Tarea 2 - Redes: Medidor ancho de banda UDP Go-Back-N

Pablo Skewes

Realización del experimento

In [1]:
```
from pathlib import Path
from tqdm import tqdm
import os
import sys
import time
import numpy as np
import time
import pandas as pd
import matplotlib.pyplot as plt
import subprocess
from contextlib import contextmanager
from sklearn.model_selection import ParameterGrid

import jsockets
from go_back_n import compute_bandwidth
from timer import Timer
```

Definamos las globales para conectarnos al servidor.

In [2]:
```
FILEPATH = Path('/etc/services').resolve()
SERVER_URL = 'anakena.dcc.uchile.cl'
SERVER_PORT = '1819'
```

Podemos usar el "main" del programa para realizar un experimento de prueba y ver que todo funciona correctamente.

In [6]:
```
socket = jsockets.socket_udp_connect(SERVER_URL, SERVER_PORT)

data = compute_bandwidth(
    socket=socket,
    packet_size=4000,
    timeout=10,
    loss=0.1,
    window_size=99,
    filepath=FILEPATH,
    verbose=False,
)

data
```

Out[6]:
```
{'data_size': 12813,
 'bytes_received': 12813,
 'send_time': 0.07861900329589844,
 'bandwidth': 0.155425895824741}
```

Vemos que recuperamos un diccionario con los resultados como esperábamos, ahora podemos realizar el experimento completo para distintas combinaciones de valores.

In [7]:
```
packet_sizes = list(range(1000, 10000, 1000)) + [9999]
window_sizes = [1, 5, 10, 20]
losses = [0, 0.1, 0.2, 0.3, 0.4, 0.5]
timeouts = [1, 5, 10, 20, 50, 100, 200, 500, 1000]

params = ParameterGrid({
    'packet_size': packet_sizes,
    'window_size': window_sizes,
    'loss': losses,
    'timeout': timeouts,
})

print(f"En total hay {len(params)} combinaciones de parámetros")
```

```
En total hay 2160 combinaciones de parámetros
```

In [31]:
```
results = []

for i, param in tqdm(enumerate(params), total=len(params)):
    verbose = False

    init_time = time.time()
    try:
        socket = jsockets.socket_udp_connect(SERVER_URL, SERVER_PORT)
        if socket is None:
            print("No se pudo abrir el socket")
            sys.exit(1)

        if verbose:
            print(f"Experiment {i+1}/{len(params)}")
            print(f"Parameters: {param}")

        data = compute_bandwidth(
            socket=socket,
            packet_size=param['packet_size'],
            timeout=param['timeout'],
            loss=param['loss'],
            window_size=param['window_size'],
            filepath=FILEPATH,
            verbose=verbose,
        )

        result = {
            'data_size': data['data_size'],
            'bytes_received': data['bytes_received'],
            'send_time': data['send_time'],
            'bandwidth': data['bandwidth'],
            'packet_size': param['packet_size'],
            'window_size': param['window_size'],
            'loss': param['loss'],
            'timeout': param['timeout'],
```

```
        }

    except Exception as e:
        # print("Experiment failed with parameters:")
        # print(param)
        result = {
            'data_size': np.nan,
            'bytes_received': np.nan,
            'send_time': np.nan,
            'bandwidth': np.nan,
            'packet_size': param['packet_size'],
            'window_size': param['window_size'],
            'loss': param['loss'],
            'timeout': param['timeout'],
        }

    results.append(result)

    time.sleep(0.1)
```

```
100%|██████████| 2160/2160 [26:40<00:00,  1.35it/s]
```

In [ ]:
```
socket = jsockets.socket_udp_connect(SERVER_URL, SERVER_PORT)

data = compute_bandwidth(
    socket=socket,
    packet_size=4000,
    timeout=10,
    loss=0.3,
    window_size=1,
    filepath=FILEPATH,
    verbose=True,
)
```

In [36]:
```
df = pd.DataFrame.from_records(results)
```

In [3]:
```
# df.to_csv('results.csv', index=False)
df = pd.read_csv('results.csv')
```

Análisis de Resultados

Experimentos fallidos

In [45]:
```
perc = df['bandwidth'].isna().sum() / len(df)
print(f"Porcentaje de experimentos fallidos: {perc:.2%}")

failed = df[df['bandwidth'].isna()]
```

```
Porcentaje de experimentos fallidos: 10.97%
```

Out[45]:

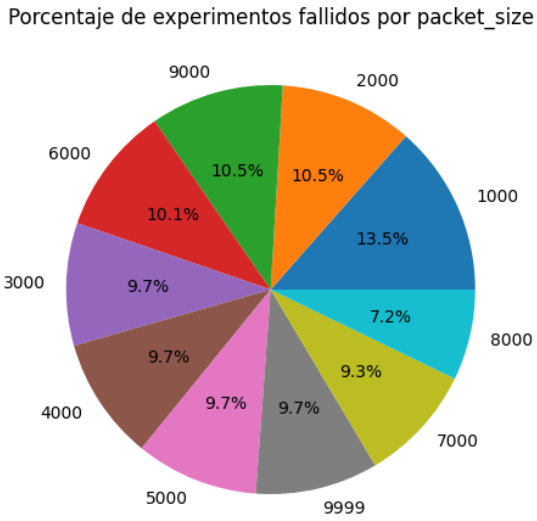
	data_size	bytes_received	send_time	bandwidth	packet_size	window_size	los
0	NaN	NaN	NaN	NaN	1000	1	0
2	NaN	NaN	NaN	NaN	1000	10	0
3	NaN	NaN	NaN	NaN	1000	20	0
36	NaN	NaN	NaN	NaN	2000	1	0
37	NaN	NaN	NaN	NaN	2000	5	0
...	...	...	...	...	...	...	
2090	NaN	NaN	NaN	NaN	9000	10	0
2091	NaN	NaN	NaN	NaN	9000	20	0
2124	NaN	NaN	NaN	NaN	9999	1	0
2125	NaN	NaN	NaN	NaN	9999	5	0
2126	NaN	NaN	NaN	NaN	9999	10	0

