

CS536: Data Communication and Computer Networks

LAB3 ANSWERS

Zichen Wang
wang4113@purdue.edu

October 14, 2019

1 Problem 1

1.1 Basic Function

1.1.1 Setup and test

The server is running on Pod2-1 and one client is running on escher01. The file size is set to 230000000 bytes which is about 220 MB. Table 1 shows the result of experiments.

The apps works well with empty file or the file does not exist. If the requested file is empty or does NOT exist on the server, the client will NOT create a new empty file under /tmp.

```
escher01 72 $ ./myfetchfile empty 128.10.25.206 55555
the requested file is empty
escher01 73 $ ./myfetchfile aaaaa 128.10.25.206 55555
the requested file does not exist (or is inaccessible) on the server
```

1.1.2 Findings

The speed and completion time is stable for several consecutive experiments. However, sometimes the completion time is longer than 3 seconds. It is probably because other apps might also use the connection between these two machines.

Exp. No.	File Size (bytes)	Completion Time (ms)	Speed (bps)
1	230000000	2927.743896	628470270.985610
2	230000000	3025.374023	608189263.788730
3	230000000	2851.117920	645360890.597755

Table 1: Basic function test

Machine	File Size (bytes)	Completion Time (ms)	Speed (bps)
escher01	230000000	5569.471924	330372434.795451
escher02	230000000	5662.501953	324944700.281215

Table 2: Test with 2 clients

Machine	File Size (bytes)	Completion Time (ms)	Speed (bps)
escher01	230000000	9853.185059	186741646.387245
escher02	230000000	12494.430908	147265610.856431
escher06	230000000	12775.395020	144026857.657785
escher07	230000000	13259.498047	138768450.622733
escher08	230000000	12311.780029	149450363.442294
escher09	230000000	11788.566162	156083443.456771

Table 3: Test with 6 clients

1.2 Multiple Clients

The file size is still 230000000 bytes and clients are running on different escher machines.

1.2.1 2 clients

2 clients are running on escher01 and escher02 respectively. They get started almost at the same time. Table 2 shows the result of this experiment.

From the result, we can see that the completion time of each machine is doubled compared with the baseline, and the speed decreases accordingly. However, the completion time is less than the double of the baseline, because we cannot start 2 clients exactly at the same time. When we start one client, the data will transmit immediately and the speed would start to decrease after starting another client. In the following experiments, we can hardly let the clients start exactly at the same time.

1.2.2 6 clients

6 clients are running on escher01, escher02, and escher06 to escher09. escher04 to escher06 are NOT used because the baseline completion time on these machines are much longer than that on escher01. We want to make sure the network status is similar on those client machines. 6 clients get started one by one as quickly as possible. Table 3 shows the result.

From the result, we can see the average completion time is longer than that with 2 clients. It about 4 fold of the baseline time. Since we start the clients one by one, the client that we start in the middle will have the lowest speed. The clients start first will have highest speed. The speed goes up from the middle point because previous clients might finish at that time.

Machine	File Size (bytes)	Completion Time (ms)	Speed (bps)
escher01	230000000	10840.841064	169728528.354993
escher02	230000000	16157.664062	113877847.248379
escher06	230000000	22277.796875	82593445.407739
escher07	230000000	21583.452881	85250493.058585
escher08	230000000	22801.901123	80695025.825730
escher09	230000000	21412.701904	85930304.742662
escher10	230000000	22105.424072	83237489.314152
escher12	230000000	20770.479004	88587268.481096
escher13	230000000	19701.252930	93395075.255713
escher14	230000000	18739.573975	98187931.192729

Table 4: Test with 10 clients

1.2.3 10 clients

10 clients are running on escher01, escher02, escher06 to escher10, and escher12 to escher14. 10 clients get started one by one as quickly as possible. Table 4 shows the result.

From the result, we can see the similar tendency as 6 clients. The completion time is much longer than the baseline. As we start the clients one by one, the tendency of speed of each client is similar to that with 6 clients.

1.3 Block Size

1.3.1 Test of different block sizes

The server is running on Pod2-1 and one client is running on escher01. The file size is set to 230000000 bytes as baseline. Table 5 shows the result of experiments.

1.3.2 Findings

From the result, we can see that there is little difference among each block size. My guess is that the speed is stable if the block size is larger than 1472, and the speed will increase if the block size is smaller than 1472. The smaller the block size is, the higher the speed is. However, if the block size is too small (<100 bytes), the speed will decrease drastically. I do several additional experiments to see whether it is true or not.

