

INTERNET OF THINGS

LAB PRACTICAL



Submitted To:

Prof. P.K.Mishra
Professor,
Department of Computer Science,
Banaras Hindu University.

Submitted By:

Shubhankar Chaudhary
Research Scholar
Roll No: 21886SC014

TABLE OF CONTENTS

| <u>S.No.</u> | <u>Content</u> | <u>Page No.</u> |
|---------------------|---|------------------------|
| 1. | Creating multiple sky motes and log their messages to their console in Cooja Simulator. | |
| 2. | Simulating message unicasting between 7 IoT devices. | |
| 3. | Simulating message broadcasting between 7 IoT devices. | |
| 4. | Write a program to implement multi-threading using Skymote in Cooja Simulator | |
| 5. | How to use collect-view shell in Cooja Simulator. | |

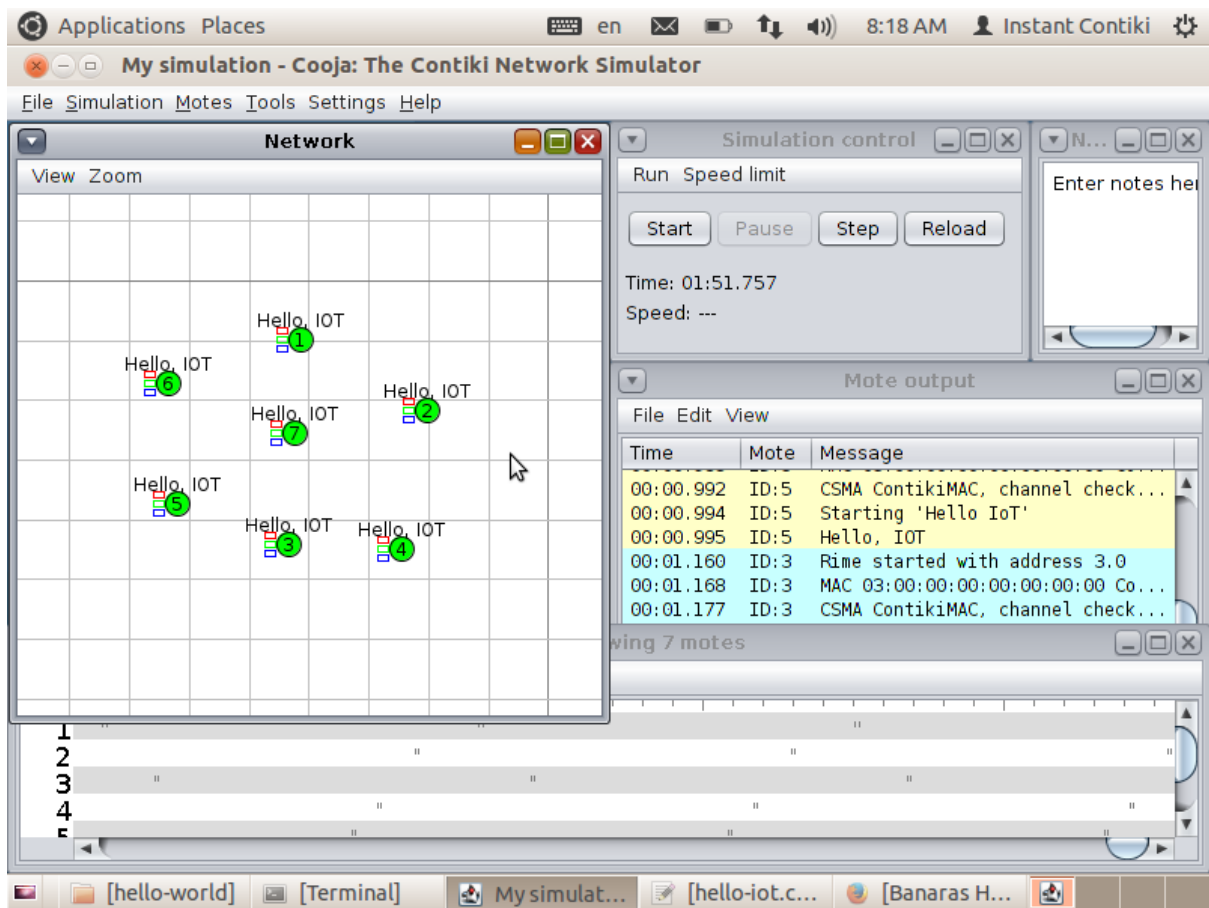
PRACTICAL – 1

Creating multiple sky motes and log their messages to their console in Cooja Simulator.