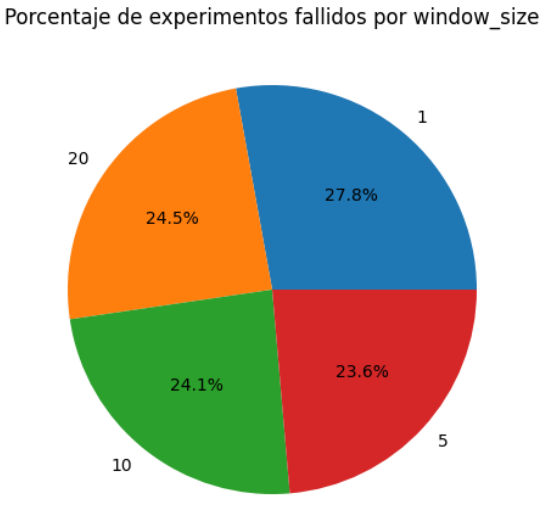
237 rows × 8 columns

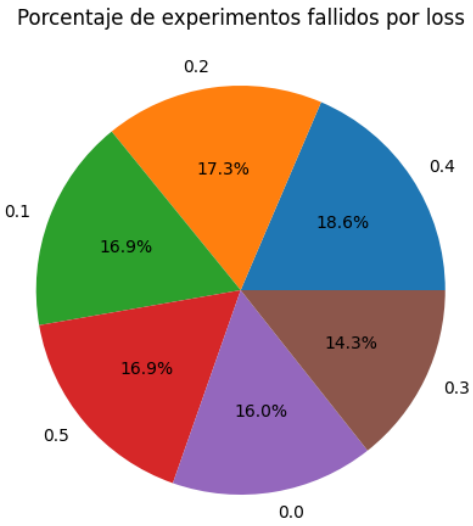
In [57]:

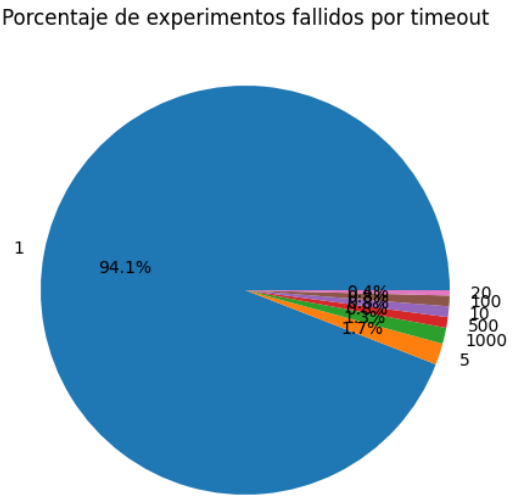
```
fig, axs = plt.subplots(2, 2, figsize=(12, 12))
axs = axs.flatten()

for i, param in enumerate(['packet_size', 'window_size', 'loss', 'timeout
    failed[param].value_counts().plot.pie(autopct='%1.1f%%', ax=axs[i])
    axs[i].set_title(f'Porcentaje de experimentos fallidos por {param}')
    axs[i].set_ylabel('')
    axs[i].set_xlabel('')
```









Podemos ver que de 2100 experimentos aproximadamente, unos 200 fallaraon y fueron principalmente por problemas de timeout, parece ser que 1ms es muy poco para que el servidor alcance a responder.

Ahora podemos eliminar los experimentos que fallaron para analizar los resultados.

In [58]:
```
df = df.dropna().reset_index(drop=True)
```

Pérdida de paquetes

In [63]:
```
(df['data_size'] - df['bytes_received']).sum()
```

Out[63]:
```
0.0
```

De los datos podemos ver que ningún experimento tuvo pérdida de paquetes, esto es porque el protocolo Go-Back-N tiene un mecanismo de retransmisión de paquetes que permite recuperarse de pérdidas de paquetes. Aunque no podremos estar 100% seguros de que no haya habido ningún error, podemos decir que la probabilidad de que haya ocurrido es muy baja y que los experimentos fueron exitosos.

In [35]:
```
df
```

Out[35]:

	data_size	bytes_received	send_time	bandwidth	packet_size	window_size	los
0	12813.0	12813.0	0.025270	0.483550	1000	5	0
1	12813.0	12813.0	0.137159	0.089089	1000	1	0
2	12813.0	12813.0	0.025814	0.473372	1000	5	0
3	12813.0	12813.0	0.027213	0.449024	1000	10	0
4	12813.0	12813.0	0.015599	0.783358	1000	20	0
...	...	...	...	...	...	...	
1918	12813.0	12813.0	2.083974	0.005864	9999	20	0
1919	12813.0	12813.0	0.033921	0.360227	9999	1	0
1920	12813.0	12813.0	2.053771	0.005950	9999	5	0
1921	12813.0	12813.0	3.020338	0.004046	9999	10	0
1922	12813.0	12813.0	1.026118	0.011908	9999	20	0

1923 rows × 8 columns

Comparación con otros clientes

Stop and Wait

In [4]:
```
def get_number_regx(s):
    return ''.join([c for c in s if c.isdigit()])

@contextmanager
def supress_stdout():
    old_stdout = sys.stdout
    try:
        sys.stdout = open(os.devnull, 'w')
        yield
    finally:
        sys.stdout = old_stdout
```

In [5]:
```
def execute_stop_and_wait(size: int, timeout: int, loss: float, filepath:
    try:
        cmd = f"python bwc-sw.py {size} {timeout} {loss} {filepath} {serv
        with supress_stdout():
            output = subprocess.check_output(cmd, shell=True)
        output = output.decode('utf-8')
        output = output.split('\n')[-2]
        output = output.split(', ')
    except Exception as e:
        print(e)
        return {
            'bytes_received': np.nan,
            'send_time': np.nan,
            'bandwidth': np.nan,
```

```
        }
    return {
        'bytes_received': int(get_number_regx(output[0])),
        'send_time': float(get_number_regx(output[1])),
        'bandwidth': float(get_number_regx(output[2]))
    }

data = execute_stop_and_wait(
    size=5000,
    timeout=20,
    loss=int(0.1 * 100),
    filepath=FILEPATH,
    server_url=SERVER_URL,
    server_port=SERVER_PORT,
)

data
```

```
propuse paquete: 5000
recibo paquete: 5000
```

Out[5]:
```
{'bytes_received': 12813,
 'send_time': 6043672561645508.0,
 'bandwidth': 2.021854905518955e+16}
```

Logramos transformar el script dado en clases para el cliente StopAndWait en una función que devuelva los valores en un diccionario para poder compararlos con los resultados del cliente Go-Back-N.

