Wireless Networks
Assignment 1: Decoding 802.11b Frames from BaseBand Signals

Nanjing University, Spring, 2024

I. Introduction

In this assignment, you will use the physical layer concepts introduced in our lecture to decode IEEE 802.11b [1] data frames from raw baseband signals. The raw baseband signal is sent by a commercial WiFi card and the signal is captured through USRP software radio [2]. You need to develop your own code to decode the raw signal and get the content of the frame. The decoding processes include the following steps: despreading the DSSS code, DBPSK decoding, descrambing and mapping bits to protocol headers. This process involves basic concepts of wireless data communication and signal processing. If you are not familiar with these concepts, please use the Internet to find related tutorials. You possibly will often refer to the IEEE standard IEEE 802.11 about the coding and decoding requirements. You are encouraged to understand important chapters, i.e., Chapter 15, of the standard [1], which is also included in your lab package.

You will use MATLAB (or Octave)/Python to process the baseband signal provided in this assignment. If you are not familiar with MATLAB/Python, please use Google to find "Matlab tutorials" or "Python tutorials" on the web. The framework of the decoding code will be given to you, and you need to fill in your algorithms in certain functions to finish the assignment.

Tip 1: You are encouraged to use Google to find out terms, concepts or even partial solutions for this assignment.

MATLAB source files included in this assignment under directory matlabsource:

- Decoding_main.m: the main script for this assignment. The code is given to you. But you may need to fill in some missed part in it.
- load_baseband.m: source code for loading the signal from the file. The code is given to you.
- find_frame_boundary.m: source code for finding the starting and ending points of each frame. The code is given to you.
- decode_dbpsk.m: source code for DBPSK decoding. You need to finish this by yourself.
- descramble.m: source code for descrambling the binary code. You need to finish this by yourself.
- find_preamble.m: source code for finding the SYNC of the frame. You need to finish this by yourself.
- bittohexstr.m: source code for converting bit stream to Hex strings. You need to finish this by yourself.
- write_pcap_header.m: source code for writing the pcap wireshark file header. The code is given to you.
- write_pcap_frame.m: source code for writing one 802.11 frame to the pcap wireshark file. The code is given to you.

Python source files included in this assignment under directory pythonsource:

- decoding_main.py: the main script for this assignment. The code is given to you. But you may need to fill in some missed part in it.
- lab1step1.py: source code for loading the signal from the file. The code is given to you.

- lab1step3.py: source code for finding the starting and ending points of each frame. The code is given to you.
- lab1step4.py: source code for DBPSK decoding. You need to finish this by yourself.
- lab1step5.py: source code for descrambling the binary code. You need to finish this by yourself.
- lab1step6.py: source code for finding the SYNC of the frame. You need to finish this by yourself.
- lab1step7.py: source code for converting bit stream to Hex strings. You need to finish this by yourself.
- pcapframe.py: source code for writing the pcap wireshark file. The code is given to you.

The sample USRP software radio file is included in the data directory. The file name is lab1.dat, see details about the format of the file in later sections.

II. From Baseband Signal to Bits

A. WiFi Background

Wireless LAN is widely used nowadays. There are many variations of WLAN protocols, see the background introduction in Wikipedia. In this experiment, we will focus on the simplest variation, namely, IEEE 802.11b. IEEE 802.11b operates on 2.4 GHz frequency band and provide transmission data rate of 1, 2, 5.5, and 11 Mbps. In this experiment, we will try to decode frames with the lowest rate, 1 Mbps.

Wireless network cards transmit radio signals in the air and these signals can be captured by equipments such as USRP software radios [2]. In this experiment, we use USRP to capture the radio signal transmitted by wireless cards and save them into a file. Your task is to use the saved data to recover what is sent by the wireless card. The basic process of recovering the data is outlined in Figure 1.