Table 6 and 7 proves my guess. From these experiments, we can see that when the block size is large enough, the completion time and speed is stable. It is because the Ethernet frame has maximum payload. One block will be split into several frames and the overhead is at the network transmission. With the decrease of block size, the speed will increase. However, if the block size is less than 100, the speed will decrease a lot, because the most overhead is on the OS. OS has to perform a lot of read and write system calls. If the block size is less than 46 (the minimum payload size of Ethernet frame), the speed is very slow. The optional value might be around 100 or slightly more than 100, but the difference between 100 and 1472 is not too much.

Block Size (bytes)	File Size (bytes)	Completion Time (ms)	Speed (bps)
500	230000000	2877.263916	639496429.145086
1000	230000000	2821.339111	652172577.416202
5000	230000000	2930.586182	627860737.052243
10000	230000000	2855.400879	644392881.431347

Table 5: Test with various block size

Block Size (bytes)	File Size (bytes)	Completion Time (ms)	Speed (bps)
50000	230000000	2872.631104	640527771.821500
100000	230000000	2931.146973	627740613.884183
500000	230000000	2882.450928	638345646.163785
1000000	230000000	2918.814941	630392826.176746

Table 6: Test with large block size

Block Size (bytes)	File Size (bytes)	Completion Time (ms)	Speed (bps)
400	230000000	2677.271973	687266747.193580
300	230000000	2609.917969	705003000.872573
200	230000000	2636.437012	697911610.185014
100	230000000	2498.822998	736346672.588726
90	230000000	2512.958984	732204549.075690
80	230000000	2807.161133	655466470.553652
70	230000000	3184.382080	577820108.808946
60	230000000	3646.682129	504568244.491294
50	230000000	4316.246826	426296287.979415
10	230000000	20711.419922	88839877.079438

Table 7: Test with small block size

File Size (bytes)	Duplicate Bytes	Completion Time (ms)	Speed (bps)
5400000	55898	4110.797119	10508910.741144
5400000	69137	4020.311035	10745437.261503
5400000	85318	4106.666016	10519482.187164

Table 8: Test of jumping the gun with dropwhen -1

2 Problem 2

2.1 Matching Problem

To solve the matching problem, we need to refine the protocol of sequence number. Since we use one byte to represent the sequence number, the total number of sequence numbers is 256. We treat it as a 1-byte unsigned integer, so the range is $[0, 255]$. In my solution, the range of sequence number is split into 3 groups. Group $[0, 3, 6, 9, \dots]$ and $[1, 4, 7, 10, \dots]$ are used to indicate this packet is not the last packet and each number from the same group represents the same packet. Group $[2, 5, 8, 11, \dots]$ is used to indicate this is the last packet. When an ACK is delayed and timeout happens, the re-transmitted sequence number is added by 3 in order to be in the same group. In a word, sequence number modulo 3 is the type of this packet (group). Sequence number divided by 3 is the counter of re-transmission.

We record the start time of each sequence number from one group and save the start time into an array. The maximum times of retry is 80. If the number of re-transmissions exceeds 80, the server will report it and terminate. On the client side, if the received sequence number modulo 3 is equal to expected sequence number, we consume the content. Otherwise, it would be a duplicate packet. The client will send ACK with the received sequence number. On the server side, if the ACK contains sequence number that is in the current sequence number group, i.e., it is one of the expected ACK and we will record the end time. The new RTT is this end time minus the start time corresponding to the sequence number (sequence number / 3) in the array.

2.2 Test of Jumping the gun

2.2.1 dropwhen is -1

The server is running on the Pod2-1 machine and client is running on the escher01 machine. Table 8 shows the result.

2.2.2 dropwhen is 10

The server is running on the Pod2-1 machine and client is running on the escher01 machine. Table 9 shows the result.

2.2.3 Findings

There might be more duplicate bytes than the fixed timeout, because sometimes the RTT is not very stable. If a long RTT appears after some short RTTs, a duplicate packet will happen. On the

File Size (bytes)	Duplicate Bytes	Completion Time (ms)	Speed (bps)
5400000	692841	4660.640869	9269111.526279
5400000	664892	4601.822998	9387584.011453
5400000	686957	4524.563965	9547881.372806

Table 9: Test of jumping the gun with dropwhen 10

other hand, the overall result is more stable. The drawback of fixed timeout is that this timeout cannot reflect the following network condition. If the connect becomes very slow later, there would be a lot of duplicate packets for fixed timeout. However, adaptive RTT estimation will update the timeout accordingly. In our experiments, there is not too much difference between fixed timeout and adaptive RTT estimation.

