```
CFLAGS += -Wall

CONTIKI_PROJECT = jerome jerome-udp-client jerome-udp-server jerome-sensor-test jerome-calc
-test
all: $(CONTIKI_PROJECT)

PROJECT_SOURCEFILES += jerome-calc.c
MODULES += $(CONTIKI_NG_SERVICES_DIR)/unit-test

CONTIKI = ../
include $(CONTIKI)/Makefile.include
```

```c
/*****************************************************************************
* File name: jerome.c
*
* Author: Jerome Samuels-Clarke
* Date: Edited November 2021
*
* Description: A program that reads the temperature and humdity values from a Tmote Sky.
               The values are passed through an overlapping sliding window filter, before
               being printed to the terminal.
*****************************************************************************/
#include "contiki.h" /* Header file included to use contiki timers */
#include <stdio.h>   /* Included for the printf statements for terminal debugging */

/* Include header files for the sht11 sensor, to read the temperature and humidity */
#include "dev/sensor/sht11/sht11.h"
#include "dev/sensor/sht11/sht11-sensor.h"

#include "jerome-calc.h" /* Header file for the OSW filter functions */
/*****************************************************************************/

/*****************************************************************************/
/* Declare and start two processes, for temperature and humiditiy */
PROCESS(temperature, "Temperature");
PROCESS(humidity, "Humidity");
AUTOSTART_PROCESSES(&temperature, &humidity);
/*****************************************************************************/

/* This process statement is used for obtaining the temperature readings from the sht11
   sensor. The readings are filtered using an overlapping sliding window filter. */
PROCESS_THREAD(temperature, ev, data)
{
   /* Decalre a timer instance */
   static struct etimer timer;

   PROCESS_BEGIN();

   /* Initialise an overlapping sliding window filter instance for the temperature
     sensor */
   static s_osw temperature_osw;
   osw_init(&temperature_osw);

   /* Setup a periodic timer that expires after 1 second */
   etimer_set(&timer, CLOCK_SECOND * 1);

   /* Activate and configure the sht11 sensor*/
   SENSORS_ACTIVATE(sht11_sensor);

   while (1)
   {
      PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));

      /* Read and store the temperature from the sht11 sensor */
      int temp_val = sht11_sensor.value(SHT11_SENSOR_TEMP);

      /* Convert the temperature to Celsius, and add it into the filter buffer */
      float s = ((0.01 * temp_val) - 39.60);
      osw_add(&temperature_osw, s);

      /* Return the average, separate into an integer and fractional part. Print the
       returned value. */
      int dec_avg = (int)osw_avg(&temperature_osw);
      int fraction_avg = (osw_avg(&temperature_osw) - dec_avg) * 100;

      printf("Average Temperature: %d.%02uC\n", dec_avg, fraction_avg);

      /* For comparision between filtered and non-filtered values, convert and print
         the raw temperature readings. */
      int dec = (int)s;
```

```
      int fraction = (s - dec) * 100;

      printf("Current Temperature: %d.%02uC\n", dec, fraction);

      etimer_reset(&timer);
   }

   PROCESS_END();
}
/*---------------------------------------------------------------------------*/

/* This process statement is used for obtaining the humidity readings from the sht11
   sensor. The readings are filtered using an overlapping sliding window filter. */
PROCESS_THREAD(humidity, ev, data)
{
   /* Decalre a timer instance */
   static struct etimer timer;

   PROCESS_BEGIN();

   /* Initialise an overlapping sliding window filter instance for the humidity
     sensor */
   static s_osw humidity_osw;
   osw_init(&humidity_osw);

   /* Setup a periodic timer that expires after 1 second */
   etimer_set(&timer, CLOCK_SECOND * 1);

   /* Activate and configure the sht11 sensor*/
   SENSORS_ACTIVATE(sht11_sensor);

   while (1)
   {
      PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));

      /* Read and store the humidity from the sht11 sensor */
      int humid_val = sht11_sensor.value(SHT11_SENSOR_HUMIDITY);

      /* Convert the humditity to relative humidity (%), and add it into the
         filter buffer */
      float s = -4 + 0.0405 * humid_val - 2.8e-6 * humid_val * humid_val;
      osw_add(&humidity_osw, s);

      /* Return the average, separate into an integer and fractional part. Print the
