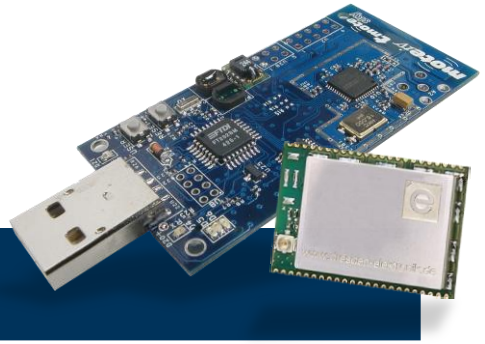
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

A Low-Power CoAP for Contiki

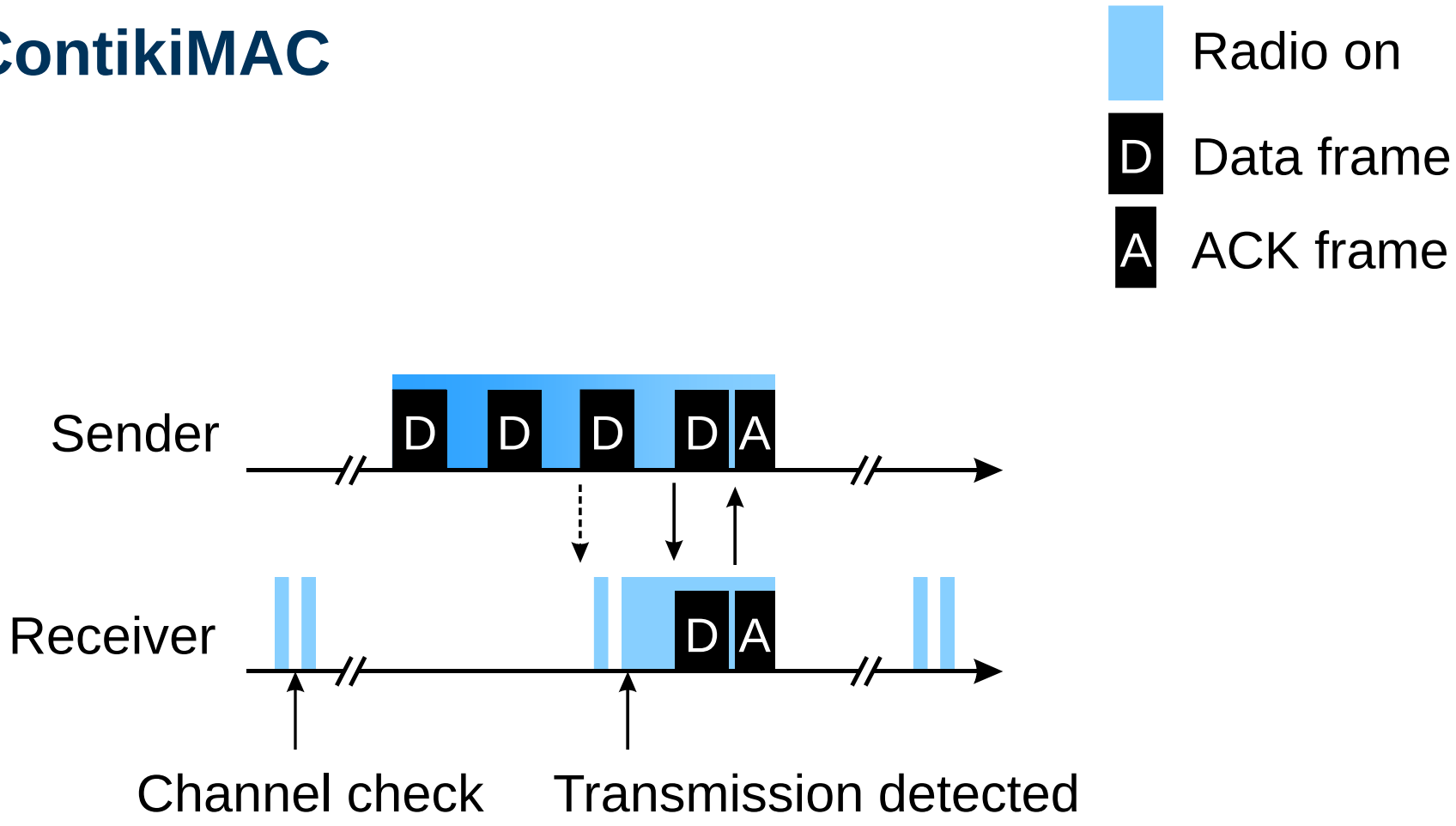
Matthias Kovatsch, Simon Duquennoy, and Adam Dunkels
kovatsch@inf.ethz.ch simonduq@sics.se adam@sics.se