In [8]:
```
print(f"Vamos a probar el cliente con los siguientes parámetros:")
print(f"{packet_sizes=}")
print(f"{timeouts=}")
print(f"{losses=}")
```

```
Vamos a probar el cliente con los siguientes parámetros:
packet_sizes=[1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 9999]
timeouts=[1, 5, 10, 20, 50, 100, 200, 500, 1000]
losses=[0, 0.1, 0.2, 0.3, 0.4, 0.5]
```

In [9]:
```
params_stop_and_wait = ParameterGrid({
    'packet_size': packet_sizes,
    'timeout': timeouts,
    'loss': losses,
})
print(f"En total hay {len(params_stop_and_wait)} combinaciones de parámet
```

```
En total hay 540 combinaciones de parámetros
```

In [10]:
```
results_stop_and_wait = []
for i, param in tqdm(enumerate(params_stop_and_wait), total=len(params_st
    verbose = False
    try:
        data = execute_stop_and_wait(
            size=param['packet_size'],
            timeout=param['timeout'],
            loss=int(param['loss'] * 100),
            filepath=FILEPATH,
            server_url=SERVER_URL,
```

```
            server_port=SERVER_PORT,
        )

        # print(data)

        result = {
            'data_size': 12813.0,
            'bytes_received': data['bytes_received'],
            'send_time': data['send_time'],
            'bandwidth': data['bandwidth'],
            'packet_size': param['packet_size'],
            'timeout': param['timeout'],
            'loss': param['loss'],
        }

    except Exception as e:
        # print("Experiment failed with parameters:")
        # print(param)
        # traceback = sys.exc_info()[2]
        # print(f"Traceback: {traceback.tb_frame.f_code.co_filename}:{tra
        result = {
            'data_size': np.nan,
            'bytes_received': np.nan,
            'send_time': np.nan,
            'bandwidth': np.nan,
            'packet_size': param['packet_size'],
            'timeout': param['timeout'],
        }

    results_stop_and_wait.append(result)

    time.sleep(0.1)
```

```
  0%|          | 0/540 [00:00<?, ?it/s]propuse paquete: 1000
recibo paquete: 1000
Traceback (most recent call last):
  File "/home/pabloskewes/Desktop/FCFM/Redes/Tarea 2/bwc-sw.py", line 150,
in <module>
    send_loss(s, b"D" + to_seq(cnt) + data)
  File "/home/pabloskewes/Desktop/FCFM/Redes/Tarea 2/bwc-sw.py", line 60,
in send_loss
    s.send(data)
TimeoutError: timed out
  0%|          | 1/540 [00:00<02:41,  3.33it/s]
Command 'python bwc-sw.py 1000 1 0 /etc/services anakena.dcc.uchile.cl 181
9' returned non-zero exit status 1.
```

```
propuse paquete: 9000
Traceback (most recent call last):
  File "/home/pabloskewes/Desktop/FCFM/Redes/Tarea 2/bwc-sw.py", line 133,
in <module>
    PACK_SIZE = to_num(data[3], data[4], data[5], data[6])
                       ~~~~^^^
TypeError: 'NoneType' object is not subscriptable
 97%|▉▉▉▉▉▉▉▉▉ | 526/540 [28:00<01:54,  8.16s/it]propuse paquete: 9000
recibo paquete: 9000
Command 'python bwc-sw.py 9000 20 50 /etc/services anakena.dcc.uchile.cl 1
819' returned non-zero exit status 1.
 98%|▉▉▉▉▉▉▉▉▉ | 527/540 [28:01<01:16,  5.89s/it]propuse paquete: 9000
recibo paquete: 9000
 98%|▉▉▉▉▉▉▉▉▉ | 528/540 [28:05<01:04,  5.37s/it]propuse paquete: 9000
Traceback (most recent call last):
  File "/home/pabloskewes/Desktop/FCFM/Redes/Tarea 2/bwc-sw.py", line 133,
in <module>
    PACK_SIZE = to_num(data[3], data[4], data[5], data[6])
                       ~~~~^^^
TypeError: 'NoneType' object is not subscriptable
 98%|▉▉▉▉▉▉▉▉▉ | 529/540 [28:15<01:14,  6.81s/it]
Command 'python bwc-sw.py 9000 200 50 /etc/services anakena.dcc.uchile.cl
1819' returned non-zero exit status 1.
propuse paquete: 9000
recibo paquete: 9000
 98%|▉▉▉▉▉▉▉▉▉ | 530/540 [28:25<01:16,  7.68s/it]propuse paquete: 9000
Traceback (most recent call last):
  File "/home/pabloskewes/Desktop/FCFM/Redes/Tarea 2/bwc-sw.py", line 133,
in <module>
    PACK_SIZE = to_num(data[3], data[4], data[5], data[6])
                       ~~~~^^^
TypeError: 'NoneType' object is not subscriptable
 98%|▉▉▉▉▉▉▉▉▉ | 531/540 [28:35<01:15,  8.43s/it]
Command 'python bwc-sw.py 9000 1000 50 /etc/services anakena.dcc.uchile.cl
1819' returned non-zero exit status 1.
 99%|▉▉▉▉▉▉▉▉▉ | 532/540 [28:36<00:50,  6.26s/it]
Command 'python bwc-sw.py 9999 1 50 /etc/services anakena.dcc.uchile.cl 18
19' returned non-zero exit status 1.
 99%|▉▉▉▉▉▉▉▉▉ | 533/540 [28:41<00:41,  5.94s/it]
Command 'python bwc-sw.py 9999 5 50 /etc/services anakena.dcc.uchile.cl 18
19' returned non-zero exit status 1.
propuse paquete: 9999
recibo paquete: 9000
 99%|▉▉▉▉▉▉▉▉▉ | 534/540 [28:42<00:25,  4.29s/it]propuse paquete: 9999
recibo paquete: 9000
 99%|▉▉▉▉▉▉▉▉▉ | 535/540 [28:42<00:15,  3.12s/it]propuse paquete: 9999
Traceback (most recent call last):
  File "/home/pabloskewes/Desktop/FCFM/Redes/Tarea 2/bwc-sw.py", line 133,
in <module>
    PACK_SIZE = to_num(data[3], data[4], data[5], data[6])
                       ~~~~^^^
TypeError: 'NoneType' object is not subscriptable
 99%|▉▉▉▉▉▉▉▉▉ | 536/540 [28:52<00:20,  5.25s/it]propuse paquete: 9999
recibo paquete: 9000
Command 'python bwc-sw.py 9999 50 50 /etc/services anakena.dcc.uchile.cl 1
819' returned non-zero exit status 1.
```

```
 99%|██████████| 537/540 [28:53<00:11,  3.92s/it]propuse paquete: 9999
Traceback (most recent call last):
  File "/home/pabloskewes/Desktop/FCFM/Redes/Tarea 2/bwc-sw.py", line 133,
in <module>
    PACK_SIZE = to_num(data[3], data[4], data[5], data[6])
                       ~~~~^^^
TypeError: 'NoneType' object is not subscriptable
100%|██████████| 538/540 [29:03<00:11,  5.80s/it]
Command 'python bwc-sw.py 9999 200 50 /etc/services anakena.dcc.uchile.cl
1819' returned non-zero exit status 1.
propuse paquete: 9999
Traceback (most recent call last):
  File "/home/pabloskewes/Desktop/FCFM/Redes/Tarea 2/bwc-sw.py", line 133,
in <module>
    PACK_SIZE = to_num(data[3], data[4], data[5], data[6])
                       ~~~~^^^
TypeError: 'NoneType' object is not subscriptable
100%|██████████| 539/540 [29:13<00:07,  7.11s/it]
Command 'python bwc-sw.py 9999 500 50 /etc/services anakena.dcc.uchile.cl
1819' returned non-zero exit status 1.
propuse paquete: 9999
Traceback (most recent call last):
  File "/home/pabloskewes/Desktop/FCFM/Redes/Tarea 2/bwc-sw.py", line 133,
in <module>
    PACK_SIZE = to_num(data[3], data[4], data[5], data[6])
                       ~~~~^^^
TypeError: 'NoneType' object is not subscriptable
100%|██████████| 540/540 [29:24<00:00,  3.27s/it]
Command 'python bwc-sw.py 9999 1000 50 /etc/services anakena.dcc.uchile.cl
1819' returned non-zero exit status 1.
```