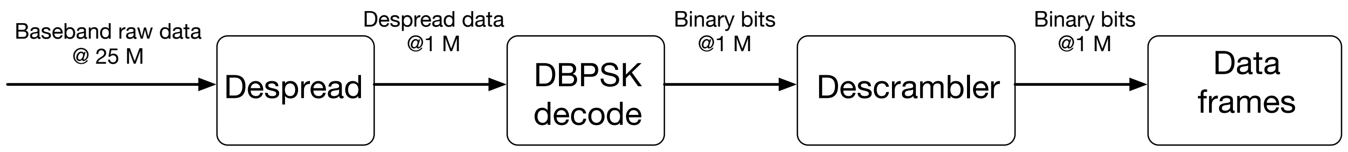


Fig. 1. Basic workflow of the packet decoding process

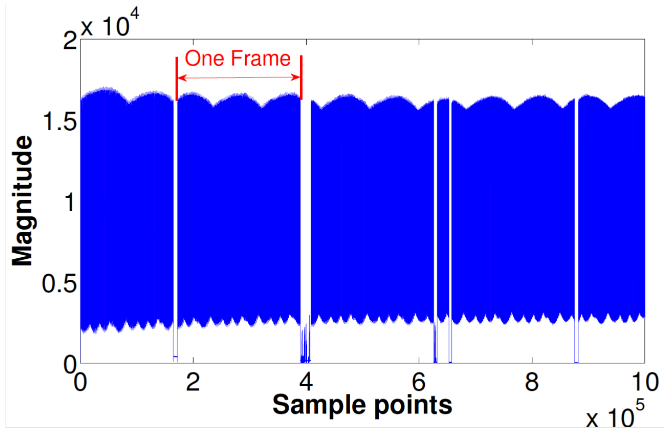
B. WiFi baseband

The baseband signal is given to you in a big dump file called “lab1.dat”. The baseband signal is captured with sampling rate of 25M samples per second. Each sample is a complex number, with both the real part and imaginary part as a 16 bit signed integer. Therefore, each sample takes 4 Bytes in the file, with the first two Bytes as the real part and the second two Bytes as the imaginary part. We have provided the Matlab code for loading the dump file, which is called “load_baseband.m”.

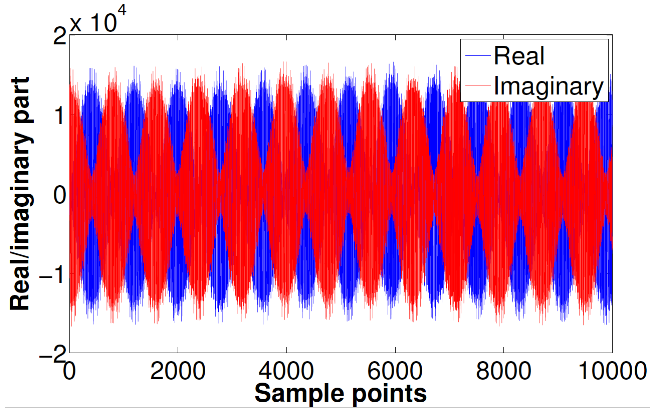
Figure 2(a) shows the plot of the magnitude of the complex baseband signals. We can observe that during certain time period, the magnitude of the signal is large, which indicates that there is someone sending messages. As the wireless channel is quite busy in our case, we see that there is only a very short pause, i.e., white gaps, between consecutive wireless frames. We see that there is one frame between sample point wit index of 1.8×10^5 and 3.8×10^5. Therefore, this frame takes about 2×10^5 samples to transmit. As our sampling rate is 25M samples per second, 2×10^5 samples takes 8 *ms* in time, which is the duration of this frame. The duration of the frame is dependent on the transmission rate and the packet size. You can infer the packet size from the packet duration. As the packet is transmitted at a rate of 1M bit per second, a packet with 8,000 bits (1,000 Bytes) will take about 8 ms to transmit.

Figure 2(b) shows the enlarged plot of the real and imaginary part of the complex signal. We observe that the real and imaginary part continuously change their amplitude.

Questions:
1. Why the magnitude of the complex signal does not change much, while the real and imaginary part frequently change their amplitude?
2. What is the frequency for the amplitude change of real/imaginary part in Figure 2(b)?
3. What is the cause of the amplitude change of the real/imaginary signal?