Internet of Things Protocol Stack

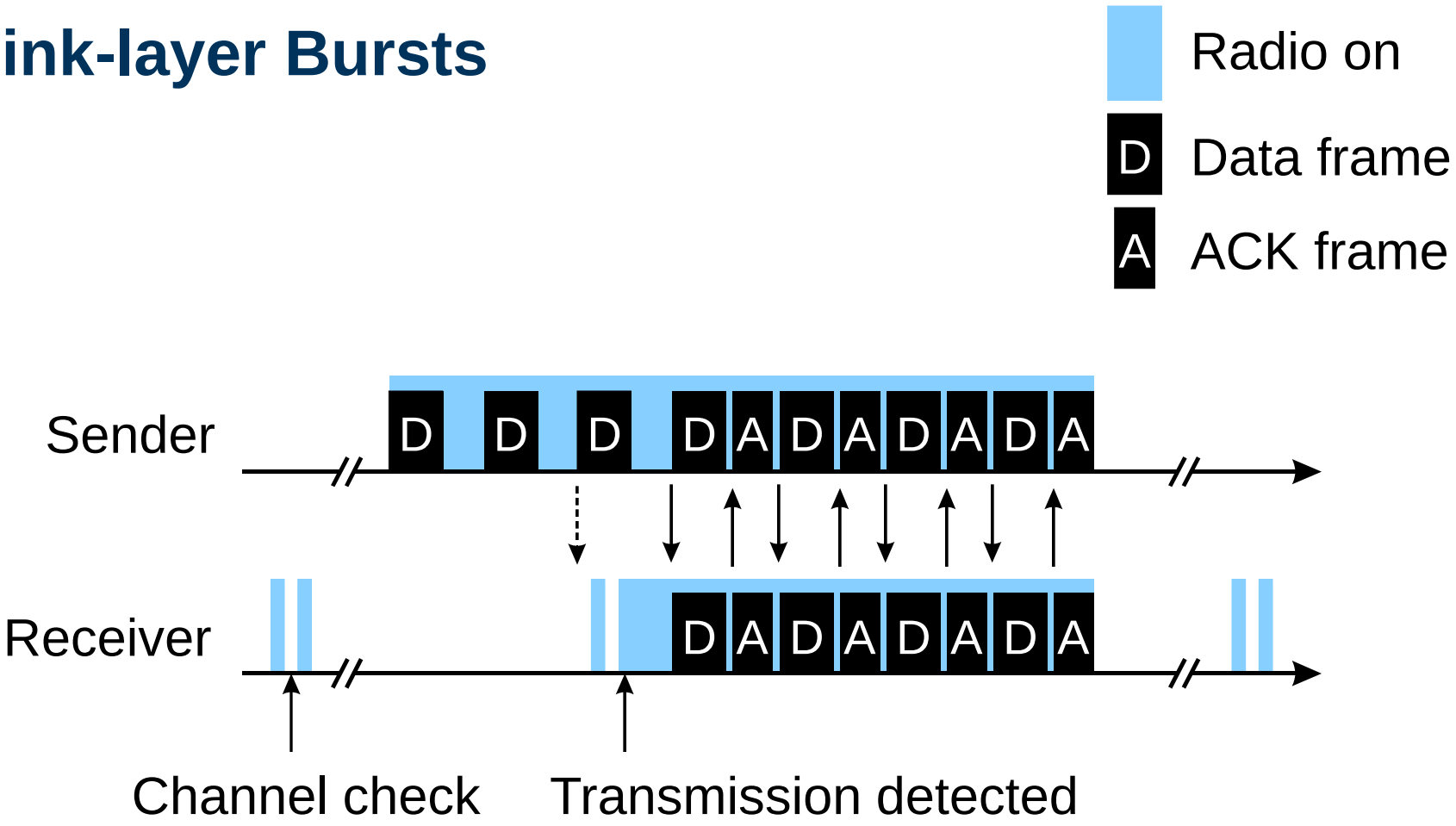


Layer	Protocol
Application	CoAP / REST Engine (Erbium)
Transport	UDP
Network	IPv6 / RPL
Adaption	6LoWPAN
MAC	CSMA / link-layer bursts
Radio Duty Cycling	ContikiMAC
Physical	IEEE 802.15.4

ContikiMAC

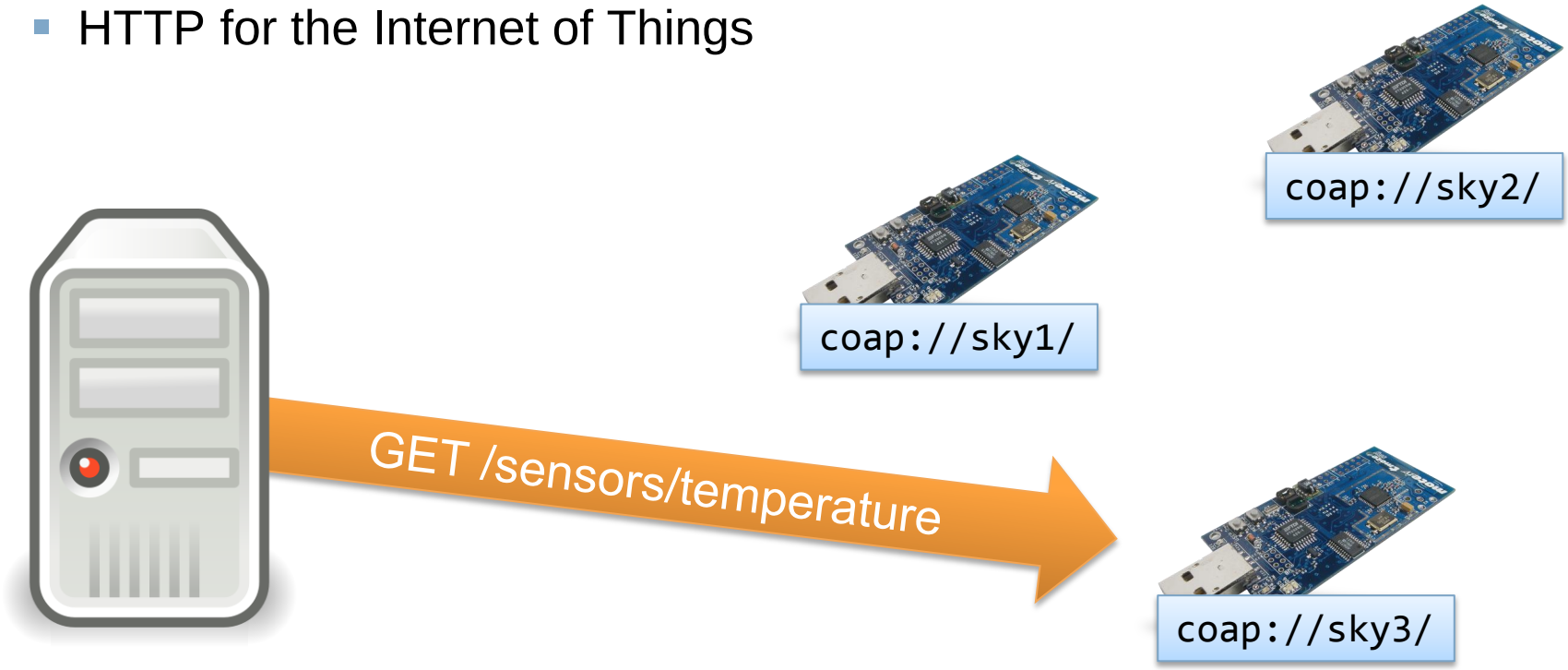

Radio on
D
Data frame
A
ACK frame
Sender
D
D
D
D
A
Receiver
D
A
Channel check
Transmission detected

