Introduction to Computer Networks

Homework #2

Submission by:

- Alexander Shender 328626114
- Netanel Rotschild 204937841

Question 1.

a.

The calculation of the throughput is similar to the exercise in the Tutorial lecture, where *n* packets were sent every time. In our case, n=2. The equations are still fully developed here:

$$M - random\ variable\ symbolizing\ no. of\ frames\ sent$$

$$M\ is\ distibuted\ Geometrically$$

Given that we send 2 frames together:

$$P = p^2 \rightarrow probability\ in\ failure\ of\ sending\ message$$

Thus:

$$E(M) = \frac{1}{1-P} = \frac{1}{1-p^2}$$

Calculating the throughput:

$$S = \frac{T_i}{T_v} = \frac{T_i}{T_t E(M)} = \frac{T_i}{(2T_i + T_{out})} \cdot \frac{(1-p^2)}{1} = \frac{1-p^2}{2 + \frac{T_{out}}{T_i}} = \frac{1-p^2}{2+\beta}$$

$$S_{new} = \frac{1-p^2}{2+\beta}$$

b.

As we can see, the throughput is dependent on 2 parameters – β and p. By taking the throughput from the original protocol, which is:

$$S_{original} = \frac{1-p}{1+\beta}$$

We are required to find the p for which the equations holds:

$$S_{new} \leq S_{original} \rightarrow \frac{1-p^2}{2+\beta} \leq \frac{1-p}{1+\beta}$$

For different β values, the p which results in the inequality is different. We use the graph to visualize. Using python, the throughput values were calculated for various values. 2 graphs are shown:

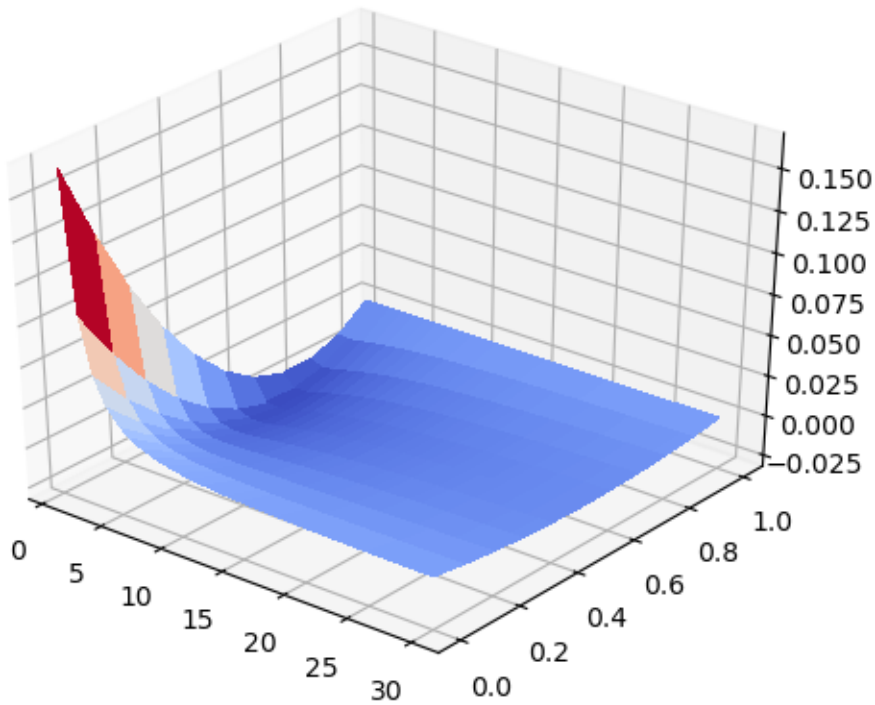
The first graph shows the result of the difference of the throughput values:

$$S_{original} - S_{new}$$

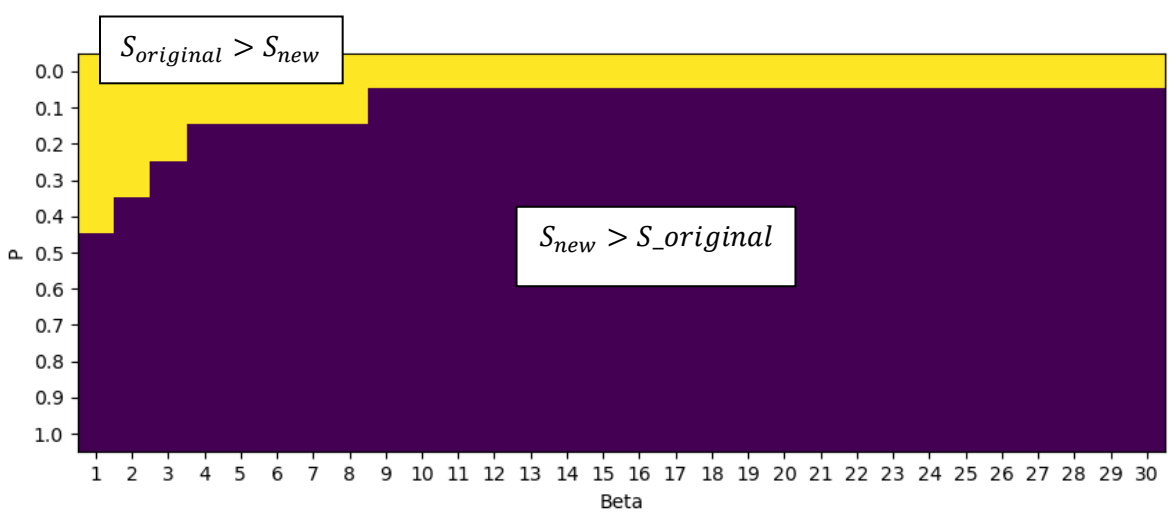
Where this values is positive, meaning using the original protocol is beneficial. Ranges used:

$$1 \leq \beta \leq 30 \quad ; \quad 0 \leq p \leq 1.0$$

Result:



As we can see, where the probability of packet not being delivered is low, and Beta is low (propagation time is small), the original algorithm is more beneficial. To be sure, another binary chart shows yellow bricks where $S_{original} > S_{new}$: (and purple if other)



We can observe that:

- With high error probability, the new protocol is beneficial

- If the error value is low, for certain Beta values we prefer the original algorithm
- If the error probability is 0, original protocol is better

All of those conclusions are also intuitive.

c.

Now, in the equation for the throughput calculation, $S = \frac{T_i}{T_v}$, what changes is the T_v value.
Recalculating:

$$T_v = P_1[1st\ message\ success]] \cdot (T_{out} + T_i) + P_2[first\ message\ fail\ \&\ second\ message\ success] \cdot (T_{out} + 2T_i) = (1-p) \cdot (T_{out} + T_i) + \big(p(1-p)\big) \cdot (T_{out} + 2T_i)$$

$$T_v = T_i(1-p) + 2T_i p(1-p) + T_{out}\big((1-p) + p(1-p)\big)$$

$$T_v = T_i(1 - p + 2p - 2p^2) + T_{out}(1 - p + p - p^2)$$

$$T_v = T_i(1 + p - 2p^2) + T_{out}(1 - p^2)$$

To validate our result, we calculate the average different between the average message in the original protocol, and in the protocol in this case:

$$T_v original = (2T_i + T_{out})(1 - p^2)$$

$$T_v original - T_v new = (2T_i + T_{out})(1 - p^2) - T_i(1 + p - 2p^2) + T_{out}(1 - p^2) = T_i\big(2(1 - p^2) - (1 + p - p^2)\big) = T_i(1 - p)$$

We indeed see that on average the length of the successful transmission is less by $T_i(1-p)$:

- If error probability is 0, we always save T_i time!
- As error probability increases, we save less time.

Which are intuitive conclusions.

Question 2.

a.

First, calculating T_i, T_{ack}, T_p:

$$T_i = \frac{message\ length\ [bits]}{R\ \left[\frac{bits}{sec}\right]} = \frac{512 * 8}{5} \cdot 10^{-6} = \ 0.0008192\ [s] = 819.2\ [us]$$

$$T_{ack} = \frac{ack\ length\ [bits]}{R\ \left[\frac{bits}{sec}\right]} = \frac{16 * 8}{5} \cdot 10^{-6} = \ 25.6 \cdot 10^{-6} [s] = 25.6\ [us]$$

$$T_p = \frac{distance}{propagation\ speed} = \frac{2000 \cdot 10^3 [m]}{2 \cdot 10^8\ \left[\frac{m}{sec}\right]} = 0.01\ [s]$$

The optimal T_{out} :

$$T_{out} = RTT + T_{ack} = 2 \cdot T_p + T_{ack} = \ 0.0200256\ [s]$$

This is the optimal time. If after this time the 'ack' is not received, it is the sign that the message (or ack) didn't reach the destination.

b.

