RF and network basics

Antonio Liñán Colina

Contiki

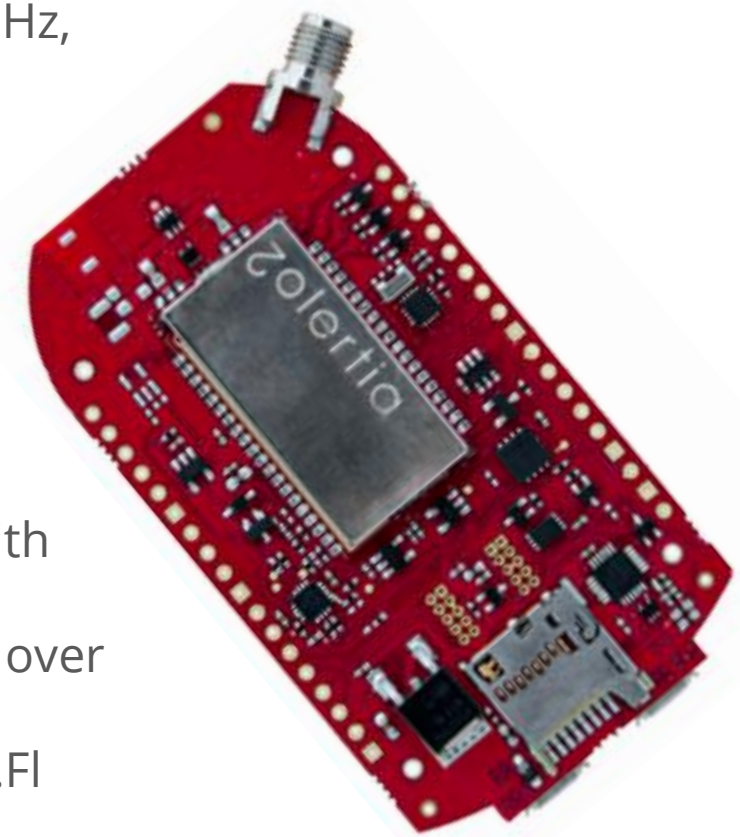
The Open Source OS for the Internet of Things

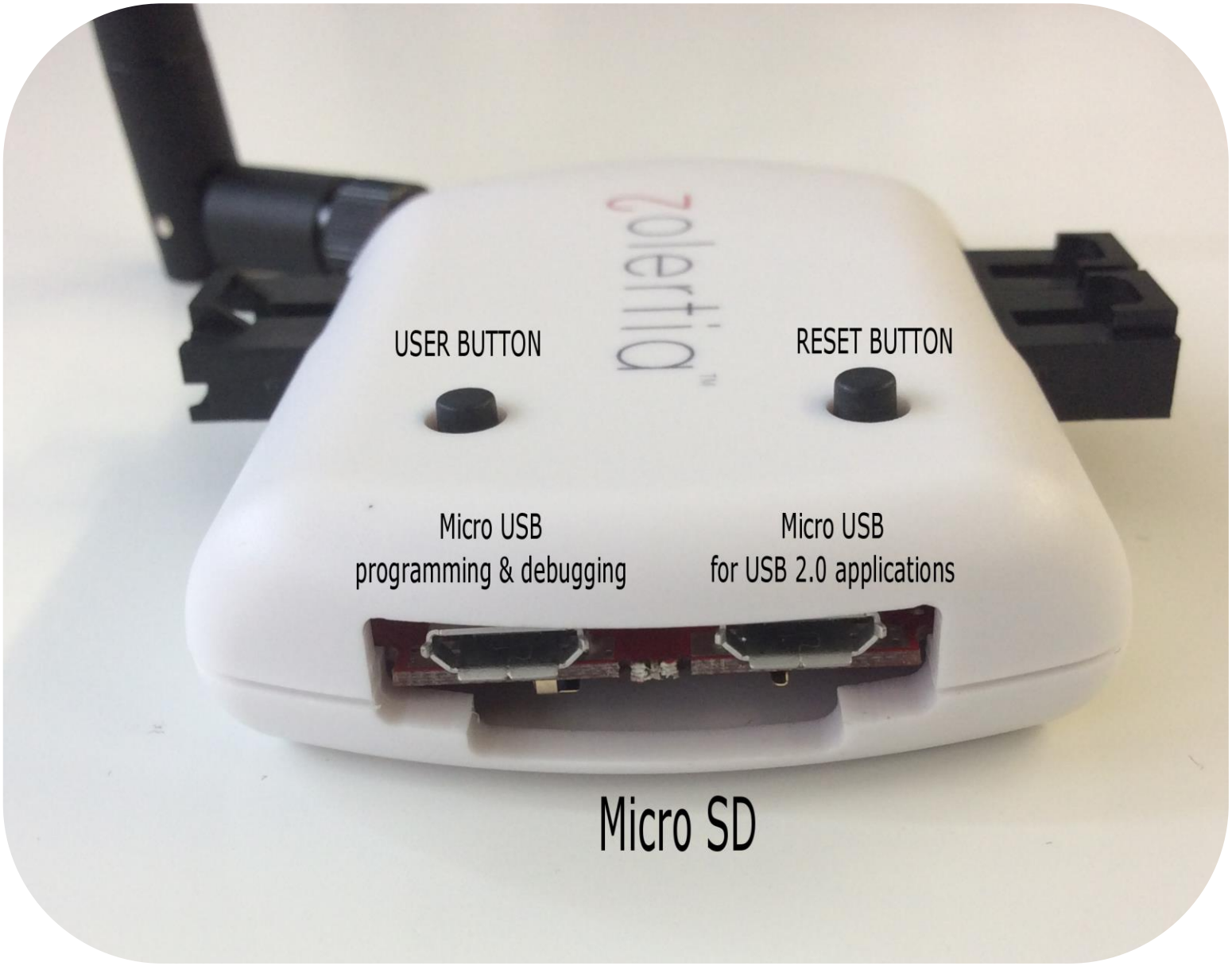
- Architectures: 8-bit, 16-bit, 32-bit
- Open Source (source code openly available)
- IPv4/IPv6/Rime networking
- Devices with < 8KB RAM
- Typical applications < 50KB Flash
- Vendor and platform independent
- C language
- Developed and contributed by Universities, Research centers and industry contributors
- +10 years development

Zolertia RE-Mote

Zolertia RE-Mote (Zoul inside)

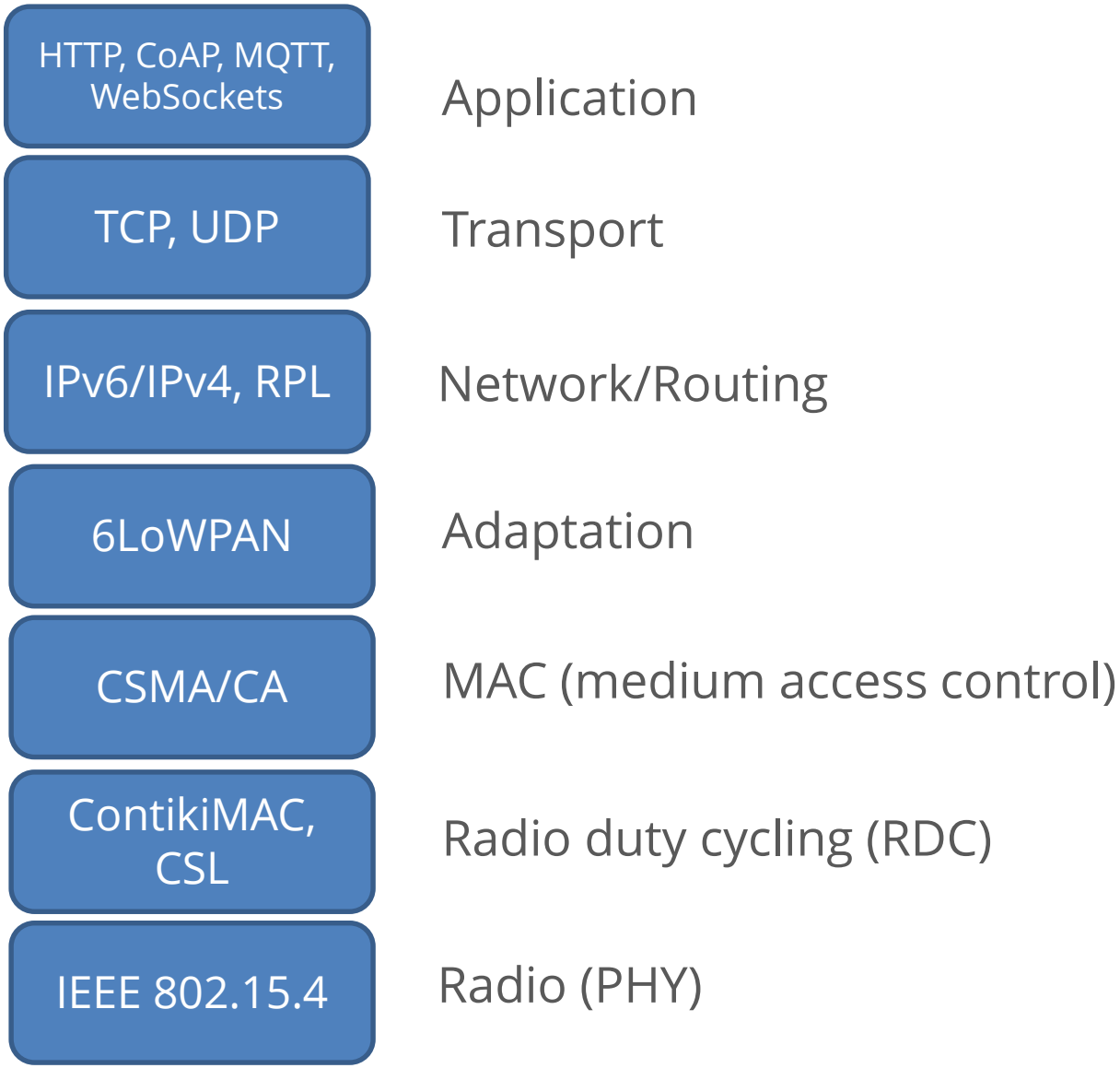
- ARM Cortex-M3, 32MHz, 32KB RAM, 512KB FLASH
- Double Radio: ISM 2.4GHz & 863-925MHz, IEEE 802.15.4-2006/e/g
- Hardware encryption engine and acceleration
- USB programing ready
- Real-Time Clock and Calendar
- Micro SD slot and RGB colors
- Shutdown mode down to 150nA
- USB 2.0 port for applications
- Built-in LiPo battery charger to work with energy harvesting and solar panels
- On-board RF switch to use both radios over the same RP-SMA connector
- Pads to use an external 2.4GHz over U.Fl connector, o solder a chip antenna