Link-layer Bursts

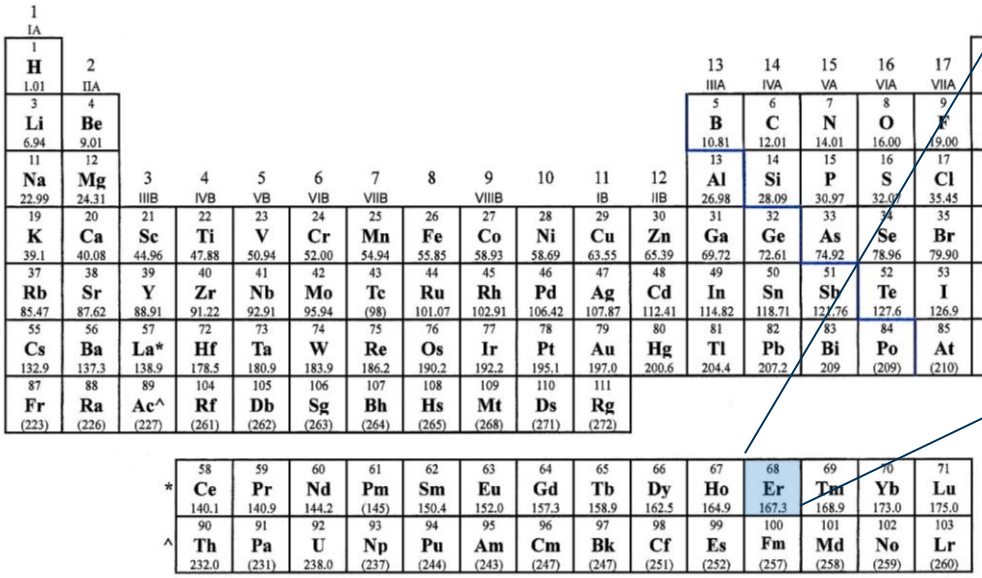


Constrained Application Protocol (CoAP)

- RESTful Web services for networked embedded devices
  - Idealized architectural style of the Web
  - HTTP for the Internet of Things



«Erbium» CoAP for Contiki



- Implements draft-ietf-core-coap-07
- Available in the Contiki repository on SourceForge

«Erbium» CoAP

- Reliable UDP transport
- Observing resources
- Blockwise transfers
- Resource discovery
- Separate responses
- Blocking client requests

```
/* Handle requests. */
if (message->code >= COAP_GET && message->code <= COAP_DELETE) {
  /* Use transaction buffer for response to confirmable request. */
  if ( (transaction = coap_new_transaction(message->tid, &UIP_IP_BUF->srcipaddr, UIP_UDP_BUF->srcport)) ) {
    static uint32_t block_num = 0;
    static uint16_t block_size = REST_MAX_CHUNK_SIZE;
    static uint32_t block_offset = 0;
    static int32_t new_offset = 0;
    if (message->type==COAP_TYPE_CON) {
      /* Reliable CON requests are answered with an ACK. */
      coap_init_message(response, COAP_TYPE_ACK, CONTENT_2_05, message->tid);
    } else {
      /* Unreliable NON requests are answered with a NON as well. */
      coap_init_message(response, COAP_TYPE_NON, CONTENT_2_05, coap_get_tid());
    }
    /* resource handlers must take care of different handling (e.g., TOKEN_OPTION_REQUIRED_240) */
    if (IS_OPTION(message, COAP_OPTION_TOKEN)) {
      coap_set_header_token(response, message->token, message->token_len);
      SET_OPTION(response, COAP_OPTION_TOKEN);
    }
    /* get offset for blockwise transfers */
    if (coap_get_header_block2(message, &block_num, NULL, &block_size, &block_offset)) {
      block_size = MIN(block_size, REST_MAX_CHUNK_SIZE);
      new_offset = block_offset;
    } else {
      new_offset = 0;
    }
    /* Invoke resource handler. */
    if (service_cbk) {
      /* Call REST framework and check if found and allowed. */
      if (service_cbk(message, response, transaction->packet+COAP_MAX_HEADER_SIZE, block_size, &new_offset)) {
        /* Apply blockwise transfers. */
        if ( IS_OPTION(message, COAP_OPTION_BLOCK2) ) {
          /* unchanged new_offset indicates that resource is unaware of blockwise transfer */
          if (new_offset==block_offset) {
            if (block_offset >= response->payload_len) {
              response->code = BAD_OPTION_4_02;
              coap_set_payload(response, (uint8_t*)"Block out of scope", 18);
            } else {
              coap_set_header_block2(response, block_num, response->payload_len - block_offset > block_size, block_size);
              coap_set_payload(response, response->payload+block_offset, MIN(response->payload_len - block_offset, block_size));
            } /* if (valid offset) */
          } else {
            /* resource provides chunk-wise data */
            coap_set_header_block2(response, block_num, response->payload_len > block_size, block_size);
            if (response->payload_len > block_size) coap_set_payload(response, response->payload, block_size);
          } /* if (resource aware of blockwise) */
        } else if (new_offset!=0) {
          coap_set_header_block2(response, 0, new_offset!=-1, REST_MAX_CHUNK_SIZE);
          coap_set_payload(response, response->payload, MIN(response->payload_len, REST_MAX_CHUNK_SIZE));
        } /* if (blockwise request) */
      } else {
        coap_error_code = INTERNAL_SERVER_ERROR_5_00;
        coap_error_message = "service callback undefined";
      } /* if (service callback) */
      /* serialize response */
      if ((transaction->packet_len = coap_serialize_message(response, transaction->packet))==0) {
        coap_error_code = PACKET_SERIALIZATION_ERROR;
      } else {
        coap_error_code = MEMORY_ALLOC_ERR;
        coap_error_message = "Transaction buffer allocation failed";
      } /* if (transaction buffer) */
      /* Responses */
      if (message->type==COAP_TYPE_ACK) {
        /* Handled by transaction layer */
      } else if (message->type==COAP_TYPE_RST) {
        /* Cancel possible
        if (IS_OPTION(message
```

