Lab03
Computer Networks (CS 252)

Aman Kansal
170050027

Ansh Khurana
170050035

February 10, 2019

Design

1.1 Physical Layer

1.1.1 Encoding

- Instead of voltage levels used in wired connections, the bits are encoded using colours.
- Bits are represented using colours. (red - '1' and blue - '0')
- The colours are flashed on the computer screen and recorded by the webcam of the second laptop.
- Bits are detected via a transition, i.e the reference state is green and a transition to red denotes '1' while a transition to blue denotes '0'. This improves reliability in detection and synchronizes clocks on both the machines automatically.
- A computer program detects which colour is displayed on the screen and hence receives the bit.
- Visible light is the medium used for communication. A wireless link is established between the screen and the camera of different computers. This link is half duplex in nature.

1.2 Link layer

1.2.1 Framing

- The message to be transmitted is padded to make it 27 bits long. This makes our scheme work independent of the size of the message. (any size <= 26 bits works)
- As padding we add a single '1' followed by the required no. of zeros to the end of the message.
- Then we add 27 redundant bits for error correction. These are added at the end of the data part. Thus, total size of the packet is 54 bits.

1.2.2 Reliability

- We are using forward error correction, thus our scheme does away with the need of any acknowledgements (assuming no packet loss) and delays caused by them.
- The technique used for error correction is a 3-D bit parity scheme.(an extension of 2-D bit parity that we learnt in class)
- Parity bit is added among all dimensions of a message is which is reshaped as a cube of width $n^{2/3}$, where n is the number of bits in the message.
- In general if the message length is n then no. of parity bits need to be added are $3 * n^{2/3}$. Hence the code rates tends to 1 for large values of n. (Our method is scalable)

Implementation

2.1 Physical Layer

- The physical layer takes the encoded bit string (as received from the link layer) and displays the corresponding sequence of images on the screen.
- Each colour is visible on the screen for *bit_duration* milliseconds. (A free parameter to adjust the throughput)
- A program running on the receiver (rec.py) measures the intensity of RGB channels in the recorded images and classifies them as red, green or blue. The program also detects transitions by maintaining two states of images. Once the message is received, the message is passed to the Link Layer for decoding and correction.

2.2 Link Layer

- A script (encode.py) takes the input message and error positions and returns the encoded message.
- The message is padded appropriately and then reshaped into a 3*3*3 numpy array.
- Then we simply calculate the parity bits along row, column and depth.
- The bit cube is flattened to produce bit string of length 54 and sent to the physical layer for transmission.
- Once the message is received on the other side, it is passed to a script called decode.py.
- Again, the received message is converted into a numpy array and the parity bits are calculated accordingly. The differing parity bits locate the flipped bits in the received message. After correction, the original message is recovered and displayed on the screen.

Evaluation

3.1 Experimental setup and procedure

- The two laptops are separated by a distance of 80 centimetres with the screens facing each other.
- The sender runs the python script send.py (In application/ folder) with appropriate inputs.
- The receiver runs rec.py (In application/ folder) and select the ROI for the current position of the laptop.
- A variable called the *bit_duration* is set at the beginning of the transmission, which fixes the time for which a specific colour is displayed on the screen.
- Once transmission of both the messages is completed, the messages (corrected) are displayed on the screen along with the positions where the bits were in error.

3.2 Metrics and Results

3.2.1 Lab demo metric

- This metric is calculated by using the time taken for completion of the experiment once the sender is handed over the message.
- In the experiment we measure this by taking the time taken from the instant the sender executes send.py and types in the input, till the receiver program terminates
- The time taken in experiment we did for practice took 98 seconds.
- The throughput taking this time into account comes out to be 1.102 bits/second. (108 bits sent over 98 seconds)

3.2.2 End-to-End throughput

- E2E Throughput is defined as: total number of bits transmitted / (time last message received – time first message generated)
- In our experiment, total number of transmitted bits were 108.
- Time taken = Calibration time + 2 * bit_duration * 108 + Time interval between the two messages
- In our experimental setup, Calibration time = 10 secs, bit_duration = 0.25 sec, time interval = 3 secs.
- Thus, E2E throughput comes out to be 2.7 bits/second.