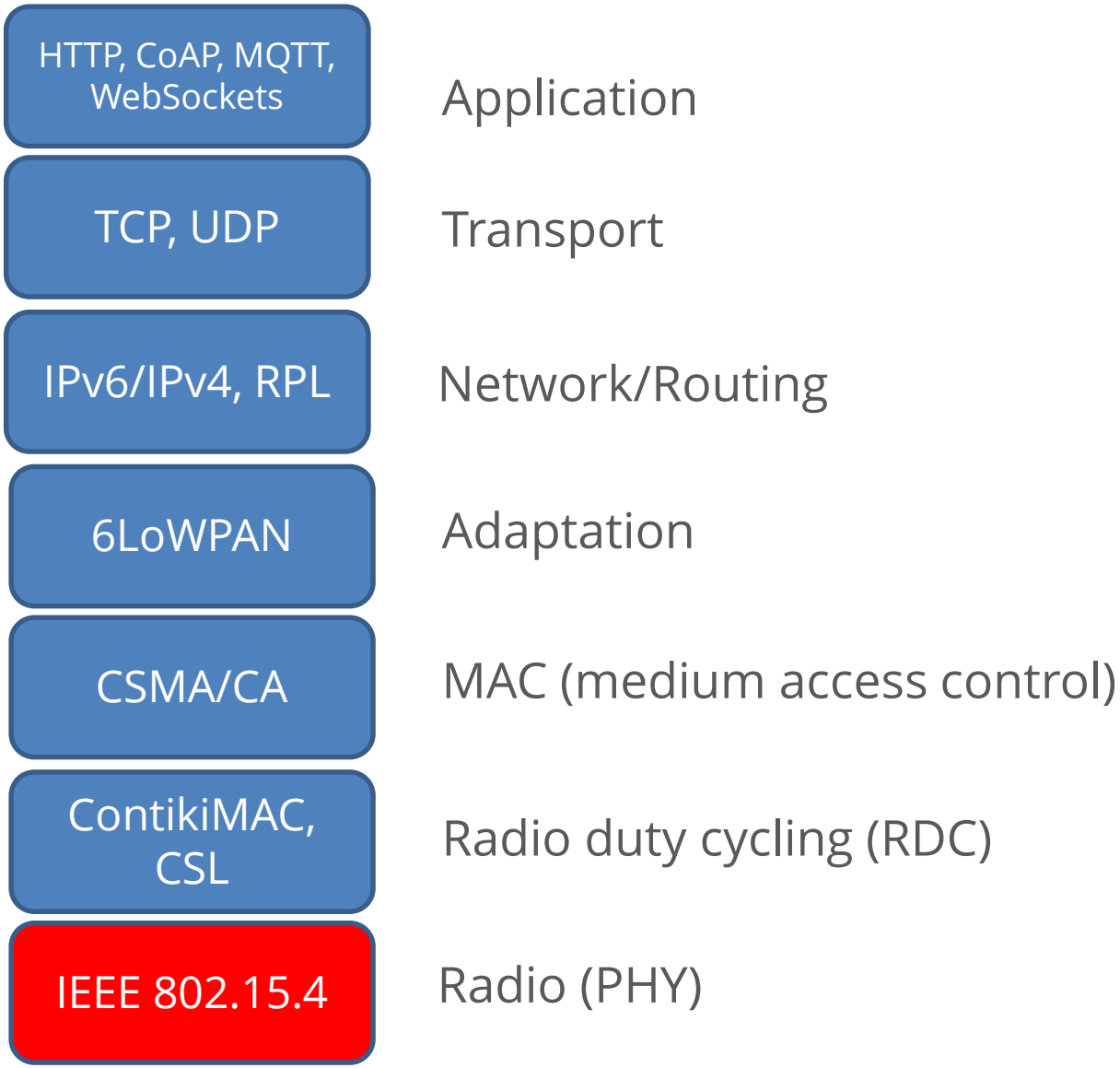




02-ipv6

examples/zolertia/tutorial/02-ipv6

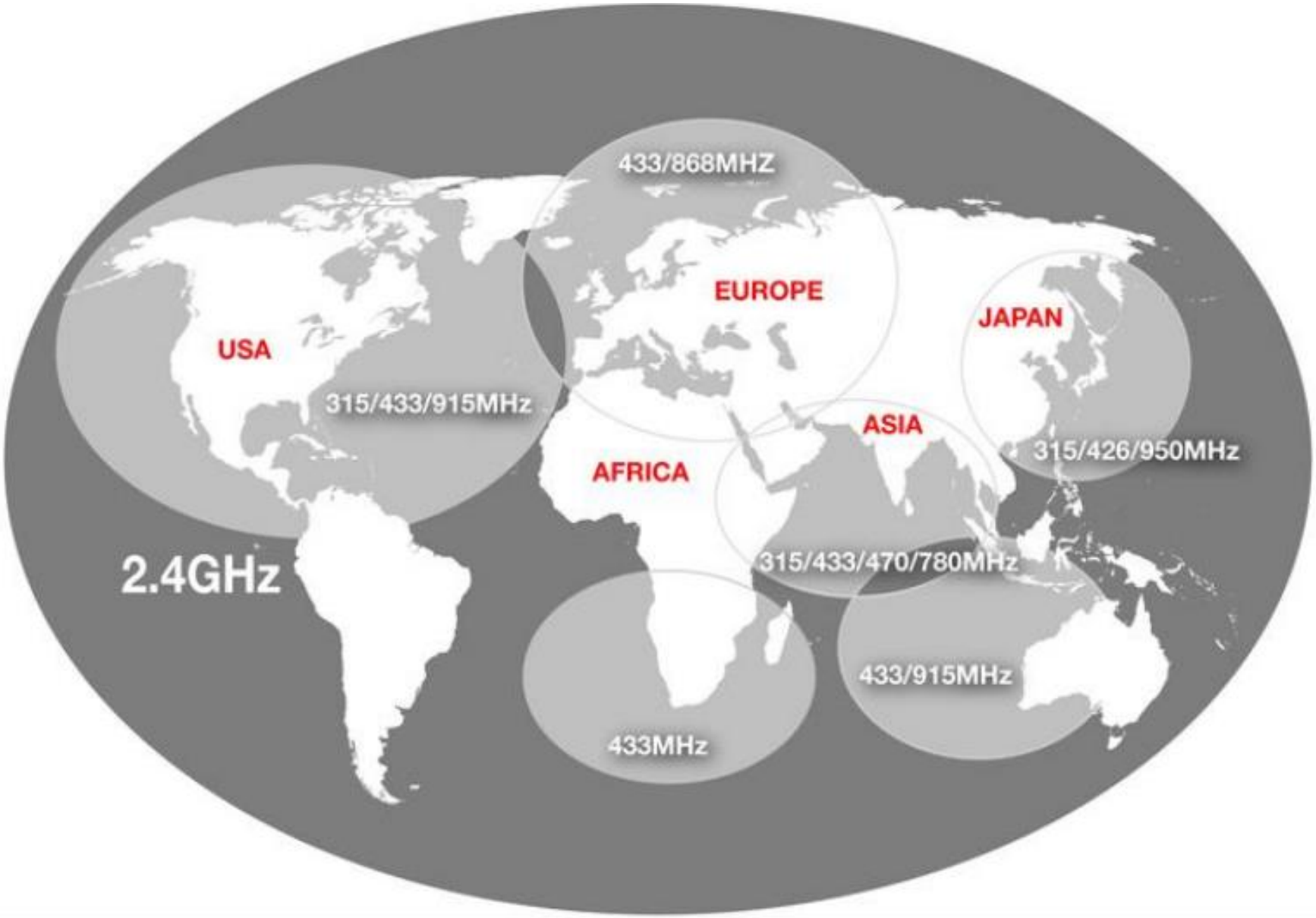



HTTP, CoAP, MQTT, WebSockets
Application
TCP, UDP
Transport
IPv6/IPv4, RPL
Network/Routing
6LoWPAN
Adaptation
CSMA/CA
MAC (medium access control)
ContikiMAC, CSL
Radio duty cycling (RDC)
IEEE 802.15.4
Radio (PHY)

examples/zolertia/tutorial/02-ipv6

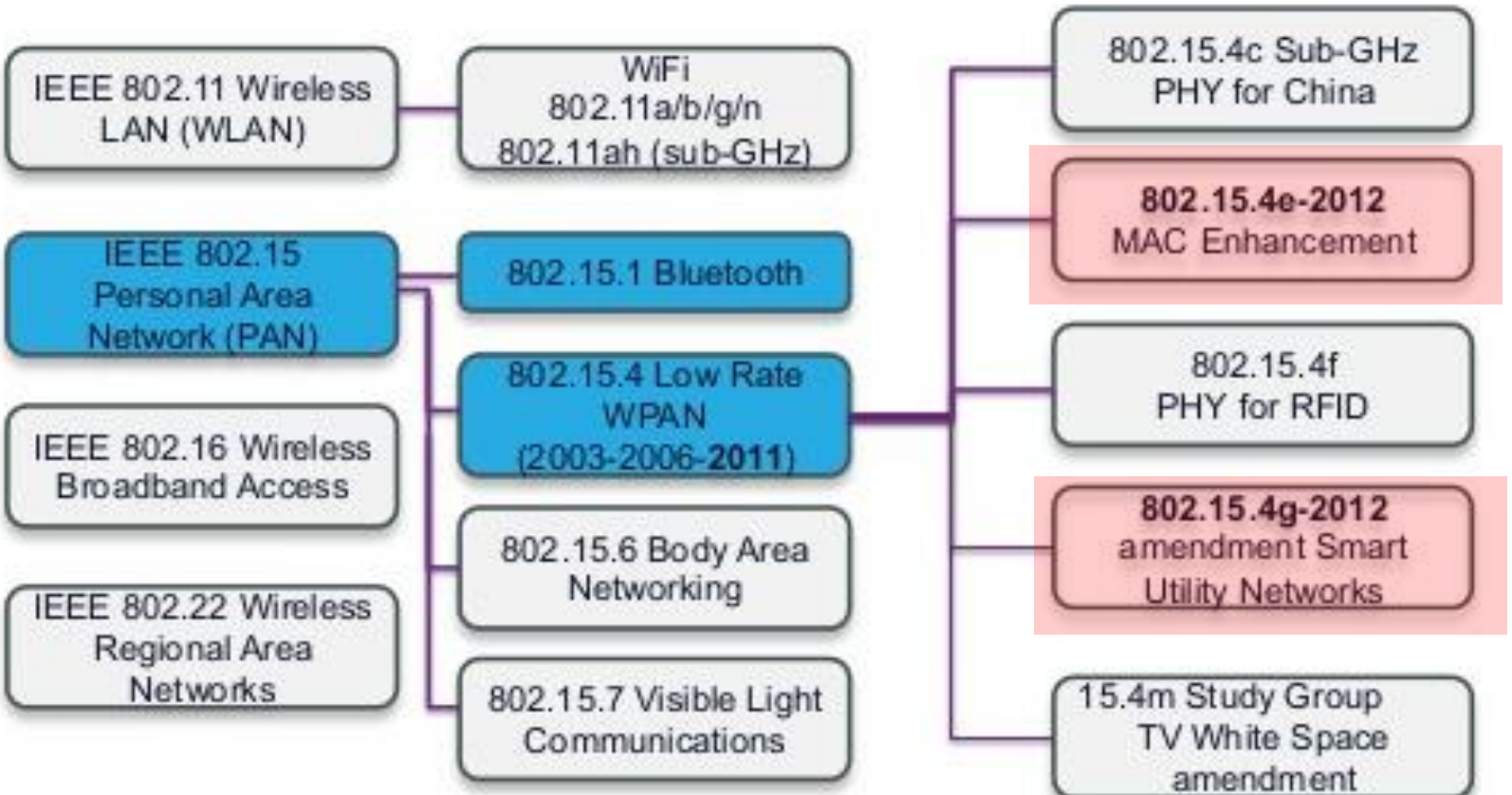
Low-power radios

- Lowest power consumption respect to WiFI and alike
- Depending on the operating frequency it might have the same or longer wireless range than WiFI
- Slower data rate: 1.2kbps to 1Mbps, depending on the transceiver
- IEEE 802.15.4 standard defines operating modes according to countries regulations



http://www.ti.com

IEEE Standards



5

http://www.slideshare.net/Techtsunami/cn-prt-iot-v1

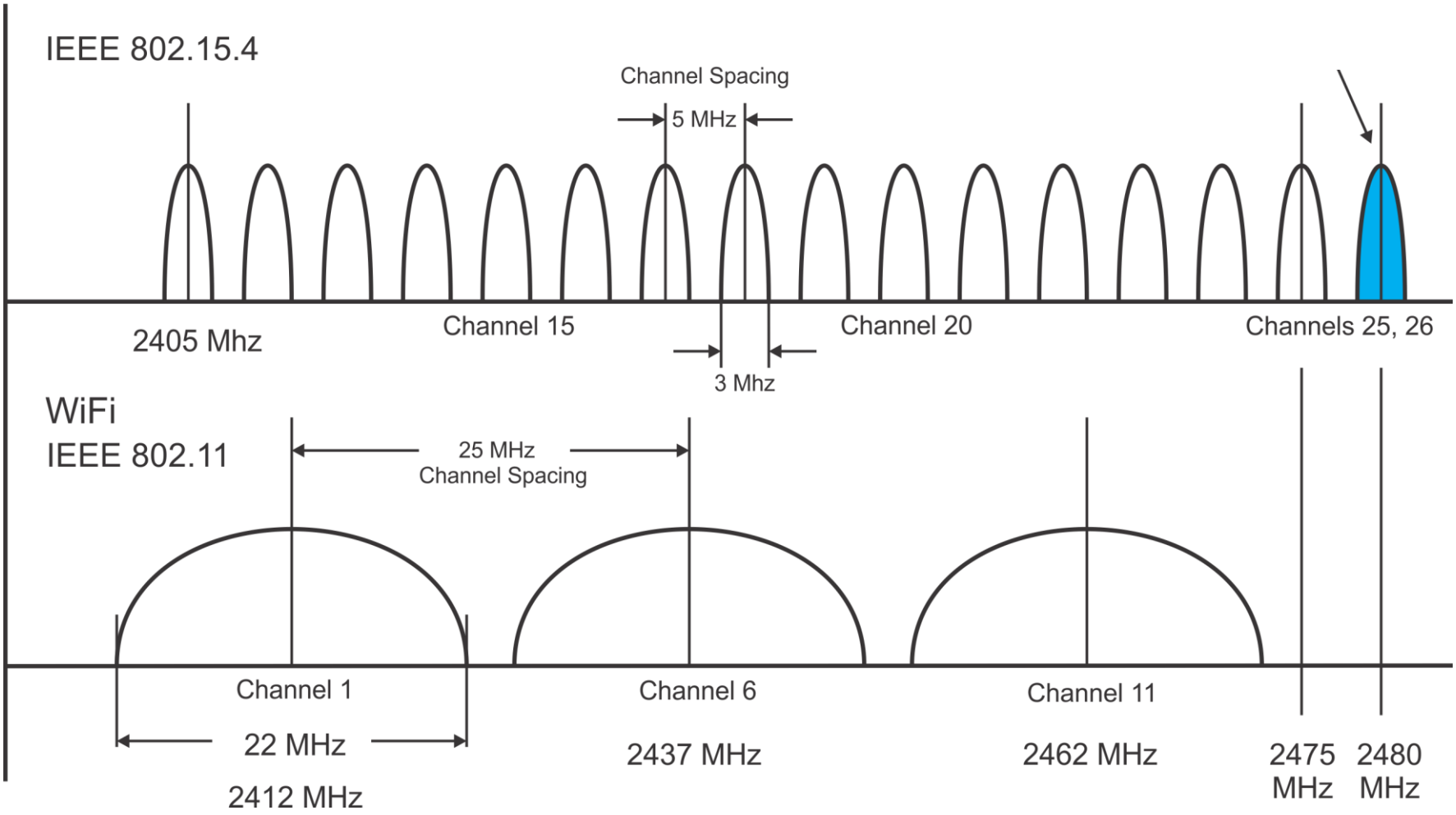
examples/zolertia/tutorial/02-ipv6

IEEE 802.15.4 standards

	802.15.4	802.15.4e	802.15.4g	802.15.4f
Frequency	2.4Ghz (DSSS + oQPSK), 868Mhz (DSSS + BPSK) 915Mhz (DSSS + BPSK)	2.4Ghz (DSSS + oQPSK, CSS+DQPSK), 868Mhz (DSSS + BPSK) 915Mhz (DSSS + BPSK)	2.4Ghz (DSSS + oQPSK, CSS+DQPSK), 868Mhz (DSSS + BPSK) 915Mhz (DSSS + BPSK)	2.4Ghz (DSSS + oQPSK, CSS+DQPSK), 868Mhz (DSSS + BPSK) 915Mhz (DSSS + BPSK) 3~10Ghz (BPM+BPSK)
Data rate	Upto 250kbps	Upto 800kbps	Up to 800kbps	
Differences	–	Mac Enhancements (time synchronization and channel hopping)	Phy Enhancements	Mac and Phy Enhancements
Frame Size (bytes)	127	N/A	Up to 2047	N/A
Range (m)	1 – 75+	1 – 75+	Upto 1km	N/A
Goals	General Low-power Sensing/Actuating	Industrial segments	Smart utilities	• Active RFID bi-directional • location determination applications

http://www.slideshare.net/usmanusb/elements-of-iot-connectivity-technologies

IEEE 802.15.4-2006/e, 2.4GHz channels (11-26)



