Microbit Network Protocol - MNP

I - Introduction to microbit

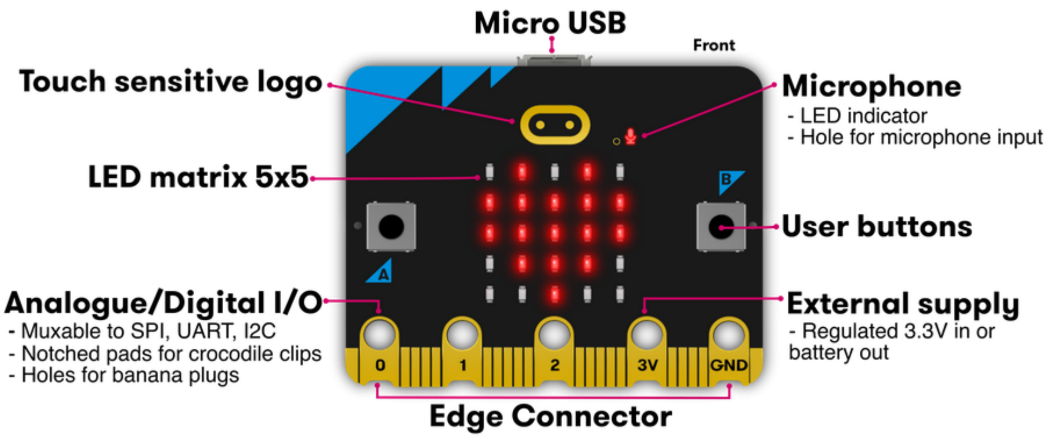
1) About the BBC micro :bit

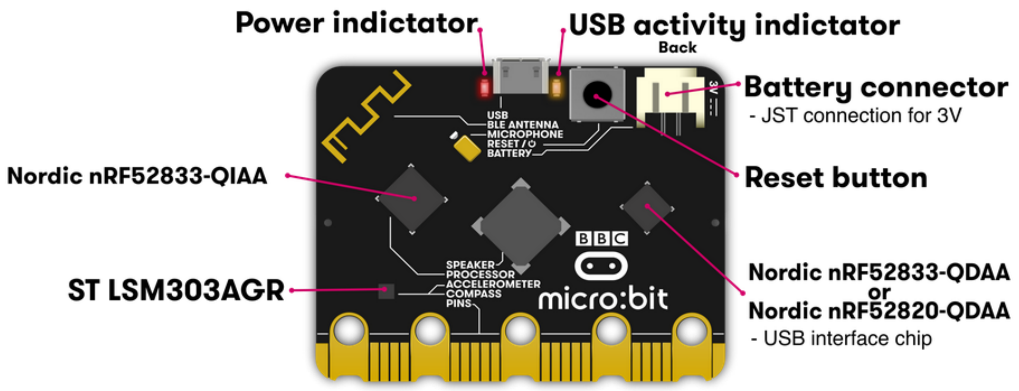
The **`micro:bit`** is an easily programmable Single Board Computer (SBC) that contains an application processor with a variety of on-chip peripherals. Other peripherals are connected to this chip.

An interface processor is connected to the application processor and manages communications via the USB interface, including the drag-and-drop code flashing process. The interface processor does not connect to any of the micro :bit peripherals.

Two key pieces of information to help understand the internals of the micro :bit are :

- The schematics, which shows the detailed component data and connectivity of the device.
- The reference design, which is a complete module design of a compatible micro :bit, and is designed to be a starting point for anyone interested in understanding the micro :bit or designing their own variant.





We are using `BBC micro:bit` V1.5 version.
The hardware description is available here .

2) micro :bit Micropython API

The **`BBC micro:bit`** is a small computing device. One of the languages it understands is the popular Python programming language. The version of Python that runs on the BBC micro :bit is called MicroPython.

