gcoap: CoAP for RIOT

- Tailored to RIOT – threaded, event loop
- Tailored to RIOT’s community, LGPL
- Built on nanocoap and sock
- Friendly – well documented and accessible
- Shoulders of giants – beautiful abstractions

nanocoap

gcoap is built with nanocoap

- Kaspar Schleiser’s minimal implementation
- nanocoap runs on Linux and RIOT
- low-level API *and* an application
- gcoap reuses constants, structs, and utility functions

https://github.com/kaspar030/sock

nanocoap Utility Functions

Examples

- coap_get_code_class(pdu)
- coap_hdr_set_type(hdr, unsigned)
- coap_put_option(option_num, data, ...)
- int coap_parse(pdu, buf, len)
- Constants are macros rather than enums

coap_pkt_t (PDU)

- hdr, token, payload also use a separate buffer

```
coap_hdr_t *hdr;
uint8_t url[NANOCOAP_URL_MAX];
uint8_t qs[NANOCOAP_QS_MAX];
uint8_t *token;
uint8_t *payload;
unsigned payload_len;
uint16_t content_type;
uint32_t observe_value;
```

gcoap Features

What works today

- Non-confirmable messaging
- Observe extension (server)
  - Confirmable notifications in #7548
- Confirmable messaging in progress
  - Piggyback ACK with retries in #7337

gcoap API

Common sequence for request, response, observe

- *xxx*_init()
- write payload
- finish()

http://doc.riot-os.org/group__net__gcoap.html
https://github.com/RIOT-OS/RIOT/blob/master/sys/include/net/gcoap.h

Build Request

- **gcoap_req_init**(pdu, buf, len, code, path)
- **coap_hdr_set_type**() if CON
    - maybe generalize to set_options(struct)
- Write payload, at pointer in the pdu
- **gcoap_finish**(pdu, payload_len, format) updates the packet for the payload

  – OR –
- **gcoap_request**(pdu, buf, len, code, path)

Send Request

- **gcoap_req_send2**(buf, len, sock_ep, handler)
  - Sock helps abstract networking; includes host, port, IP family

- **<handler>**(state, pdu, sock_ep)
  read payload, if any, from pdu

Server resources

```
coap_resource_t _resources[] = {
    { TEMP_PATH, COAP_GET, _temp_handler },
    ...
};
```

Pass to gcoap server as linked list element

```
static gcoap_listener_t _listener = {
    (coap_resource_t *)&_resources[0],
    sizeof(_resources) / sizeof(_resources[0]),
    NULL
};
```

Response Handler

- **gcoap_resp_init**(pdu, buf, len, code)
- Write payload
- return **gcoap_finish**(pdu, payload_len, format)

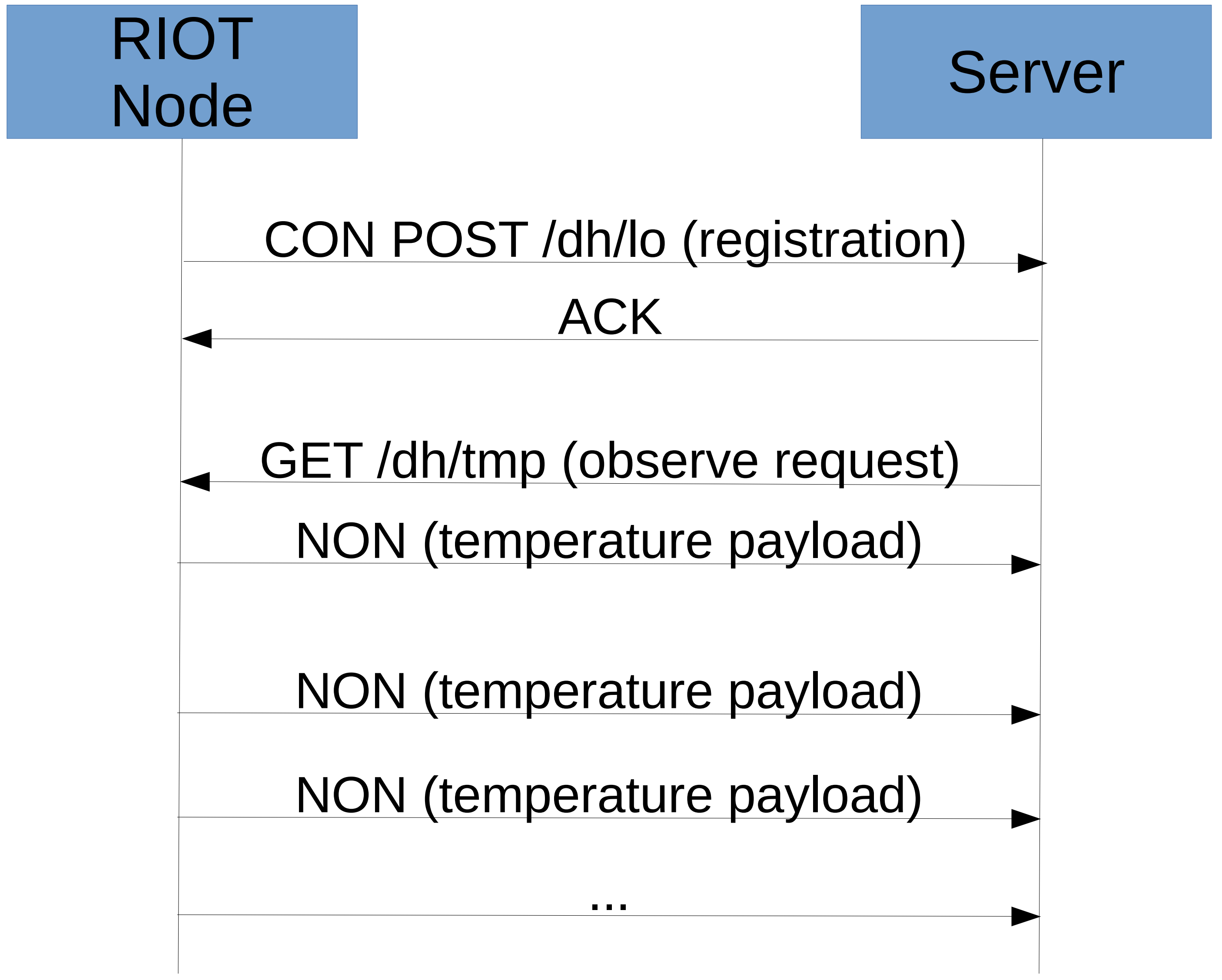
  – OR –
- return **gcoap_response**(pdu, buf, len, code)