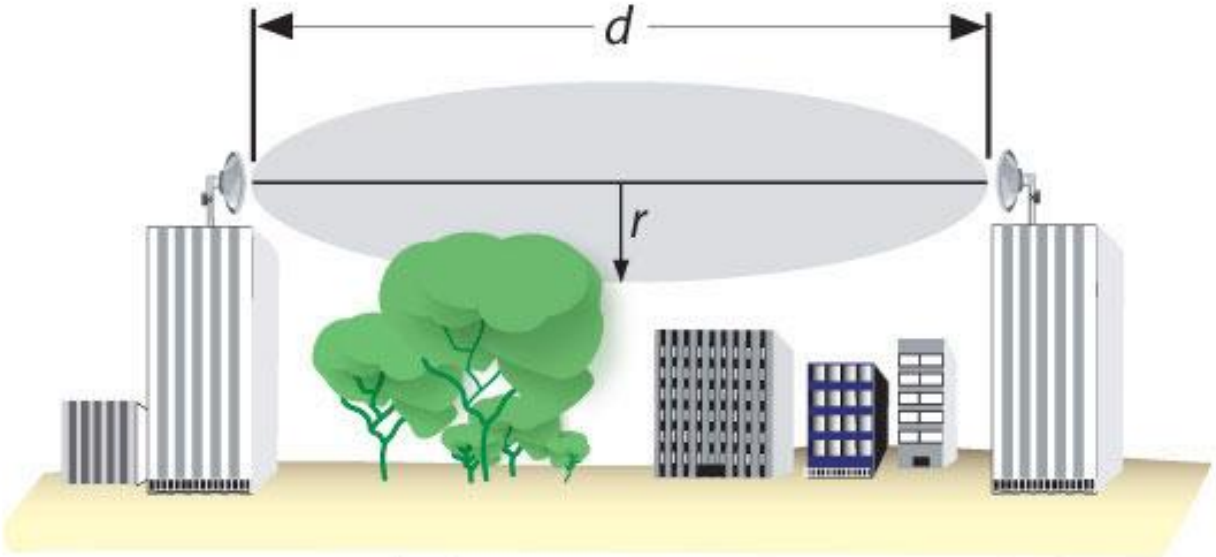
http://www.founten.com/

IEEE 802.15.4g MR-FSK

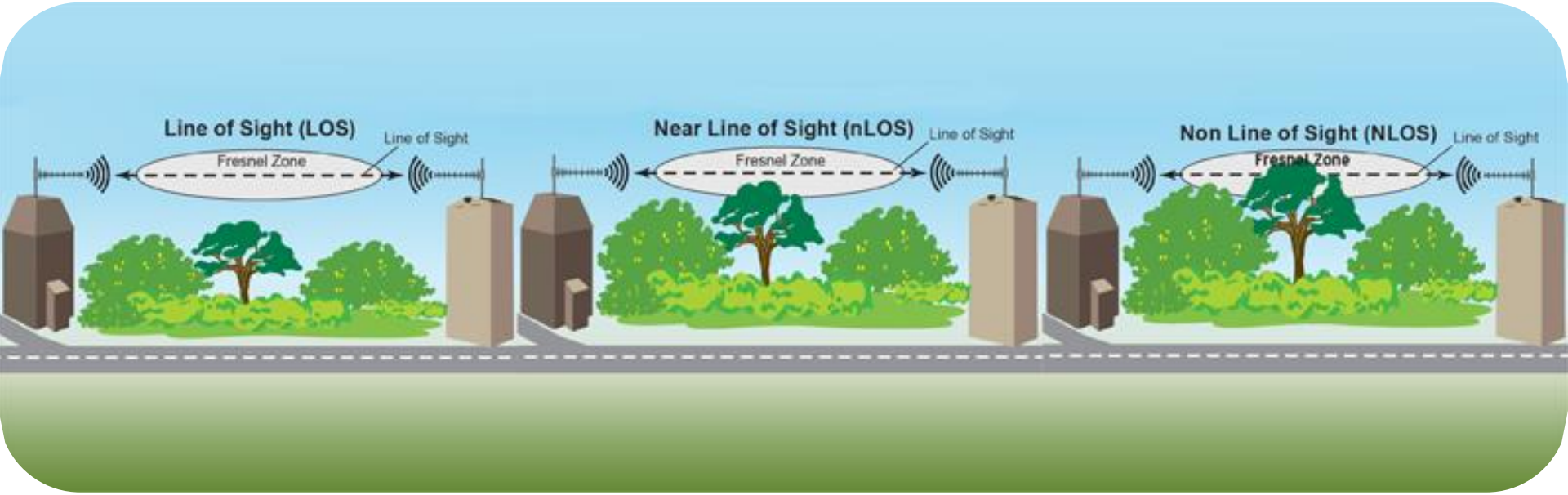
Frequency Band [MHz]	Bit rate [kbps] @ Modulation				
	Filtered 2FSK	Filtered 4FSK	O-QPSK	OFDM	
169.400 - 169.475	2.4 / 4.8	9.6	---	---	(Europe)
450 - 470	4.8	9.6	---	---	(US FCC Part 22/90)
470 - 510	50 / 100	200	6.25 - 50	*1	(China)
779 - 787	50 / 100	200	31.25 - 500	*1	(China)
863 - 870	50 / 100	200	---	*1	(Europe)
868 - 870	---	---	6.25 - 50	---	(Europe)
896 - 901	10 /20 /40	---	---	---	(US FCC Part 90)
901 - 902	10 /20 /40	---	---	---	(US FCC Part 24)
902 - 928	50 / 150 / 200	---	31.25 - 500	*1	(US)
917.0 - 923.5	50 / 150 / 200	---	31.25 - 500	*1	(Korea)
920 - 928	50 / 100 / 200	400	6.25 - 50	*1	(Japan)
928 - 960	10 /20 /40	---	---	---	(US, non-contiguous)
950 - 958	50 / 100 / 200	400	6.25 - 50	*1	(Japan)
1427 - 1518	10 /20 /40	---	---	---	(US and Canada, non-contiguous)
2400.0 - 2483.5	50 / 150 / 200	---	31.25 - 500	*1	

				Channel spacing
20 kbps	20 kbps	Filtered 2FSK	0.5	12.5 kHz
40 kbps	40 kbps		0.5	
50 kbps	50 kbps		1.0	200 kHz
100 kbps	100 kbps		1.0	400 kHz
150 kbps	150 kbps		0.5	
200 kbps	200 kbps		0.5	
			1.0	600 kHz
9.6 kbps	4.8 kbps	Filtered 4FSK	0.33	12.5 kHz
200 kbps	100 kbps		0.3	400 kHz
400 kbps	200 kbps		0.3	600 kHz

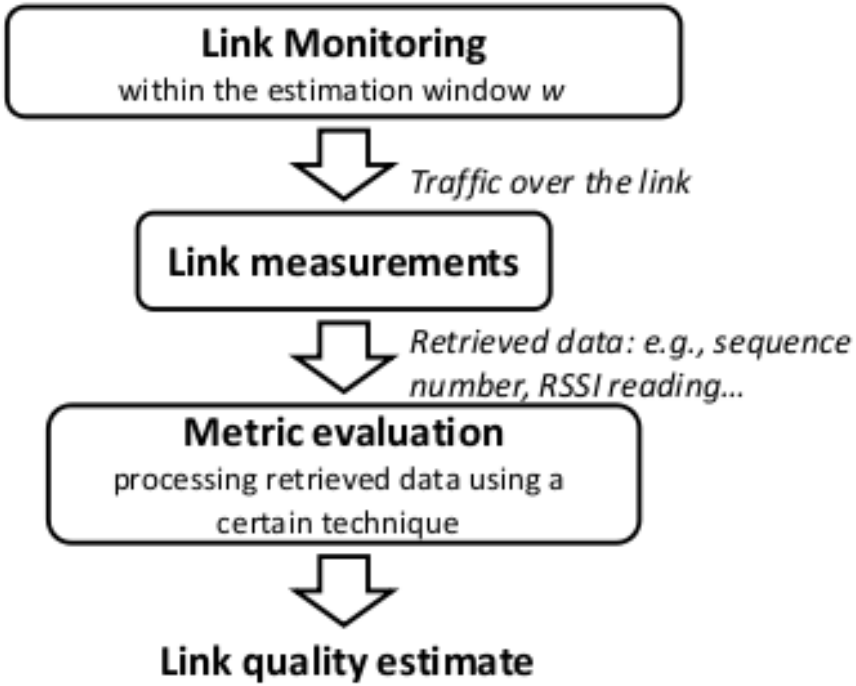
https://dl.cdn-anritsu.com



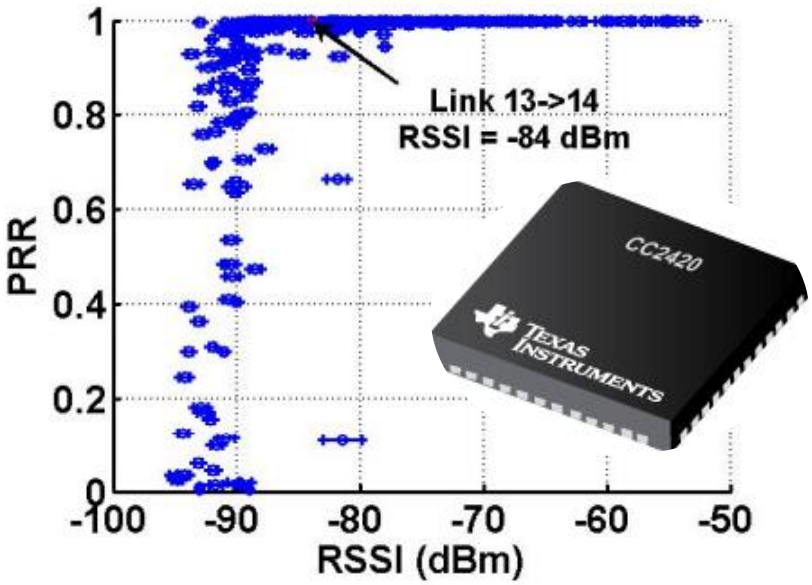
$$r_{(\text{in mts})} = 17.32 \times \sqrt{\frac{d_{(\text{in Km})}}{4f_{(\text{in GHz})}}} \qquad r_{(\text{in ft})} = 72.05 \times \sqrt{\frac{d_{(\text{in miles})}}{4f_{(\text{in GHz})}}}$$

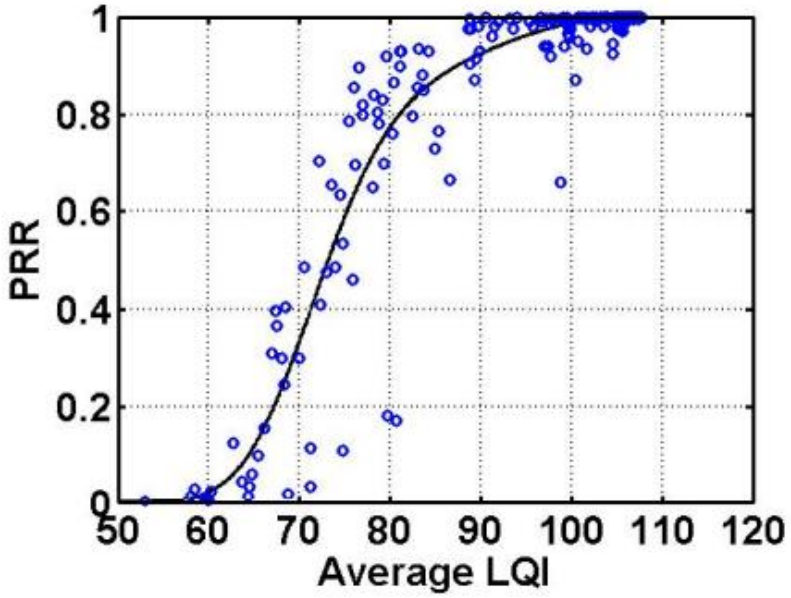


examples/zolertia/tutorial/02-ipv6



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





examples/zolertia/tutorial/02-ipv6

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

As the transmission power increases the wireless range should increase as well.

Table 4.1. CC2538 Transmission power recommended values (from SmartRF Studio[6])

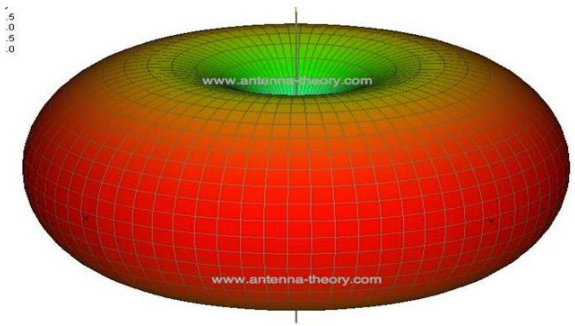
TX Power (dBm)	Value
+7	0xFF
+5	0xED
+3	0xD5
+1	0xC5
0	0xB6
-1	0xB0
-3	0xA1
-5	0x91
-7	0x88
-9	0x72
-11	0x62
-13	0x58
-15	0x42
-24	**0x00**

Table 4.3. CC1200 Transmission power recommended values (from SmartRF Studio[10])

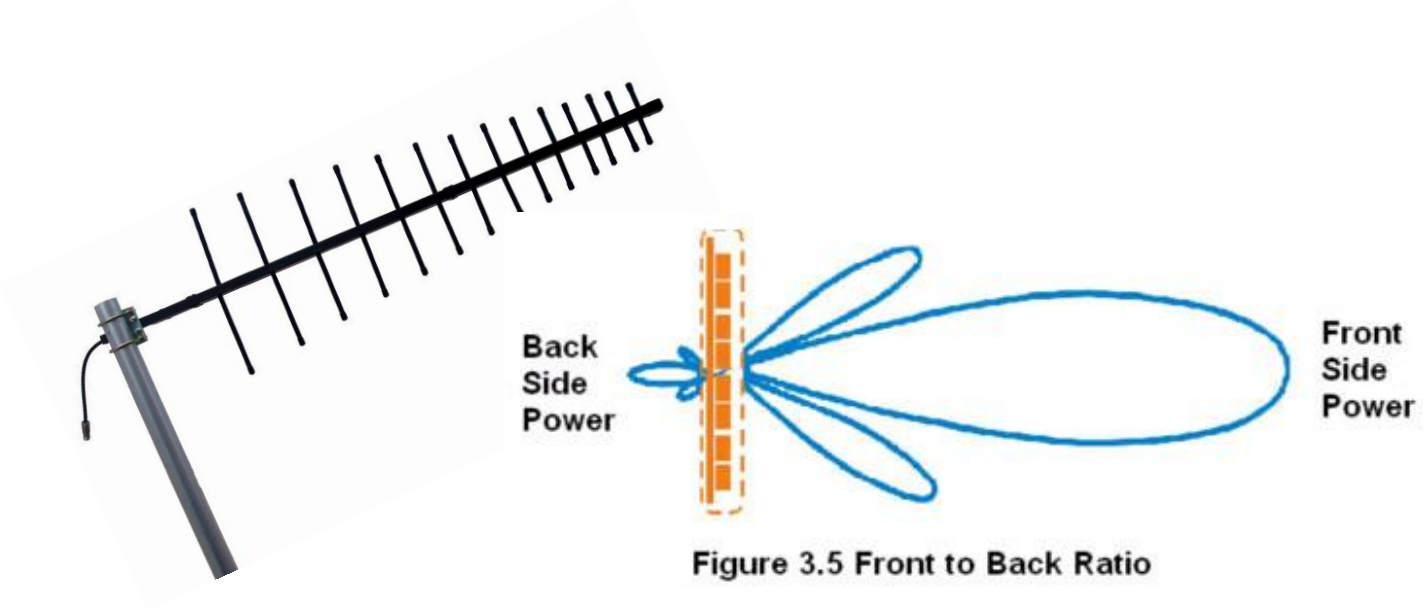
TX Power (dBm)	Value
+14	0x7F
+13	0x7C
+12	0x7A
+11	0x78
+8	0x71
+6	0x6C
+4	0x68
+3	0x66
+2	0x63
+1	0x61
0	0x5F
-3	0x58
-40	**0x41**

How to improve the wireless range:

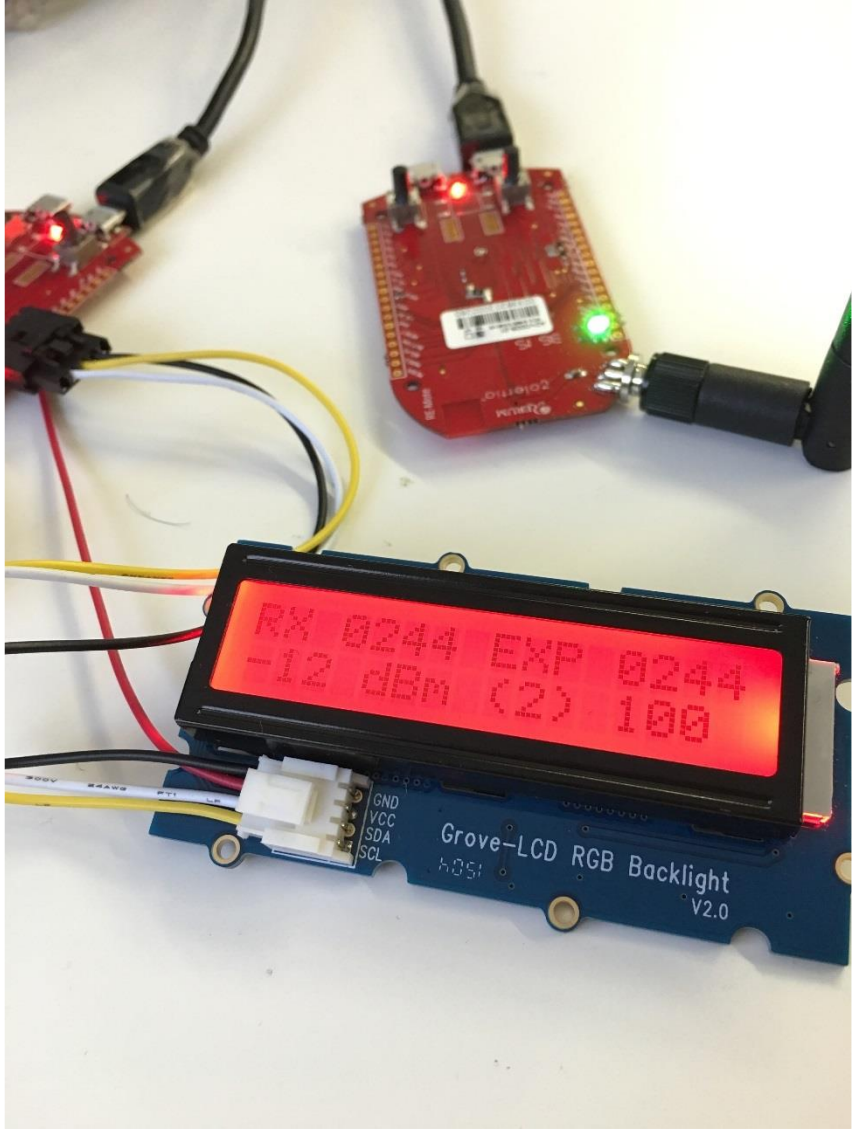
- Increase the transmission power
- Use antennas with higher gain
- Increase antenna's height
- Use directive antennas
- Try to orient the antennas



