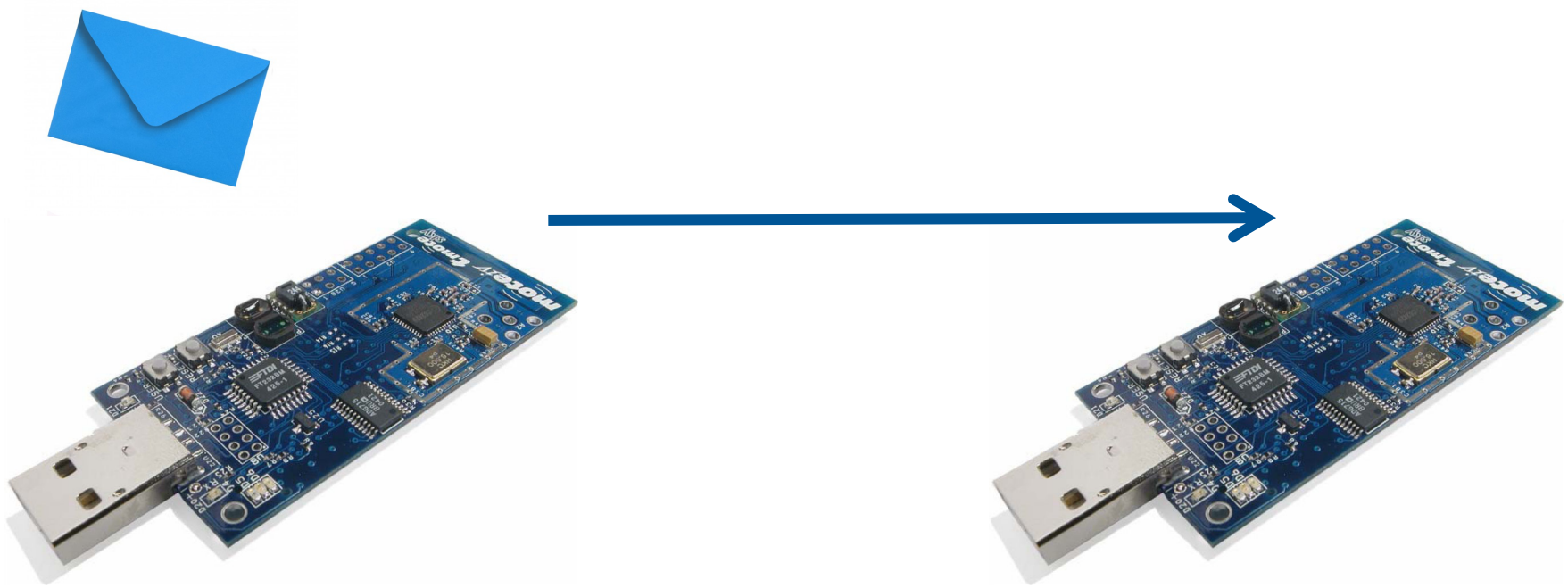
Communication

Ramona Marfievici
(ramona.marfievici@cit.ie)

