Lab 2: Packet Losses and Their Impact on Streaming Video Quality

Student ID: 2357943 Full name: Zexi_Li

TASK: ANALYSIS OF PACKET LOSS IMPACT ON STREAMING VIDEO QUALITY
#1 [**30 points**] Read the types and sizes of 10,000 video frames from the *The Silence of the Lambs* trace and generate symbol loss sequences for two symbol loss rates (pL) of 1×10^{-4} and 1×10^{-3} using the SGM with p=1×10^{-4} . Based on the procedure described in Sec. II, calculate DFR for each loss rate.
#2 [**25 points**] Repeat #1 with RS(204, 188, t=8).
#3 [**25 points**] Repeat #2 with convolutional interleaving/deinterleaving.
#4 [**20 points**] Generate a plot comparing the resulting DFRs (similar to the one shown in the Lab slides for DFR) and discuss the advantages and disadvantages of using RS code and/or convolutional interleaving/deinterleaving.

- For convolutional interleaving/deinterleaving part:

```
if ci:
    # apply convolutional interleaving/deinterleaving.
    # N.B.:
    # 1. Append 2244 zeros before interleaving.
    # 2. Interleaved sequence experiences symbol losses.
    # 3. Remove leading 2244 elements after deinterleaving.
    
    if fec:
        x1 = np.concatenate([np.ones(204 * n_pkts), np.zeros(2244)])
    else:
        x1 = np.concatenate([np.ones(188 * n_pkts), np.zeros(2244)])
    d1 = [int(i) for i in "17,34,51,68,85,102,119,136,153,170,187".split(',')]
    d2 = d1[::-1]
    x2 = conv_interleave(x1, d1)    # convolved symbols
    loss_seq1 = sgm_generate(random_seed, len(x2), p, q)    # symbol loss sequence 1: loss, 0: not loss
    y = x2 * loss_seq1    # symbols affected by losses
    z = conv_deinterleave(y, d2)
    loss_seq2 = z[2244:]    # remove prepended zeros (i.e., those from SRSs)

    # TODO: Implement.
else:    # not convolutional interleaving/deinterleaving (CI)
    x1 = np.ones(188 * n_pkts)
    loss_seq1 = sgm_generate(random_seed, len(x1), p, q)    # symbol loss sequence 1: loss, 0: not loss
    loss_seq2 = x1 * loss_seq1    # symbols affected by losses
    # TODO: Implement.
```

- For FEC part(RS code):

```
if fec:
    # to see each pkt in ith frame, and see if any loss in current pkt
    for k in range(0, f_pkts[i]):
        # check each pkt from current ith frame
        pkt = loss_seq2[(num_pkts_received - k - 1) * 204: (num_pkts_received - k) * 204]

        pkt_list = list(pkt)
        # if current pkt for current frame has more than 8 symbols loss
        # then it means that current pkt is loss and so that current ith frame is loss
        # and do not need to see the rest of pkt of ith frame
        if pkt_list.count(1) > 8:
            frame_loss = True    # current frame is loss
            break
        else:
            frame_loss = False    # current frame is not loss

         # TODO: Set "frame_loss" based on "pkt_losses" with FEC.

else:    # not fec
    for k in range(0, f_pkts[i]):
        # check each pkt from current ith frame
        pkt = loss_seq2[(num_pkts_received - k - 1) * 188: (num_pkts_received - k) * 188]

        pkt_list = list(pkt)
        # if current pkt for current frame has more than 1 symbols loss
        # then it means that current pkt is loss and so that current ith frame is loss
        # and do not need to see the rest of pkt of ith frame
        if pkt_list.count(1) >= 1:
            frame_loss = True    # current frame is loss
            break
        else:
            frame_loss = False    # current frame is not loss
    # TODO: Set "frame_loss" based on "pkt_losses" without FEC.
```