(a) Magnitude of baseband signal



(b) Real and imaginary part of baseband signal

Fig. 2. Raw baseband signals

C. Despreading the signal

The 802.11b protocol uses Direct Sequence Spread Spectrum (DSSS) modulation for 1 Mbps transmission rate, which is similar to the widely used CDMA technology. Referring to Chapter 15.4.6.3 in IEEE 802.11b standard [1], an 11-chip Barker sequence is used as the PN code. In other words, each of the transmitted bit is spread by a 11 bit Barker code with sequence of {+1, -1, +1, +1, -1, +1, +1, +1, -1, -1, -1}. Recall that our transmission rate is 1 Mbps. This means we transmit 1×10^6 bits per second and each bit takes 1 μs. The spreading code replaces the bit with the Barker sequence when transmitting one data bit. Sometimes, the phase of the transmitted signal is inverted by the modulation process. Figure 3 shows a sample signal that contains two bits. Each data bit is transmitted as 11 small chips of +1 and -1, and the second bit has inverted phases. Each small chip takes 1/11 μs and the data bit takes 1 μs to transmit.

When we received a DSSS code as in Figure 3, we need to despread it and recover the original data bits. A good thing about DSSS code is that they can be recovered easily by performing a correlation operation between the received code and the Barker code. The correlation operation is performed as follows. Define the received data as $x(n)$. For example, the DSSS code in Figure 3 has $x(0) = +1$, $x(1) = -1$, and the whole sequence of $x(n)$ in Figure 3 is {+1, -1, +1, +1, -1, +1, +1, +1, -1, -1, -1, -1, +1, -1, -1, +1, -1, -1, -1, +1, +1, +1}. The Barker code is defined as the 11 bit chip of $b(n), n = 0 \ldots, 10$, as {+1, -1, +1, +1, -1, +1, +1, +1, -1, -1, -1}. Then the correlation at time n is defined as:

$$X(n) = \sum_{i=0}^{10} x(n+i)b(i) \tag{1}$$

Using the sample in Figure 3, you can verify that we have:

$$X(0) = x(0)b(0) + x(1)b(1) + \ldots + x(10)b(10) = 11 \tag{2}$$

$$X(1) = x(1)b(0) + x(2)b(1) + \ldots + x(11)b(10) = 1 \tag{3}$$

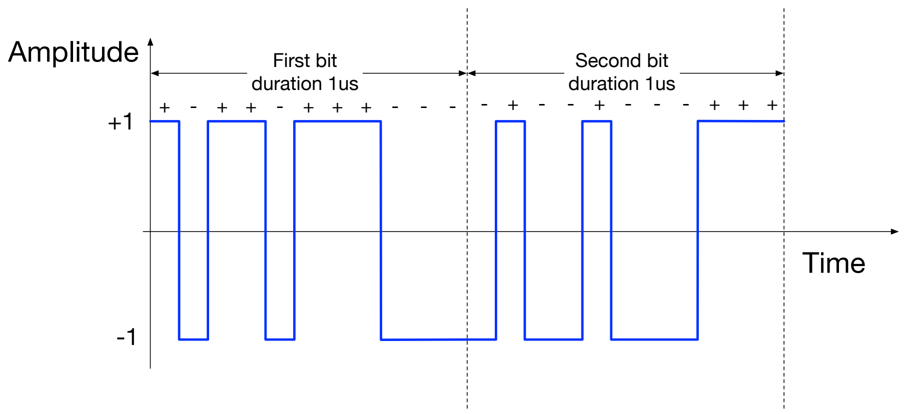


Fig. 3. Example of DSSS code

Similarly, we have $X(2)$ to $X(11)$ as {-1,1, -1,..., -11}. The correlation can be easily performed by Matlab by calling the function of xcorr; just remember to cut the head and tail points.

Tip 2: When performing correlation between a short and a long sequence, xcorr function in Matlab zero pads the short sequence so that the result is VERY long. To save the calculation time, you may choose to use a similar function called convolution to perform the correlation operation. The convolution function in Matlab is called conv, and you need to use a time-reversed version of Barker code, e.g., barker_25(end:-1:1) instead of the original Barker code, when using the convolution to calculate the correlation.

