Habib University

EE 322L - Analog and Digital Communication

Demonstrating Scheduling Algorithms

Lab Project Report

Group Members:
Syed Sameer Nadeem (sn02902)
Muhammaed Haris (mh02272)
Section: L2

Instructor:
Sir Tariq Mumtaz
Ma'am Zareen Tabssum
Sir Muhammad Raza Rizvi

April 29, 2019

Contents

1 Abstract

The aim of this project was to demonstrate scheduling algorithm after synchronizing nodes in an Ad-Hoc network for pair to pair communication. The need for scheduling arises due to the fact that the spectrum is limited and in order to cater to more users, the users are scheduled such that their messages do not interfere with each other. Arduino-RF module pairs were used to demonstrate nodes in an Ad-Hoc network and each node was assigned individual slots for receiving and transmitting. The first part was to synchronize, where one of the nodes triggers a signal and then the rest wait till they recieve the trigger. Then the actual communication starts. However since the channel is not perfect the trigger is sent multiple times whihc in turn creates discrepancies and does not allow perfect synchronization. However, the communication goes fine if the channel is satisfactory and the scheduling is demonstrated.

2 Introduction

Wireless communication is an integral part of our daily lives. Unlike wired communication where the channel or medium of the is wire/fiber-optics, wireless communication relies on air as its medium. Although having air as the medium saves cost on laying cables, however the spectrum in which the messages are modulated and transmitted is quite limited and expensive. There are tonnes of techniques to increase the throughput capacity. One of the techniques is to divide in time, and schedule effectively in such a way that the same spectrum (of 433 MHz carrier in this case) is shared by all but not all the time but rather each user is given a slot to receive and transmit.

The project makes use of pair-to-pair communication instead of a centralized network. Pair-to-pair network or Ad-hoc network doesn't rely on a centralized Base Station to schedule. In this system the individual nodes schedule themselves and synchronize on their own rather than having a single clock being fed to the Base Stations. This adds up to the challenge of locally synchronizing the nodes.

3 Rationale of the Solution

The limited nature of the spectrum makes us to use it more efficiently in such a way that more number of users can be incorporated. It is to ensure that the throughput capacity of the system is increased. An algorithm is devised to synchronize the nodes first. And then each node is assigned a slot, x seconds, in time every T seconds. The lesser is x the better the system and lower is the latency. However for demonstration purposes $x = 2$ seconds is this case.

The solution partially takes care of the issue of security breach as well. If a node tries to receive data when it is not its time then the node can't interpret the data is ciphered such that the node won't be able to make sense of the data it receives even if it tries to intercept the data. As of yet we haven't incorporated the idea of ciphering but rather we have made

it in such a way that for example node 1 has to receive specific bits in a specific slot if it recieves something else then the LED won't blink.

4 Discussion

The approach taken to form an algorithm is to synchronize the nodes first. That is a trigger signal sent by the first node and the the rest of the two nodes stay silent until they recieve the trigger node and then they start the communication procedure. The following is the code for the syncronisation,

```
void setup()
{
    delay(2000);
    Serial.begin(9600);  // Debugging only
    Serial.println("setup");
    uint8_t buf[VW_MAX_MESSAGE_LEN];
    uint8_t buflen = VW_MAX_MESSAGE_LEN; // creates a variable to store the data

    // Initialise the IO and ISR
    vw_set_tx_pin(transmit_pin);
    vw_set_rx_pin(receive_pin); // Define Rx Tx pins
    //vw_set_ptt_pin(transmit_en_pin);
    vw_set_ptt_inverted(true); // Required for DR3100
    vw_setup(2000);  // Bits per sec
    char sync1[7] = {'4'};
    for (int i=0; i<10; i++){
      vw_send((uint8_t *)sync1, 7); // Signal a trigger to synchronize
      delay(10);
    }
    pinMode(led_pin, OUTPUT);
```

```
void setup()
{
  uint8_t buf[VW_MAX_MESSAGE_LEN];
  uint8_t buflen = VW_MAX_MESSAGE_LEN;
  // Initialise the IO and ISR
  Serial.begin(9600);
  vw_set_tx_pin(transmit_pin);
  vw_set_rx_pin(receive_pin);
  //vw_set_ptt_pin(transmit_en_pin);
  vw_set_ptt_inverted(true); // Required for DR3100
  vw_setup(2000);   // Bits per sec
  pinMode(led_pin, OUTPUT);
  vw_rx_start();       // Start the receiver PLL running

main: //Label to make sure it returns to receiving loop incase wrong message recieved.

  vw_wait_rx();
  if (vw_get_message(buf, &buflen)) {
    if (buf[0] != 52) {goto main;} // Check if the right msg or else jump back to label
    Serial.println(buf[0]);
    //if(buf[0] == 49) {Serial.println("Perfect");digitalWrite(led_pin, HIGH);}
    }
```