         returned value. */
      int dec_avg = (int)osw_avg(&humidity_osw);
      int fraction_avg = (osw_avg(&humidity_osw) - dec_avg) * 100;

      printf("Average Humidity: %d.%02u%%\n", dec_avg, fraction_avg);

      /* For comparision between filtered and non-filtered values, convert and print
         the raw humidity readings. */
      int dec = (int)s;
      int fraction = (s - dec) * 100;

      printf("Current Humidity: %d.%02u%%\n\n", dec, fraction);

      etimer_reset(&timer);
   }

   PROCESS_END();
}
/*---------------------------------------------------------------------------*/
```

```c
/*****************************************************************************
* File name: jerome-calc.c
*
* Author: Jerome Samuels-Clarke
* Date: Edited November 2021
*
* Description: C file containing the function implementations for the overlapping
               sliding window filter.
*****************************************************************************/
#include "jerome-calc.h" /* Included header file */
/*****************************************************************************/

/* Initialse the s_osw instance, setting write and count variables to zero */
void osw_init(s_osw *osw)
{
    osw->write = 0;
    osw->count = 0;
}
/*****************************************************************************/

/* Add data into the buffer, and check write and count variables  */
void osw_add(s_osw *osw, float data_in)
{
    osw->buf[osw->write] = data_in;
    osw->write++;

    if (osw->write >= MAX_BUFFER_SIZE)
    {
        osw->write = 0;
    }

    if (osw->count < MAX_BUFFER_SIZE)
    {
        osw->count++;
    }
}
/*****************************************************************************/

/* Depending on the value of count, average the data in the buffer and return the answer
   as a float. */
float osw_avg(s_osw *osw)
{
    int i;
    float osw_average = 0;

    for (i = 0; i < osw->count; i++)
    {
        osw_average += osw->buf[i];
    }

    return osw_average / osw->count;
}
/*****************************************************************************/
```

```c
/*****************************************************************************
* File name: jerome-calc.h
*
* Author: Jerome Samuels-Clarke
* Date: Edited November 2021
*
* Description: Header file for the overlapping sliding window filter.
*****************************************************************************/
#ifndef JEROME_CALC_H
#define JEROME_CALC_H

#define MAX_BUFFER_SIZE 7 // Maximum size of buffer

/**@brief   Structure for handling the overlapping window buffer */
typedef struct s_osw
{
    int write, count;
    float buf[MAX_BUFFER_SIZE];
} s_osw;

/**@brief       Function for initialising the Overlapping Sliding Window filter
 *
 * @param[in]   osw - structure of type s_osw
 */
void osw_init(s_osw *osw);

/**@brief       Function for adding data into the buffer
 *
 * @param[in]   osw - structure of type s_osw
 * @param[in]   data_in - input data to store in buffer
 */
void osw_add(s_osw *osw, float data_in);

/**@brief       Function for getting the average from the data in the buffer
 *              using an overlapping sliding window filter
 *
 * @param[in]   osw - structure of type s_osw
 * @param[out]  average of data stored in buffer (float)
 */
float osw_avg(s_osw *osw);

#endif
```

```c
/*****************************************************************************
* File name: jerome-calc-test.c
*
* Author: Jerome Samuels-Clarke
* Date: Edited November 2021
*
* Description: A unit test done in C, using the unit-test program provided by Contiki.
               Unit tests are done to check the overlapping sliding window (OSW) filter.
*****************************************************************************/
#include <stdio.h>       /* Included for the printf statements for terminal debugging */
#include "unit-test.h"   /* Included to use the unit test macros */
#include "jerome-calc.h" /* Header file for the OSW filter functions */
/*****************************************************************************/

/*****************************************************************************/
/* Set up three test registers for initialising the OSW filter, adding
   value into the buffer, and returning the average */
UNIT_TEST_REGISTER(OSW_INIT, "Overlapping Sliding Window Filter Initialise");
UNIT_TEST_REGISTER(OSW_ADD, "Overlapping Sliding Window Filter Adding");
