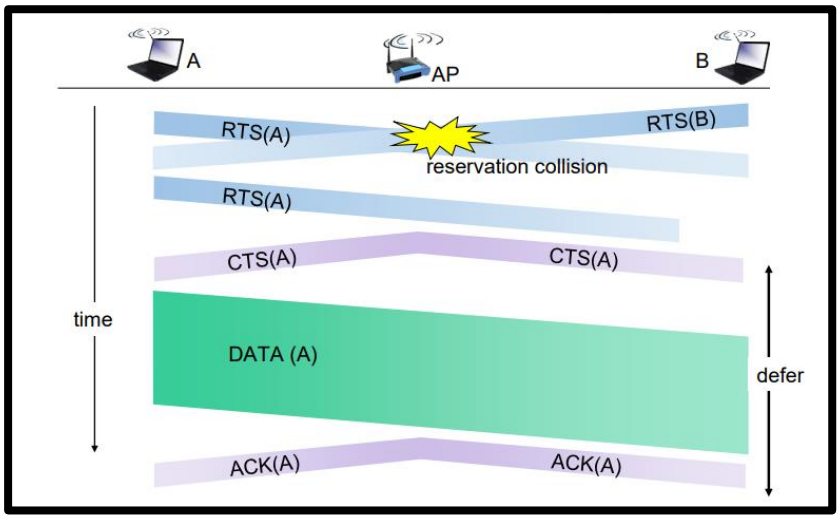
NSCAP FINAL PROJECT

110550039 劉詠

Topic

簡易實作 **CSMA/CA**

Framework



I use the above picture to show my basic framework, there are many hosts, and a BS, when host A want to send to host B, it should first sense the medium, and if idle, it will send RTS to BS, and BS will send CTS to all other hosts to let them know there is someone going to send something. And host A will then send the whole data to host B when it receives CTS, and host B will send ACK back to host A if it receives the data from A. On the other hand, when host A sense the medium busy at the beginning, it will start a random backoff time countdown, when the countdown ends, host A will re-sense the medium.

Implementation

In this project, I will need to implement the following mechanisms:

1. RTS/CTS sending
2. Random Backoff Mechanism
3. Simple Collision Simulation

a. I use **socket** for communication, which allow each host has a UDP socket bound to a specific port (based on the host ID, given when running the program in the terminal). Then, using **select** for **Non-blocking I/O**, the select function is used to monitor multiple file descriptors (including sockets and standard input) and notify when they are ready for reading or writing. This

allows program to handle multiple I/O events simultaneously without needing multiple threads or processes, the code is as following:

```
read_sockets, _, _ = select.select(host.inputs + [sys.stdin], [], [], 1)
```

And it also uses the following to handle events based on which file descriptor received the event, if “self.server_socket” is readable, it means network data has been received, and the “handle_data()” function is called to process this data. If “sys.stdin” is readable, it means there is a user command, and the “process_command()” function is called to process this command.

```
for sock in read_sockets:
    if sock == host.server_socket:
        data, _ = host.server_socket.recvfrom(1024)
        host.handle_data(data)
    elif sock == sys.stdin:
        # Handle user input
        command = input()
        host.process_command(command)
```

b. Next, when a host enter command like “ send ‘destination_id’ ‘message’ ”, the host will call “process_command()” to handle that, for each host, there is a queue (**send_queue**) holding the message of the command. Since host should check if the medium is idle for DIFS time, I make a function call “check_medium_idle()” which will return the boolean value about whether the medium is idle or not. And if host want to send something, it should first call this function, and if it is true, host should wait for DIFS time (which here I set it for 3 seconds, for easier demoing), then call “check_medium_idle()” again, if at this time, the medium is still idle, then host can send RTS to BS. However, if not, host will start random backoff countdown.

   But how “check_medium_idle()” check the medium is idle? I make every host to keep its own status of transmitting and a vector of all the other hosts’ status. In “broadcast_transmitting_status()”, I make every host to keep calling this function, which will send host’s status to the BS, and BS will pass on that to other hosts.

c. When a host is able to send RTS, the status “transmitting” of that host will become true and it will send the packet with type “RTS”, if the BS receives that, BS will call “handle_data()” to send CTS to all the other hosts, note that in “handle_data()”, it deal with different type of packets, such as RTS, CTS, ACK, and so on. And the CTS will be received by the hosts, they will all call “handle_data()”, but only the one that wants to send can send the whole data. The data is kept in the host’s “send_queue”, pop that out and send. When receiver receives that packet, it will wait for SIFS time, and send ACK back to

sender, finally, the “transmitting” of the host(sender) will become false.

d. Next, we are going to talk about the most important and complicate part, random backoff mechanism. I create a thread to handle random backoff countdown, in order to allow any other host can still execute when someone is in backoff mechanism. When a host senses the medium busy, it will enter backoff mechanism, which will call “handle_backoff()”, as the following shows:

```
def handle_backoff(self, destination_id):
    print("----------Start Random Backoff Timer.----------")
    self.backoff_timer = random.randint(7, 10)
    if self.last_backoff_time == 0:
        self.last_backoff_time = self.backoff_timer
    if self.backoff_timer < self.last_backoff_time:
        self.backoff_timer = (self.last_backoff_time + 1)
    self.continue_backoff(destination_id)
```

