The transmission rate of ESP32

**Objective**

The objective of this document is to put in evidence the current performance of Smooth’s MQTT. To do so, Smooth MQTT is contrasted with TCP/IP from ESP-IDF. To provide a good comparison, the same hardware setup is used for both tests.

**Hardware Setup**

* Router set on 2.4 GHz frequency.
* ESP32-DevKitC-VE with chip ESP32-WROVER-E 8MB
* Lenovo Laptop running Mosquitto 2.0.10 and/or Netcat 1.11 for NT.

**Procedure**

This document is divided into two main sections. The first part tests the transmission capabilities of Smooth MQTT, the second is a simple test of ESP32 transmission rate using TCP/IP.

# Benchmark Smooth MQTT

*Objective*

The goal is to evaluate and analyze the transmission capabilities of Smooth MQTT on ESP32.

*Procedure*

For this experiment, we use the template code provided in Smooth examples (it can be retrieved from “Smooth/test/mqtt”. The code was modified to time its transmission rate. The complete code can be found here:

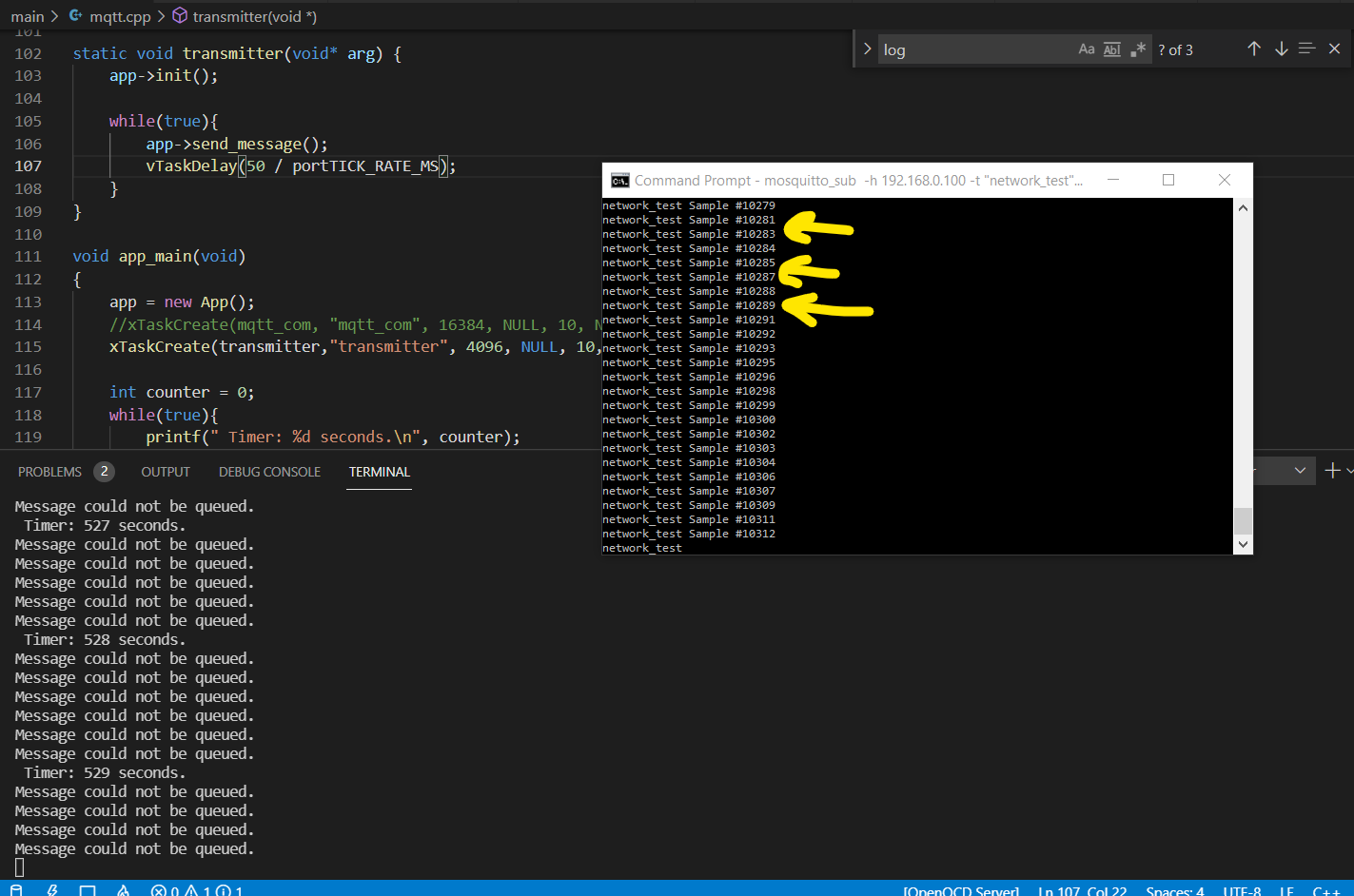
<https://gist.github.com/Elfelsoufim/f9ef715755d385cce8258ba149ab7861>

