**MÔ PHỎNG TRIỂN KHAI DSSS\_ChaoticMap**

**1. Mục tiêu**

Trong bài lab này, chúng ta sẽ tìm hiểu và triển khai hệ thống truyền thông **Direct Sequence Spread Spectrum (DSSS)** sử dụng ánh xạ hỗn loạn (chaotic map) dựa trên Python. Bài lab tập trung vào các bước:

* Mã hóa tin nhắn thành tín hiệu BPSK.
* Tạo mã trải phổ hỗn loạn sử dụng ánh xạ logistic.
* Trải phổ tín hiệu và nhúng vào âm thanh.
* Giải trải phổ, khôi phục dữ liệu, và phân tích kết quả (tính BER, vẽ biểu đồ).

**2. Yêu cầu đối với sinh viên**

* Kiến thức cơ bản về hệ điều hành Linux.
* Hiểu biết về ngôn ngữ lập trình Python và các thư viện như numpy, scipy, matplotlib.
* Nắm bắt các khái niệm cơ bản về xử lý tín hiệu số, ánh xạ hỗn loạn, và truyền thông DSSS.

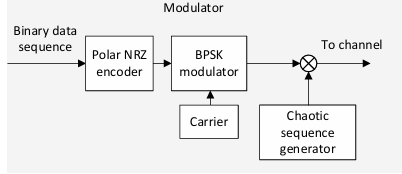
**3. Giới thiệu chung về bài thực hành**

**DSSS (Trải phổ trực tiếp)**: Dữ liệu gốc (ví dụ: tin nhắn văn bản) được nhân với một chuỗi mã hỗn loạn có tốc độ cao hơn, gọi là chip, để trải phổ tín hiệu trên băng tần rộng. Quá trình này giúp tăng khả năng chống nhiễu và bảo mật tín hiệu.

**Ánh xạ hỗn loạn**: Sử dụng ánh xạ logistic (( x\_{n+1} = r \cdot x\_n \cdot (1 - x\_n) )) để tạo chuỗi mã trải phổ có tính chất ngẫu nhiên, nhạy với điều kiện ban đầu, thay cho mã giả ngẫu nhiên (PN code) truyền thống.

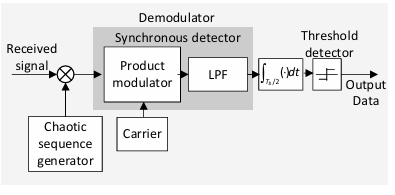
**Nguyên lý hoạt động**:

**Phía truyền (Transmitter):**

****

* **Tạo tín hiệu BPSK**: Chuyển tin nhắn thành chuỗi nhị phân (bit 1 → +1, bit 0 → -1).
* **Tạo mã hỗn loạn**: Sử dụng ánh xạ logistic để tạo chuỗi mã trải phổ nhị phân (+1/-1).
* **Trải phổ tín hiệu**: Nhân tín hiệu BPSK với mã hỗn loạn để tạo tín hiệu DSSS.
* **Nhúng vào âm thanh**: Cộng tín hiệu DSSS vào âm thanh với biên độ nhỏ, tạo tín hiệu ẩn.
* **Thêm nhiễu AWGN**: Mô phỏng kênh truyền bằng cách thêm nhiễu Gauss trắng.

**Phía nhận (Receiver):**

****

* **Trích xuất tín hiệu**: Tách tín hiệu DSSS từ file âm thanh.
* **Giải trải phổ**: Nhân tín hiệu nhận được với mã hỗn loạn để khôi phục dữ liệu nhị phân.
* **Giải mã và phân tích**: Chuyển dữ liệu nhị phân thành tin nhắn, tính tỷ lệ lỗi bit (BER), và vẽ biểu đồ tín hiệu.

**4. Nội dung thực hành**

Bài lab được thực hiện trên môi trường Linux (tương tự Labtainer) với hai terminal ảo (máy alice để truyền và bob để nhận). Dưới đây là các bước chi tiết để mô phỏng hệ thống DSSS\_ChaoticMap.

**Khởi động môi trường lab**

Chạy lệnh sau để khởi động môi trường Labtainer:

labtainer -r dsss\_chaoticmap

Hai terminal ảo sẽ xuất hiện: máy alice (truyền) và máy bob (nhận).