- Compute DFR

```
# frame decodability
if not frame_loss:    # see the fec-dependent handling of "frame_loss" above.
    match f_type[i]:
        case 'I':
            # current I frame is decodable
            num_frames_decoded += 1

            # For the cases to B frame: 'IBBBP' or 'PBBBP' or 'PBBBI',
            # check the flag whether is False(if False, it means
            # the previous I or P frame for B frame is decodable),
            # and so it need to check I frame behind B frame by checking the flag I_loss
            if B1_frame_decodeable == False and I_loss == False:
                num_frames_decoded += 1
                # print('B decode')
                B1_frame_decodeable = False
            if B2_frame_decodeable == False and I_loss == False:
                num_frames_decoded += 1
                # print('B decode')
                B2_frame_decodeable = False
            if B3_frame_decodeable == False and I_loss == False:
                num_frames_decoded += 1
                # print('B decode')
                B3_frame_decodeable = False
            # TODO: Implement.

        case 'P':
            # for the case: 'IBBBP' or 'PBBBP' or 'IPBBB'
            # IBBBP: if the current index i - 4 == i_frame_number(the lase decodable I frame
index),
            # it means the previous I frame for current P frame is decodable and so that
current P frame
            # is decodable. The other situations are similar.
            if i - 4 == i_frame_number or i - 4 == p_frame_number or i - 1 == i_frame_number:
                num_frames_decoded += 1

            # For the cases to B frame: 'IBBBP' or 'PBBBP' or 'PBBBI',
            # check the flag whether is False(if False, it means
            # the previous I or P frame for B frame is decodable),
            # and so it need to check P frame behind B frame by checking the flag P_loss
            if B1_frame_decodeable == False and P_loss == False:
                num_frames_decoded += 1

                B1_frame_decodeable = True
            if B2_frame_decodeable == False and P_loss == False:
                num_frames_decoded += 1

                B2_frame_decodeable = True
            if B3_frame_decodeable == False and P_loss == False:
                num_frames_decoded += 1

                B3_frame_decodeable = True
            # TODO: Implement.
        case 'B':
            # for the first B frame in case 'IPBBB'
            # IPBBB: if the current index i - 2 == i_frame_number(the lase decodable I frame
index),
            # and i - 1 == p_frame_number(the lase decodable P frame index),
            # it means the previous I and P frame for current B frame are decodable
            # and so that current B frame is decodable. The other situations are similar.
            if i - 2 == i_frame_number and i - 1 == p_frame_number:
                num_frames_decoded += 1
                continue

            # for the second B frame in case 'IPBBB'
            if i - 3 == i_frame_number and i - 2 == p_frame_number:
                num_frames_decoded += 1
                continue

            # for the third B frame in case 'IPBBB'
            if i - 4 == i_frame_number and i - 3 == p_frame_number:
                num_frames_decoded += 1
                continue

            # for the first B frame in case 'IBBBP' or 'PBBBP' or 'PBBBI'
            # IBBBP: if the current index i - 1 == i_frame_number(the lase decodable I frame
index),
            # it means the previous I frame for current B frame is decodable.
            # So it need to check the I or P frame behind current B frame
            # whether is decodable or not. The other situations are similar.
            if i - 1 == i_frame_number or i - 1 == p_frame_number:
                # if the previous I or P frame is decodable,
                # set flag to False to wait for the future I or P frame to see if decodable
                B1_frame_decodeable = False
                continue

            # for the second B frame in case 'IBBBP'
            if i - 2 == i_frame_number or i - 2 == p_frame_number:
                # if the previous I or P frame is decodable,
                # set flag to False to wait for the future I or P frame to see if decodable
                B2_frame_decodeable = False
                continue

            # for the third B frame in case 'IBBBP'
            if i - 3 == i_frame_number or i - 3 == p_frame_number:
                # if the previous I or P frame is decodable,
                # set flag to False to wait for the future I or P frame to see if decodable
                B3_frame_decodeable = False
                continue
            # TODO: Implement.

        case _:
            sys.exit("Unkown frame type is detected.")

if f_type[i] == 'I':
    if frame_loss:
        I_loss = True    # I_loss: the flag indicates whether current I frame is decodable
    else:
        I_loss = False
        # update:the index of the last decodable I frame,
        # it means current I frame is not loss and decodable
        i_frame_number = i
if f_type[i] == 'P':
    if frame_loss:
        P_loss = True    # P_loss: the flag indicates whether current P frame is decodable
    else:
        P_loss = False
        # update:the index of the last decodable P frame,
        # it means current P frame is not loss and decodable
        p_frame_number = i
```

- Plot figure

```
# obtain data
DFR = np.zeros((4, 3))
for i in range(0, 4):
    if i == 0:
        loss_probability = 1e-4
    elif i == 1:
        loss_probability = 4e-4
    elif i == 2:
        loss_probability = 7e-4
    else:
        loss_probability = 1e-3
    for j in range(0, 3):
        if j == 0:
            fec = False
            ci = False
        elif j == 1:
            fec = True
            ci = False
        else:
            fec = True
            ci = True
        DFR[i][j] = dfr_simulation(random_seed=777, num_frames=10000,
loss_probability=loss_probability,
                                   video_trace=video, fec=fec, ci=ci)

colors = ['red', 'blue', 'green', 'black']
labels = ['No FEC, NO ci', 'FEC, NO ci', 'FEC, ci']
[m, n] = np.shape(DFR)
x = [1e-4, 4e-4, 7e-4, 1e-3]
# plot figure
for i in range(0, n):
    plt.scatter(x[0], DFR[0][i], color=colors[0], marker='o')
    plt.scatter(x[1], DFR[1][i], color=colors[1], marker='^')
    plt.scatter(x[2], DFR[2][i], color=colors[2], marker='x')
    plt.scatter(x[3], DFR[3][i], color=colors[3], marker='h')
    #
    array = np.array(DFR)
    plt.plot(x, array[:, i], label=labels[i], linestyle='--')
```