## Part 1: Illustrating data loss.

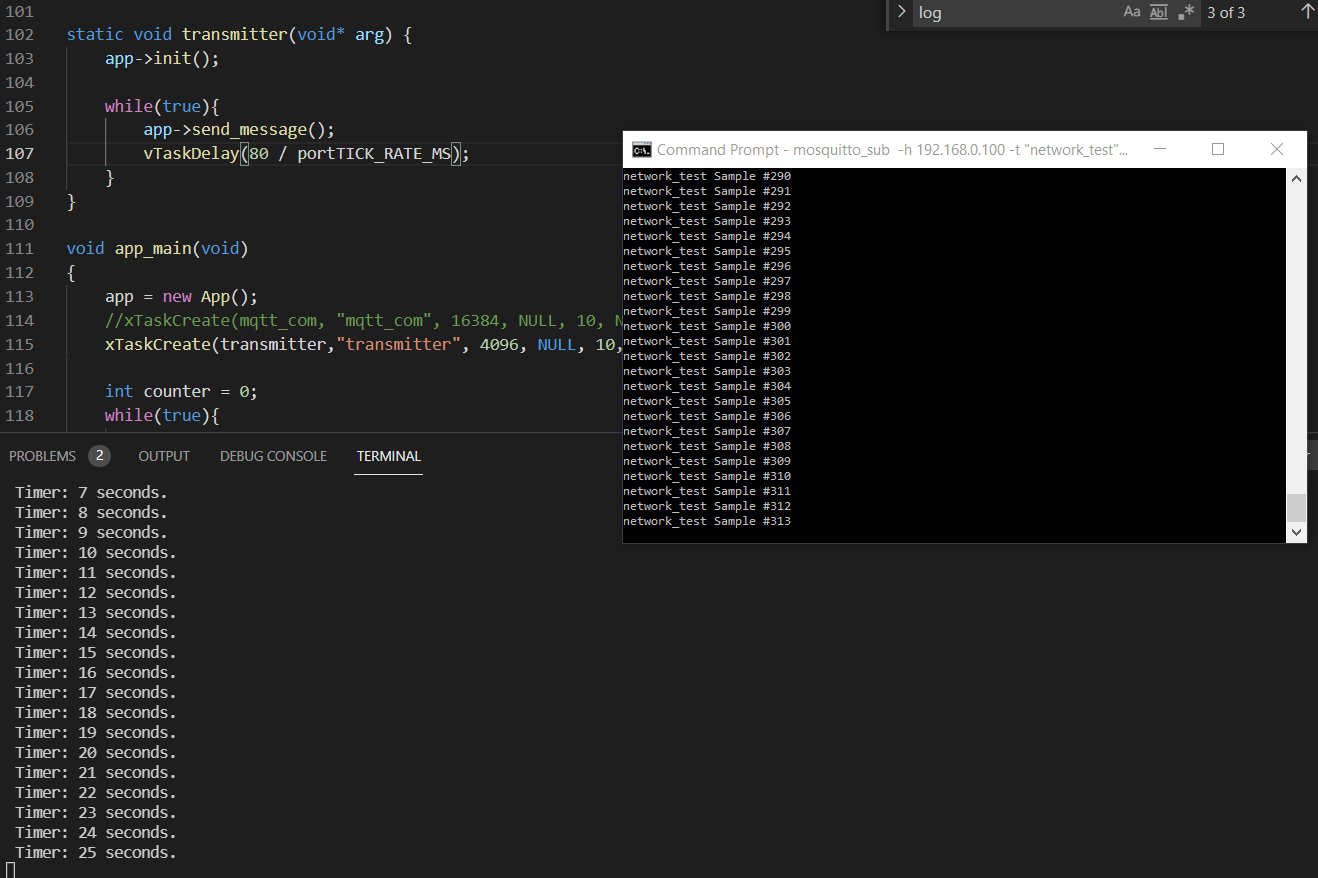
Before testing Smooth MQTT’s transmission speed, it is important to see what happens when it is driven over its capacity. When trying to send messages too quickly, the transmit buffer gets filled faster than it gets emptied, which results in data not getting added to the buffer array.