UNIT_TEST_REGISTER(OSW_AVG, "Overlapping Sliding Window Filter Average");
/*****************************************************************************/

/* This unit test checks the initialisation of the OSW instance. Initialise the instance
   should set the write and count members to zero. */
UNIT_TEST(OSW_INIT)
{
    UNIT_TEST_BEGIN();

    /* Declare an instance and initialise the OSW filter */
    s_osw osw;
    osw_init(&osw);

    UNIT_TEST_ASSERT(osw.write == 0);
    UNIT_TEST_ASSERT(osw.count == 0);

    UNIT_TEST_END();
}
/*****************************************************************************/

/* This unit test checks the values being added into the buffer */
UNIT_TEST(OSW_ADD)
{
    UNIT_TEST_BEGIN();

    int i; /* Variable used for indexing into test data. */

    /* Test data used for adding into the buffer */
    const int test_data[] =
        {5, 6, 7, 8, 9, 10, 11,
         12, 13, 14, 15, 16, 17, 18,
         19, 20, 21, 22};

    /* Declare an instance and initialise the OSW filter */
    s_osw osw;
    osw_init(&osw);

    /* Add data into the buffer and test the buffer holds the added data */
    osw_add(&osw, 3);
    UNIT_TEST_ASSERT(osw.buf[0] == 3);

    /* Initialise the OSW filter and add data into the buffer. Test the buffer holds
       the added data */
    osw_init(&osw);
    osw_add(&osw, 7);
    osw_add(&osw, 9);
    UNIT_TEST_ASSERT(osw.buf[1] == 9);

    /* Initialise the OSW filter and add data into the buffer. Test the buffer holds
```

```
       the added data. As the buffer has a limit size of 7 elements, adding more data
       will wrap around. */
    osw_init(&osw);
    for (i = 0; i < 18; i++)
    {
        osw_add(&osw, test_data[i]);
    }
    UNIT_TEST_ASSERT(osw.buf[3] == 22);

    UNIT_TEST_END();
}
/*---------------------------------------------------------------------------*/

/* This unit test checks the return value is the average for the filter */
UNIT_TEST(OSW_AVG)
{
    UNIT_TEST_BEGIN();

    int i; /* Variable used for indexing into test data */

    /* Declare an instance and initialise the OSW filter  */
    s_osw osw;
    osw_init(&osw);

    /* Test data used for adding into the buffer */
    const int test_data[] =
        {5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22};

    /* Add a value into the buffer and test the average */
    for (i = 0; i < 1; i++)
    {
        osw_add(&osw, test_data[i]);
    }
    UNIT_TEST_ASSERT(osw_avg(&osw) == 5);

    /* Initialise the OSW filter and add data into the buffer. Test the average being retur
ned. */
    osw_init(&osw);
    for (i = 0; i < 4; i++)
    {
        osw_add(&osw, test_data[i]);
    }
    UNIT_TEST_ASSERT(osw_avg(&osw) == 6.5);

    /* Initialise the OSW filter and add data into the buffer. Test the average being
       returned. */
    osw_init(&osw);
    for (i = 0; i < 8; i++)
    {
        osw_add(&osw, test_data[i]);
    }
    UNIT_TEST_ASSERT(osw_avg(&osw) == 9);

    /* Initialise the OSW filter and add data into the buffer. Test the average being
       returned. */
    osw_init(&osw);
    for (i = 0; i < 18; i++)
    {
        osw_add(&osw, test_data[i]);
    }
    UNIT_TEST_ASSERT(osw_avg(&osw) == 19);

    UNIT_TEST_END();
}
/*---------------------------------------------------------------------------*/

/* Initialise and start a process thread, to run the unit tests */
PROCESS(test_process, "unit testing");
```

```
AUTOSTART_PROCESSES(&test_process);

PROCESS_THREAD(test_process, ev, data)
{
    PROCESS_BEGIN();

    /* Run the unit tests for the three OSW functions */
    UNIT_TEST_RUN(OSW_INIT);
    UNIT_TEST_RUN(OSW_ADD);
    UNIT_TEST_RUN(OSW_AVG);

    PROCESS_END();
}
/*---------------------------------------------------------------------------*/
```

```c
/*****************************************************************************
* File name: jerome-sensor-test.c
*
* Author: Jerome Samuels-Clarke
* Date: Edited November 2021
*
* Description: A unit test done in C, using the unit-test program provided by Contiki.
               The unit tests are done to check the formula outputs for the temperature
               and humidity conversions.
*****************************************************************************/
#include <stdio.h>     /* Included for the printf statements for terminal debugging */
#include "unit-test.h" /* Included to use the unit test macros */