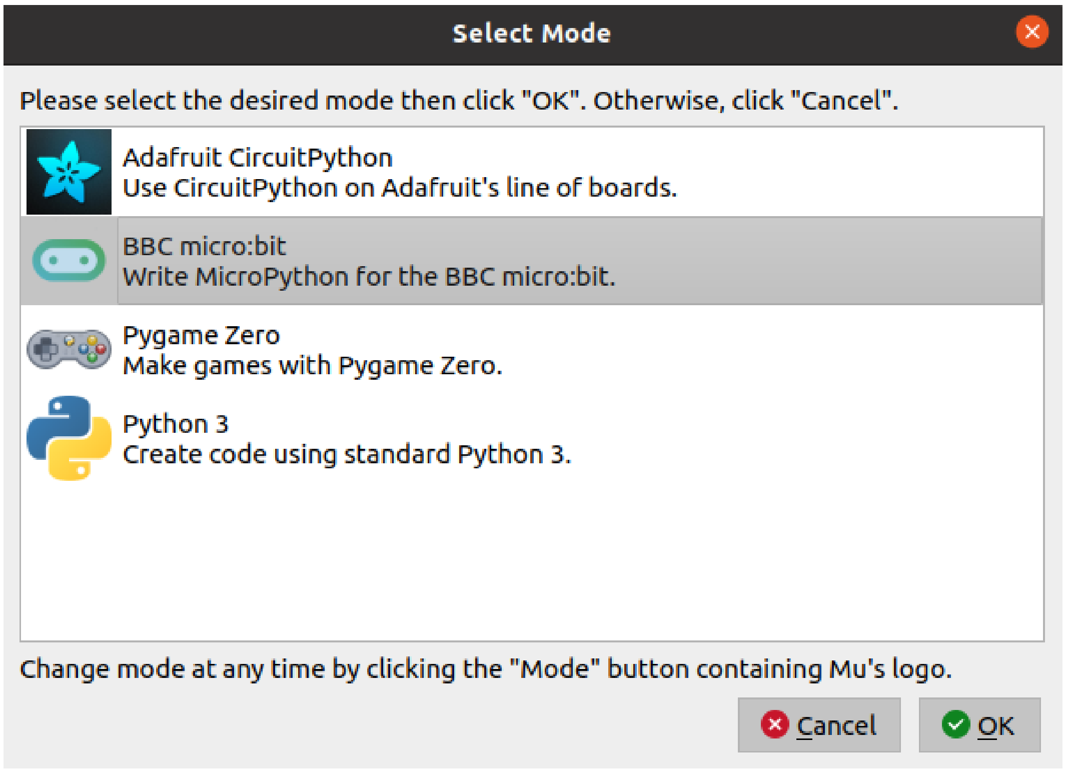
The documentation (available here) includes API documentation for developers.

3) How to installl software on your microbit ?

We suggest you download and use the mu editor when working through these tutorials. Instructions for downloading and installing Mu are on its website. You may need to install a driver, depending on your platform (instruction are on the website).

Mu works with Windows, OSX and Linux.

You shall select the **BBC micro :bit** mode to be able to connect the `micro:bit` to your computer.



Select Mode

Please select the desired mode then click "OK". Otherwise, click "Cancel".

Adafruit CircuitPython
Use CircuitPython on Adafruit's line of boards.

BBC micro:bit
Write MicroPython for the BBC micro:bit.