This data loss is obvious when every message sent is numbered. At 50 ms between each message sent (see line 107 in Figure 1), which means 20 messages per second, about 5 messages cannot get added to the queue every second. For example, in Figure 1, the message “Sample #10282” was never sent nor received.

In Figure 1 to 5, the command prompt window is the laptop receiver, a Mosquitto client that receives all messages sent by ESP32. The VSCode Terminal window monitors the ESP32 through a USB cable.

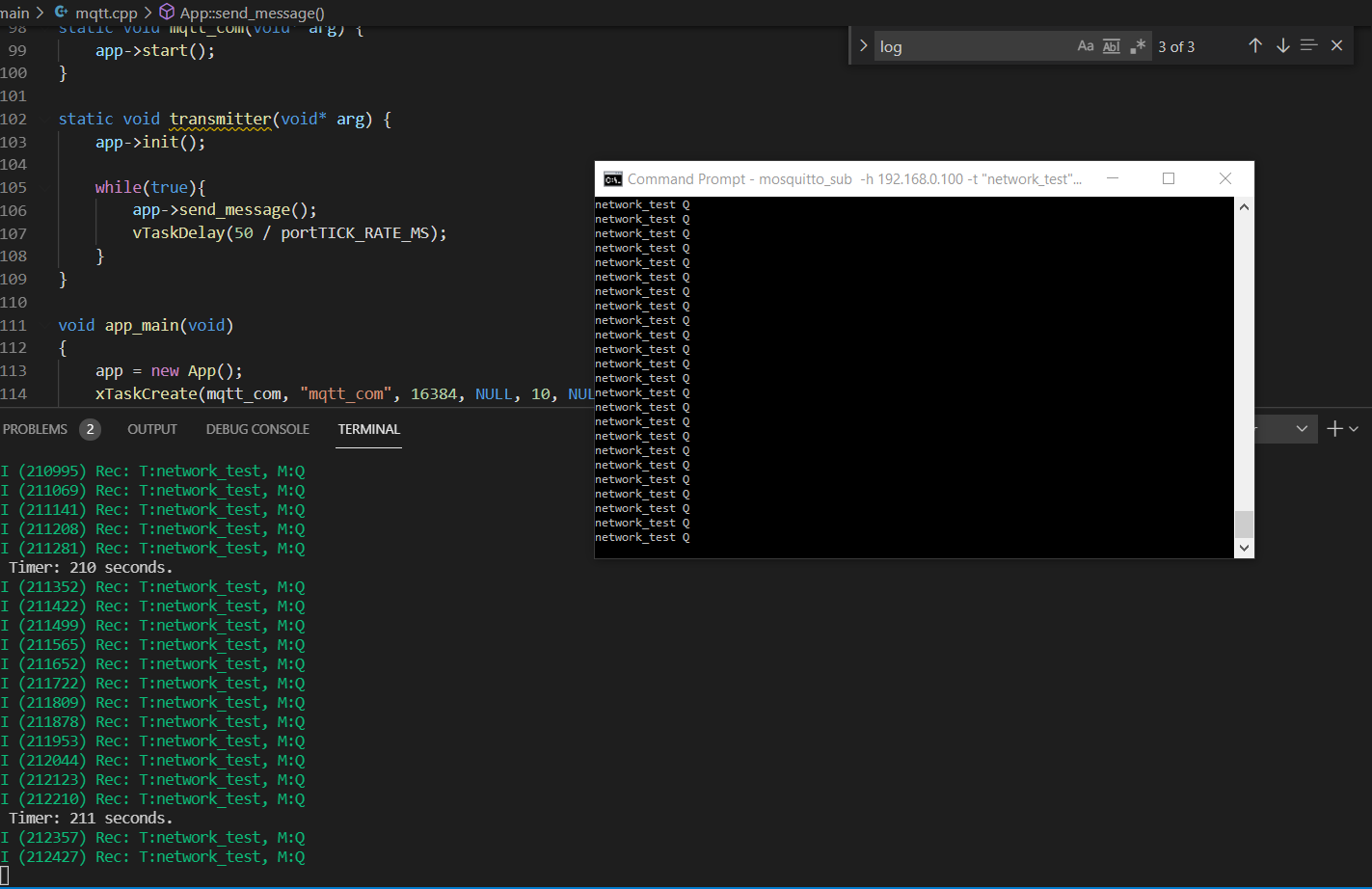
**Figure 1:** Output of the laptop receiver when ESP32 sends messages at a rate of 20 Hz.