In [11]:
```
stop_and_wait_df = pd.DataFrame.from_records(results_stop_and_wait)
```

In [13]:
```
stop_and_wait_df = stop_and_wait_df.dropna().reset_index(drop=True)
```

In [14]:
```
stop_and_wait_df
```

Out[14]:

	data_size	bytes_received	send_time	bandwidth	packet_size	timeout	los
0	12813.0	12813.0	1.100440e+16	1.110413e+16	1000	5	0
1	12813.0	12813.0	1.771064e+16	6.899485e+15	1000	10	0
2	12813.0	12813.0	1.104748e+16	1.106083e+16	1000	20	0
3	12813.0	12813.0	1.091242e+16	1.119773e+16	1000	50	0
4	12813.0	12813.0	1.217692e+16	1.003491e+15	1000	100	0
...	...	...	...	...	...	...	
381	12813.0	12813.0	3.014477e+15	4.053581e+15	9000	100	0
382	12813.0	12813.0	4.554467e+15	2.682955e+16	9000	500	0
383	12813.0	12813.0	4.411912e+15	2.769645e+16	9999	10	0
384	12813.0	12813.0	2.509015e+14	4.870210e+16	9999	20	0
385	12813.0	12813.0	6.556120e+15	1.863820e+15	9999	100	0

386 rows × 7 columns

In [15]:
```
print(f"Recuperamos {len(stop_and_wait_df)} experimentos exitosos")
```

```
Recuperamos 386 experimentos exitosos
```

In [21]:
```
(stop_and_wait_df['data_size'] - stop_and_wait_df['bytes_received']).valu
```

Out[21]:
```
0.0        383
4813.0       1
7813.0       1
10813.0      1
Name: count, dtype: int64
```

Hubieron a penas 3 casos que perdieron bytes. Pero los vamos a eliminar para poder comparar los resultados.

In [23]:
```
stop_and_wait_df = stop_and_wait_df[stop_and_wait_df['data_size'] == stop
```

