

Application Source Code:

- This is outside project source to build and deploy to the devices.

AP (Orion Border Router)

Makefile

- contain project configurations, such as which module will be used and the config parameter.
- CONTIKI_PROJECT : project name of **source.c** file
- MODULES: Add module for project (Ex: Orchestra, CoAP, ...)
- MAKE_ROUTING: Config for Routing Protocol
- MAKE_NET: Config for Network Layer
- MAKE_MAC: Config for Mac Layer
- CONTIKI: path to source code of Contiki Project

```
1 CONTIKI_PROJECT = ap
2 all:${CONTIKI_PROJECT}
3 MODULES += os/services/orchestra
4
5 DEFINES += CNLAB_CONF_PROTOCOL_ENABLE
6 MAKE_ROUTING = MAKE_ROUTING_R_RPT
7 MAKE_NET = MAKE_NET_NULLNET
8 MAKE_MAC = MAKE_MAC_TSCH
9
10 CONTIKI = /home/huynn/CNLAB/CNLAB_PROTOCOL_DOWNLINK/contiki-ng
11 include ${CONTIKI}/Makefile.include
```

ap.c

- contain AP Application source code.

```

1  #include "contiki.h"
2  #include "net/mac/tsch/tsch.h"
3  #include "net/netstack.h"
4  #include "node-id.h"
5  #include "dev/button-hal.h"
6  #include "leds.h"
7  #include "dev/serial-line.h"
8  #include "sys/log.h"
9  #include <string.h>
10
11 #define LOG_MODULE "CNLAB_NODE"
12 #define LOG_LEVEL LOG_LEVEL_INFO
13
14 static button_hal_button_t *button;
15 char uart_buffer[128];
16 PROCESS(node_process, "Node Process");
17 AUTOSTART_PROCESSES(&node_process);
18
19 PROCESS_THREAD(node_process, ev, data)
20 {
21     PROCESS_BEGIN();
22     while (1)
23     {
24         PROCESS_WAIT_EVENT();
25         if(ev == button_hal_press_event) {
26             button = (button_hal_button_t *)data;
27             LOG_INFO("%s Pressed\r\n", BUTTON_HAL_GET_DESCRIPTION(button));
28             leds_toggle(LEDS_GREEN);
29             NETSTACK_MAC.on();
30         } else if(ev == button_hal_release_event){
31             button = (button_hal_button_t *)data;
32             LOG_INFO("%s Released\r\n", BUTTON_HAL_GET_DESCRIPTION(button));
33             break;
34         }
35     }
36     LOG_INFO("WAITING 'START' CMD FROM SERVER!!!\r\n");
37     memset(&uart_buffer, 0, sizeof(uart_buffer));
38     while (1)
39     {
40         PROCESS_YIELD();
41         if(ev == serial_line_event_message) {
42             int size = snprintf(uart_buffer, sizeof(uart_buffer), "%s", (char *)data);
43             LOG_INFO("Message with size %d received from UART: '%s'\r\n", size, (char *)data);
44             if(strstr(uart_buffer, "START") != NULL){
45                 LOG_INFO("Node %d run as Coordinator\r\n", node_id);
46                 tsch_set_coordinator(1);
47                 PROCESS_EXIT();
48             }
49         }
50     }
51     PROCESS_END();
52 }

```

Sensor Node (Zolertia RE Mote)

Makefile

- contain project configurations, such as which module will be used and the config parameter.

```

1  CONTIKI_PROJECT = node node-sht
2  all:$(CONTIKI_PROJECT)
3  MODULES += os/services/orchestra
4  CONTIKI_TARGET_SOURCEFILES += sht25.c
5  DEFINES += CNLAB_CONF_PROTOCOL_ENABLE
6  DEFINES += SD_CARD_ENABLE
7  MAKE_ROUTING = MAKE_ROUTING_R_RPT
8  MAKE_NET = MAKE_NET_NULLNET
9  MAKE_MAC = MAKE_MAC_TSCH
10
11  CONTIKI = /home/huynn/CNLAB/CNLAB_PROTOCOL_DOWNLINK/contiki-ng
12  include $(CONTIKI)/Makefile.include
13

```

node.c

- contain Sensor Node Application source code with DTPS-I2C sensor.

```

1  #include "contiki.h"
2  #include "net/mac/tsch/tsch.h"
3  #include "net/netstack.h"
4  #include "node-id.h"
5  #include "dev/button-hal.h"
6  #include "leds.h"
7  #include "dev/serial-line.h"
8  #include "dev/i2c.h"
9  #include "sys/log.h"
10 #include <string.h>
11
12 #define LOG_MODULE "CNLAB_NODE"
13 #define LOG_LEVEL LOG_LEVEL_INFO
14 #define BUTTON_PRESSED 1
15
16 static button_hal_button_t *button;
17 PROCESS(node_process, "Node Process");
18 PROCESS(node_send_event_process, "send Event Packet");
19 AUTOSTART_PROCESSES(&node_process);
20
21 PROCESS_THREAD(node_send_event_process, ev, data)
22 {
23     PROCESS_BEGIN();
24     while (1)
25     {
26         PROCESS_WAIT_EVENT();
27         if(ev == button_hal_press_event) {