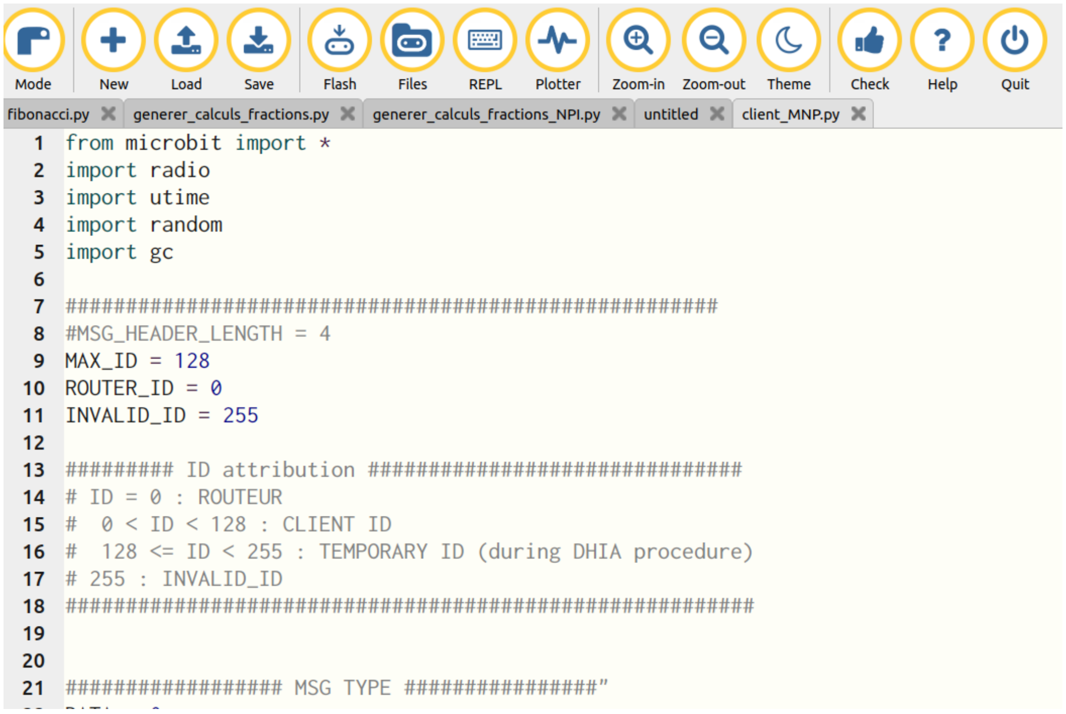
Pygame Zero
Make games with Pygame Zero.

Python 3
Create code using standard Python 3.

Change mode at any time by clicking the "Mode" button containing Mu's logo.

Once Mu is installed connect your micro :bit to your computer via a USB lead.

Write your script in the editor window and click the "Flash" button to transfer it to the micro :bit. If it doesn't work, make sure your micro :bit appears as a USB storage device in your file system explorer.



```
from microbit import *
import radio
import utime
import random
import gc

#######################################################
#MSG_HEADER_LENGTH = 4
MAX_ID = 128
ROUTER_ID = 0
INVALID_ID = 255

######### ID attribution ###############################
# ID = 0 : ROUTEUR
#  0 < ID < 128 : CLIENT ID
#  128 <= ID < 255 : TEMPORARY ID (during DHIA procedure)
# 255 : INVALID_ID
##########################################################


################## MSG TYPE ################"
```

4) `Micro:bit` memory constrains

On the `Micro:bit` V.5, the main constrain is the available ROM memory on the device.

- ROM memory (program) : 512 Ko (but the firmware occupies the major part of this memory location, around 400-500 ko). There are 30-40 ko available for your python program, which usually corresponds to a file size of 8-12 ko (around 200-300 code lines)
- RAM memory (used data during execution) : 16 Ko (which is very limited).
- If the maximum ROM memory size is reached by your program, the device will raise an error when trying to launch your program.

Before starting flashing code on your device, you shall make some measurements of the available ROM memory depending on the Python modules imported in your code, and the size of your code. Example of diagnostic program :

```
from microbit import *
import gc

while True:

    if button_a.was_pressed():
       print("o")
       display.scroll(temperature(), wait=False)

    if button_b.was_pressed():
        print("Memoire libre :", gc.mem_free(), "octets")
        print("Memoire utilisee :", gc.mem_alloc(), "octets")
```

Try also this diagnostic program with other python modules : radio, utime, random ... and report your measurement in your design documentation.