In [24]:
```
stop_and_wait_df
```

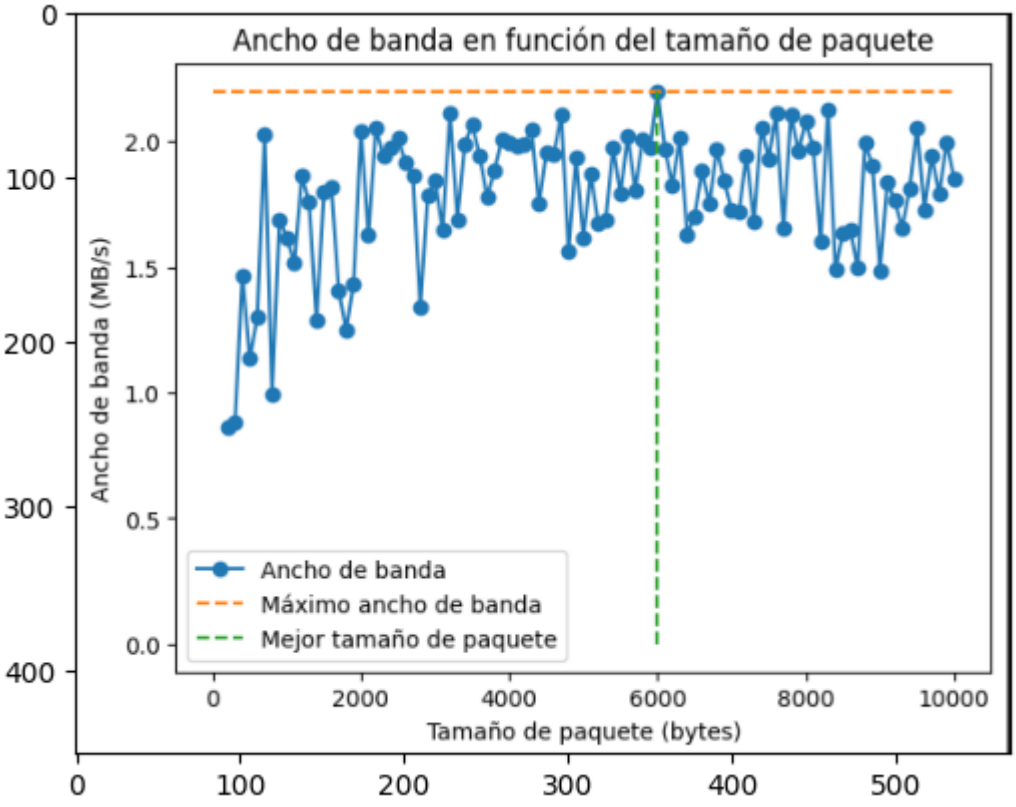
Out[24]:

	data_size	bytes_received	send_time	bandwidth	packet_size	timeout	los
0	12813.0	12813.0	1.100440e+16	1.110413e+16	1000	5	0
1	12813.0	12813.0	1.771064e+16	6.899485e+15	1000	10	0
2	12813.0	12813.0	1.104748e+16	1.106083e+16	1000	20	0
3	12813.0	12813.0	1.091242e+16	1.119773e+16	1000	50	0
4	12813.0	12813.0	1.217692e+16	1.003491e+15	1000	100	0
...	...	...	...	...	...	...	
378	12813.0	12813.0	3.014477e+15	4.053581e+15	9000	100	0
379	12813.0	12813.0	4.554467e+15	2.682955e+16	9000	500	0
380	12813.0	12813.0	4.411912e+15	2.769645e+16	9999	10	0
381	12813.0	12813.0	2.509015e+14	4.870210e+16	9999	20	0
382	12813.0	12813.0	6.556120e+15	1.863820e+15	9999	100	0

383 rows × 7 columns

De la Tarea 1 podemos tomar el gráfico ya existente para el análisis

In [29]:
```
plt.imshow(plt.imread('tarea1.png'));
```



Con todo esto podemos responder las preguntas.

Pregunta 1

Genere algunos experimentos con diversos tamaños de paquete, venta- na, timeout y pérdidas y haga una recomendación de valores a utilizar para las distintas pérdidas. Compare con la Tarea1 y con el cliente stop-and-wait. Grafique sus resultados.

Lo que queremos estudiar es el ancho de banda, por lo que vamos a graficar esto en el eje z siempre, y el resto de los parámetros los podremos representar de distinta forma:

- en el eje x: tamaño de paquete
- en el eje y: probabilidad de pérdida
- en el color: timeout
- en el tamaño de los puntos: tamaño de ventana

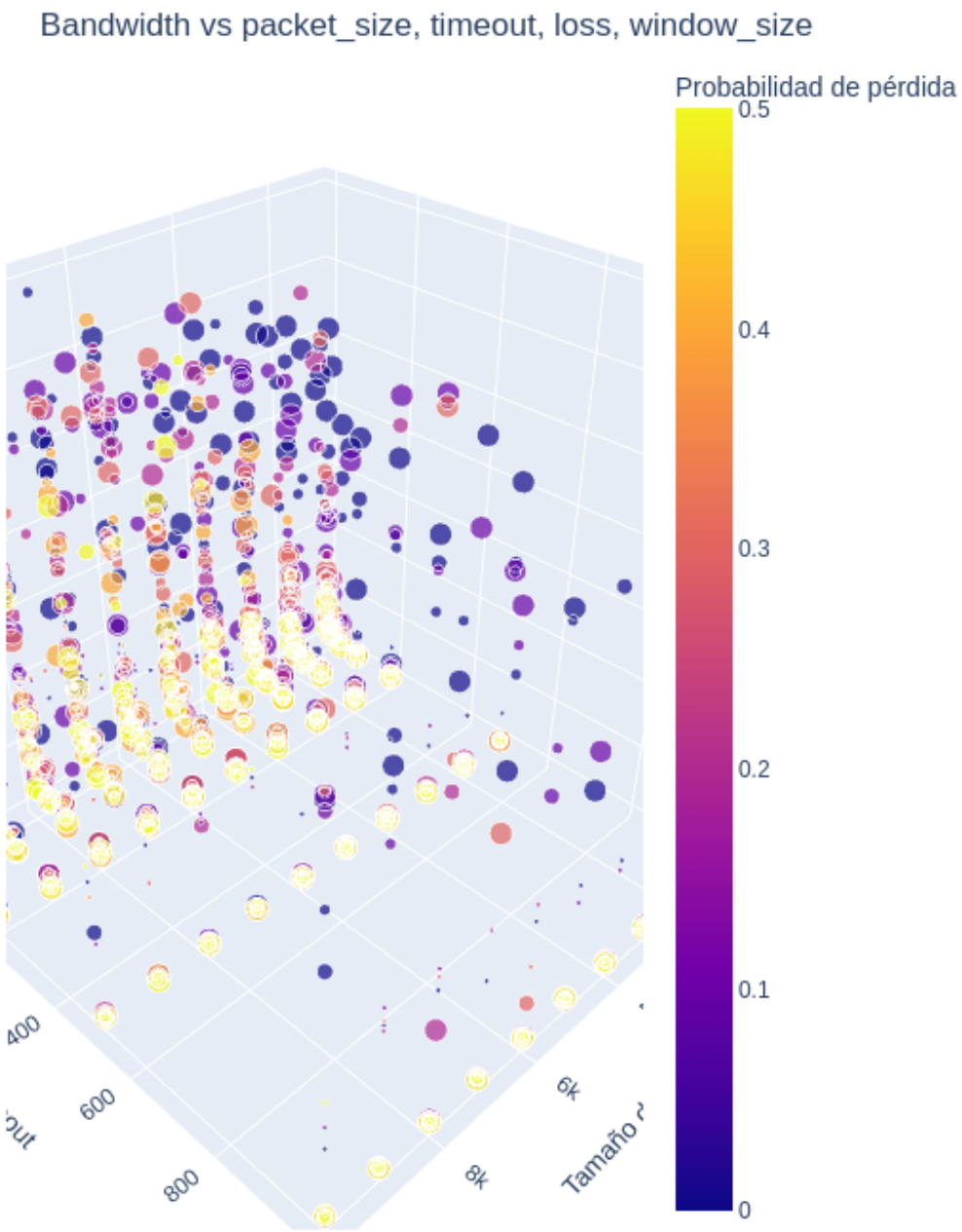
In [41]:
```
import plotly.express as px
from copy import deepcopy
```