«Erbium» REST Engine

- Resource abstraction

```
RESOURCE(handle, METHODs, URI-Path, Web Linking info);
```

  - Resource handler

```
PERIODIC_RESOURCE(handle, METHODs, URI-Path, info, period);
```

  - Additional periodic handler

```
EVENT_RESOURCE(handle, METHODs, URI-Path, info);
```

  - Additional event handler

CoRE Link Format

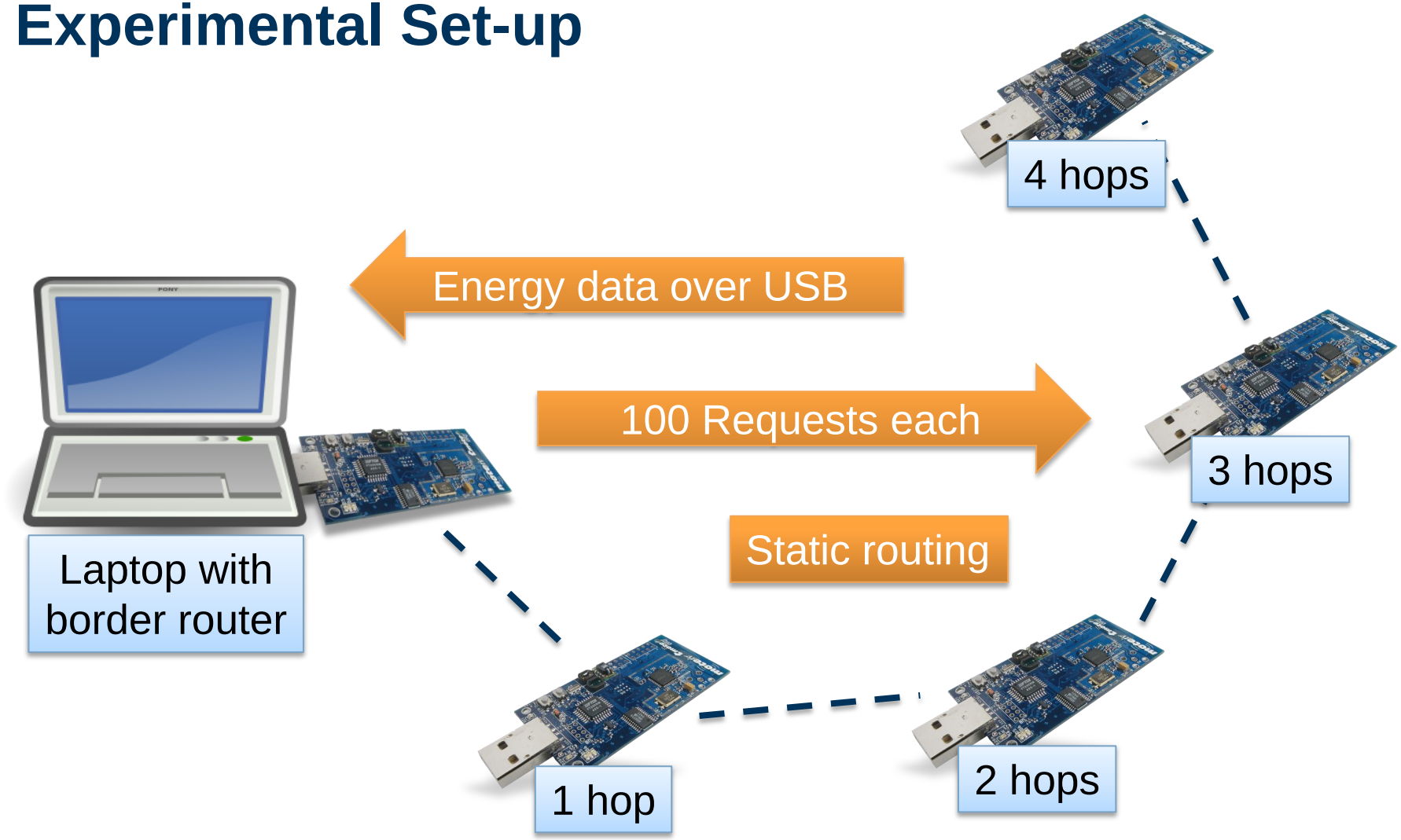
```
title="Hello world";
rt="TemperatureC";
```