```

```

30     button = (button_hal_button_t *)data;
31     LOG_INFO("%s Pressed\r\n",BUTTON_HAL_GET_DESCRIPTION(button));
32     if(cnlab_current_state == CNLAB_PROTOCOL_SEND_DATA){
33         cnlab_aperidic_data_output(BUTTON_PRESSED);
34     }
35 }
36 }
37 PROCESS_END();
38 }
39 |
40 PROCESS_THREAD(node_process, ev, data)
41 {
42     float SensorTemp = 0;
43     //float ObjTemp = 0;
44     uint8_t buf[6];
45     uint16_t raw_temp[2];
46     PROCESS_BEGIN();
47     clock_delay(1000000);
48     i2c_init(I2C_SDA_PORT, I2C_SDA_PIN, I2C_SCL_PORT, I2C_SCL_PIN, I2C_SCL_NORMAL_BUS_SPEED);
49     while (1)
50     {
51         PROCESS_WAIT_EVENT();
52         if(ev == button_hal_press_event) {
53             button = (button_hal_button_t *)data;
54             LOG_INFO("%s Pressed\r\n",BUTTON_HAL_GET_DESCRIPTION(button));
55             leds_toggle(LED_GREEN);
56             NETSTACK_MAC.on();
57         }else if(ev == button_hal_release_event){
58             button = (button_hal_button_t *)data;
59             LOG_INFO("%s Released\r\n",BUTTON_HAL_GET_DESCRIPTION(button));
60             break;
61         }
62     }
63     process_start(&node_send_event_process, NULL);
64     while (1){
65         PROCESS_WAIT_EVENT_UNTIL(ev == send_data_event && data != NULL);
66         if(i2c_single_send(0x10, 0x80) == I2C_MASTER_ERR_NONE) {
67             if(i2c_burst_receive(0x10, buf, 6) == I2C_MASTER_ERR_NONE) {
68                 raw_temp[0] = buf[0] + buf[1]*256 + buf[2]*65536;
69                 raw_temp[1] = buf[3] + buf[4]*256 + buf[5]*65536;
70                 if(buf[2] < 0x80){
71                     SensorTemp = (float)raw_temp[0] / 200.0;
72                 }else{
73                     SensorTemp = ( (float)(raw_temp[0]-16777216) ) / 200.0;
74                 }
75                 /*if(buf[5] < 0x80){
76                     ObjTemp = (float)raw_temp[1] / 200.0;
77                 }else{
78                     ObjTemp = ( (float)(raw_temp[1]-16777216) ) / 200.0;
79                 }*/
80             }
81         }
82         uint16_t *seqno = (uint16_t *)data;
83         cnlab_periodic_data_output(*seqno, (uint8_t)SensorTemp);
84     }
85     PROCESS_END();
86 }

```

node-sht.c

- contain Sensor Node Application source code with SHT sensor.

```

1  #include "contiki.h"
2  #include "net/mac/tsch/tsch.h"
3  #include "net/netstack.h"
4  #include "node-id.h"
5  #include "dev/button-hal.h"
6  #include "leds.h"
7  #include "dev/serial-line.h"
8  #include "dev/sht25.h"
9  #include "sys/log.h"
10 #include <string.h>
11
12 #define LOG_MODULE "CNLAB_NODE"
13 #define LOG_LEVEL LOG_LEVEL_INFO
14 #define BUTTON_PRESSED 1
15
16 static button_hal_button_t *button;
17 PROCESS(node_process, "Node Process");
18 PROCESS(node_send_event_process, "send Event Packet");
19 AUTOSTART_PROCESSES(&node_process);
20
21
22 PROCESS_THREAD(node_send_event_process, ev, data)
23 {
24     PROCESS_BEGIN();
25     while (1)
26     {
27         PROCESS_WAIT_EVENT();
28         if(ev == button_hal_press_event) {
29             button = (button_hal_button_t *)data;
30             LOG_INFO("%s Pressed\r\n",BUTTON_HAL_GET_DESCRIPTION(button));
31             if(cnlab_current_state == CNLAB_PROTOCOL_SEND_DATA){
32                 cnlab_aperidic_data_output(BUTTON_PRESSED);
33             }
34         }
35     }
36     PROCESS_END();
37 }
38
39
40 PROCESS_THREAD(node_process, ev, data)
41 {
42     int16_t temperature;
43     PROCESS_BEGIN();
44     SENSORS_ACTIVATE(sht25);
45     while (1)
46     {
47         PROCESS_WAIT_EVENT();
48         if(ev == button_hal_press_event) {
49             button = (button_hal_button_t *)data;
50             LOG_INFO("%s Pressed\r\n",BUTTON_HAL_GET_DESCRIPTION(button));
51             leds_toggle(LED_GREEN);
52             NETSTACK_MAC.on();
53         }else if(ev == button_hal_release_event){
54             button = (button_hal_button_t *)data;
55             LOG_INFO("%s Released\r\n",BUTTON_HAL_GET_DESCRIPTION(button));
56             break;
57         }
58     }
59     process_start(&node_send_event_process, NULL);