Code:

```
#include "contiki.h"
#include <stdio.h>
PROCESS(HelloIoT, "Hello IoT");
AUTOSTART_PROCESSES(&HelloIoT);
PROCESS_THREAD(HelloIoT, ev, data)
{
    PROCESS_BEGIN();
    printf("Hello, IOT\n");
    PROCESS_END();
}
```

Output:



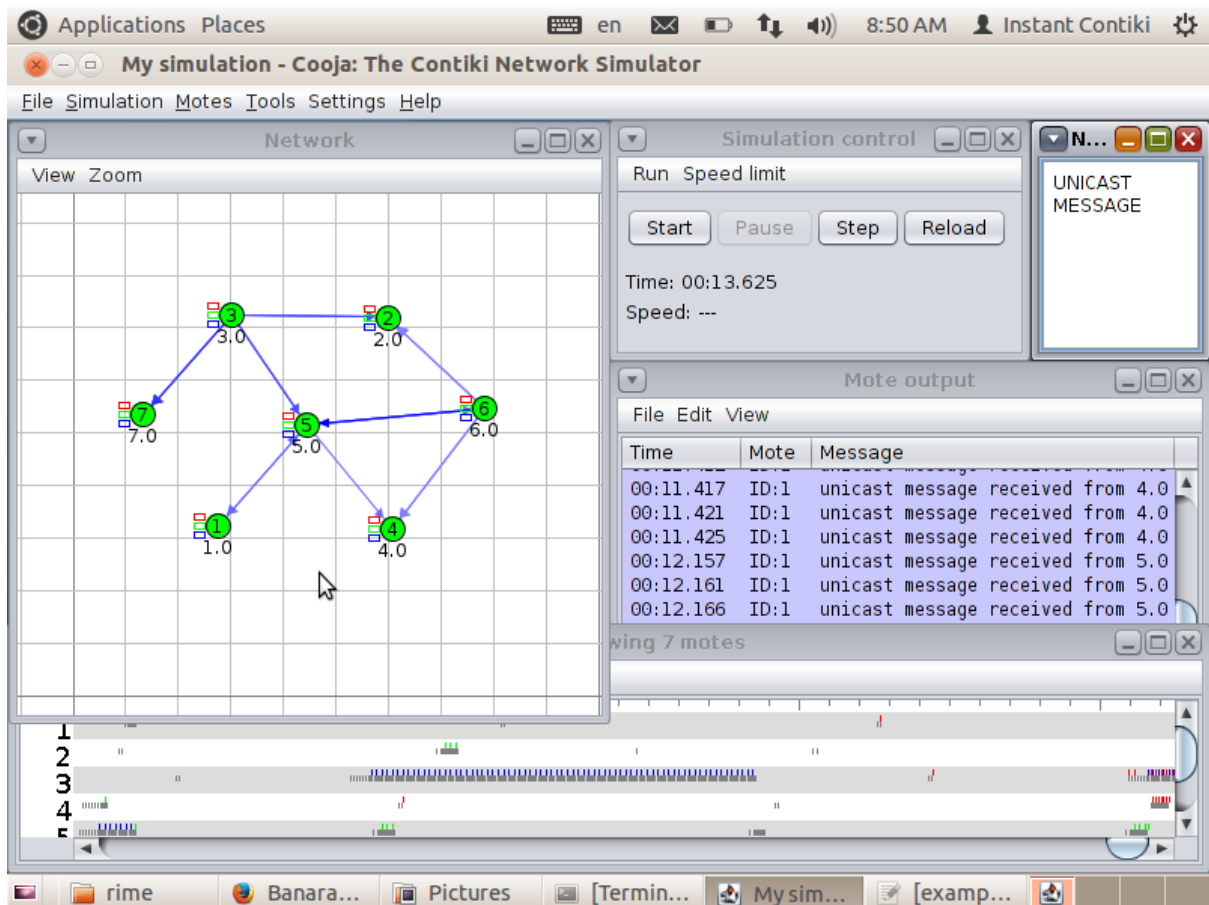
PRACTICAL – 2

Simulating message unicasting between 7 IoT devices.