/*****************************************************************************/

/*****************************************************************************/
/* Set up two test registers for temperature and humidity */
UNIT_TEST_REGISTER(Temperature_test, "Temperature Conversion");
UNIT_TEST_REGISTER(Humidity_test, "Humidity Conversion");
/*****************************************************************************/

/* This unit test checks the temperature conversion formula. */
UNIT_TEST(Temperature_test)
{
  UNIT_TEST_BEGIN();
  float s;

  int test_temp[] = {6630, 3960, 8069, 2378, 1365};

  s = ((0.01 * test_temp[0]) - 39.60);
  UNIT_TEST_ASSERT(s == 26.7);

  s = ((0.01 * test_temp[1]) - 39.60);
  UNIT_TEST_ASSERT(s == 0);

  s = ((0.01 * test_temp[2]) - 39.60);
  UNIT_TEST_ASSERT(s == 41.09);

  s = ((0.01 * test_temp[3]) - 39.60);
  UNIT_TEST_ASSERT(s == -15.83);

  s = ((0.01 * test_temp[4]) - 39.60);
  UNIT_TEST_ASSERT(s == -25.95);

  UNIT_TEST_END();
}
/*****************************************************************************/

/* This unit test checks the humditity conversion formula. */
UNIT_TEST(Humidity_test)
{
  UNIT_TEST_BEGIN();

  float s;

  int test_humid[] = {1250, 622, 3340, 2500, 200};

  s = -4 + (0.0405 * test_humid[0]) - 2.8e-6 * test_humid[0] * test_humid[0];
  UNIT_TEST_ASSERT(s == 42.5);

  s = -4 + (0.0405 * test_humid[1]) - 2.8e-6 * test_humid[1] * test_humid[1];
  UNIT_TEST_ASSERT(s == 20.1);

  s = -4 + (0.0405 * test_humid[2]) - 2.8e-6 * test_humid[2] * test_humid[2];
  UNIT_TEST_ASSERT(s == 100.03);

  s = -4 + (0.0405 * test_humid[3]) - 2.8e-6 * test_humid[3] * test_humid[3];
  UNIT_TEST_ASSERT(s == 79.75);
```

```
  s = -4 + (0.0405 * test_humid[4]) - 2.8e-6 * test_humid[4] * test_humid[4];
  UNIT_TEST_ASSERT(s == 3.98);

  UNIT_TEST_END();
}
/*---------------------------------------------------------------------------*/

/* Initialise and start a process thread, to run the unit tests */
PROCESS(test_process, "unit testing");
AUTOSTART_PROCESSES(&test_process);

PROCESS_THREAD(test_process, ev, data)
{
  PROCESS_BEGIN();

  /* Run the unit tests for the temperature and humidity */
  UNIT_TEST_RUN(Temperature_test);
  UNIT_TEST_RUN(Humidity_test);

  PROCESS_END();
}
/*---------------------------------------------------------------------------*/
```

```c
/*****************************************************************************
* File name: jerome-udp-client.c
*
* Author: Jerome Samuels-Clarke
* Date: Edited November 2021
*
* Description: Client code for sending data to a server node, based on RPL-UDP. The data
               being sent is temperature and humdity readings.
*****************************************************************************/
#include "contiki.h"
#include "net/routing/routing.h"
#include "random.h"
#include "net/netstack.h"
#include "net/ipv6/simple-udp.h"

#include "dev/sensor/sht11/sht11.h"
#include "dev/sensor/sht11/sht11-sensor.h"
#include "jerome-calc.h"

#include "sys/log.h"
/*****************************************************************************/

#define LOG_MODULE "App"
#define LOG_LEVEL LOG_LEVEL_INFO

#define WITH_SERVER_REPLY 1
#define UDP_CLIENT_PORT 8765
#define UDP_SERVER_PORT 5678

#define SEND_INTERVAL (1 * CLOCK_SECOND)

static struct simple_udp_connection udp_conn;

/* Store the temperature and humidity converted readings in a buffer */
int sensor_buffer[4] = {0};

/*****************************************************************************/
PROCESS(udp_client_process, "UDP client");
PROCESS(temperature, "Temperature");
PROCESS(humidity, "Humidity");
AUTOSTART_PROCESSES(&udp_client_process, &temperature, &humidity);
/*****************************************************************************/

PROCESS_THREAD(udp_client_process, ev, data)
{
  static struct etimer periodic_timer;
  static char str[32];
  uip_ipaddr_t dest_ipaddr;

  PROCESS_BEGIN();

  /* Initialize UDP connection */
  simple_udp_register(&udp_conn, UDP_CLIENT_PORT, NULL,
                      UDP_SERVER_PORT, NULL);

  etimer_set(&periodic_timer, SEND_INTERVAL);

  while (1)
  {
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&periodic_timer));

    if (NETSTACK_ROUTING.node_is_reachable() && NETSTACK_ROUTING.get_root_ipaddr(&dest_ipad
dr))
    {
      /* Send to DAG root */
      LOG_INFO("Sending request to ");
      LOG_INFO_6ADDR(&dest_ipaddr);
      LOG_INFO_("\n");
```

