18-441/741: Computer Networks

Project 1: Exploring the Wi-Fi PHY

Questions? Ask on Piazza

1. The Wi-Fi Physical Layer

Have you ever wondered what a Wi-Fi signal from your computer or phone looks like? The objective of this project is to make you learn in action, how a popular physical layer protocol (Wi-Fi) works. In this project, you will build a very simple Wi-Fi decoder in software that mimics the very same processes that your phone/laptop's Wi-Fi chip does. You will be given a transmitter Python design that simulates Wi-Fi packets: taking the contents of a packet and creating a Wi-Fi signal. Your role is to reverse this process: take as input a Wi-Fi signal and return the corresponding Wi-Fi message.

2. You are given a Wi-Fi transmitter code

You are given a Python *wifitransmitter.py* file which generates a Wi-Fi packet, a function defined as follows:

```
def WifiTransmitter(*args):
```

The `WifiTransmitter` function can take maximum 3 values as input arguments in the order given below (with first argument required and other 2 arguments optional):

1. `message`: (**Required**) A text 'message' that contains the contents of a Wi-Fi packet. It should be able to handle arbitrary textual input (i.e., any sequence of ASCII characters). The sequence will be no longer than `10000` bytes.
2. `level`: (**Optional**) Level indicates various stages of encoding. Each message undergoes four levels of encoding before becoming a wireless signal: (1) Level-1/Interleaving: The bits are permuted by a well-known permutation; (2) Level-2/Turbo Coding and Modulation: A turbo code is applied to the bits and then the bits are mapped to complex numbers by a simple mapping (4-QAM); (3) Level-3/OFDM: The bits are converted to an OFDM sequence; (5) Level-4/Noise: Some noise is added to the packet and we apply some random zero-padding to the beginning and end of the packet. Default level is 4.
3. `snr`: (**Optional**) Specifies the signal to noise ratio (in dB) applied to the signal. Value could be any positive or negative real number. Default snr is infinity.

Once the function is called with appropriate inputs, it returns the Numpy array '`output`' which is a sequence of complex numbers containing the encoded packet. It also includes the 'Length of the message' prefixed before the encoded message packet. The function only returns the Numpy array '`output`' for Level 1, Level 2, and Level 3. For Level 4, it also returns the number of zeros which were padded at the

beginning and the length of the message along with the Numpy array as outputs.
You are encouraged to run this program for different input configurations and observe the output values. You are also encouraged to thoroughly read the provided .py file to better understand all the different levels.

Note: The implementation you are given is a very simplistic version of the Wi-Fi physical layer that excludes many bells and whistles for simplicity (e.g., cyclic prefix, pilots, guard bands, etc.). The goal is to give you a sense of how both a Wi-Fi transmitter and receiver work at the PHY layer. However, since we are not following the Wi-Fi protocol to the tee, you will be unable to solve this project with standard Wi-Fi decoding libraries or tools. You are therefore strongly encouraged to write your own receiver code from scratch. **Note that you are only allowed to use the Numpy and Commpy library to create the receiver code.**

3. Your Task: Write the Wi-Fi receiver code

Your task is very simple: create a Wi-Fi receiver program called *wifireceiver.py* with a function `WifiReceiver` that takes as input the transmitted signal `output` generated by `WifiTransmitter` and outputs `zero_padding, message,` and `length`. The Gradescope will run your program by running the following function that you need to write:

```
def WifiReceiver(output, level):
```

This function should have the following statement or an equivalent return statement at the end for ensuring correct grading on Gradescope:

```
return begin_zero_padding, message, length
```

where,
`begin_zero_padding = Number of zeros appended at the beginning of the message in Level 4`
`message = The message used to create WiFi packet`
`length = the length of the message`

Note that even with Level 1, Level 2 and Level 3 where there is no zero padding, your function should output 0 for the variable `begin_zero_padding.`

Your main deliverable is the Python script *wifireceiver.py* containing the definition of the function `WifiReceiver`. You should submit your original source code and **not** an executable.

We will run the following tests as explained below. For illustrative purposes, the following examples use '`hello world`' as the transmitted message. Your program should work correctly for any valid input message to function `WifiTransmitter`.

(1) Level 1 Test (10 pts)

In the level-1 test, only interleaving step is enabled at the transmitter.

In Python3:

```
>>> txsignal = WifiTransmitter('hello world', 1)

>>> WifiReceiver(txsignal, 1)
```

Should output the following:

```
0, 'hello world', 11
```

Your program at this level should undo the Interleaving correctly.

(2) Level 2 Test (10 pts)

In the level-2 test, the turbo encoding, modulation and interleaving steps are enabled at the transmitter.

In Python3:

```
>>> txsignal = WifiTransmitter('hello world', 2)

>>> WifiReceiver(txsignal, 2)
```

Should output the following:

```
0, 'hello world', 11
```

Your program at this level should undo the turbo encoding, demodulation and interleaving correctly.

