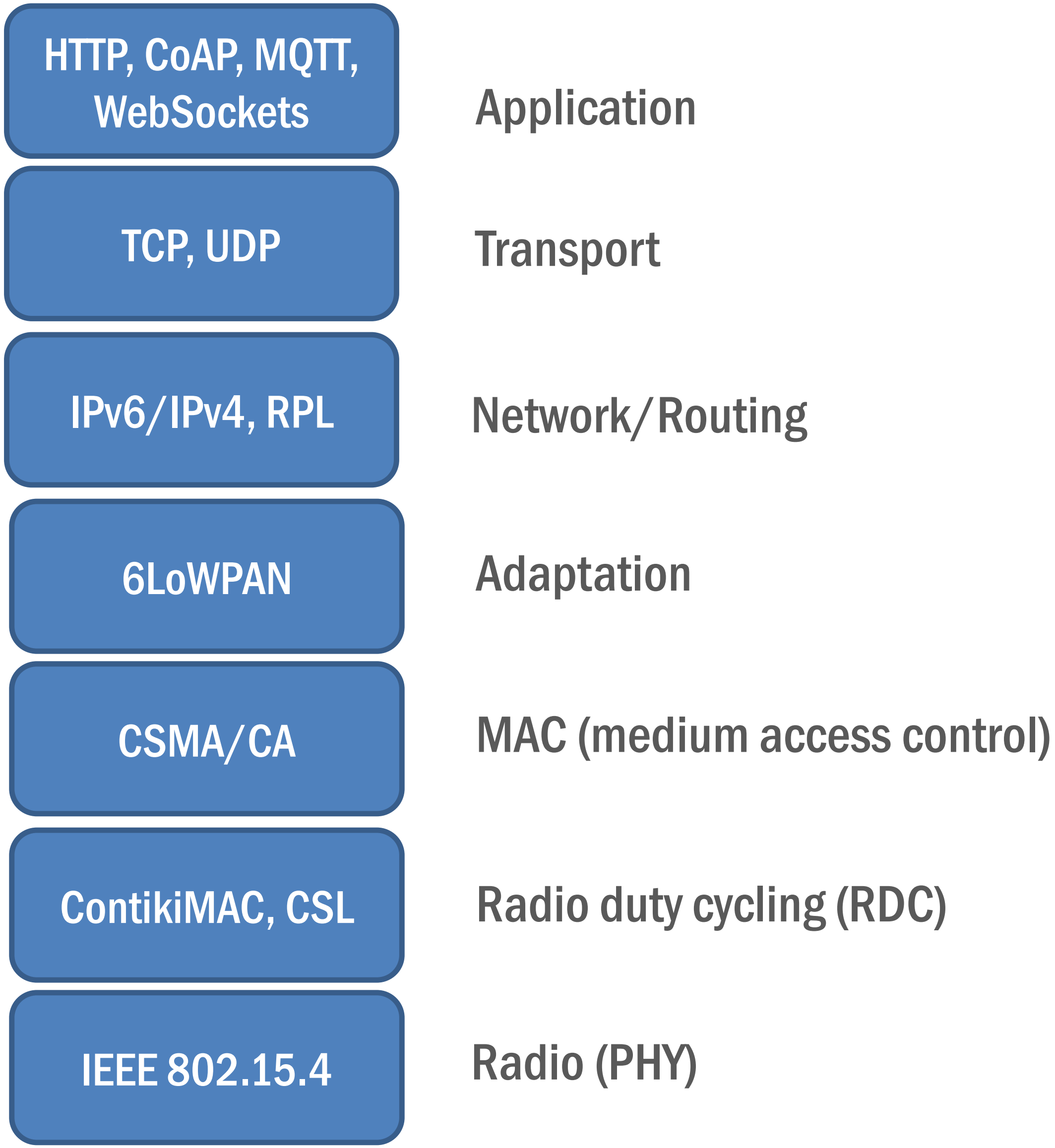
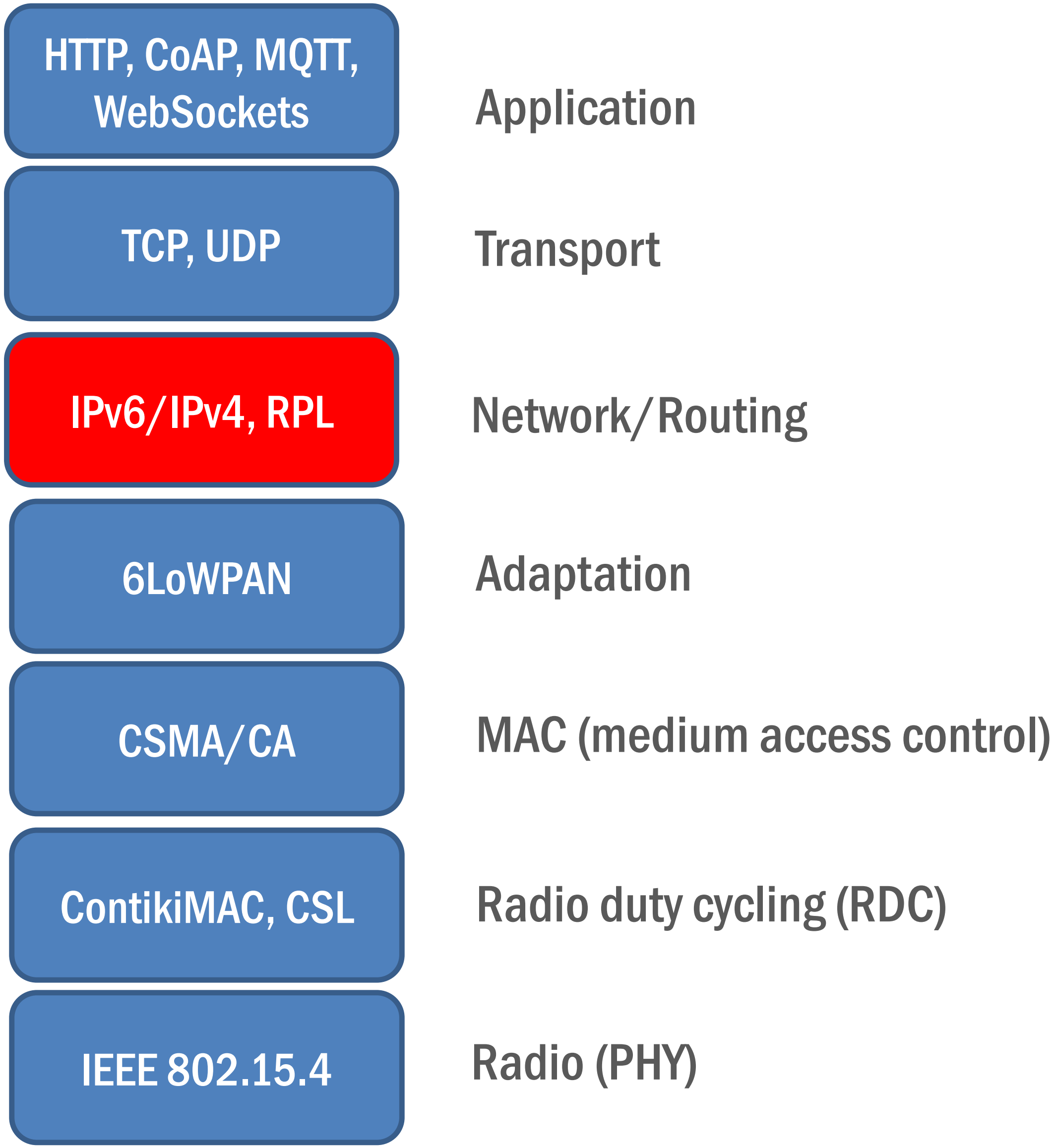
UDP on 6LoWPAN and IPv6

Antonio Liñán Colina,
Zolertia

Contiki Stack



HTTP, CoAP, MQTT, WebSockets	Application
TCP, UDP	Transport
IPv6/IPv4, RPL	Network/Routing
6LoWPAN	Adaptation
CSMA/CA	MAC (medium access control)
ContikiMAC, CSL	Radio duty cycling (RDC)
IEEE 802.15.4	Radio (PHY)



HTTP, CoAP, MQTT, WebSockets — Application

TCP, UDP — Transport

IPv6/IPv4, RPL — Network/Routing

6LoWPAN — Adaptation

CSMA/CA — MAC (medium access control)

ContikiMAC, CSL — Radio duty cycling (RDC)

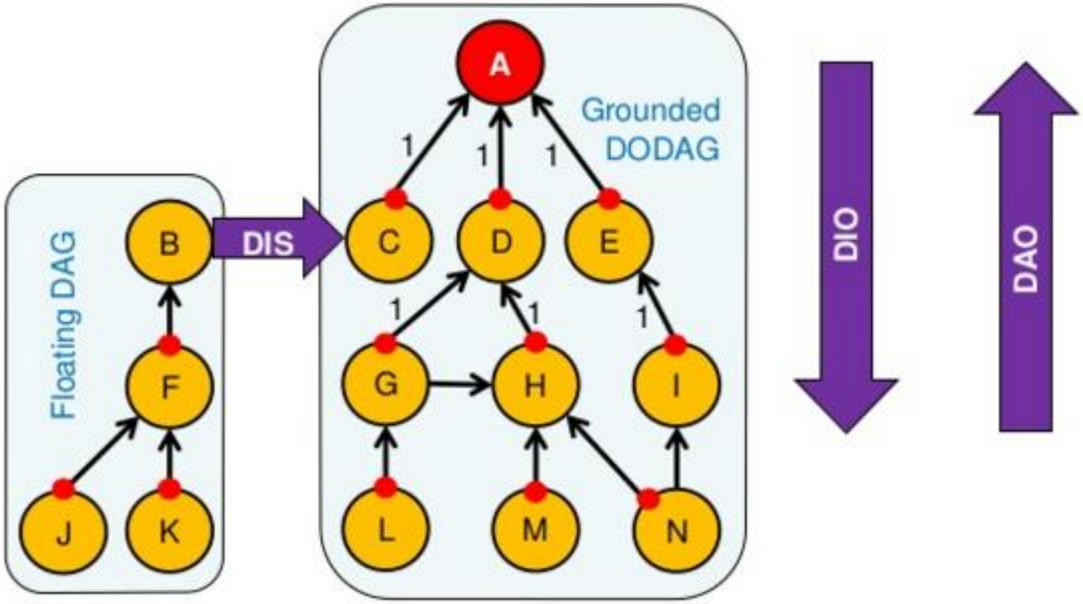
IEEE 802.15.4 — Radio (PHY)

RPL: IPv6 Routing Protocol for Low power and Lossy Networks

Directed Acyclic Graph **(DAG)**
Destination Oriented DAG **(DODAG)**

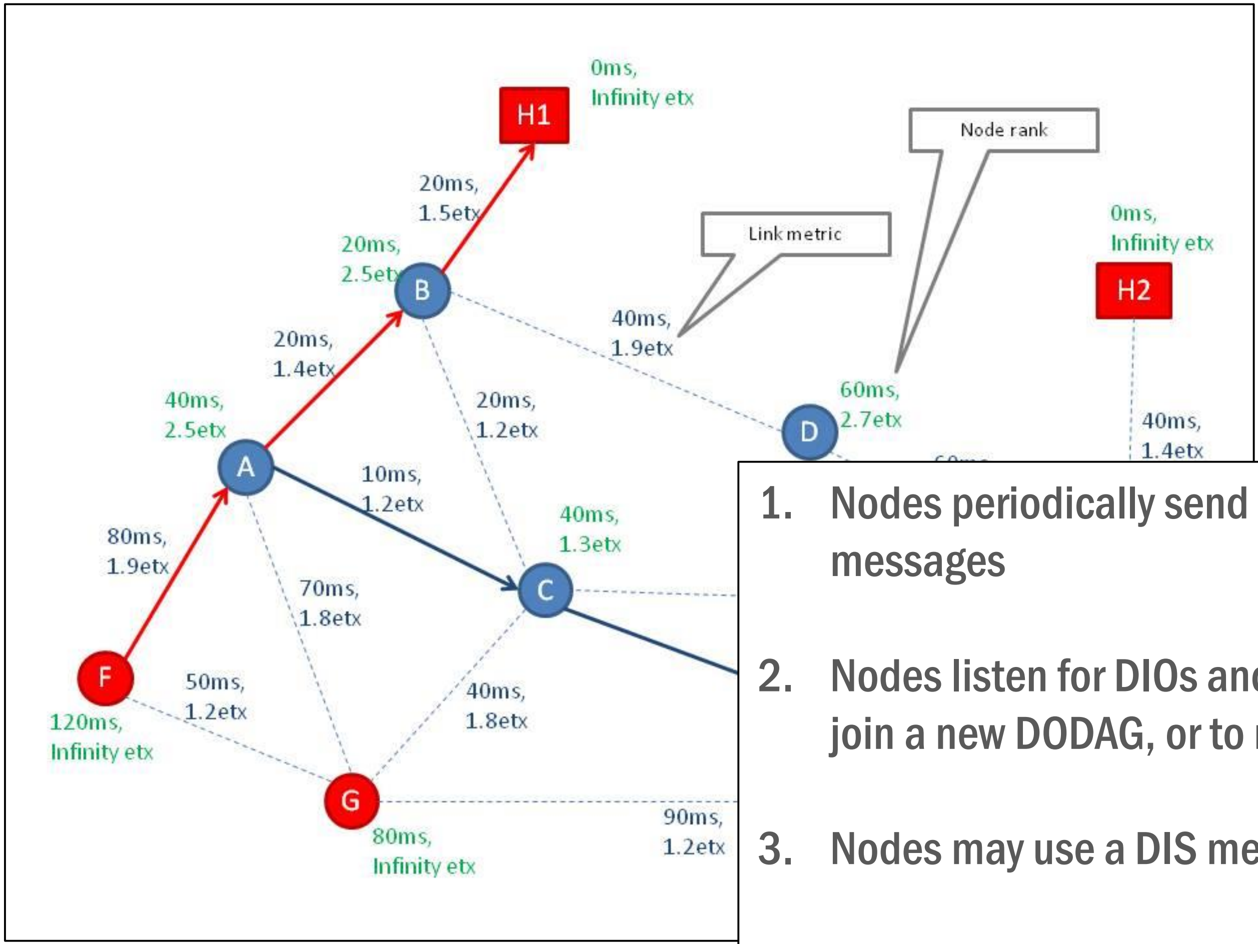
ICMPv6 control messages

- DAG Information Object (DIO): sends DODAG information downwards
- Destination Advertisement Object (DAO): sends destination information upwards
- DAG Information Solicitation (DIS): requests a DIO



https://tools.ietf.org/id/draft-ietf-roll-rpl-19.txt

RPL: IPv6 Routing Protocol for Low power and Lossy Networks



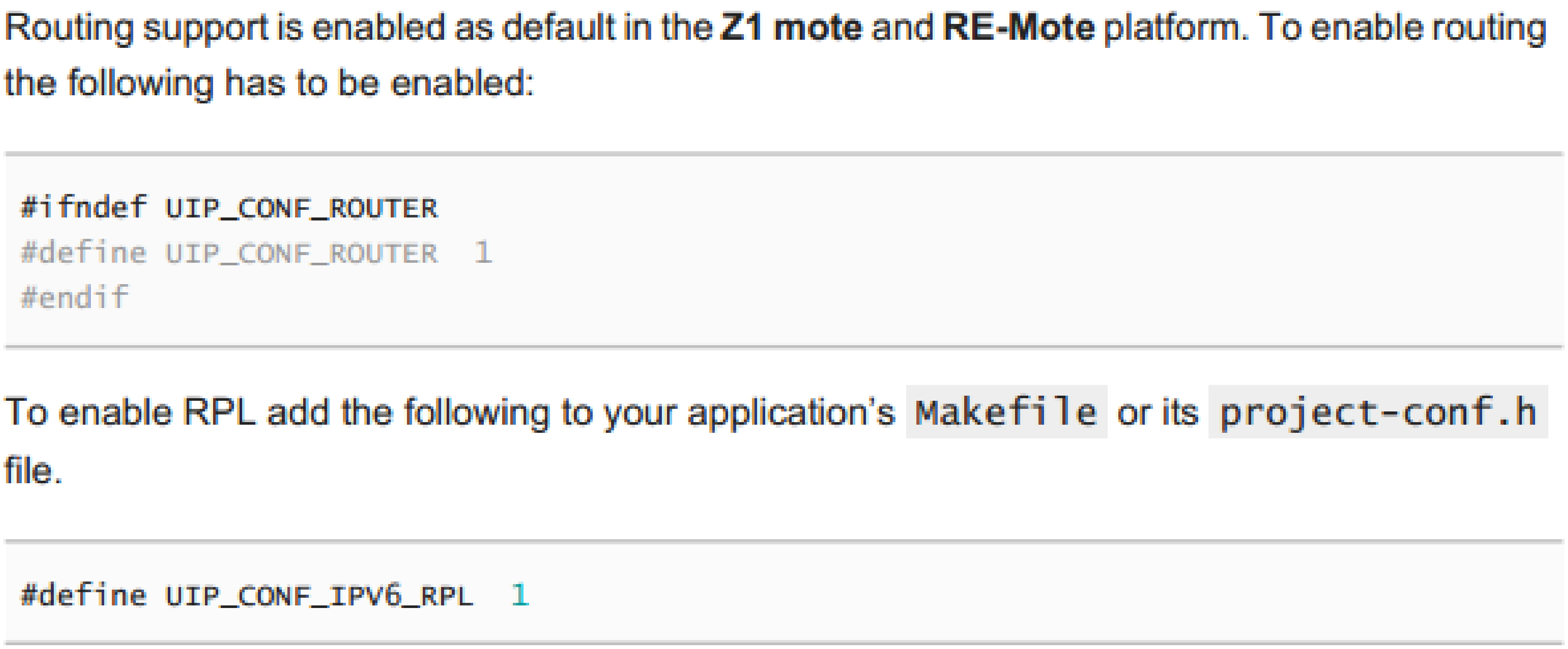
1. Nodes periodically send link-local multicast DIO messages

2. Nodes listen for DIOs and use their information to join a new DODAG, or to maintain an existing DODAG

3. Nodes may use a DIS message to solicit a DIO

4. Based on information in the DIOs the node chooses parents that minimize path cost to the DODAG root

https://tools.ietf.org/id/draft-ietf-roll-rpl-19.txt

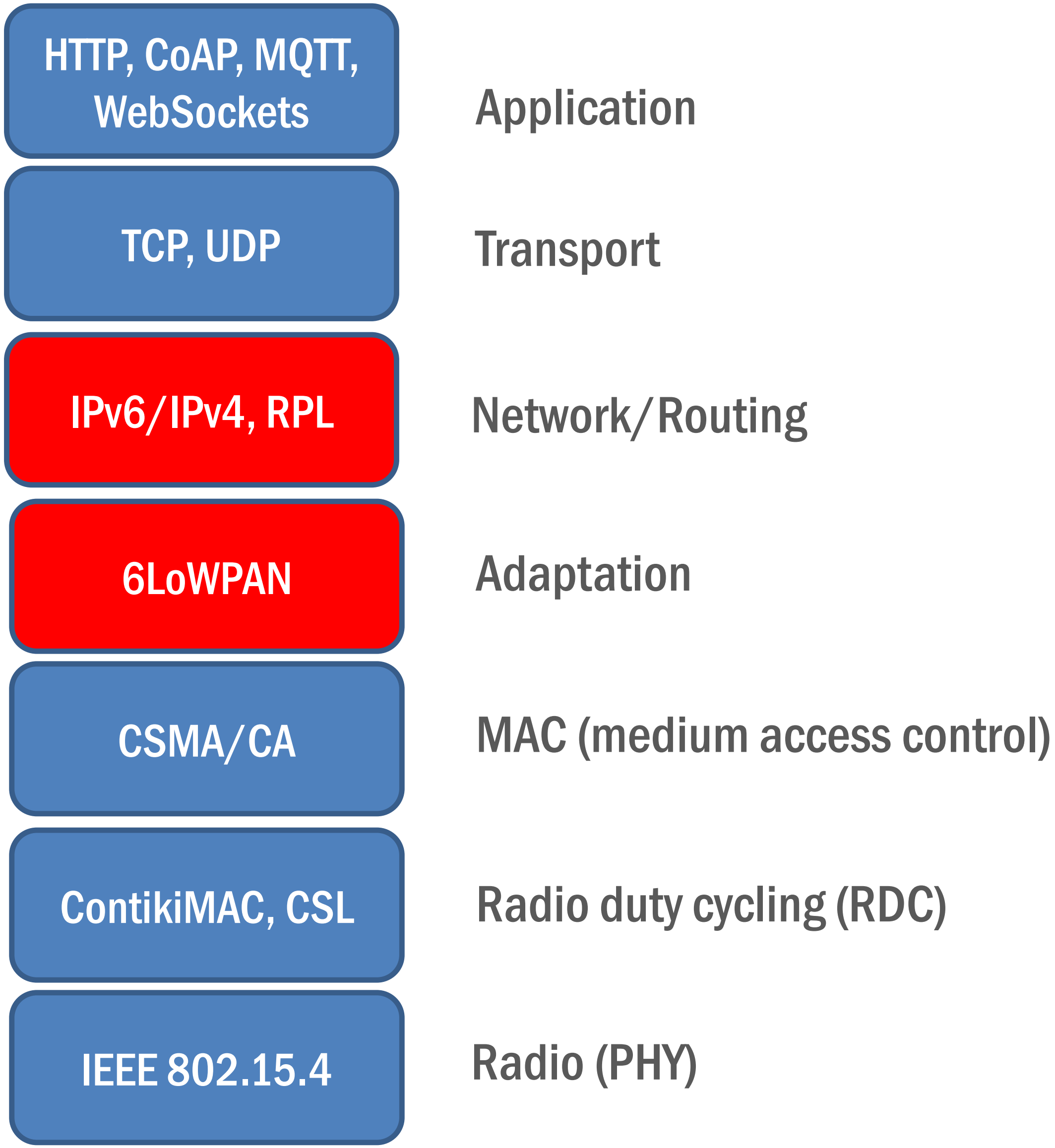


Routing support is enabled as default in the **Z1 mote** and **RE-Mote** platform. To enable routing the following has to be enabled:

```
#ifndef UIP_CONF_ROUTER
#define UIP_CONF_ROUTER  1
#endif
```

To enable RPL add the following to your application's `Makefile` or its `project-conf.h` file.

```
#define UIP_CONF_IPV6_RPL  1
```



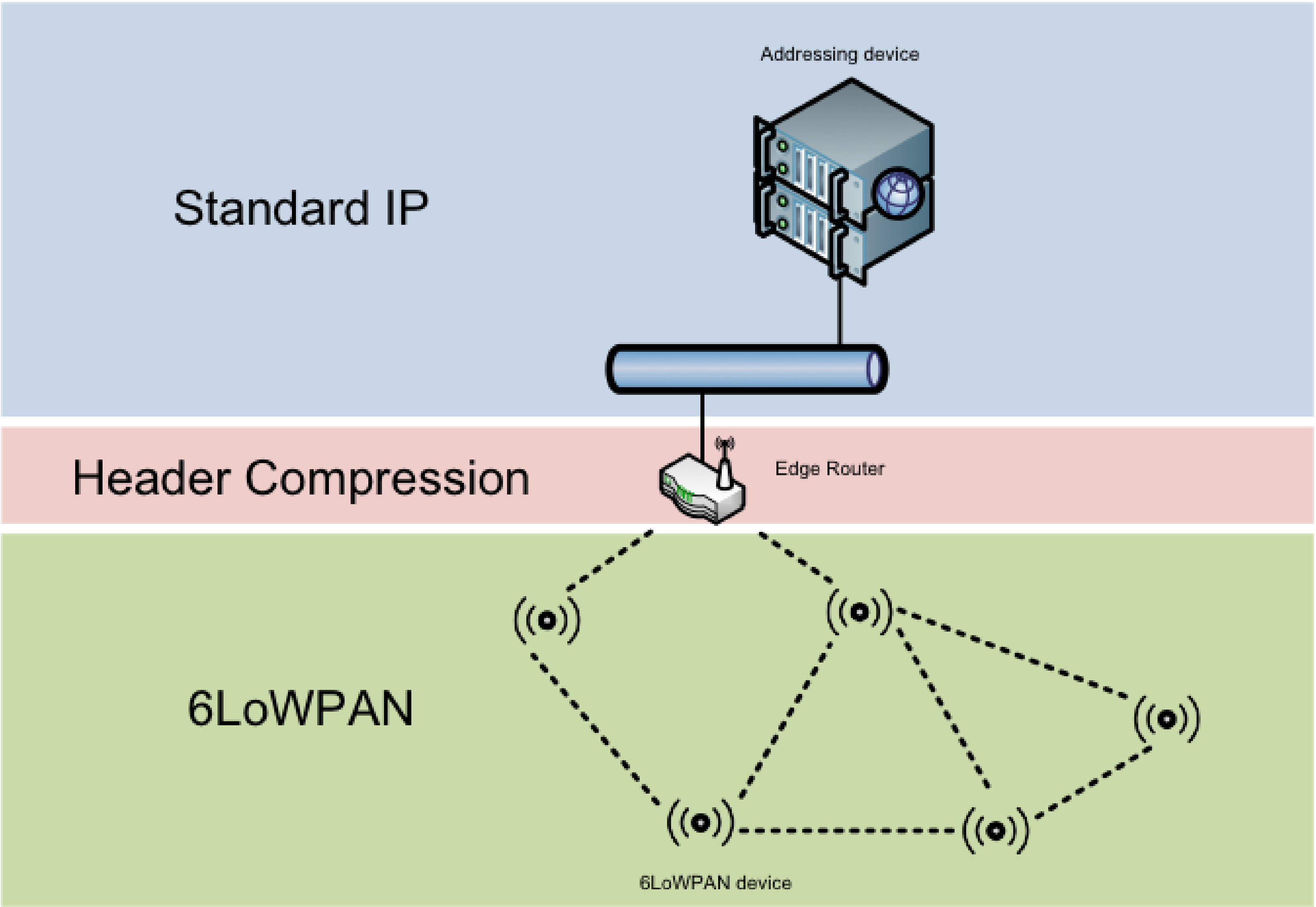
HTTP, CoAP, MQTT, WebSockets	Application
TCP, UDP	Transport
IPv6/IPv4, RPL	Network/Routing
6LoWPAN	Adaptation
CSMA/CA	MAC (medium access control)
ContikiMAC, CSL	Radio duty cycling (RDC)
IEEE 802.15.4	Radio (PHY)