Observe Notification

- **gcoap_obs_init**(pdu, buf, len, *resource*)
- **coap_hdr_set_type**() if CON
- Write payload, at pointer in the pdu struct
- **gcoap_finish**(pdu, payload_len, format) updates the packet for the payload

- **gcoap_obs_send**(buf, len, resource);

Data Collection Demo


RIOT Node
Server
CON POST /dh/lo (registration)
ACK
GET /dh/tmp (observe request)
NON (temperature payload)
NON (temperature payload)
NON (temperature payload)
...

https://github.com/kb2ma/riot-data-collector

Register with Server

```
uint8_t buf[GCOAP_PDU_BUF_SIZE];
coap_pkt_t pdu;
size_t len;
sock_udp_ep_t server_sock;
ipv6_addr_t addr;

gcoap_req_init(&pdu, &buf[0], GCOAP_PDU_BUF_SIZE, COAP_METHOD_POST,
                      DATAHEAD_HELLO_PATH);
coap_hdr_set_type(pdu.hdr, COAP_TYPE_CON);
len = gcoap_finish(&pdu, 0, COAP_FORMAT_NONE);

server_sock.family = AF_INET6;
server_sock.netif  = SOCK_ADDR_ANY_NETIF;

/* parse destination address */
ipv6_addr_from_str(&addr, DATAHEAD_ADDR;
memcpy(&server_sock.addr.ipv6[0], &addr.u8[0], sizeof(addr.u8));

/* parse port */
server_sock.port = (uint16_t)atoi(DATAHEAD_PORT);

return gcoap_req_send2(buf, len, &server_sock, _hello_handler);
```

Handle hello Response

```
static void _hello_handler(unsigned req_state, coap_pkt_t* pdu,
                           sock_udp_ep_t *remote)
{
    (void)remote;

    if (req_state == GCOAP_MEMO_TIMEOUT) {
        printf("Timeout on 'hello' response; msg ID %02u\n", coap_get_id(pdu));
    }
    else if (req_state == GCOAP_MEMO_ERR) {
        puts("Error in 'hello' response");
    }
    else {
        puts("'hello' response OK");
    }
}
```

Server resources

```
static const coap_resource_t _resources[] = {
    { DATAHEAD_TEMP_PATH, COAP_GET, _temp_handler },
};
static gcoap_listener_t _listener = {
    (coap_resource_t *)&_resources[0],
    sizeof(_resources) / sizeof(_resources[0]),
    NULL
};

…

int main(void)
{
    ...
    gcoap_register_listener(&_listener);
    …
}
```

Handle Temperature Request

```
static ssize_t _temp_handler(coap_pkt_t* pdu, uint8_t *buf, size_t len)
{
        phydat_t phy;

        int res = saul_reg_read(_saul_dev, &phy);

        if (res) {
                gcoap_resp_init(pdu, buf, len, COAP_CODE_CONTENT);
                payload_len = fmt_u16_dec((char *)pdu.payload, phy.val[0]);
                return gcoap_finish(pdu, payload_len, COAP_FORMAT_TEXT);
        } else {
                return gcoap_response(pdu, buf, len,
                                        COAP_CODE_INTERNAL_SERVER_ERROR);
        }
}
```

Send Temperature Notifications

```
static void _run_sensor_loop(void)
{
    uint8_t buf[GCOAP_PDU_BUF_SIZE];
    coap_pkt_t pdu;
    size_t len, payload_len;

    while (1) {
        phydat_t phy;

        int res = saul_reg_read(_saul_dev, &phy);

        if (res) {
            res = gcoap_obs_init(&pdu, &buf[0], GCOAP_PDU_BUF_SIZE, &_resources[0]);
            if (res == GCOAP_OBS_INIT_OK) {
                payload_len = fmt_u16_dec((char *)pdu.payload, phy.val[0]);
                len = gcoap_finish(&pdu, payload_len, COAP_FORMAT_TEXT);
                gcoap_obs_send(&buf[0], len, &_resources[0]);
            }
        }

        xtimer_usleep(20 * US_PER_SEC);
    }
}
```