Note: To decode and demodulate the complex numbers i.e., converting the complex number samples to decoded bits, you can use Hard or Soft Decoding. In case of Hard Decoding, you can use the Commpy package to convert complex numbers to coded bits and then write your own Hard Viterbi decoding function to convert from coded bits to decoded bits. In case of Soft Decoding, you can directly convert from complex numbers to decoded bits by writing your own Soft Viterbi decoder (without the intermediate step of converting from complex numbers to coded bits). However, in either case you are **not allowed** to use *viterbi_decode()* function of the CommPy package. You must write the Viterbi decoding function by yourself from the scratch. If you choose not to do so and use the *viterbi_decode()* function of CommPy package, there is a 30 point penalty and you will be graded out of 70 points only. This is already ensured in Gradescope and a manual check will also be done before releasing final scores to ensure that a student did not use viterbi_decode() or a similar function.

(3) Level 3 Test (10 pts)

In the level-3 test, the OFDM signal generation, turbo encoding, modulation and interleaving steps are enabled at the transmitter.

In Python3:

```
>>> txsignal = WifiTransmitter('hello world', 3)

>>> WifiReceiver(txsignal, 3)
```

Should output the following:

```
0, 'hello world', 11
```

Your program at this level should perform the OFDM demodulation, turbo decoding, interleaving and

demodulation steps correctly.

(4) Level 4 Test (30 pts)

In the level-4 test, all steps are enabled at the transmitter.

In Python3:

```
>>> noise_pad_begin, txsignal, length = WifiTransmitter('hello world', 4)

>>> WifiReceiver(txsignal, 4)
```

Should output the following:

```
noise_pad_begin, 'hello world', 11
```

Your program at this level should undo the turbo decoding, interleaving, modulation and OFDM steps correctly. You will be awarded 10 points each for each of the three outputs. To get the full 30 points, you are guaranteed that the snr input will be 30 dB or higher.

(5) Checking SNR limits (10 pts)

For the above test cases, it is ensured that the SNR input will be 30 dB or higher. The points here are to check the SNR limits of your receiver code. There is a minimum SNR limit (unknown to students) below which no receiver can decode the packet. This limit has been set by surveying the past instances of the submissions of this project. You are required to make your code as robust as possible so that it can achieve that limit. In this test, you are graded based on how close you are to the minimum SNR limit. **Remember that snr values can be negative. If you chose to use *viterbi_decode()*, you will get 0 in this test.**

For example – Given the minimum SNR limit is 5 dB, and your code fails to decode Wi-Fi messages below SNR of 10 dB, more than 50% of the time, you will be awarded ((30-10)/(30-5))*10 = 8 points out of 10.

Hint: Look up differences between Soft Viterbi Decoding and Hard Viterbi Decoding

4. Discussions

I'm confused, where do I start?

Relax; the project may not be as hard (or as easy) as you think initially. Make sure you start early, there are plenty of good online resources available to get started. A good starting point is to start with Level 1 and then progressively include Levels 2, 3 and 4 to build your script. Reach out to the TAs or use piazza should you face any questions.

Help! I accidentally delete my working code.

Make a lot of backups, if you do not have the source when the deadline comes, you do not have the project. You may also want to use version control tools to manage your source. The ECE undergraduate cluster machines are equipped with both `git` and `subversion` tools (although learning to use them is not within the scope of this course). Make sure you have a working branch/revision available for final submission before start tuning for performance or adding features. You could always submit an early working version to the hand-in directory. So make sure you contact the TA and give us information about your submission early.

5. Hand-in Procedure

You will submit your project to Gradescope which will run your code for some of the test cases for all the levels and output the scores for the visible test cases in real-time. There are some test cases which are hidden, and their scores will be revealed after the final scores.

6. Deliverable Items

The deliverables are enumerated as follows,

1. **wifireceiver.py** containing the WifiReceiver function definition.

2. **Do not submit the executable files** (we will DELETE them and deduct your logistic points).

3. **A brief design document** regarding your server design in `design.pdf` in the submission directory (1-2 pages). Tell us about the following,
    a. Your decoder design
    b. How did you infer the length of the packet?
    c. Extra capabilities you implemented (optional)

Note: Do not submit any zip/rar files. In the Gradescope submission pop-up box, submit both the files *wifireceiver.py* and *design.pdf* separately to ensure that your submission can be autograded correctly.

7. Grading

Partial credit will be available according to the following grading scheme,

Items	Points (100)
Logistics - Successful code submission - Successful Design document submission	 5 5
Levels - Level-1 test - Level-2 test - Level-3 test - Level-4 test (10 points per output)	 10 10 10 30
viterbi_decode() not used	20
Reach the minimum SNR limit Remember that **SNR values can be negative**.	10 (0 if viterbi_decode() used)