Header Size Calculation

- IPv6 header is 40 octets, UDP header is 8 octets
- 802.15.4 MAC header can be up to 25 octets (null security) or 25+21=46 octets (AES-CCM-128)
- With the 802.15.4 frame size of 127 octets, we have only following space left for application data!
  - 127-25-40-8 = 54 octets (null security)
  - 127-46-40-8 = 33 octets (AES-CCM-128)

IPv6 MTU Requirements

- IPv6 requires that links support an MTU of 1280 octets
- Link-layer fragmentation / reassembly is needed

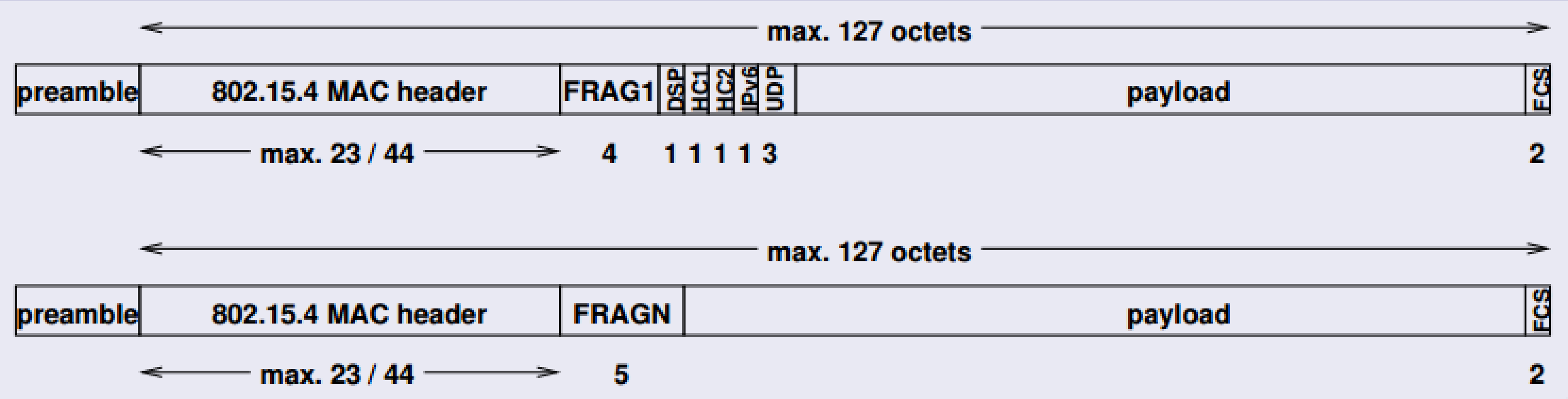


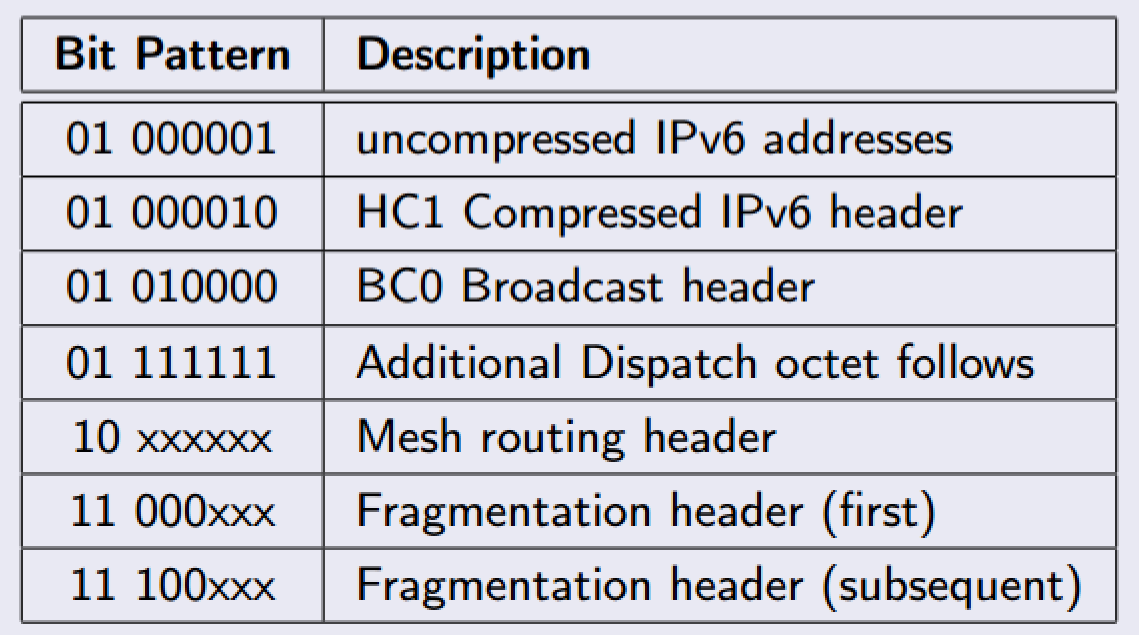
Standard IP

Header Compression

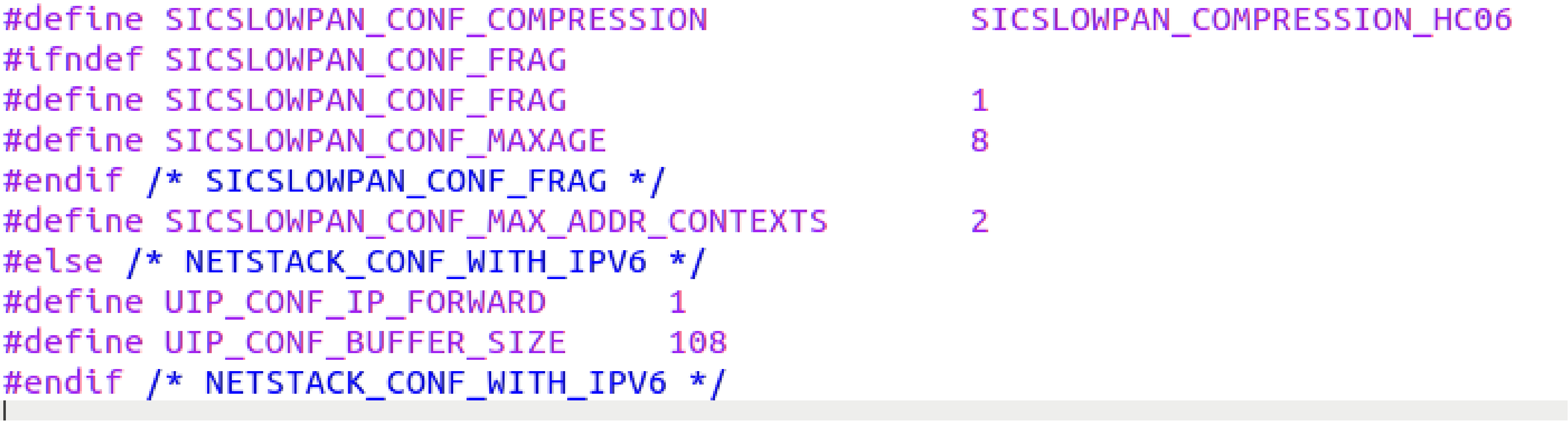
6LoWPAN

Fragmentation Example (compressed link-local IPv6/UDP)





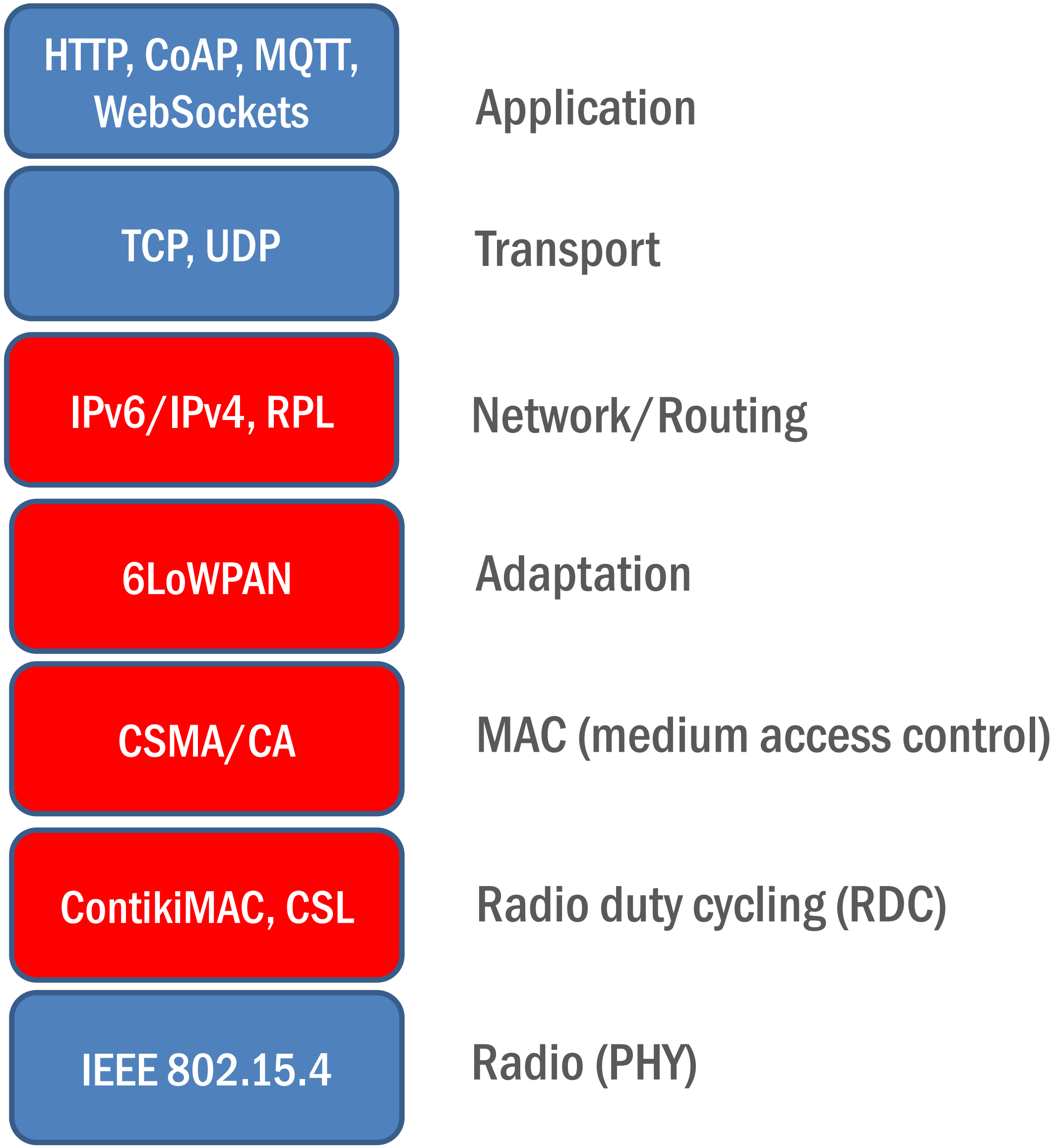
Bit Pattern	Description
01 000001	uncompressed IPv6 addresses
01 000010	HC1 Compressed IPv6 header
01 010000	BC0 Broadcast header
01 111111	Additional Dispatch octet follows
10 xxxxxx	Mesh routing header
11 000xxx	Fragmentation header (first)
11 100xxx	Fragmentation header (subsequent)



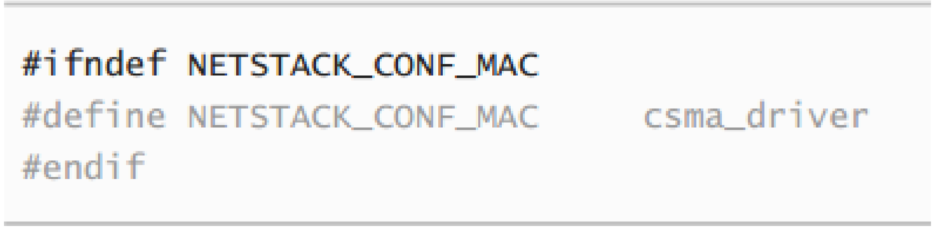
```
#define SICSLOWPAN_CONF_COMPRESSION             SICSLOWPAN_COMPRESSION_HC06
#ifndef SICSLOWPAN_CONF_FRAG
#define SICSLOWPAN_CONF_FRAG                    1
#define SICSLOWPAN_CONF_MAXAGE                  8
#endif /* SICSLOWPAN_CONF_FRAG */
#define SICSLOWPAN_CONF_MAX_ADDR_CONTEXTS       2
#else /* NETSTACK_CONF_WITH_IPV6 */
#define UIP_CONF_IP_FORWARD      1
#define UIP_CONF_BUFFER_SIZE     108
#endif /* NETSTACK_CONF_WITH_IPV6 */
```

core/net/ipv6/sicslowpan.c
platforms/z1/contiki-conf.h

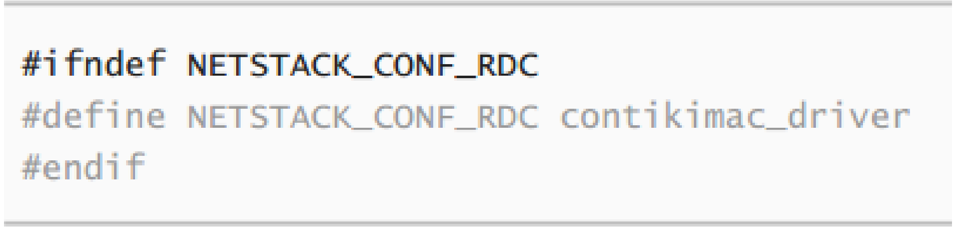
https://tools.ietf.org/html/draft-ietf-6lowpan-hc-06



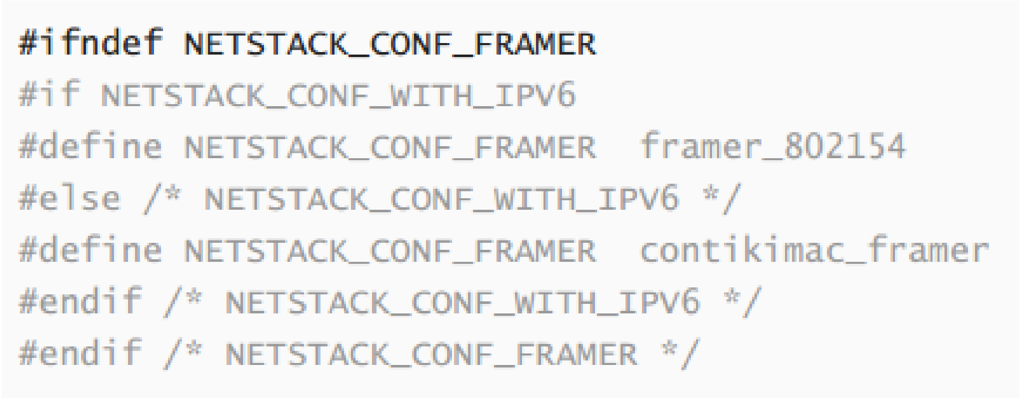
HTTP, CoAP, MQTT, WebSockets	Application
TCP, UDP	Transport
IPv6/IPv4, RPL	Network/Routing
6LoWPAN	Adaptation
CSMA/CA	MAC (medium access control)
ContikiMAC, CSL	Radio duty cycling (RDC)
IEEE 802.15.4	Radio (PHY)



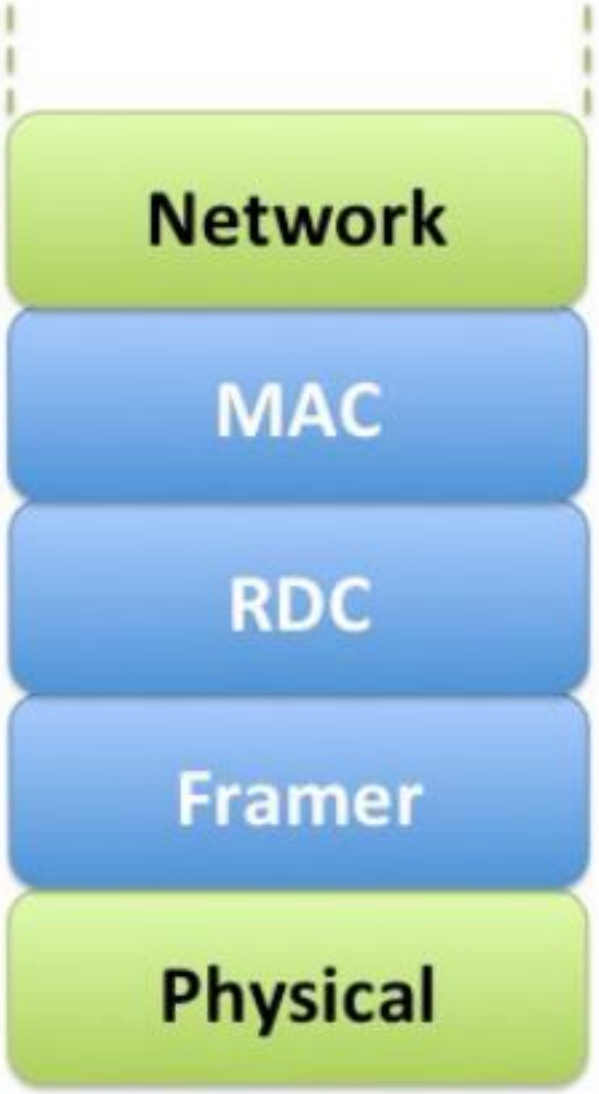
```
#ifndef NETSTACK_CONF_MAC
#define NETSTACK_CONF_MAC     csma_driver
#endif
```

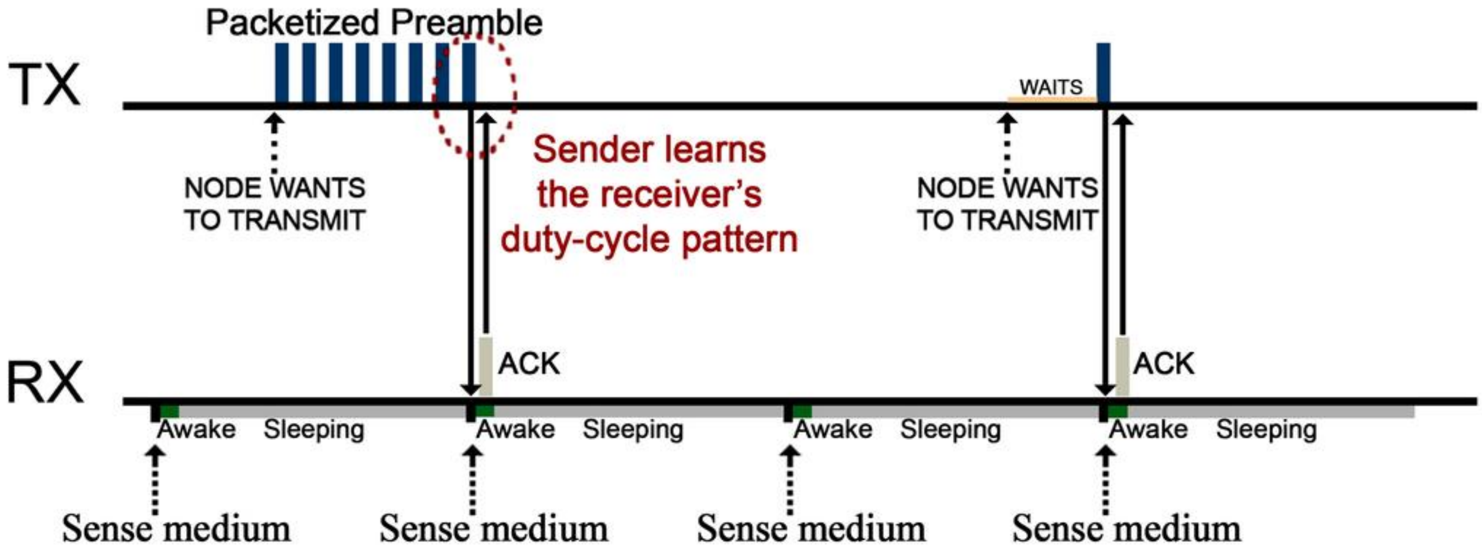


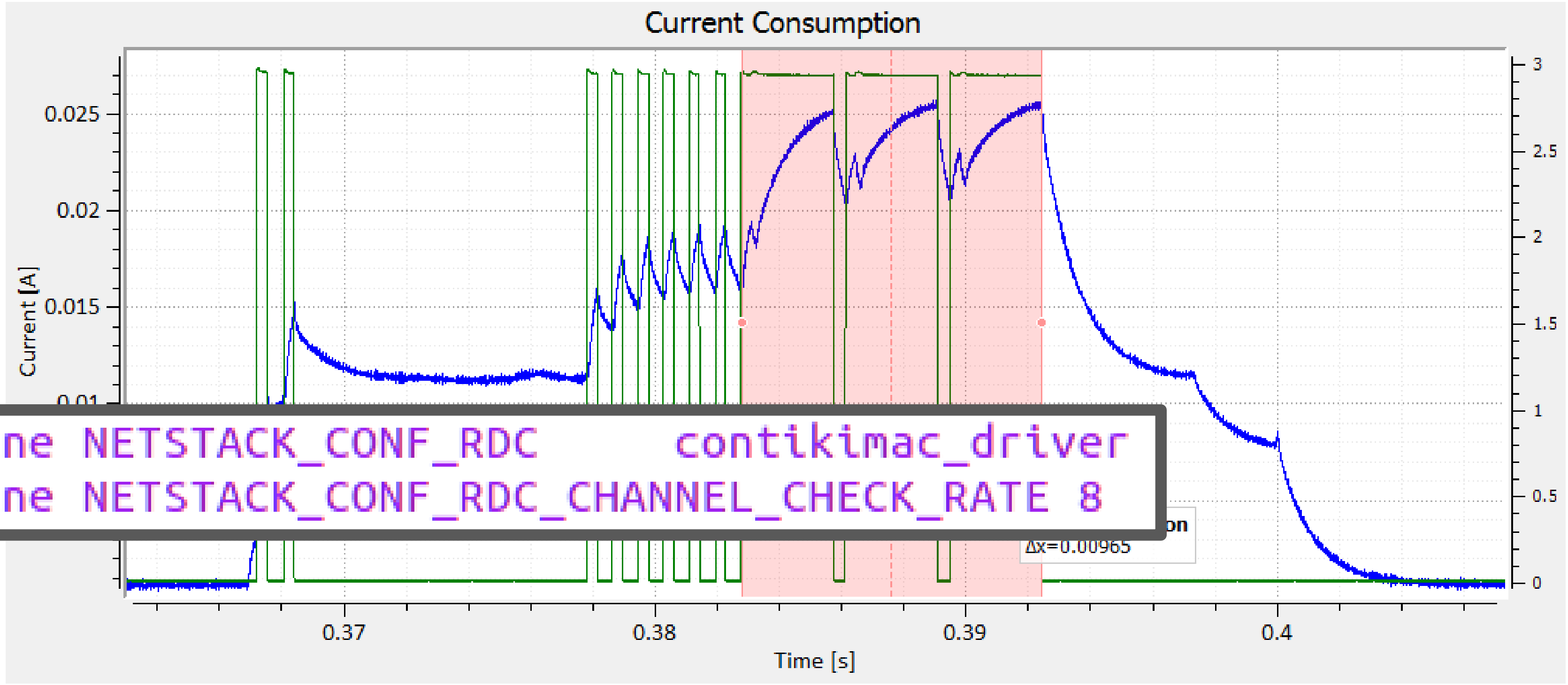
```
#ifndef NETSTACK_CONF_RDC
#define NETSTACK_CONF_RDC contikimac_driver
#endif
```



```
#ifndef NETSTACK_CONF_FRAMER
#if NETSTACK_CONF_WITH_IPV6
#define NETSTACK_CONF_FRAMER  framer_802154
#else /* NETSTACK_CONF_WITH_IPV6 */
#define NETSTACK_CONF_FRAMER  contikimac_framer
#endif /* NETSTACK_CONF_WITH_IPV6 */
#endif /* NETSTACK_CONF_FRAMER */
```

