# Device parameters

Board: WiPy 3.0

Upload speed: 921600 bps

Flash frequency: 80MHz

Core frequency: 240 MHz

Serial port baudrate: 230400 bps

# Characterization of TN

The purpose is to find a stable value of TN which does not crash the CPU core nor induces memory overflows and to see the influence of TN over the success probability.

Because of the limited transmission queue and of a memory leak in the IP stack (not solved yet by Espressif and not really documented), when too much activity is performed on the Wi-Fi interface the possible errors can occur:

* Heap full / memory overflow -> causes full reset of the CPU
* CPU core panic -> causes complete deadlock of both cores of the CPU
* Wi-Fi out of memory -> The transmit queue gets discarded without transmissions.

The solution to prevent errors on the beacon transmissions are:

1. Insert a 1ms delay after each beacon transmission. This gives time to the driver to perform the transmission;
2. Insert a 10ms delay after the broadcasts to give time to the driver to perform cleanup and empty the transmit queue;
3. Check for transmission errors after calling esp\_wifi\_80211\_tx(WIFI\_IF\_STA, packet, 81, false);

These measurements allows to reduce the duration of the broadcast state to ~30ms on average.

The solution to prevent errors on IP networking are:

1. Always check if the WiFi state machine is in WL\_CONNECTED state. Do not call UDP or TCP functions when not connected.
2. Check that “udp.parsePacket();” returns a value bigger than zero which means a packet has been received. This prevents accessing unallocated memory causing a reset.
3. Always call udp.flush() after any UDP operation to be sure the queues are clean reducing the amount of memory errors.
4. Put a delay of at least 20ms after every network operation to give time to the WiFi driver and the OS to complete their operations. If not done, the CPU stall condition occurs and no software reset is possible anymore.
5. The only way (that I found) to prevent the memory leak is to disconnect from the WiFi after every UDP operation and reconnect before starting a new UDP operation. These takes in total approximately 180ms (no association is needed anymore, that is why it is so fast). Disconnect and reconnect operations force the WiFi driver to reallocate the memory dedicated to IP networking preventing the memory leak that causes reset and sometimes the CPU stall.

## Methodology

All timings are measured by toggling a GPIO pin at the beginning and at the end of each function call and measure the average duration with the oscilloscope. The duration of the broadcastSSID() operation is influenced by the serial port communication in case the pc is not ready to transmit when the node reads data. The time out of the serial read is fixed at 100ms.

An open terminal shows the result of IP communication while a python script takes care of sending numbered messages through the serial port and logging what comes from the serial port.

2x ESP32 modules are programmed as nodes performing broadcasting, scanning and networking while a third one is configured as access point (ESP32 does not support EAP and all ports on ESAT networks are closed for UDP, which makes it impossible to use EDUROAM or CAMPUSROAM).

The ESP32 sends via UDP the amount of free heap and gets back a received confirmation message.

Once a safe value for TN is found, it can be changed gradually to see how it influences the probability of success.

**Expected duration: 6 hours**

## Measurement parameters

* Broadcast delay after tx: from 0 to 100 ms;
* Broadcast delay in between transmissions: from 1 to 5ms;
* Network delay: from 0 to 100 ms;

# Test different configurations of (PS,PN,PB)

The purpose of this experiment is to see how the success probability changes for different values of probabilities. Since we know that the success probability is the product between PS and PB, increasing PN will reduce it.

From this experiment, it will be possible to get:

* Success probability distribution for 1s time windows
* Total Success probability vs PN

## Methodology

From the analytical model, the best success probabilities are obtained for PS = PB. Fixed this conditions, different values of PN can be programmed in the ESP32.

The correct values to be compared with the random generator in the firmware must be computed keeping into account the different duration of the three states (Scan, network and broadcast).

**Expected duration: 1 hour per condition + 2h development time (7h total)**

## Measurement parameters

* PN: 0.10, 0.50
* PS (and PB): 0.45, 0.25

# Range influence measurements

The goal of this measurement is to evaluate the working range of the broadcast/scan.

## Methodology

The experiment can be conducted equipping two drones with ESP32 Wi-Fi modules programmed to broadcast and scan.

One drone will fly stationary in front of ESAT at 10m altitude from the ground. The second drone will fly progressively further towards the castle first and then towards the computer science department. The GPS coordinates of each drone will be recorded periodically. Measurements will be conducted in a stationary condition (the moving drone will stop at specific coordinates to record received messages.

**Expected duration: 1 day**

## Measurement parameters

Distance between the drones/GPS points: from 0 to 500m (or until no messages are received anymore).

# Collisions: 1 to 15 nodes

The goal of this experiment is to evaluate the impact of colliding transmissions from multiple nodes over the probability of success.

## Methodology

Starting with 2 devices in the Faraday cage, with fixed PS,PB and PN, increase the number of nodes up to 15 and evaluate how the message loss increases. Results can be compared with the simulation.

**Expected duration: 1 day**

## Measurement parameters

Number of nodes: 2, 5, 10, 15

3.1) i) N\_success=1=> Pn,b,s; ii) Pn=0.5 iii) Pn=0.1  
4.2) i) logscale the inter-drone distance; ii) actually hovering vs moving is interesting  
5.2) № of interferers 1, 5, 10, 15