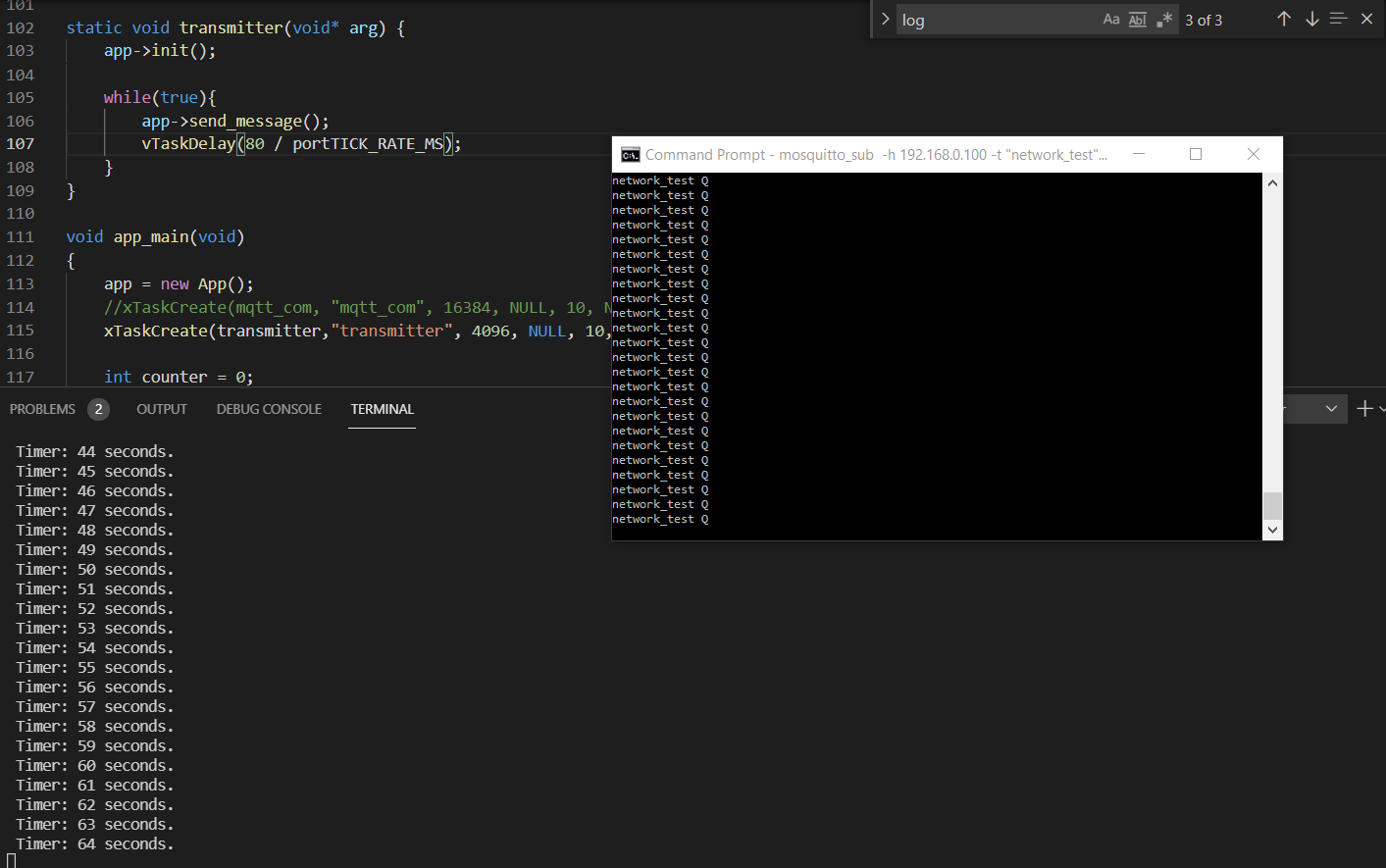
If I increase the delay between messages to 80 ms (see line 107 in Figure 2), all messages can get added to the transmit queue since the transmit buffer gets emptied faster than it gets filled. This is a state that I call “reliable transmission”, where every single message gets transmitted correctly. Following these results, I conclude that Smooth MQTT's maximum transmission rate is 12 messages per second.