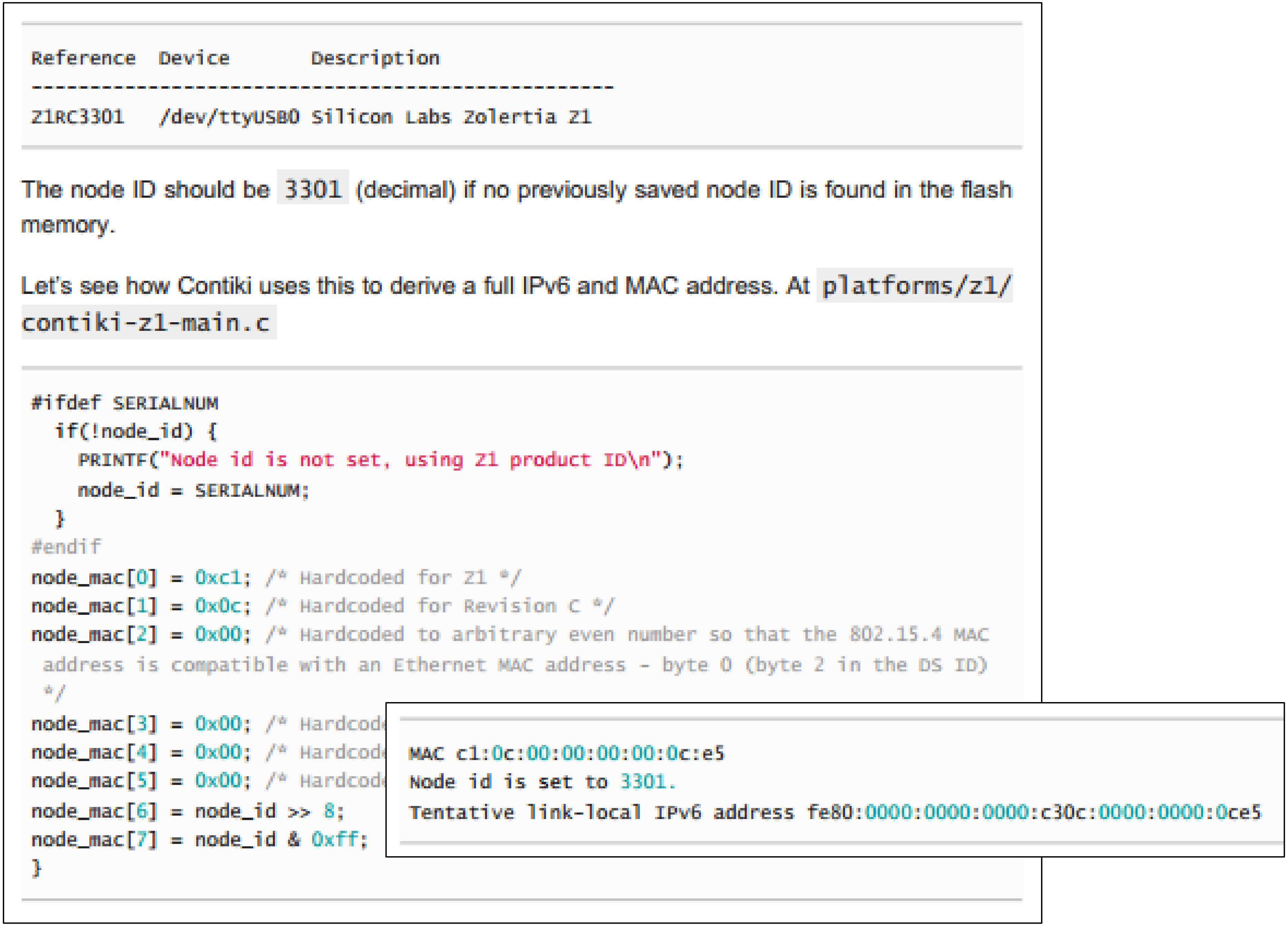






```
#define NETSTACK_CONF_RDC      contikimac_driver
#define NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 8
```

02-ipv6



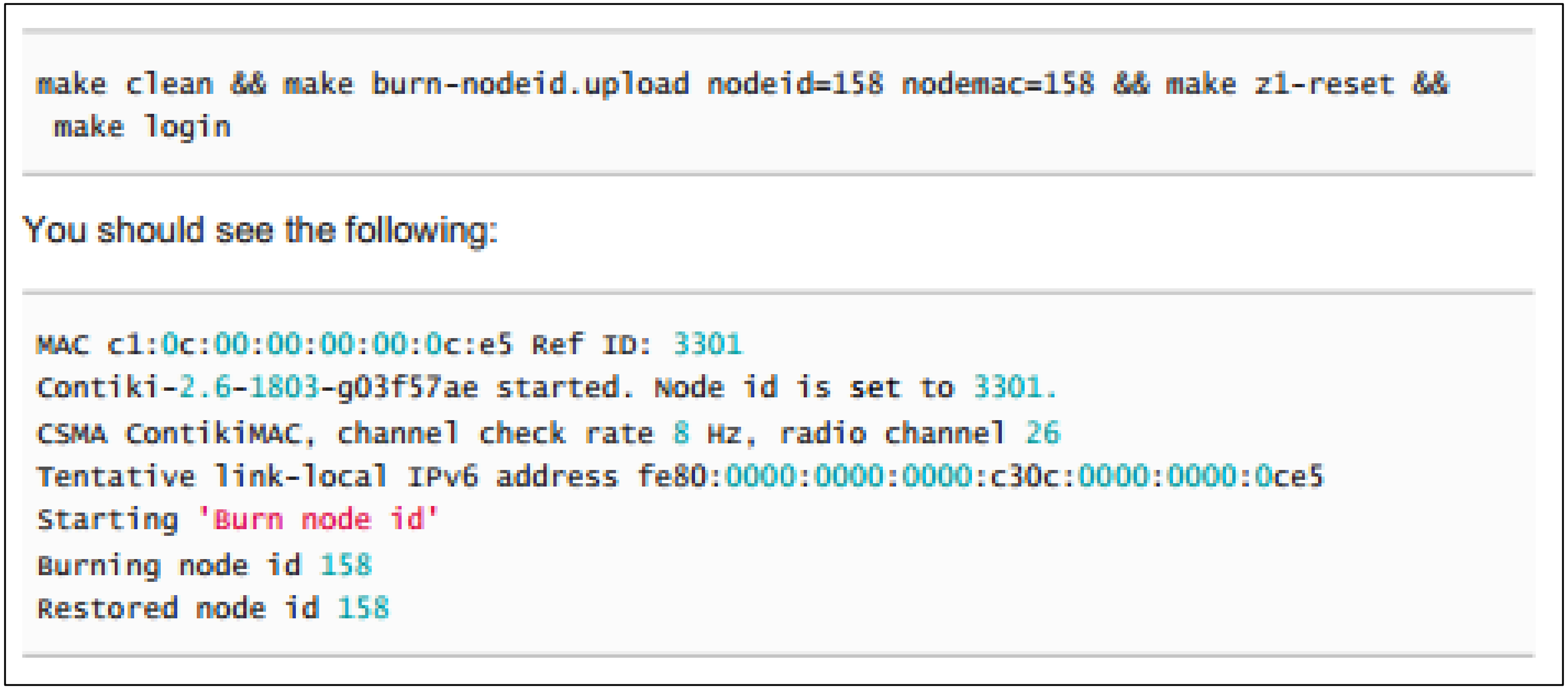
```
Reference  Device       Description
---------------------------------------------------
Z1RC3301   /dev/ttyUSB0 Silicon Labs Zolertia Z1
```

The node ID should be `3301` (decimal) if no previously saved node ID is found in the flash memory.

Let's see how Contiki uses this to derive a full IPv6 and MAC address. At `platforms/z1/contiki-z1-main.c`

```
#ifdef SERIALNUM
  if(!node_id) {
    PRINTF("Node id is not set, using Z1 product ID\n");
    node_id = SERIALNUM;
  }
#endif
node_mac[0] = 0xc1; /* Hardcoded for Z1 */
node_mac[1] = 0x0c; /* Hardcoded for Revision C */
node_mac[2] = 0x00; /* Hardcoded to arbitrary even number so that the 802.15.4 MAC
 address is compatible with an Ethernet MAC address - byte 0 (byte 2 in the DS ID)
 */
node_mac[3] = 0x00; /* Hardcode
node_mac[4] = 0x00; /* Hardcode
node_mac[5] = 0x00; /* Hardcode
node_mac[6] = node_id >> 8;
node_mac[7] = node_id & 0xff;
}
```

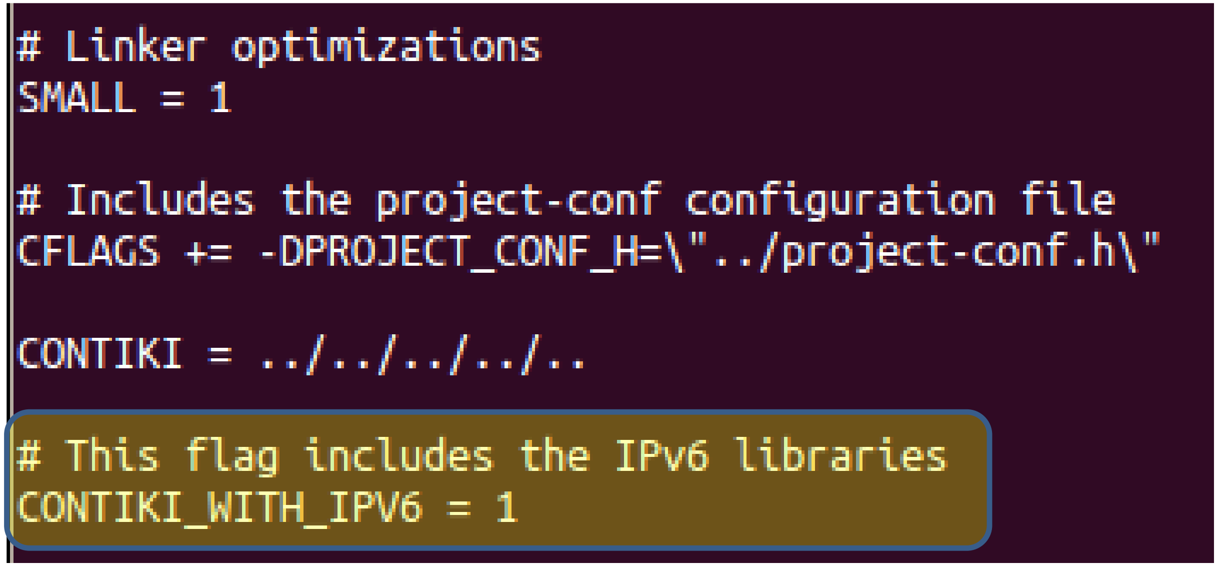
```
MAC c1:0c:00:00:00:00:0c:e5
Node id is set to 3301.
Tentative link-local IPv6 address fe80:0000:0000:0000:c30c:0000:0000:0ce5
```



```
make clean && make burn-nodeid.upload nodeid=158 nodemac=158 && make z1-reset &&
 make login
```

You should see the following:

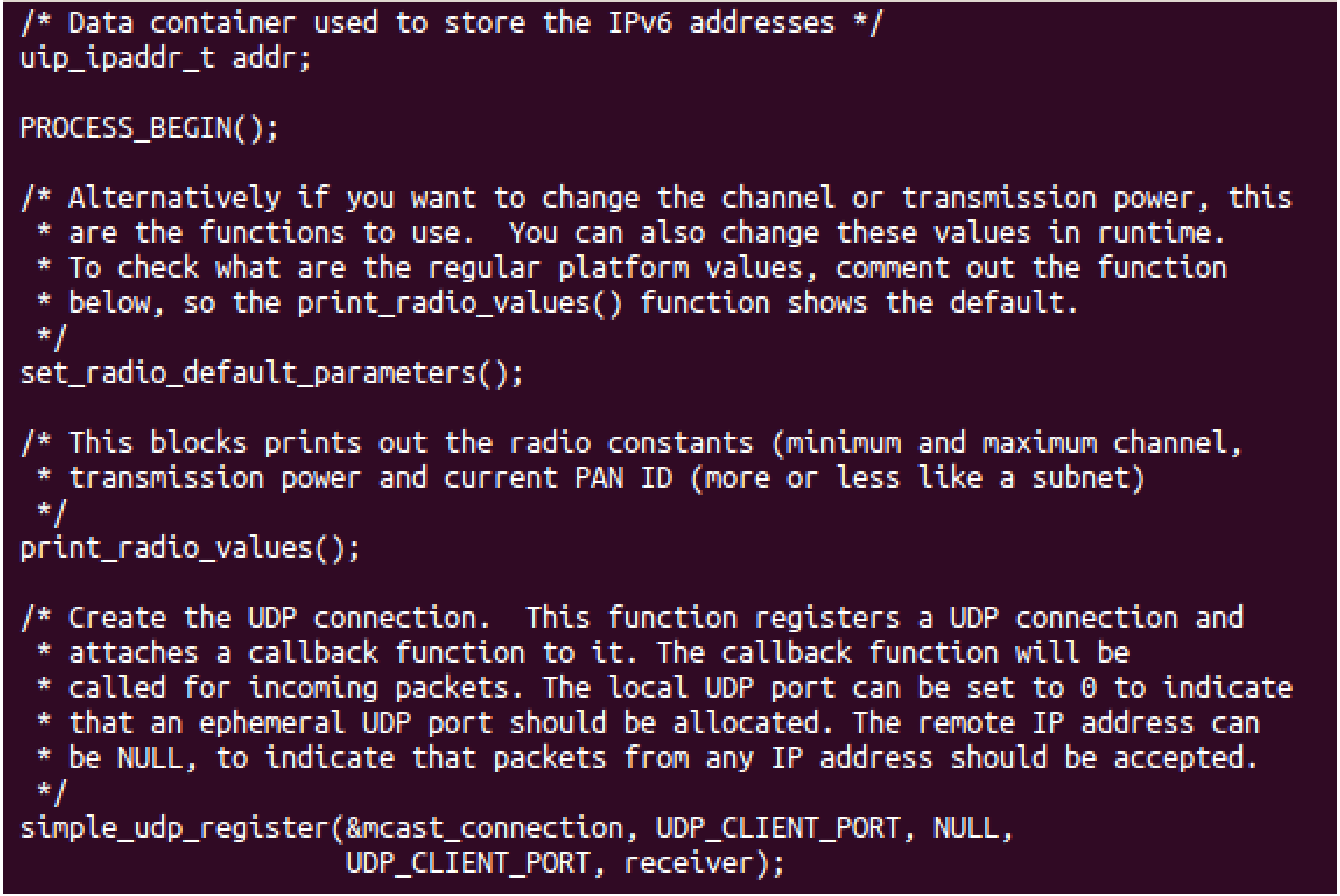
```
MAC c1:0c:00:00:00:00:0c:e5 Ref ID: 3301
Contiki-2.6-1803-g03f57ae started. Node id is set to 3301.
CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
Tentative link-local IPv6 address fe80:0000:0000:0000:c30c:0000:0000:0ce5
Starting 'Burn node id'
Burning node id 158
Restored node id 158
```



```
# Linker optimizations
SMALL = 1

# Includes the project-conf configuration file
CFLAGS += -DPROJECT_CONF_H=\"../project-conf.h\"

CONTIKI = ../../../../..

# This flag includes the IPv6 libraries
CONTIKI_WITH_IPV6 = 1
```

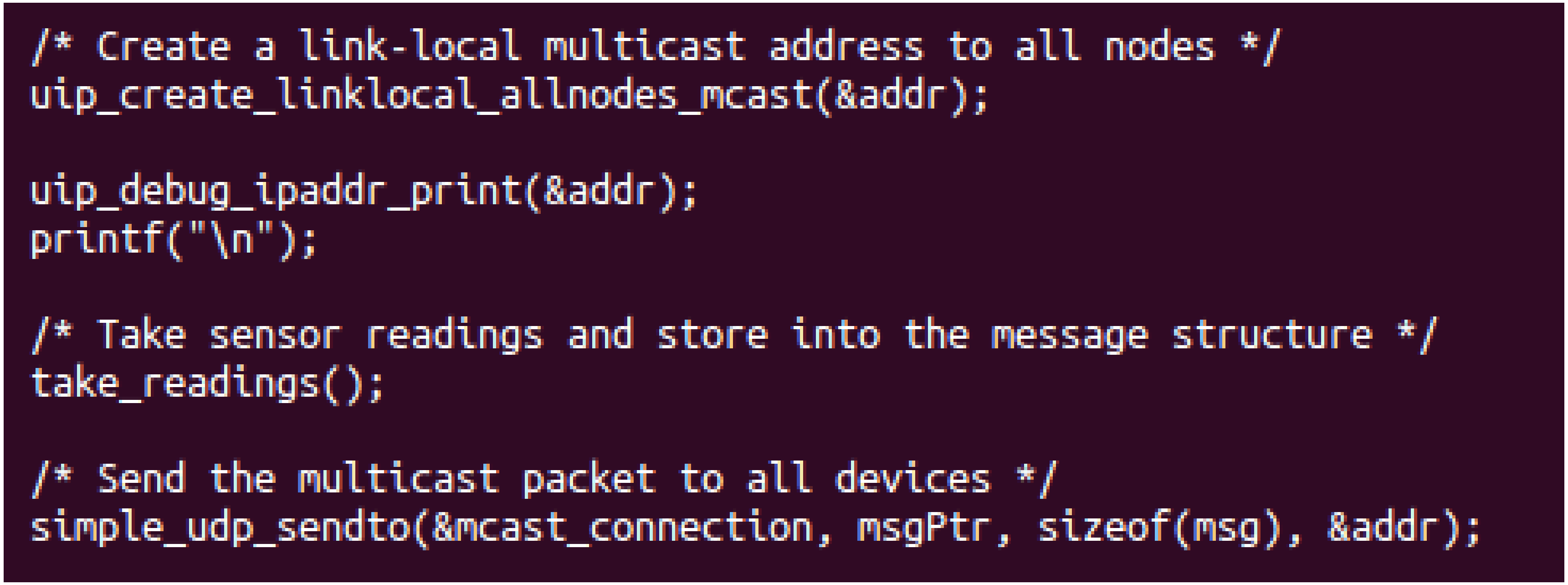
This flag enables IPv6 support

examples/zolertia/tutorial/02-ipv6/01-udp-local-multicast/Makefile



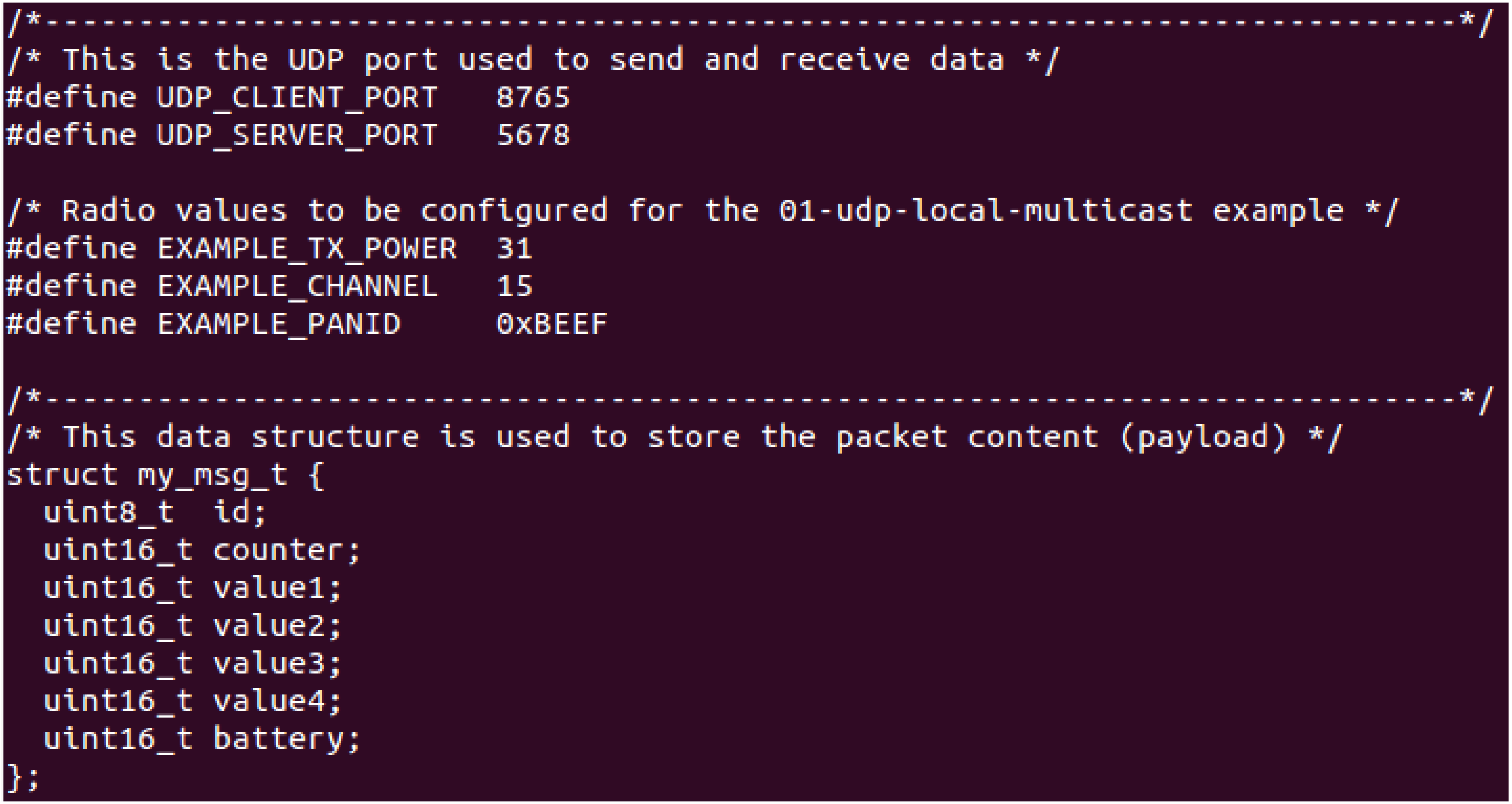
```
/* Data container used to store the IPv6 addresses */
uip_ipaddr_t addr;

PROCESS_BEGIN();

/* Alternatively if you want to change the channel or transmission power, this
 * are the functions to use.  You can also change these values in runtime.
 * To check what are the regular platform values, comment out the function
 * below, so the print_radio_values() function shows the default.
 */
set_radio_default_parameters();

/* This blocks prints out the radio constants (minimum and maximum channel,
 * transmission power and current PAN ID (more or less like a subnet)
 */
print_radio_values();

/* Create the UDP connection.  This function registers a UDP connection and
 * attaches a callback function to it. The callback function will be
 * called for incoming packets. The local UDP port can be set to 0 to indicate
 * that an ephemeral UDP port should be allocated. The remote IP address can
 * be NULL, to indicate that packets from any IP address should be accepted.
 */
simple_udp_register(&mcast_connection, UDP_CLIENT_PORT, NULL,
                    UDP_CLIENT_PORT, receiver);
```

examples/zolertia/tutorial/02-ipv6/01-udp-local-multicast



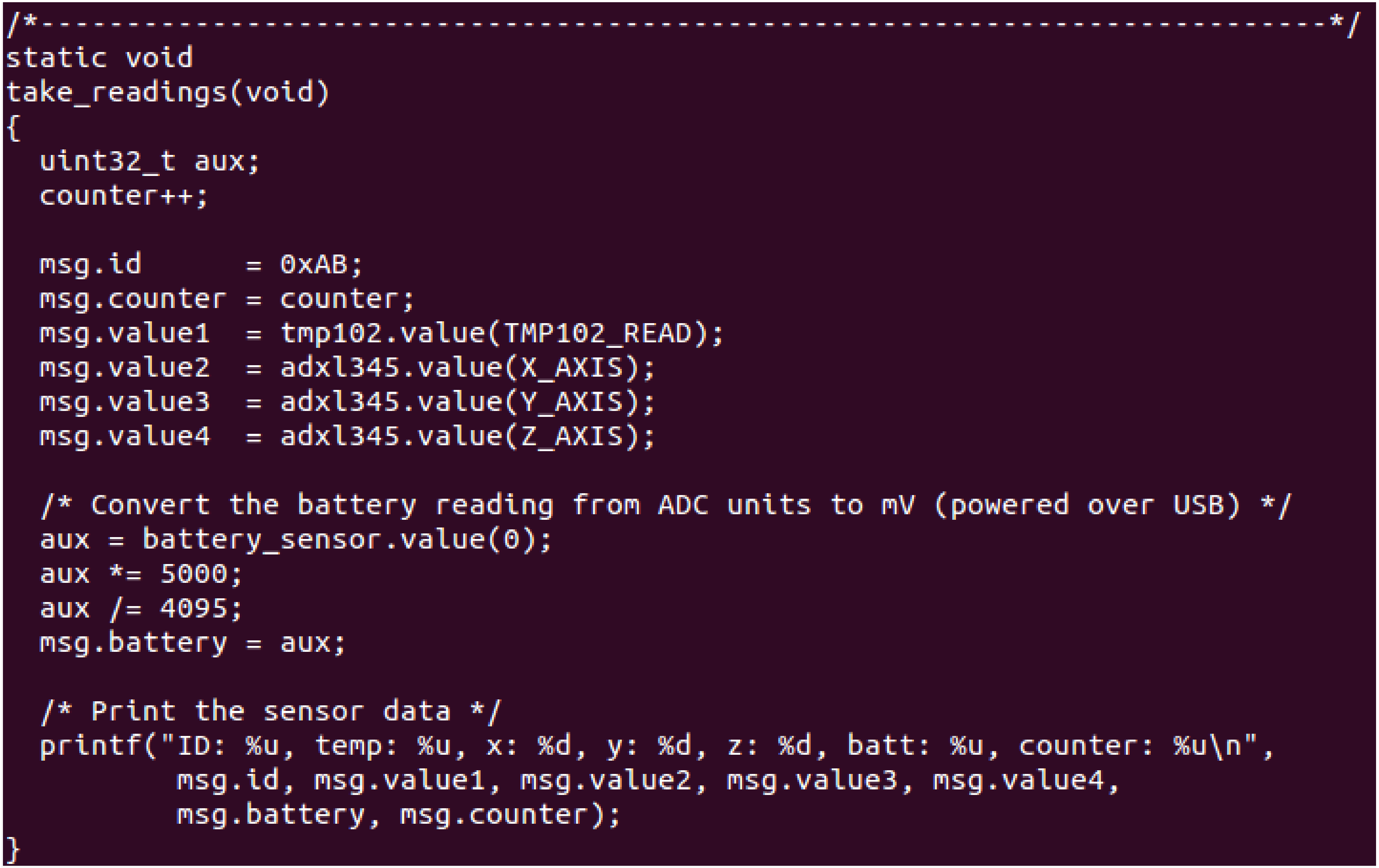
```
/* Create a link-local multicast address to all nodes */
uip_create_linklocal_allnodes_mcast(&addr);

uip_debug_ipaddr_print(&addr);
printf("\n");

/* Take sensor readings and store into the message structure */
take_readings();

/* Send the multicast packet to all devices */
simple_udp_sendto(&mcast_connection, msgPtr, sizeof(msg), &addr);
```