Observable

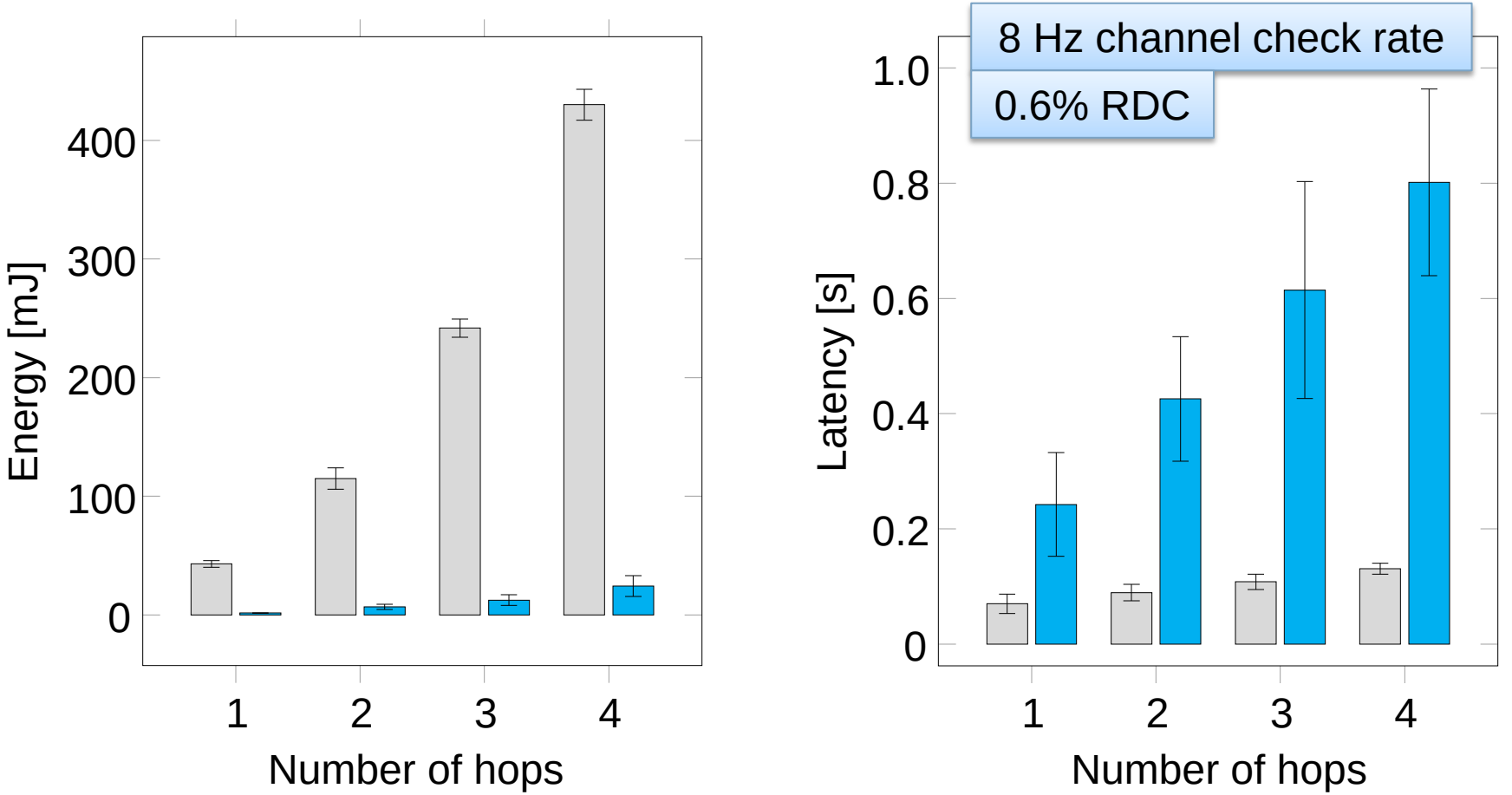
«Erbium» Memory Footprint

	ROM [kB]	RAM [kB]
CoAP REST Engine total	**8.5**	**1.5**
Measured stack usage	–	0.1
REST Engine	0.7	0
CoAP-07 base	4.5	0
CoAP-07 server	1.9	0.3
CoAP-07 transport	0.4	0.9
CoAP-07 observing	0.9	0.2
CoAP-07 separate	0.1	0

