Lab 2: Compression and Huffman Codes, Channel Capacity, and Data Flow

Lab Goals

By the end of this lab, you will:

1. Demonstrate how **compression** affects file size and discuss why it relates to source coding.
2. Work through **guided written problems** on Huffman coding and channel capacity.
3. Explore **LAN and network flow concepts** using your Kali + Metasploitable 2 setup.
4. Bonus: Apply **symmetric and asymmetric encryption** using OpenSSL in Kali.

Part 1: Written Problems

Please refer to the following site, specifically Que 2 through Que - 4, which provide step-by-step instructions on how to solve the following problems below: Practice Questions on Huffman Encoding - GeeksforGeeks

Problem A (Similar to GfG Que-2)

How many bits are required to encode the word **bookkeeper** using Huffman encoding?

Steps:

1. Count frequencies of each character in "bookkeeper".
2. Build Huffman tree (merge least frequent each time, use tie-breaker rule you decide).
3. Assign binary codes to each character.
4. Compute total bits = ∑(frequency × code length).
5. Compute average bits per character (total bits ÷ total characters).

Problem B (Similar to GfG Que-3)

Suppose you have created a Huffman code for *bookkeeper*. Given the following Huffman code (not necessarily unique), **decode** the bit-string:

101 00 110 10 100 1110 00 10 1111 110

Assume these mappings (example):

- b → 00
- o → 01
- k → 10
- e → 110
- p → 1110
- r → 1111

Decode the above string into the corresponding sequence of letters.

Problem C (Similar to GfG Que-4)

If fixed-length encoding is used (say, each character in *bookkeeper* is assigned the same number of bits, enough to cover all distinct characters), how many bits will be saved by using Huffman encoding instead of fixed-length encoding?

Steps:

1. Determine how many distinct characters are in *bookkeeper*.
2. Compute the fixed-length bits: (number of characters in word) × (bits per character under fixed-length).
3. Use your total bits from Problem A.
4. Bits saved = fixed-length total − Huffman total.

Problem D: Channel Capacity of a Noisy Binary Channel

A binary symmetric channel (BSC) has an **error probability** p=0.1p = 0.1p=0.1.
Work through the steps below. Show all work in your report.

Step 1 — Recall the formula
The capacity of a BSC is:

$$C = 1 - H(p)$$

Where:

$$H(p) = -p\log_2 p - (1 - p)\log_2(1 - p)$$

Step 2 — Substitute values
Plug p=0.1p = 0.1p=0.1 into the formula. Write out the full expression for H(p)H(p)H(p) and for CCC, but do not evaluate yet.

Step 3 — Interpret

- What does H(p)H(p)H(p) represent in this context?
- Why is the capacity less than 1 when p>0p > 0p>0?
- What does capacity mean in terms of how much reliable information can be transmitted?

Sep 4 — Optional Extension
Consider if the channel error probability increases to p=0.25p = 0.25p=0.25. Write the new formula for CCC with this value substituted. Do not solve, just set it up.

Part 2: Hands-On Section

A. Compression Demo

1. On Kali, create a file with repetitive data:
2. `echo "AAAAABBBBBCCCCCDDDDD" > test.txt`
3. ***`ls -lh test.txt`***
4. Compress it:
5. ***`gzip -k test.txt`***
6. ***`ls -lh test.txt test.txt.gz`***
7. Record file sizes.
8. Reflect: Why does compression succeed here, and what kind of data would compress poorly?

B. Symmetric Encryption (AES) – IF ANY COMMAND IS OFF, USE GOOGLE. RESEARCHING A COMMAND IS PART OF THE GAME. 😊

1. On Kali:
2. `echo "Secret symmetric demo" > sym.txt`
3. ***`openssl aes-256-cbc -a -salt -pbkdf2 -in sym.txt -out sym.enc`***
4. Decrypt:
5. ***`openssl enc -aes-256-cbc -d -in sym.enc -out sym.dec -k password123`***
6. ***`cat sym.dec`***
7. Take a screenshot of encryption and decryption.

C. Asymmetric Encryption (RSA)

1. Generate keys:
2. ***`openssl genrsa -out private.pem 2048`***
3. ***`openssl rsa -in private.pem -pubout -out public.pem`***
4. Encrypt:
5. ***`echo "Asymmetric demo" > asym.txt`***
6. ***`openssl rsautl -encrypt -inkey public.pem -pubin -in asym.txt -out asym.enc`***
7. Decrypt:
8. ***`openssl rsautl -decrypt -inkey private.pem -in asym.enc -out asym.dec`***
9. ***`cat asym.dec`***
10. Record observations: How does asymmetric differ from symmetric?