```

```

60     if(sht25.value(SHT25_VOLTAGE_ALARM)) {
61         LOG_INFO("Voltage is lower than recommended for the sensor operation\n");
62         PROCESS_EXIT();
63     }
64     sht25.configure(SHT25_RESOLUTION, SHT2X_RES_14T_12RH);
65     while (1){
66         PROCESS_WAIT_EVENT_UNTIL(ev == send_data_event && data != NULL);
67         temperature = sht25.value(SHT25_VAL_TEMP);
68         uint16_t *seqno = (uint16_t *)data;
69         cnlab_periodic_data_output(*seqno, (uint8_t)(temperature / 100));
70     }
71     PROCESS_END();
72 }

```

Contiki-NG CNLAB Protocol Implementation Source Code:

- Implement CNLAB Protocol base on current Contiki-NG open source code.
- Supported standard protocols, such as IPv6/6LoWPAN, 6TiSCH, RPL, CoAP, and Orchestra services, ...

Handle NodeID to make unique node id.

[contiki-ng/os/sys/node-id.h](#)

- Contain node_id global variables to handle the node ID

```

46 #ifndef NODE_ID_H_
47 #define NODE_ID_H_
48
49 /* Total number of nodes
50  *
51  * Author: Huy
52  * Date: 2020/05/15
53  */
54 #define NUMBER_OF_NODES 5
55 /* Type of nodes
56  *
57  * Author: Huy
58  * Date: 2020/08/08
59  */
60 enum CNLAB_NODE_TYPES {
61     AP = 1,
62     SENSOR = 2,
63     ACTUATOR = 3,
64     ROUTER = 4
65 };
66
67 /* A global variable that hosts the node ID */
68 extern uint8_t node_id;
69 extern uint8_t cnlab_node_type;
70 /**
71  * Initialize the node ID. Must be called after initialized of linkaddr
72  */
73 void node_id_init(void);

```

```

74  /* Initialize the node ID. Set up manually
75  *
76  * Author: Huy
77  * Date: 2019/09/20
78  */
79  void set_node_id(uint8_t id);
80  /* Set the node types
81  *
82  * Author: Huy
83  * Date: 2020/08/08
84  */
85  void set_node_type(uint8_t type);
86  #endif /* NODE_ID_H */
87  /**
88  * @}
89  * @}
90  */
91

```

contiki-ng/os/sys/node-id.c

```

40  #include "contiki.h"
41  #include "sys/node-id.h"
42  #include "net/linkaddr.h"
43  #include "services/deployment/deployment.h"
44
45  uint8_t node_id = 0;
46  uint8_t cnlab_node_type = 0;
47
48  void
49  node_id_init(void) {
50  #if BUILD_WITH_DEPLOYMENT
51  | deployment_init();
52  #else /* BUILD_WITH_DEPLOYMENT */
53  | /* Initialize with a default value derived from linkaddr */
54  | node_id = linkaddr_node_addr.u8[LINKADDR_SIZE - 1]
55  | | | | + (linkaddr_node_addr.u8[LINKADDR_SIZE - 2] << 8);
56  #endif /* BUILD_WITH_DEPLOYMENT */
57  }
58  /* Initialize the node ID. Set up manually
59  *
60  * Author: Huy
61  * Date: 2019/09/20
62  */
63  void
64  set_node_id(uint8_t id) {
65  | node_id = id;
66  }
67
68  /* Initialize the node type. Set up manually
69  *
70  * Author: Huy
71  * Date: 2020/08/08
72  */
73  void
74  set_node_type(uint8_t type) {
75  | cnlab_node_type = type;
76  }

```

contiki-ng/os/contiki-main.c

- Contain init Contiki-NG OS libraries, init platform, and handle main loop of Contiki-NG processes.

```

59 #include <stdio.h>
60 #include <stdint.h>
61 /*-----*/
62 /* Configuration for each node.
63  * Node ID is integer number start from 1.
64  * Node Type is AP, SENSOR, ACTUATOR, ROUTER
65  * Date: 2020/08/09
66  * Author: Huy
67  */
68 #define NODE_ID 1
69 #define NODE_TYPE AP
70 /* Log configuration */
71 #include "sys/log.h"
72 #define LOG_MODULE "Main"
73 #define LOG_LEVEL LOG_LEVEL_MAIN
74 /*-----*/
75 int
76 #if PLATFORM_MAIN_ACCEPTS_ARGS
77 main(int argc, char **argv)
78 {
79     platform_process_args(argc, argv);
80 #else
81 main(void)
82 {
83 #endif
84     platform_init_stage_one();
85
86     clock_init();
87     rtimer_init();
88     process_init();
89     process_start(&etimer_process, NULL);
90     ctimer_init();
91     watchdog_init();
92
93     energest_init();
94
95 #if STACK_CHECK_ENABLED
96     stack_check_init();
97 #endif
98
99     platform_init_stage_two();
100
101 #if QUEUEBUF_ENABLED
102     queuebuf_init();
103 #endif /* QUEUEBUF_ENABLED */
104
105 /* Comment out unused init node id from lladdr.
106  * Set up manually in #define NODE_ID, NODE_TYPE.
107  *
108  * Author: Huy
109  * Date: 2020/08/08
110  */
111     // node_id_init();
112     set_node_id(NODE_ID);
113     set_node_type(NODE_TYPE);
114     netstack_init();
115
116     LOG_INFO("Starting " CONTIKI_VERSION_STRING "\r\n");
117     LOG_INFO("- Routing: %s\r\n", NETSTACK_ROUTING.name);