examples/zolertia/tutorial/02-ipv6/01-udp-local-multicast



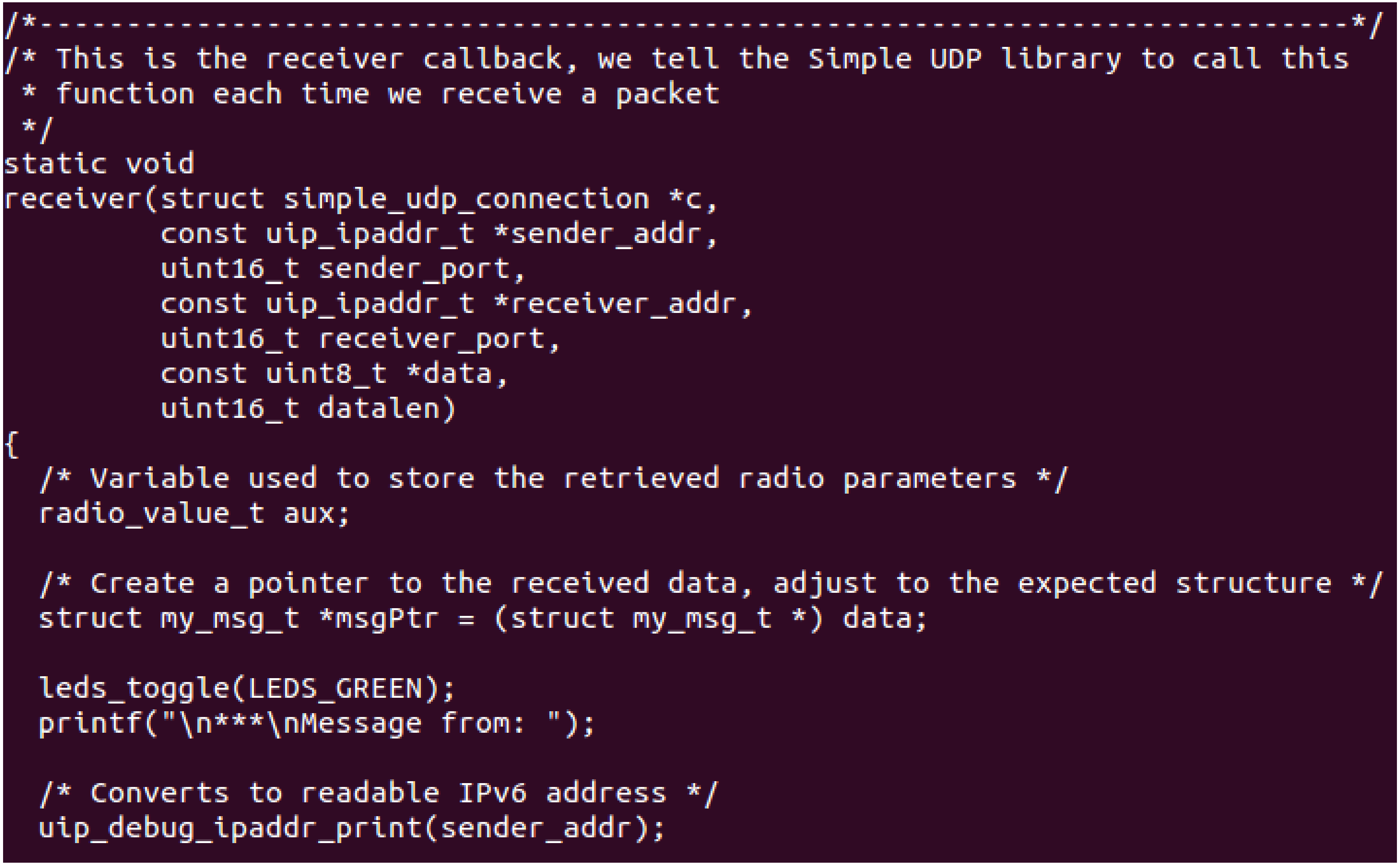
```
/*---------------------------------------------------------------------------*/
/* This is the UDP port used to send and receive data */
#define UDP_CLIENT_PORT   8765
#define UDP_SERVER_PORT   5678

/* Radio values to be configured for the 01-udp-local-multicast example */
#define EXAMPLE_TX_POWER  31
#define EXAMPLE_CHANNEL   15
#define EXAMPLE_PANID     0xBEEF

/*---------------------------------------------------------------------------*/
/* This data structure is used to store the packet content (payload) */
struct my_msg_t {
  uint8_t  id;
  uint16_t counter;
  uint16_t value1;
  uint16_t value2;
  uint16_t value3;
  uint16_t value4;
  uint16_t battery;
};
```

examples/zolertia/tutorial/02-ipv6/example.h



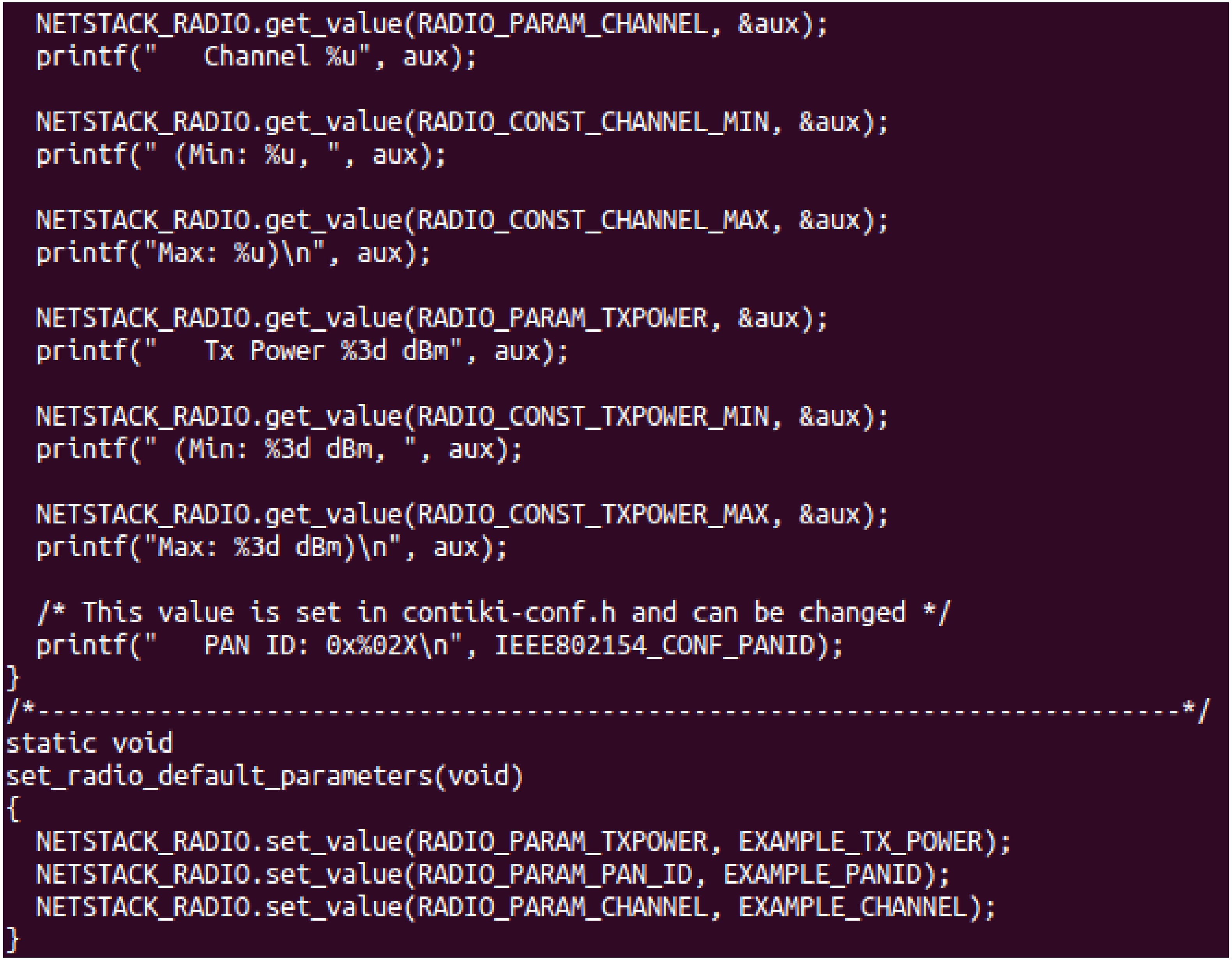
```
/*---------------------------------------------------------------------------*/
static void
take_readings(void)
{
  uint32_t aux;
  counter++;

  msg.id      = 0xAB;
  msg.counter = counter;
  msg.value1  = tmp102.value(TMP102_READ);
  msg.value2  = adxl345.value(X_AXIS);
  msg.value3  = adxl345.value(Y_AXIS);
  msg.value4  = adxl345.value(Z_AXIS);

  /* Convert the battery reading from ADC units to mV (powered over USB) */
  aux = battery_sensor.value(0);
  aux *= 5000;
  aux /= 4095;
  msg.battery = aux;

  /* Print the sensor data */
  printf("ID: %u, temp: %u, x: %d, y: %d, z: %d, batt: %u, counter: %u\n",
          msg.id, msg.value1, msg.value2, msg.value3, msg.value4,
          msg.battery, msg.counter);
}
```

examples/zolertia/tutorial/02-ipv6/01-udp-local-multicast



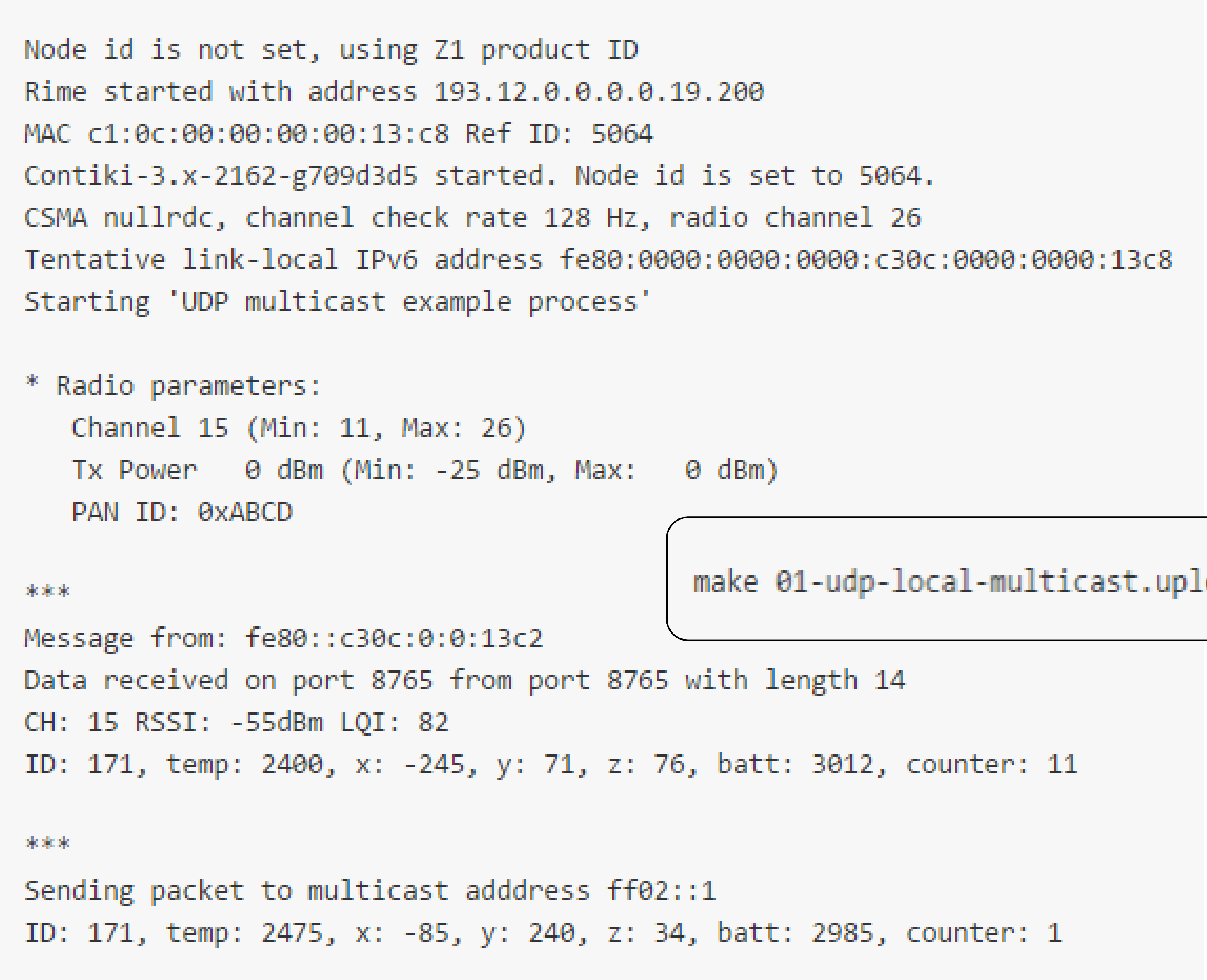
```
/*----------------------------------------------------------------------------*/
/* This is the receiver callback, we tell the Simple UDP library to call this
 * function each time we receive a packet
 */
static void
receiver(struct simple_udp_connection *c,
         const uip_ipaddr_t *sender_addr,
         uint16_t sender_port,
         const uip_ipaddr_t *receiver_addr,
         uint16_t receiver_port,
         const uint8_t *data,
         uint16_t datalen)
{
  /* Variable used to store the retrieved radio parameters */
  radio_value_t aux;

  /* Create a pointer to the received data, adjust to the expected structure */
  struct my_msg_t *msgPtr = (struct my_msg_t *) data;

  leds_toggle(LEDS_GREEN);
  printf("\n***\nMessage from: ");

  /* Converts to readable IPv6 address */
  uip_debug_ipaddr_print(sender_addr);
```

examples/zolertia/tutorial/02-ipv6/01-udp-local-multicast



```
  NETSTACK_RADIO.get_value(RADIO_PARAM_CHANNEL, &aux);
  printf("   Channel %u", aux);

  NETSTACK_RADIO.get_value(RADIO_CONST_CHANNEL_MIN, &aux);
  printf(" (Min: %u, ", aux);

  NETSTACK_RADIO.get_value(RADIO_CONST_CHANNEL_MAX, &aux);
  printf("Max: %u)\n", aux);

  NETSTACK_RADIO.get_value(RADIO_PARAM_TXPOWER, &aux);
  printf("   Tx Power %3d dBm", aux);

  NETSTACK_RADIO.get_value(RADIO_CONST_TXPOWER_MIN, &aux);
  printf(" (Min: %3d dBm, ", aux);

  NETSTACK_RADIO.get_value(RADIO_CONST_TXPOWER_MAX, &aux);
  printf("Max: %3d dBm)\n", aux);

  /* This value is set in contiki-conf.h and can be changed */
  printf("   PAN ID: 0x%02X\n", IEEE802154_CONF_PANID);
}
/*---------------------------------------------------------------------------*/
static void
set_radio_default_parameters(void)
{
  NETSTACK_RADIO.set_value(RADIO_PARAM_TXPOWER, EXAMPLE_TX_POWER);
  NETSTACK_RADIO.set_value(RADIO_PARAM_PAN_ID, EXAMPLE_PANID);
  NETSTACK_RADIO.set_value(RADIO_PARAM_CHANNEL, EXAMPLE_CHANNEL);
}
```

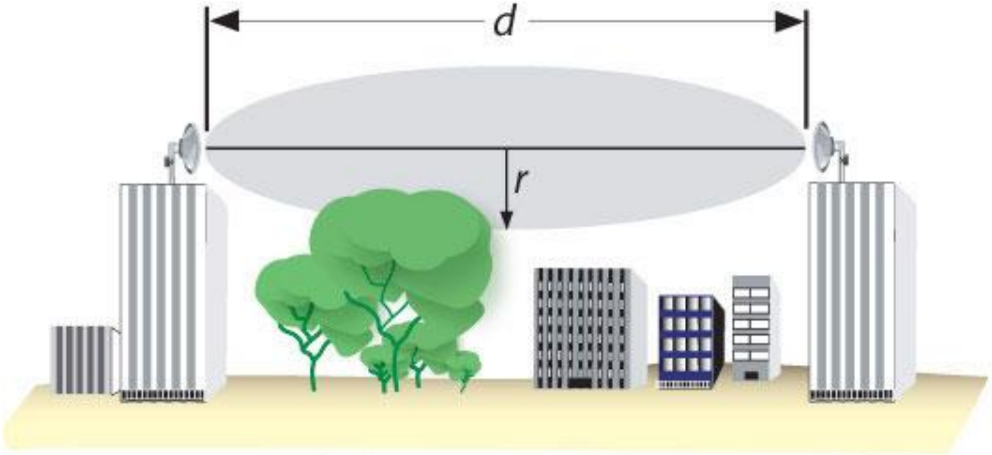
examples/zolertia/tutorial/02-ipv6/01-udp-local-multicast



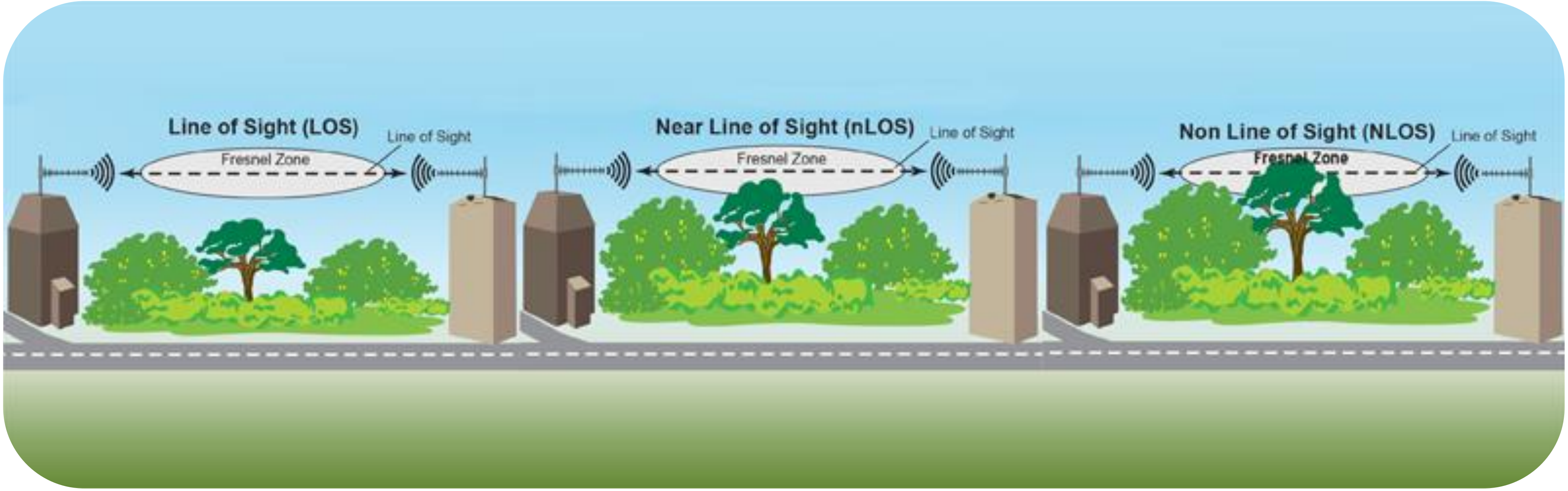
```
Node id is not set, using Z1 product ID
Rime started with address 193.12.0.0.0.0.19.200
MAC c1:0c:00:00:00:00:13:c8 Ref ID: 5064
Contiki-3.x-2162-g709d3d5 started. Node id is set to 5064.
CSMA nullrdc, channel check rate 128 Hz, radio channel 26
Tentative link-local IPv6 address fe80:0000:0000:0000:c30c:0000:0000:13c8
Starting 'UDP multicast example process'

* Radio parameters:
   Channel 15 (Min: 11, Max: 26)
   Tx Power   0 dBm (Min: -25 dBm, Max:   0 dBm)
   PAN ID: 0xABCD

***
Message from: fe80::c30c:0:0:13c2
Data received on port 8765 from port 8765 with length 14
CH: 15 RSSI: -55dBm LQI: 82
ID: 171, temp: 2400, x: -245, y: 71, z: 76, batt: 3012, counter: 11

***
Sending packet to multicast adddress ff02::1
ID: 171, temp: 2475, x: -85, y: 240, z: 34, batt: 2985, counter: 1
```

```
make 01-udp-local-multicast.upload && make login
```

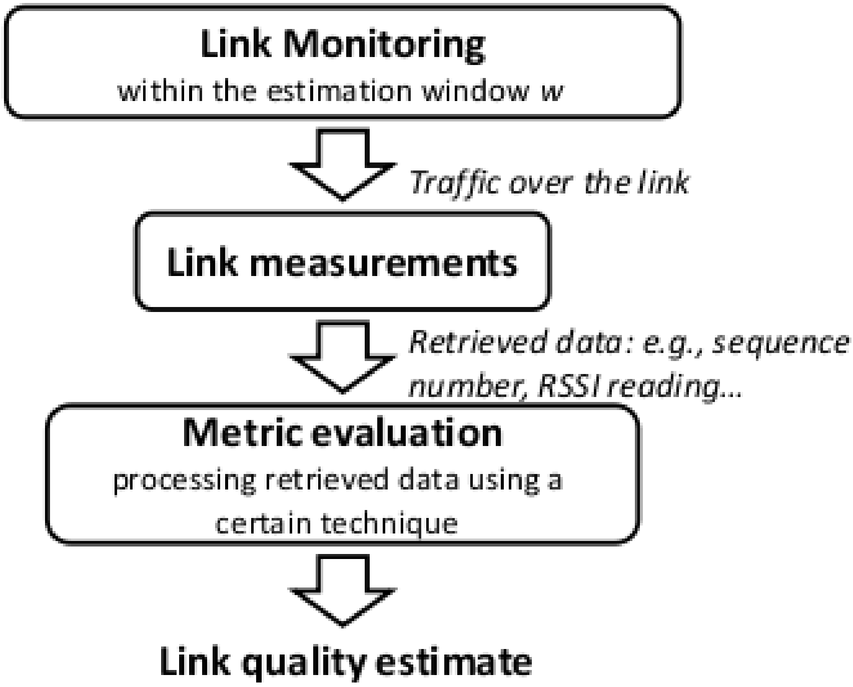
examples/zolertia/tutorial/02-ipv6/01-udp-local-multicast



$$r_{(\text{in mts})} = 17.32 \times \sqrt{\frac{d_{(\text{in Km})}}{4f_{(\text{in GHz})}}} \qquad r_{(\text{in ft})} = 72.05 \times \sqrt{\frac{d_{(\text{in miles})}}{4f_{(\text{in GHz})}}}$$



examples/zolertia/tutorial/02-ipv6/01-udp-local-multicast



Link Monitoring
within the estimation window *w*

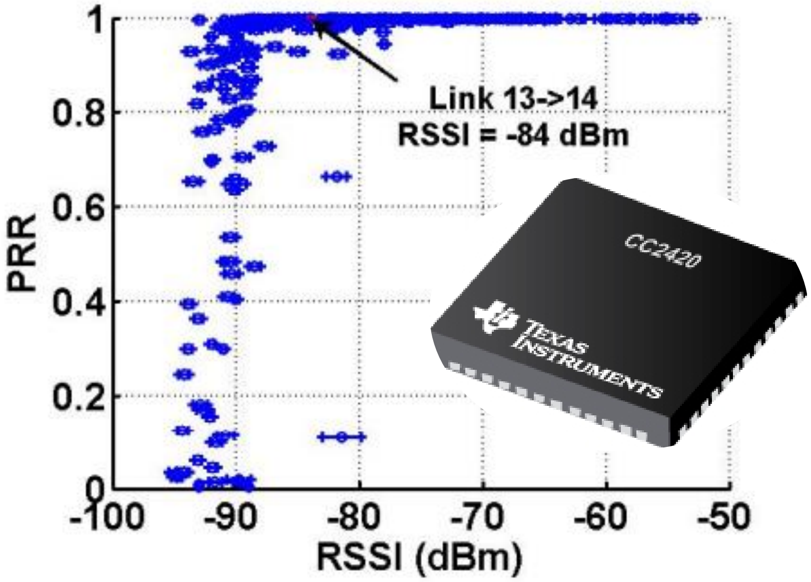
Traffic over the link

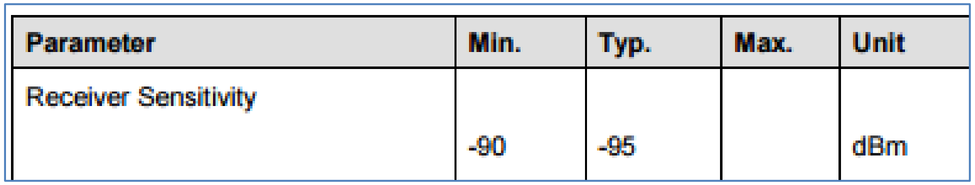
Link measurements

Retrieved data: e.g., sequence number, RSSI reading...