In [39]:
```
fig = px.scatter_3d(
    df,
    x='packet_size',
    y='timeout',
    z='bandwidth',
    color='loss',
    opacity=0.7,
    size='window_size',
    hover_data=['send_time'],
    title='Bandwidth vs packet_size, timeout, loss, window_size',
    labels={
        'packet_size': 'Tamaño de paquete',
        'bandwidth': 'Ancho de banda',
        'timeout': 'Timeout',
        'loss': 'Probabilidad de pérdida',
        'window_size': 'Tamaño de ventana',
        'send_time': 'Tiempo de envío',
    },
    width=1000,
    height=800,
)

fig.update_layout(
    scene = dict(
        xaxis_title='Tamaño de paquete',
        yaxis_title='Timeout',
        zaxis_title='Ancho de banda',
    ),
    scene_aspectmode='cube',
    title={
        'text': 'Bandwidth vs packet_size, timeout, loss, window_size',
        'y': 0.9,
        'x': 0.5,
        'xanchor': 'center',
        'yanchor': 'top'
    },
)
```

```
print()
```

In [44]:
```
fig.show()
```

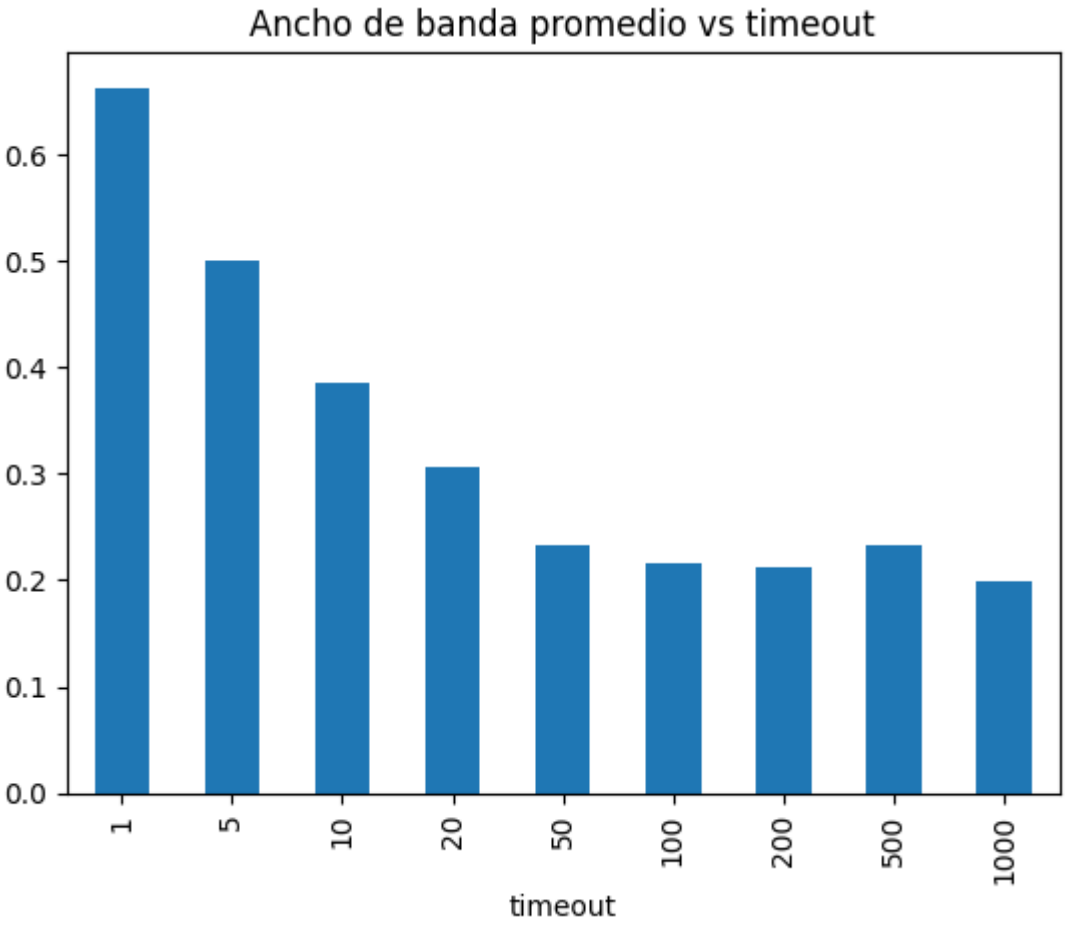


Del gráfico podemos concluir lo siguiente:

- Claramente se ve que a mayor probabilidad de pérdida, menor es el ancho de banda. Esto es porque el protocolo tiene que retransmitir más paquetes.
- El tamaño de ventana se distribuye de forma uniforme o un poco aleatoria, no se ve una tendencia clara y parece depender de los otros parámetros.

- El timeout tampoco se ve una tendencia clara, asi que podemos plotearlo más en específico.