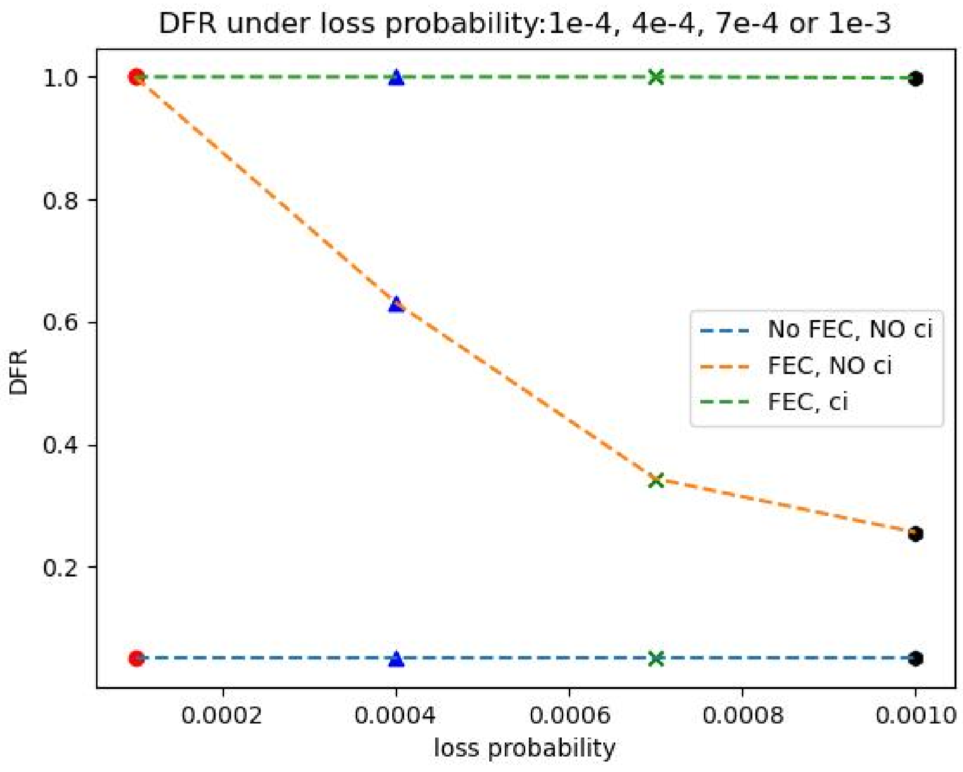

Fig1. comparing DFR under the different loss probabilities

Discuss :

According to Fig1, it is obvious that DFR would be very low with any loss probabilities, meaning that most of frames are loss.

When using FEC(RS code) but not convolutional interleaving/deinterleaving, when the loss probability gets large(1e-3), the effect seems limited: the loss errors are slightly improved. As the loss probability decreases, the effect grows better. Finally, when the loss probability gets small(1e-4), the effect is so significant that almost all loss errors are corrected, and nearly no losses happen in sequence.

When using FEC(RS code) and convolutional interleaving/deinterleaving, if the loss probability gets small(1e-4), since FEC improves DFR so great that can not find whether convolutional interleaving/deinterleaving helps or not in improving DFR; but when using it in loss probability with 1e-3, it is clear that convolutional interleaving/deinterleaving indeed makes a great impact in improving DFR, which jumps to almost 1.

Based on the above finding, the advantage of using RS code is that it raises DFR, especially when loss probability is low since the number of consecutive multi-bit errors would be small in this case. However, the disadvantage is that when the loss probability becomes larger, the number of packets that symbol errors exceeding the maximum error correcting limit of RS code also increases (here, it refers to an increase in the number of packets that symbol errors exceeding 8 within a packet), so the effect becomes limited. The advantage of using convolutional interleaving/deinterleaving is that it can help to improve DFR significantly when the effect of RS code is a little because it disperses consecutive multi-bit errors into different packets, which lets the number of symbol errors decreases below 8 within a packet so that FEC(RS code) can detect and correct errors easily. However, the disadvantage is that it needs to process data in advance with complex steps and takes lots of time.