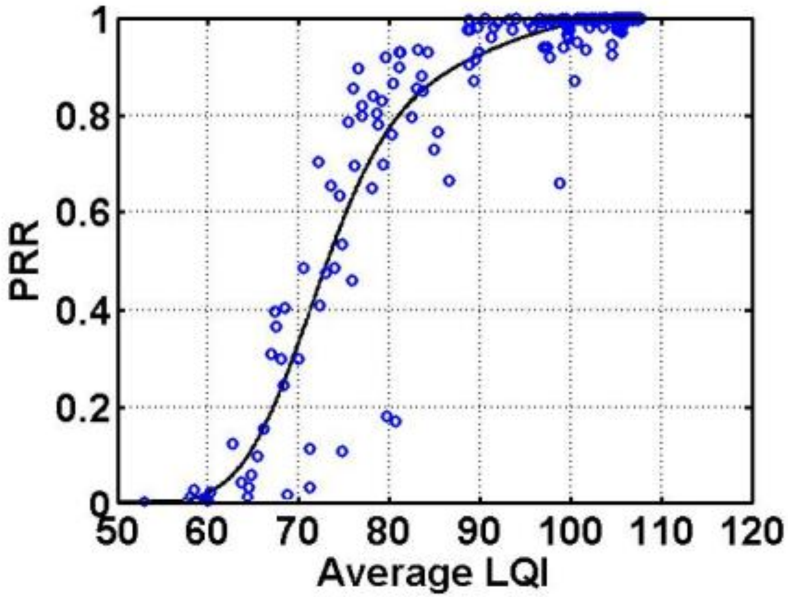
Metric evaluation
processing retrieved data using a certain technique

Link quality estimate





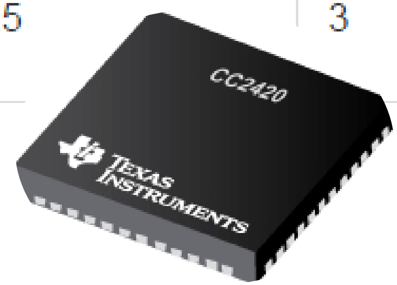
Parameter	Min.	Typ.	Max.	Unit
Receiver Sensitivity	-90	-95		dBm

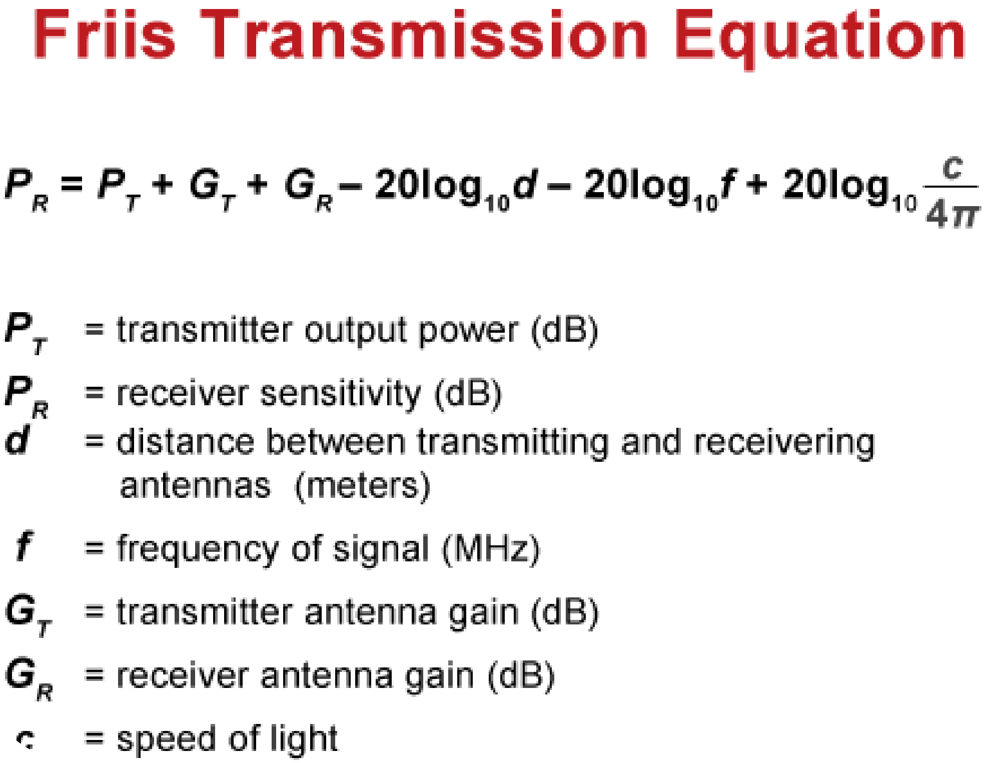


RSSI close to -95dBm is the limit
LQI 97-104 indicates a good demodulation

examples/zolertia/tutorial/02-ipv6/01-udp-local-multicast

TX Power (dBm)	Value	mA
0	31	17.4
-1	27	16.5
-3	23	15.2
-5	19	13.9
-7	15	12.5
-10	11	11.2
-15	7	9.9
-25	3	8.5





Friis Transmission Equation

$$P_R = P_T + G_T + G_R - 20\log_{10} d - 20\log_{10} f + 20\log_{10} \frac{c}{4\pi}$$

P_T = transmitter output power (dB)

P_R = receiver sensitivity (dB)

d = distance between transmitting and receivering antennas (meters)

f = frequency of signal (MHz)

G_T = transmitter antenna gain (dB)

G_R = receiver antenna gain (dB)

c = speed of light

examples/zolertia/tutorial/02-ipv6/01-udp-local-multicast

How to improve our range:

- Increase the transmission power
- Use antennas with higher gain
- Increase antenna’s height
- Use directive antennas

Thumb-rule: every 6dB we double the range

Antena 2.4GHz 5dBi “whip”

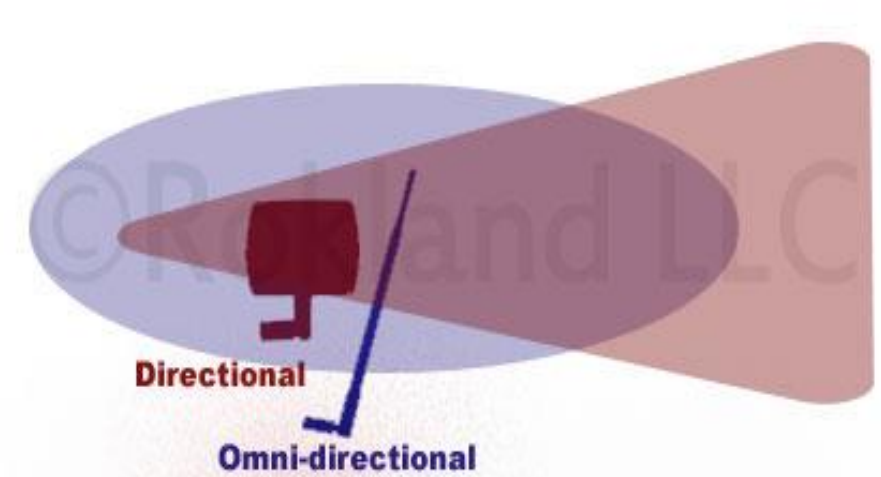
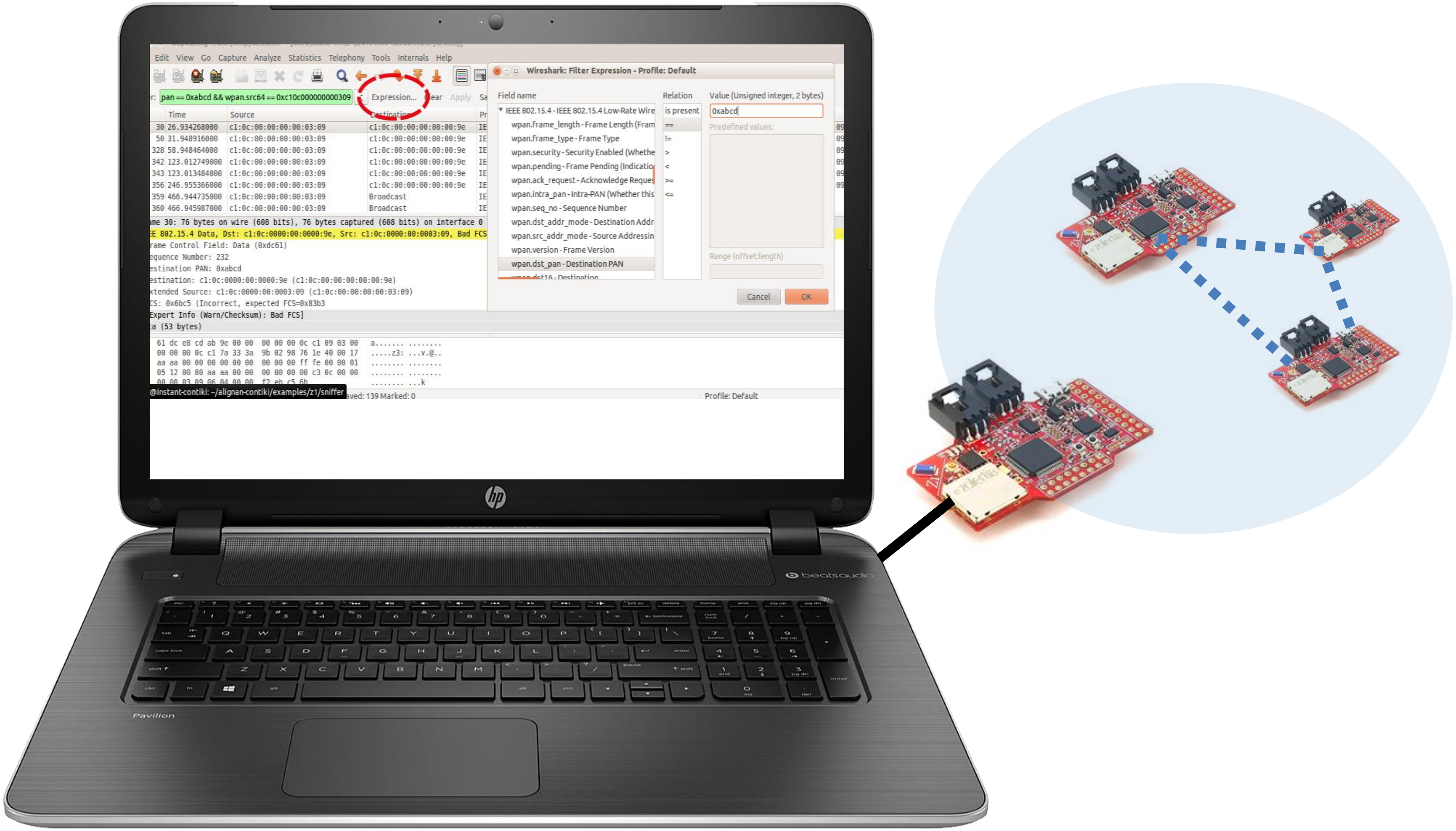


Image from Rokland LLC

examples/zolertia/tutorial/02-ipv6/01-udp-local-multicast


Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression...
lear
Apply
Sa
Wireshark: Filter Expression - Profile: Default
Cancel
OK
hp
beatsaudio
Pavilion

```
r: pan == 0xabcd && wpan.src64 == 0xc10c000000000309
```

```
     Time            Source                       Destination                  Pr
 30  26.934268000   c1:0c:00:00:00:00:03:09      c1:0c:00:00:00:00:00:9e      IE
 50  31.948916000   c1:0c:00:00:00:00:03:09      c1:0c:00:00:00:00:00:9e      IE
328  58.948464000   c1:0c:00:00:00:00:03:09      c1:0c:00:00:00:00:00:9e      IE
342 123.012749000   c1:0c:00:00:00:00:03:09      c1:0c:00:00:00:00:00:9e      IE
343 123.013484000   c1:0c:00:00:00:00:03:09      c1:0c:00:00:00:00:00:9e      IE
356 246.955366000   c1:0c:00:00:00:00:03:09      c1:0c:00:00:00:00:00:9e      IE
359 466.944735000   c1:0c:00:00:00:00:03:09      Broadcast                    IE
360 466.945987000   c1:0c:00:00:00:00:03:09      Broadcast                    IE
```

```
ame 30: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface 0
E 802.15.4 Data, Dst: c1:0c:0000:00:0000:9e, Src: c1:0c:0000:00:0003:09, Bad FCS
rame Control Field: Data (0xdc61)
equence Number: 232
estination PAN: 0xabcd
estination: c1:0c:0000:00:0000:9e (c1:0c:00:00:00:00:00:9e)
xtended Source: c1:0c:0000:00:0003:09 (c1:0c:00:00:00:00:03:09)
CS: 0x6bc5 (Incorrect, expected FCS=0x83b3
Expert Info (Warn/Checksum): Bad FCS]
ta (53 bytes)
```

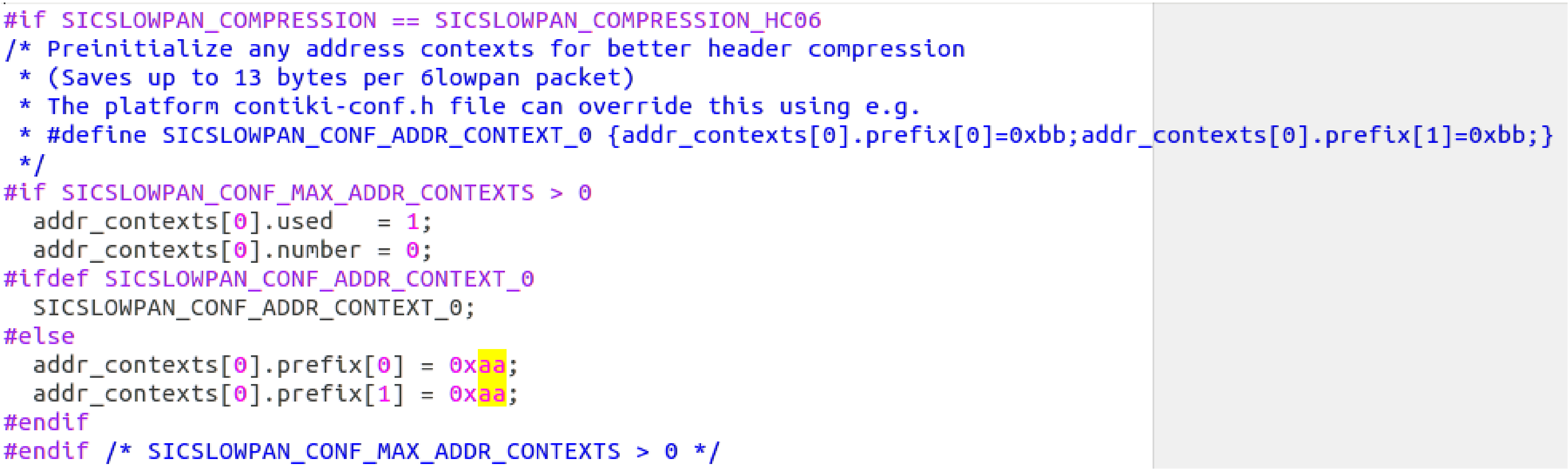
```
61 dc e8 cd ab 9e 00 00  00 00 00 0c c1 09 03 00   a....... ........
00 00 00 0c c1 7a 33 3a  9b 02 98 76 1e 40 00 17   .....z3: ...v.@..
aa aa 00 00 00 00 00 00  00 00 00 ff fe 00 00 01   ........ ........
05 12 00 80 aa aa 00 00  00 00 00 00 c3 0c 00 00   ........ ........
00 00 03 09 06 04 00 00  f2 eb c5 6b               ........ ...k
```

```
@instant-contiki: ~/alignan-contiki/examples/z1/sniffer
aved: 139 Marked: 0
Profile: Default
```

Field name | Relation | Value (Unsigned integer, 2 bytes)

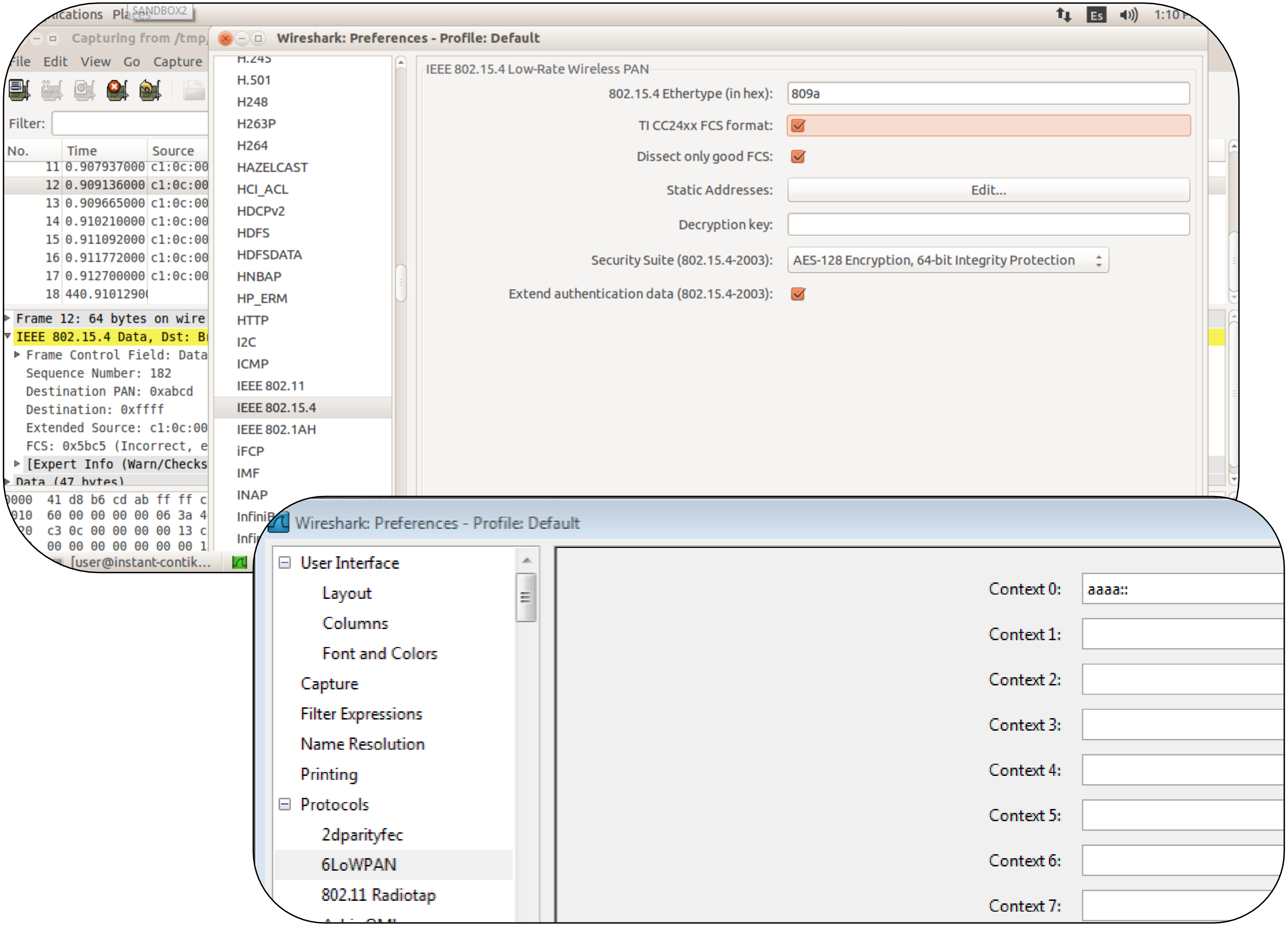
```
Field name                                          Relation     Value (Unsigned integer, 2 bytes)
▼ IEEE 802.15.4 - IEEE 802.15.4 Low-Rate Wire       is present   0xabcd
   wpan.frame_length - Frame Length (Fram           ==           Predefined values:
   wpan.frame_type - Frame Type                     !=
   wpan.security - Security Enabled (Whethe         >
   wpan.pending - Frame Pending (Indicatio          <
   wpan.ack_request - Acknowledge Reques            >=
   wpan.intra_pan - Intra-PAN (Whether this         <=
   wpan.seq_no - Sequence Number
   wpan.dst_addr_mode - Destination Addr
   wpan.src_addr_mode - Source Addressin
   wpan.version - Frame Version
   wpan.dst_pan - Destination PAN                                Range (offset:length)
   wpan.dst16 - Destination
```

examples/zolertia/tutorial/02-ipv6/03-sniffer + wireshark



```
#if SICSLOWPAN_COMPRESSION == SICSLOWPAN_COMPRESSION_HC06
/* Preinitialize any address contexts for better header compression
 * (Saves up to 13 bytes per 6lowpan packet)
 * The platform contiki-conf.h file can override this using e.g.
 * #define SICSLOWPAN_CONF_ADDR_CONTEXT_0 {addr_contexts[0].prefix[0]=0xbb;addr_contexts[0].prefix[1]=0xbb;}
 */
#if SICSLOWPAN_CONF_MAX_ADDR_CONTEXTS > 0
  addr_contexts[0].used   = 1;
  addr_contexts[0].number = 0;
#ifdef SICSLOWPAN_CONF_ADDR_CONTEXT_0
  SICSLOWPAN_CONF_ADDR_CONTEXT_0;
#else
  addr_contexts[0].prefix[0] = 0xaa;
  addr_contexts[0].prefix[1] = 0xaa;
#endif
#endif /* SICSLOWPAN_CONF_MAX_ADDR_CONTEXTS > 0 */
```

core/net/ipv6/sicslowpan.c



```
 11 0.907937000 c1:0c:00
 12 0.909136000 c1:0c:00
 13 0.909665000 c1:0c:00
 14 0.910210000 c1:0c:00
 15 0.911092000 c1:0c:00
 16 0.911772000 c1:0c:00
 17 0.912700000 c1:0c:00
 18 440.9101290(
Frame 12: 64 bytes on wire
IEEE 802.15.4 Data, Dst: Br
  Frame Control Field: Data
  Sequence Number: 182
  Destination PAN: 0xabcd
  Destination: 0xffff
  Extended Source: c1:0c:00
  FCS: 0x5bc5 (Incorrect, e
  [Expert Info (Warn/Checks
Data (47 bytes)
0000  41 d8 b6 cd ab ff ff c
0010  60 00 00 00 00 06 3a 4
0020  c3 0c 00 00 00 00 13 c
      00 00 00 00 00 00 00 1
```

IEEE 802.15.4 Low-Rate Wireless PAN

802.15.4 Ethertype (in hex): 809a

TI CC24xx FCS format: ☑

Dissect only good FCS: ☑

Static Addresses: Edit...

Decryption key:

Security Suite (802.15.4-2003): AES-128 Encryption, 64-bit Integrity Protection

Extend authentication data (802.15.4-2003): ☑

Wireshark: Preferences - Profile: Default

Context 0: aaaa::

Context 1:

Context 2:

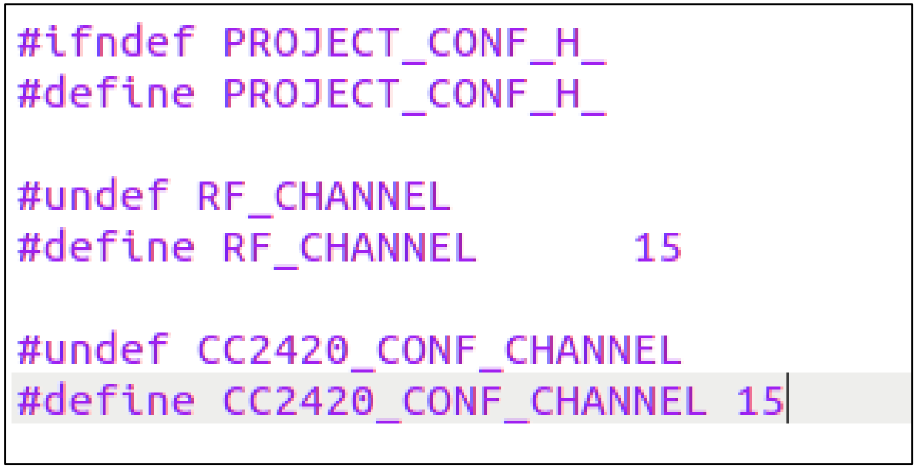
Context 3:

Context 4:

Context 5:

Context 6:

Context 7:



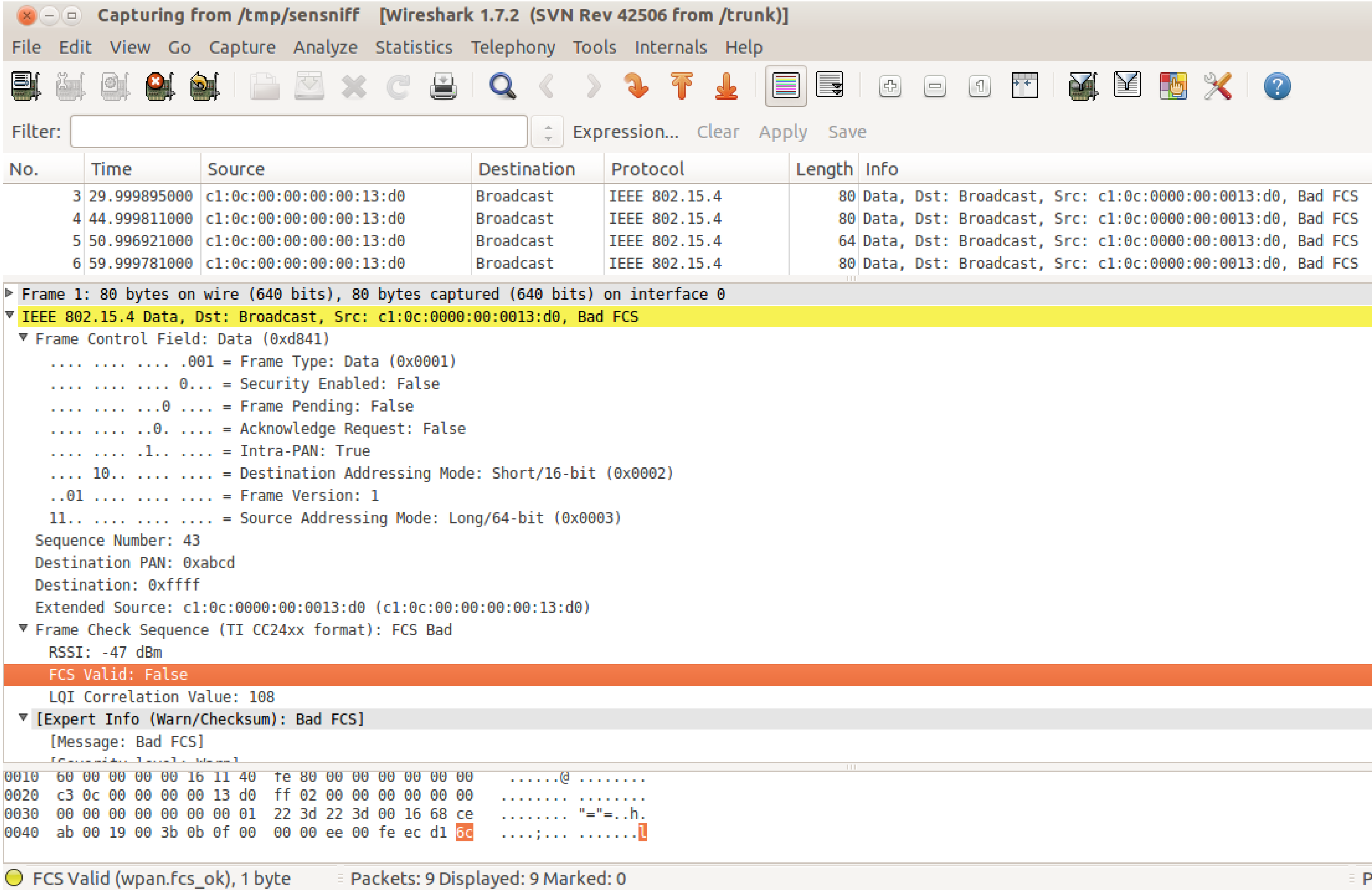
```
#ifndef PROJECT_CONF_H_
#define PROJECT_CONF_H_

#undef RF_CHANNEL
#define RF_CHANNEL      15

#undef CC2420_CONF_CHANNEL
#define CC2420_CONF_CHANNEL 15
```

