling Mote Code

To recompile motes you simply reload the simulation

`Simulation > Reload Simulation > Same seed`



SHORTCUT: You can simply press **Ctrl + R**

⚠ Errors will be raised:

Recap: Node IDs

Node IDs are **unique IDs** assigned to each **physical chip**. They allow us to identify each chip when it sends or receives messages.

Cooja additionally **simulates IDs** assigning them to the nodes.

```
#include "sys/node-id.h"


printf("Node ID: %d\n", node_id);
```

Recap: Logging

Rather than using prints to **facilitate visualising** information on serial output we use a “logger”.

Fundamentally any time we would use **printf** we can now use **LOG_INFO**, there is **no other difference**.

```
#include "sys/node-id.h"


printf("Node ID: %d\n", node_id);
LOG_INFO("Node ID: %d\n", node_id);
```

Recap: Leveraging Node IDs

We will begging using Node IDs to know where are coming from:

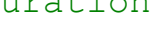
- **team_id** is for team membership
- **node_id** tracks packet origin

file.c

```
#include "config.h"

char tid = message.team_id;
int nid = message.node_id;

printf("node_id '%d', team'%c'\n", nid, tid);
```

config.h

```
#ifndef CONFIG_H
#define CONFIG_H

/* Message Configuration */
typedef struct
{
    char team_id;
    int node_id;
} message_t;

#endif // CONFIG_H
```



Recap: More Message Configurations

We are changing the **message_t** struct to now also be able to disseminate data together with each packet

- **team_id** is for team membership
- **node_id** tracks packet origin
- **data** payload of each packet

config.h

```
#ifndef CONFIG_H
#define CONFIG_H

/* Message Configuration */
typedef struct
{
    char team_id;
    int node_id;
    int data;
} message_t;

#endif // CONFIG_H
```

Recap: Counting in Circles

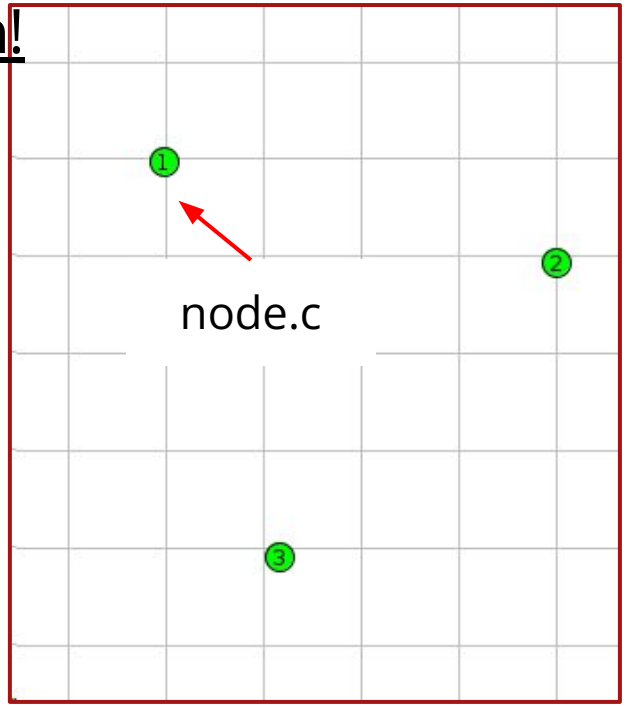
⚠ We changed the **TEAM_ID** in **`config.h`**!

Open used the following Cooja topology:

We used a single Node:

1) **Node (node.c)**:

```
// If sender is node 1 and I am node 2: reply
// If sender is node 2 and I am node 3: reply
// If sender is node 3 and I am node 1: reply
```



Recap: Counting in Circles

```
if (msg_team_id == TEAM_ID)
{
        // Turn the LED on ONLY IF we receive data
        leds_single_on(LEDS_LED1);
        if (msg_node_id == 1 && node_id == 2) {
                send_nullnet_data(msg_data + 1);
        } else if (msg_node_id == 2 && node_id == 3) {
                send_nullnet_data(msg_data + 1);
        } else if (msg_node_id == 3 && node_id == 1) {
                send_nullnet_data(msg_data + 1);
        }
}
```

Test that it works

⚠ Open a new terminal in **week16/hello**

1) Flash the hello firmware to get the IDs of the nodes:

```
make hello.dfu-upload
```

⚠ Open a new terminal in **week16/counting**

2) Change your **node.c** code to use the new IDs:

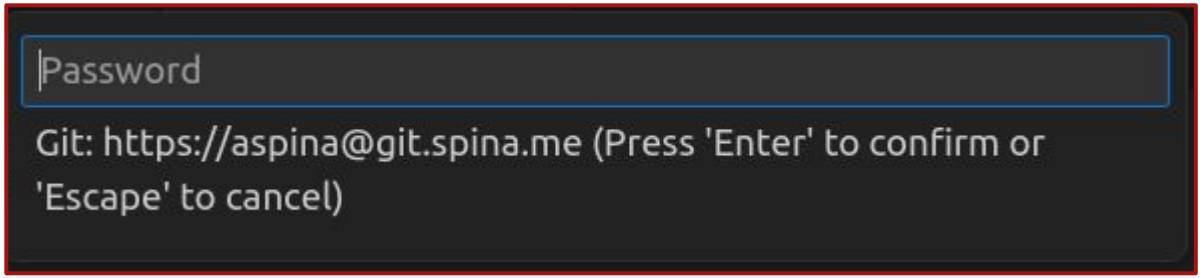
3) Flash the node firmware on **ALL motes** and make login:

```
make node.dfu-upload
```

```
make login
```

Save remotely your Changes

1. `make save`

2. 

3. 