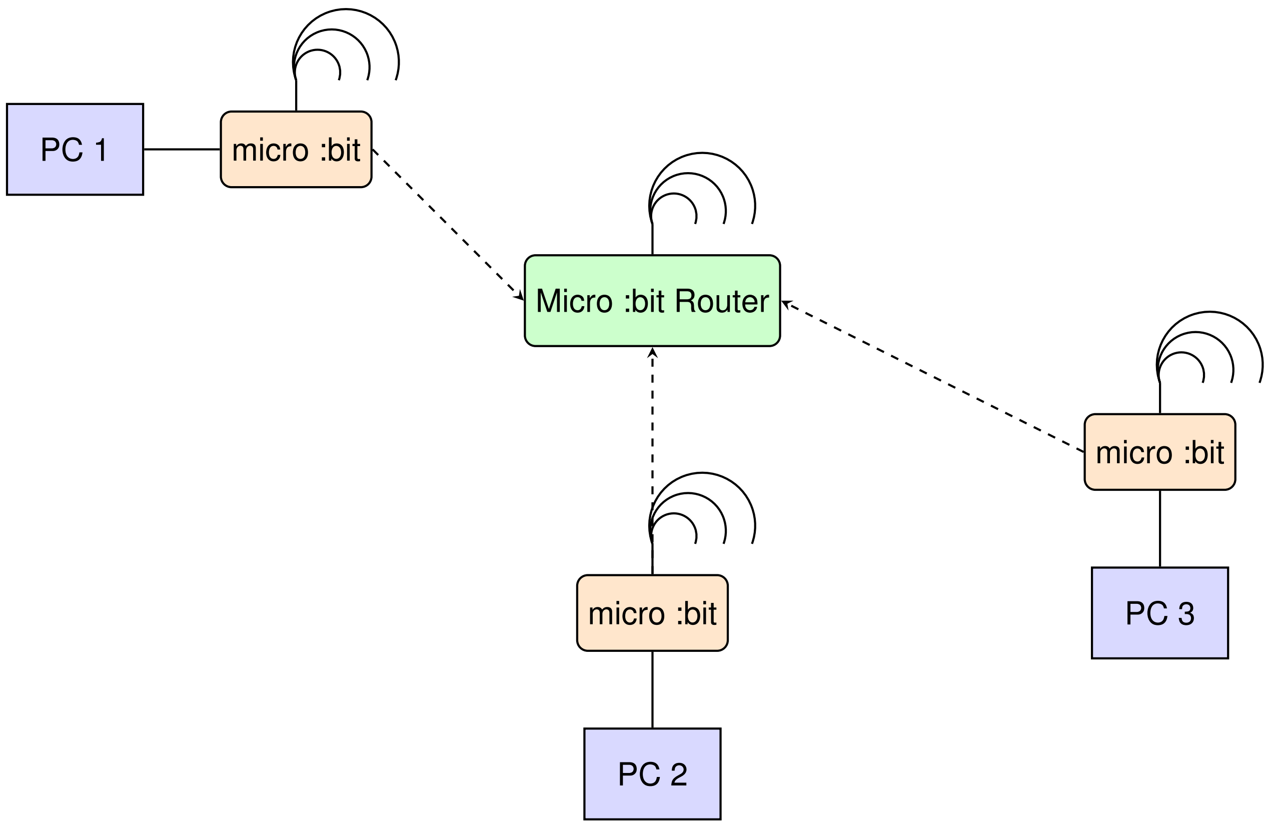
II - Project overview

The objective of the project is to create a **wireless local area network (WLAN)** and use **`BBC micro:bit`** as devices to exchange messages between users (by using the radio module of the **`BBC micro:bit`**).

To ensure communications between devices, a **protocol of communication** (a kind a common language) shall be defined. You already know many communication protocols (IP, TCP, HTTP...).

During this project, you will create your own protocol : **MNP (Microbit Network Protocol)**. MNP will ensure that devices (`BBC micro:bit`) in the wireless local area network will be able to correctly communicate.

Here is an exemple of architecure of the network :



Wireless local area network architecture

There are two type of devices : Router and Client.

Router (Standalone `micro:bit`)

The router is a `micro:bit` device which is the central component of the wireless network.
The device ID of the router is 0.
The main functionnalities of the router are :

- Allocate dynamically a device ID to every client devices in the network (similar as DHCP protocol which allocates IP address).
- Forward messages coming from a **sender** client to a **receiver** client. Every messages exchanged between client devices shall be sent first to the router. The router shall forward messages to the right client device.
- Check the availability of every client devices in the network (keep-alive service)

Client (`micro:bit` linked to a PC)

The client is a `micro:bit` device which is connected to a PC using a serial link. Every client devices shall have a unique device ID in the network, which is a number between 1 and 127.
The main functionnalities of the client are :

- Send a message to another client device in the network. Message shall be writen by using a graphic interface on the PC connected to the client device.
- Handle received messages from other client devices and show these messages in the graphic interface.
- Anwser to signalling messages coming from the router device. (A signalling message is a specific message usually used to configured the client device or to verify the availability of the device).