In the above code, it will choose a random number to be counted down. The reason that I chose from 7 to 10 is for demoing it clearly in the class. Each host will also keep an information about its last random backoff time, in order to wait for longer when the host enter backoff mechanism again in some situation. Then, we start “continue_backoff()”, which is as the following:

```
def continue_backoff(self,destination_id):
    if self.backoff_timer > 0:  # If countdown not finished
        if self.check_medium_idle():
            print("Remain Time: ", self.backoff_timer)
            time.sleep(1)  # Simulated slot time
            self.backoff_timer -= 1
            self.continue_backoff(destination_id)  # Continue counting down
        else:
            print("----------Medium Is Busy. Backoff Paused.----------")
            self.pause_backoff(destination_id)
    else:
        print("----------Backoff Countdown Finished. Try Sending Packet.----------")
        if self.check_medium_idle():
            time.sleep(3)#DIFS time
            if self.check_medium_idle():
                packet = {'type': 'RTS', 'source': self.host_id, 'destination': destination_id}
                self.transmitting = True
                self.other_status[self.host_id] = True
                self.broadcast_transmitting_status()
                self.send_packet(0, json.dumps(packet).encode()) #指定0為BS
            else:
                self.backoff_thread = threading.Thread(target=self.handle_backoff, args=(destination_id,))
                self.backoff_thread.start()
        else:
            self.backoff_thread = threading.Thread(target=self.handle_backoff, args=(destination_id,))
            self.backoff_thread.start()
```

Since timer will counts down when medium is idle, we need to keep checking the medium’s status. However, since the backoff mechanism is done in thread, how to synchronize the situation is a problem. In the above code, we can see that I keep check whether the medium is idle of not, I create a thread event about “transmitting” status:

```
    self.other_status[host_id] = packet['transmitting']
    self.status_changed_event.set()  # Signal that status has changed
```

In “continue_backoff()”, it will count down when the medium is idle, but when it senses that the medium is busy, it will call another

function ”pause_backoff()” to wait for that event as the comment shows, while the medium is busy, it will keep waiting until the medium is idle, it calls “continue_backoff()” to resume countdown:

```
def pause_backoff(self,destination_id):
    while not self.check_medium_idle():
        #print(self.other_status)
        time.sleep(1)  # Check medium state periodically
        self.status_changed_event.wait()  # Wait for status change event
        self.status_changed_event.clear()  # Clear the event
    print("----------Medium Become Idle. Resuming Backoff.----------")
    self.continue_backoff(destination_id)  # Resume counting down
```

When countdown finishes, it still senses the medium to check sending status, if the medium is still busy, it will enter random backoff mechanism again. The above is the whole process of backoff mechanism.

e. Although the project’s topic is CSMA/CA, there is still a chance that a collision happens. In order to simulate the situation that when there is a collision, how the recovery will be done, I first make a host to send when someone is sending to create collision, I know that this is not correct in the real world, but I just want to make a “collision”. Now, the collision happens, how can I let hosts to know that? I use the “transmitting” status vector of each hosts, let hosts to know collision by calling “collision_detect()” from time to time, when a host is sending, and found that there is someone sending also, it will print out “Collision Happened”, and enter random backoff mechanism, then resend data in queue after countdown.

From point a to point e is the whole implementation of my project.

Experiments

Following are some experiments I had done, assume host 0 as BS:

1. Simple RTS/CTS sending (with 2 hosts, one is sender and the other is receiver) The sender will first send RTS to BS, BS will reply the sender with CTS, sender will then send the whole data to the receiver, and receiver response with an ACK:

```
yong@DESKTOP-KISD6CN:/mnt/c/Users/User/Documents/python_program/nscap_project$ python3 csmaca.py 0
BS received RTS from Host 1
```

```
yong@DESKTOP-KISD6CN:/mnt/c/Users/User/Documents/python_program/nscap_project$ python3 csmaca.py 1
send 2 hi
Host 1 received CTS from BS
Host 1 received ACK from Host 2
```

```
yong@DESKTOP-KISD6CN:/mnt/c/Users/User/Documents/python_program/nscap_project$ python3 csmaca.py 2
Host 2 received DATA from Host 1
```

The procedure is clear in the above pictures.

2. With 3 hosts, host 1 and 3 be sender and 2 be receiver. When host 1 first sends, if host 3 sends at this time, it should run into random backoff procedure, after that, it senses the medium again and send:
   In host 1:

```
send 2 hi
Host 1 received CTS from BS
Host 1 received ACK from Host 2
```

   In host 2:

```
Host 2 received DATA from Host 1
Host 2 received DATA from Host 3
```

   In host 3:

```
send 2 hi
----------Start Random Backoff Timer.----------
----------Medium Is Busy. Backoff Paused.----------
----------Medium Become Idle. Resuming Backoff.----------
Remain Time:  9
Remain Time:  8
Remain Time:  7
Remain Time:  6
Remain Time:  5
Remain Time:  4
Remain Time:  3
Remain Time:  2
Remain Time:  1
----------Backoff Countdown Finished. Try Sending Packet.----------
Host 3 received CTS from BS
Host 3 received ACK from Host 2
```

   We can see that host 3 enter backoff mechanism, and resend after that.

