femto-st SCIENCES & TECHNOLOGIES

Modular Robots: Blinky Blocks Networking

Frédéric Lassabe

cnrs
UBFC UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ
UNIVERSITÉ DE FRANCHE-COMTÉ
SUP MICRO TECH ENSMM
utbm université de technologie Belfort-Montbéliard

Networking for Programmable Matter

- Programmable matter
  - Self-reconfiguration
  - moving robots from location A to location B
  - knowing locations
  - avoiding bad configurations (blocking, impossible moves)
  - No global knowledge
  - Exchange data through networking
- Blinky Block : 6 UART interface
  - 4 layers
    - Serial
    - sublayer 1 – 2
    - layer 2 : frames
    - layer 3 : packets
  - Layer 3 exposed to developpers
    - one layer 3 implementation for each application
    - User layer 3 defined for each application

Knowing connections

- A function to know an interface's status
- `is_connected(unit8_t interface_index)`
- Returns
  - 1 if there is a peer on the other side of the connector (i.e., there is a neighbor)
  - 0 if there is no peer, i.e. there is no neighbor
- Useful to avoid waiting for a response from an unexisting neighbor

Packet structure

Functions

Application protocol

Fields

- A packet identifier
  - used for packets buffer management
- io_port
  - receiving interface (inbound packets)
  - sending interface (outbound packets)
- packet_content
  - user defined payload
- functions
  - pointers to packet management functions
  - not modified by the user
  - callback functions for packet processing
  - Callbacks are called **out of BBloop** !

Packet structure

Functions

Application protocol

Functions to transmit

- `send_message` sends a message on one interface
- `send_delayed_message` sends a message on one interface after a delay
- `send_multicast_message` sends a message to a subset of BB's interfaces
- `send_delayed_multicast_message` sends a message to a subset of BB's interfaces after a delay
- `send_broadcast_message` sends a message to all interfaces
- `send_delayed_broadcast_message` sends a message to all interfaces after a delay

Functions to receive

- `process_standard_packet`
  - callback function called when receiving a packet
- `process_standard_unack`
  - called when a packet is not acknoledged after 3 retransmissions
- `process_standard_ack`
  - called when a packet was acknowledged by its destination

send_message

- Sends a message to one interface. Requires 4 parameters :
  - target interface (`uart_index`), the number of the interface
  - data : an array `uint8_t` (any data can be cast into this array)
  - size : the length of the data array
  - `has_ack` : 1 if you require an acknowledgment of your packet, 0 else
- Example (send a 32 bits unsigned value to the top connector with ack requested)

```
uint32_t value = 0xabcd0123;
send_message(MY_TOP, (uint8_t *) &value, sizeof(uint32_t), 1);
```

send_delayed_message

- Sends a message to one interface, after a delay
- Takes the same first 4 parameters as `send_message`, and adds a delay in ms.
- Delay is a `uint16_t` value
- Example : send a message after 1 second :

```
uint32_t value = 0xabcd0123;
send_delayed_message(MY_TOP, (uint8_t *) &value, \
                     sizeof(uint32_t), 1, 1000);
```

send_multicast_message

- Sends a message to a subset of the BB's interfaces
- Uses a mask to select interfaces (bit number is equal to interface number)
- Returns a mask of the actual neighbors it sent to (based on `is_connected` function)
- Example : send a message to TOP and BOTTOM connectors

```
uint8_t iface_mask = (1 << MY_TOP) | (1 << MY_BOTTOM);
char value[] = "hello";
uint8_t sent_to = \
  send_multicast_message(value, 6, iface_mask, 1);
```

send_delayed_multicast_message

- Sends a message to a subset of the BB's interfaces, after a delay
- Takes the same first 4 parameters as `send_multicast_message`, and adds a delay in ms.
- Delay is a `uint16_t` value
- Example : sends a message after 2 seconds to LEFT and RIGHT :

```
uint8_t iface_mask = (1 << MY_LEFT) | (1 << MY_RIGHT);
char value[] = "hello";
uint8_t sent_to = \
  send_delayed_multicast_message(value, 6, iface_mask, 1, 2000);
```