III - MNP protocol

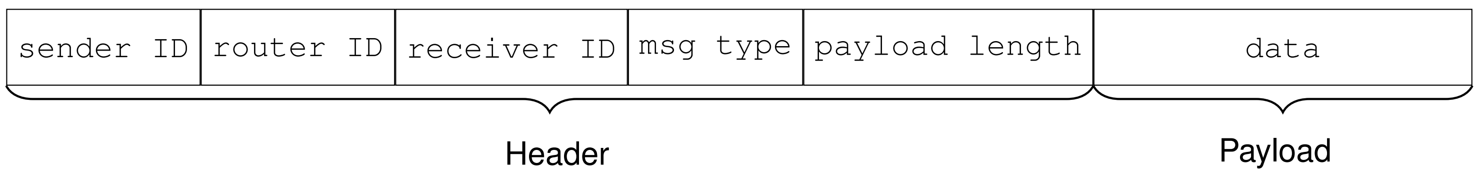
To ensure the communication between router and client, the same language shall be used : in the domain of telecommunication, this language is called a **protocol**.
The protocol defines :

1. The **type and the format** of every messages exchanged between devices.
2. The **authorised use cases**. A use case is a sequence of messages exchanged between devices. Some sequences are not permitted and lead to some errors that shall be reported.

1) MNP messages format

We are using the radio interface of the microbit device. The maximum lenght of messages that can be sent through this radio interface is 250 bytes.
Messages shall be encoded in UTF-8 encoding format. (UTF-8 is a character encoding standard used for electronic communication, every characters is encoded on 8 bits (1 byte)). It means that a message containing 8 characters as a length of 8 bytes.
It also means that messages sent through the radio interface have a maximum length of 250 characters (as 1 character is encoded on 1 byte).

Every messages sent through the radio interface shall be constructed according to the MNP protocol specification below. The message contains 2 parts : the **header** and the **payload**.



Header

The header is the part of the message that contains **control information**. It tells the network how to deliver and interpret the message.
In the MNP protocol, the header contains 5 fields and its length is 5 bytes.

- `sender ID` : 1 byte, positive integer, range value : $[0 : 254]$. Indicates the device ID of the sender of the message.
- `router ID` : 1 byte, fixed value : 0. The device ID of router (always 0).
- `receiver ID` : 1 byte, positive integer, range value : $[0 : 127]$. Indicates the device ID of the receiver of the message.
- `msg type` : 1 byte, positive integer, range value : $[0 : 255]$. Indicates the type of the message sent. The differents type of messages are detailled below.
- `payload length` : 1 byte, positive integer, $[0 : 245]$. The length of the payload of the message. As maximum message length is 250 bytes and the header lentgh if 5 bytes, the maximum payload length is 245 bytes.

Payload

The payload is the part of the message that contains the usefull data or content being sent. For example a text, an image, or a command. The content depends on the message type. The maximum lenght of the payload is 245 bytes.

2) MNP messages type

Data message

This message is used by a client device to sent data (as text) to another client device.

- **`sender ID`** : sender's device ID
- **`router ID`** : 0
- **`receiver ID`** : receiver's device ID
- **`msg type`** : DATA
- **`payload length`** : payload length in bytes.
- **`payload`** : data as text.

Signalling message : Dynamic Device Id Allocation Request

This message is sent by a client device to the router to request the allocation of a device ID. At firt the client device uses a temporary device ID randomly chosen by the client itself (range value : $[128 : 254]$).

- **`sender ID`** : temporary device ID. At this step, no device ID has been allocated by the router to the client device.
- **`router ID`** : 0
- **`receiver ID`** : 0
- **`msg type`** : $\mathrm{DDIA\_REQ}$
- **`payload length`** : 0
- **`payload`** : None

Signalling message : Dynamic Device Id Allocation Response

This message is sent by the router to a client device. It contains the client device ID allocated by the router.

- **`sender ID`** : 0
- **`router ID`** : 0
- **`receiver ID`** : temporary device ID of the client that request a valid device ID.
- **`msg type`** : $\mathrm{DDIA\_RESP}$
- **`payload length`** : 1
- **`payload`** : new device ID. Range value : $[1 : 127]$

Signalling message : Dynamic Device Id Allocation Acknowledgment

This message is sent by a client device to the router. It acknowledges and validates the new device ID allocated by the router. The client device is then considered as **connected** to the network.