The throughput is given by:

$$S_{S\&W} = \frac{T_i}{T_v} = ;\ \ T_v = T_t E(M)$$

Where T_t is the total time for 1 message for being processed

$$T_t = T_i + T_{out}$$

M is a random variable showing the number of times the message is sent. Distributed geometrically with the probability of error p. Thus, $E(M)$:

$$E(M) = \frac{1}{1-p}$$

$$S_{S\&W} = \frac{T_i}{T_i + T_{out}}(1-p) = \ 0.039(1-p)$$

c.

The optimal window size would be the one, which can send maximal amount of messages before the T_{out} for the first message has to arrive. This way, if there was no ack received, the first message will be resent. Calculating β:

$$\beta = \frac{RTT}{T_i} = \frac{T_{out}}{T_i} = \frac{(2T_p + T_{ack})}{T_i} = 24.4$$

We use the ceil() function to round it up:

$$ceil(\beta) = 25$$

Thus, the window size is given by:

$$T_{window} = (ceil(\beta) + 1) \cdot T_i = \ 0.0213\ [s]$$

d.

Given: $p = 0.3$

The throughput is given again by:

$$S_{GBN} = \frac{T_i}{T_v}$$

As in tutorial, we define k as number of times the message is resent. Then, for a given k, the time until a message is transmitted:

$$T_k = T_i + k(T_{out} + T_i)$$

The average time for 1 message to finish:

$$T_v = E(T_k) = T_i + E(k) \cdot (T_{out} + T_i)$$

As in tutorial,

$$E(k) = \frac{p}{1-p}$$

Thus,

$$T_v = T_i + \frac{p}{1-p} \cdot (T_{out} + T_i)$$

$$S_{GBN} = \frac{T_i}{T_v} = \frac{T_i}{T_i + \frac{p}{1-p} \cdot (T_{out} + T_i)} = 0.0824$$

Calculating the $S_{S\&W}$:

$$S_{S\&W} = \frac{T_i}{T_i + T_{out}}(1-p) = \ 0.039(1-p) = 0.0275$$

As we can see:

$$S_{GBN} > S_{S\&W}$$

Question 3.

a.

The scenario is similar to the one explained in the lecture, where it is promised that second package will reach the receiver with 100% certainty. To calculate the optimal window size, we take into account T_i', which is not equal to T_i.

$$sender\ window\ size = T_i + T_i' + 4T_p = T_i \cdot \left(1 + \frac{T_i'}{T_i} + 2\beta\right)$$

Any increase in this window size will not benefit the throughput.

b.

The receiver optimal size is then:

$$receiver\ window\ size = T_p + T_i' + T_p = T_i \cdot \left(\frac{T_i'}{T_i} + \beta\right)$$

c.

Calculating the T_v:

P is the probability of error on the first packet.

Summing up the probabilities of getting each transmission time for the packet to get the average transmission time for 1 packet.

$$T_v = \underbrace{T_i \cdot (1 - P)}_{no\ error\ on\ 1st} + \underbrace{(T_i + T_i') \cdot P}_{error\ on\ 1st, resend} = T_i + T_i' P = T_i \cdot \left(1 + \frac{T_i'}{T_i} P\right)$$

d.

Finding the throughput through the equation:

$$S = \frac{T_i}{T_v} = \frac{T_i}{T_i \cdot \left(1 + \frac{T_i'}{T_i} P\right)} = \frac{1}{1 + \frac{T_i'}{T_i} P}$$

Question 4.

a.

In S&W, the change is done to T_t, total time to transmit 1 message, and to probability of error $p \to p'$.

The throughput calculation is then:

$$T_v = T_t E(M) = \frac{(T_i + T_h + RTT)}{1 - p'}$$

$$S_{new} = \frac{T_i(1 - p')}{T_i + T_h + RTT} = \frac{1 - p'}{1 + \beta + \frac{T_h}{T_i}}$$

We can see that we cannot say for sure whether the throughput increases or decreases:

- Decreasing p' will increase S_{new}
- Increasing T_h will decrease S_{new}

We can elaborate a condition which will ensure $S_{new} > S_{old}$:

$$\frac{1 - p'}{1 + \beta + \frac{T_h}{T_i}} > S_{old}$$

$$1 - p' > S_{old}\left(1 + \beta + \frac{T_h}{T_i}\right)$$

$$1 - p' > S_{old}(1 + \beta) + S_{old}\frac{T_h}{T_i}$$

$$1 - p' > \frac{1 - p}{1 + \beta}(1 + \beta) + S_{old}\frac{T_h}{T_i}$$

$$p - p' > S_{old}\frac{T_h}{T_i}$$

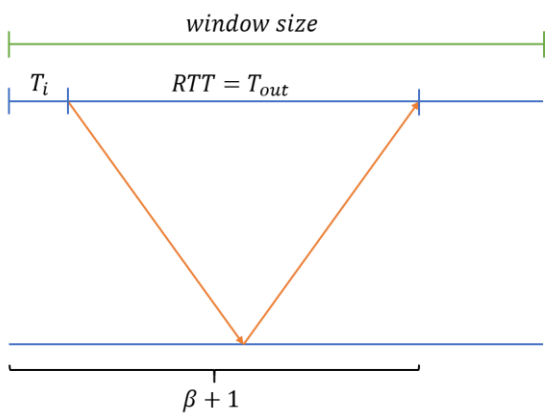
When the last condition is satisfied, the new throughput is bigger than the original.

b.

In GBN, if window size is less than $\beta + 1$, the throughput **decreases** for sure. There will be an 'idle' time, when the first package still hasn't returned ack, but small window size doesn't allow to send another packet, and the system will simply wait for ack from first package and in the meanwhile will not transmit

c.

The window increase beyond $\beta + 1$ will not affect the throughput in this case. The packets beyond the $\beta + 1$ size will be ready to be sent, but not be sent until the correct ACK was received for the first packet from sender ($SN_{\min}$), which will happen after RTT. Thus, it will **not be different** from a case where the window size is $\beta + 1$. The situation is presented below:

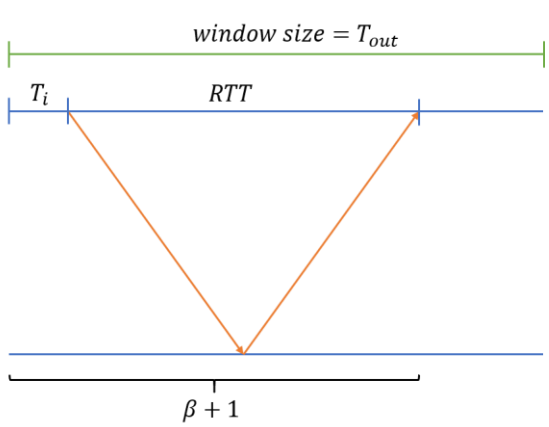


Summing up: $S_{GBN\ original} = S_{GBN\ new}$

d.

In this case, the sender will keep on sending packets until the T_{out}, and only then discover that there was no ACK for message i . Which will make him resend the whole window of size T_{out}, instead of window of size $T_i + RTT$, or $T_i(1 + \beta)$. This protocol **does NOT gain improvement** in performance (the sender will always transmit also in the original GBN), but loses when the packet is not received correctly ($p > 0$).

The situation is depicted below:



Summing up: $S_{GBN\ original} > S_{GBN\ new}$, and **throughput decreases**

e.

Each packet is sent twice in GBN. From intuition, this can be beneficial is p is large, and can be redundant when p is small. Thus, it is impossible to know without more information.

f.

Infinite window size on both ends means that the sender will send packets all the time. Thus, the throughput **will not change** if $T_{out} > RTT$, since the packet will be resent anyway, if T_{out} is a finite time. In SR, the throughput is not affected by other packets, since only the failed packet will be resent if case it failed to arrive (or the ACK was lost).

g.

We need a calculation here. Let k be the number of resends we have to make. Then, the mean amount is:

$$E(k) = \underbrace{P\left(\frac{RTT}{T_i} - 1\right)}_{will\ send\ at\ least\ this\ amount} + \sum_{k=1}^{\infty} \underbrace{k \cdot P \cdot (1-P)}_{failed\ at\ sending\ first\ time} \cdot \underbrace{P^{k-1}}_{failed\ at\ next\ times}$$

The first part of the equation $\left(\frac{RTT}{T_i} - 1\right)$ is here because the sender will send the failed packet until the indication for it was received, this means at least this amount of times.

Calculating the average time for 1 packet is then:

$$T_v = T_i + E(k) \cdot T_i$$

We multiply by T_i since we already summed the Timeout to the E(k).

$$E(k) = P \cdot (\beta - 1) + P(1-P)\sum_{k=0}^{\infty} kP^{k-1} = \frac{P}{1-P} + P(\beta - 1) = P\left(\frac{1}{1-P} + (\beta - 1)\right)$$

$$T_v = T_i\left(1 + P\left(\frac{1}{1-P} + (\beta - 1)\right)\right)$$

$$S_{new} = \frac{1}{1 + P\left(\frac{1}{1-P} + (\beta - 1)\right)}$$

This throughput is of course less than the throughput of the original SR protocol $S_{SR} = 1 - P$. Intuitively, the protocol does a redundant job by sending over and over a package until the first ACK. While the good package may be on its way to receiver, or its ACK back to sender, the sender may have already sent same packages which will now be thrown. So the throughput will decrease