In a terminal

```
make sniffer.upload
python sensniff.py --non-interactive -d /dev/ttyUSB0 -b 115200
```

In another terminal

```
sudo wireshark -i /tmp/sensniff
```

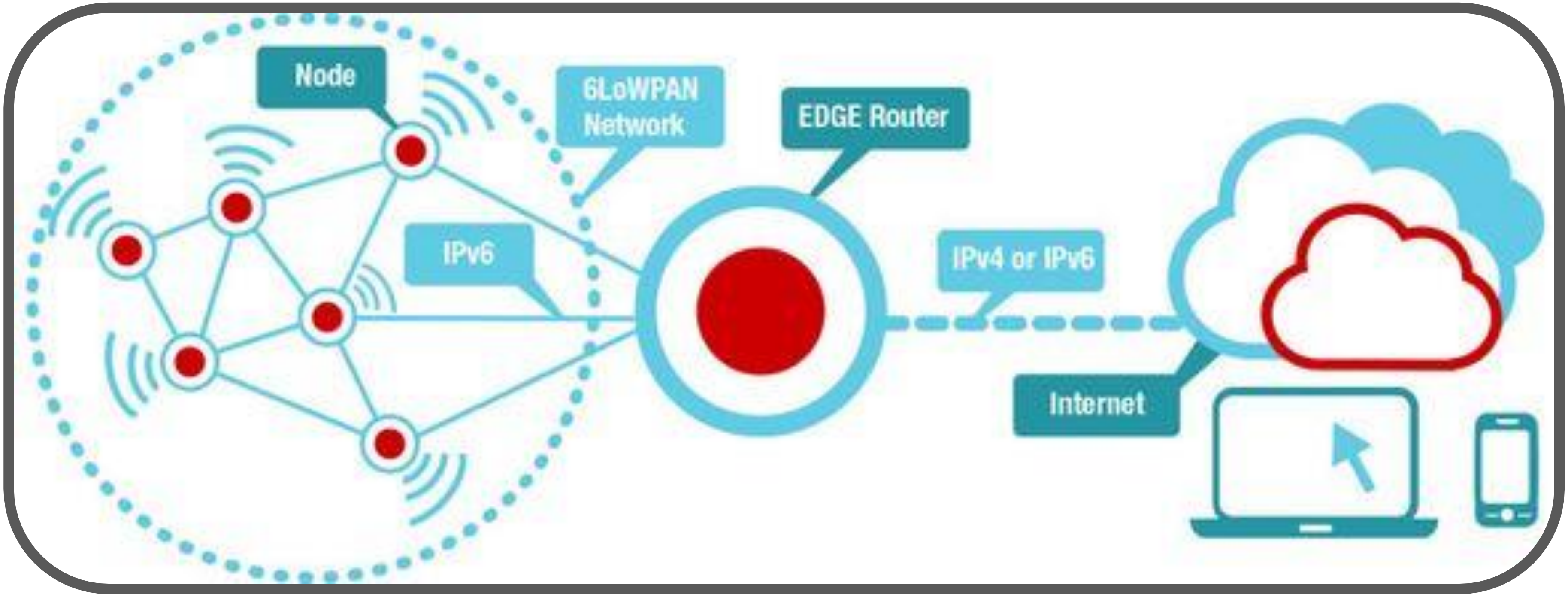


```
No.  Time          Source                   Destination  Protocol       Length Info
   3 29.999895000  c1:0c:00:00:00:00:13:d0  Broadcast    IEEE 802.15.4      80 Data, Dst: Broadcast, Src: c1:0c:0000:00:0013:d0, Bad FCS
   4 44.999811000  c1:0c:00:00:00:00:13:d0  Broadcast    IEEE 802.15.4      80 Data, Dst: Broadcast, Src: c1:0c:0000:00:0013:d0, Bad FCS
   5 50.996921000  c1:0c:00:00:00:00:13:d0  Broadcast    IEEE 802.15.4      64 Data, Dst: Broadcast, Src: c1:0c:0000:00:0013:d0, Bad FCS
   6 59.999781000  c1:0c:00:00:00:00:13:d0  Broadcast    IEEE 802.15.4      80 Data, Dst: Broadcast, Src: c1:0c:0000:00:0013:d0, Bad FCS
```

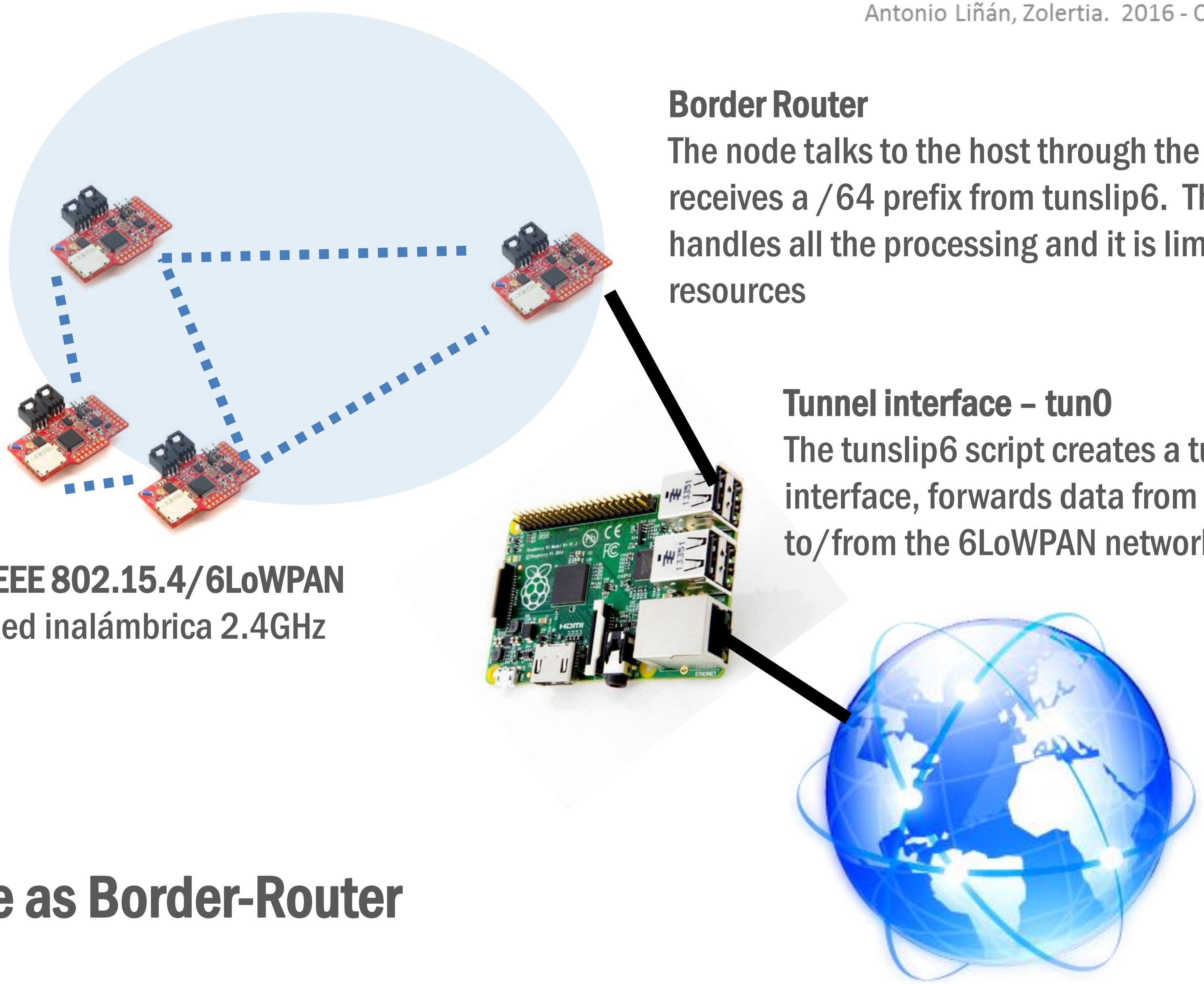
```
▶ Frame 1: 80 bytes on wire (640 bits), 80 bytes captured (640 bits) on interface 0
▼ IEEE 802.15.4 Data, Dst: Broadcast, Src: c1:0c:0000:00:0013:d0, Bad FCS
  ▼ Frame Control Field: Data (0xd841)
      .... .... .... .001 = Frame Type: Data (0x0001)
      .... .... .... 0... = Security Enabled: False
      .... .... ...0 .... = Frame Pending: False
      .... .... ..0. .... = Acknowledge Request: False
      .... .... .1.. .... = Intra-PAN: True
      .... 10.. .... .... = Destination Addressing Mode: Short/16-bit (0x0002)
      ..01 .... .... .... = Frame Version: 1
      11.. .... .... .... = Source Addressing Mode: Long/64-bit (0x0003)
    Sequence Number: 43
    Destination PAN: 0xabcd
    Destination: 0xffff
    Extended Source: c1:0c:0000:00:0013:d0 (c1:0c:00:00:00:00:13:d0)
  ▼ Frame Check Sequence (TI CC24xx format): FCS Bad
      RSSI: -47 dBm
      FCS Valid: False
      LQI Correlation Value: 108
  ▼ [Expert Info (Warn/Checksum): Bad FCS]
      [Message: Bad FCS]
      [Severity level: Warn]
```

```
0010  60 00 00 00 00 16 11 40  fe 80 00 00 00 00 00 00   ......@ ........
0020  c3 0c 00 00 00 00 13 d0  ff 02 00 00 00 00 00 00  ........ ........
0030  00 00 00 00 00 00 00 01  22 3d 22 3d 00 16 68 ce  ........ "="=..h.
0040  ab 00 19 00 3b 0b 0f 00  00 00 ee 00 fe ec d1 6c  ....;... .......l
```

FCS Valid (wpan.fcs_ok), 1 byte   Packets: 9 Displayed: 9 Marked: 0


Node
6LoWPAN Network
EDGE Router
IPv6
IPv4 or IPv6
Internet

A node as Border-Router



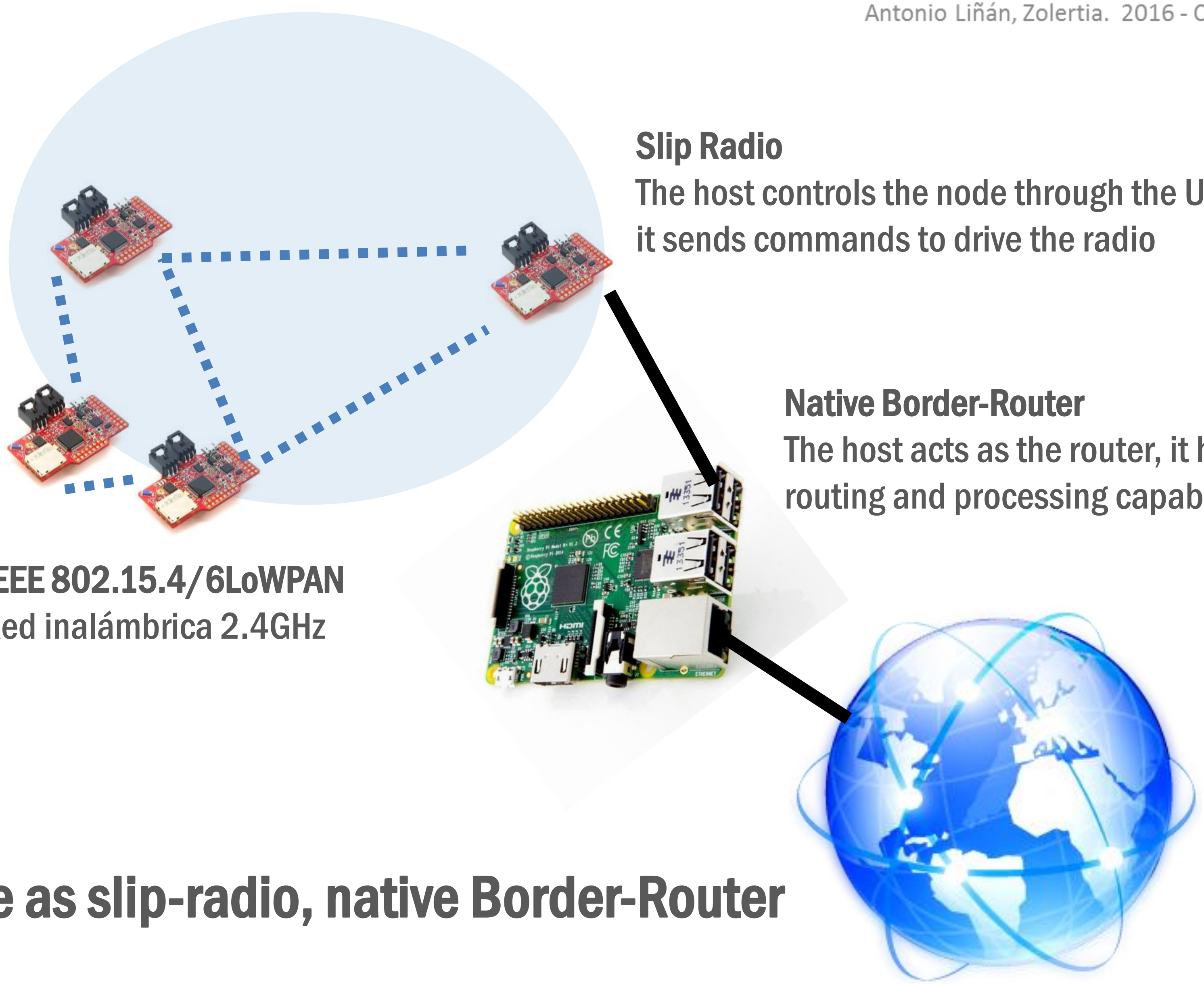
Border Router
The node talks to the host through the USB, it receives a /64 prefix from tunslip6. The node handles all the processing and it is limited by its resources

Tunnel interface – tun0
The tunslip6 script creates a tunnel interface, forwards data from IPv6 to/from the 6LoWPAN network

IEEE 802.15.4/6LoWPAN
Red inalámbrica 2.4GHz

examples/zolertia/tutorial/02-ipv6/02-border-router

A node as slip-radio, native Border-Router

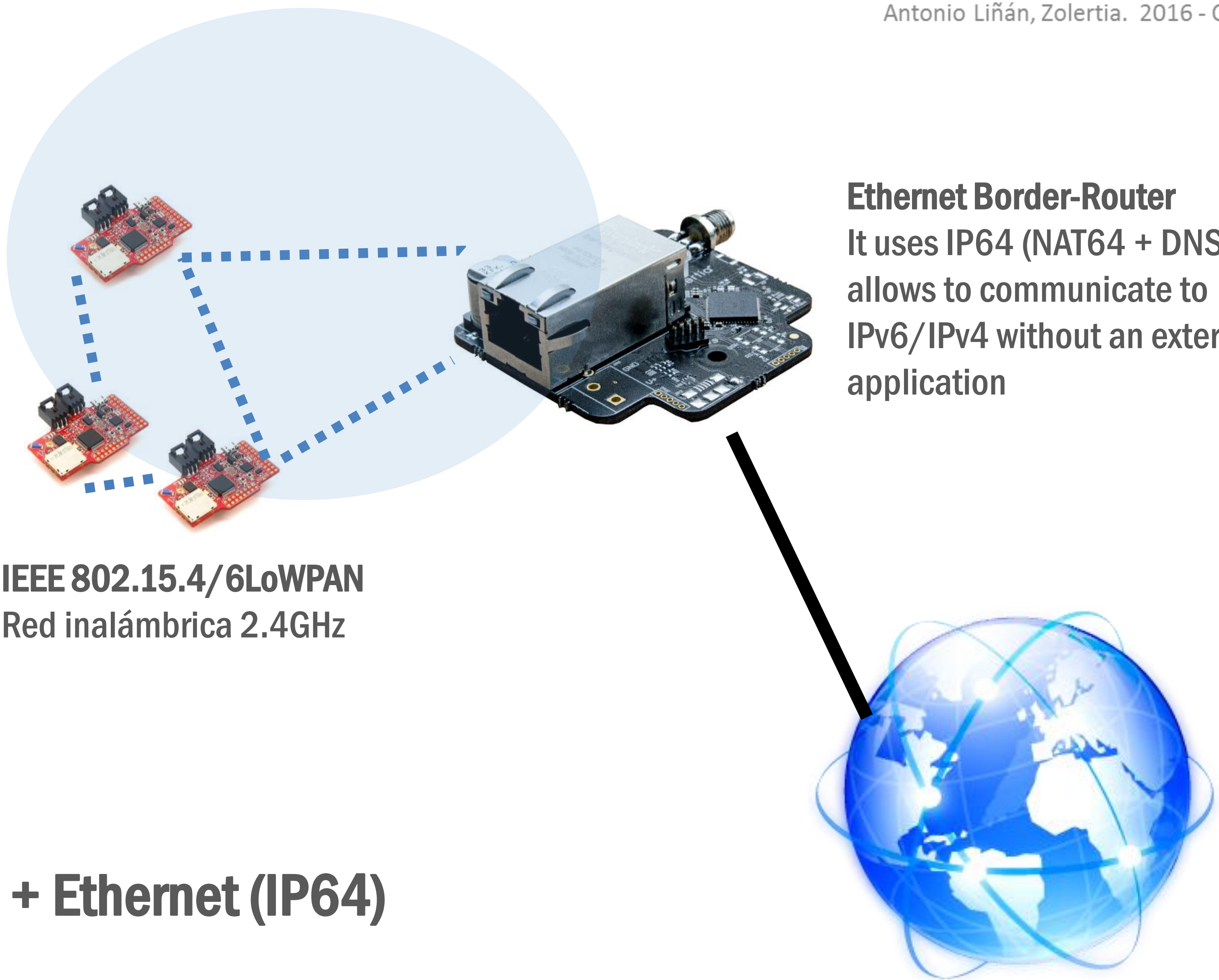


Slip Radio
The host controls the node through the USB,
it sends commands to drive the radio

Native Border-Router
The host acts as the router, it has more
routing and processing capabilities

IEEE 802.15.4/6LoWPAN
Red inalámbrica 2.4GHz

examples/zolertia/tutorial/02-ipv6/02-border-router

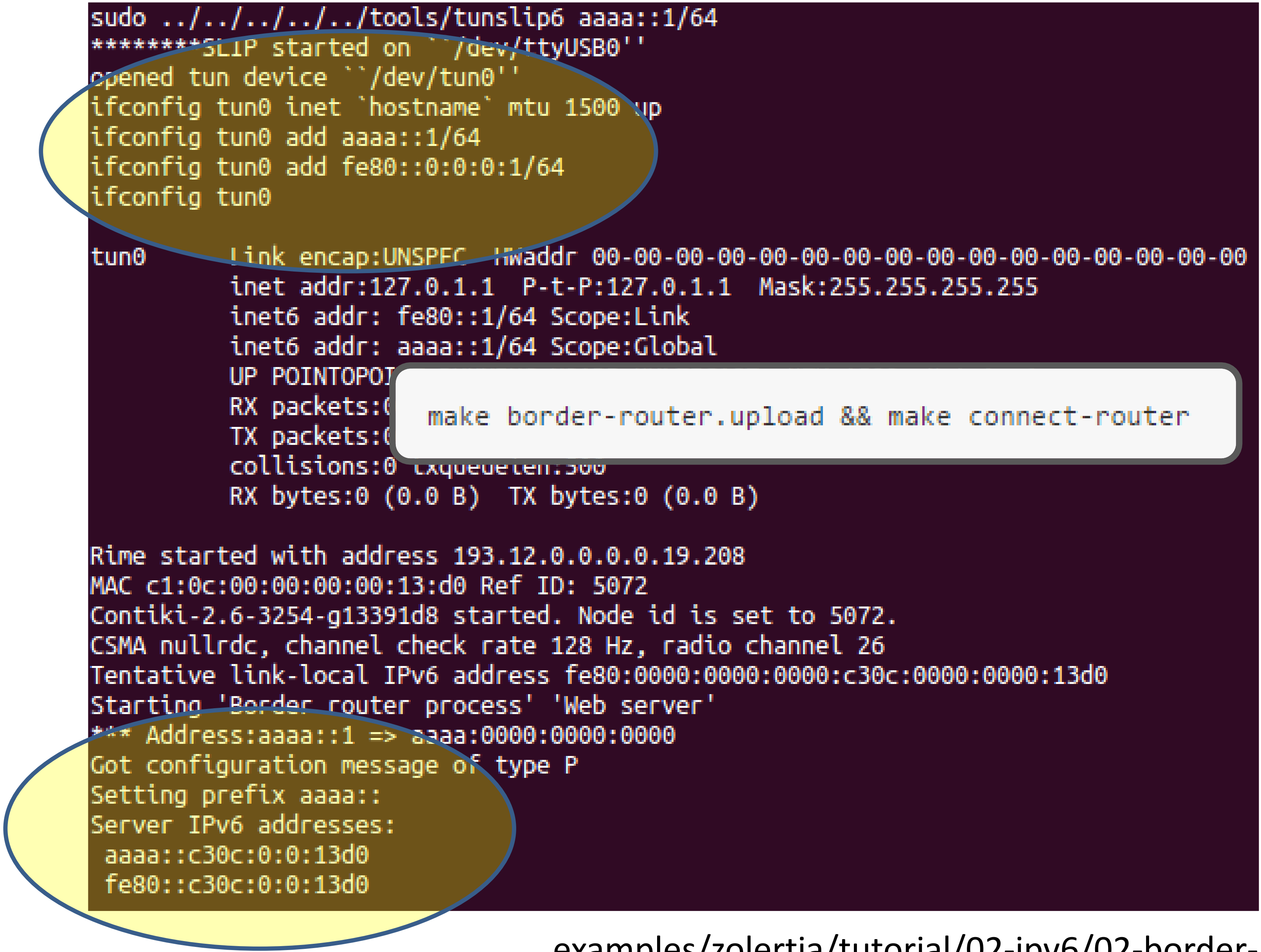


Ethernet Border-Router
It uses IP64 (NAT64 + DNS64), allows to communicate to IPv6/IPv4 without an external application

IEEE 802.15.4/6LoWPAN
Red inalámbrica 2.4GHz

Node + Ethernet (IP64)