For Python, you may use numpy.coorelate or numpy.convolve to perform correlation or convolution. Be careful about the direction of Barker code when you are using different approaches.

Figure 4 shows the correlation result of the DSSS code. We have the following observations from Figure 4. First, there are some high peaks of +11 and -11 in the correlation result and other points are very small. Second, the position of high peaks corresponds to the starting point of the bit, and its value corresponds to the transmitted bit (+11 for a non-inverted bit, -11 for the inverted one). Therefore, we can use the value of these maximum values among the 11 chip period to represent the transmitted bits. Each peak will correspond to one data bit.

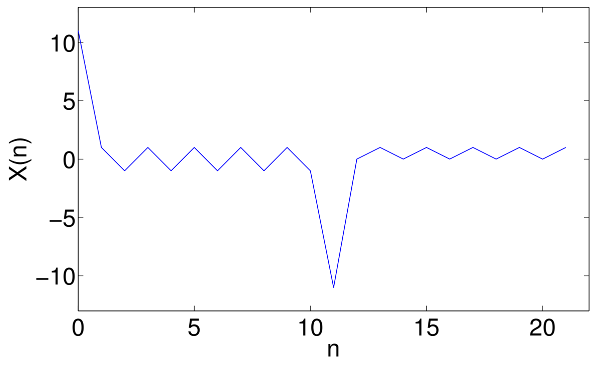


Fig. 4. Correlation result of the DSSS code

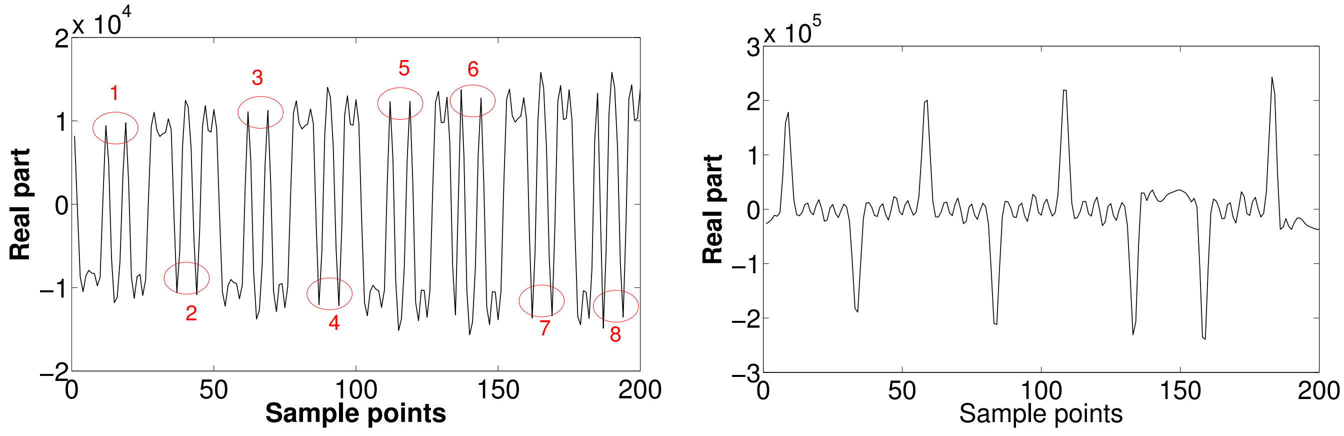
Figure 5(a) shows the real part of the baseband signal captured by USRP. We can roughly recognize the shape of the Barker code, i.e., two sharp peaks followed by a wide peak. As our sampling rate is 25 M, there are 25 sample points within 1 μs period. We can see that there are 8 full cycle of Barker codes in the 200 sample points in Figure 5(a). Therefore, each data bit contains 25 sample points (25 complex numbers) in the baseband signal. To despread the baseband, we need to use a Barker code that contains 25 points rather than 11 points. We have resampled the barker code and give you the 25-point sequence as {+1, +1, -1, -1, +1, +1, +1, +1, +1, -1, -1, +1, +1, +1, +1, +1, +1, +1, -1, -1, -1, -1, -1, -1, -1}. Therefore, when you perform the despreading operation, you need to:

1) Perform correlation between the 25-point barker code with the baseband signal sequence. Note that you need to do the correlation on the complex signal.
2) Find out the peak points, i.e., points with maximum magnitude, among each small segment with 25 correlation result points. Therefore, you need to pick only one point in every 25 points to get the despread result. In other words, you only have 1 M samples per second after the despread operation.