```



```

118 LOG_INFO("- Net: %s\r\n", NETSTACK_NETWORK.name);
119 LOG_INFO("- PHY: IEEE 802.15.4\r\n");
120 LOG_INFO("- MAC: %s\r\n", NETSTACK_MAC.name);
121 LOG_INFO("- 802.15.4 PANID: 0x%04x\r\n", IEEE802154_PANID);
122 #if MAC_CONF_WITH_TSCH
123 LOG_INFO("- 802.15.4 TSCH channel hopping sequence length: %u\r\n", (unsigned)sizeof(TSCH_D
124 LOG_INFO("- 802.15.4 TSCH default hopping sequence: [ ");
125 for(uint8_t hop_index = 0; hop_index < (unsigned)sizeof(TSCH_DEFAULT_HOPPING_SEQUENCE); hop
126 | LOG_INFO("%u ", TSCH_DEFAULT_HOPPING_SEQUENCE[hop_index]);
127 | }
128 | LOG_INFO("]\r\n");
129 #else /* MAC_CONF_WITH_TSCH */
130 LOG_INFO("- 802.15.4 Default channel: %u\r\n", IEEE802154_DEFAULT_CHANNEL);
131 #endif /* MAC_CONF_WITH_TSCH */
132 LOG_INFO("Node ID: %u Node Type: %u\r\n", node_id, cnlab_node_type);
133 LOG_INFO("Link-layer address: ");
134 LOG_INFO_LLADDR(&linkaddr_node_addr);
135 LOG_INFO("\r\n");
136
137 #if NETSTACK_CONF_WITH_IPV6
138 {
139 | uip_ds6_addr_t *lladdr;
140 | memcpy(&uip_lladdr.addr, &linkaddr_node_addr, sizeof(uip_lladdr.addr));
141 | process_start(&tcpip_process, NULL);
142 |
143 | lladdr = uip_ds6_get_link_local(-1);
144 | LOG_INFO("Tentative link-local IPv6 address: ");
145 | LOG_INFO_6ADDR(lladdr != NULL ? &lladdr->ipaddr : NULL);
146 | LOG_INFO("\r\n");
147 | platform_init_stage_three();
148 |
149 | #if BUILD_WITH_RPL_BORDER_ROUTER
150 | rpl_border_router_init();
151 | LOG_DBG("With RPL Border Router\r\n");
152 | #endif /* BUILD_WITH_RPL_BORDER_ROUTER */
153 |
154 | #if BUILD_WITH_ORCHESTRA
155 | orchestra_init();
156 | LOG_DBG("With Orchestra\r\n");
157 | #endif /* BUILD_WITH_ORCHESTRA */
158 |
159 | #if BUILD_WITH_SHELL
160 | serial_shell_init();
161 | LOG_DBG("With Shell\r\n");
162 | #endif /* BUILD_WITH_SHELL */
163 |
164 | #if BUILD_WITH_COAP
165 | coap_engine_init();
166 | LOG_DBG("With CoAP\r\n");
167 | #endif /* BUILD_WITH_COAP */
168 |
169 | #if BUILD_WITH_SIMPLE_ENERGY
170 | simple_energy_init();
171 | #endif /* BUILD_WITH_SIMPLE_ENERGY */
172 |
173 | #if BUILD_WITH_TSCH_CS
174 | /* Initialize the channel selection module */
175 | tsch_cs_adaptations_init();
176 | #endif /* BUILD_WITH_TSCH_CS */

```

```

180
181     autostart_start(autostart_processes);
182
183     watchdog_start();
184
185     #if PLATFORM_PROVIDES_MAIN_LOOP
186         platform_main_loop();
187     #else
188         while(1) {
189             uint8_t r;
190             do {
191                 r = process_run();
192                 watchdog_periodic();
193             } while(r > 0);
194
195             platform_idle();
196         }
197     #endif
198
199     return 0;
200 }

```

Orchestra Services implementation.

[contiki-ng/os/services/orchestra/orchestra-conf.h](#)

```

50  /* Comment out unused ORCHESTRA_RULES
51  * Define our rule : cnlab_eb_period
52  *
53  * Author: Huy
54  * Date: 2019/09/20
55  */
56  #define ORCHESTRA_RULES {&cnlab_eb_period}
57  // #define ORCHESTRA_RULES { &eb_per_time_source, &unicast_per_neighbor_rpl_ns, &default_common
58  /* Example configuration for RPL non-storing mode: */
59  /* #define ORCHESTRA_RULES { &eb_per_time_source, &unicast_per_neighbor_rpl_ns, &default_comm
60
61  #endif /* ORCHESTRA_CONF_RULES */
62
63  /* Define config of Slotframe size. Prime number was preferred.
64  *
65  * Author: Huy
66  * Date: 2020/05/18
67  */
68  #define CNLAB_SLOTFRAME_SIZE 17
69  /* Length of the various slotframes. Tune to balance network capacity,
70  * contention, energy, latency. */
71  #ifdef ORCHESTRA_CONF_EBSF_PERIOD
72  #define ORCHESTRA_EBSF_PERIOD ORCHESTRA_CONF_EBSF_PERIOD
73  #else /* ORCHESTRA_CONF_EBSF_PERIOD */
74  #define ORCHESTRA_EBSF_PERIOD 397
75  #endif /* ORCHESTRA_CONF_EBSF_PERIOD */