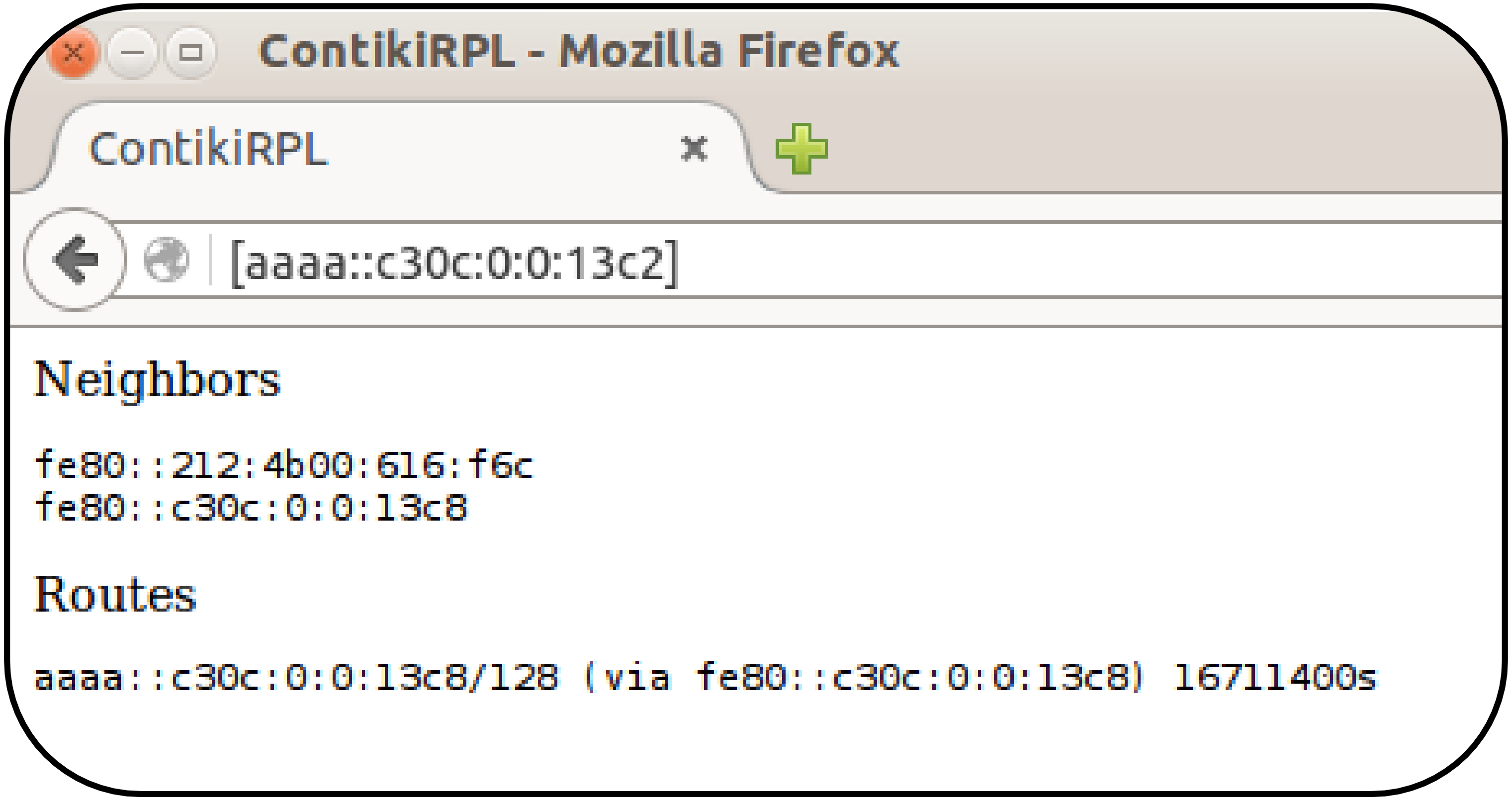
examples/zolertia/tutorial/02-ipv6/02-border-router



```
sudo ../../../../../tools/tunslip6 aaaa::1/64
********SLIP started on ``/dev/ttyUSB0''
opened tun device ``/dev/tun0''
ifconfig tun0 inet `hostname` mtu 1500 up
ifconfig tun0 add aaaa::1/64
ifconfig tun0 add fe80::0:0:0:1/64
ifconfig tun0

tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:127.0.1.1  P-t-P:127.0.1.1  Mask:255.255.255.255
          inet6 addr: fe80::1/64 Scope:Link
          inet6 addr: aaaa::1/64 Scope:Global
          UP POINTOPOI
          RX packets:(
          TX packets:(
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

Rime started with address 193.12.0.0.0.0.19.208
MAC c1:0c:00:00:00:00:13:d0 Ref ID: 5072
Contiki-2.6-3254-g13391d8 started. Node id is set to 5072.
CSMA nullrdc, channel check rate 128 Hz, radio channel 26
Tentative link-local IPv6 address fe80:0000:0000:0000:c30c:0000:0000:13d0
Starting 'Border router process' 'Web server'
*** Address:aaaa::1 => aaaa:0000:0000:0000
Got configuration message of type P
Setting prefix aaaa::
Server IPv6 addresses:
 aaaa::c30c:0:0:13d0
 fe80::c30c:0:0:13d0
```

```
make border-router.upload && make connect-router
```

examples/zolertia/tutorial/02-ipv6/02-border-router



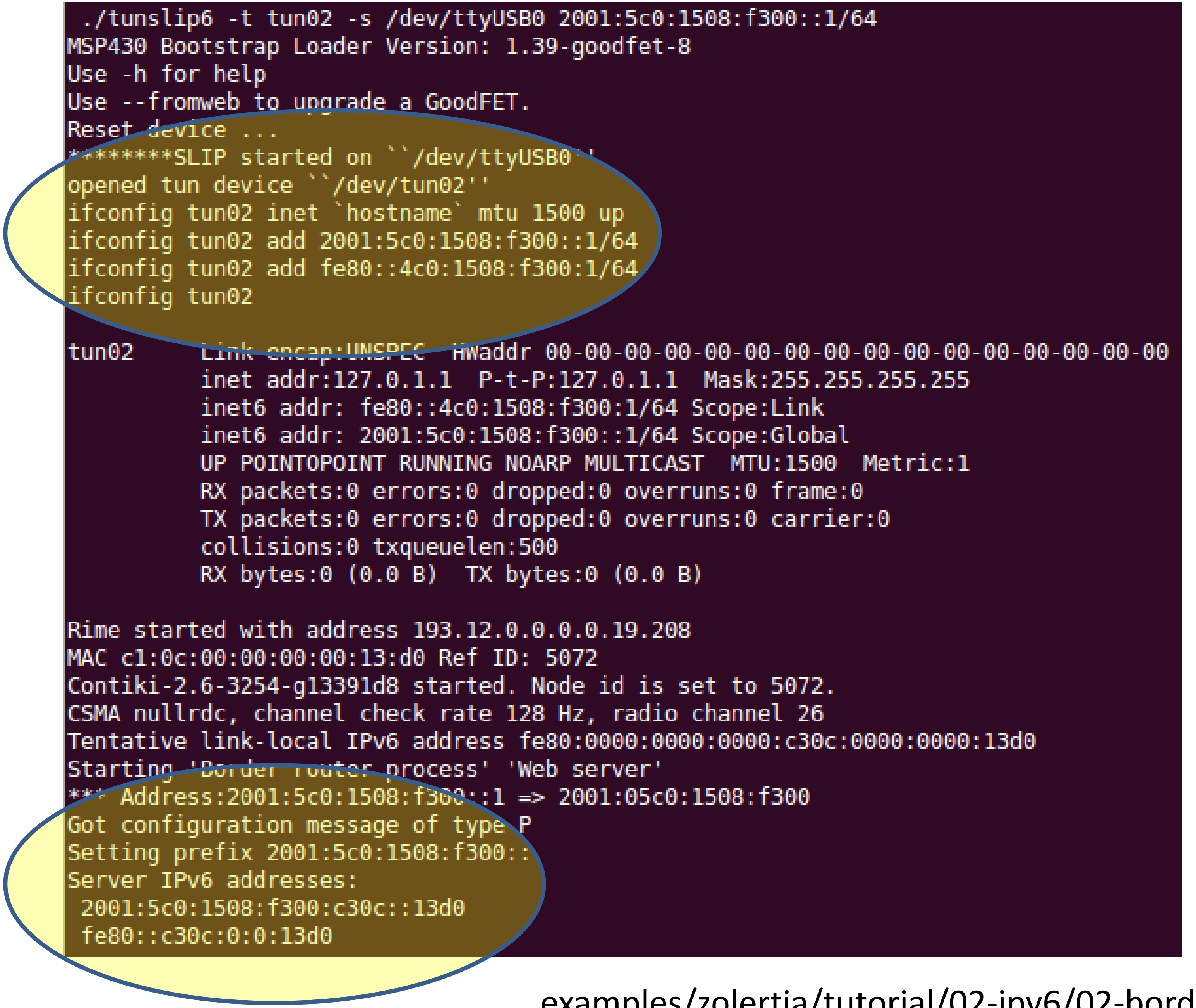
Neighbors

```
fe80::212:4b00:616:f6c
fe80::c30c:0:0:13c8
```

Routes

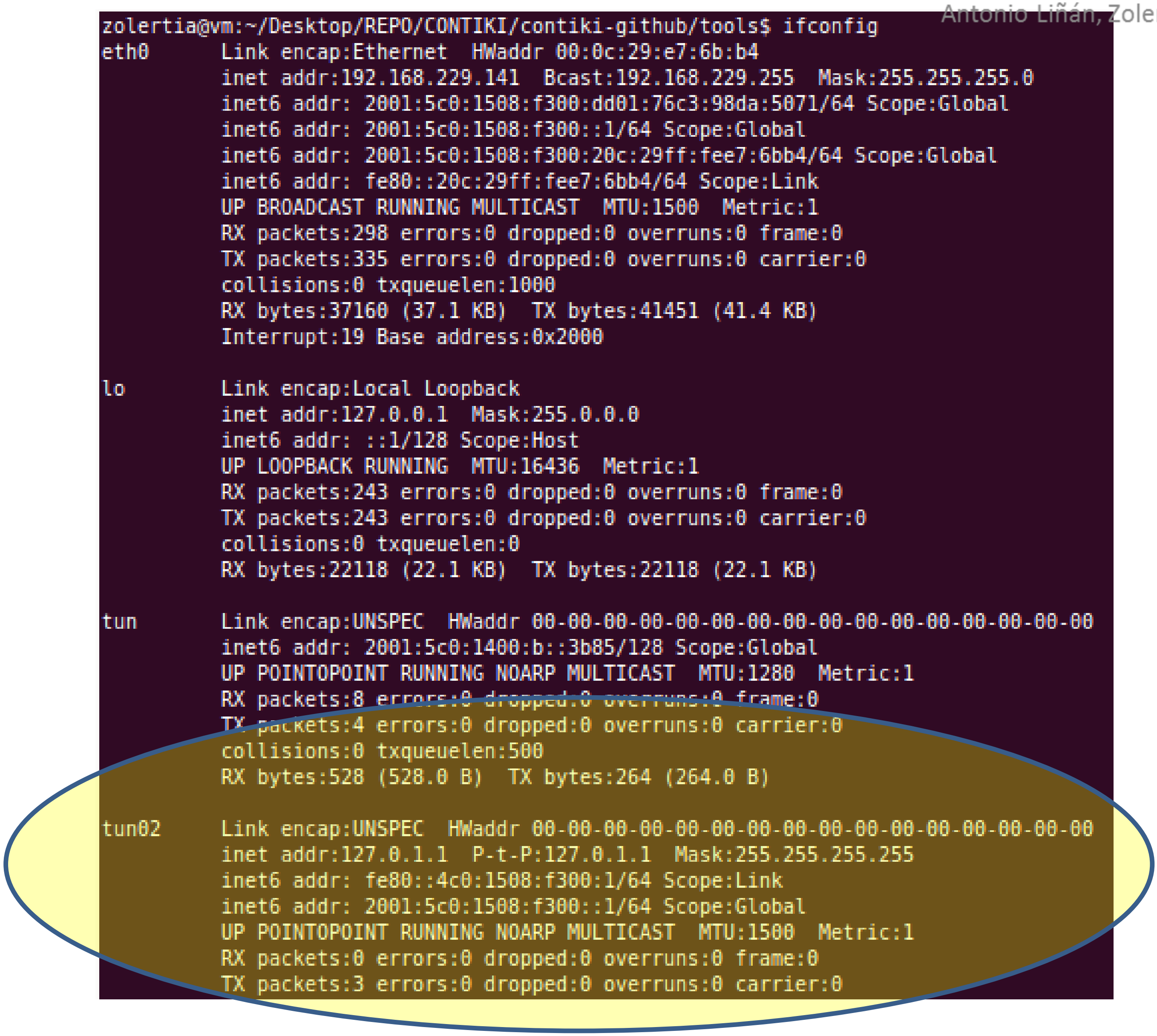
```
aaaa::c30c:0:0:13c8/128 (via fe80::c30c:0:0:13c8) 16711400s
```

examples/zolertia/tutorial/02-ipv6/02-border-router



```
 ./tunslip6 -t tun02 -s /dev/ttyUSB0 2001:5c0:1508:f300::1/64
MSP430 Bootstrap Loader Version: 1.39-goodfet-8
Use -h for help
Use --fromweb to upgrade a GoodFET.
Reset device ...
********SLIP started on ``/dev/ttyUSB0''
opened tun device ``/dev/tun02''
ifconfig tun02 inet `hostname` mtu 1500 up
ifconfig tun02 add 2001:5c0:1508:f300::1/64
ifconfig tun02 add fe80::4c0:1508:f300:1/64
ifconfig tun02

tun02     Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:127.0.1.1  P-t-P:127.0.1.1  Mask:255.255.255.255
          inet6 addr: fe80::4c0:1508:f300:1/64 Scope:Link
          inet6 addr: 2001:5c0:1508:f300::1/64 Scope:Global
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

Rime started with address 193.12.0.0.0.0.19.208
MAC c1:0c:00:00:00:00:13:d0 Ref ID: 5072
Contiki-2.6-3254-g13391d8 started. Node id is set to 5072.
CSMA nullrdc, channel check rate 128 Hz, radio channel 26
Tentative link-local IPv6 address fe80:0000:0000:0000:c30c:0000:0000:13d0
Starting 'Border router process' 'Web server'
*** Address:2001:5c0:1508:f300::1 => 2001:05c0:1508:f300
Got configuration message of type P
Setting prefix 2001:5c0:1508:f300::
Server IPv6 addresses:
 2001:5c0:1508:f300:c30c::13d0
 fe80::c30c:0:0:13d0
```

examples/zolertia/tutorial/02-ipv6/02-border-router



```
zolertia@vm:~/Desktop/REPO/CONTIKI/contiki-github/tools$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:e7:6b:b4
          inet addr:192.168.229.141  Bcast:192.168.229.255  Mask:255.255.255.0
          inet6 addr: 2001:5c0:1508:f300:dd01:76c3:98da:5071/64 Scope:Global
          inet6 addr: 2001:5c0:1508:f300::1/64 Scope:Global
          inet6 addr: 2001:5c0:1508:f300:20c:29ff:fee7:6bb4/64 Scope:Global
          inet6 addr: fe80::20c:29ff:fee7:6bb4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:298 errors:0 dropped:0 overruns:0 frame:0
          TX packets:335 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:37160 (37.1 KB)  TX bytes:41451 (41.4 KB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:243 errors:0 dropped:0 overruns:0 frame:0
          TX packets:243 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:22118 (22.1 KB)  TX bytes:22118 (22.1 KB)

tun       Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet6 addr: 2001:5c0:1400:b::3b85/128 Scope:Global
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1280  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:528 (528.0 B)  TX bytes:264 (264.0 B)

tun02     Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:127.0.1.1  P-t-P:127.0.1.1  Mask:255.255.255.255
          inet6 addr: fe80::4c0:1508:f300:1/64 Scope:Link
          inet6 addr: 2001:5c0:1508:f300::1/64 Scope:Global
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
```

UDP6 Server + client

We need the following:

A Border Router (running on a Z1 mote)

```
cd examples/zolertia/tutorial/02-ipv6/02-border-router
make border-router.upload && make connect-router PREFIX=aaaa::1/64
```

An UDP Client (running on a Z1 mote)

```
cd examples/zolertia/tutorial/02-ipv6/03-udp-client-and-server
make 03-udp-client.upload MOTES=/dev/ttyUSB1 && make login MOTES=/dev/ttyUSB1
```

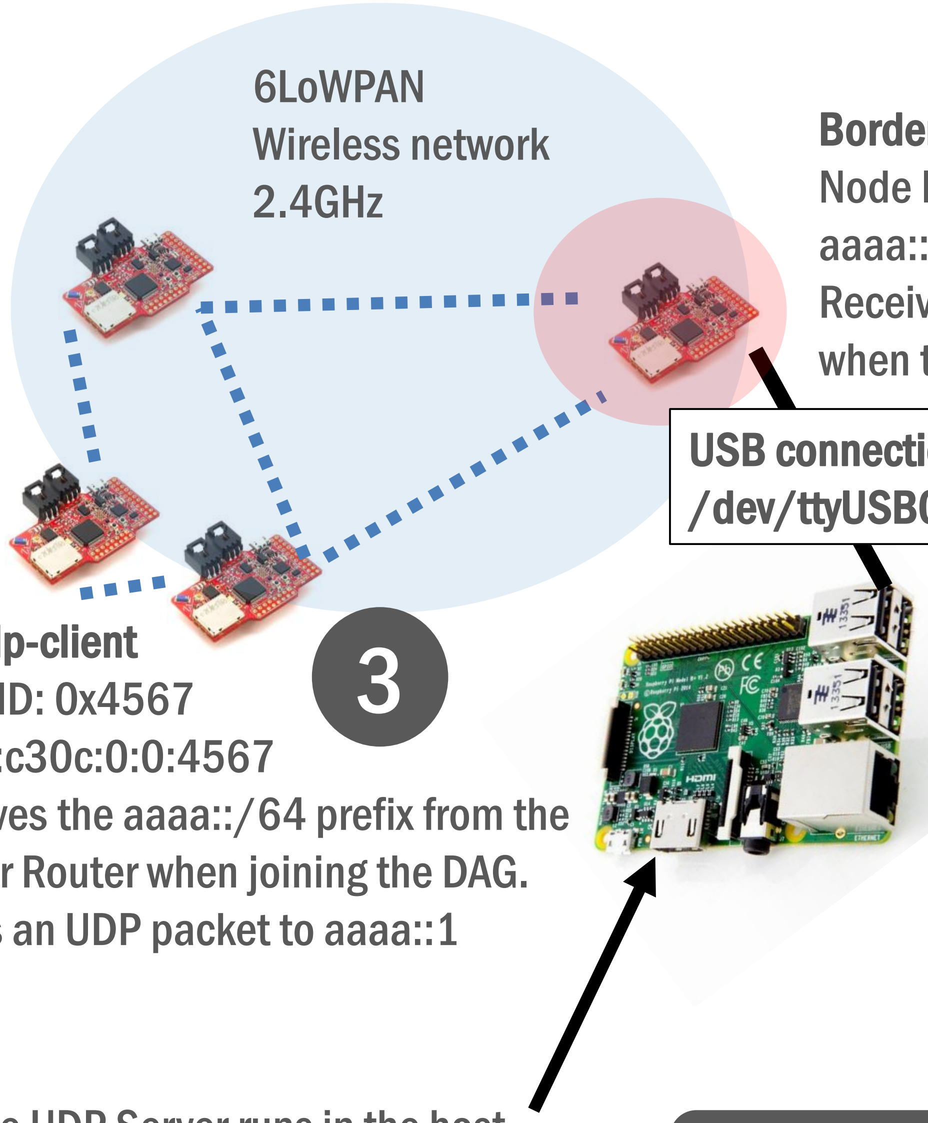
The UDP server (running on the Raspberry Pi)

A Python script named “IFTTT_client.py” or “UDPServer.py”

```
cd examples/zolertia/tutorial/02-ipv6/03-udp-client-and-server
python UDPServer.py  or alternatively Python IFTTT_client.py
```

Always make a “make motelist” to know what USB ports the Z1 motes are using, remember to write down the Product ID numbers to help recognizing the devices! If you are unsure about what application is running in a Z1 mote, use “make login” with the MOTES argument and press the RESET button, it will show the name of the process running ;-)

examples/zolertia/tutorial/02-ipv6/03-client-and-server



6LoWPAN
Wireless network
2.4GHz

Border Router 2
Node ID: 0x1234
aaaa::c30c:0:0:1234
Receives the prefix from tunslip6 (over the USB) when the tunnel “tun0” is created with tunslip6

USB connection to
/dev/ttyUSB0

Tunnel interface “tun0” 1
aaaa::1/64
Created when running the tunslip6 script in the host. Is a virtual tunnel interface, it sends the aaaa::/64 prefix to the Border Router. The Raspberry Pi will have a “tun0” interface with an aaaa::1/64 address

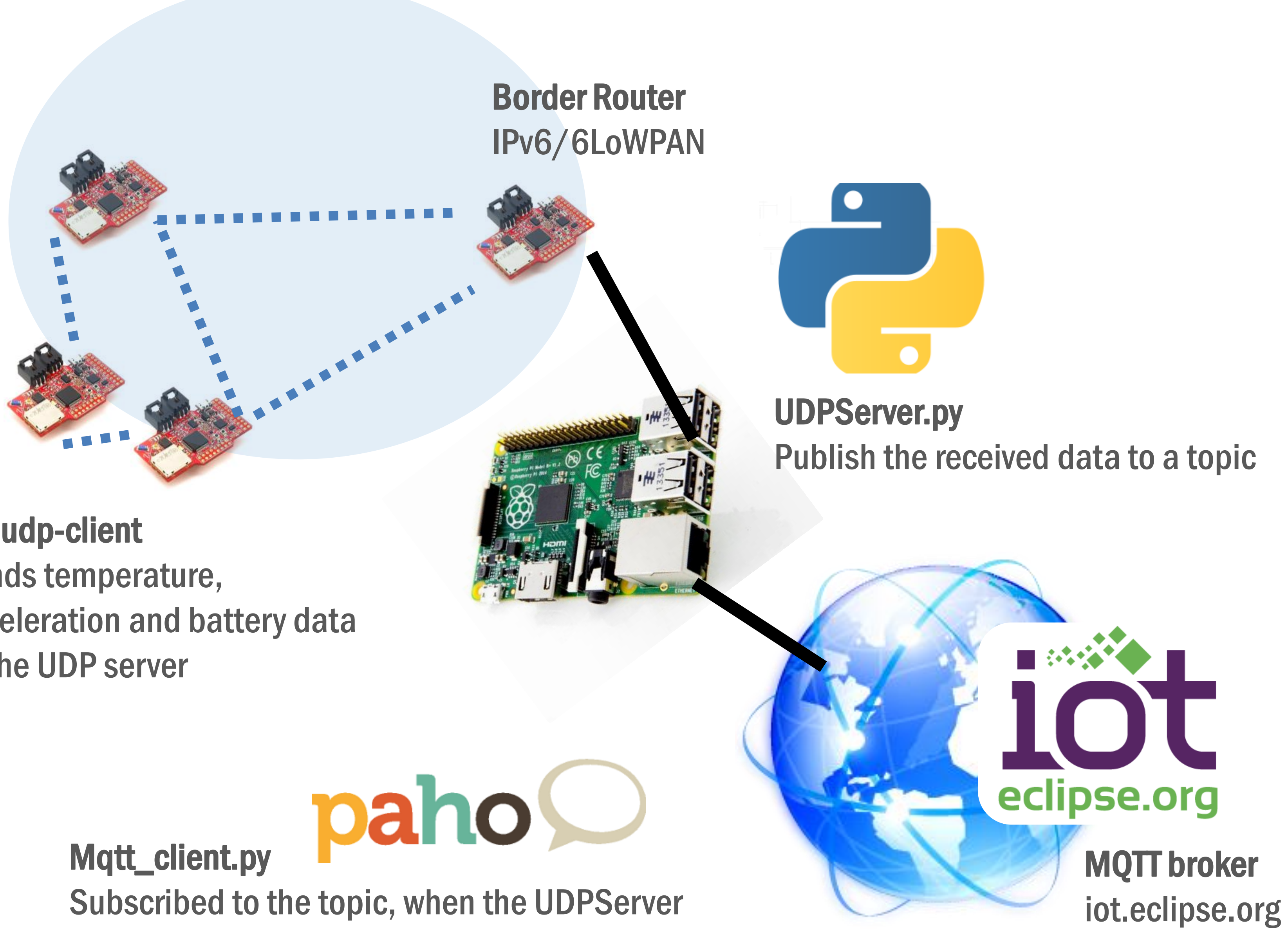
03-udp-client 3
Node ID: 0x4567
aaaa::c30c:0:0:4567
Receives the aaaa::/64 prefix from the Border Router when joining the DAG.
Sends an UDP packet to aaaa::1

When the UDP Server runs in the host, it will use the same address as the host, in this case the aaaa::1/64. 4

In the “contiki/tools” location, to create a tunnel type:

```
sudo ./tunslip6  -s /dev/ttyUSB0 –t tun0 aaaa::1/64
```

examples/zolertia/tutorial/02-ipv6/03-client-and-server



Border Router
IPv6/6LoWPAN

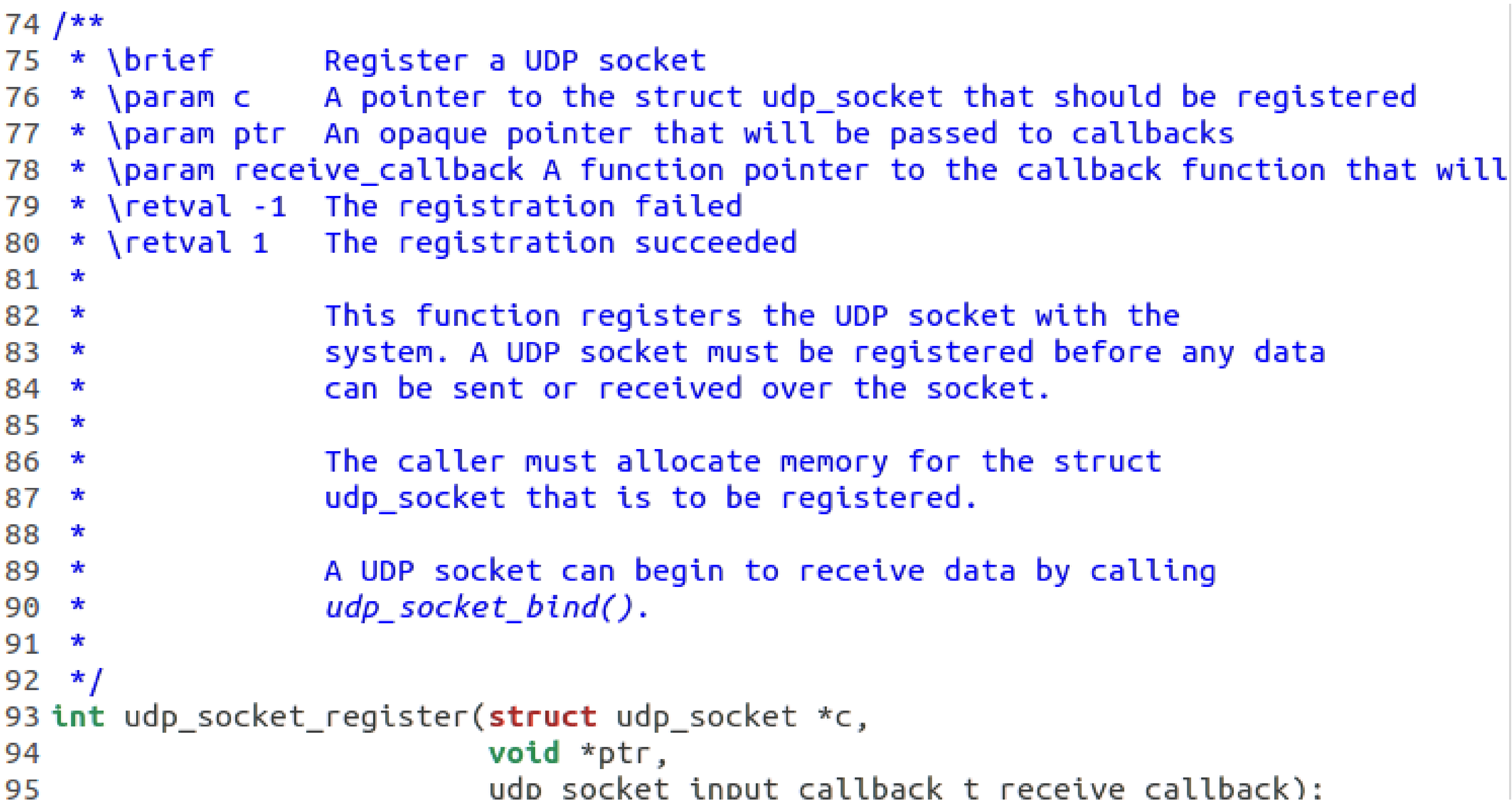
UDPServer.py
Publish the received data to a topic

