Department of
Computer Science and Engineering

University of Dhaka

Title: Implementation of Simple Spreading Techniques in Data and Telecommunications

CSE 2213: Data and Telecommunication Lab

Batch: 29/2nd Year 2nd Semester 2024

Course Instructors

Dr. Md. Mustafizur Rahman (MMR)
Mr. Palash Roy (PR)

1 Objective(s)

- To understand the concept of spread spectrum in digital communication.
- To implement a simple spreading and despreading technique using Direct Sequence Spread Spectrum (DSSS) logic.
- To demonstrate how a data bit can be spread using a chip code and how it can be recovered.

2 Background Theory

In modern digital communication, data is often transmitted over a shared or noisy medium. To enhance the reliability, security, and bandwidth efficiency of transmission, spread spectrum techniques are used. A spread spectrum system spreads the baseband signal over a wider bandwidth than the minimum required. The spreading process makes the signal:

- Less susceptible to interference and jamming
- More secure (difficult to intercept without the spreading code)
- Capable of supporting multiple users simultaneously (as in CDMA)

Some applications of Spreading Techniques are as follows: Wireless Communication (Wi-Fi, LTE), Satellite Systems, Military Communications, and Bluetooth & Zigbee. Different types of Spread Spectrum Techniques are available, such as DSSS (Direct Sequence Spread Spectrum) and FHSS (Frequency Hopping Spread Spectrum). In DSSS, each bit is multiplied by a high-frequency pseudo-random bit sequence (chip code). In FHSS, the signal hops between different frequency channels based on a pseudo-random sequence.
This lab focuses on **Direct Sequence Spread Spectrum (DSSS)** due to its simplicity and relevance. The main advantages of DSSS are as follows:

- **Anti-jamming:** Spread signal is more resistant to narrowband interference.
- **Security:** Without the chip code, decoding is extremely difficult.
- **Multiple Access:** Allows multiple users to transmit over the same channel using different chip codes (CDMA).
- **Error Tolerance:** Minor noise or bit flips may not affect the recovery of the correct data bit.

2.1 Direct Sequence Spread Spectrum (DSSS)

In DSSS, each bit of the original data is multiplied (XORed) with a faster chip sequence. This chip sequence is a known pseudo-random binary code. The key concepts of DSSS are as follows,

- **Chip Code:** A binary sequence (e.g., [1, 0, 1]) shared between sender and receiver.
- **Spreading:** Each bit of data is converted to multiple bits (called chips) using the XOR operation with the chip code.
- **Despreading:** At the receiver, the received signal is XORed again with the same chip code to retrieve the original data.

2.2 How Spreading Works

Suppose, Data Bit: 1 and Chip Code: 1 0 1, Spreading will be as follows,
1 XOR 1 = 0
1 XOR 0 = 1
1 XOR 1 = 0
Therefore, spread Signal: 0 1 0. Each original bit is now represented by a 3-bit chip segment.
Now question comes why XOR?
Here, XOR is used because it allows reversible encoding. XOR again with the same chip code recovers the original bit: (bit XOR chip) XOR chip = bit

2.3 How Despreading Works

To retrieve the original bit:

1. The receiver uses the same chip code.
2. XOR each bit of the received spread signal with the chip.
3. Use a majority rule:
   - If most XOR results are 1, the original bit was 1
   - Else, original bit was 0.

This technique is robust to minor errors, as a single chip error won't flip the majority.

2.4 Illustrative Example

Original Text: `A` (from `input1.txt`)

1. **Convert to ASCII:**
   `A` = ASCII value `65`
2. **Convert to 8-bit Binary:**
   `65` = `01000001`
3. **Spreading using Chip Sequence:** `[1, 0, 1]`
   Each bit of the binary string is XORed with the chip sequence:

Bit	Chip Sequence	Spread Output (XOR)
0	1 0 1	1 0 1
1	1 0 1	0 1 0
0	1 0 1	1 0 1
0	1 0 1	1 0 1
0	1 0 1	1 0 1
0	1 0 1	1 0 1
0	1 0 1	1 0 1
1	1 0 1	0 1 0

4. **Final Spread Signal (written to file):**

```
1 0 1  0 1 0  1 0 1  1 0 1  1 0 1  1 0 1  1 0 1  0 1 0
```

5. **Despreading (Receiver Side):** We divide the spread signal into 3-bit chunks: Reconstructed binary:

Table 1: Despreading Steps with Chip Code [1, 0, 1]

Spread Chunk	XOR Result	1s Count	Decision Rule	Recovered Bit
[1,0,1]	[0,0,0]	0	Majority = 0	0
[0,1,0]	[1,1,1]	3	Majority = 1	1
[1,0,1]	[0,0,0]	0	Majority = 0	0
[1,0,1]	[0,0,0]	0	Majority = 0	0
[1,0,1]	[0,0,0]	0	Majority = 0	0
[1,0,1]	[0,0,0]	0	Majority = 0	0
[1,0,1]	[0,0,0]	0	Majority = 0	0
[0,1,0]	[1,1,1]	3	Majority = 1	1

   01000001, Convert to ASCII: 65 → A

6. **Final Output:** recovered_input1.txt contains: A

3 Experiment Overview

In this lab, you are required to design and implement a simulation of the Direct Sequence Spread Spectrum (DSSS) technique, a fundamental concept in digital communication systems. The purpose of this lab is to demonstrate how data can be securely and efficiently transmitted by spreading each data bit using a chip code and later recovering it through despreading.

