CoAP observing + client

Carlo Vallati

Associate Professor @ University of Pisa

carlo.vallati@iet.unipi.it - http://www.iet.unipi.it/c.vallati/

Observing

- CoAP introduces observing, an additional mode to subscribe to the changes in the status of a CoAP resource
- A client can issue an observing relationship and obtain a notification every time the status changes

Define an observable resource

```
EVENT_RESOURCE(res_event,
                "title=\"Event demo\";obs",
                res_get_handler,
                NULL,
                NULL,
                NULL,
                res_event_handler);


// A different declaration is defined for
resources with observing

// An additional handler is included, it will be
automatically called when an event is triggered by
the system
```

Observe handler

```
static void
res_event_handler(void)
{
    // Notify all the observers
    // Before sending the notification the handler associated with the GET methods is called
    coap_notify_observers(&res_event);
}
```

Trigger the event

// When needed, e.g. when certain conditions are met the server can trigger the event associated with an observable resource and notify subscribers

```
res_event.trigger();
```

Exercise 1

- Write a CoAP server with an observable resource
- The server must be programmed to trigger the event that notify the subscribers with a given period
- The notification must include the value of a counter that is incremented every time an event is triggered
- Test the server with an external client

```
coap-client -m get coap://[ADDR]:5683/obs -s 100
```

- observing.c

CoAP Client

- Contiki includes the implementation of a CoAP client
- It can be used to allow sensor nodes to retrieve information from CoAP server installed over the Internet or to retrieve data from other sensors installed on the same local network

Header and variables

```
#include “contiki.h”
#include "coap-engine.h“
#include "coap-blocking-api.h"

// Server IP and resource path
#define SERVER_EP "coap://[fd00::202:2:2:2]:5683"
char *service_url = "/hello";
```

Response handler

```
// Define a handler to handle the response from the server
void
client_chunk_handler(coap_message_t *response)
{
  const uint8_t *chunk;

  if(response == NULL) {
    puts("Request timed out");
    return;
  }

  int len = coap_get_payload(response, &chunk);

  printf("|%.*s", len, (char *)chunk);
}
```

Main thread

```
// The client includes two data structures
// coap_endpoint_t -> represents an endpoint
// coap_message_t -> represent the message


PROCESS_THREAD(er_example_client, ev, data)
{
  static coap_endpoint_t server_ep;
  static coap_message_t request[1];  /* This way
the packet can be treated as pointer as usual. */


  PROCESS_BEGIN();
  …
```

Main thread

```
// Populate the coap_endpoint_t data structure
coap_endpoint_parse(SERVER_EP, strlen(SERVER_EP),
&server_ep);

// Prepare the message
coap_init_message(request, COAP_TYPE_CON,
      COAP_POST, 0);
coap_set_header_uri_path(request, service_url);

// Set the payload (if needed)
const char msg[] = "Toggle!";
coap_set_payload(request, (uint8_t *)msg,
      sizeof(msg) - 1);
```

Main thread

```
// Issue the request in a blocking manner
// The client will wait for the server to reply
      (or the transmission to timeout)

COAP_BLOCKING_REQUEST(&server_ep, request,
      client_chunk_handler);
```

Exercise 2

- Write a CoAP client that issues a POST request to a server
- Deploy a server on a second sensor (e.g. an hello coap server) and configure the client to issue the request to that server

- coap-example-client.c

Test on the testbed

- The test coap-server can be deployed on the testbed via the coap-server tool
- Run the coap-server tool:

```
coap-server -v 10
```

- On the code of the client you must change the URI for the requests to:

```
#define SERVER_EP "coap://[fd00::1]:5683"

char *service_url = "/";
```

References

- https://github.com/contiki-ng/contiki-ng/wiki/Tutorial:-CoAP