Reliability Challenge

⚠ Change the **TEAM_ID** in **`config.h`**!

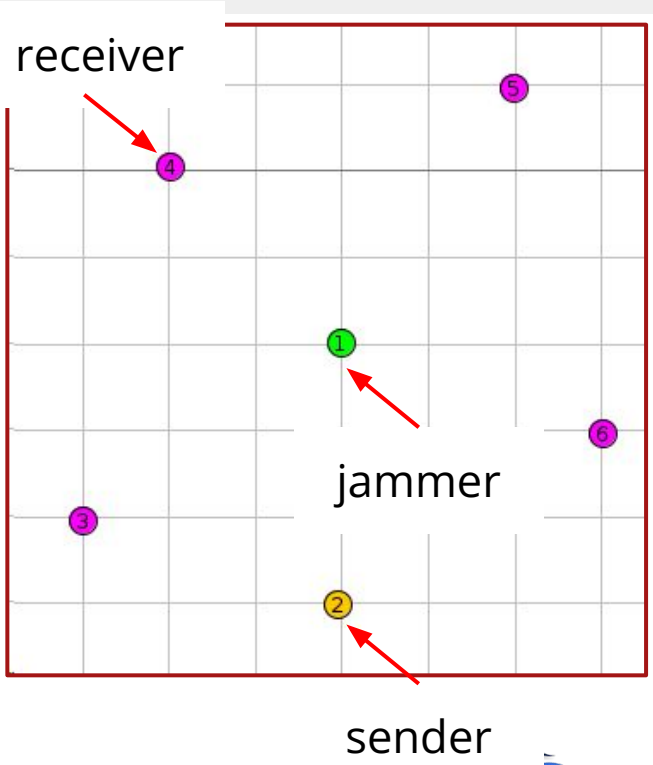
Cooja simulation: **week17/topology.csc**

1) **Sender (sender.c)**:

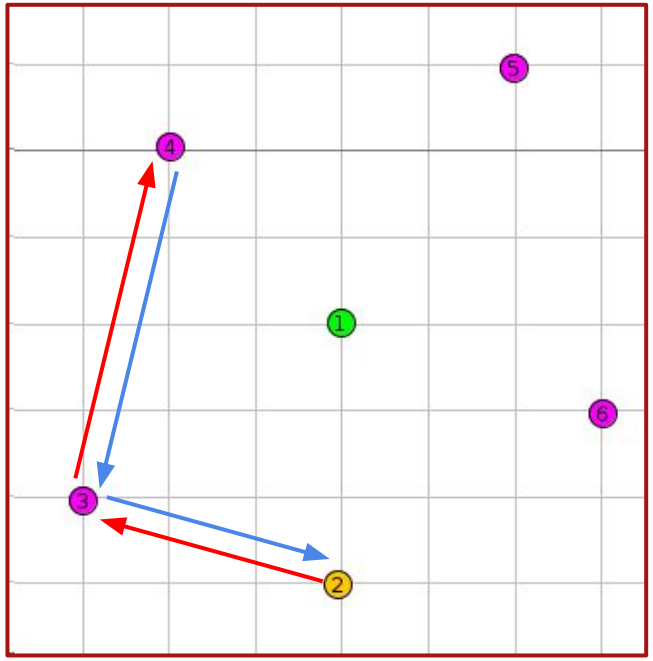
```
// TODO(1): Use `send_nullnet_data` to actually send the message
```

2) **Receiver (receiver.c)**:

```
// TODO(1): Change `true` to a different condition

// TODO(2): Reply using `send_nullnet_data` to send the payload back!

// TODO(3): What if the target node is not the current node?

// TODO(4): What can you try to do to increase the network's reliability?

// TODO(5): Should you try to do anything to prevent duplicate messages?
```



Communication Example




```
typedef struct
{
    char team_id;
    int source_node_id;
    int target_node_id;
    int payload;
} message_t;
```

```
{
    'A';
    2;
    4;
    12345678;
};
```

```
{
    'A';
    4;
    2;
    9998886;
};
```

Test that it works

⚠ Open a new terminal in **week16/hello**

1) Flash the hello firmware to get the IDs of the nodes:

```
make hello.dfu-upload
```

⚠ Open a new terminal in **week17**

2) Change your **config.h** code to use the new IDs and make clean:

```
make clean
```

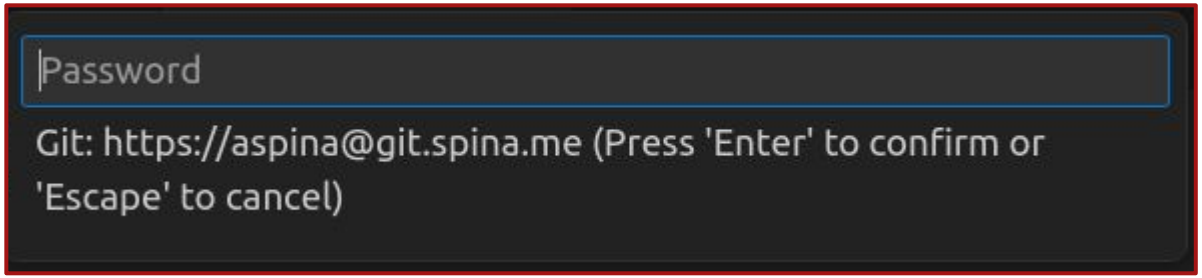
3) Flash the sender firmware on **ONE motes** and make login:
4) Flash the sender firmware on **ALL OTHER motes** and make login:

```
make sender.dfu-upload
```

```
make receiver.dfu-upload
```

Save remotely your Changes

1. `make save`

2. 

3. 

End of Class

See you all next week!