- **`sender ID`** : sender's new device ID
- **`router ID`** : 0
- **`receiver ID`** : 0
- **`msg type`** : $\mathrm{DDIA\_ACK}$
- **`payload length`** : 0
- **`payload`** : None

Signalling message : Ping Request

This message is sent by the router device to a client device. It checks the availability of the client device. If the client does not answer within a window of time of 5 seconds, then the router considers the client device as disconnected from the network. Note that this message is alos used to provide the liste of connected devices to a client device.

- **`sender ID`** : 0
- **`router ID`** : 0
- **`receiver ID`** : client device ID
- **`msg type`** : `PING_REQ`
- **`payload length`** : length of the list of connected device
- **`payload`** : List of client device connected to the network

Signalling message : Ping Response

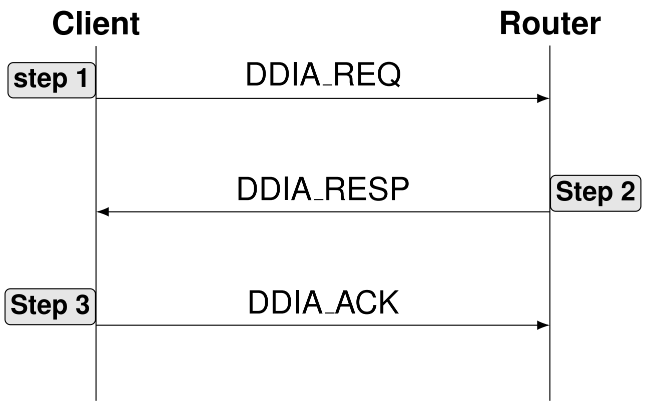
This message is sent by client device to the router device. The is the answer to Ping request. It indicates that the client device is alive and still connected to the network.

- **`sender ID`** : client device ID
- **`router ID`** : 0
- **`receiver ID`** : 0
- **`msg type`** : `PING_RESP`
- **`payload length`** : 0
- **`payload`** : `None`

3) MNP use cases

Dynamic Device Id Allocation procedure

When a client device wants to reach the network, it shall request to the router a device ID that will identifie the client device in the network. This device ID will be needed to send and receive messages. The router allocates a client device ID and shall maintain a list of the device IDs connected to the network.
The several steps of this use case are detailled below :



Dynamic Device Id Allocation diagram

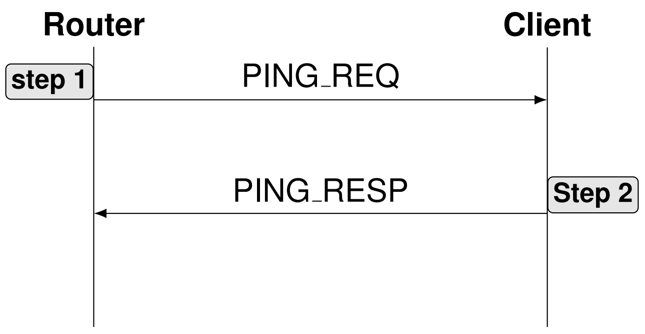
- **Step 1** : A client device, not already connected to the network, sends a `DDIA_REQ` message to the router to request the allocation of a device ID. Until the client device receives the new device ID, it shall use a temporary device ID randomly chosen by itself (range value : $[128 : 254]$).

- **Step 2** : If the maximum number of connected devices (127) is not reached, the router sends a `DDIA_RESP` message to the client device, including in the payload the new device ID (range value : $[1 : 127]$). This new device ID shall be used by the client in every communication. At this step, the client is not considered as connected.
- **Step 3** : The client has received the new device ID and sends a `DDIA_ACK` message to the router to acknowledge the allocation of the new device ID. The client is now considered as connected to the network, and the router add this client in its list of connected devices. The temporary device ID is no more used.