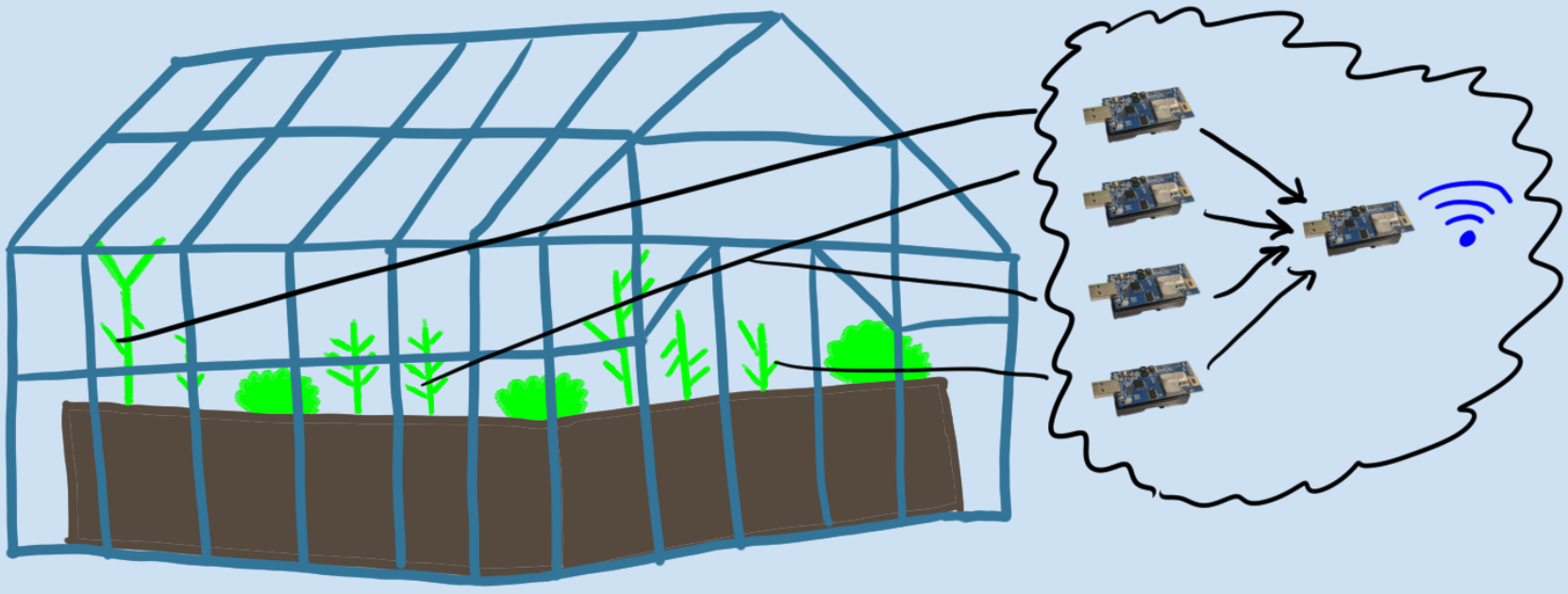
```
      /* Convert the data in the buffer to a string */
      snprintf(str, sizeof(str), "T: %d.%dC, H: %d.%d%%", sensor_buffer[0], sensor_buffer[1
], sensor_buffer[2], sensor_buffer[3]);
      simple_udp_sendto(&udp_conn, str, strlen(str), &dest_ipaddr);
    }
    else
    {
      LOG_INFO("Not reachable yet\n");
    }

    /* Add some jitter */
    etimer_set(&periodic_timer, SEND_INTERVAL);
  }

  PROCESS_END();
}
/*---------------------------------------------------------------------------*/

/* This process statement is used for obtaining the temperature readings from the sht11
   sensor. The readings are filtered using an overlapping sliding window filter. */
PROCESS_THREAD(temperature, ev, data)
{
  /* Decalre a timer instance */
  static struct etimer timer;

  PROCESS_BEGIN();

  /* Initialise an overlapping sliding window filter structure for the temperature
     sensor */
  static s_osw temperature_osw;
  osw_init(&temperature_osw);

  /* Setup a periodic timer that expires after 1 second */
  etimer_set(&timer, CLOCK_SECOND * 1);

  /* Activate and configure the sht11 sensor*/
  SENSORS_ACTIVATE(sht11_sensor);

  while (1)
  {
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));

    /* Read and store the temperature from the sht11 sensor */
    int temp_val = sht11_sensor.value(SHT11_SENSOR_TEMP);

    /* Convert the temperature to Celsius, and add it into the filter buffer */
    float s = ((0.01 * temp_val) - 39.60);
    osw_add(&temperature_osw, s);

    /* Store data in the buffer */
    sensor_buffer[0] = (int)osw_avg(&temperature_osw);
    sensor_buffer[1] = (osw_avg(&temperature_osw) - sensor_buffer[0]) * 100;

    etimer_reset(&timer);
  }

  PROCESS_END();
}
/*---------------------------------------------------------------------------*/

/* This process statement is used for obtaining the humidity readings from the sht11
   sensor. The readings are filtered using an overlapping sliding window filter. */
PROCESS_THREAD(humidity, ev, data)
{
  /* Decalre a timer instance */
  static struct etimer timer;
```

```
  PROCESS_BEGIN();

  /* Initialise an overlapping sliding window filter structure for the humidity
     sensor */
  static s_osw humidity_osw;
  osw_init(&humidity_osw);

  /* Setup a periodic timer that expires after 1 second */
  etimer_set(&timer, CLOCK_SECOND * 1);

  /* Activate and configure the sht11 sensor*/
  SENSORS_ACTIVATE(sht11_sensor);

  while (1)
