ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

School of Computer and Communication Sciences

Handout 22 Principles of Digital Communications
Project May. 12, 2021

Channel Model

Consider a channel which takes an input vector $\mathbf{X} \in [-1, 1]^n$ and returns an output vector $\mathbf{Y} \in \mathbb{R}^n$. The input and the output of this channel is related as follows, for all $i \in \{1, \ldots, n\}$

$$\tilde{X}_i = \begin{cases} 0 & i \equiv H \text{ [mod 3]} \\ X_i & \text{otherwise} \end{cases}$$

$$Y_i = \tilde{X}_i + Z_i$$

where H is taken from $\{0, 1, 2\}$ with uniform probability and Z_i's are i.i.d. with $Z_i \sim N(0, 10)$. In other words, the channel erases a third of its inputs depending on the value of H before adding a gaussian noise with variance of 10. The block length of your input signal, i.e., n is part of your design, with the constraint $n \leq 60000$. We will give a bonus point for the team which uses the smallest n to achieve reliable communication.

Assignment

Develop a system capable of reliably transmitting text files over this channel. Specifically:

- Design a transmitter that reads a text file and returns real-valued samples of an information-bearing signal $\mathbf{X}$.
- We used a server to simulate the channel. You send $\mathbf{X}$ to the server and the server will return $\mathbf{Y}$.
- After receiving $\mathbf{Y}$, your receiver must reconstruct the contents of the text file.

Submission and Evaluation Rules

- You work in teams of *two or three*.

  Please choose your teammates at latest by **Friday, May 21** and send an email to `reka.inovan@epfl.ch` in order to register your team.

- You can use any programming language, as long as all the code pertaining to the transmitter and receiver is produced by your team.
- During the last session (June 4), each group presents their project in 5–10 minutes and gives a demonstration by transmitting a file that we provide.

  (i) You will run the transmitter and the receiver on your own laptop.

  (ii) You must submit your team code to Moodle before **Friday, June 4, 10am**. For each team, it is sufficient to submit through one of the member's Moodle account.

  (iii) The text file which you will be asked to transmit will contain *roughly 80 characters/bytes*. Please ensure that your implementation can work with `utf8` encoded text, i.e., it is not sufficient to only accepts alphabets and punctuations.

(iv) Your presentation should contain a brief explanation of your signaling scheme, followed by the transmission and decoding of the chosen text (that will be given to you on the spot).

(v) You will be given *two* chances for transmission. I.e., if the received text is different than the sent one at the first attempt, you can repeat the transmission once more.

- The grading is based on the reliability of your scheme during the demonstration phase.

  (a) If you manage to transmit the file without errors during the first or second transmission you will get the full mark (15/15 pts).

  (b) Otherwise your mark will be $12-\varepsilon$ (out of 15 pts) where ε is number of incorrect or deleted *characters* in the decoded text.

  (c) On top of that, the team with the smallest n (among the error-free ones) will get a bonus of 5 points.

Python Client

To simplify communication with the channel server, we provide you with a Python script `client.py` that you can download on Moodle. The client takes a text file containing the value of $\mathbf{X}$ as its input and will create a text file containing the value of $\mathbf{Y}$.

You can only connect to the server if you are inside EPFL's VPN (please visit the following link[1] for more information). Please use the following connection parameters

- `--srv_hostname=iscsrv72.epfl.ch`
- `--srv_port=80`

For example, the command,

```
python client.py --input_file input.txt --output_file output.txt --srv_hostname
iscsrv72.epfl.ch --srv_port 80
```

will read the value of $\mathbf{X}$ from `input.txt` and write down the value of $\mathbf{Y}$ to `output.txt`. Please verify whether you need to call `python` or `python3` to invoke Python 3 interpreter on your system.

The client is tested on Python 3.6 and requires the `numpy` library.

[1] https://www.epfl.ch/campus/services/en/it-services/network-services/remote-intranet-access/

Tips and Practical Considerations

- To avoid overloading the server, we only allow each client to connect once every 30 seconds.
- The processing on the server is equivalent to the following function:

```
import numpy as np

def channel(chanInput):
  chanInput = np.clip(chanInput,-1,1)
  erasedIndex = np.random.randint(3)
  chanInput[erasedIndex:len(chanInput):3] = 0
  return chanInput + np.sqrt(10)*np.random.randn(len(chanInput))
```

  You can use this Python function to do local tests on your implementation.
- To save a `numpy` array into a suitable format for the Python client, you can use `np.savetxt(fileName,numpyArray)`. Conversely, to read the output file of the Python client, you can use `numpyArray = np.loadtxt(fileName)`.
- From past years experience, it is a good idea to implement your system incrementally. For example, a possible milestones would be :

  1. Design an implementation to convert a string to an array of bits, and the other way around.
  2. Design an error correction method to recover the message if a third of your bits are erased but there is no noise.
  3. Design a method to send the message through the complete channel

  Note that this is only a recommendation and you are more than welcome to design your own milestones.
- Verify your implementation before moving to the next milestone. In general, this will make the whole process easier, as you do not have to test and debug a complex system from the get go. It is also a good idea to test the system on low noise variance before increasing it to the real value.
- The parameters of the channel are designed so you can achieve reliable communication with a relatively simple scheme[2]. However, we do not know the most optimal method to communicate through this channel. Don't be afraid to experiment once you have implemented a minimum viable scheme.

[2]Our implementation is around 150 lines of code. Note that this is only to give you a rough idea of the workload, we encourage you to optimize your code for readability instead of length.