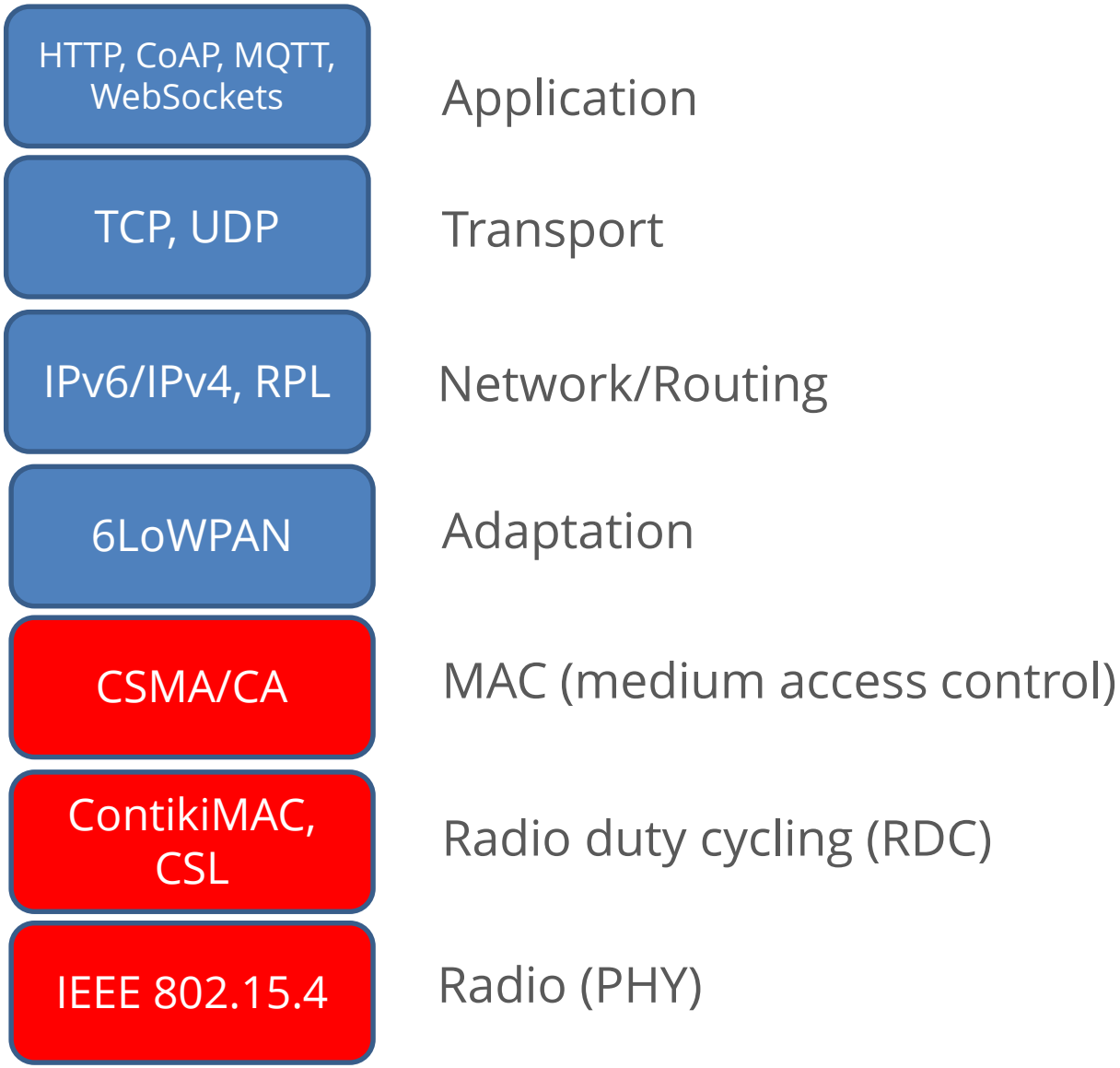
Omnidirectiona Antena 2.4GHz 5dBi “whip”



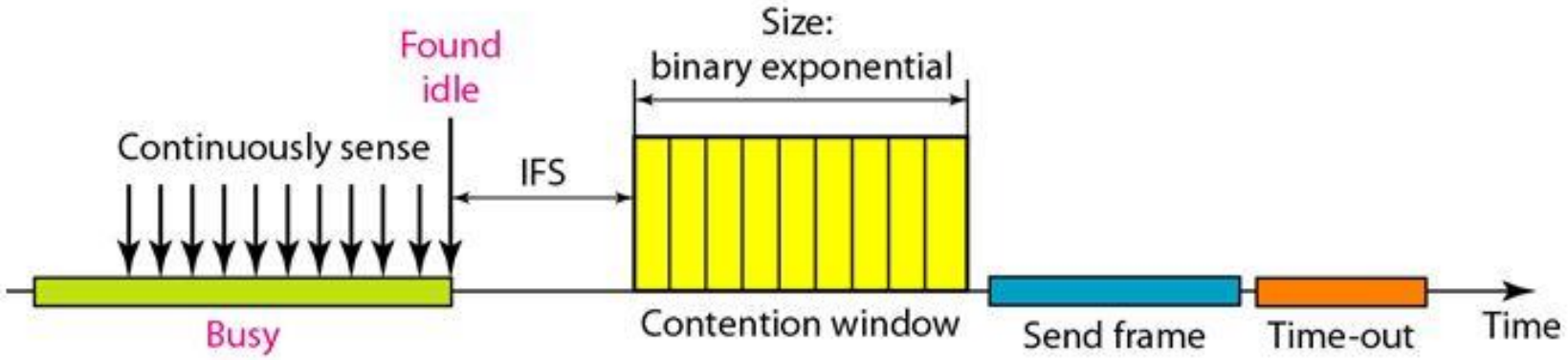
Yagi Directional antenna

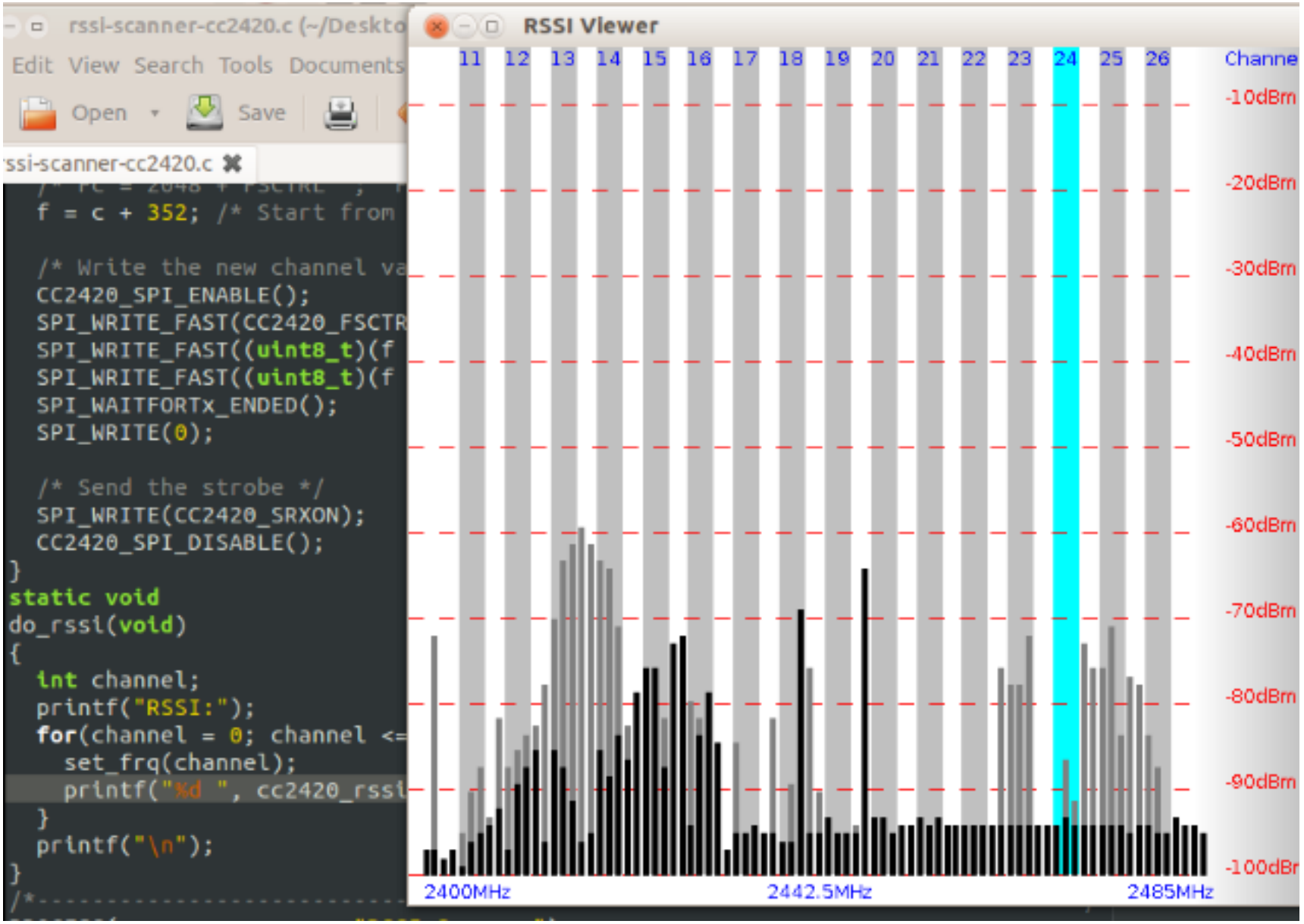


http://hackster.io/zolertia



- CSMA/CA: Carrier Sensing Multiple Access with collision avoidance
- Carrier Sensing: Listen before talking.





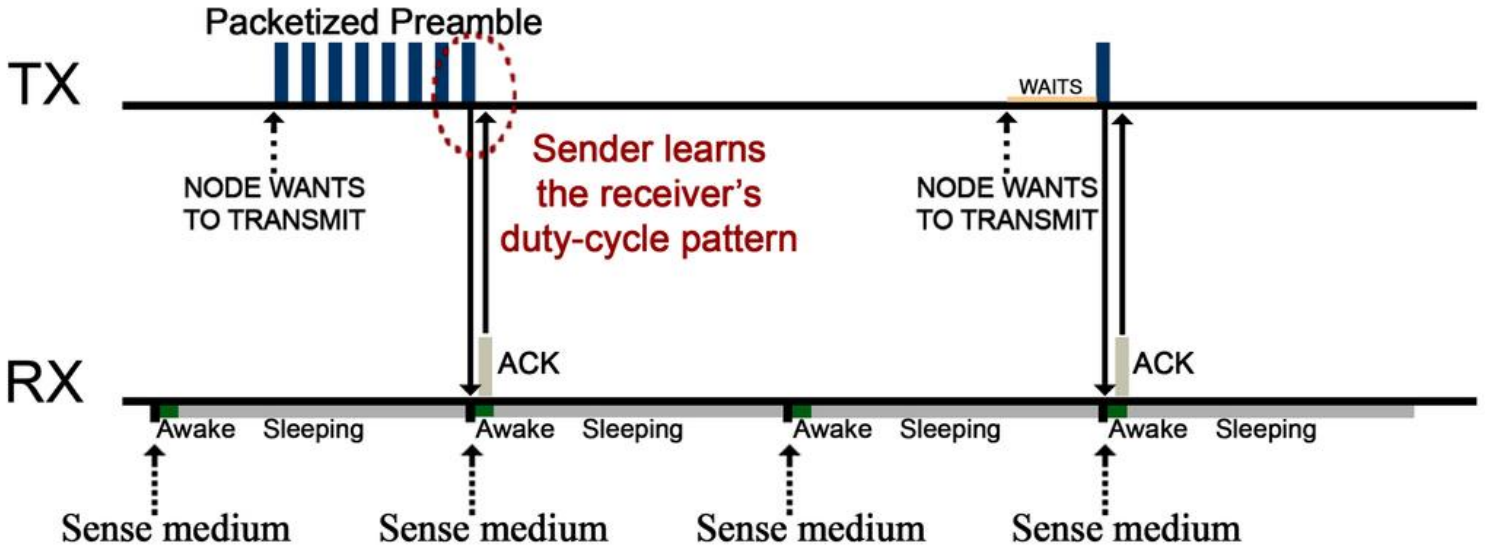
examples/rssi-scanner/rssi-scanner-cc2420.c

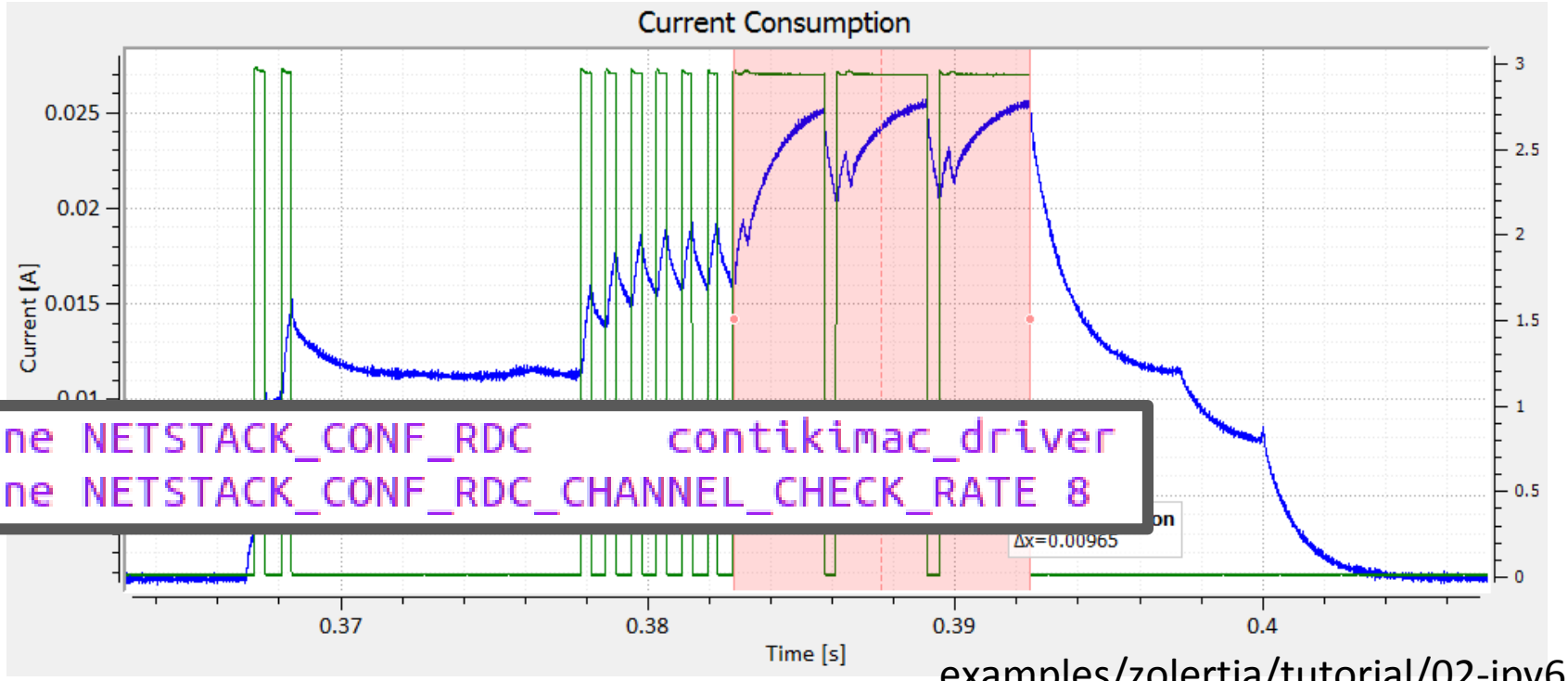
```
#ifndef NETSTACK_CONF_MAC
#define NETSTACK_CONF_MAC      csma_driver
#endif
```

```
#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 */
```