```

[contiki-ng/os/services/orchestra/orchestra.h](#)

- contain the define structure, functions name of Orchestra.

```

37
38 #ifndef __ORCHESTRA_H__
39 #define __ORCHESTRA_H__
40
41 #include "net/mac/tsch/tsch.h"
42 #include "orchestra-conf.h"
43
44 /* The structure of an Orchestra rule */
45 struct orchestra_rule {
46     void (* init)(uint16_t slotframe_handle);
47     void (* new_time_source)(const struct tsch_neighbor *old, const struct tsch_neighbor *new);
48     int (* select_packet)(uint16_t *slotframe, uint16_t *timeslot);
49     void (* child_added)(const linkaddr_t *addr);
50     void (* child_removed)(const linkaddr_t *addr);
51 };
52 /* Comment out orchestra rules because we don't use
53  * their rules.
54  * Define our rule : cnlab_eb_period
55  *
56  * Author: Huy
57  * Date: 2019/09/20
58  */
59 // struct orchestra_rule eb_per_time_source;
60 // struct orchestra_rule unicast_per_neighbor_rpl_storing;
61 // struct orchestra_rule unicast_per_neighbor_rpl_ns;
62 // struct orchestra_rule default_common;
63 #if CNLAB_PROTOCOL_ENABLE
64 struct orchestra_rule cnlab_eb_period;
65 #endif /* CNLAB_PROTOCOL_ENABLE */
66 // extern linkaddr_t orchestra_parent_linkaddr;
67 // extern int orchestra_parent_knows_us;
68
69 /* Call from application to start Orchestra */
70 void orchestra_init(void);
71 /* Callbacks required for Orchestra to operate */
72 /* Set with #define TSCH_CALLBACK_PACKET_READY orchestra_callback_packet_ready */
73 void orchestra_callback_packet_ready(void);
74 /* Set with #define TSCH_CALLBACK_NEW_TIME_SOURCE orchestra_callback_new_time_source */
75 void orchestra_callback_new_time_source(const struct tsch_neighbor *old, const struct tsch_neighbor *new);
76 /* Set with #define NETSTACK_CONF_ROUTING_NEIGHBOR_ADDED_CALLBACK orchestra_callback_child_added */
77 void orchestra_callback_child_added(const linkaddr_t *addr);
78 /* Set with #define NETSTACK_CONF_ROUTING_NEIGHBOR_REMOVED_CALLBACK orchestra_callback_child_removed */
79 void orchestra_callback_child_removed(const linkaddr_t *addr);
80

```

[contiki-ng/os/services/orchestra/orchestra.c](https://contiki-ng.org/os/services/orchestra/orchestra.c)

- contain the implementation of Orchestra: an autonomous scheduler.

```

39 #include "contiki.h"
40 #include "orchestra.h"
41 #include "net/packetbuf.h"
42 /* Comment out unused code : IPv6, RPL, Network Sniffer
43  *
44  * Author: Huy
45  * Date: 2019/09/20
46  */
47 // #include "net/ipv6/uiplib.h"
48 // #include "net/routing/routing.h"
49 // #if ROUTING_CONF_RPL_LITE
50 // #include "net/routing/rpl-lite/rpl.h"
51 // #elif ROUTING_CONF_RPL_CLASSIC
52 // #include "net/routing/rpl-classic/rpl.h"
53 // #endif
54
55 // #define DEBUG DEBUG_PRINT
56 // #include "net/ipv6/uiplib-debug.h"
57
58 /* A net-layer sniffer for packets sent and received */
59 // static void orchestra_packet_received(void);
60 // static void orchestra_packet_sent(int mac_status);
61 // NETSTACK_SNIFFER(orchestra_sniffer, orchestra_packet_received, orchestra_packet_sent);
62
63 /* The current RPL preferred parent's link-layer address */
64 // linkaddr_t orchestra_parent_linkaddr;
65 /* Set to one only after getting an ACK for a DAO sent to our preferred parent */
66 // int orchestra_parent_knows_us = 0;
67
68 /* The set of Orchestra rules in use */
69 const struct orchestra_rule *all_rules[] = ORCHESTRA_RULES;
70 #define NUM_RULES (sizeof(all_rules) / sizeof(struct orchestra_rule *))
71 /* Comment out unused functions
72  *
73  * Author: Huy
74  * Date: 2019/09/20
75  */
76 /*-----*/
77 // static void
78 // orchestra_packet_received(void)
79 // {
80 // }
81 /*-----*/
82 // static void
83 // orchestra_packet_sent(int mac_status)
84 // {
85 //     /* Check if our parent just ACKed a DAO */
86 //     if(orchestra_parent_knows_us == 0
87 //         && mac_status == MAC_TX_OK
88 //         && packetbuf_attr(PACKETBUF_ATTR_NETWORK_ID) == UIP_PROTO_ICMP6
89 //         && packetbuf_attr(PACKETBUF_ATTR_CHANNEL) == (ICMP6_RPL << 8 | RPL_CODE_DAO)) {
90 //         if(!linkaddr_cmp(&orchestra_parent_linkaddr, &linkaddr_null)
91 //             && linkaddr_cmp(&orchestra_parent_linkaddr, packetbuf_addr(PACKETBUF_ADDR_RECEIVER))
92 //             orchestra_parent_knows_us = 1;
93 //     }
94 // }
95 // }
96 /*-----*/