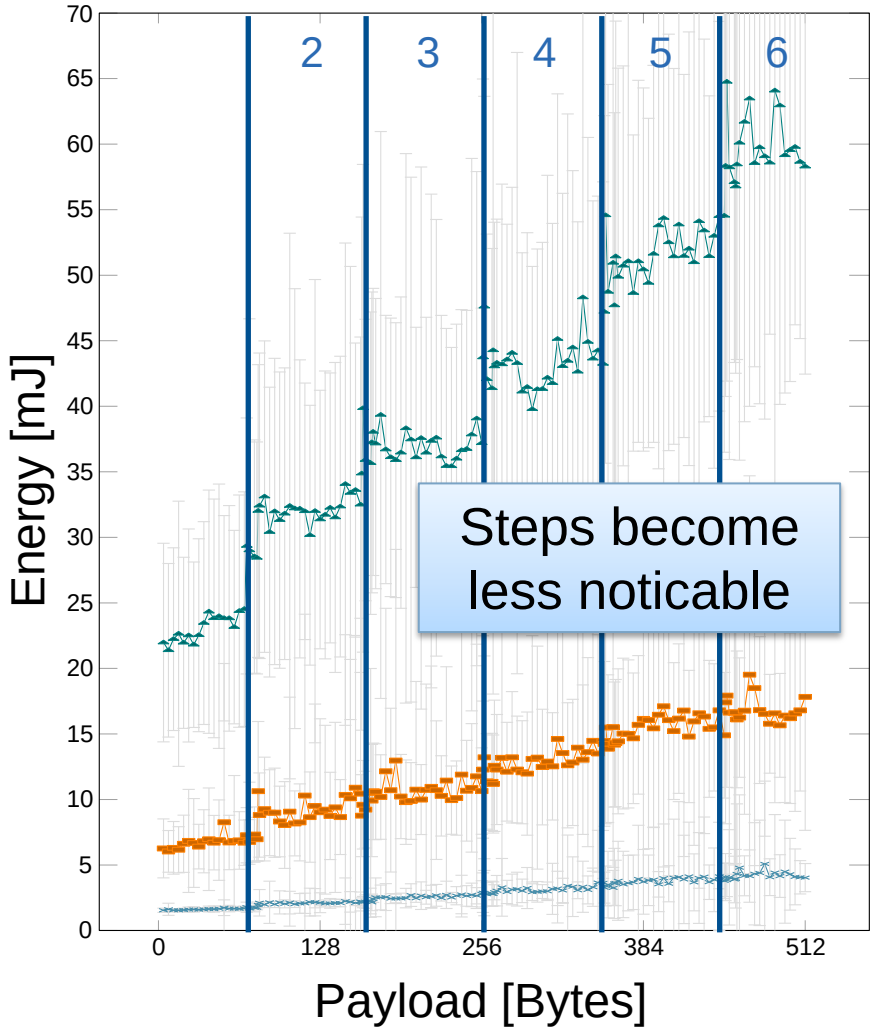
Experimental Set-up

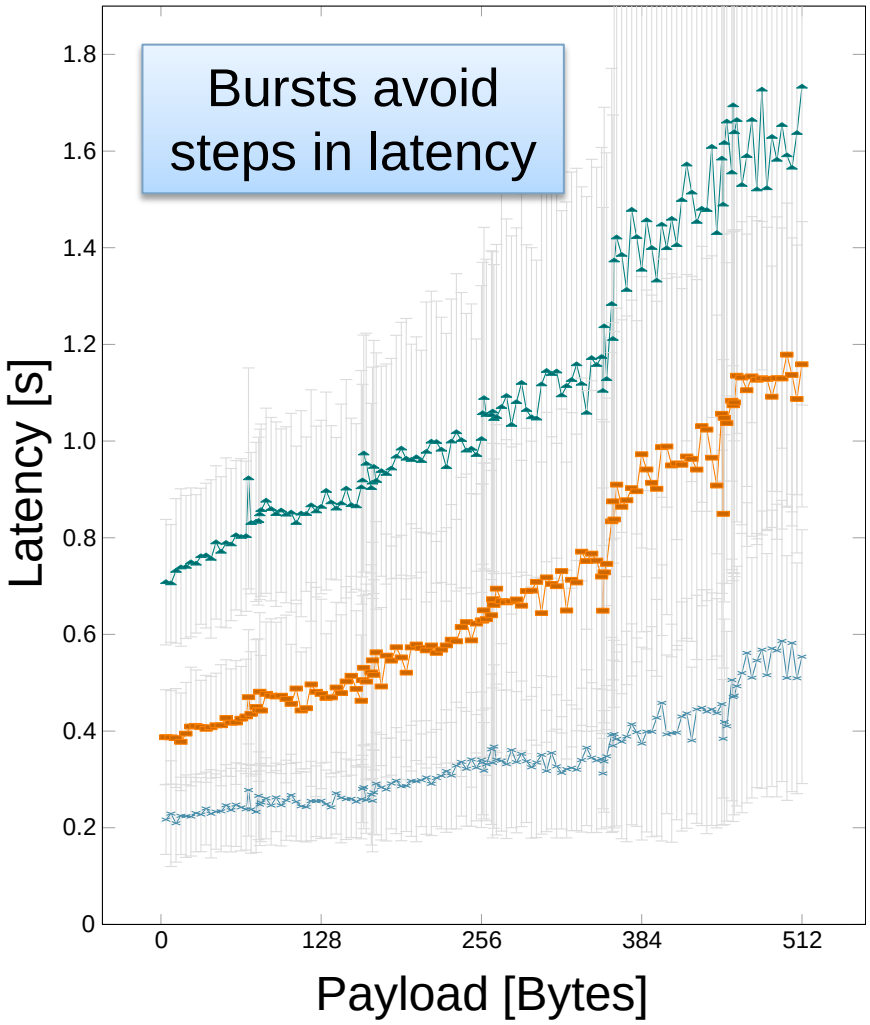


Energy vs Latency for 64B Payload



Transmitting Large Data





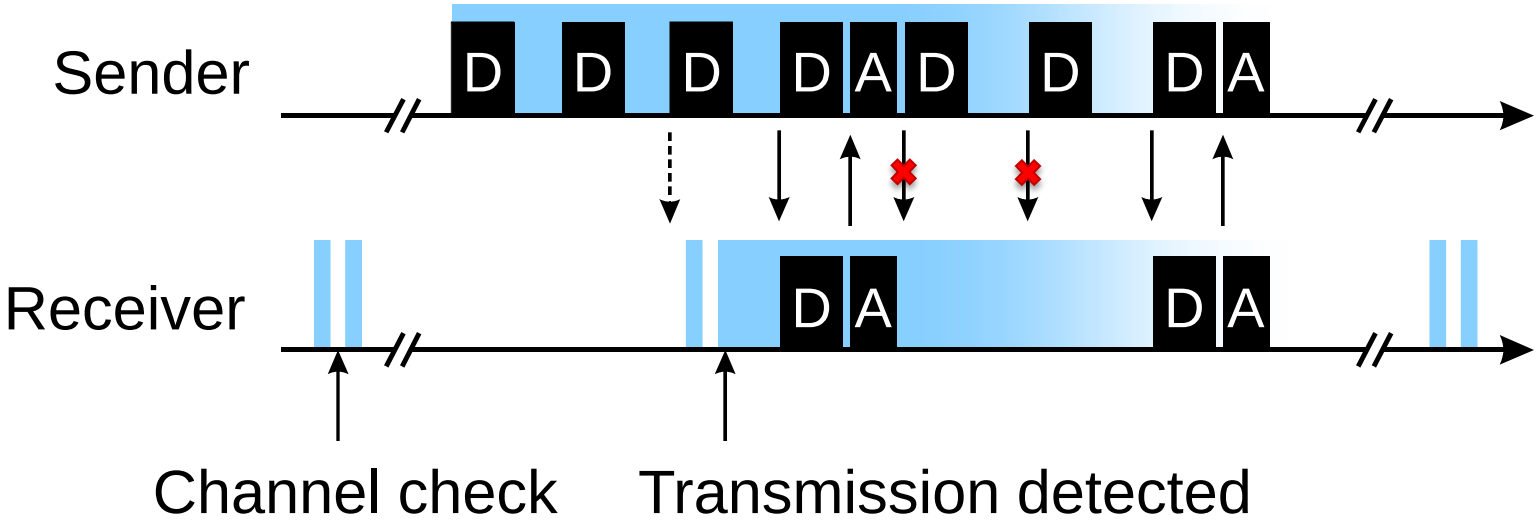