Tip 3: You need to check your result from time to time to ensure that you are on the right track. Here is an important check point to check your correlation results. If you see correlation results similar to Figure 5(b), you are possibly on the right track. Then, try to write code that only retains the high peaks that appear in every 25 samples.

Tip 4: You must remember that you are operating on samples that are represented by complex numbers. Although Matlab can operate on complex numbers in a similar way as real numbers, you must be careful about your operations. For example, when you are picking the peaks, you need to find the point with maximum magnitude (absolute value) of the complex number, not just the maximum of the real or imaginary parts.

As the data size is large, you need to find a good way to perform the value picking operation in MATLAB or Python. The right approach should only take a few seconds to decode the whole file.



(a) Enlarged baseband (real part)

(b) Correlation result of baseband signal

Fig. 5. Despreading baseband signals

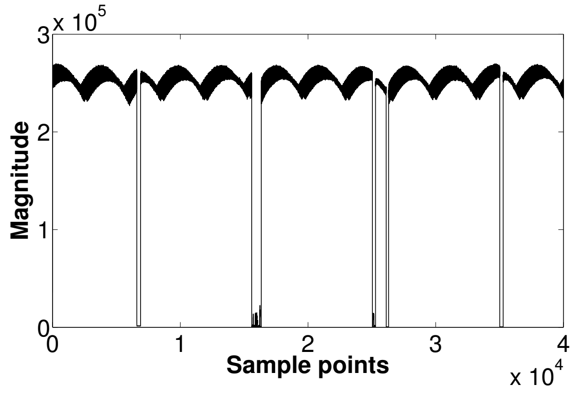
Task 1: Write your own code to despread the raw baseband signal. The code can be written within the main function as it only takes a few lines.

D. Finding start point of the frame

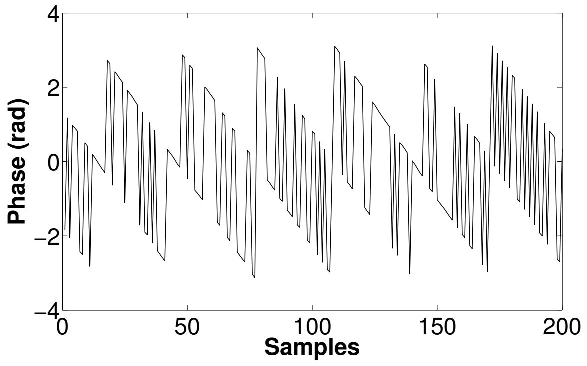
The second step is to find out the start and ending point of the frame. Figure 6(a) shows the magnitude of the despread signal. We observe that frame boundaries are quite clear in this figure.

Therefore, we can use the magnitude change to determine the start and ending of the frames. The code for detecting start and endings are given to you as find_frame_boundary.m. The input is the despread signal and the output is a $2 \times n$ matrix, where n is the number of detected frames and the two rows contains the starting point and ending point of the frames, respectively.

Questions:
1. Why the magnitude of the despread signal in Figure 6(a) has regular ripples?
2. What does the frequency of these ripples mean?



(a) Magnitude



(b) Phase

Fig. 6. Magnitude and phase of the despread signal

E. Decode DBPSK sequence

After you get the despread signal of each frame, you need to decode the Differential Phase-Shift Keying (DBPSK) modulation that is used to encode the bits when 802.11b is sending at 1 Mbps rate. You need to look at the DBPSK specification in Chapter 15.4.6.4 of the IEEE 802.11 standard [1].

The basic concept of DBPSK is to use the phase change to represent the transmitted bits. You need to compare two consecutive sample points in the despread signal to determine the transmitted bits. If the phase of these two complex numbers are similar (close to 0), this means a bit “0” is transmitted. If the phase of these two complex numbers are inverted (close to π), this means a bit “1” is transmitted. From Figure 6(b), the absolute phase of the complex signal keeps changing in the real-world signal. Therefore, you cannot use the absolute phase to determine the transmitted bits. You can only compare the phase of two consecutive complex numbers.