Your program will simulate this spreading process **by reading input data from one or more text files.** Each character from the file will be converted into its **8-bit binary representation.** Each bit of this binary stream will then be spread using a **user-defined or predefined chip code (a binary sequence of higher frequency).** The spreading will be performed using a bitwise XOR operation between each data bit and the chip code.

The spread signal will be stored in **an output file for each input stream**. A separate receiver-side program (or function) will then read the spread signal, despread it using the same chip code, and recover the original data, which will be **written to a separate output file**. The main requirements of this lab are as follows:

- Read data from one or more text files (e.g., input1.txt, input2.txt).
- Convert each character to its 8-bit binary representation.
- Spread each bit using a predefined chip code (e.g., 101) via XOR.
- Save the resulting spread bit stream to a file (e.g., spread_input1.txt).
- Implement a despreading function that:
  - Divides the spread bits into chip-sized chunks,
  - Performs XOR with the chip code
  - Recovers the original bit using majority logic,
  - Reconstructs the binary into ASCII characters.
- Save the recovered output into a new file (e.g., recovered_input1.txt).
- Print or log both the original, spread, and recovered messages for verification.

3.1 Algorithm: Spreading (Sender Side)

Input: One or more text files (`input1.txt`, `input2.txt`, ...) and a chip sequence (e.g., [1, 0, 1]).
Output: Spread signals written to corresponding `spread_<input>.txt` files.

1. Create a Socket object that takes the IP address and port number as input.
2. Create an object of the DataOutputStream class, which is used to send data to the server side.
3. For each input file:
   (a) Open the file and read character-by-character.
   (b) For each character:
      i. Convert the character to an 8-bit binary string.
      ii. For each bit in the binary string:
         A. XOR the bit with the chip sequence to produce a spread bit sequence.
   (c) Append all spread bits to a list.
   (d) Write the spread bits to `spread_<input>.txt`.
4. Repeat for all files.
5. End.

3.2 Algorithm: Despreading (Receiver Side)

Input: Spread files (`spread_input1.txt`, ...), and the same chip sequence used in spreading.
Output: Recovered original files: `recovered_input1.txt`, ...

1. Create a ServerSocket object, namely handshaking socket, which takes a port number as input.
2. Create a plain Socket object that accepts client requests
3. Create an object of the DataInputStream class, which is used to read data
4. For each `spread_inputX.txt` file:
   (a) Read the file and load the spread bits into an array.
   (b) Divide the spread bit array into chunks of size equal to the chip length.
   (c) For each chunk:
      i. XOR the chunk with the chip sequence.
      ii. Count the number of 1's in the result.
      iii. If the majority of the result is 1, set the original bit to 1.
      iv. Else, set the original bit to 0.
   (d) Group every 8 bits to get the ASCII character.
   (e) Reconstruct the text and write to `recovered_inputX.txt`.
5. Repeat for all spread files.
6. End.

3.3 Input/Output Format

Input:

- One or more plain text files named as `input1.txt`, `input2.txt`, etc.
- Each file contains standard ASCII character data (e.g., letters, digits, punctuation).
- A chip code (e.g., `101` or `110`), either:
  – Provided by the user at runtime, or
  – Hardcoded in the program.

Intermediate Output (Spreader Side):

- For each input file, the spread output is saved in a file named `spread_inputX.txt`.
- The file contains a sequence of bits (0s and 1s), each data bit expanded by the length of the chip sequence.
- Example file content (for input `A` and chip `101`):

```
1 0 1 0 1 0 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 0 1 0
```

Final Output (Despreader Side):

- For each spread file, the recovered output is saved as `recovered_inputX.txt`.
- The file contains the original ASCII characters recovered from the despreading process.
- Example output file content (if original was `A`):

```
A
```

Note:

- The spreading and despreading process should be lossless under noise-free simulation.
- Any mismatch in chip code between sender and receiver may lead to incorrect output.

4 Extended Problem Statement by Introducing Channel Noise Simulation

In this extension of the DSSS experiment, you are required to simulate the effect of channel noise by introducing random bit flips into the spread signal. After spreading a binary message using a chip code, the signal is passed through a simulated channel that randomly flips a certain percentage of bits (e.g., 5–20%).

The receiver must then despread the noisy signal and attempt to recover the original data using majority logic. This experiment demonstrates how DSSS provides inherent fault tolerance, as redundant bits help in recovering the correct data even in the presence of noise.

4.1 Conceptual Foundation

- Each bit is spread using a chip code, e.g., [1, 0, 1].
- This results in multiple bits per data bit, offering redundancy.
- Random bit flips are introduced in the spread signal.
- The despreading process uses majority voting to guess the original bit.
- In most cases, a single or minority flip does not change the majority, so the bit is recovered correctly.

4.2 Channel Noise Simulation

You channel noise simulation, modify your program by including following logic.

- Define a noise probability (e.g., 10% or 20% or 30% chance of flip per bit).
- Traverse the spread signal.
- For each bit, use a random number generator to decide if it will be flipped ($0 \rightarrow 1$, $1 \rightarrow 0$).
- Save the noisy signal.

4.3 Performance Analysis

Now, analyze your program by finding answer of below questions.

- Will redundancy increases tolerance? Normally, A 3-bit chip code can tolerate 1 bit flip per segment.
- Can longer chip codes increase immunity at the cost of bandwidth?

5 Discussion & Conclusion

Based on the focused objective(s), this task will help us to understand the principle and working of Direct Sequence Spread Spectrum and learn to simulate secure and redundant transmission techniques.

6 Policy

Copying from the Internet, classmates, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.