**Tại máy alice**

**4.1. Tạo tín hiệu BPSK**

Chuyển tin nhắn từ file văn bản thành tín hiệu BPSK và lưu vào file bpsk\_data.txt.

python3 handle\_message\_and\_modulation.py -i sec.txt

cat bpsk\_data.txt

* **Đầu vào**: File sec.txt chứa tin nhắn (ví dụ: "Hello").
* **Kết quả**: File bpsk\_data.txt chứa chuỗi nhị phân (data: ...) và tin nhắn gốc (original\_message: ...).
* **Giải thích**: Tin nhắn được mã hóa thành chuỗi nhị phân (8 bit/ký tự), sẵn sàng cho trải phổ.

**4.2. Tạo mã trải phổ hỗn loạn**

Tạo chuỗi mã hỗn loạn sử dụng ánh xạ logistic và lưu vào file spreading\_code.txt.

python3 generate\_chaotic\_code.py -m bpsk\_data.txt

cat spreading\_code.txt

* **Đầu vào**: File bpsk\_data.txt để xác định số bit (N).
* **Kết quả**: File spreading\_code.txt chứa mã trải phổ nhị phân (+1/-1).
* **Giải thích**: Chuỗi hỗn loạn được tạo với ( r = 3.9 ), ( x\_0 = 0.5 ), độ dài ( N \cdot SF ) (mặc định ( SF = 31 )).

**4.3. Trải phổ và nhúng tín hiệu vào âm thanh**

Trải phổ tín hiệu BPSK, nhúng vào âm thanh, thêm nhiễu AWGN, và lưu kết quả.

python3 process\_audio\_and\_signal.py -m bpsk\_data.txt -c spreading\_code.txt -a input1.wav

cat audio\_signal.txt

* **Đầu vào**:
  + bpsk\_data.txt: Dữ liệu nhị phân.
  + spreading\_code.txt: Mã trải phổ.
  + input1.wav: File âm thanh WAV (hoặc tạo âm thanh mẫu nếu không có).
* **Kết quả**:
  + File embedded\_audio.wav: Âm thanh chứa tín hiệu DSSS.
  + File audio\_signal.txt: Chứa tín hiệu nhiễu (noisy\_signal), tín hiệu trải phổ (spread\_signal), và số bit (N).
* **Giải thích**: Tín hiệu BPSK được trải phổ bằng mã hỗn loạn, nhúng vào âm thanh với biên độ nhỏ (0.05), và thêm nhiễu với SNR = 10 dB.

**4.4. Nén và truyền file**

Nén các file cần thiết và gửi qua mạng tới máy bob.

tar czf file.tar.gz embedded\_audio.wav spreading\_code.txt bpsk\_data.txt

ncat --send-only 172.10.0.12 4444 < file.tar.gz

* **Kết quả**: File file.tar.gz chứa embedded\_audio.wav, spreading\_code.txt, và bpsk\_data.txt được gửi qua mạng.
* **Giải thích**: Sử dụng ncat để truyền file qua giao thức TCP.

**Tại máy bob**

**4.5. Nhận và giải nén file**

Nhận file nén từ máy alice và giải nén.

ncat -l 4444 > file.tar.gz

tar -xzf file.tar.gz

* **Kết quả**: Các file embedded\_audio.wav, spreading\_code.txt, và bpsk\_data.txt được giải nén.
* **Giải thích**: Máy bob lắng nghe trên cổng 4444 để nhận file từ máy alice.

**4.6. Giải trải phổ tín hiệu**

Trích xuất tín hiệu từ file âm thanh và giải trải phổ để khôi phục dữ liệu nhị phân.

python3 despread\_signal.py -a embedded\_audio.wav -c spreading\_code.txt

cat recovered\_data.txt

* **Đầu vào**:
  + embedded\_audio.wav: Âm thanh chứa tín hiệu DSSS.
  + spreading\_code.txt: Mã trải phổ.
* **Kết quả**: File recovered\_data.txt chứa dữ liệu nhị phân khôi phục (recovered\_data: ...).
* **Giải thích**: Tín hiệu được nhân với mã hỗn loạn, tính tương quan để xác định bit (1 hoặc 0).