Task 2: You need to write your own code in decode_dbpsk.m or lab1step4.py that uses the input despread signal to determine the sequence of transmitted bits.

Checkpoint: If after DBPSK, the first few bits in your frame are $\{1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1\}$ or something similar to this, then you are possibly on the right track. The first bit in the above sequence is the bit received first. Note that it is possible to miss some bits or have more bits than the given sequence. That would be fine.

F. Descrambler

The bit sequence obtained from DBPSK decoding is still not the bit transmitted. We need to descramble them. As the transmitted bits may contain a large number of consecutive “0”s or “1”s, before transmitting them, we often need to scramble them so that they appear as pseudo-random binary sequences. Therefore, when we receive the data, we need to descramble the received bits and get back the original data bits .

You need to look at Chapter 15.2.4 of the IEEE 802.11 standard [1] to understand how the descrambler works. Basically, the scrambler and descrambler are Linear Feedback Shift Registers (LFSR) that manipulate on the binary bits. We only need to focus on the descrambler here. The polynomial for the descrambler is $G(z) = Z^{-7} + Z^{-4} + 1$, where Z^{-n} means delay for n samples in Z-transform. In other words, if you want to get bit at time n, you need to calculate $A(n) = B(n-7) \oplus B(n-4) \oplus B(n)$, where $A(n)$ is the output sequence, $B(n)$ is the input binary sequence before descrambling and $\oplus$ is the XOR operation.

Task 3: You need to write your own code in descramble.m or lab1step5.py to perform the descramble operation.

Tip 5: You must be careful about the order of the bit sequence here. In the sequence of the raw bits, the first received bits comes first. In the formula for LFSR, $B(n-7)$ means 7 bits before the current received bit.

Checkpoint: In the descrambled bit sequence, you should find a long sequence of all “1”s near the beginning of the frame.

III. From Bits to Packets

After descrambling, you get the transmitted bit sequence of the frame. The next step is to assemble these bits into real data packets.

A. Finding the preamble

The first step for assembling the data packet is to detect the preamble of the packet. The preamble is a fixed binary sequence to mark the start of the frame. It is also called Synchronization (SYNC), which is part of the Physical Layer Convergence Procedure (PLCP) header of the 802.11b frame. To better understand the organization of 802.11 PLCP header, please refer to Chapter 15.2 in IEEE 802.11 standard [1].

The SYNC in 802.11b is 128 bit of all “1”s. Therefore, after descramble, you only need to search for an all-one sequences at the beginning of the frame. Note that in some cases, you may miss several initial ones due to the delay in frame start detection. Sometimes, there might be some “0”s at the beginning of the frame due to decode errors. However, if you find a long sequence of “1”s at the beginning of the frame, it is possibly the SYNC.

After you get the preamble, you need to find the Start Frame Delimiter (SFD) that follows the SYNC. The SFD starts with a “0”, therefore, we can use the position of the first “0” after the long sequence of “1”s as the start of SFD. As defined in Chapter 15.2.3.2 in the standard [1], the SFD is 16 bits long with value of 0xF3A0. You need to check the SFD to see if the decoding is OK. If you cannot find the SYNC or get a wrong SFD, you may choose to discard the frame as it cannot be decoded. But, be careful, most of the frames in the file that we give to you can be decoded.

Task 4: You need to write your own code for finding the SYNC and verify the SFD. In the Matlab code framework given to you, the code can be added in find_preamble.m and bittohexstr.m. In the Python code framework given to you, the code can be added in lab1step6.py and lab1step7.py.

B. Get PLCP header and ignore frames that cannot be decoded

After finding the SFD, you can decode the rest of the PLCP header. You need to focus on the SIGNAL field, which indicates the transmission rate of the frame. As stated in Chapter 15.2.3.3, the SINGAL field for a 1 Mbps frames that uses DBPSK is set to 0x0A. Therefore, if you find other values in the SINGAL field, you may discard the frame, as you are only interested in decoding the 1 Mbps frames.