3 Problem 3

The file size given to ‘myftpd’ is **150** bytes and the dropwhen is set to **-1**, so there are exactly **5** data packets and **5** ACK packets under the block size 30 bytes. Both ‘myftp’ and ‘myftpd’ are running on machine Pod3-4.

Figure 1 shows the screen shot of Wireshark that contains these 10 UDP packets. From the figure, we can see that the size of frames containing ACK is 43 bytes, which is smaller than the minimum Ethernet frame size (64 bytes). Padding should be applied to Ethernet’s payload as discussed in the class. The first 6 bytes is the destination MAC address. The next 6 bytes is the source MAC address. Then, a two-byte 0x0800 indicates this frame is of type IPv4. The following 28 bytes is the header of IP and the header of UDP. Then, the UDP data can be seen.

Figure 2 and 3 show the last two stop-and-wait application packets containing sequence number 2. We can see 02 is at the beginning of UDP data.

Next, we discuss the captured raw data in hexadecimal form. Let’s use the first packet. The raw data is as follows.

```

e2 72 9f 80 14 01 f6 22 19 06 55 84 08 00 45 00
00 3b 91 b7 40 00 40 11 25 a7 c0 a8 01 02 c0 a8
01 01 d3 37 c9 03 00 27 83 8c 00 33 33 33 33 33
33 33 33 33 33 33 33 33 33 33 33 33 33 33 33
33 33 33 33 33 33 33 33 33

```

The first 6 bytes 0xe2 0x72 0x9f 0x80 0x14 0x01 is the destination MAC address e2:72:9f:80:14:01. The second 6 bytes 0xf6 0x22 0x19 0x06 0x55 0x84 is the source MAC address f6:22:19:06:55:84. Then, 0x08 0x00 identifies IPv4 (0x0800) as the payload type of the Ethernet frame. The next 28 bytes are related to IP layer, UDP port numbers and checksum. The 1-byte application layer sequence number followed by the checksum is 0x00. Then, 30-byte application layer payload is 0x33 repeated 30 times.