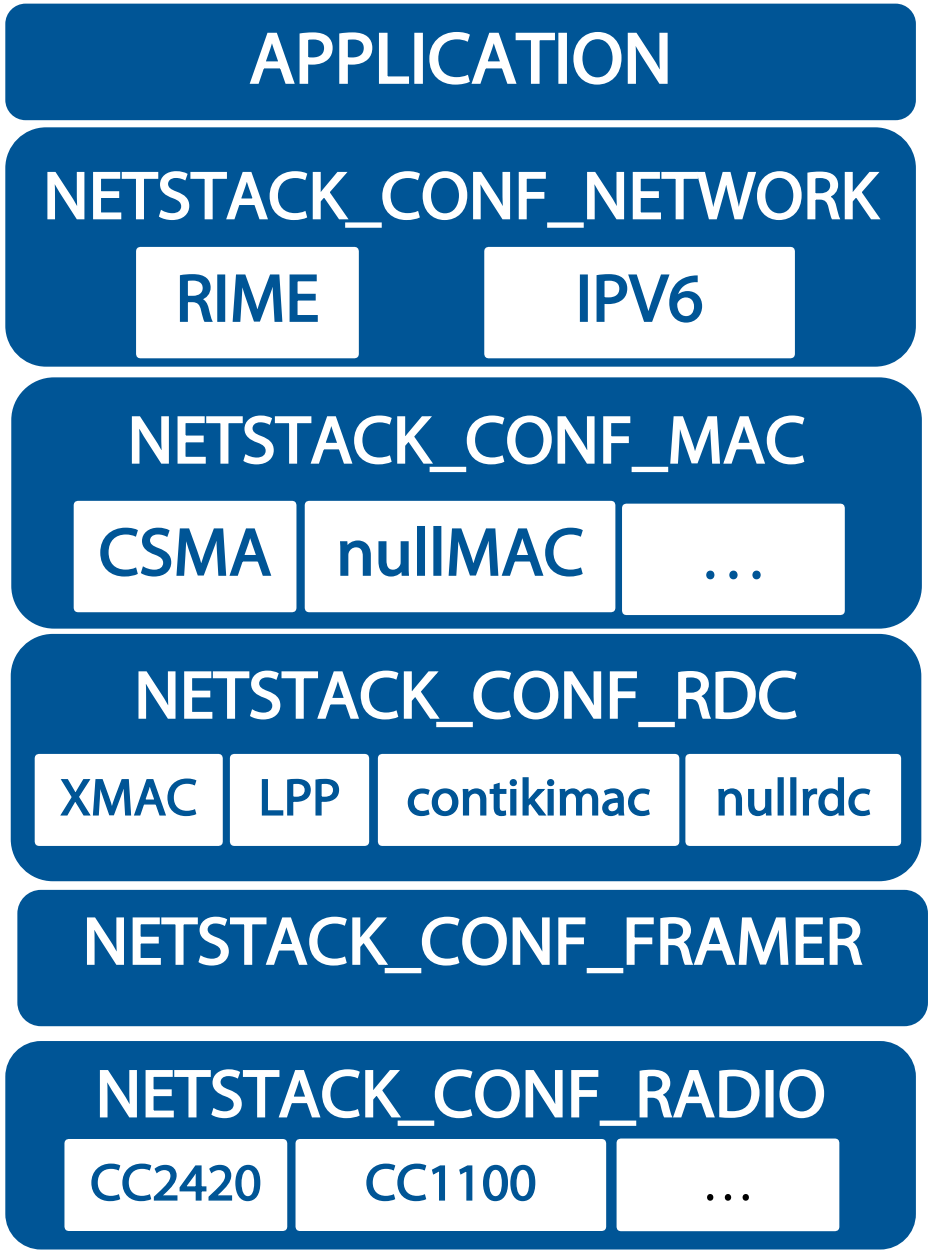
Node to node communication



Ingredients

- Network stack
- Packet buffer
- Addresses

Contiki network stack



Defined in `core/net/netstack.h`

Directly in the `Makefile`

```
CONTIKI_WITH_IPv6 = 1
CONTIKI_WITH_RIME = 1
```

Addressing and retransmission of lost packets

Radio Duty-Cycle
Takes care of the sleep period of nodes

Not a regular layer, a collection of auxiliary functions called before transmitting a packet or after reception

Contiki network stack

If no definitions are made, by default ContikiOS stack layers will be

- **Network layer**: rime_driver
- **MAC layer**: nullmac_driver
- **RDC**: nullrdc_driver
- **Framer**: framer_nullmac
- **Radio**: nullradio_driver

Stack layers usually defined in *contiki-conf.h*
There is one *contiki-conf.h* file for each platform (e.g., *platform/sky*)

TMoteSky default stack: *csma_driver*, *contikimac_driver*, *framer_802154* and *cc2420_driver*

Changing the protocols

```
DEFINES = NETSTACK_CONF_RDC=cxmac_driver,NETSTACK_CONF_MAC=null_mac
```