Task 5: You need to write your own code to determine the SINGAL field and check the rate of the frame.

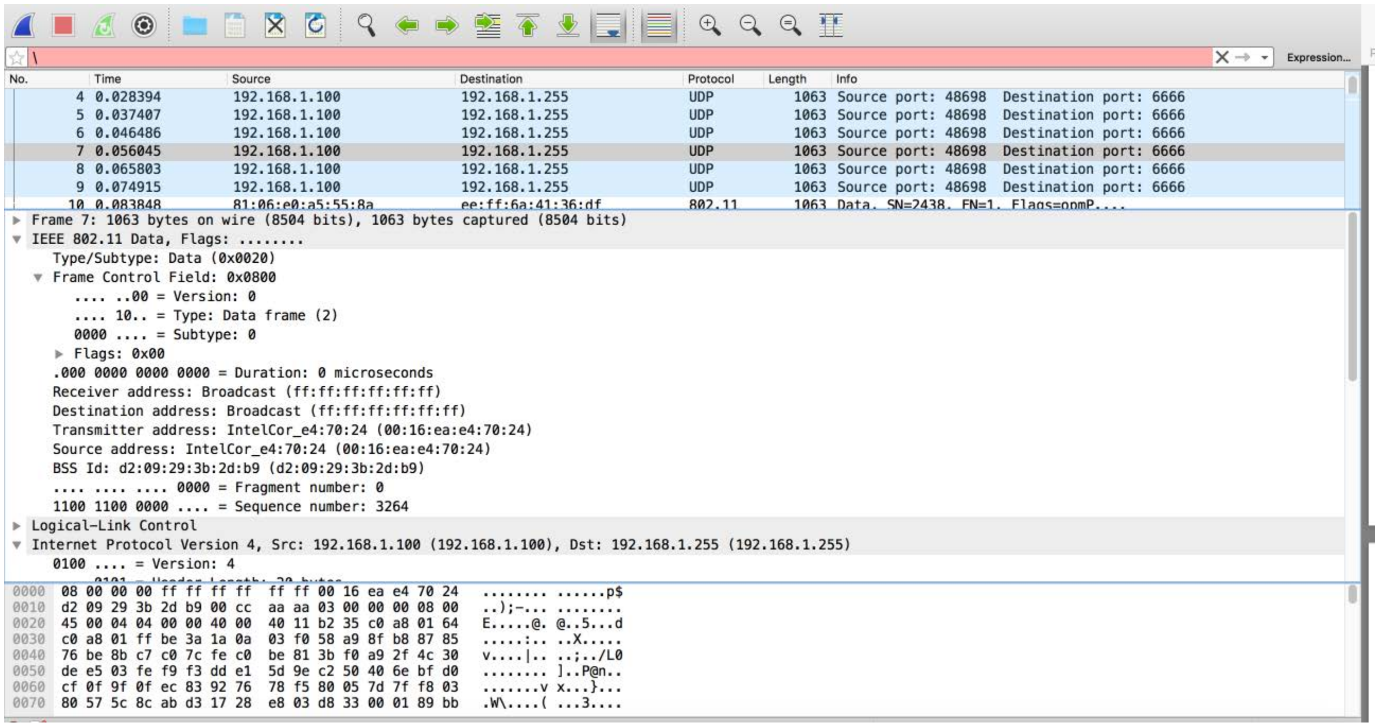
C. MAC and TCP/IP layer

After you checked all the remaining fields in the PLCP header, you will face the real payload of this frame, the MAC Protocol Data Units (MPDU). It is your task to decode all the data from the MAC layer to the TCP/IP and application layer from the MPDU you discovered. There are two ways to decode these protocol layers. The first way is to look at Chapter 7 in IEEE 802.11 [1], and figure out the meaning of all fields in the MPDU. This could take some time, as you need to understand all the fields in different layers. The second way is to ask the powerful tool – Wireshark to help you decode all these binary sequences. We will focus on the second way to do the packet decoding.