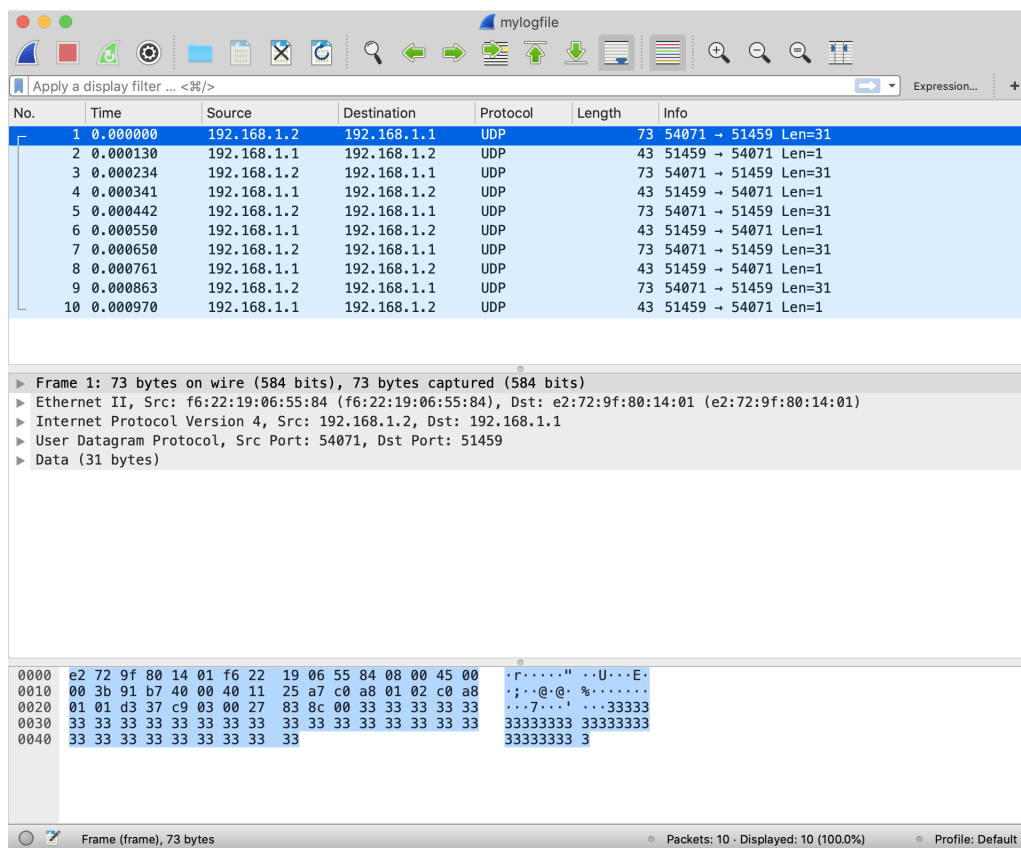


Figure 1: 10 UDP packets

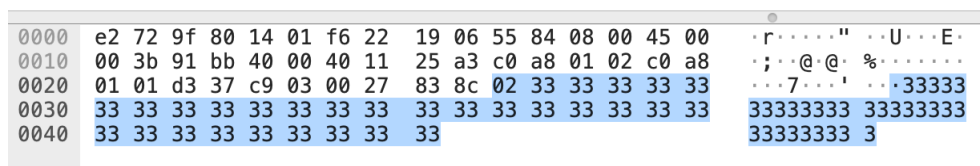


Figure 2: The last packet

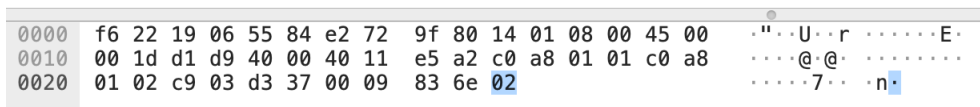


Figure 3: The last packet

4 Bonus Problem

4.1 ICMP packets capture experiment

Figure 4 shows the detail of en0 interface in my laptop. The IP address of en0 interface is 192.168.27.19 and the MAC address is f0:18:98:b4:2b:f7. I performed ping to pod2-1 10 times, so there would be 20 ICMP packets captured by Wireshark. Figure 5 shows the packets.

```
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
    options=400<CHANNEL_IO>
    ether f0:18:98:b4:2b:f7
    inet6 fe80::10ed:1ea9:2627:764a%en0 prefixlen 64 secured scopeid 0x6
    inet 192.168.27.19 netmask 0xfffff000 broadcast 192.168.31.255
    nd6 options=201<PERFORMNUD,DAD>
    media: autoselect
    status: active
```