```

```

164 orchestra_init(void)
165 {
166     int i;
167     /* Snoop on packet transmission to know if our parent knows about us
168      * (i.e. has ACKed at one of our DAOs since we decided to use it as a parent) */
169     /* Comment out unused functions
170      *
171      * Author: Huy
172      * Date: 2019/09/20
173      */
174     // netstack_sniffer_add(&orchestra_sniffer);
175     // linkaddr_copy(&orchestra_parent_linkaddr, &linkaddr_null);
176     /* Initialize all Orchestra rules */
177     for(i = 0; i < NUM_RULES; i++) {
178         if(all_rules[i]->init != NULL) {
179             printf("Orchestra: initializing rule %u\r\n", i);
180             all_rules[i]->init(i);
181         }
182     }
183     printf("Orchestra: initialization done\r\n");
184 }

```

[contiki-ng/os/services/orchestra/cnlab-orchestra-rule.c](#)

- contain the implementation of CNLAB Orchestra rule (Tx at its node id and Rx at all the other timeslots).

```

1 /* cnlabEbPeriod rule implementation
2  *
3  * Author: Huy
4  * Date: 2019/09/20
5  */
6 #include "contiki.h"
7 #include "orchestra.h"
8 #include "net/packetbuf.h"
9 #include "sys/node-id.h"
10
11 static uint16_t slotframe_handle = 0;
12 static struct tsch_slotframe *sfEb;
13 static uint8_t timeslot_space = 1;
14 /*-----*/
15 static uint16_t
16 cnlab_get_timeslot()
17 {
18     return (node_id - 1)*timeslot_space;
19 }
20 /*-----*/
21 static int
22 select_packet(uint16_t *slotframe, uint16_t *timeslot)
23 {
24     if(slotframe != NULL) {
25         *slotframe = slotframe_handle;
26     }
27     if(timeslot != NULL) {
28         *timeslot = cnlab_get_timeslot();
29     }
30     return 1;
31 }

```

```

32  /*-----*/
33  static void
34  init(uint16_t sf_handle)
35  {
36      int i;
37      uint16_t tx_timeslot;
38      slotframe_handle = sf_handle;
39      timeslot_space = CNLAB_SLOTFRAME_SIZE/NUMBER_OF_NODES;
40      uint16_t sf_size = CNLAB_SLOTFRAME_SIZE;
41      sf_eb = tsch_schedule_add_slotframe(slotframe_handle, sf_size);
42      tx_timeslot = cnlab_get_timeslot();
43      /*
44       * Add a Tx link at node id
45       * Add a Rx link at each available timeslot */
46      for(i = 0; i < sf_size; i+=timeslot_space) {
47          tsch_schedule_add_link(sf_eb,
48                                ((i == tx_timeslot) ? LINK_OPTION_TX : LINK_OPTION_RX),
49                                LINK_TYPE_ADVERTISING, &tsch_broadcast_address,
50                                i, 0);
51      }
52  }
53  /*-----*/
54  struct orchestra_rule cnlab_eb_period = {
55      init,
56      NULL,
57      select_packet,
58      NULL,
59      NULL,
60  };
61

```

Protocol Configuration:

[contiki-ng/os/net/mac/tsch/tsch-const.h](#)

- contain the define constant of TSCH.

```

63  /* 4 channels, sequence length 4 */
64  #define TSCH_HOPPING_SEQUENCE_4_4 (uint8_t[]){ 20, 22, 23, 25 }

```

[contiki-ng/os/net/mac/tsch/tsch-conf.h](#)

- contain the configuration params of TSCH.

```

58
59  /* With TSCH_ADAPTIVE_TIMESYNC enabled: keep-alive timeout used after reaching
60   * accurate drift compensation. */
61  /* Original Code
62  #ifdef TSCH_CONF_MAX_KEEPALIVE_TIMEOUT
63  #define TSCH_MAX_KEEPALIVE_TIMEOUT TSCH_CONF_MAX_KEEPALIVE_TIMEOUT
64  #else
65  #define TSCH_MAX_KEEPALIVE_TIMEOUT (60 * CLOCK_SECOND)
66  #endif*/
67  /*Change config
68   * Date: 06/11/2020
69   * Author: Huy
70   */
71  #ifdef TSCH_CONF_MAX_KEEPALIVE_TIMEOUT
72  #define TSCH_MAX_KEEPALIVE_TIMEOUT TSCH_CONF_MAX_KEEPALIVE_TIMEOUT
73  #else
74  #define TSCH_MAX_KEEPALIVE_TIMEOUT (12 * CLOCK_SECOND)
75  #endif
76
77  /* Max time without synchronization before leaving the PAN */
78  #ifdef TSCH_CONF_DESYNC_THRESHOLD
79  #define TSCH_DESYNC_THRESHOLD TSCH_CONF_DESYNC_THRESHOLD
80  #else
81  #define TSCH_DESYNC_THRESHOLD (2 * TSCH_MAX_KEEPALIVE_TIMEOUT)
82  #endif