Transmitting Large Data

Radio on

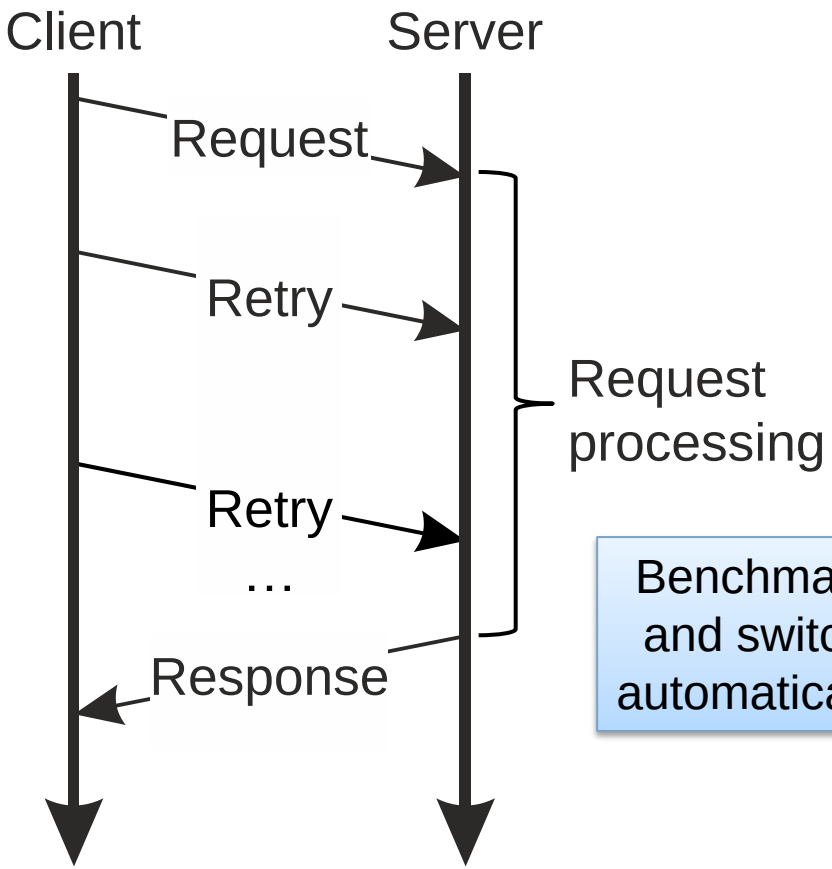
D Data frame

A ACK frame

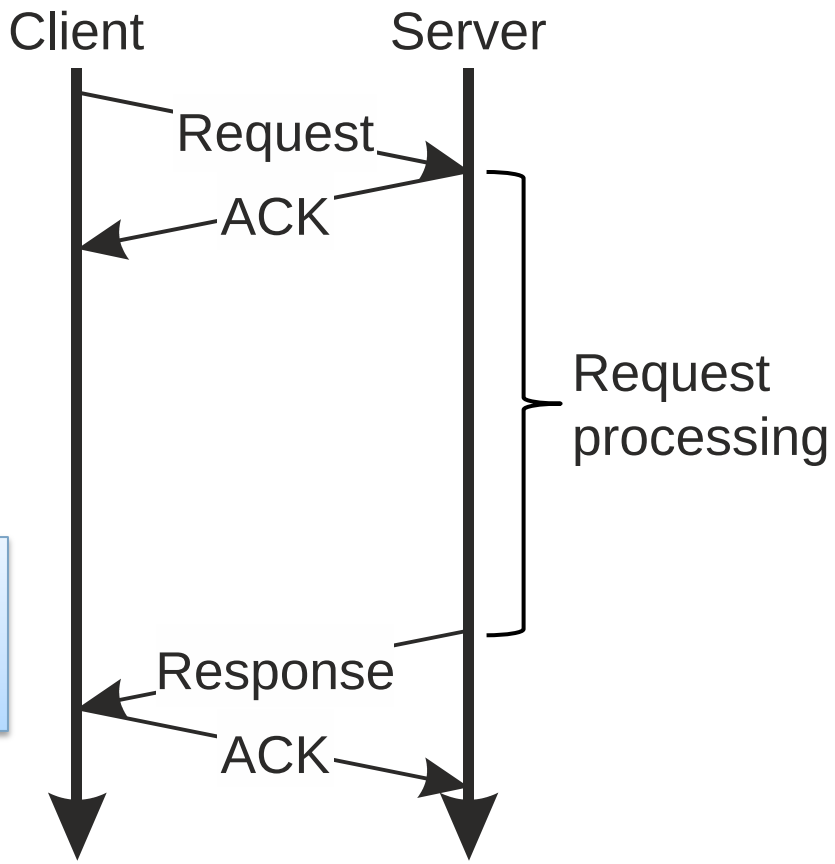
Number of frames sent depends on link quality, not number of fragments