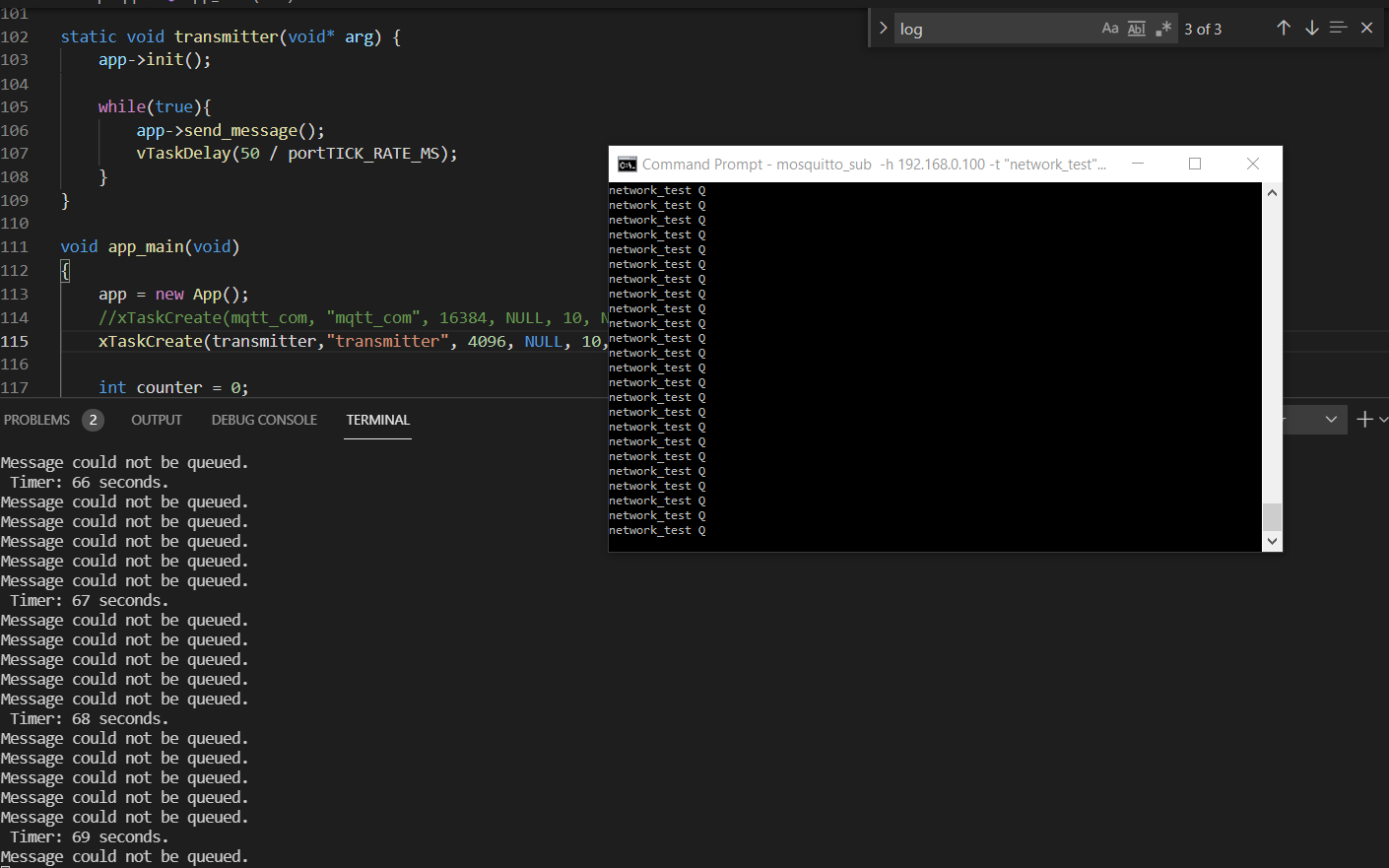
**Figure 2:** Output of laptop-based receiver when ESP32 sends messages at a rate of 12 Hz.