3. With 3 hosts, host 1,2,3 are all senders, host 1 be the receiver for 2, host 2 be that for host 3 and for host 1, let host 1 send first, then host 2, then host 3. We will see the random backoff procedure will pause and restart at this certain situation:
   In host 1:

```
send 2 hi
Host 1 received CTS from BS
Host 1 received ACK from Host 2
Host 1 received DATA from Host 2
```

   In host 2:
   We see that when host 2 is in random backoff mechanism, and actually the current medium is idle, when host 3 send, the medium becomes busy, the backoff mechanism is paused, and resume when the medium becomes idle again.

```
Host 2 received DATA from Host 1
send 1 hi
----------Start Random Backoff Timer.----------
----------Medium Is Busy. Backoff Paused.----------
----------Medium Become Idle. Resuming Backoff.----------
Remain Time:  7
Remain Time:  6
Remain Time:  5
Remain Time:  4
Remain Time:  3
Host 2 received DATA from Host 3
----------Medium Is Busy. Backoff Paused.----------
----------Medium Become Idle. Resuming Backoff.----------
Remain Time:  2
Remain Time:  1
----------Backoff Countdown Finished. Try Sending Packet.----------
Host 2 received CTS from BS
Host 2 received ACK from Host 1
```

In host 3:

```
send 2 hi
Host 3 received CTS from BS
Host 3 received ACK from Host 2
```

4. Simulate a situation of the occurrence of collision, host 3 and 4 are all senders with host 2 be the receiver, host 4 will assume to do sending when host 3 is using the medium. Then we will see that there is a collision happened, host 3 will enter the random backoff time and restart sending data.

   In host 3:

```
send 2 hi
Host 3 received CTS from BS
----------Collision Happened.----------
----------Start Random Backoff Timer.----------
Remain Time:  10
Remain Time:  9
Remain Time:  8
Remain Time:  7
Remain Time:  6
Remain Time:  5
Remain Time:  4
Remain Time:  3
Remain Time:  2
Remain Time:  1
----------Backoff Countdown Finished. Try Sending Packet.----------
Host 3 received CTS from BS
Host 3 received ACK from Host 2
```

   Since I use host 4 to make collision, we can see that host 3 senses that there is a collision happened, it enters random backoff mechanism and resend after that.

5. With 3 hosts, host 1,2,3 are all senders, host 1 be the receiver for 2, host 2 be that for host 3 and for host 1, let host 1 send first, then host 2, then host 3. We will see the random backoff procedure in host 2, but when host 2 is going to restart, host 3 send at this time, the host 2 will enter backoff mechanism again:

   In host 1:

```
send 2 hi
Host 1 received CTS from BS
Host 1 received ACK from Host 2
Host 1 received DATA from Host 2
```

   In host 2:

```
Host 2 received DATA from Host 1
send 1 hi
----------Start Random Backoff Timer.----------
----------Medium Is Busy. Backoff Paused.----------
----------Medium Become Idle. Resuming Backoff.----------
Remain Time:  9
Remain Time:  8
Remain Time:  7
Remain Time:  6
Remain Time:  5
Remain Time:  4
Remain Time:  3
Remain Time:  2
Remain Time:  1
Host 2 received DATA from Host 3
----------Backoff Countdown Finished. Try Sending Packet.----------
----------Start Random Backoff Timer.----------
----------Medium Is Busy. Backoff Paused.----------
----------Medium Become Idle. Resuming Backoff.----------
Remain Time:  7
Remain Time:  6
Remain Time:  5
Remain Time:  4
Remain Time:  3
Remain Time:  2
Remain Time:  1
----------Backoff Countdown Finished. Try Sending Packet.----------
Host 2 received CTS from BS
Host 2 received ACK from Host 1
```

   We can see that when host 2 is about to resend, it receives the data from host 3, then it senses the medium become busy now, then it enters backoff mechanism again.

   In host 3:

```
send 2 hi
Host 3 received CTS from BS
Host 3 received ACK from Host 2
```

Conclusion

In this project, the hardest part to be implemented is random backoff mechanism, you can see that I spent a lot of pages to show the experiments about it. However, I know that there are more techniques that CSMA/CA has done in the real world, all I have done is only based on what I have learned in the class, and there are a lot of protocols try hard solving lots of issues, hope that through this project, I can have an eye to others and deepen my impression of those ones.

Prospect

Since this is a simple implementation of CSMA/CA, there may be a lot of things to be improved. Here are points that I think they can be done in the future:

1. Send actual data not just message in the real world.
2. Implement random backoff mechanism in exponential time.
3. Allow dynamic number of hosts.
4. Can simulate collision more wisely.
5. Synchronization can be done more efficiently and immediately.
6. ……

The above is the whole of my report of the final project, thank you for reading it.