Code:

```
#include "contiki.h"
#include "net/rime.h"
#include "dev/button-sensor.h"
#include "dev/leds.h"
#include <stdio.h>
PROCESS(example_unicast_process, "Example unicast");
AUTOSTART_PROCESSES(&example_unicast_process);
static void recv_uc(struct unicast_conn *c, const rimeaddr_t *from)
{
    printf("unicast message received from %d.%d\n",
        from->u8[0], from->u8[1]);
}
static const struct unicast_callbacks unicast_callbacks = {recv_uc};
static struct unicast_conn uc;
PROCESS_THREAD(example_unicast_process, ev, data)
{
    PROCESS_EXITHANDLER(unicast_close(&uc);)
    PROCESS_BEGIN();
    unicast_open(&uc, 146, &unicast_callbacks);
    while(1)
    {
        static struct etimer et;
        rimeaddr_t addr;
        etimer_set(&et, CLOCK_SECOND);
        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
        packetbuf_copyfrom("Hello", 5);
        addr.u8[0] = 1;
        addr.u8[1] = 0;
        if(!rimeaddr_cmp(&addr, &rimeaddr_node_addr))
        {
            unicast_send(&uc, &addr);
        }
    }
    PROCESS_END();
}
```

Output:



PRACTICAL – 3

Simulating message broadcasting between 7 IoT devices.

Code:

```
#include "contiki.h"
#include "net/rime.h"
#include "random.h"
#include "dev/button-sensor.h"
#include "dev/leds.h"
#include <stdio.h>
PROCESS(example_broadcast_process, "Broadcast example");
AUTOSTART_PROCESSES(&example_broadcast_process);
static void broadcast_rcv(struct broadcast_conn *c, const rimeaddr_t *from)
{
    printf("broadcast message received from %d.%d: '%s'\n",
        from->u8[0], from->u8[1], (char *)packetbuf_dataptr());
}
static const struct broadcast_callbacks broadcast_call = {broadcast_rcv};
static struct broadcast_conn broadcast;
PROCESS_THREAD(example_broadcast_process, ev, data)
{
    static struct etimer et;
    PROCESS_EXITHANDLER(broadcast_close(&broadcast));
    PROCESS_BEGIN();
    broadcast_open(&broadcast, 129, &broadcast_call);
    while(1)
    {
        etimer_set(&et, CLOCK_SECOND * 4 + random_rand() % (CLOCK_SECOND * 4));
        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
        packetbuf_copyfrom("Hello", 6);
        broadcast_send(&broadcast);
        printf("broadcast message sent\n");
    }
    PROCESS_END();
}
```

Output:

The screenshot displays the Cooja network simulator interface. The main window is titled "My simulation - Cooja: The Contiki Network Simulator". It features a menu bar with "File", "Simulation", "Notes", "Tools", "Settings", and "Help".

The central "Network" panel shows a grid-based topology with seven nodes (ID 1-7) connected by blue lines. Node 1 is at the bottom right, connected to nodes 2, 3, and 7. Node 2 is connected to node 3. Node 3 is connected to nodes 4 and 5. Node 4 is connected to node 5. Node 5 is connected to node 6. Node 6 is connected to node 7.

The "Simulation control" panel on the right includes buttons for "Start", "Pause", "Step", and "Reload". It displays the current time as "01:01.705" and the speed as "---".

The "Mote output" panel shows a log of messages. The log is as follows:

| Time | Mote | Message |
|-----------|------|-----------------------------------|
| 01:01.153 | ID:6 | broadcast message received fro... |
| 01:01.178 | ID:7 | broadcast message received fro... |
| 01:01.204 | ID:3 | broadcast message sent |
| 01:01.243 | ID:5 | broadcast message received fro... |
| 01:01.249 | ID:4 | broadcast message received fro... |
| 01:01.263 | ID:2 | broadcast message received fro... |

The "Timeline showing 7 motes" panel at the bottom displays a timeline for each node, with a play button and a zoom slider.

The bottom of the screen shows a taskbar with several open applications: "[rime]", "[Banaras H...", "[Pictures]", "Terminal", and "My simulat...".

PRACTICAL – 4

Write a program to implement multi-threading using Skymote in Cooja Simulator.