send_broadcast_message

- Sends a message to all interfaces
- Only uses 3 parameters (destination is not required)
- Returns a mask of the actual neighbors it sent to
- Useful for flooding
- Example : send a value to every neighbor :

```
uint8_t my_distance = 0;
uint8_t neighbors;
neighbors = send_broadcast_message(&my_distance, \
                                   sizeof(uint8_t), 1);
```

send_delayed_broadcast_message

- Sends a message to all interfaces, after a delay
- Takes the same first 3 parameters as `send_broadcast_message`, and adds a delay in ms.
- Delay is a `uint16_t` value
- Example : sends a a value to every neighbor after 1 second :

```
uint8_t my_distance = 0;
uint8_t neighbors;
neighbors = send_delayed_broadcast_message(&my_distance, \
                                           sizeof(uint8_t), 1, 1000);
```

Packet structure

Functions

Application protocol

Why Protocols ?

- When designing an application with network communication :
  - Usually, several messages types
  - With different payloads and meanings
  - How to distinguish messages ?
- Protocol
  - Messages types
  - Messages identification
  - Rules for messages processing

Case study

- To illustrate the next slides
- Application example :
  - A leader sends each second a message to all BBs :
    - One of two messages : change color
    - The other message : play a tone
  - BBs shall avoid looping messages in the ensemble

Messages definition

- most effective way : define structs with your messages data
- For our example :

```
typedef struct { // or typedef struct __attribute__((__packed__)
        uint8_t message_type; // Will be 0
        uint16_t sequence_number; // don't propagate twice
        uint8_t red;
        uint8_t green;
        uint8_t blue;
} color_msg_t;

typedef struct { // or typedef struct __attribute__((__packed__)
        uint8_t message_type; // Will be 1
        uint16_t sequence_number; // don't propagate twice
        uint16_t frequency;
        uint16_t duration;
} sound_msg_t;
```

Use structures

```
uint32_t clk = 0;
void BBloop() {
  static int action = 0; // 0 -> LED, 1 -> sound
  static uint16_t sequence = 0;
  if (HAL_GetTick() - 1000 > clk) {
    clk = HAL_GetTick();
    if (action) {
      sound_msg_t msg = {
        .message_type=1, .frequency=440,
        .duration=900, .sequence=sequence,
      };
      send_broadcast_message((uint8_t *) &msg, sizeof(sound_msg_t), 1);
    } else {
      // init r,g,b based on seq or any other changing values
      color_msg_t msg = {
        .message_type=0, .red=r, .green=g,
        .blue=b, .sequence=sequence,
      };
      send_broadcast_message((uint8_t *) &msg, sizeof(color_msg_t), 1);
    }
    ++sequence;
    action = 1 - action; // swap between 1 and 0
  }
}
```

Processing messages

```
uint8_t process_standard_packet(L3_packet *packet) {
  static int current_sequence = -1;
  if (packet->packet_content[0] == 0) {
    color_msg_t *pkt = (color_msg_t *) packet->packet_content;
    if (pkt->sequence > current_sequence) {
      set_RGB(pkt->red, pkt->green, pkt->blue);
      send_broadcast_message(packet->packet_content, \
                             sizeof(color_msg_t), 1);
      current_sequence = pkt->sequence;
    }
  } else if (packet->packet_content[0] == 1) {
    sound_msg_t *pkt = (sound_msg_t *) packet->packet_content;
    if (pkt->sequence > current_sequence) {
      make_sound(pkt->frequency, pkt->duration);
      send_broadcast_message(packet->packet_content, \
                             sizeof(sound_msg_t), 1);
      current_sequence = pkt->sequence;
    }
  }
  return 0;
}
```

Messages transmission

- Structs can be transmitted "as is"
  - BBs all have the same architecture
  - All have the same code
- Communication with other devices
  - Pay attention to architecture !
    - Big Endian vs Little Endian
    - LE : most computers
    - BE : network order

Thanks !