## Part 2: Benchmark publish().

To isolate the problem and avoid any slowdown caused by intermediary computations, I send a fixed-value: a constant string of value “Q”. The results are very similar, the ESP32 can reliably send messages only if they are queued with a delay of 70 to 80 ms between each other. In Figure 3, ESP32 was programmed to send a message whenever it receives one, allowing Smooth MQTT to run at cruise speed. This is done by calling Task::start(). From Figure 3, we can confirm that the natural transmission rate of ESP32 is around 12 Hz.

**Figure 3:** Output of ESP32 transmitting messages and receiving them through the Mosquitto broker.

** Figure 4:** Output of the laptop receiver when ESP32 sends a fixed message at a rate of 12 Hz.

**Figure 5:** Output of the laptop receiver when ESP32 sends a fixed message at a rate of 20 Hz.

# The transmission rate of ESP32 on TCP/IP

*Objective*

The goal of this experiment is to determine the raw transmission rate of ESP32. To do so, we set the ESP32 to send short messages to a laptop using TCP.

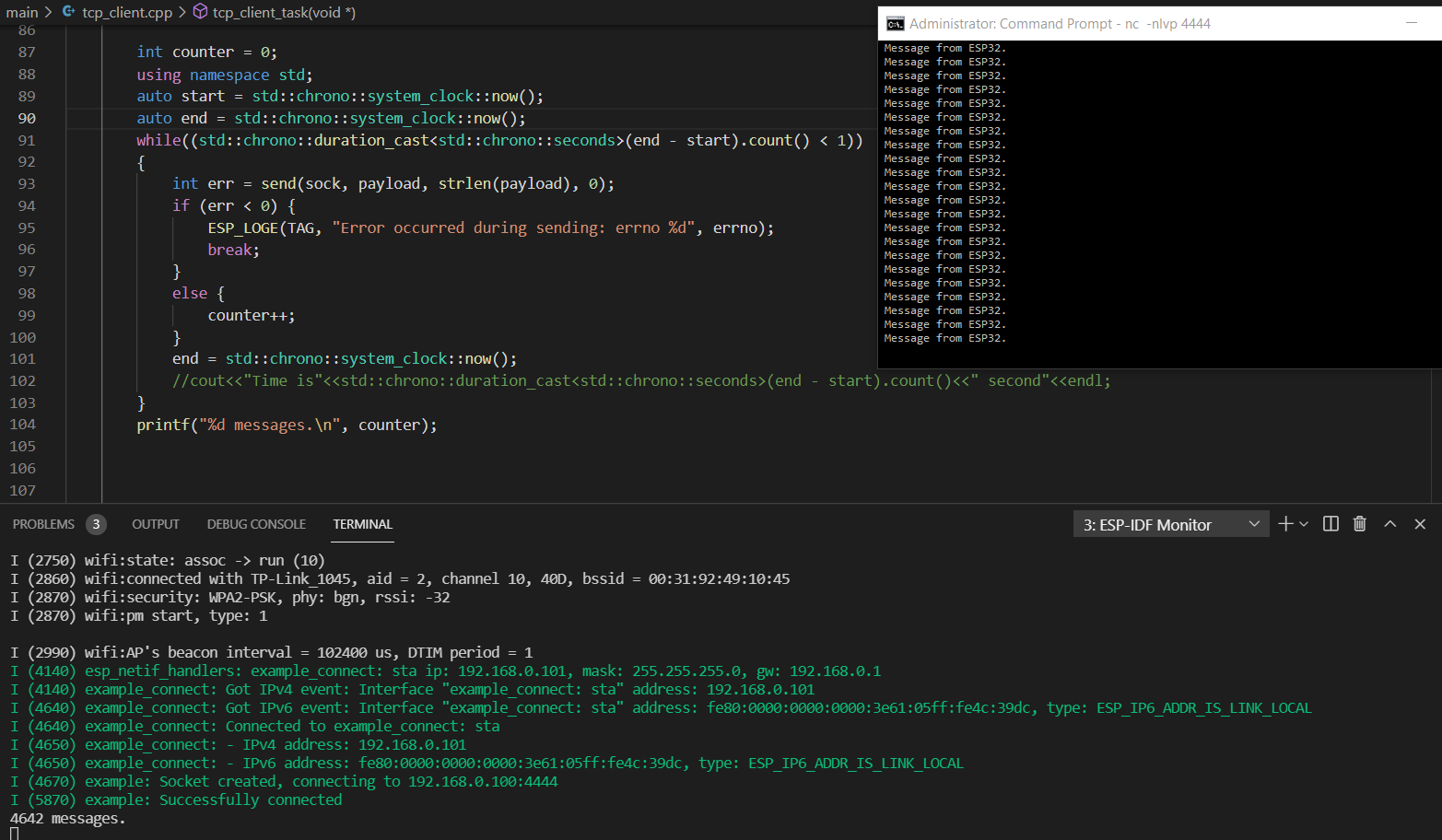
*Procedure*

The code to send TCP/IP messages from the ESP32 was taken from:

<https://github.com/espressif/esp-idf/tree/c13afea635adec735435961270d0894ff46eef85/examples/protocols/sockets/tcp_client>

And modified in the following manner:

The receive code was removed. The program was set to run for 1 second using system\_clock from <chrono> library. A counter included inside the loop was incremented every time a transmission was successful. When the time is up, the value of the counter is printed.

**Figure 1:** The code snippet that sends TCP/IP messages.

*Results*

|  |  |
| --- | --- |
| Run # | Transmission rate (messages/second) |
| 1 | 684 |
| 2 | 684 |