  {
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));

    /* Read and store the humidity from the sht11 sensor */
    int humid_val = sht11_sensor.value(SHT11_SENSOR_HUMIDITY);

    /* Convert the humditity to relative humidity (%), and add it into the filter buffer */
    float s = -4 + 0.0405 * humid_val - 2.8e-6 * humid_val * humid_val;
    osw_add(&humidity_osw, s);

    /* Store data in the buffer */
    sensor_buffer[2] = (int)osw_avg(&humidity_osw);
    sensor_buffer[3] = (osw_avg(&humidity_osw) - sensor_buffer[2]) * 100;

    etimer_reset(&timer);
  }

  PROCESS_END();
}
/*---------------------------------------------------------------------------*/
```

```c
/*****************************************************************************
* File name: jerome-udp-server.c
*
* Author: Jerome Samuels-Clarke
* Date: Edited November 2021
*
* Description: Server code for recieving data from a RPL-UDP client node.
*****************************************************************************/
#include "contiki.h"
#include "net/routing/routing.h"
#include "net/netstack.h"
#include "net/ipv6/simple-udp.h"

#include "sys/log.h"
/*****************************************************************************/

#define LOG_MODULE "App"
#define LOG_LEVEL LOG_LEVEL_INFO

#define WITH_SERVER_REPLY 1
#define UDP_CLIENT_PORT 8765
#define UDP_SERVER_PORT 5678

static struct simple_udp_connection udp_conn;

PROCESS(udp_server_process, "UDP server");
AUTOSTART_PROCESSES(&udp_server_process);
/*****************************************************************************/
static void
udp_rx_callback(struct simple_udp_connection *c,
                const uip_ipaddr_t *sender_addr,
                uint16_t sender_port,
                const uip_ipaddr_t *receiver_addr,
                uint16_t receiver_port,
                const uint8_t *data,
                uint16_t datalen)
{
  LOG_INFO("Received request '%.*s' from ", datalen, (char *)data);
  LOG_INFO_6ADDR(sender_addr);
  LOG_INFO_("\n");
}
/*****************************************************************************/
PROCESS_THREAD(udp_server_process, ev, data)
{
  PROCESS_BEGIN();

  /* Initialize DAG root */
  NETSTACK_ROUTING.root_start();

  /* Initialize UDP connection */
  simple_udp_register(&udp_conn, UDP_SERVER_PORT, NULL,
                      UDP_CLIENT_PORT, udp_rx_callback);

  PROCESS_END();
}
/*****************************************************************************/
```