```

```

133  /* CNLab Protocol Enable
134  *
135  * Author: Huy
136  * Date: 2020/05/15
137  */
138  #ifndef CNLAB_CONF_PROTOCOL_ENABLE
139  #define CNLAB_PROTOCOL_ENABLE 1
140  #else
141  #define CNLAB_PROTOCOL_ENABLE 0
142  #endif
143
144  /***** Configuration: channel hopping *****/
145
146  /* Default hopping sequence, used in case hopping sequence ID == 0 */
147  /* Use TSCH_HOPPING_SEQUENCE_4_4.
148  *
149  * Author: Huy
150  * Date: 2019/09/20
151  */
152  #ifndef TSCH_CONF_DEFAULT_HOPPING_SEQUENCE
153  #define TSCH_DEFAULT_HOPPING_SEQUENCE TSCH_CONF_DEFAULT_HOPPING_SEQUENCE
154  #else
155  #define TSCH_DEFAULT_HOPPING_SEQUENCE TSCH_HOPPING_SEQUENCE_4_4
156  #endif

```

```

174  /***** Configuration: association *****/
175
176  /* Start TSCH automatically after init? If not, the upper layers
177  * must call NETSTACK_MAC.on() to start it. Useful when the
178  * application needs to control when the nodes are to start
179  * scanning or advertising.*/
180  /* Turn off TSCH Auto start, we call NETSTACK_MAC.on() in our application.
181  *
182  * Author: Huy
183  * Date: 2019/09/20
184  */
185  #ifndef TSCH_CONF_AUTOSTART
186  #define TSCH_AUTOSTART TSCH_CONF_AUTOSTART
187  #else
188  #define TSCH_AUTOSTART 0
189  #endif
190
191  /* By default: initialize schedule from EB when associating, using the
192  * slotframe and links Information Element */
193  /* Turn off TSCH init schedule from EB. We use PC-LLF in our application.
194  *
195  * Author: Huy
196  * Date: 2019/09/20
197  */
198  #ifndef TSCH_CONF_INIT_SCHEDULE_FROM_EB
199  #define TSCH_INIT_SCHEDULE_FROM_EB TSCH_CONF_INIT_SCHEDULE_FROM_EB
200  #else
201  #define TSCH_INIT_SCHEDULE_FROM_EB 0
202  #endif

```

```

296  /* Size of the ring buffer storing incoming packets.
297  * Must be power of two */
298  #ifndef TSCH_CONF_MAX_INCOMING_PACKETS
299  #define TSCH_MAX_INCOMING_PACKETS TSCH_CONF_MAX_INCOMING_PACKETS
300  #else
301  #define TSCH_MAX_INCOMING_PACKETS 32
302  #endif

```

```

370  /* Max number of links */
371  #ifdef TSCH_SCHEDULE_CONF_MAX_LINKS
372  #define TSCH_SCHEDULE_MAX_LINKS TSCH_SCHEDULE_CONF_MAX_LINKS
373  #else
374  #define TSCH_SCHEDULE_MAX_LINKS 256
375  #endif

```

```

455  /* TSCH timeslot timing template */
456  #ifdef TSCH_CONF_DEFAULT_TIMESLOT_TIMING
457  #define TSCH_DEFAULT_TIMESLOT_TIMING TSCH_CONF_DEFAULT_TIMESLOT_TIMING
458  #else
459  #define TSCH_DEFAULT_TIMESLOT_TIMING tsch_timeslot_timing_us_15000
460  #endif

```

contiki-ng/os/net/mac/tsch/tsch-timeslot-timing.c

- contain the define constant of TSCH timeslot timing.
- Different between 10ms and 15ms config as below

10ms	15ms
<pre> 64 const tsch_timeslot_timing_us 65 1800, /* CCAOffset */ 66 128, /* CCA */ 67 2120, /* TxOffset */ 68 (2120 - (TSCH_CONF_RX_WAIT 69 800, /* RxAckDelay */ 70 1000, /* TxAckDelay */ 71 TSCH_CONF_RX_WAIT, /* RxWai 72 400, /* AckWait */ 73 192, /* RxTx */ 74 2400, /* MaxAck */ 75 4256, /* MaxTx */ 76 10000, /* TimeslotLength */ 77 }; </pre>	<pre> 84 const tsch_timeslot_timing_us 85 1800, /* CCAOffset */ 86 128, /* CCA */ 87 4000, /* TxOffset */ 88 (4000 - (TSCH_CONF_RX_WAIT 89 3600, /* RxAckDelay */ 90 4000, /* TxAckDelay */ 91 TSCH_CONF_RX_WAIT, /* RxWai 92 800, /* AckWait */ 93 2072, /* RxTx */ 94 2400, /* MaxAck */ 95 4256, /* MaxTx */ 96 15000, /* TimeslotLength */ 97 }; </pre>

