Static Context Header Compression
[sjiek]

Where do we want to go in RIOT?
Integrate libSCHC for

- standard, IPv6-based connectivity to the smallest devices
- reliable fragmentation at MAC layer
OBJECTIVES

1. SCHC for end-to-end security
   1. CoAP + OSCORE

2. RIOT on LoRaWAN gateway/network server

3. SCHC as a generic framework
   1. HTTP, CoAP, MQTT
   2. NDN?

4. SCHC for reliable fragmentation
COMPRESSION
Currently, GNRC_NETTYPE_SCHC bypasses the gnrc_udp and gnrc_ipv6 layers. The following code snippet demonstrates how packets are sent through the SCHC thread:

```c
{  /* send packet */
    gnrc_pktbuf_release(ip);
    if (!gnrc_netapi_dispatch_send(GNRC_NETTYPE_SCHC, GNRC_NETREG_DEMUX_CTX_ALL, ip)) {
        puts("Error: unable to locate SCHC thread");
        gnrc_pktbuf_release(ip);
        return;
    }
}
```

This code snippet shows how packets are sent through the SCHC thread, bypassing the UDP and IPv6 layers.
LIBSCHC

What it should be

• Behave like *gnrc_6lo* adaptation layer (fragmentation + compression)
• Fragmentation w/o compression
LIBSCHC

Layers can be identified by RIOT using

GNRC_NETTYPE_x: no configuration

Currently set at compile time inside compressor

#define USE_COAP 1
#define USE_UDP 1
#define USE_IPv6 1
Use a protocol parser per layer
• detect protocol stack at compile time

```c
struct schc_rule_t* schc_compress(uint8_t *data,
                                 uint16_t total_length,
                                 schc_bitarray_t* dst, uint32_t device_id, direction dir) {
    #if USE_IPv6
        compress(dst, &src, (const struct schc_layer_rule_t*) ipv6_rule, dir);
    #endif
    #if USE_UDP
        compress(dst, &src, (const struct schc_layer_rule_t*) udp_rule, dir);
    #endif
    #if USE_COAP
        compress(dst, &coap_src, (const struct schc_layer_rule_t*) coap_rule, dir);
    #endif
}
```
LIBSCHC

Protocol parser per layer

• What about OSCORE? How can we offer SCHC (L2) + OSCORE (L5)?
  • Compression in 2 steps
FRAGMENTATION
LIBSCHC

Fragmentation
- **libSCHC** requires
  - information about the netif (MTU, duty cycle)
  - configuration per packet for
    - reliability mode

```c
schc_conn.mtu = 8; // network driver MTU
schc_conn.dc = 5000; // 5 seconds duty cycle
schc_conn.device_id = device_id; // the device id of the connection
schc_conn.bit_arr = &schc_bitbuff;
```
CURRENT IMPLEMENTATION

Memory management

- fragmented packet uses pre-allocated chunk of memory, stored in a mbuf (network memory buffer) chain
- contains a pointer to headers and payload

```c
typedef struct schc_mbuf_t {
    ...
    /* the length of the fragment */
    uint16_t len;
    /* pointer to the chunk of memory */
    uint8_t * data;
    /* pointer to the next mbuf */
    struct schc_mbuf_t next;
} schc_mbuf_t;
```
Abstraction required

• differentiate between end-devices

• possibility to
  • reorder linked list (missing fragments)
  • take a single payload byte from the chain
    • to calculate MIC
LIBSCHC

libSCHC works on a per fragment basis that relates to the original packet

- Receiver can ask the retransmission of a specific fragment
Receiver should be able to reconstruct the original packet (removing the fragmentation headers) to calculate the RCS (MIC)

- mbuf
Currently poor memory management in libSCHC.

- Separate memory and mbuf logic and instead use pktsnip
RIOT INTEGRATION

- RIOT as a SCHC network gateway
  - `pktqueue` to queue packets for multiple devices that are restricted in downlink communication

- Configuration?
Device configuration

/* combine compression and fragmentation parameters */
const struct schc_rule_t schc_rule_1 = { 0x01, 
  &not_found_404, NOT_FRAGMENTED, 0, 0, 0, 0 };
const struct schc_rule_t schc_rule_3 = { 0x03, 
  &not_found_404, ACK_ON_ERROR, 3, 6, 1, 0 };

/* save rules in flash */
const struct schc_rule_t* node1_schc_rules[] = {
  &schc_rule_1, &schc_rule_2, 
  &schc_rule_3, &schc_rule_4, &schc_rule_5, 
  &schc_rule_6, &schc_rule_7, &schc_rule_8, 
  &schc_rule_9, &schc_rule_10, &schc_rule_11, 
  &schc_rule_12, &schc_rule_13, &schc_rule_14, 
  &schc_rule_15, &schc_rule_16, 
  &schc_rule_17 
};

/* rules for a particular device */
const struct schc_device node1 = { 0x06, 17, 
  &node1_schc_rules 
};
const struct schc_device node2 = { 0x01, 17, 
  &node1_schc_rules 
};
RIOT INTEGRATION

• Setting up the context
• RIOT filesystem?

```c
#if USE_UDP
const static struct schc_udp_rule_t udp_rule1 = {
    1, 4, 4, 4, {
        { "src port", 2, 16, 1, BI, { 0x33, 0x17, 0x33, 0x17 }, &mo_matchmap, MAPPINGSENT },
        { "dest port", 2, 16, 1, BI, { 0x33, 0x16, 0x33, 0x17 }, &mo_matchmap, MAPPINGSENT },
        { "length", 0, 16, 1, BI, { 0, 0 }, &mo_ignore, COMPLENGTH },
        { "checksum", 0, 16, 1, BI, { 0, 0 }, &mo_ignore, COMPCHK } }
};
#endif
```
Static Context Header Compression [sjiek] in RIOT

BART MOONS

Ghent University – IDLab – imec

bamoons.moons@ugent.be
idlab.ugent.be