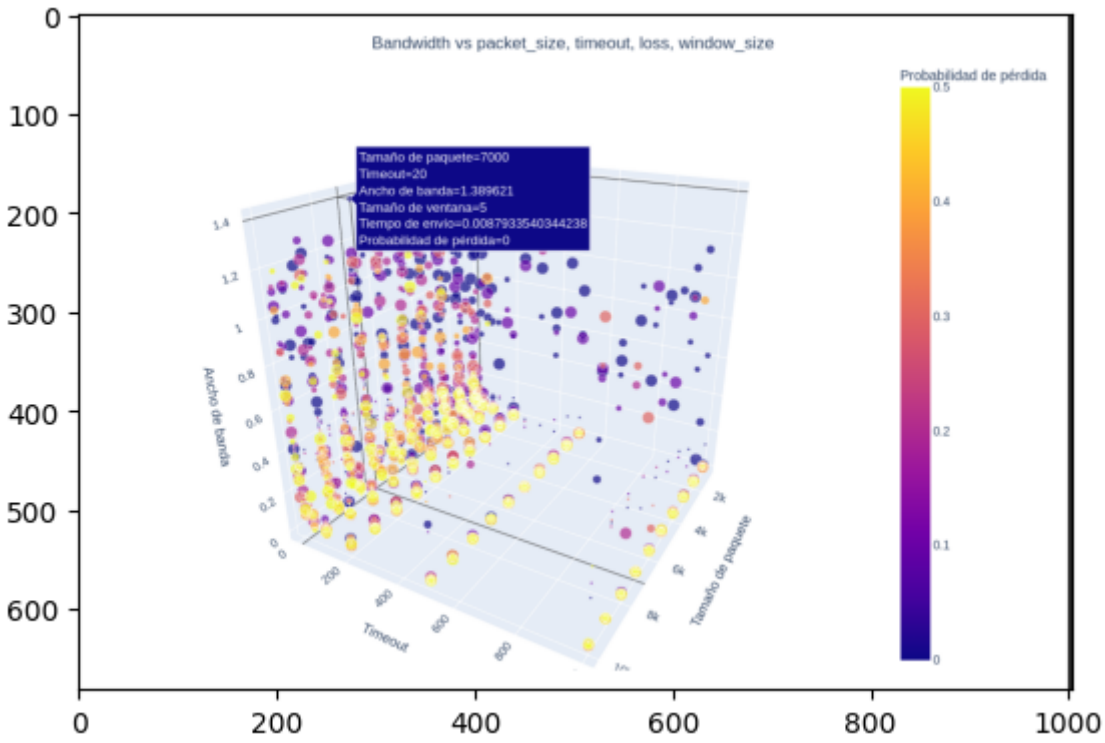
In [53]:
```
df.groupby('timeout')['bandwidth'].mean().plot.bar(title='Ancho de banda
```



Vemos que a menor timeout mejor se aprovecha el ancho de banda, esto es porque el protocolo tiene que esperar menos tiempo para retransmitir paquetes. Sin embargo ya vimos que muchos experimentos fallaron para timeout=1, por lo que es mejor descartarlo.

Los puntos interesantes a analizar puede ser:

In [57]:
```
plt.imshow(plt.imread('optimo.png'));
```



El set de parámetros óptimo encontrado para este experimento es:

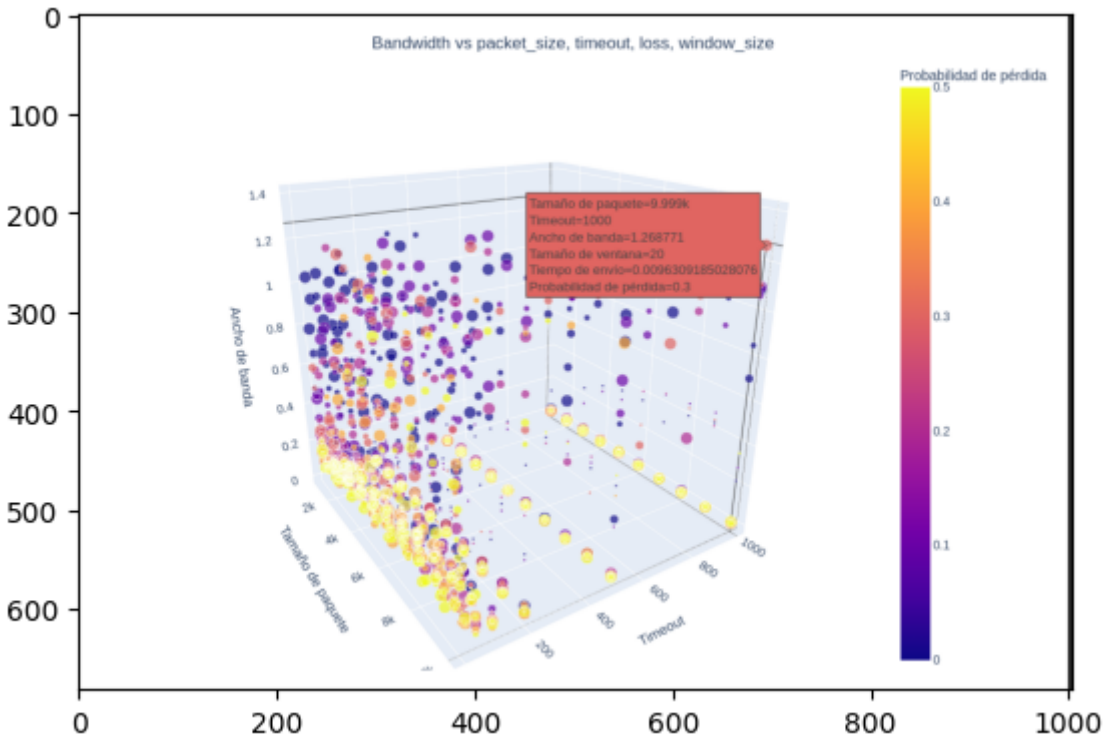
- packet_size = 7000
- timeout = 20
- window_size = 5
- loss_prob = 0 (claramente)

Obteniendo un ancho de banda de 1.38 Mbps.

Esto se condice con el gráfico de la primera tarea, donde el óptimo se encuetra para un packet_size parecido. Cabe mencionar que los datos tienen ruido a causa de la red, por lo que no podemos asegurar que estos sean los valores óptimos, pero si podemos decir que son los mejores que encontramos.