**4.7. Phân tích và trực quan hóa**

Giải mã dữ liệu, tính BER, và vẽ biểu đồ tín hiệu.

python3 analyze\_and\_visualize.py -r recovered\_data.txt -a embedded\_audio.wav -c spreading\_code.txt -m bpsk\_data.txt

cat results.txt

* **Đầu vào**:
  + recovered\_data.txt: Dữ liệu khôi phục.
  + embedded\_audio.wav: Âm thanh chứa tín hiệu.
  + spreading\_code.txt: Mã trải phổ.
  + bpsk\_data.txt: Dữ liệu gốc để tính BER.
* **Kết quả**:
  + File results.txt: Chứa tin nhắn khôi phục (recovered\_message), BER (ber), và tin nhắn gốc (original\_message).
  + Hình ảnh chaotic\_dsss\_audio\_simulation.png: Biểu đồ dữ liệu gốc, tín hiệu trải phổ, và tín hiệu nhiễu.
* **Giải thích**: Dữ liệu nhị phân được giải mã thành văn bản, BER được tính bằng cách so sánh dữ liệu khôi phục với dữ liệu gốc.

**4.8. Kiểm tra kết quả**

Xem thông số BER và biểu đồ trực quan.

nano results.txt

# Tìm "ber" trong file results.txt bằng Ctrl+W và nhập "ber"

feh chaotic\_dsss\_audio\_simulation.png

* **Kết quả**:
  + Xem giá trị BER trong results.txt để đánh giá hiệu suất hệ thống.
  + Xem biểu đồ tín hiệu trong chaotic\_dsss\_audio\_simulation.png.

**5. Kết quả mong đợi**

* **File đầu ra**:
  + bpsk\_data.txt: Dữ liệu nhị phân và tin nhắn gốc.
  + spreading\_code.txt: Mã trải phổ hỗn loạn.
  + embedded\_audio.wav: Âm thanh chứa tín hiệu DSSS.
  + audio\_signal.txt: Tín hiệu nhiễu và trải phổ.
  + recovered\_data.txt: Dữ liệu nhị phân khôi phục.
  + results.txt: Tin nhắn khôi phục, BER, và tin nhắn gốc.
  + chaotic\_dsss\_audio\_simulation.png: Biểu đồ trực quan.
* **Tin nhắn khôi phục**: Nếu SNR đủ cao (mặc định 10 dB) và không có lỗi nghiêm trọng, tin nhắn khôi phục sẽ giống hoặc gần giống tin nhắn gốc.
* **BER**: Thấp (gần 0) nếu hệ thống hoạt động tốt, phụ thuộc vào SNR và yếu tố trải phổ (SF = 31).

**6. Lưu ý**

* Đảm bảo cài đặt các thư viện Python: numpy, scipy, matplotlib.
* File âm thanh đầu vào (input1.wav) phải là định dạng WAV. Nếu không có, mã sẽ tạo âm thanh mẫu (440 Hz).
* Các tham số như SF (Spreading Factor), SNR\_dB, embedding\_amplitude có thể được điều chỉnh trong mã để thử nghiệm.
* Nếu gặp lỗi về độ dài mã trải phổ hoặc tín hiệu, mã sẽ tự động lặp lại hoặc cắt ngắn để phù hợp.
* Để cải thiện hiệu suất:
  + Tăng SF để cải thiện khả năng chống nhiễu.
  + Tăng SNR\_dB để giảm BER.
  + Giảm embedding\_amplitude để tín hiệu DSSS khó phát hiện hơn.

**7. Hình ảnh minh họa**

* **Biểu đồ tín hiệu**: Xem file chaotic\_dsss\_audio\_simulation.png để trực quan hóa dữ liệu gốc, tín hiệu trải phổ, và tín hiệu nhiễu.
* **So sánh với CDMA**: Không như CDMA sử dụng mã Gold cho đa người dùng, DSSS\_ChaoticMap sử dụng ánh xạ hỗn loạn cho một người dùng, tập trung vào bảo mật và tính ngẫu nhiên.