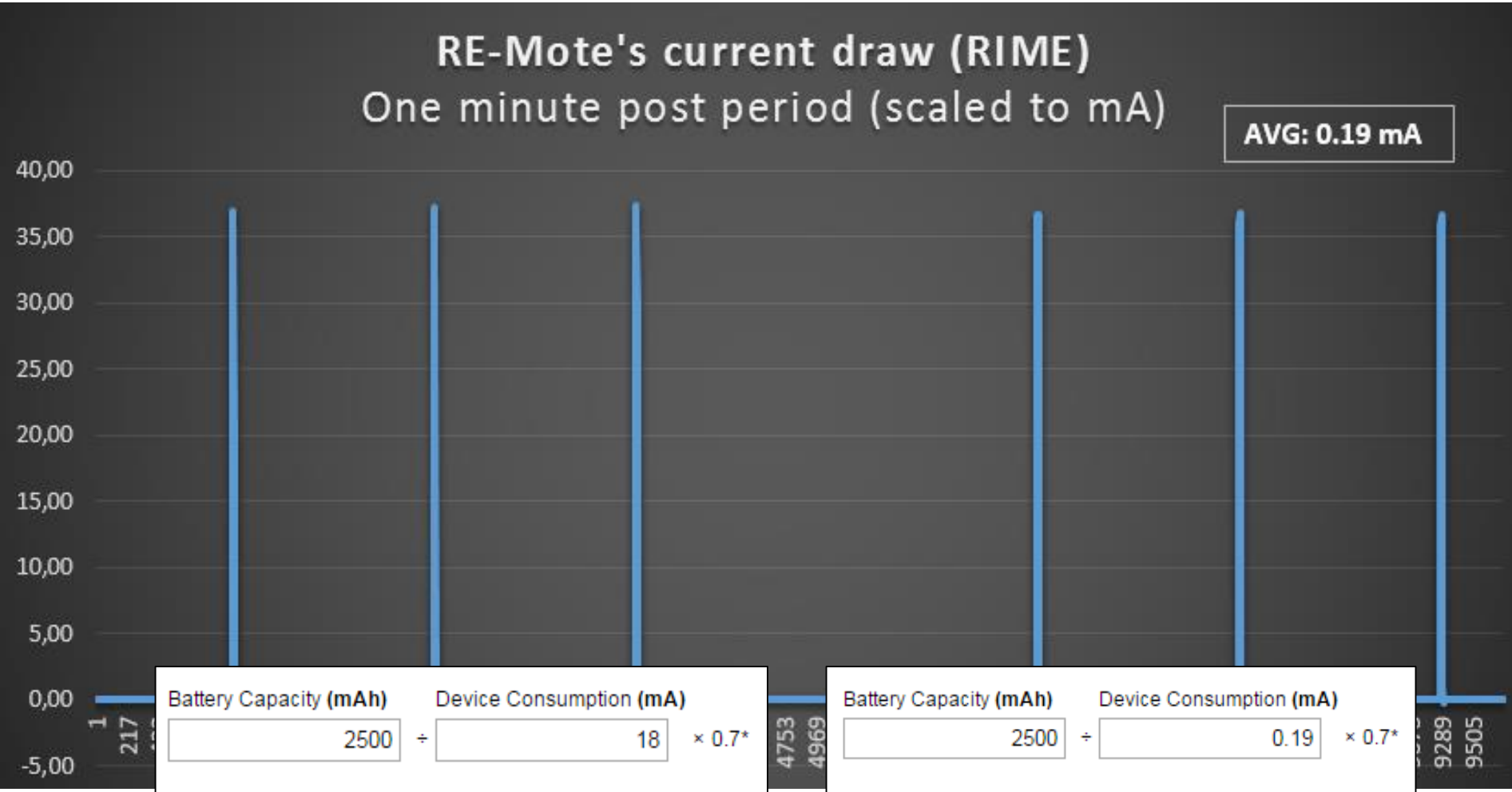
examples/zolertia/tutorial/02-ipv6



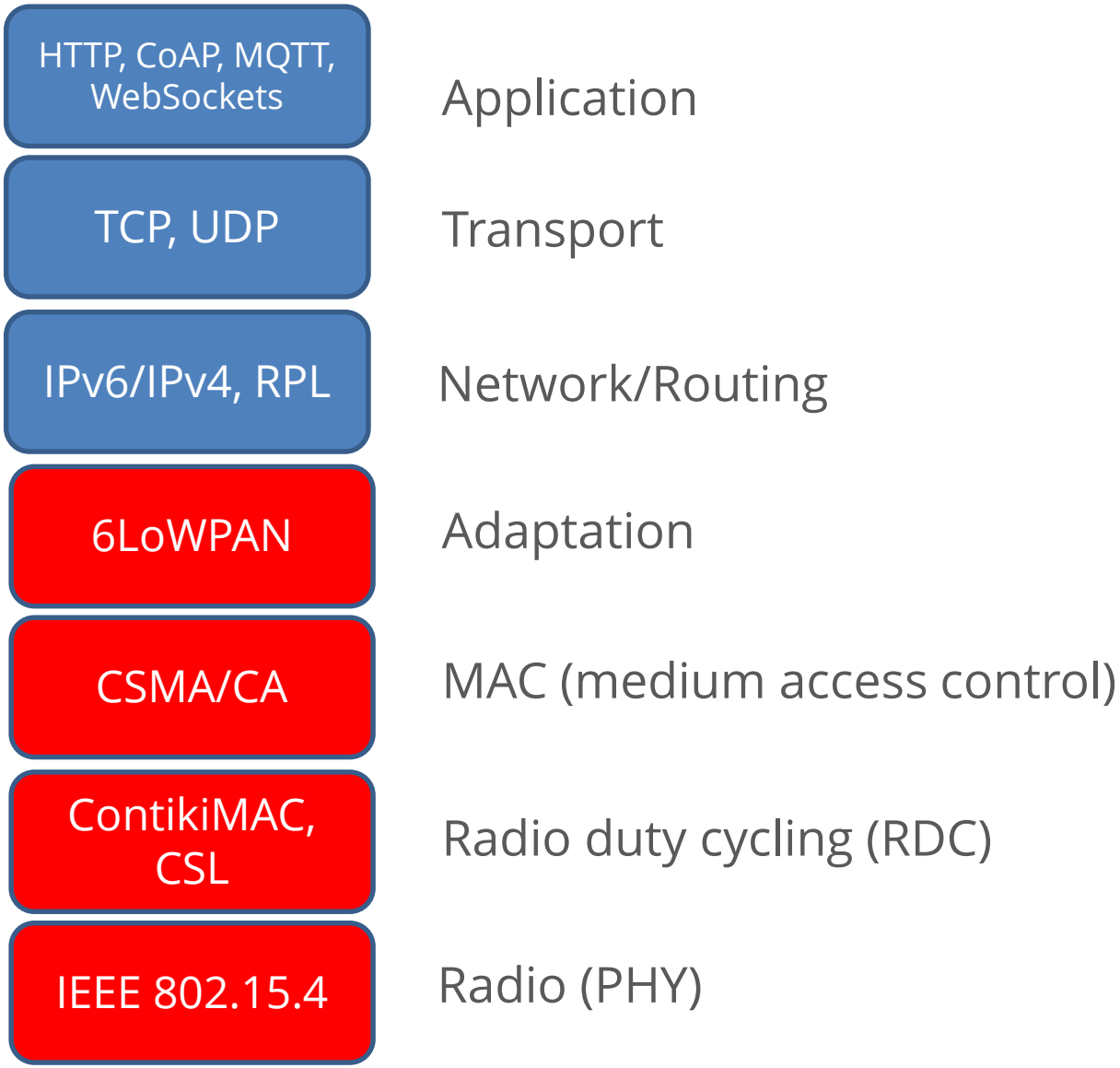



```
#define NETSTACK_CONF_RDC      contikimac_driver
#define NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 8
```

examples/zolertia/tutorial/02-ipv6



http://www.digikey.com/en/resources/conversion-calculators/conversion-calculator-battery-life

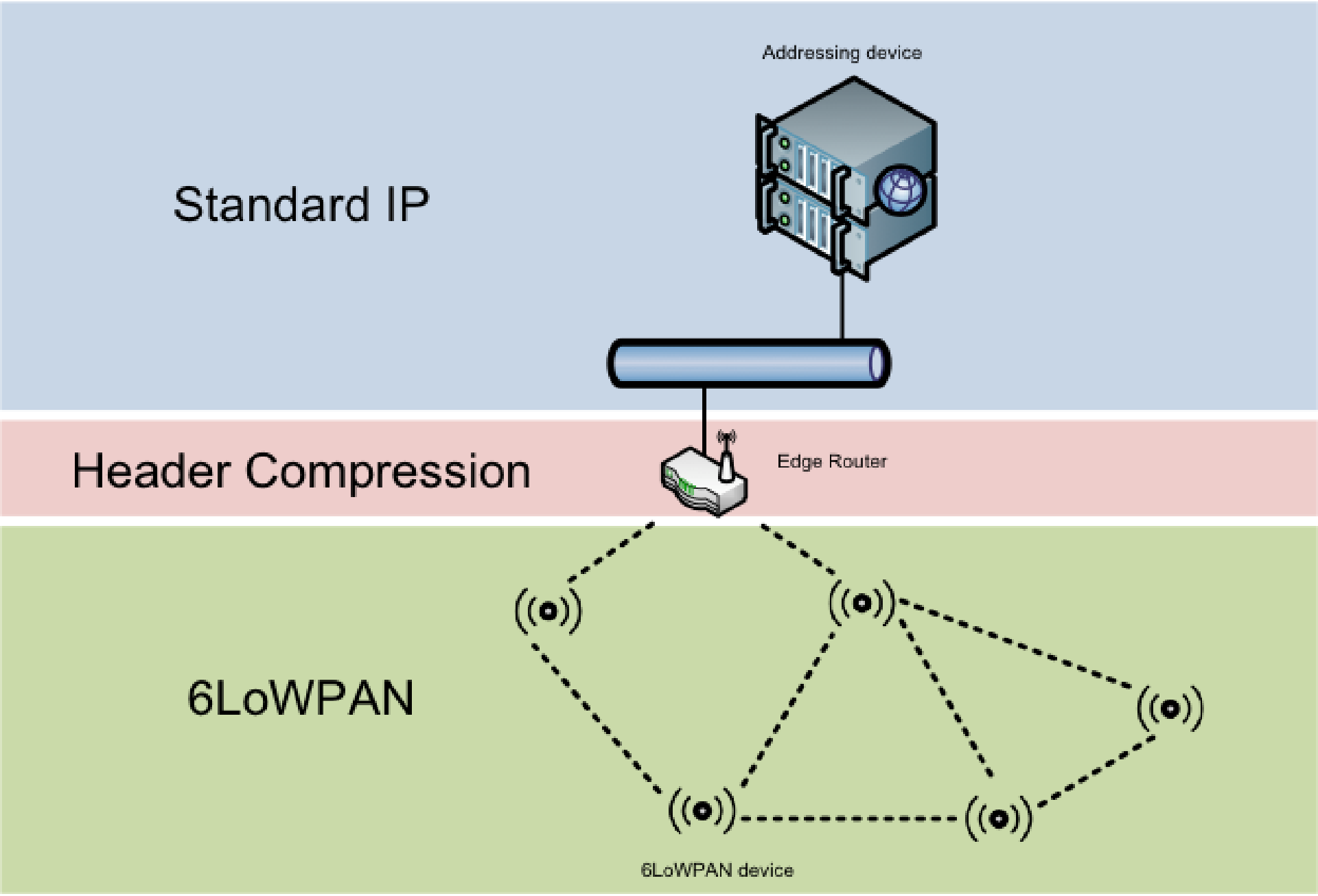


Header Size Calculation

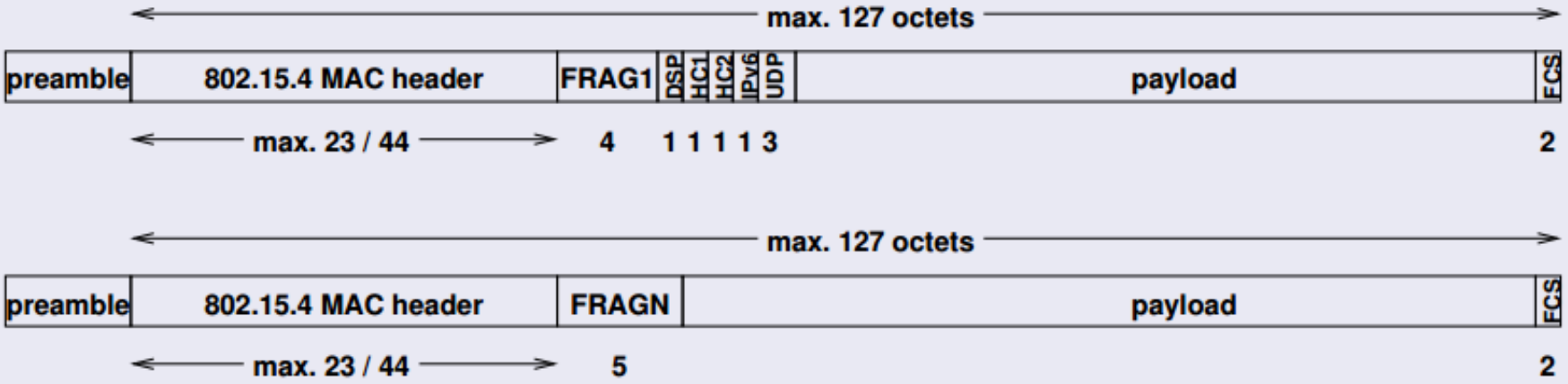
- IPv6 header is 40 octets, UDP header is 8 octets
- 802.15.4 MAC header can be up to 25 octets (null security) or 25+21=46 octets (AES-CCM-128)
- With the 802.15.4 frame size of 127 octets, we have only following space left for application data!
  - 127-25-40-8 = 54 octets (null security)
  - 127-46-40-8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

IPv6 MTU Requirements

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



Fragmentation Example (compressed link-local IPv6/UDP)