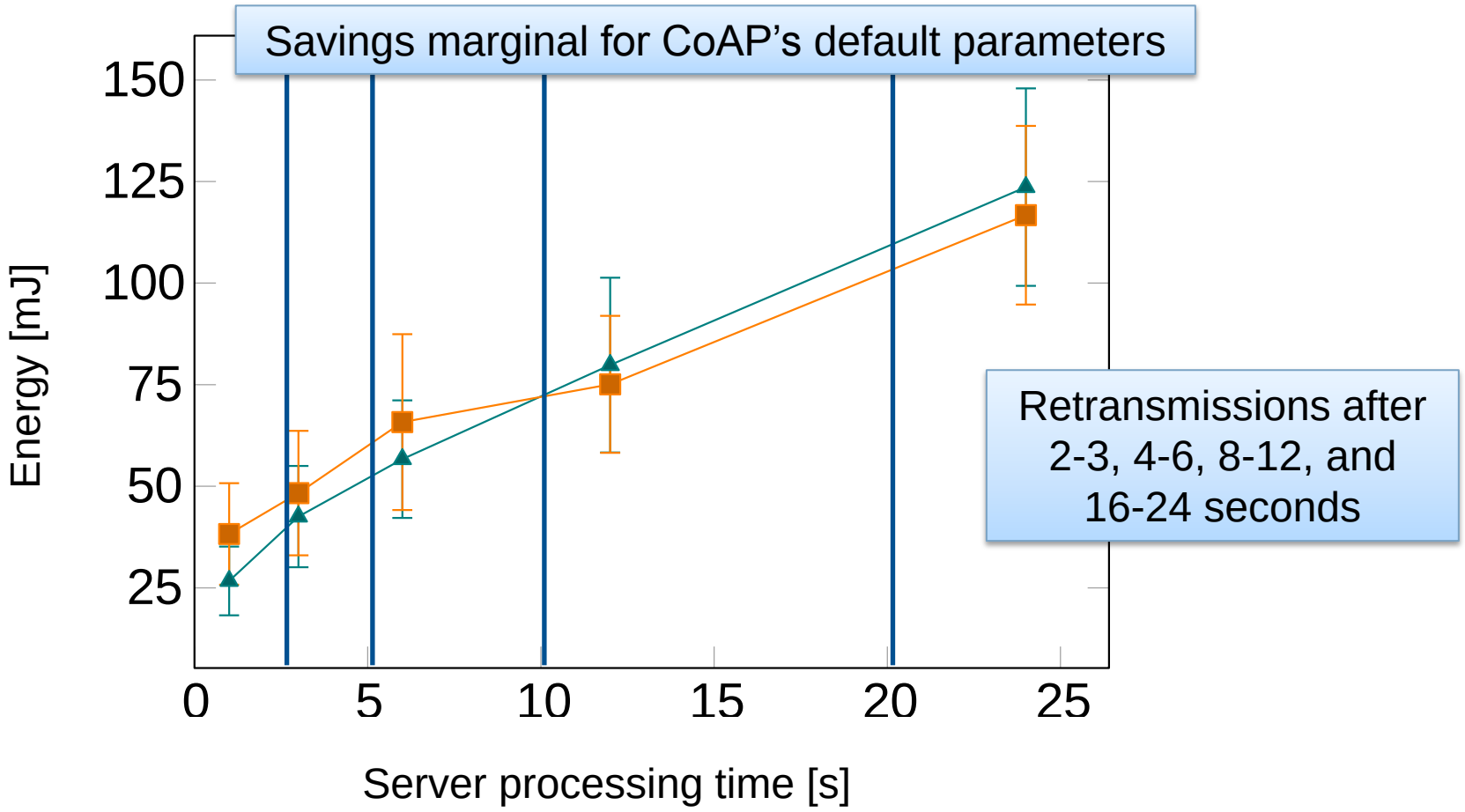
Separate Responses



Benchmark and switch automatically



Separate Responses over 4 Hops

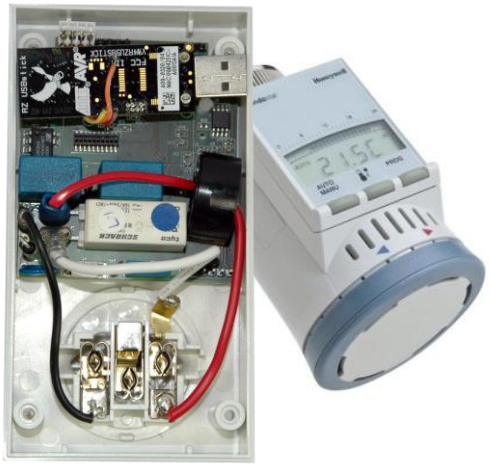


Conclusion

- Internet of Things stack
  - Isolated layers with different goals work together
  - Layered architecture lowers complexity
  - Optimize for single fragments
  - No need to optimize payload size once fragmented

- CoAP
  - Draft 07 implementation for Contiki
  - Block size most important parameter for memory-constrained devices: Exponential sizes too coarse-grained

Future Work



- Extend «Erbium» CoAP
  - Blocking resource handler for UART
  - Hierarchical resources / URI-Path handling
  - Improve observer handling (only one buffer)

- Study RESTful approach
  - Long-term deployment with smart appliances
  - Separate application logic from device firmware and combine with «Californium»-based App Server

https://github.com/mkovatsc/Californium

THANK YOU

Questions?

Blockwise Transfers

Automatic or controlled by handler

```
RESOURCE(handle, METHODs, URI-Path, Web Linking info);
```

- Resource handler

```
handle_handler(void* request, void* response,
               uint8_t *buffer, uint16_t preferred_size,
               int32_t *offset);
```

Unaware: fill buffer up to `REST_MAX_CHUNK_SIZE`

Chunk-aware: use `offset` and fill buffer up to `preferred_size`

(update `offset` and set to -1 when finished)

Observing Resources

Post-handler registers observers automatically

```
PERIODIC_RESOURCE(handle, METHODs, URI-Path, info, period);
```

- Additional periodic handler

```
EVENT_RESOURCE(handle, METHODs, URI-Path, info);
```

- Additional event handler

```
int handle_event_handler(resource_t *r) {
  ...
  REST.notify_subscribers(r->url, CON/NON, seqno, buf, len);
}
```

Every x-th is CON to resolve orphans

«Erbium» REST Engine

- Manipulate messages with *REST* API
  - `REST.set_header_etag(response, etag_buf, etag_len);`
  - `REST.get_query_variable(request, name, value_buffer);`

- Link implementation
  - coap-03
  - coap-06
  - coap-07
  - …
  - HTTP (not yet implemented for Erbium)