Protocol Implementation:

contiki-ng/os/net/mac/tsch/cnlab-protocol.h

- contain the define constant, data structure, declare of processes and functions of CNLAB Protocol.


```

1  /* CNLAB PROTOCOL implementation header.
2  *
3  * Author: Huy
4  * Date: 2019/09/20
5  */
6  #ifndef CNLAB_CONF_PROTOCOL_ENABLE
7  #ifndef CNLAB_PROTOCOL_
8  #define CNLAB_PROTOCOL_
9
10 /***** Includes *****/
11 #include "contiki.h"
12 #include "sys/node-id.h"
13 #include "net/nbr-table.h"
14 #include "net/linkaddr.h"
15
16 #define NUMBER_OF_PROBE_RSSI 30
17 #define NUMBER_OF_PROBE_PRR 500
18 #define MAX_TREE_DEPTH 16
19 #define MIN_PRR_LINK 0.85f
20 #define MAX_PRR_LINK 0.99f
21
22 /* Protocol states */
23 enum CNLAB_PROTOCOL_STATES {
24     CNLAB_PROTOCOL_DEFAULT,
25     CNLAB_PROTOCOL_INITIALIZE,
26     CNLAB_PROTOCOL_MEASURE_LINK_QUALITY,
27     CNLAB_PROTOCOL_MAKE_SCHEDULE,
28     CNLAB_PROTOCOL_SEND_DATA,
29     CNLAB_PROTOCOL_RESET
30 };
31
32 /* Uart Packet Types */
33 enum CNLAB_UART_PACKET_TYPES {
34     ROUTING_UART_PACKET,
35     DOWNSCHEDULE_UART_PACKET,
36     UPSCHEDULE_UART_PACKET
37 };
38
39 /* Handle fixed point */
40 typedef uint32_t fixed_point_t;
41
42 /* RSSI Array */
43 struct rssi_array
44 {
45     uint8_t id;
46     uint8_t count;
47     int16_t rssi_avg;
48 } __attribute__((packed));
49 typedef struct rssi_array rssi_array_t;

```

```

50
51  /* Node Information */
52  struct node_info {
53      uint8_t id;
54      uint8_t type;
55      uint8_t level;
56      fixed_point_t rank;
57      uint8_t degree;
58      uint8_t primary_parent;
59      uint8_t reserve_parent;
60      uint8_t next_hop_child;
61      uint8_t number_of_nbr;
62  }__attribute__((packed));
63  typedef struct node_info node_info_t;
64
65  /* Neighbor Infomation */
66  struct nbr_info {
67      uint8_t id;
68      uint8_t level;
69      uint16_t rx_probe_count;
70      int16_t avg_rssi;
71      fixed_point_t prr_incoming; //Neighbor->Node
72      fixed_point_t prr_outgoing; //Node->Neighbor
73  }__attribute__((packed));
74  typedef struct nbr_info nbr_info_t;
75
76  /* Uptree Information */
77  struct uptree_info {
78      uint8_t node;
79      fixed_point_t rank;
80      uint8_t primary_parent;
81      uint8_t reserve_parent;
82  }__attribute__((packed));
83  typedef struct uptree_info uptree_info_t;
84
85  /* Schedule Cell Structure */
86  struct cell {
87      uint16_t timeslot;
88      uint8_t channel;
89      linkaddr_t lladdr;
90  }__attribute__((packed));
91  typedef struct cell cell_t;
92
93  /* Scheduling Information */
94  struct schedule_data {
95      uint16_t sf_size;
96      uint16_t num_transmit_cells;
97      uint16_t num_receive_cells;
98      cell_t *transmit_cells;
99      cell_t *receive_cells;
100  }__attribute__((packed));
101  typedef struct schedule_data schedule_data_t;

```

```

103  /* Data Packet */
104  struct data {
105      char cmd[2];
106      char value1[3];
107      char value2[6];
108      char value3[6];
109  }__attribute__((packed));
110  typedef struct data data_t;
111
112  /***** Variables *****/
113  process_event_t send_data_event;
114
115  /***** CNLAB Processes *****/
116
117  /* Process Measure Link Quality */
118  PROCESS_NAME(cnlab_measure_link_process);
119
120  /* Process Receive Network Information */
121  PROCESS_NAME(cnlab_receive_network_info);
122
123  /* Process send data packet */
124  PROCESS_NAME(cnlab_send_data_process);
125
126  /* Process send command packet */
127  PROCESS_NAME(cnlab_send_command_process);
128
129  /***** Functions *****/
130
131  /* Write log to SD Card files */
132  void cnlab_write_log_to_file(const unsigned char *data, unsigned int len);
133
134  /* Init protocol */
135  void cnlab_protocol_init();
136
137  /* Start protocol */
138  void cnlab_process_start();
139
140  /* Send periodic data packet */
141  void cnlab_periodic_data_output(uint16_t seq_no, uint8_t temperature);
142
143  /* Send command packet */
144  void cnlab_aperidic_data_output(uint8_t type);
145
146  /* Processing received packet */
147  void cnlab_packet_input();
148
149  /* Reset protocol */
150  void cnlab_protocol_reset();
151
152  #endif /* __CNLAB_PROTOCOL__ */
153  #endif

```