Temperature and Humidity Wireless Sensor Network

Coventry University



System Features

- Power - The mote can either be powered from a two 1.5V AA batteries, or from the USB connector. This provides a flexible solution as wireless motes can use the batteries, whereas a fixed mote can be powered via a connected PC.
- Sensors - The designed system uses a SHT11 sensor, that can read both temperature and humidity (mention the resolution). There is also a light sensor on the mote, and GPIO connectors for adding additional sensors.
- Communication - The IP protocol used to communicate between two (or more) motes is RPL-UDP. This allows for one mote to act as a server and another (or multiple) to be clients.
- Contiki - An embedded operating system allows for concurrency between different thread. In particular Contiki is also an embedded operating system design for low powered wireless devices.

Practical Use

Monitoring temperature and humidity can be used in a greenhouse. Multiple motes can be placed in around the greenhouse, depending on the size of the building. Sensor placement is important, as false readings will reduce the validity of the data. Direct sunlight or placing the sensors near equipment emitting heat, will also not give an accurate representation of the readings.

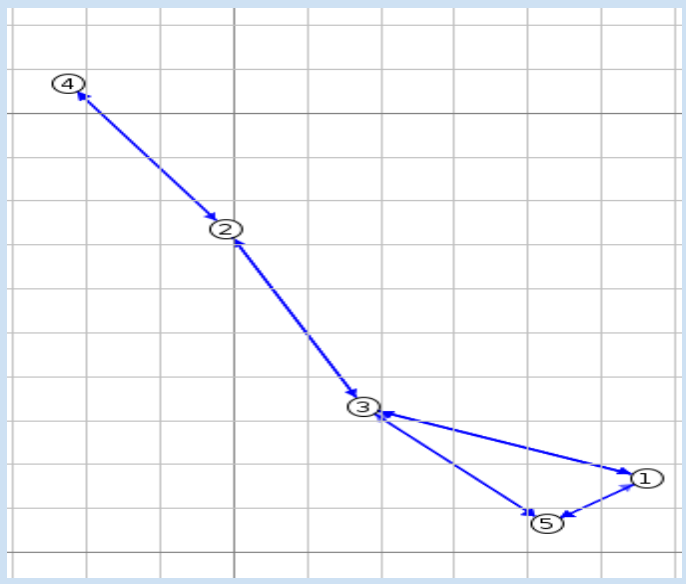
Collecting and processing the data can be done with a client/server approach. Each of the clients can be powered with batteries, collect and process the data before send to the sever periodically. The server can then store and publish the data.