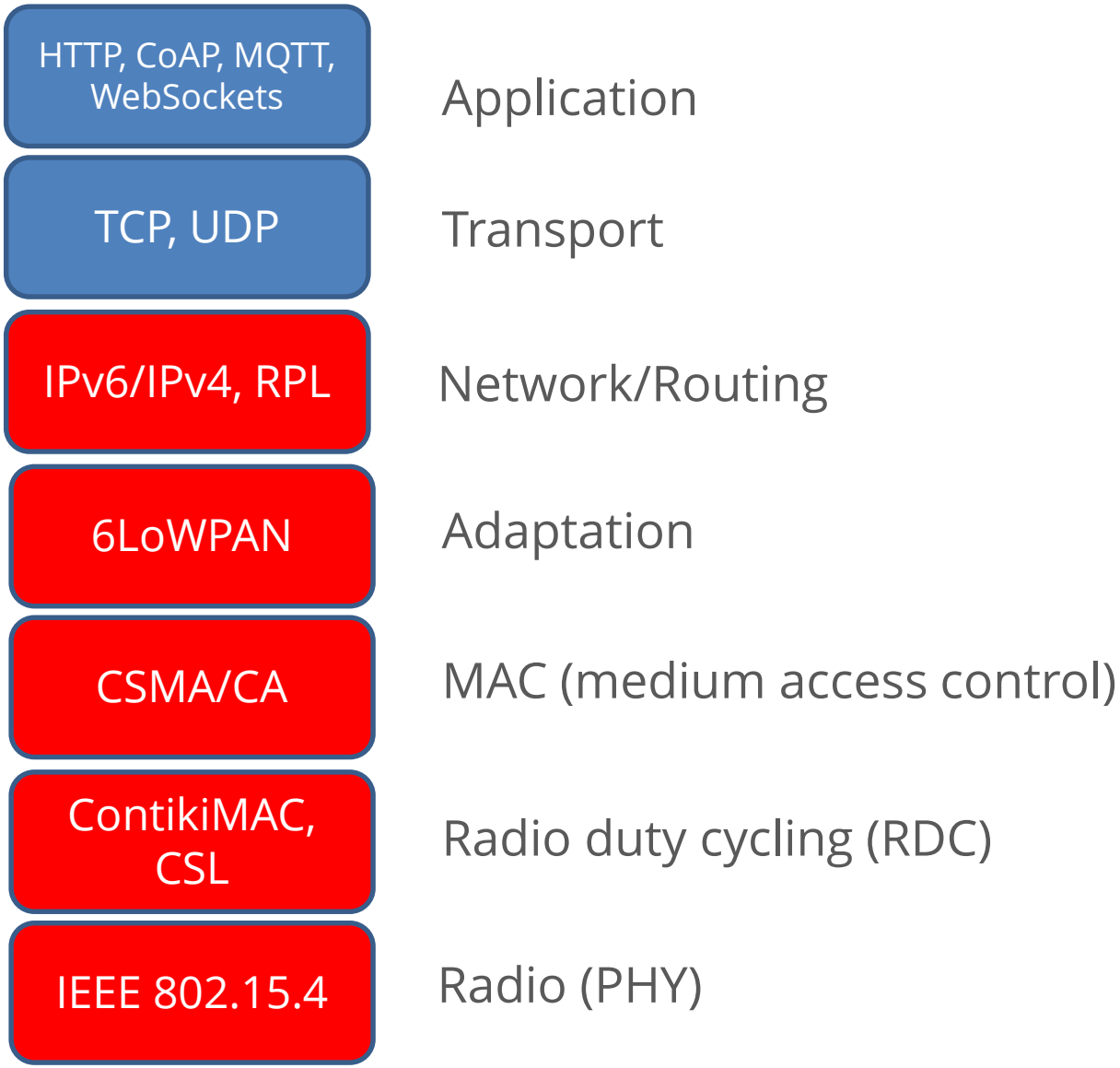
Bit Pattern	Description
01 000001	uncompressed IPv6 addresses
01 000010	HC1 Compressed IPv6 header
01 010000	BC0 Broadcast header
01 111111	Additional Dispatch octet follows
10 xxxxxx	Mesh routing header
11 000xxx	Fragmentation header (first)
11 100xxx	Fragmentation header (subsequent)

```
/* 6lowpan */
#define SICSLOWPAN_CONF_COMPRESSION          SICSLOWPAN_COMPRESSION_HC06
#ifndef SICSLOWPAN_CONF_COMPRESSION_THRESHOLD
#define SICSLOWPAN_CONF_COMPRESSION_THRESHOLD 63
#endif
#ifndef SICSLOWPAN_CONF_FRAG
#define SICSLOWPAN_CONF_FRAG                 1
#endif
#define SICSLOWPAN_CONF_MAXAGE               8

/* Define our IPv6 prefixes/contexts here */
#define SICSLOWPAN_CONF_MAX_ADDR_CONTEXTS    1
#ifndef SICSLOWPAN_CONF_ADDR_CONTEXT_0
#define SICSLOWPAN_CONF_ADDR_CONTEXT_0 { \
  addr_contexts[0].prefix[0] = UIP_DS6_DEFAULT_PREFIX_0; \
  addr_contexts[0].prefix[1] = UIP_DS6_DEFAULT_PREFIX_1; \
}
#endif
```

The default prefix is FD00::

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

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

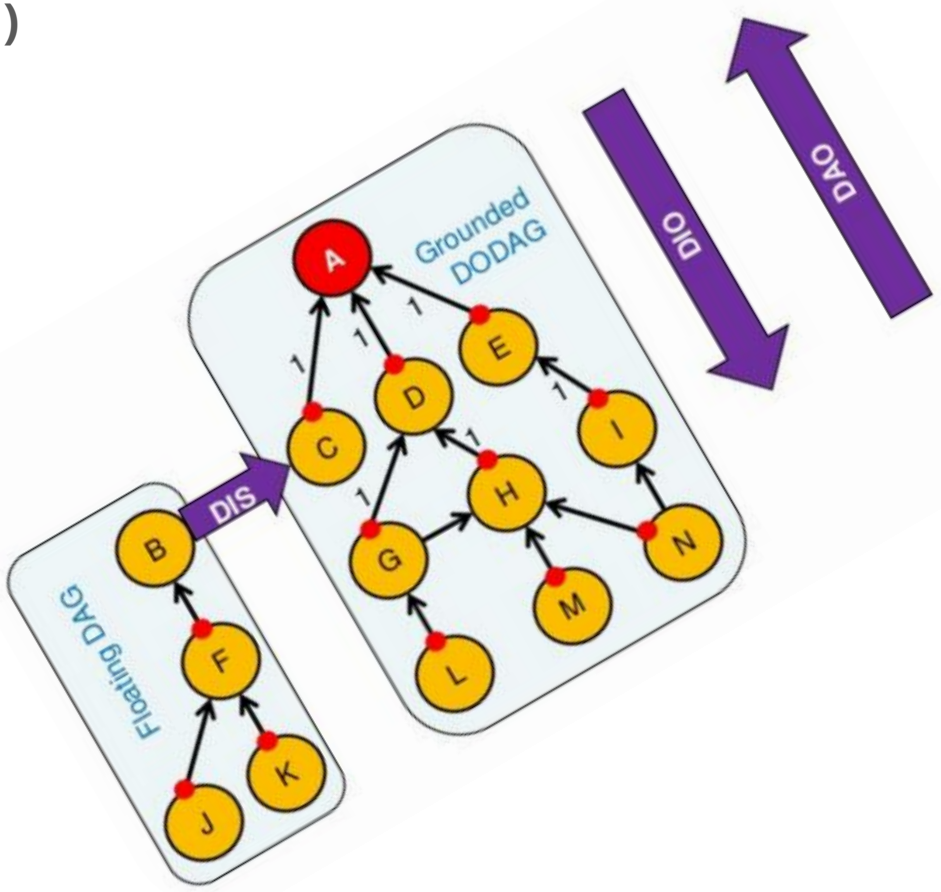


RPL: IPv6 Routing Protocol for Low power and Lossy Networks

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

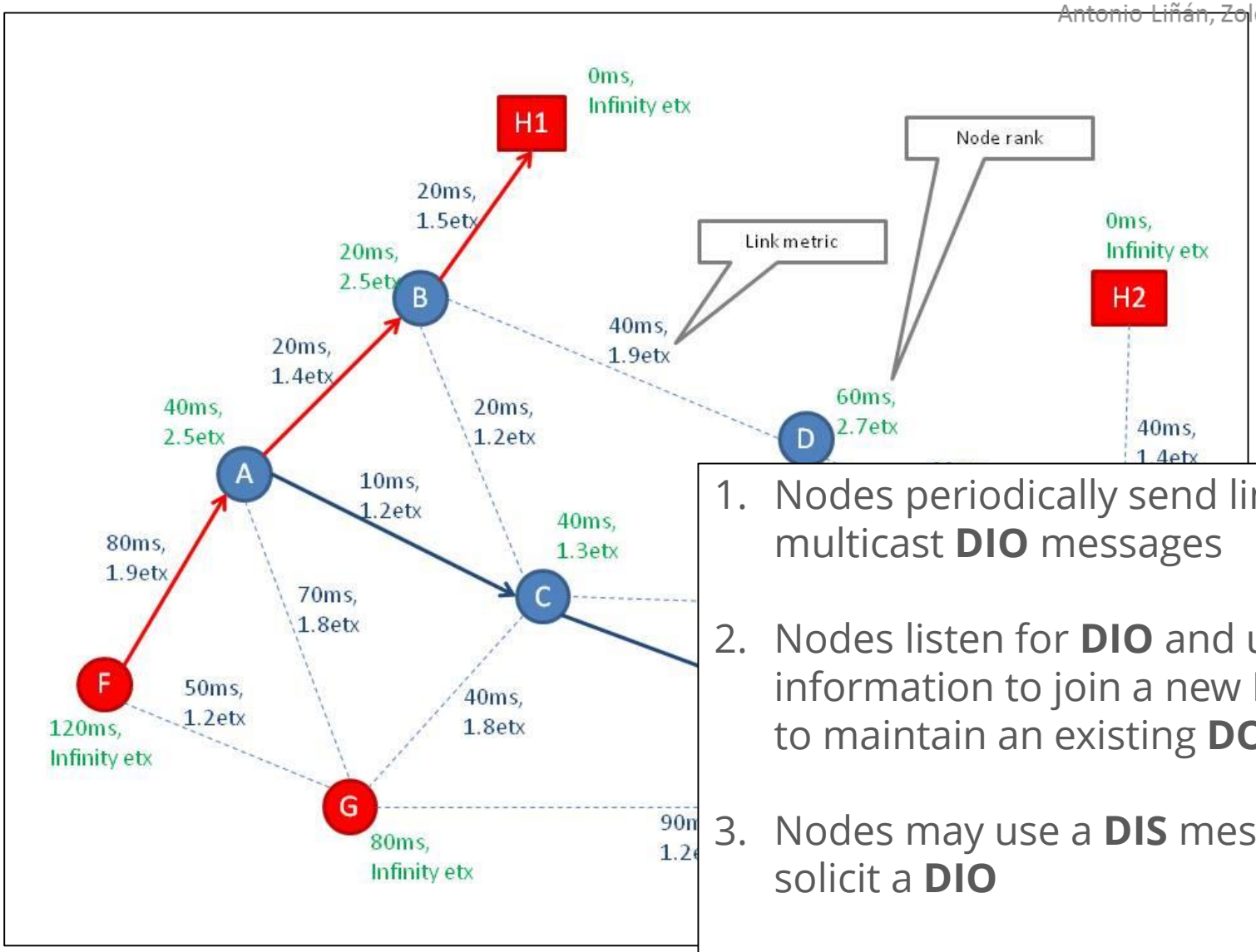
ICMPv6 control messages

- **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

examples/zolertia/tutorial/02-ipv6



1. Nodes periodically send link-local multicast **DIO** messages
2. Nodes listen for **DIO** 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 **DIO** the node chooses parents that minimize path cost to the **DODAG** root

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

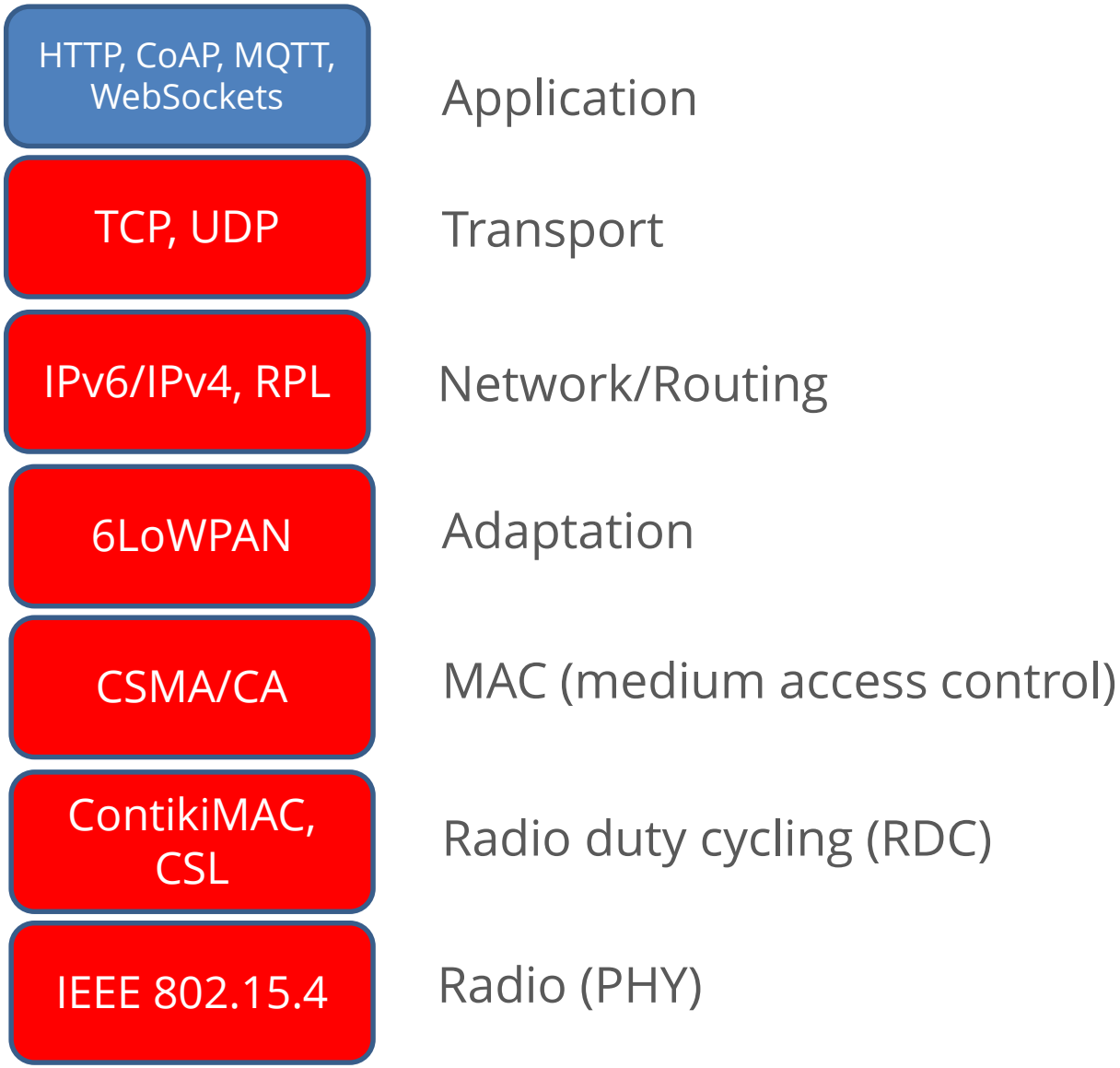
examples/zolertia/tutorial/02-ipv6

```
CONTIKI_WITH_IPV6 = 1
include $(CONTIKI)/Makefile.include
```