To ask Wireshark to help you decode the packet, you need to write your MPDU data in a specific file format called Libpcap file format. You need to look at the link and figure out how to generate a correct file header and headers for each frame. The MPDU of 802.11 can be decoded by Wireshark as LINKTYPE_IEEE802_11. After that you can browse the packet in Wireshark as in Figure 7.

Task 6: You need to use your own ways to find out the content of the frames.

Tip 6: When playing with the binary bits, be careful of the bit order in your data and the endianness of the machine.



```
No.   Time       Source              Destination         Protocol  Length  Info
    4 0.028394   192.168.1.100       192.168.1.255       UDP         1063  Source port: 48698  Destination port: 6666
    5 0.037407   192.168.1.100       192.168.1.255       UDP         1063  Source port: 48698  Destination port: 6666
    6 0.046486   192.168.1.100       192.168.1.255       UDP         1063  Source port: 48698  Destination port: 6666
    7 0.056045   192.168.1.100       192.168.1.255       UDP         1063  Source port: 48698  Destination port: 6666
    8 0.065803   192.168.1.100       192.168.1.255       UDP         1063  Source port: 48698  Destination port: 6666
    9 0.074915   192.168.1.100       192.168.1.255       UDP         1063  Source port: 48698  Destination port: 6666
   10 0.083848   81:06:e0:a5:55:8a   ee:ff:6a:41:36:df   802.11      1063  Data, SN=2438, FN=1, Flags=opmP....
```

```
▶ Frame 7: 1063 bytes on wire (8504 bits), 1063 bytes captured (8504 bits)
▼ IEEE 802.11 Data, Flags: ........
    Type/Subtype: Data (0x0020)
  ▼ Frame Control Field: 0x0800
      .... ..00 = Version: 0
      .... 10.. = Type: Data frame (2)
      0000 .... = Subtype: 0
    ▶ Flags: 0x00
    .000 0000 0000 0000 = Duration: 0 microseconds
    Receiver address: Broadcast (ff:ff:ff:ff:ff:ff)
    Destination address: Broadcast (ff:ff:ff:ff:ff:ff)
    Transmitter address: IntelCor_e4:70:24 (00:16:ea:e4:70:24)
    Source address: IntelCor_e4:70:24 (00:16:ea:e4:70:24)
    BSS Id: d2:09:29:3b:2d:b9 (d2:09:29:3b:2d:b9)
    .... .... .... 0000 = Fragment number: 0
    1100 1100 0000 .... = Sequence number: 3264
▶ Logical-Link Control
▼ Internet Protocol Version 4, Src: 192.168.1.100 (192.168.1.100), Dst: 192.168.1.255 (192.168.1.255)
    0100 .... = Version: 4
```

```
0000  08 00 00 00 ff ff ff ff  ff ff 00 16 ea e4 70 24   ........ ......p$
0010  d2 09 29 3b 2d b9 00 cc  aa aa 03 00 00 00 08 00   ..);-... ........
0020  45 00 04 04 00 00 40 00  40 11 b2 35 c0 a8 01 64   E.....@. @..5...d
0030  c0 a8 01 ff be 3a 1a 0a  03 f0 58 a9 8f b8 87 85   .....:.. ..X.....
0040  76 be 8b c7 c0 7c fe c0  be 81 3b f0 a9 2f 4c 30   v....|.. ..;../L0
0050  de e5 03 fe f9 f3 dd e1  5d 9e c2 50 40 6e bf d0   ........ ]..P@n..
0060  cf 0f 9f 0f ec 83 92 76  78 f5 80 05 7d 7f f8 03   .......v x...}...
0070  80 57 5c 8c ab d3 17 28  e8 03 d8 33 00 01 89 bb   .W\....( ...3....
```

Fig. 7. Wireshark packet viewer

IV. Submission Requirements

Please submit both your code and your assignment report to the course web site before the deadline. Any late submission will be rejected. There is also a template for your assignment report in the handouts for this assignment.

V. Acknowledgement

We would like to thank Mr. Lei Wang's efforts in developing this assignment.

References

[1] "Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," IEEE Standard 802.11, 1999.
[2] "USRP software radio," http://www.ettus.com/.