También nos interesaría estudiar un caso donde haya una probabilidad de pérdida no nula, para ver como se comporta el protocolo cuando las condiciones no son ideales.

In [58]:
```
plt.imshow(plt.imread('alto_loss.png'));
```



En este caso encontramos el siguiente set de parámetros:

- packet_size = 9999
- timeout = 1000
- window_size = 20
- loss_prob = 0.3

Encotramos un ancho de banda de 1.27 Mbps, que es bastante parecido al caso anterior. Esto es porque el protocolo se recupera de las pérdidas de paquetes, por lo que no se ve tan afectado por ellas. Para esto el protocolo prefirió enviar la mayor cantidad de información posible a la vez: el mayor packet_size posible + un buen timeout para esperar el envio de los paquetes + una ventana grande.

Ahora comparemoslo con el stop and wait:

In [65]:
```
stop_and_wait_df['bandwidth'].describe()
```

Out[65]:
```
count    3.830000e+02
mean     1.532408e+16
std      1.473602e+16
min      3.017255e+13
25%      3.687372e+15
50%      9.685045e+15
75%      2.307348e+16
max      6.455643e+16
Name: bandwidth, dtype: float64
```

Claramente estos valores no tienen sentido, por lo que en realidad no se logró usar el cliente stop and wait para este experimento. Pero podemos usar los datos para window_size=1 para compararlos con el cliente Go-Back-N, Ya que este es equivalente al stop and wait.

In [66]:
```
stopwait = df[df['window_size'] == 1].reset_index(drop=True)
```

In [67]:
```
stopwait['bandwidth'].describe()
```

Out[67]:
```
count    474.000000
mean       0.189270
std        0.159912
min        0.000867
25%        0.031116
50%        0.157953
75%        0.325661
max        0.546041
Name: bandwidth, dtype: float64
```

In [70]:
```
con_perdida = stopwait[stopwait['loss'] == 0.3]
sin_perdida = stopwait[stopwait['loss'] == 0]
```

In [76]:
```
con_perdida['bandwidth'].describe()
```

Out[76]:
```
count    80.000000
mean      0.145051
std       0.140508
min       0.001208
25%       0.023036
50%       0.106999
75%       0.237465
max       0.520732
Name: bandwidth, dtype: float64
```

In [77]:
```
sin_perdida['bandwidth'].describe()
```

Out[77]:
```
count    80.000000
mean      0.306648
std       0.116558
min       0.089089
25%       0.217224
50%       0.326369
75%       0.403148
max       0.525344
Name: bandwidth, dtype: float64
```

De aquí podemos ver que tanto para los casos con pérdida como sin pérdida, el cliente Go-Back-N tiene un mejor ancho de banda que el cliente Stop and Wait. Esto es porque el cliente Go-Back-N puede enviar más de un paquete a la vez, mientras que el cliente Stop and Wait solo puede enviar uno a la vez.

Pregunta 2

Discuta si la medición de ancho de banda obtenida es ”válida”: al haber retransmisiones, ¿estamos midiendo mal? Los resultados obtenidos: ¿son coherentes? ¿se parecen a la realidad? Argumente con sus propias mediciones realizadas en el punto anterior.

Si bien el protocolo tiene retransmisiones, estas son necesarias para poder recuperarse de las pérdidas de paquetes. Por lo que no estamos midiendo mal, ya que el protocolo está diseñado para funcionar así. Es cierto que las condiciones son simuladas y que son bastante variables como para poder asegurar que los resultados son válidos, pero si podemos decir que son coherentes con los resultados de la tarea 1 y que se parecen a la realidad (argumentos en los gráficos anteriores)

Pregunta 3

Revise si ocurre desorden de paquetes: ¿recibe siempre los ACKs en orden? Si no es así, ¿el protocolo debiera funcionar igual? ¿En algún caso extremo de desorden podría fallar?

Revisando los prints podemos ver que en algunos casos los ACKs no llegan en orden, pero esto no afecta al protocolo ya que este solo se preocupa de que llegue un ACK para poder enviar el siguiente paquete. Por lo que el protocolo debiera funcionar igual. Para poder ver este funcionamiento, se puede llamar a la función compute_bandwidth con el parámetro verbose=True.

El desorden extremo podría hacer fallar al protocolo si $MAX_SEQ < window_size$, ya que el receiver podría confundir paquetes del pasado con paquetes del futuro. Pero esto no ocurre en nuestro caso ya aseguramos que un cliente así no se pueda construir. Sin embargo casos extremos pueden existir de todas formas, por lo que el protocolo no es 100% seguro. Por ejemplo podría ocurrir que un paquete con número de secuencia X se quede dando vueltas en la red por mucho tiempo, entonces el sender haría timeout ya que lo daría por perdido, luego lo reenviaría y todo seguiría normal, pero si tenemos suficiente mala suerte puede ocurrir que el paquete que haya quedado dando vueltas llegue al receiver justo cuando este esperaba un paquete con número de secuencia X, por lo que el receiver lo tomaría como válido y lo aceptaría, lo que duplicaría el paquete

Pregunta 4

Les pedimos implementar Go-Back-N con ACKs acumulativos. Si uno no quisiera eso, y quisiera esperar los ACKs uno por uno, ¿funcionaría el protocolo con este servidor? ¿Se puede hacer que funcione?

A priori el protocolo no funcionaría, ya que el sender no podría saber si el receiver recibió todos los paquetes o no, la gracia de los ACKs acumulativos es que el sender sabe que si llegó un ACK con número de secuencia X, entonces el receiver recibió todos los paquetes con número de secuencia menor a X. Por lo que el sender puede enviar el siguiente paquete sin problemas. Si no fueran acumulativos sería fácil para el receiver confudirse. Sin embargo para arreglarlo tendríamos que mandar un ACK por cada paquete recibido en específico y prácticamente estaríamos implementando un Selective Repeat (obviamente habría que implementar una ventana de recepción para

que el receiver no se confunda con los paquetes que llegan). Por lo que si se puede hacer que funcione, pero no sería un Go-Back-N.