APPENDIX – python code

Question 1

```
import numpy as np
from matplotlib import pyplot as plt

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import numpy as np



def main():


    beta_list = np.linspace(start = 1, stop = 30, num = 30, dtype = int)
    fail_prob_list =  np.linspace(start = 0, stop = 1.0, num = 11, dtype=float)


    calc_s_new = lambda B, p : (1-(p**2)) / (2 + B)
    calc_s_original = lambda B, p: (1-p) / (1 + B)

    s_new_arr          = np.empty((len(beta_list),len(fail_prob_list)))
    s_original_arr      = np.empty((len(beta_list),len(fail_prob_list)))
    s_difference        = np.empty((len(beta_list),len(fail_prob_list)))
    s_difference_binary = np.empty((len(beta_list),len(fail_prob_list)))

    for B_idx, B in enumerate(beta_list):
        for p_idx, p in enumerate(fail_prob_list):
            s_new_arr[B_idx][p_idx] = calc_s_new(B,p)
            s_original_arr[B_idx][p_idx] = calc_s_original(B,p)
            s_difference[B_idx][p_idx] = s_original_arr[B_idx][p_idx]  - s_new_arr[B_idx][p_idx]

            if s_difference[B_idx][p_idx] > 0 :
                s_difference_binary[B_idx][p_idx] = 1
            else:
                s_difference_binary[B_idx][p_idx] = 0

    # for the plot
    s_new_rot = np.fliplr(np.rot90(s_new_arr, axes=(1,0)))
    s_original_rot = np.fliplr(np.rot90(s_original_arr, axes=(1,0)))
    s_diff_rot = np.fliplr(np.rot90(s_difference, axes=(1,0)))
    s_binary_rot = np.fliplr(np.rot90(s_difference_binary, axes=(1,0)))

    beta_grid, p_grid = np.meshgrid(beta_list, fail_prob_list)

    fig = plt.figure()
    ax = fig.gca(projection='3d')
```

```
    # Plot the surface.
    surf = ax.plot_surface(beta_grid, p_grid, s_diff_rot, cmap=cm.coolwarm,
                           linewidth=0, antialiased=False)

    plt.show()



    fig, ax = plt.subplots()
    ax.imshow(s_binary_rot)

    ax.set_xticks(np.arange(len(beta_list)))
    ax.set_yticks(np.arange(len(fail_prob_list)))

    ax.set_xticklabels(beta_list)
    ax.set_yticklabels(np.round(fail_prob_list, 3))

    ax.set_xlabel("Beta")
    ax.set_ylabel("P")

    fig.tight_layout()
    plt.show()


if __name__ == '__main__':
    main()
```

Question 2

```
import math



def print_var(var_name : str):
    print(f"\t{var_name} = {eval(var_name)}")

dist = 2 * 10**6
R = 5 * 10**6

msg_len = 512*8
ack_len = 16*8

speed = 2*(10**8)


print("\nA:")
# SNW

T_i = msg_len/R
T_ack = ack_len/R
T_p = dist / speed

print_var("T_i")
print_var("T_ack")
print_var("T_p")

RTT = T_p*2
T_out = RTT + T_ack

print_var("T_out")

print("\nB:")
# calculating throughput

T_t = T_i + T_out
print_var("T_t")
M1_avg = lambda p : 1/(1-p)


T_i_over_T_t = T_i / T_t
print_var("T_i_over_T_t")

beta_exact = RTT / T_i

throughput_SNW = lambda p : T_i_over_T_t / M1_avg(p)
throughput_SNW_2 = lambda p : (1-p) / (1 + beta_exact)


print("\nC:")
```

```
# GBN

beta_round = math.ceil(beta_exact)

print_var("beta_exact")
print_var("beta_round")

n = beta_round + 1
T_window = n * T_i
print_var("T_window")


print("\nD:")

p = 0.3

S_SWN = throughput_SNW(p)
S_SWN_2 = throughput_SNW(p)

print_var("S_SWN")
print_var("S_SWN_2")

E_k = lambda p : p / (1-p)

T_v = T_i + E_k(p) * (T_window)

print_var("T_v")

S_GBN = T_i / T_v

throughput_GBN_2 = lambda p : 1 / (1 + (p/(1-p)*(beta_round + 1)))
S_GBN_2 = throughput_GBN_2(p)

print_var("S_GBN")
print_var("S_GBN_2")
```