The motes can run off the batteries for years. This reduces the maintenance as they don't need to be constantly monitored. If a single mote stopped working, it should be relatively easy to locate and fix the problem.

System Evaluation

Doing simulations using Cooja showed that the motes can communicate with one another, using an RPL-UDP connection. The simulation also made the connection between the motes instantons. However, in practice connecting between modules provided difficulty. Trying to scale the system up to use more motes, can amplify this problem and make the system unreliable. If the connection is weak, once the system is in the field considerable time will need to be taken to debug the problems.

A simulation was also done with one server and four clients. Each of the clients request to send data to the server, even if not directly within range. Motes that are not in range of the server can forward their data to another node via a technique called multi-hopping.



Design Process Reflection

Unit testing was done throughout the development. The benefit of using a test driven approach is that software can be developed without the need of hardware, as well as significantly reducing the number of errors by testing multiple cases. For example multiple test cases were done when designing the Overlapping Sliding Window filter. Doing the test driven approach can reduce design time, however time can be wasted figuring out why the test cases are failing, even if the test should pass.

```
Unit test: Overlapping Sliding Window Filter Average
Result: success
Exit point: jerome-calc-test.c:86
Start: 777334
End: 777334
Duration: 0
Ticks per second: 1000
```

The terminal output shows a server mote receiving temperature and humidity values from a client using RPL-UDP.

```
vagrant@ubuntu-bionic: ~/contiki-ng/rpl-udp-example-cus
inal  Help
Received request 'T: 26.92C, H: 45.74%' from fd00::212:7400:13cb:82
Received request 'T: 26.92C, H: 45.57%' from fd00::212:7400:13cb:82
Received request 'T: 26.91C, H: 45.35%' from fd00::212:7400:13cb:82
Received request 'T: 26.91C, H: 45.10%' from fd00::212:7400:13cb:82
Received request 'T: 26.91C, H: 44.85%' from fd00::212:7400:13cb:82
Received request 'T: 26.90C, H: 44.61%' from fd00::212:7400:13cb:82
Received request 'T: 26.90C, H: 44.39%' from fd00::212:7400:13cb:82
Received request 'T: 26.90C, H: 44.20%' from fd00::212:7400:13cb:82
Received request 'T: 26.90C, H: 44.5%' from fd00::212:7400:13cb:82
Received request 'T: 26.90C, H: 43.92%' from fd00::212:7400:13cb:82
Received request 'T: 26.91C, H: 43.82%' from fd00::212:7400:13cb:82
Received request 'T: 26.91C, H: 43.72%' from fd00::212:7400:13cb:82
Received request 'T: 26.92C, H: 43.65%' from fd00::212:7400:13cb:82
Received request 'T: 26.92C, H: 43.59%' from fd00::212:7400:13cb:82
Received request 'T: 26.93C, H: 43.53%' from fd00::212:7400:13cb:82
Received request 'T: 26.94C, H: 43.48%' from fd00::212:7400:13cb:82
```

By: Jerome Samuels-Clarke 10473050

Video Submission Link

https://livecoventryac-my.sharepoint.com/:v:/g/personal/clarkej12_uni_coventry_ac_uk/ESUnEdxiAfBMr-Ezi7BohrQBi8Tkj5tEVysxey2IaXATrQ?e=E2KJHH