examples/zolertia/tutorial/02-ipv6

```
/* ND and Routing */
#ifndef UIP_CONF_ROUTER
#define UIP_CONF_ROUTER                         1
#endif

#define UIP_CONF_ND6_SEND_RA                    0
#define UIP_CONF_IP_FORWARD                     0
#define RPL_CONF_STATS                          0

#ifndef RPL_CONF_OF
#define RPL_CONF_OF rpl_mrhof
#endif

#define UIP_CONF_ND6_REACHABLE_TIME     600000
#define UIP_CONF_ND6_RETRANS_TIMER       10000

#ifndef NBR_TABLE_CONF_MAX_NEIGHBORS
#define NBR_TABLE_CONF_MAX_NEIGHBORS        16
#endif
#ifndef UIP_CONF_MAX_ROUTES
#define UIP_CONF_MAX_ROUTES                 16
#endif
```

platforms/zoul/contiki-conf.h



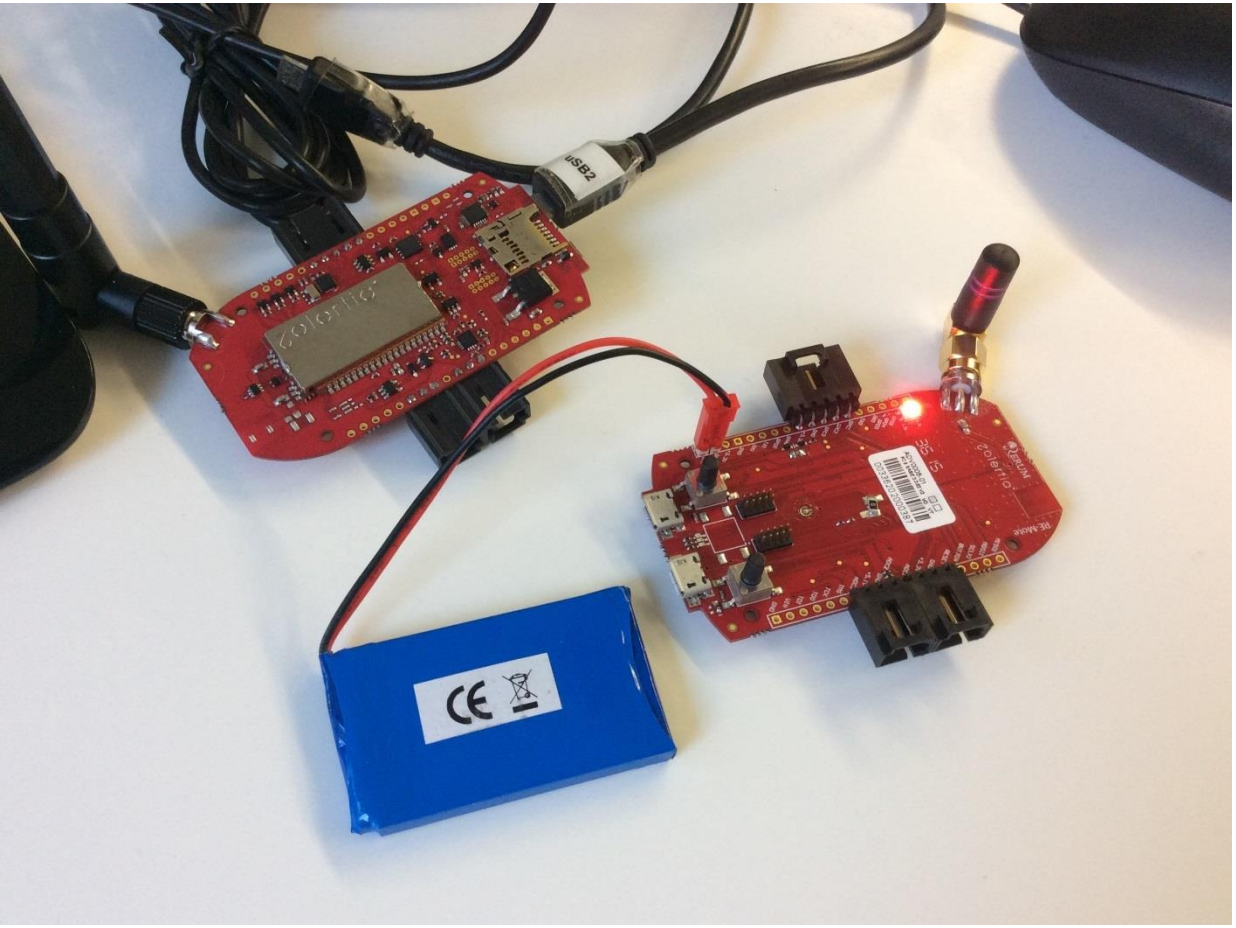
UDP: User Datagram Protocol

- IETF RFC 768
- Minimal, unreliable, best-effort service
- Connectionless
- Best tailored for time-sensitive applications, as dropping packets is preferable to waiting for delayed packets
- Less energy-expensive (than TCP), lesser messages are required to be exchanged
- Used in DNS, DHCP, SNMP, RIP, OpenVPN, etc

UDP Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Length																Checksum															

examples/zolertia/tutorial/02-ipv6



Connect two RE-Motes and program the example:

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

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

```
Contiki-3.x-2614-g19f3dc5
Zolertia RE-Mote platform
CC2538: ID: 0xb964, rev.: PG2.0, Flash: 512 KiB, SRAM: 32 KiB, AES/SHA: 1, ECC/RSA: 1
System clock: 16000000 Hz
I/O clock: 16000000 Hz
Reset cause: External reset
Rime configured with address 00:12:4b:00:06:15:ab:25
 Net: sicslowpan
 MAC: CSMA
 RDC: nullrdc

* Radio parameters:
   Channel 15 (Min: 11, Max: 26)
   Tx Power   3 dBm (Min: -24 dBm, Max:   7 dBm)
   PAN ID: 0xABCD

***
Sending packet to multicast adddress ff02::1
ID: 171, core temp: 36.190, ADC1: 2328, ADC2: 0, ADC3: 1496, batt: 3300, counter: 1

***
Message from: fe80::212:4b00:616:fb9
Data received on port 8765 from port 8765 with length 14
CH: 15 RSSI: -71dBm LQI: 107
ID: 171, core temp: 35.0, ADC1: 2288, ADC2: 0, ADC3: 1452, batt: 3280, counter: 1
```

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

```
Contiki-3.x-2614-g19f3dc5
Zolertia RE-Mote platform
CC2538: ID: 0xb964, rev.: PG2.0, Flash: 512 KiB, SRAM: 32 KiB, AES/SHA: 1, ECC/RSA: 1
System clock: 16000000 Hz
I/O clock: 16000000 Hz
Reset cause: External reset
Rime configured with address 00:12:4b:00:06:15:ab:25
 Net: sicslowpan
 MAC: CSMA
```

```
/**
 * \brief Location of the IEEE address
 * 0 => Read from InfoPage,
 * 1 => Use a hardcoded address, configured by IEEE_ADDR_CONF_ADDRESS
 */
#ifndef IEEE_ADDR_CONF_HARDCODED
#define IEEE_ADDR_CONF_HARDCODED              0
#endif

/**
 * \brief The hardcoded IEEE address to be used when IEEE_ADDR_CONF_HARDCODED
 * is defined as 1
 */
#ifndef IEEE_ADDR_CONF_ADDRESS
#define IEEE_ADDR_CONF_ADDRESS { 0x00, 0x12, 0x4B, 0x00, 0x89, 0xAB, 0xCD, 0xEF }
#endif
```

RE-Motes have 2 factory MAC addresses stored in its flash memory, it can be overriden

platforms/zoul/contiki-conf.h

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

```
Contiki-3.x-2614-g19f3dc5
Zolertia RE-Mote platform
CC2538: ID: 0xb964, rev.: PG2.0, Flash: 512 KiB, SRAM: 32 KiB, AES/SHA: 1, ECC/RSA: 1
System clock: 16000000 Hz
I/O clock: 16000000 Hz
Reset cause: External reset
Rime configured with address 00:12:4b:00:06:15:ab:25
 Net: sicslowpan
 MAC: CSMA
 RDC: nullrdc
```

```
/* Comment this out to use Radio Duty Cycle (RDC) and save energy */
#undef NETSTACK_CONF_RDC
#define NETSTACK_CONF_RDC          nullrdc_driver
```

***

Sicslowpan (6LoWPAN implementation), CSMA and ContikiMAC (RDC) are enabled as default.

NullRDC leaves the radio on, increasing the throughput at the cost of energy consumption. Recommended for testing only, or if the devices are not running on batteries

examples/zolertia/tutorial/02-ipv6/project-conf.h
examples/zolertia/tutorial/02-ipv6/01-udp-local-multicast

Contiki-3.x-2614-g19f3dc5

```
/*---------------------------------------------------------------------------*/
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);
}
```

```
* Radio parameters:
   Channel 15 (Min: 11, Max: 26)
   Tx Power   3 dBm (Min: -24 dBm, Max:   7 dBm)
   PAN ID: 0xABCD
```

***

```
/* Radio values to be configured for the 01-udp-local-multicast example */
#if CONTIKI_TARGET_ZOUL
#define EXAMPLE_TX_POWER  0xFF
#else /* default is Z1 */
#define EXAMPLE_TX_POWER  31
#endif
#define EXAMPLE_CHANNEL   15
#define EXAMPLE_PANID     0xBEEF
```

examples/zolertia/tutorial/02-ipv6/example.h
examples/zolertia/tutorial/02-ipv6/01-udp-local-multicast

Octets: 2	1	0/2	0/2/8	0/2	0/2/8	0/5/6/10/14	variable	2
Frame Control	Sequence Number	Destination PAN Identifier	Destination Address	Source PAN Identifier	Source Address	Auxiliary Security Header	Frame Payload	FCS
		Addressing fields						
MHR							MAC Payload	MFR

```
* Radio parameters:
   Channel 15 (Min: 11, Max: 26)
   Tx Power   3 dBm (Min: -24 dBm, Max:   7 dBm)
   PAN ID: 0xABCD
```

PAN identifiers (PANID) allows the radio to filter out messages addressed to different PANs, allowing networks to share the same channel

examples/zolertia/tutorial/02-ipv6/example.h
examples/zolertia/tutorial/02-ipv6/01-udp-local-multicast

```c
static void
print_radio_values(void)
{
  radio_value_t aux;

  printf("\n* Radio parameters:\n");

  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);
}
```

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

```
Contiki-3.x-2614-g19f3dc5
Zolertia RE-Mote platform
CC2538: ID: 0xb964, rev.: PG2.0, Flash: 512 KiB, SRAM: 32 KiB, AES/SHA: 1, ECC/RSA: 1
System clock: 16000000 Hz
I/O clock: 16000000 Hz
Reset cause: External reset
Rime configured with address 00:12:4b:00:06:15:ab:25
 Net: sicslowpan
 MAC: CSMA
 RDC: nullrdc

* Radio parameters:
   Channel 15 (Min: 11, Max: 26)
   Tx Power   3 dBm (Min: -24 dBm, Max:   7 dBm)
   PAN ID: 0xABCD

***
Sending packet to multicast adddress ff02::1
ID: 171, core temp: 36.190, ADC1: 2328, ADC2: 0, ADC3: 1496, batt: 3300, counter: 1

***
Message from: fe80::212:4b00:616:fb9
Data received on port 8765 from port 8765 with length 14
CH: 15 RSSI: -71dBm LQI: 107
ID: 171, core temp: 35.0, ADC1: 2288, ADC2: 0, ADC3: 1452, batt: 3280, counter: 1
```

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

```c
#include "contiki.h"

/* The following libraries add IP/IPv6 support */
#include "net/ip/uip.h"
#include "net/ipv6/uip-ds6.h"

/* This is quite handy, allows to print IPv6 related stuff in a readable way */
#include "net/ip/uip-debug.h"

/* The simple UDP library API */
#include "simple-udp.h"

/* Library used to read the metadata in the packets */
#include "net/packetbuf.h"

/* And we are including the example.h with the example configuration */
#include "../example.h"
```

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

Structures to be used in the example: UDP and custom one for payload data

```
/* The structure used in the Simple UDP library to create an UDP connection */
static struct simple_udp_connection mcast_connection;
/*---------------------------------------------------------------------------*/
/* Create a structure and pointer to store the data to be sent as payload */
static struct my_msg_t msg;
static struct my_msg_t *msgPtr = &msg;
```

The my_msg_t structure content

```
/* This data structure is used to store the packet content (payload) */
struct my_msg_t {
  uint8_t  id;          An ID type if required
  uint16_t counter;     Number of messages sent
  uint16_t value1;      The core temperature will be stored here
  uint16_t value2;      The ADC1 will be stored here
  uint16_t value3;      The ADC2 will be stored here
  uint16_t value4;      The ADC3 will be stored here
  uint16_t battery;     The voltage level of the RE-Mote
};
```

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

```
/**
 * \brief      Register a UDP connection
 * \param c    A pointer to a struct simple_udp_connection
 * \param local_port The local UDP port in host byte order
 * \param remote_addr The remote IP address
 * \param remote_port The remote UDP port in host byte order
 * \param receive_callback A pointer to a function of to be called for incoming
 * \retval 0   If no UDP connection could be allocated
 * \retval 1   If the connection was successfully allocated
 *
 *     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.
 *
 */
int simple_udp_register(struct simple_udp_connection *c,
                        uint16_t local_port,
                        uip_ipaddr_t *remote_addr,
                        uint16_t remote_port,
                        simple_udp_callback receive_callback);
```

```
/* Create the UDP
 * attaches a cal
 * called for inc
 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

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

```
/* 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/example.h
examples/zolertia/tutorial/02-ipv6/01-udp-local-multicast

```
/* Create the UDP co
 * attaches a callba
 * called for incomi
 that an ephemeral U
 * be NULL, to indic
 */
simple_udp_register(
 receiver);
```

```
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);

  printf("\nData received on port %d from port %d with length %d\n",
         receiver_port, sender_port, datalen);

  /* The following functions are the functions provided by the RF library to
   * retrieve information about the channel number we are on, the RSSI (received
   * strenght signal indication), and LQI (link quality information)
   */

  NETSTACK_RADIO.get_value(RADIO_PARAM_CHANNEL, &aux);
  printf("CH: %u ", (unsigned int) aux);

  aux = packetbuf_attr(PACKETBUF_ATTR_RSSI);
  printf("RSSI: %ddBm ", (int8_t)aux);

  aux = packetbuf_attr(PACKETBUF_ATTR_LINK_QUALITY);
  printf("LQI: %u\n", aux);
```

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

```
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);
```

```
Message from: fe80::212:4b00:616:fb9
Data received on port 8765 from port 8765 with length 14
CH: 15 RSSI: -71dBm LQI: 107
ID: 171, core temp: 35.0, ADC1: 2288, ADC2: 0, ADC3: 1452, batt: 3280, counter: 1
```

```
  NETSTACK_RADIO.get_value(RADIO_PARAM_CHANNEL, &aux);
  printf("CH: %u ", (unsigned int) aux);

  aux = packetbuf_attr(PACKETBUF_ATTR_RSSI);
  printf("RSSI: %ddBm ", (int8_t)aux);

  aux = packetbuf_attr(PACKETBUF_ATTR_LINK_QUALITY);
  printf("LQI: %u\n", aux);
```