Contiki Rime communication primitives

reliable unicast, bulk

reliable unicast

stubborn unicast

unicast

reliable broadcast, bulk

reliable broadcast

stubborn broadcast

identified broadcast

anonymous broadcast

network flooding, bulk

network flooding

Detour…Packet buffer

- Structure used to create an outbound packet or store a inbound packet
- Also used to operate on a packet and it can store only one packet at a time
- A single buffer for the network stack, represents the buffer in the radio chip
- Interface in `core/net/packetbuf.h`
- Whatever is in the buffer when *_send is called, gets sent

- Copying into the buffer
  - `packetbuf_copyfrom(const void *from, uint16_t length;`
    - **copies data from the memory location pointed by `from` into the packetbuf**
    - **if data is larger than the packetbuf => only data that fits**
    - **number of bytes that could be copied is returned**
  - `void *packetbuf_dataptr();`
    - **more flexible, get pointer to data section, process as needed**

- Reading the buffer
  - `packetbuf_copyto(void *to)` **or, using the** `packetbuf_dataptr`

Detour…Addresses

Rime addresses are of type `linkaddr_t`

- Data type and interface defined in `core/net/linkaddr.h`
- In COOJA, `LINKADDR_SIZE = 2`, so address is 2 bytes
- TMoteSky, **`LINKADDR_SIZE = 8`**, so address is 8 bytes

Contiki has other addresses as well

What is my address?

- `make login` + RESET button
- `"Rime started with address …"`

Easiest way to send data:

- `linkaddr_t receiver;`
- `receiver[0] = <1st byte>; receiver[1] = <2nd byte>;`
  `// need to know the full address`
- `unicast_send(&conn, &receiver);`

Rime broadcast

core/net/rime/broadcast.h

```
struct broadcast_callbacks {
  void (* recv)(struct broadcast_conn *ptr, const linkaddr_t *sender);
  void (* sent)(struct broadcast_conn *ptr, int status, int num_tx);
};
```

- called when a packet has been received/sent by the broadcast module
- parses a packet and displays the message and the address of sender
- `broadcast_conn *`: structure with 2 structures
- `linkaddr_t`: rime address

```
struct broadcast_conn {
struct abc_conn c;
const struct broadcast_callbacks *u;
};
```

- `abc` module sends packets to all local area neighbors
- `broacast_callbacks struct`: called when a pkt has been received by the broadcast module

Rime broadcast cont.

```
# sets up a broadcast connection on the specified channel
void broadcast_open(struct broadcast_conn *c, uint16_t channel,
                    const struct broadcast_callbacks *u);
```

- the caller allocates memory for the `struct broadcast_conn` by declaring it as a static variable
- `struct_broadcast_callbacks` pointer points to a structure containing a pointer to a function that will be called when a packet arrives on the channel
- the function opens a connection of type `abc_conn` and sets the callbacks to structure passed
- `channel`: connection will operate on this channel (<128 reserved by the system)

```
#close  broadcast connection
void broadcast_close(struct broadcast_conn *c);

#send a packet
int broadcast_send(struct broadcast_conn *c);
```

```
#include "net/rime/rime.h"
PROCESS(example_broadcast_process, "Broadcast example");
AUTOSTART_PROCESSES(&example_broadcast_process);
static void broadcast_recv(struct broadcast_conn *c, const linkaddr_t *from)
{
       printf("broadcast message received from %d.%d: '%s'\n",
              from->u8[0], from->u8[1], (char *)packetbuf_dataptr());
}
static const struct broadcast_callbacks broadcast_call = {broadcast_recv};
static struct broadcast_conn broadcast;

PROCESS_THREAD(example_broadcast_process, ev, data)
       {
         static struct etimer et;
         PROCESS_EXITHANDLER(broadcast_close(&broadcast);)
         PROCESS_BEGIN();
         broadcast_open(&broadcast, 129, &broadcast_call);
         while(1) {
           etimer_set(&et, CLOCK_SECOND * 4);
           PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
           packetbuf_copyfrom("Hello", 6);
           broadcast_send(&broadcast);
           printf("broadcast message sent\n");
         }
         PROCESS_END();}
```

Exercises

1. Run the example-broadcast from `examples/rime` in COOJA.
2. Change the example-broadcast to send your name. Test the program on your mote.

Note. Remember to account for the "null character" associated with character strings in C when changing the number of characters.

3. Write a program to broadcast your name and the temperature of your node every 5 seconds.