Figure 1: Code for (a) Node 1 triggers a signal (b) Node 2 and 3 wait till they recieve the right message

Then comes the part where the nodes communicate with each other the following are the slots for each of the nodes recieving and transmitting,

Node	Transmit (s)	Recieve (s)
1	0-2	2-4, 4-6
2	2-4	0-2, 4-6
3	4-6	0-2, 2-4

As mentioned above the slots can be made smaller and the communication can be made more smooth but the problem lies in the fact that this project is to demonstrate and the synchronization is not perfect so the slots are of 2 seconds each. Ideally they should be as short as possible. The following is the example code for the the schedulling (replicated for the other nodes),

```
byte count = 1;
double milli = millis();

void loop()
{
  char msg[7] = {'2'};
  uint8_t buf[VW_MAX_MESSAGE_LEN];
  uint8_t buflen = VW_MAX_MESSAGE_LEN;

  //msg[0] = count;
  if (millis() - milli >= 6000) {milli = millis();} // Timing the slots to repeat
                                                    // after every 6 seconds
  if (millis() - milli >= 2000 && millis() - milli <= 3980){ //Slot 1 to trasnmit
    //digitalWrite(led_pin, HIGH); // Flash a light to show transmitting
    vw_set_ptt_inverted(true); // Required for DR3100
    vw_rx_stop();
    //Serial.println("2 Tx");
    vw_send((uint8_t *)msg, 7);
    vw_wait_tx(); // Wait until the whole message is gone
    //digitalWrite(led_pin, LOW);
  }
```

```
if (millis() - milli >= 10 && millis() - milli <= 1980){ //Slot 1 to recieve from node 2
  //digitalWrite(led_pin, HIGH); // Flash a light to show transmitting
  vw_rx_start();       // Start the receiver PLL running
  //Serial.println("2 Rx");
  if (vw_get_message(buf, &buflen)) {
    Serial.println(buf[0]);
    if(buf[0] == 49) {Serial.println("Perfect");digitalWrite(led_pin, HIGH);}
    } // Non-blocking
  //digitalWrite(led_pin, LOW);
}

if (millis() - milli >= 4000 && millis() - milli <= 5980){ //Slot 1 to recieve from node 3
  //digitalWrite(led_pin, HIGH); // Flash a light to show transmitting
  vw_rx_start();       // Start the receiver PLL running
  //Serial.println("2 Rx");
  if (vw_get_message(buf, &buflen)) {
    Serial.println(buf[0]);
    if(buf[0] == 51) {Serial.println("Perfect");digitalWrite(led_pin, HIGH);}
    } // Non-blocking
  //digitalWrite(led_pin, LOW);
}

delay(10);
digitalWrite(led_pin, LOW);
//count = count + 1;
```

Figure 2: Code for the scheduling algorithm

Notice also the guard band we have left in time slots of nearly 10 ms. This gaurd band is ideally in the order of nano seconds and is left to ensure multipaths do not reach however here it is left since the nodes are not perfectly synchronized.

5 Circuit Diagram

The following is the circuit diagram of one such node. The same circuit is replicated for each node.

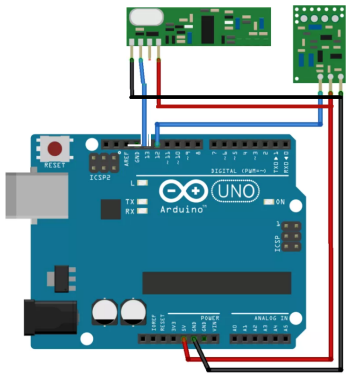

Figure 3: Circuit Diagram of the nodes of the Solution presented

6 Choice of Material

The following items were,

- Arduinos
- RF Modules Pairs, 433 MHz
- Connectors

Arduino paired with 433 MHz RF modules provide a very cheap base for demonstrating any sort of scheduling algorithm. There exist expensive alternatives as well where the module has built-in transmitter and receiver within the same module and it is easy to code since the entire interrupt service routine is not required to be defined but these items are cost effective. We have used built-in LEDs for the Arduinos.

7 Conclusions and Future Recommendations

The project overall went fine except for a few improvements that coul be made. Initially we were using radio head library but that did not give us freedom to use both the transmitter and the reciever on the same arduino since once the transmitter transmits it needs to be manually disabled or else it wont allow the reciever to work using some ISR. However then we used virtual wire library which gives us flexibility to enable and disable the transmitter manually although increases the length of the code.

Another feature that could be added is the acknowledgement protocol where the nodes send an acknowledgement after recieving the message. This would ensure data is not lost if acknowledgement is not sent then actions may be taken to investigate the breach.

Another improvement would be to cipher the message before sending them. Right now we have just programmed the LED not to blink if the message sent is not the right one with regard to the slot. But in future in order to prevent the breach a method could be to cipher at the transmitter and decode using pseudo random key at the reciever.

The project successfully demonstrated scheduling algorithms in an Ad-Hoc network.