Keep Alive procedure

Every 5 seconds the router shall send a Ping request to every connected client devices. The Ping request includes the list of every connected client devices, so that client devices are aware, in real-time, of the network topology.
When receiving the Ping request, client device shall send a Ping response to the router. If the client does not answer to the router (client no more connected, or client in a unstable state), then the router shall consider the client as disconnected and shall remove it from its list of connected client devices.
There a two use cases :

- **Alive use case**

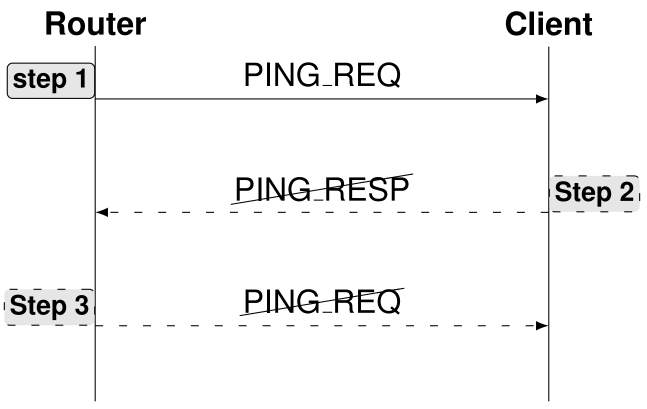


Keep alive diagram : alive use case

  - **Step 1** : The router sends a `PING_REQ` to the client device. The payload of the message contains the list of all the client devices connected to the network.
  - **Step 2** : If the client device is alive, it sends a `PING_RESP` to the router. The router considers the client device alive and shall maintain the client in its list of connected client devices.

  This procedure continues every 5 secondes.

- **Not alive use case**



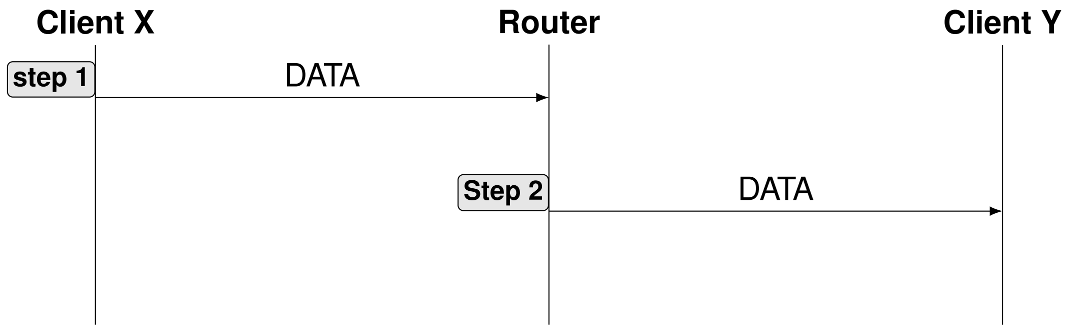
Keep alive diagram : not alive use case

  - **Step 1** : The router sends a `PING_REQ` to the client device. The payload of the message contains the list of all the client devices connected to the network.

- **Step 2** : If the client device is not alive, the `PING_RESP` is not sent to the router.
- **Step 3** : 5 seconds after the router sent the last `PING_REQ`, the router checks if it has received a `PING_RESP` from the client device. If not, the router does not send the next `PING_REQ`, the client device is considered as not alive and removed from the list of connected client devices.

Data messages procedure

When a client device wants to send a message (like a chat message) to another client, it shall send the message to the router which will forward it to the receiver's client.



Data messages diagram

- **Step 1** : The client X sends a `DATA` message to the router. The message includes the device ID of client Y (the final receiver).
- **Step 2** : The router forwards the `DATA` message to the client Y.

IV - Man-machine interface

The Man-machine interface is located on a PC connected to a `micro:bit` client device.
The connection is done by using the serial port of both `micro:bit` device and PC.

Functionalities of the Man-machine interface

- Connect/disconnect button :
  - ⋆ To connect : send a request to the `micro:bit` device to start the dynamic id allocation procedure.
  - ⋆ To disconnect : change the state of the client device to disconnected. Ping request won't be answered anymore, so that the client device will be naturally disconneted from the network.
- Message writing frame : write a message
- Send button : send the mesage written in the writing frame to the sender client device (as `DATA` message type).
- Received messages frame : print the received messages from other client devices.
- List of connected devices : use to select the receiver of the message written in the writing frame.
- Log frame : print the signalling messages or errors messages received from the router or the client device itself..