Figure 4: The detail of en0 interface

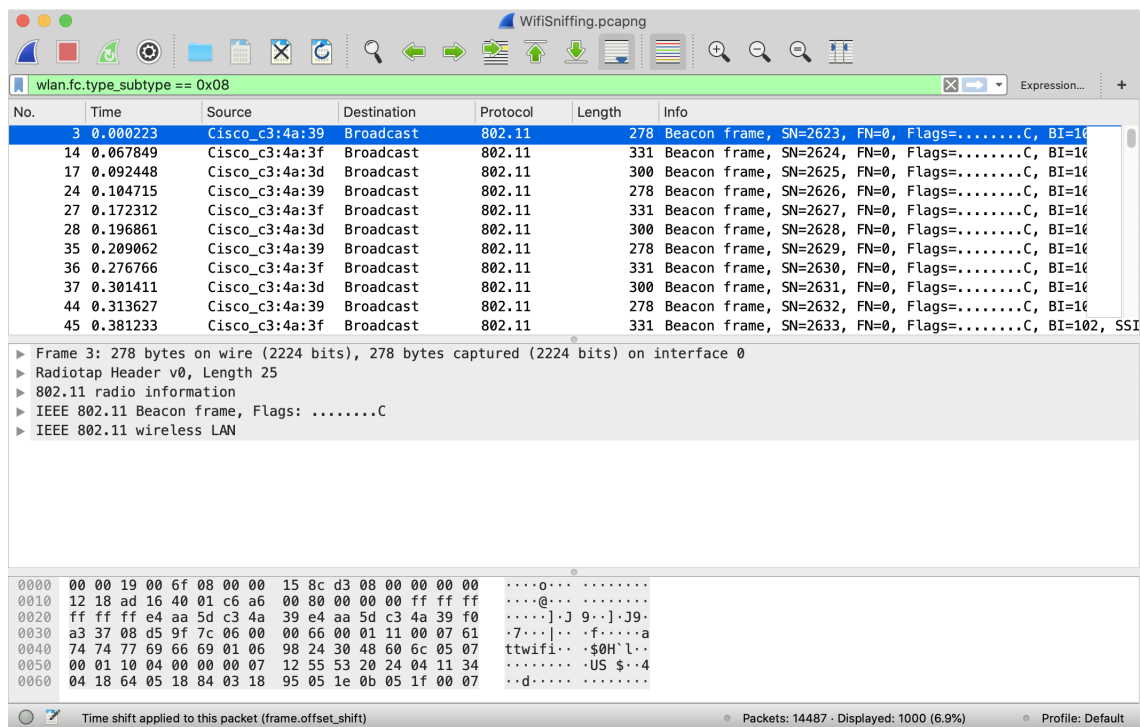
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.27.19	128.10.25.206	ICMP	98	Echo (ping) request id=0x3626, seq=0/0, ttl=64 (reply in 2)
2	0.012859	128.10.25.206	192.168.27.19	ICMP	106	Echo (ping) reply id=0x3626, seq=0/0, ttl=61 (request in 1)
7	1.000070	192.168.27.19	128.10.25.206	ICMP	98	Echo (ping) request id=0x3626, seq=1/256, ttl=64 (reply in 8)
8	1.016720	128.10.25.206	192.168.27.19	ICMP	106	Echo (ping) reply id=0x3626, seq=1/256, ttl=61 (request in 7)
9	2.003089	192.168.27.19	128.10.25.206	ICMP	98	Echo (ping) request id=0x3626, seq=2/512, ttl=64 (reply in 10)
10	2.012472	128.10.25.206	192.168.27.19	ICMP	106	Echo (ping) reply id=0x3626, seq=2/512, ttl=61 (request in 9)
15	3.004703	192.168.27.19	128.10.25.206	ICMP	98	Echo (ping) request id=0x3626, seq=3/768, ttl=64 (reply in 16)
16	3.014565	128.10.25.206	192.168.27.19	ICMP	106	Echo (ping) reply id=0x3626, seq=3/768, ttl=61 (request in 15)
17	4.009819	192.168.27.19	128.10.25.206	ICMP	98	Echo (ping) request id=0x3626, seq=4/1024, ttl=64 (reply in 18)
18	4.021102	128.10.25.206	192.168.27.19	ICMP	106	Echo (ping) reply id=0x3626, seq=4/1024, ttl=61 (request in 17)
19	5.014419	192.168.27.19	128.10.25.206	ICMP	98	Echo (ping) request id=0x3626, seq=5/1280, ttl=64 (reply in 20)
20	5.024214	128.10.25.206	192.168.27.19	ICMP	106	Echo (ping) reply id=0x3626, seq=5/1280, ttl=61 (request in 19)
23	6.019513	192.168.27.19	128.10.25.206	ICMP	98	Echo (ping) request id=0x3626, seq=6/1536, ttl=64 (reply in 24)
24	6.043349	128.10.25.206	192.168.27.19	ICMP	106	Echo (ping) reply id=0x3626, seq=6/1536, ttl=61 (request in 23)
25	7.019611	192.168.27.19	128.10.25.206	ICMP	98	Echo (ping) request id=0x3626, seq=7/1792, ttl=64 (reply in 26)
26	7.028990	128.10.25.206	192.168.27.19	ICMP	106	Echo (ping) reply id=0x3626, seq=7/1792, ttl=61 (request in 25)
27	8.019706	192.168.27.19	128.10.25.206	ICMP	98	Echo (ping) request id=0x3626, seq=8/2048, ttl=64 (reply in 28)
28	8.034042	128.10.25.206	192.168.27.19	ICMP	106	Echo (ping) reply id=0x3626, seq=8/2048, ttl=61 (request in 27)
29	9.021224	192.168.27.19	128.10.25.206	ICMP	98	Echo (ping) request id=0x3626, seq=9/2304, ttl=64 (reply in 30)
30	9.035817	128.10.25.206	192.168.27.19	ICMP	106	Echo (ping) reply id=0x3626, seq=9/2304, ttl=61 (request in 29)

Figure 5: The captured ICMP packets

Let's analyze the first request ICMP packet send from my laptop to Pod2-1. Figure 6 shows the hex of this frame. 00:00:0c:9f:f0:01 is the MAC address of one of the routers in LWSN. The MAC address of my en0 interface is f0:18:98:b4:2b:f7 as I showed before. The type is 0x0800 which is IPv4 packet. The next 20 bytes are related to IP protocol. We may find the source IP address is 192.168.27.19 and the destination IP address is 128.10.25.206. The rest bytes are ICMP payload. The first byte is 8 which is the Echo (ping) request type. After that, the header includes some control information. The last 48 bytes are ICMP data.