Code:

```
#include "contiki.h"
#include "sys/mt.h"
#include <stdio.h>
static char *ptr;
PROCESS(multi_threading_process, "Multi-threading process");
AUTOSTART_PROCESSES(&multi_threading_process);
static void thread_func(char *str, int len)
{
    ptr = str + len;
    mt_yield();
    if(len)
    {
        thread_func(str, len - 1);
        mt_yield();
    }
    ptr = str + len;
}
static void thread_main(void *data)
{
    while(1)
    {
        thread_func((char *)data, 9);
    }
    mt_exit();
}
PROCESS_THREAD(multi_threading_process, ev, data)
{
    static struct mt_thread alpha_thread;
    static struct mt_thread count_thread;
    static struct etimer timer;
    static int toggle;
    PROCESS_BEGIN();
    mt_init();
    mt_start(&alpha_thread, thread_main, "JIHGFEDCBA");
    mt_start(&count_thread, thread_main, "9876543210");
    etimer_set(&timer, CLOCK_SECOND / 2);
    while(1)
    {
        PROCESS_WAIT_EVENT();
        if(ev == PROCESS_EVENT_TIMER)
        {
            if(toggle)
            {
```

```

    mt_exec(&alpha_thread);
    toggle--;
}
else
{
    mt_exec(&count_thread);
    toggle++;
}
puts(ptr);
etimer_set(&timer, CLOCK_SECOND/2);
}
}
mt_stop(&alpha_thread);
mt_stop(&count_thread);
mt_remove();
PROCESS_END();
}

```

Output:

The screenshot displays the Cooja: The Contiki Network Simulator interface. The main window shows a network topology with five nodes, each represented by a small icon and labeled with a unique ID and a sequence of characters (e.g., JIHGFEDCBA, 9876543210). The nodes are connected by lines, forming a network structure. The interface includes a 'Simulation control' panel with buttons for 'Start', 'Pause', 'Step', and 'Reload', and a 'Mote output' panel showing a log of messages.

The 'Mote output' panel displays the following log entries:

| Time | Mote | Message |
|-----------|------|------------|
| 00:12.650 | ID:4 | JIHGFEDCBA |
| 00:12.664 | ID:1 | JIHGFEDCBA |
| 00:12.681 | ID:3 | 876543210 |
| 00:12.997 | ID:5 | JIHGFEDCBA |
| 00:13.018 | ID:2 | 76543210 |
| 00:13.130 | ID:4 | 76543210 |
| 00:13.164 | ID:1 | 76543210 |
| 00:13.181 | ID:3 | JIHGFEDCBA |
| 00:13.497 | ID:5 | 76543210 |

PRACTICAL – 5

How to use collect-view shell in Cooja Simulator.

Code:

```
#include "contiki.h"
#include "shell.h"
#include "serial-shell.h"
#include "collect-view.h"
#define WITH_COFFEE 0
PROCESS(collect_view_shell_process, "Contiki Collect View Shell");
AUTOSTART_PROCESSES(&collect_view_shell_process);
PROCESS_THREAD(collect_view_shell_process, ev, data)
{
    PROCESS_BEGIN();
    serial_shell_init();
    shell_blink_init();
    #if WITH_COFFEE
        shell_file_init();
        shell_coffee_init();
    #endif
    shell_reboot_init();
    shell_rime_init();
    shell_rime_netcmd_init();
    shell_powertrace_init();
    shell_text_init();
    shell_time_init();
    #if CONTIKI_TARGET_SKY
        shell_sky_init();
    #endif
    shell_collect_view_init();
    PROCESS_END();
}
```

Output:

The screenshot displays the Cooja: The Contiki Network Simulator interface. The main window is titled "My simulation - Cooja: The Contiki Network Simulator". It features a menu bar with "File", "Simulation", "Notes", "Tools", "Settings", and "Help".

The central "Network" panel shows a grid-based topology with five nodes, each represented by a small icon and labeled with an IP address and "Contiki>":

- 4.0: Contiki> (top center)
- 2.0: Contiki> (middle left)
- 1.0: Contiki> (middle right)
- 5.0: Contiki> (bottom left)
- 3.0: Contiki> (bottom right)

Connections are indicated by lines between the nodes. The "Simulation control" panel on the right includes buttons for "Run", "Speed limit", "Start", "Pause", "Step", and "Reload". It also displays the current "Time: 00:24.009" and "Speed: ---".

The "Mote output" panel at the bottom right shows a log of messages:

| Time | Mote | Message |
|-----------|------|-----------------------------------|
| 00:01.008 | ID:5 | CSMA ContikiMAC, channel check... |
| 00:01.011 | ID:5 | Starting 'Contiki Collect View... |
| 00:01.020 | ID:5 | 5.0: Contiki> |
| 00:01.175 | ID:3 | Rime started with address 3.0 |
| 00:01.183 | ID:3 | MAC 03:00:00:00:00:00:00:00 Co... |
| 00:01.192 | ID:3 | CSMA ContikiMAC, channel check... |

The bottom status bar shows the title "My simulation - Cooja: The Contiki Network Simulator". The taskbar at the very bottom includes icons for "Banaras Hindu Univer...", "[Terminal]", and "My simulation - Cooja:..."