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

multicast adddress ff02::1

```
etimer_set(&periodic_timer, SEND_INTERVAL);

while(1) {
  PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&periodic_timer));
  /* 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

```c
static void
take_readings(void)
{
  uint32_t aux;
  counter++;

  msg.id      = 0xAB;
  msg.counter = counter;

#if CONTIKI_TARGET_ZOUL
  msg.value1  = cc2538_temp_sensor.value(CC2538_SENSORS_VALUE_TYPE_CONVERTED);
  msg.value2  = adc_zoul.value(ZOUL_SENSORS_ADC1);
  msg.value3  = adc_zoul.value(ZOUL_SENSORS_ADC2);
  msg.value4  = adc_zoul.value(ZOUL_SENSORS_ADC3);

  aux = vdd3_sensor.value(CC2538_SENSORS_VALUE_TYPE_CONVERTED);
  msg.battery = (uint16_t) aux;

  /* Print the sensor data */
  printf("ID: %u, core temp: %u.%u, ADC1: %d, ADC2: %d, ADC3: %d, batt: %u, counter: %u\n",
         msg.id, msg.value1 / 1000, msg.value1 % 1000, msg.value2, msg.value3,
         msg.value4, msg.battery, msg.counter);
```

```c
  /* 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

```
/**
 * \brief      Send a UDP packet to a specified IP address
 * \param c    A pointer to a struct simple_udp_connection
 * \param data A pointer to the data to be sent
 * \param datalen The length of the data
 * \param to   The IP address of the receiver
 *
 *     This function sends a UDP packet to a specified IP
 *     address. The packet will be sent with the UDP ports
 *     that were specified when the connection was registered
 *     with simple_udp_register().
 *
 * \sa simple_udp_send()
 */
int simple_udp_sendto(struct simple_udp_connection *c,
                      const void *data, uint16_t datalen,
                      const uip_ipaddr_t *to);
```

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

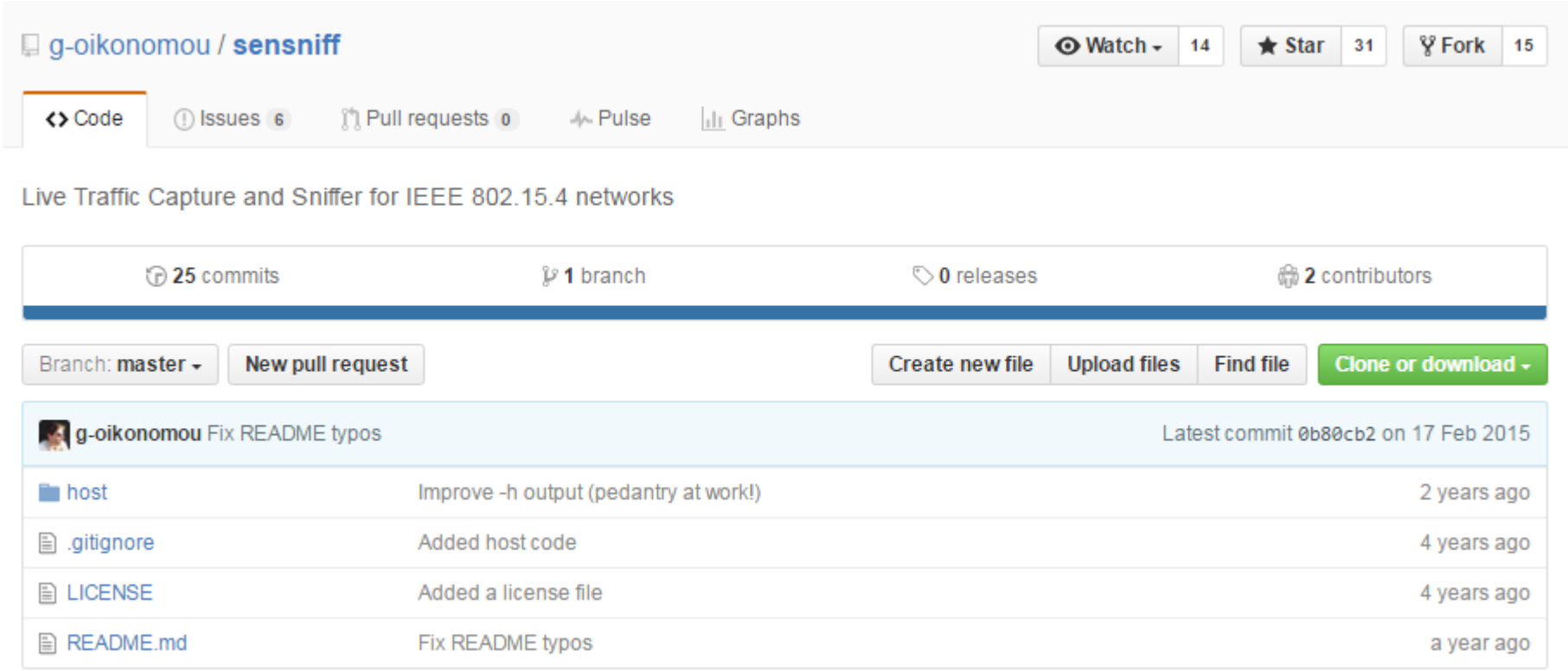
examples/zolertia/tutorial/02-ipv6/01-udp-local-multicast

Experiment with the Radio settings and wireless link:

- How does the RSSI/LQI change if moving the devices apart, or putting an obstable in between?
- Rotate the RE-Mote's so the antenna position is different, in which position yields better results in terms of RSSI?
- Try to decrease the transmission power to the minimum, and move the devices until they no longer hear each other.
- Increase the transmission power to the maximum and move the devices apart, what is the maximum distance? What would happen if you increase the height of one of the devices by 1 meter?
- Write down the addresses of your devices and your neighbors, try to identify the other groups devices

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

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



https://github.com/g-oikonomou/sensniff

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

Check the channel and PAN ID matches your application

```
#undef IEEE802154_CONF_PANID
#define IEEE802154_CONF_PANID  0xABCD

#if CONTIKI_TARGET_ZOUL

/* The following are Zoul (RE-Mote, etc) specific */
#undef CC2538_RF_CONF_CHANNEL
#define CC2538_RF_CONF_CHANNEL     26
```

examples/zolertia/tutorial/02-ipv6/03-sniffer/project-conf.h

Program a RE-Mote as sniffer (change the USB port accordingly)

```
make sniffer.upload PORT=/dev/ttyUSB0
```

Execute the Sensniff application

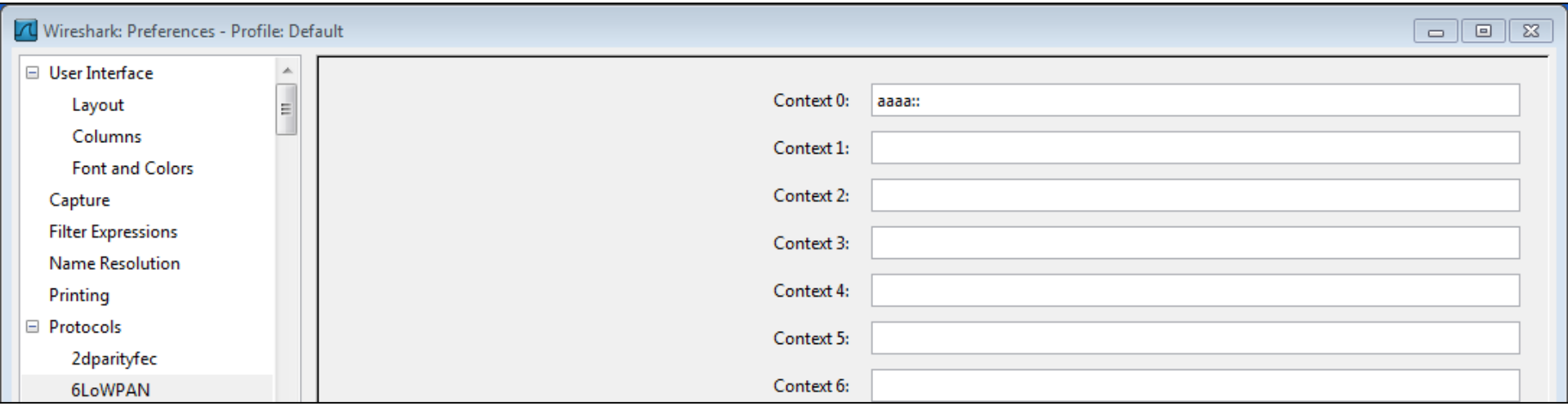
```
python sensniff.py --non-interactive -d /dev/ttyUSB0 -b 460800 –p test.pcap
```

In another terminal run:

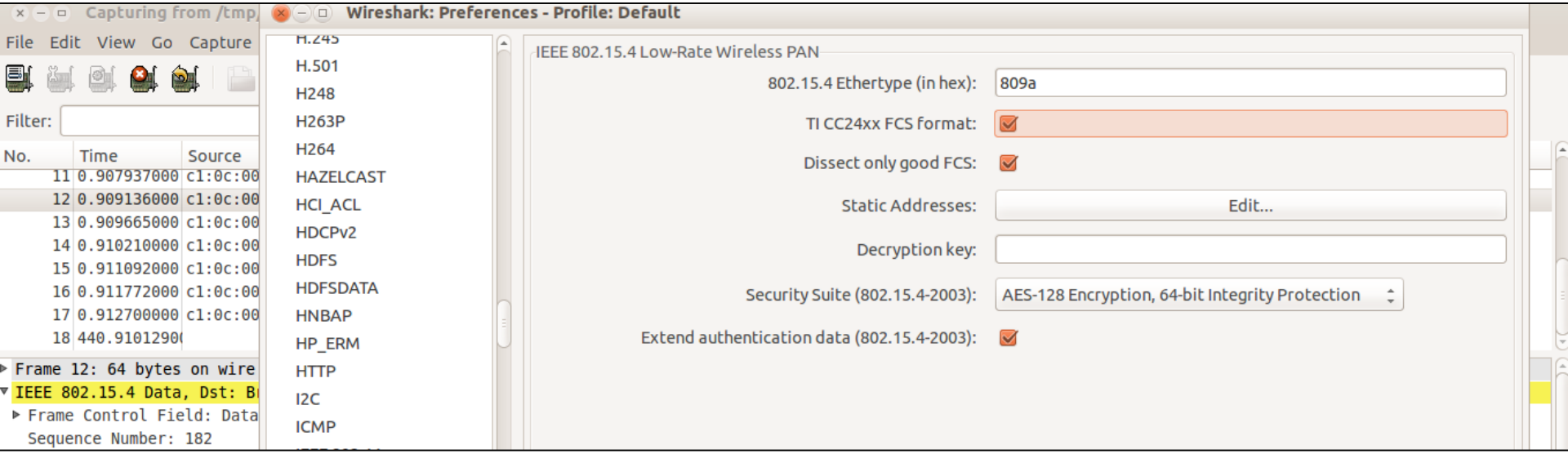
```
sudo wireshark -i /tmp/sensniff
```

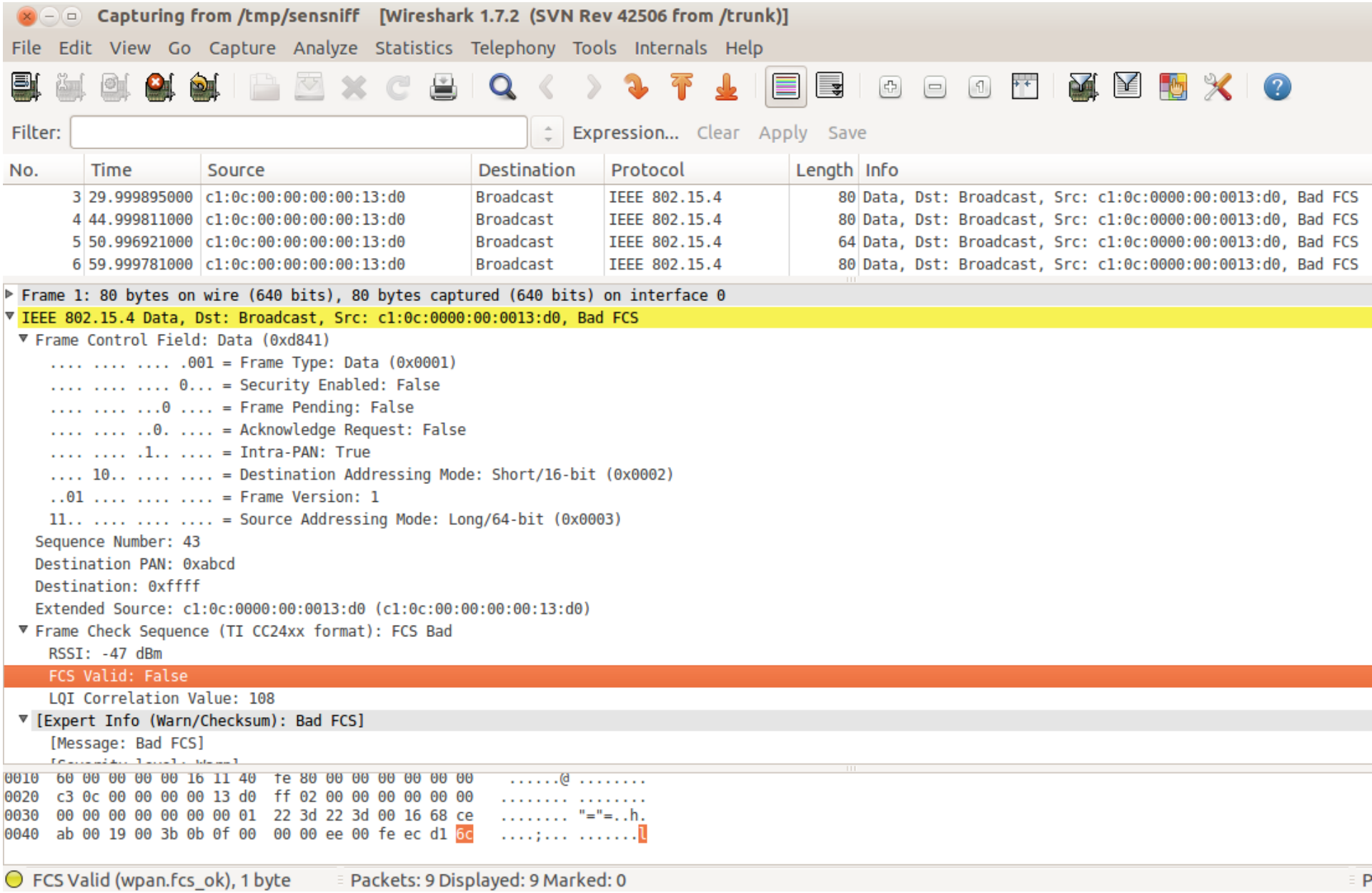
examples/zolertia/tutorial/02-ipv6/03-sniffer

Configure the context, default in Contiki are “aaaa::” and “fd00::”



Configure the frame format as shown:





Experiment with the radio sniffer:

- Try to filter the sniffed packets and only visualize your own
- Try to identify the RPL control messages from the wireshark capture, what types of messages can you spot?
- Expand the protocol's tabs and check the headers and its fields, search for the PANID and other fields like the RSSI and LQI metadata.

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

Conclusions

You should be able to:

- Understand how low power radio devices work and how to use link quality estimators
- Understand concepts like RDC, MAC and RPL networking basics
- Understand the basics about 6LoWPAN
- Change parameters like the transmission power, channel and PAN ID
- Create link-local networks and send sensor data wirelessly over UDP
- Use a Wireless packet sniffer to debug your application

Antonio Liñán Colina

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

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