0000	00 00 0c 9f f0 01 f0 18	98 b4 2b f7 08 00 45 00+...E.
0010	00 54 51 65 00 00 40 01	b3 b0 c0 a8 1b 13 80 0a	·TQe·@·.....
0020	19 ce 08 00 92 fc 36 26	00 00 5d a4 a7 ad 00 036&·].....
0030	3e 85 08 09 0a 0b 0c 0d	0e 0f 10 11 12 13 14 15	>.....!""#\$%
0040	16 17 18 19 1a 1b 1c 1d	1e 1f 20 21 22 23 24 25&'()*+,-./012345
0050	26 27 28 29 2a 2b 2c 2d	2e 2f 30 31 32 33 34 35	67
0060	36 37		

Figure 6: One of the request ICMP packet



The image shows a Wireshark packet capture of IEEE 802.11 Beacon frames. The packet list shows several beacon frames from source MAC Cisco_c3:4a:39 to destination Broadcast on interface 802.11. The selected packet (No. 3) is expanded to show the Radiotap Header, 802.11 radio information, and the IEEE 802.11 Beacon frame structure. The packet details pane shows the frame type as Beacon frame, SN=2623, FN=0, Flags=.....C, BI=10. The packet bytes pane shows the raw data of the beacon frame, including the frame control field, duration, address fields, and the beacon frame body.

No.	Time	Source	Destination	Protocol	Length	Info
3	0.000223	Cisco_c3:4a:39	Broadcast	802.11	278	Beacon frame, SN=2623, FN=0, Flags=.....C, BI=10
14	0.067849	Cisco_c3:4a:3f	Broadcast	802.11	331	Beacon frame, SN=2624, FN=0, Flags=.....C, BI=10
17	0.092448	Cisco_c3:4a:3d	Broadcast	802.11	300	Beacon frame, SN=2625, FN=0, Flags=.....C, BI=10
24	0.104715	Cisco_c3:4a:39	Broadcast	802.11	278	Beacon frame, SN=2626, FN=0, Flags=.....C, BI=10
27	0.172312	Cisco_c3:4a:3f	Broadcast	802.11	331	Beacon frame, SN=2627, FN=0, Flags=.....C, BI=10
28	0.196861	Cisco_c3:4a:3d	Broadcast	802.11	300	Beacon frame, SN=2628, FN=0, Flags=.....C, BI=10
35	0.209062	Cisco_c3:4a:39	Broadcast	802.11	278	Beacon frame, SN=2629, FN=0, Flags=.....C, BI=10
36	0.276766	Cisco_c3:4a:3f	Broadcast	802.11	331	Beacon frame, SN=2630, FN=0, Flags=.....C, BI=10
37	0.301411	Cisco_c3:4a:3d	Broadcast	802.11	300	Beacon frame, SN=2631, FN=0, Flags=.....C, BI=10
44	0.313627	Cisco_c3:4a:39	Broadcast	802.11	278	Beacon frame, SN=2632, FN=0, Flags=.....C, BI=10
45	0.381233	Cisco_c3:4a:3f	Broadcast	802.11	331	Beacon frame, SN=2633, FN=0, Flags=.....C, BI=10, SSI

Figure 7: The captured beacon frames

4.2 Beacon frames capture experiment

I got a trace file from Tinghan's PSO and analyzed in my laptop by Wireshark. Figure 7 shows 3 different beacon frames, because the classroom only has three kinds of WiFi signal. Table 10 shows the information about these beacon frames. From the table, I found that they all transmitted from one access point, because the period is same. Although they transmitted at the same time, their signal strength may be different. Actually, this AP has 3 interfaces because their source MAC addresses are different.

SSID	Strength	Src MAC Address	Dst MAC Address	Frequency
	Channel	PHY Layer Coding	Amount of FEC	Period
attwifi	-58dBm	e4:aa:5d:c3:4a:39	ff:ff:ff:ff:ff:ff	5805MHz
	161	OFDM, 5 GHz spectrum	4 Bytes	0.104448s
PAL3.0	-57dBm	e4:aa:5d:c3:4a:3f	ff:ff:ff:ff:ff:ff	5805MHz
	161	OFDM, 5 GHz spectrum	4 Bytes	0.104448s
eduroam	-57dBm	e4:aa:5d:c3:4a:3d	ff:ff:ff:ff:ff:ff	5805MHz
	161	OFDM, 5 GHz spectrum	4 Bytes	0.104448s

Table 10: The relevant information about beacon frames