| 3 | 3166 |
| 4 | 4087 |
| 5 | 3837 |
| 6 | 4066 |
| 7 | 4230 |
| 8 | 2920 |
| 9 | 4654 |
| 10 | 3418 |
| 11 | 4982 |
| 12 | 4642 |

**Table 1:** Results of ESP32 transmission speed test

|  |  |
| --- | --- |
| Minimum | 2920 |
| Maximum | 4982 |
| Average | 4000 |
| Standard Deviation | 674 |

**Table 2:** Statistics from ESP32 transmission speed test

*Discussion*

Run #1 and #2 executed a print statement at each transmission. As it was removed for run #3 to #12, we can see that the print statement caused important delays in the execution of the program. Therefore, the results from the first two runs were discarded.

Runs #3 to #6 were conducted inside a room (where the router is not present) with the door closed. Runs #7 to #12 were conducted 2 meters from the router without any obstacles in the way. We can see that obstacles and walls negatively influence the transmission rate as the average of run #3 to #6 is 3789 while the average of run #7 to #12 is 4141.

During the test, I noticed that closing other programs running on the receiving computer as well as the orientation and position of the router and/or the ESP32 can influence the throughput, which might explain why the last runs had better results overall.

*Conclusion*

The goal of this test was to verify the bare transmission speed of ESP32. As TCP/IP is a protocol that necessitates an acknowledgment of every data packet sent, we can conclude that ESP32’s bare-metal transmission rate is likely much higher than 5000 messages per second.

**Conclusion**

Considering that MQTT is based on the TCP/IP protocol, the important gap between the benchmarks of the two protocols suggests that there is room for improvement for Smooth’s MQTT in order to take advantage of the full potential of Smooth for ESP32.