D. Channel Capacity in Practice

1) On Metasploitable 2 (the receiver)

Open a terminal and run the listener:

- common OpenBSD-style:

```
nc -l -p 12345 > /tmp/received.bin
```

Leave this running — it will accept one connection and write the incoming bytes to `/tmp/received.bin`.

(If you get "permission denied" use `/tmp` or your home dir.)

2) On Kali (the sender) — measure time while sending

Use `dd` to generate data and pipe it through `nc`. Prepend with `time` to record elapsed time.

Example: send **50 MiB** of zeros:

```
time dd if=/dev/zero bs=1M count=50 | nc <METASPLOITABLE_IP> 12345
```

Notes:

- Replace `<METASPLOITABLE_IP>` with the target IP.
- `count=50` => 50 blocks of 1 MiB = 50 MiB.

When the send completes, `time` prints real/user/sys; record the `real` (elapsed) seconds.

3) On Metasploitable 2 after transfer finishes

Check how many bytes were received:

```
ls -l /tmp/received.bin
stat -c%s /tmp/received.bin    # prints size in bytes
```

4) Compute throughput (precise method)

Let:

- `B` = bytes received (from `stat -c%s`)
- `T` = elapsed time in seconds (from `time` output)

Bits per second:

$$Bps = B \times 8 / T$$

Mbps (megabits per second, decimal):

$$Mbps = bps / 1{,}000{,}000$$

Example (worked):

- Suppose `B = 50 MiB = 50 * 1024 * 1024 = 52,428,800 bytes.`
  (Note: dd reports MiB = 1,048,576 bytes per MB.)
- Suppose `T = 5.0` seconds (measured by `time`).
- Bits = `52,428,800 * 8 = 419,430,400` bits.
- bps = `419,430,400 / 5 = 83,886,080` bps.
- Mbps = `83,886,080 / 1,000,000 = 83.88608 Mbps.`

Compare compressed vs uncompressed transfers (Channel Capacity demo)

Create a real file and compare transfers:

On Kali:

```
# create a 50MB pseudo-file with repeated text (compressible)
yes "BOOKKEEPERBOOKKEEPER" | head -c 50M > test_uncompressed.txt

# create compressed copy
gzip -c test_uncompressed.txt > test_compressed.txt.gz
ls -lh test_uncompressed.txt test_compressed.txt.gz
```

Transfer uncompressed:

- Start receiver on Metasploitable: `nc -l -p 12345 > /tmp/received_uncomp`
- Send from Kali: `time cat test_uncompressed.txt | nc <MS_IP> 12345`

Transfer compressed:

- Receiver: `nc -l -p 12345 > /tmp/received_comp.gz`
- Sender: `time cat test_compressed.txt.gz | nc <MS_IP> 12345`

Compute throughput for both transfers. Compare:

- which finishes faster?
- compressed file smaller → less data to send → apparent effective capacity higher for given physical link.

This demonstrates practical benefit of source coding (Ch.4) and how that affects observed throughput relative to channel capacity.

E. Data Flow in a Network

Goal: Visualize and measure how data flows through your LAN and how delays/queues change performance.

1. **Baseline ping**
    - On Kali:
    - `ping -c 10 <Metasploitable_IP>`
    - Record the average round-trip time (RTT).
2. **Simulate queueing delay**
    - On Kali:
    - `sudo tc qdisc add dev eth0 root netem delay 100ms`
    - `ping -c 5 <Metasploitable_IP>`
    - `sudo tc qdisc change dev eth0 root netem delay 200ms`
    - `ping -c 5 <Metasploitable_IP>`
    - `sudo tc qdisc del dev eth0 root`
    - Record RTTs at 100ms and 200ms delay.
3. **Run iPerf with delay**
    - With 200ms delay active, rerun your iPerf3 throughput test.
    - Record throughput and compare with baseline.
4. **Discussion**
    - Explain how delays represent packets waiting in a queue.
    - Why does throughput drop as delay increases?
    - Relate to M/M/1 or M/D/1 queue models from Ch. 6.

Deliverables

Lab Report

- **Table of Contents**
- **Written Solutions to Problems A-D (1–2 pages):**
    - **You are allowed to insert images of any handwritten solutions you have in this section.**
    - **You are free to go beyond the 2 page limit here, it's just a suggestion**
- **2 – 3-page Analysis Section**
    - Discuss what you learned from the Written Solutions section and the Hands-on section of this lab.
    - I want an analysis of the concepts you learned about in chapters 4 – 6 and this lab, not a replay of the steps you completed.

- **Appendix:**
  - Screenshots from compression, AES & RSA