03-udp-client
Sends temperature, acceleration and battery data to the UDP server

Mqtt_client.py
Subscribed to the topic, when the UDPServer publishes something we received the message

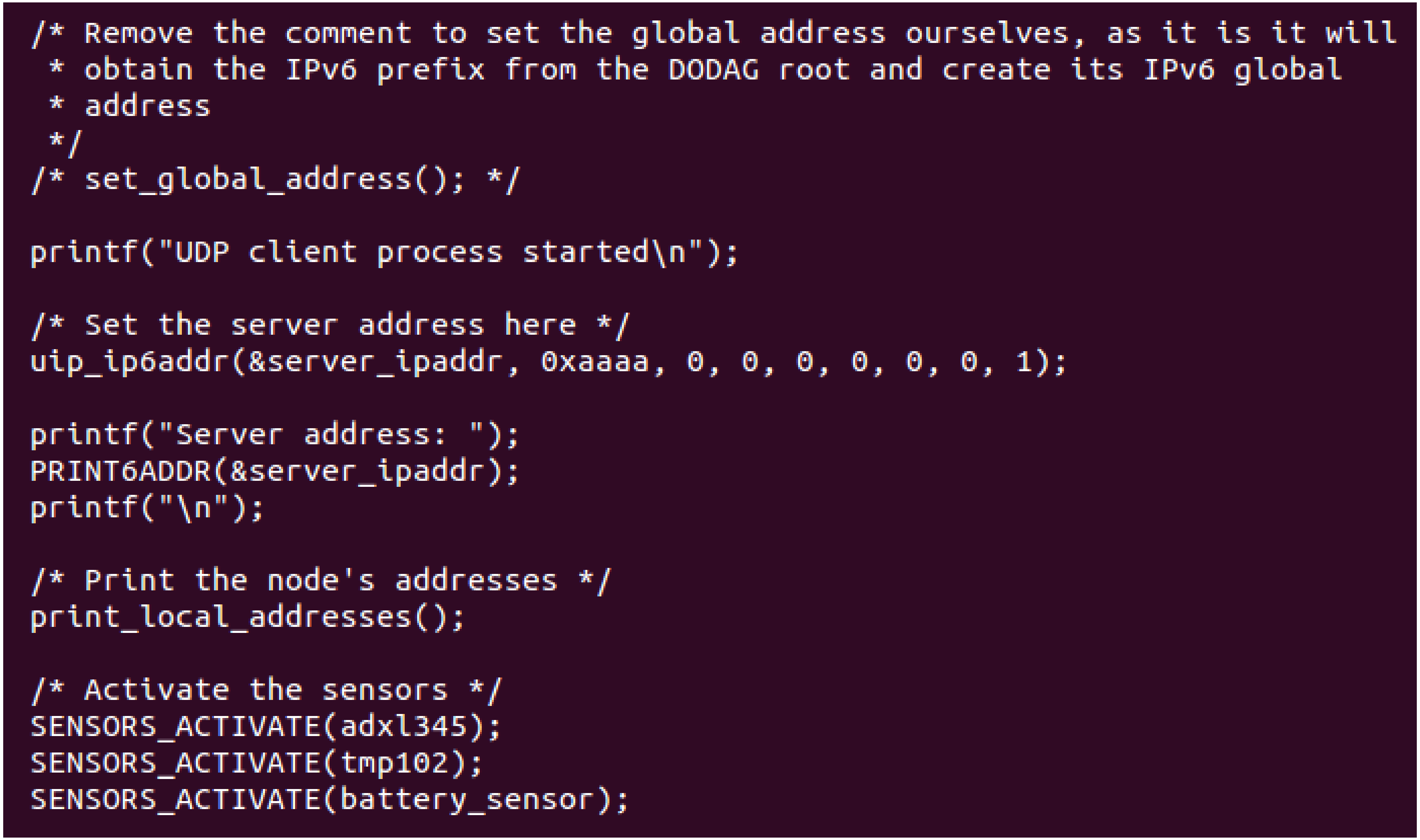
MQTT broker
iot.eclipse.org

examples/zolertia/tutorial/02-ipv6/03-client-and-server



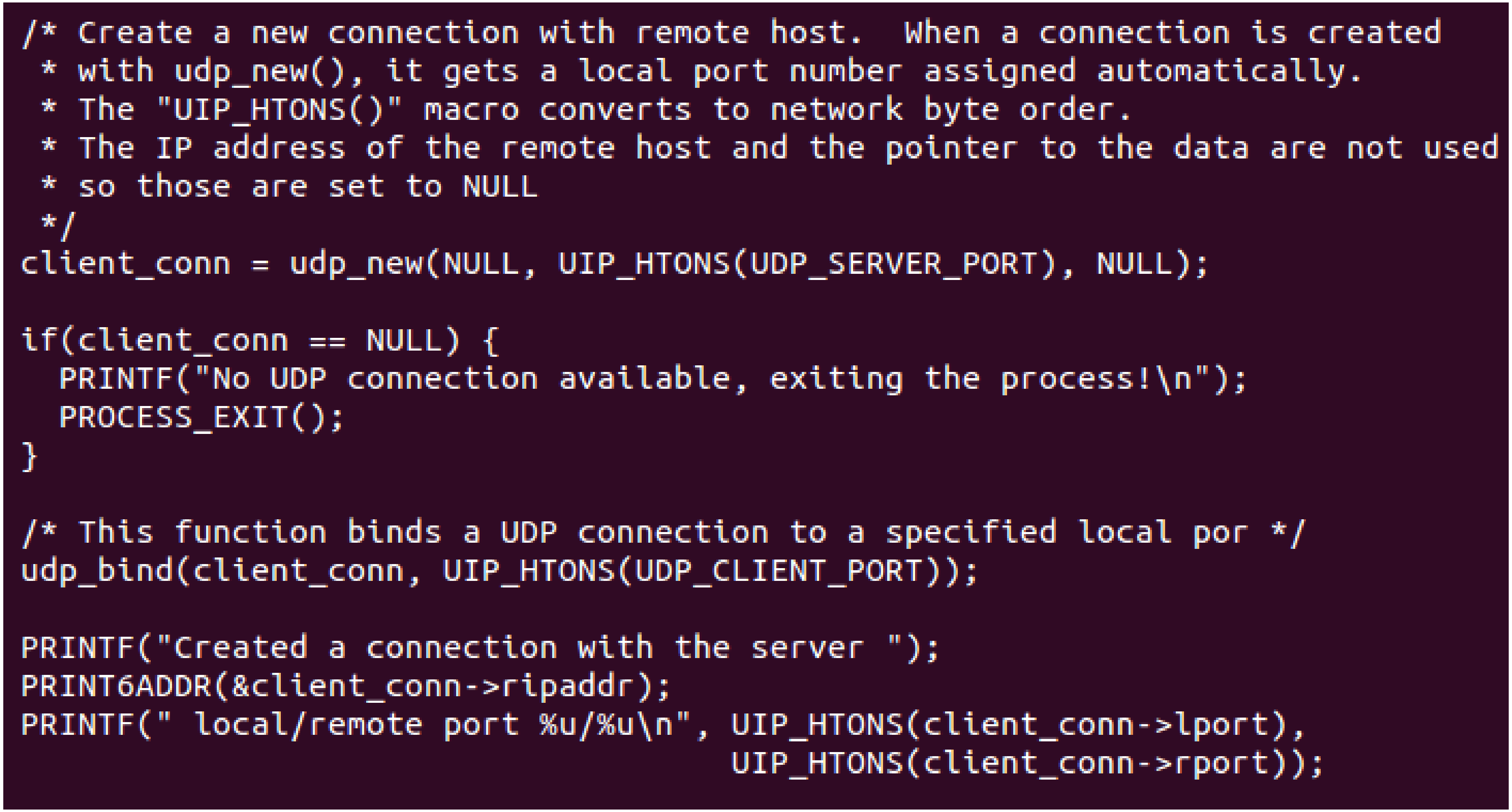
```
/**
 * \brief      Register a UDP socket
 * \param c    A pointer to the struct udp_socket that should be registered
 * \param ptr  An opaque pointer that will be passed to callbacks
 * \param receive_callback A function pointer to the callback function that will
 * \retval -1  The registration failed
 * \retval 1   The registration succeeded
 *
 *             This function registers the UDP socket with the
 *             system. A UDP socket must be registered before any data
 *             can be sent or received over the socket.
 *
 *             The caller must allocate memory for the struct
 *             udp_socket that is to be registered.
 *
 *             A UDP socket can begin to receive data by calling
 *             udp_socket_bind().
 *
 */
int udp_socket_register(struct udp_socket *c,
                        void *ptr,
                        udp_socket_input_callback_t receive_callback);
```

core/net/ip/udp-socket.h
examples/zolertia/tutorial/02-ipv6/03-client-and-server



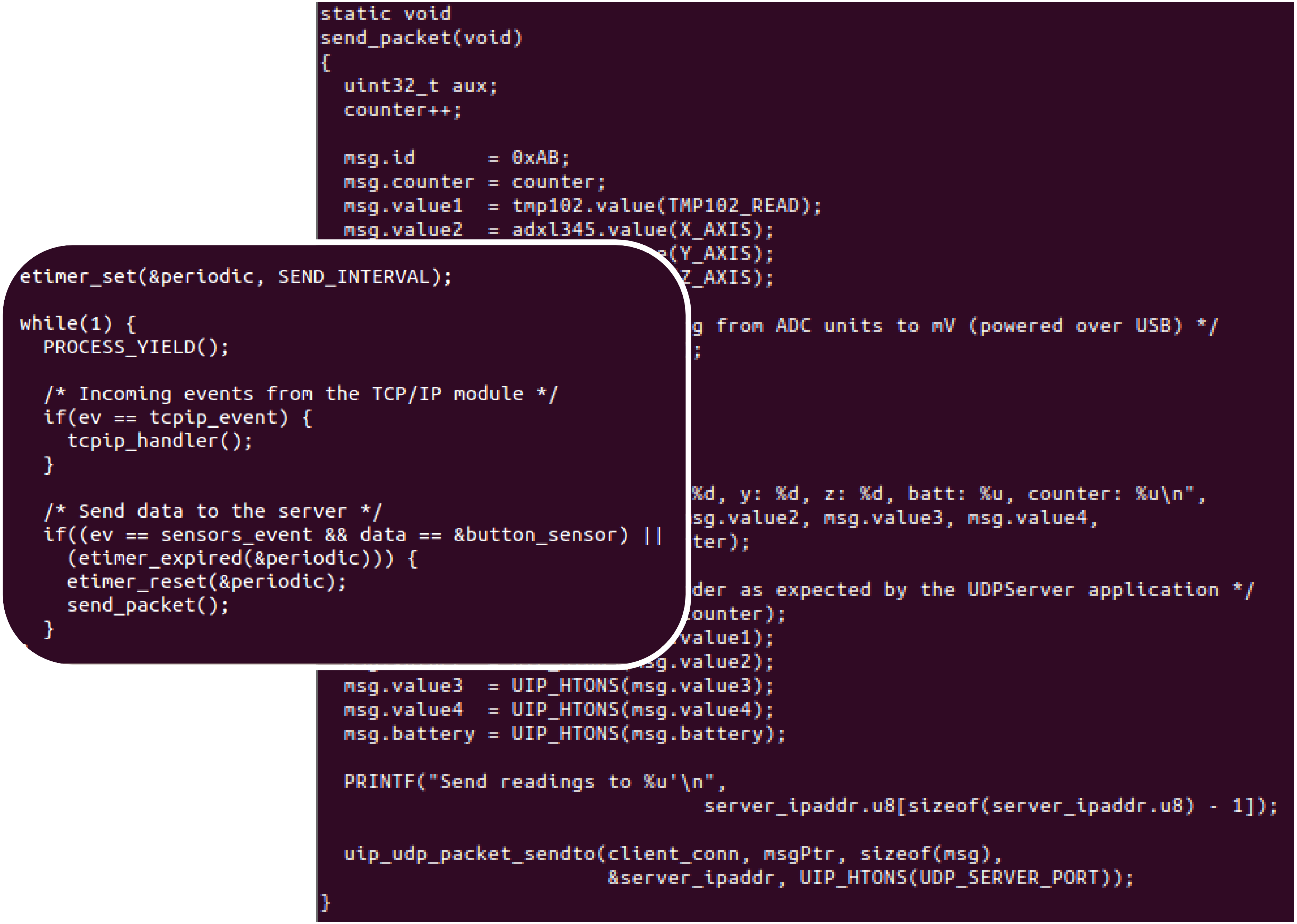
```
/* Remove the comment to set the global address ourselves, as it is it will
 * obtain the IPv6 prefix from the DODAG root and create its IPv6 global
 * address
 */
/* set_global_address(); */

printf("UDP client process started\n");

/* Set the server address here */
uip_ip6addr(&server_ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 1);

printf("Server address: ");
PRINT6ADDR(&server_ipaddr);
printf("\n");

/* Print the node's addresses */
print_local_addresses();

/* Activate the sensors */
SENSORS_ACTIVATE(adxl345);
SENSORS_ACTIVATE(tmp102);
SENSORS_ACTIVATE(battery_sensor);
```

examples/zolertia/tutorial/02-ipv6/03-client-and-server



```
/* Create a new connection with remote host.  When a connection is created
 * with udp_new(), it gets a local port number assigned automatically.
 * The "UIP_HTONS()" macro converts to network byte order.
 * The IP address of the remote host and the pointer to the data are not used
 * so those are set to NULL
 */
client_conn = udp_new(NULL, UIP_HTONS(UDP_SERVER_PORT), NULL);

if(client_conn == NULL) {
  PRINTF("No UDP connection available, exiting the process!\n");
  PROCESS_EXIT();
}

/* This function binds a UDP connection to a specified local por */
udp_bind(client_conn, UIP_HTONS(UDP_CLIENT_PORT));

PRINTF("Created a connection with the server ");
PRINT6ADDR(&client_conn->ripaddr);
PRINTF(" local/remote port %u/%u\n", UIP_HTONS(client_conn->lport),
                                     UIP_HTONS(client_conn->rport));
```

examples/zolertia/tutorial/02-ipv6/03-client-and-server

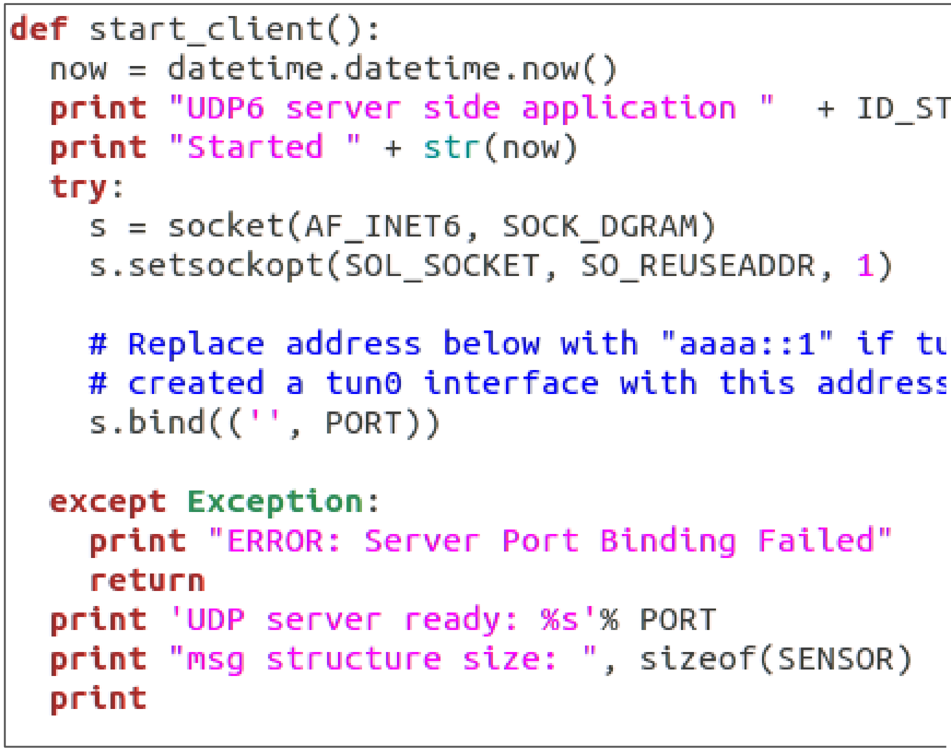


```
static void
send_packet(void)
{
  uint32_t aux;
  counter++;

  msg.id      = 0xAB;
  msg.counter = counter;
  msg.value1  = tmp102.value(TMP102_READ);
  msg.value2  = adxl345.value(X_AXIS);
                            e(Y_AXIS);
                             Z_AXIS);

                              g from ADC units to mV (powered over USB) */
                              ;

                              %d, y: %d, z: %d, batt: %u, counter: %u\n",
                              sg.value2, msg.value3, msg.value4,
                              ter);

                              der as expected by the UDPServer application */
                              ounter);
                              value1);
                              sg.value2);
  msg.value3  = UIP_HTONS(msg.value3);
  msg.value4  = UIP_HTONS(msg.value4);
  msg.battery = UIP_HTONS(msg.battery);

  PRINTF("Send readings to %u'\n",
                                server_ipaddr.u8[sizeof(server_ipaddr.u8) - 1]);

  uip_udp_packet_sendto(client_conn, msgPtr, sizeof(msg),
                        &server_ipaddr, UIP_HTONS(UDP_SERVER_PORT));
}
```

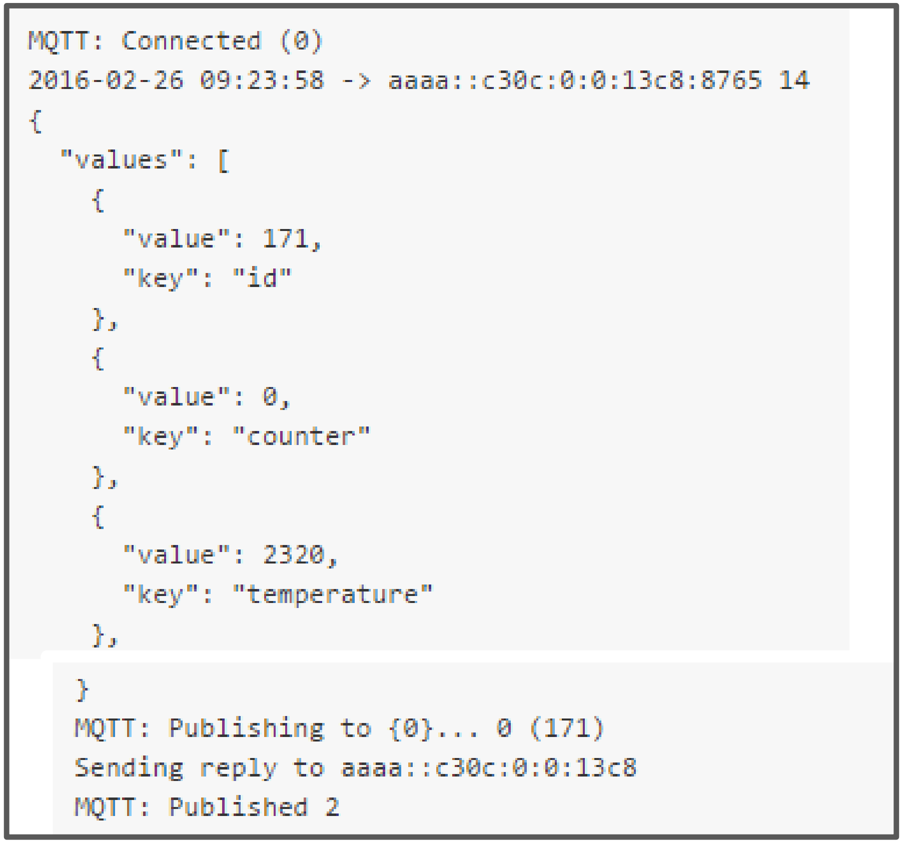
```
etimer_set(&periodic, SEND_INTERVAL);

while(1) {
  PROCESS_YIELD();

  /* Incoming events from the TCP/IP module */
  if(ev == tcpip_event) {
    tcpip_handler();
  }

  /* Send data to the server */
  if((ev == sensors_event && data == &button_sensor) ||
    (etimer_expired(&periodic))) {
    etimer_reset(&periodic);
    send_packet();
  }
```

examples/zolertia/tutorial/02-ipv6/03-client-and-server

python UDPServer.py



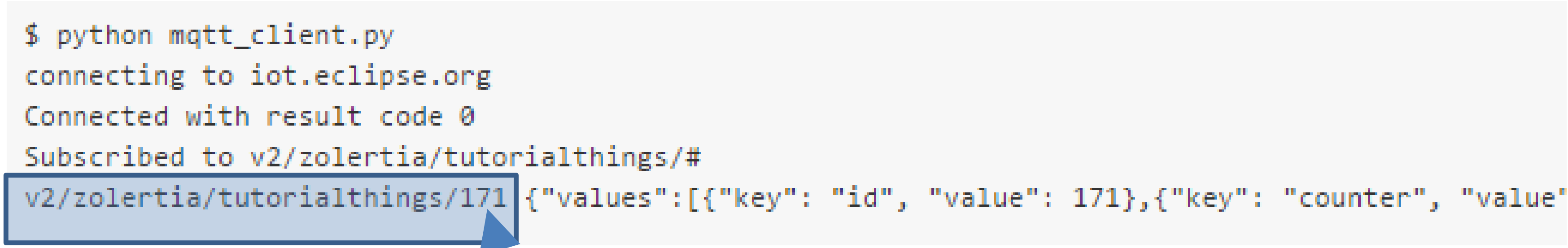
```
def start_client():
  now = datetime.datetime.now()
  print "UDP6 server side application "  + ID_STRING
  print "Started " + str(now)
  try:
    s = socket(AF_INET6, SOCK_DGRAM)
    s.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)

    # Replace address below with "aaaa::1" if tu
    # created a tun0 interface with this address
    s.bind(('', PORT))

  except Exception:
    print "ERROR: Server Port Binding Failed"
    return
  print 'UDP server ready: %s'% PORT
  print "msg structure size: ", sizeof(SENSOR)
  print
```



```
MQTT: Connected (0)
2016-02-26 09:23:58 -> aaaa::c30c:0:0:13c8:8765 14
{
  "values": [
    {
      "value": 171,
      "key": "id"
    },
    {
      "value": 0,
      "key": "counter"
    },
    {
      "value": 2320,
      "key": "temperature"
    },
  }
MQTT: Publishing to {0}... 0 (171)
Sending reply to aaaa::c30c:0:0:13c8
MQTT: Published 2
```

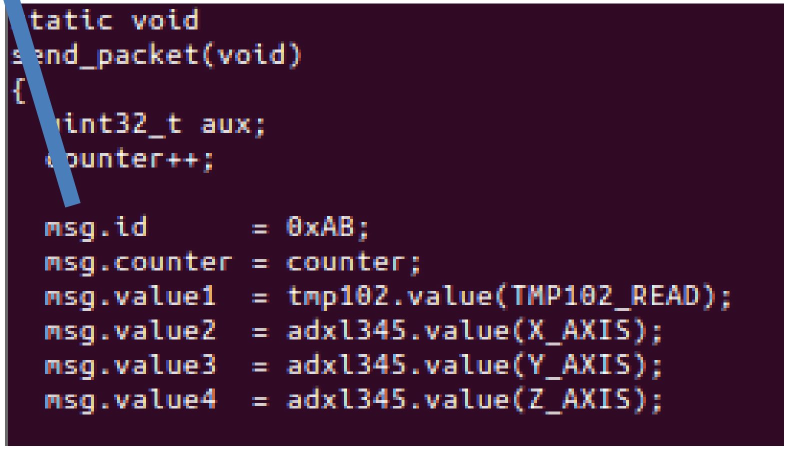
examples/zolertia/tutorial/02-ipv6/03-client-and-server

python mqtt_client.py



```
$ python mqtt_client.py
connecting to iot.eclipse.org
Connected with result code 0
Subscribed to v2/zolertia/tutorialthings/#
v2/zolertia/tutorialthings/171 {"values":[{"key": "id", "value": 171},{"key": "counter", "value"
```

This is the topic



```
static void
send_packet(void)
{
  uint32_t aux;
  counter++;

  msg.id      = 0xAB;
  msg.counter = counter;
  msg.value1  = tmp102.value(TMP102_READ);
  msg.value2  = adxl345.value(X_AXIS);
  msg.value3  = adxl345.value(Y_AXIS);
  msg.value4  = adxl345.value(Z_AXIS);
```

examples/zolertia/tutorial/02-ipv6/03-client-and-server

Antonio Liñán Colina

alinan@zolertia.com
antonio.lignan@gmail.com

Twitter: @4Li6NaN
LinkedIn: Antonio Liñan Colina
github.com/alignan
hackster.io/alinan