Title: WHAT IS THE BEST WAY FOR THE FLOW OF COMMUNICATION?

Wasefi Mohammad Asif

Table of contents:

CONTENTS

Intro

For this part of the project we need to delve a bit deeper into flow of communication between the cars. In previous sessions we have concluded that UDP would be a more suitable channel for the devices to interact with each other. But in which order do we want our cars and the server to communicate? Which device has to initiate the commands and which devices need to act first? This would be discussed in this project.

Materials

- Ubuntu server
- 3 RC cars with integrated nodeMCUs

Methods

Assuming the programming part has been done nicely i.e. the nodeMCUs understand the content of the packet sent from the server and vice versa. How do we think the communication will look like?

Communication via:

- UDP
    - Faster, the packets do not need to be acknowledged.
    - Unreliable: the packets are pushed on one-way, there is no way the lost packets could be resent.
- TCP
    - Reliable: it ensures the packets are sent completely
    - Two-way communication: the packets needs to be tracked if they travelled completely otherwise they are resent.
    - Can be sometimes time consuming

UDP communication:

We have yet only tested a direct connection between UDP server (our Ubuntu server) and a smartphone receiver which resulted in 36% packet loss in two ways communication and 53 milliseconds transmission time in one-way only. For this project we have simulated similar conditions as if a nodeMCU is communication with an Ubuntu server i.e. 2.4 GHz band of frequency.

Why we used a smartphone and not a nodeMCU itself?

This piece of code needed Python programming language which needed to be researched and installed on nodeMCU. This could have taken more time and research.

Note: we assume the same results will be there if a nodeMCU had been used instead of a smartphone.

We have a central Ubuntu server with 3 RC cars with Wi-Fi capability via nodeMCU and integrated distance sensor. To be able to give a whole picture of the whole communication. We need to draw a few scenarios:

Scenario 1:

The server communicates only with one of the RC cars which stays in the middle of the two other. This car will be named main car here forth. The other cars are to be named secondary cars:

Say if the server sends an accelerate message to main car. The distance on the front car shrinks and enlarges on the tailing car. This distance change is fed to the nodeMCU. The secondary nodeMCUs will communicate it to the main car. The main car will report it to the server and the server will reply specifically to the secondary cars via IP address to accelerate as well. The braking concept is an analogue to the acceleration process.

Time elapsed: for a simple command mentioned above we would need to make a calculation of the elapsed time.

1. Server tries to accelerate the main car which is on average 53 milliseconds.
2. The two lagging cars will communicate simultaneously the distance change to the main car which will also be 53 milliseconds.
3. The main car feeds this report to the server in about 53 milliseconds.
4. The server replies simultaneously to the lagging cars on average in 53 milliseconds.

A simple task as accelerating the traffic would take 4 x 53 milliseconds which would be roughly 212 milliseconds under ideal conditions.

Note: every aspect of the communication is assumed to take place on UDP on 2.4 GHz with average one-way transmission time of 53 milliseconds.

Scenario 2:

Every car communicates with the Ubuntu server. Say the server tells the main car to accelerate. The distance change will be felt in the secondary cars. This change will be reported to the server directly. The server processes the feed and acts accordingly i.e. tell the secondary cars to accelerate as well.

This scenario meets the needs for real-life applications. Because we would not want a car to act as a middleman to represent other cars and talk to the server. This way the main car would be a mini server. So there is a need for a central server which can analyze all the data coming from the RC cars and save it for researching purposes on how to make it better.

Time elapsed: for the same simple task mentioned above there is only one step which is eliminated i.e. the main car doesn't receive the distance change from the secondary cars in order to forward them to server. This could be a significant advantage in terms of transmission time.

Scenario 3:

The main car acts as a self-sufficient server:

If we eliminate the Ubuntu server and make the main car act as the server, there would be a significant amount of processing load put on the main car i.e. programming capability and hardware resources to process the data. If we assume these obstacles are overcome, we would still be far away from the real-life concept of controlling and managing the flow of the car. Because there would be no way to improve the concept. All the data would be transmitted between the cars only and this data is then not stored on a central place.

Time elapsed: for a simple task of accelerating the main car and making other two follow, the time consumption would be as follows:

The main car accelerates where the distance change is felt on the secondary cars. These cars report the change to the main car in around 53 milliseconds. The main car processes the data and sends the simultaneous commands accordingly which would also take 53 milliseconds under ideal conditions.

Conclusion

After calculating and assessing the time-lapse and compatibilities of the above 3 scenarios with real-life situation, it seems that scenario 2 would be the best option to consider i.e. every RC car reports individually to the central server. The server analyzes the incoming traffic and acts accordingly.

Bibliography

Holm Security. (2018, March 8). *Holm Security*. Consulté le April 18, 2018, sur Holm Security: https://support.holmsecurity.com/hc/en-us/articles/212963869-What-is-the-difference-between-TPC-and-UDP-