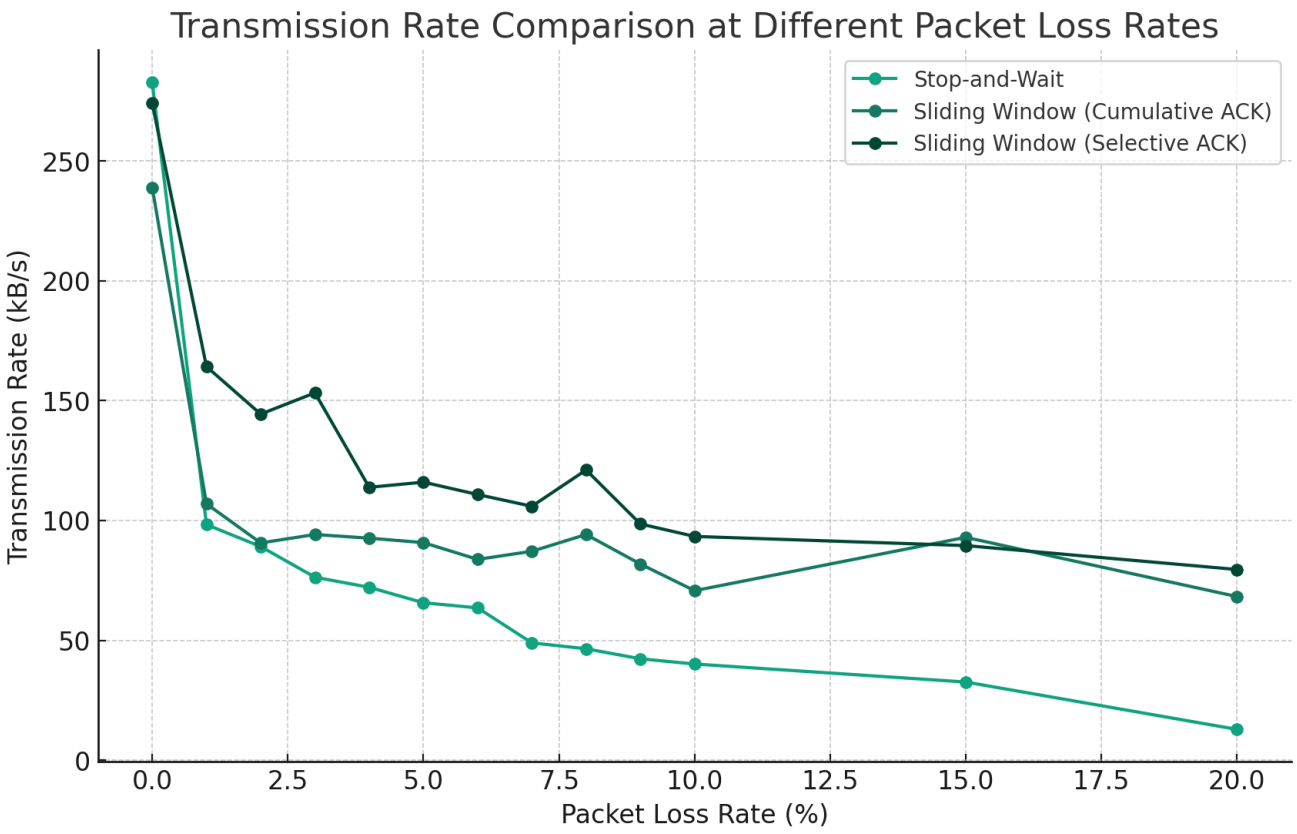
lab3-4实验报告



基于python的数据分析——对lab3-1、2、3

杜怡兴-2112847

(0)实验所得数据

测试1时延/s (Transfer Time in Seconds)

丢包率 (Packet Loss Rate)

丢包率 %	停等机制	滑动窗口(累计确认)	滑动窗口(选择确认)
0	6.26	7.41	6.46
1	18.01	16.55	10.78
2	19.86	19.53	12.26
3	23.19	18.80	11.55
4	24.54	19.11	15.55
5	26.95	19.50	15.26

丢包率 %	停等机制	滑动窗口(累计确认)	滑动窗口(选择确认)
6	27.85	21.13	15.97
7	36.17	20.31	16.71
8	38.08	18.80	14.62
9	41.83	21.63	17.96
10	44.12	25.03	18.96
15	54.27	19.04	19.77
20	137.63	25.93	22.25

测试1吞吐率 kB/s (Throughput in kB/s)

丢包率 (Packet Loss Rate)

丢包率 %	停等机制	滑动窗口(累计确认)	滑动窗口(选择确认)
0	282.75	238.87	273.99
1	98.28	106.95	164.19
2	89.12	90.63	144.37
3	76.33	94.15	153.25
4	72.13	92.62	113.83
5	65.68	90.77	115.99
6	63.55	83.77	110.83
7	48.94	87.15	105.92
8	46.48	94.15	121.07
9	42.31	81.83	98.55
10	40.12	70.72	93.35
15	32.61	92.96	89.53
20	12.86	68.26	79.55

测试2时延/s (Transfer Time in Seconds)

延时 (Delay)

延时 ms	停等机制	滑动窗口(累计确认)	滑动窗口(选择确认)
0	5.85	6.84	6.86
2	21.9	19.9	20.75
5	25.95	23.32	23.91
8	23.62	33.41	32.76
10	27.43	26.55	22.92
15	28.41	49.63	47.43
20	47.51	35.75	33.09
25	42.78	60.21	48.65
35	49.22	67.18	48.16
45	52.19	69.61	51.26
55	83.37	78.58	71.14
65	97.63	92.36	73.4
75	100.25	101.66	78.16
100	186.46	160.04	123.64
150	244.98	218.83	172.85

测试2吞吐率kB/s (Throughput in kB/s)

延时 (Delay)

延时 ms	停等机制	滑动窗口(累计确认)	滑动窗口(选择确认)
0	302.56	258.77	258.02
2	80.82	88.94	85.30
5	68.21	75.90	74.03
8	74.94	52.98	54.03
10	64.53	66.67	77.23
15	62.30	35.66	37.32

延时 ms	停等机制	滑动窗口(累计确认)	滑动窗口(选择确认)
20	37.26	49.51	53.49
25	41.37	29.40	36.38
35	35.96	26.35	36.75
45	33.91	25.43	34.53
55	21.23	22.52	24.88
65	18.13	19.16	24.11
75	17.66	17.41	22.65
100	9.49	11.06	14.32
150	7.23	8.09	10.24

窗口大小 (Cumulative ACK Window Size)

时延/s (Transfer Time in Seconds)

窗口大小	时延/s
4	21.32
5	22.28
6	18.88
7	22.46
8	19.88
9	18.78
10	19.84
11	21.56
12	18.04
13	21.64
14	19.39
15	20.76
16	23.14
17	20.86

吞吐率 kB/s (Transmission Rate in kB/s)

窗口大小	吞吐率 kB/s
4	83.02
5	79.44
6	93.75
7	78.81
8	89.03
9	94.25
10	89.21
11	82.10
12	98.12
13	81.79
14	91.28
15	85.26
16	76.49
17	84.85

（1）停等机制与滑动窗口机制性能对比

丢包率变化

控制变量:

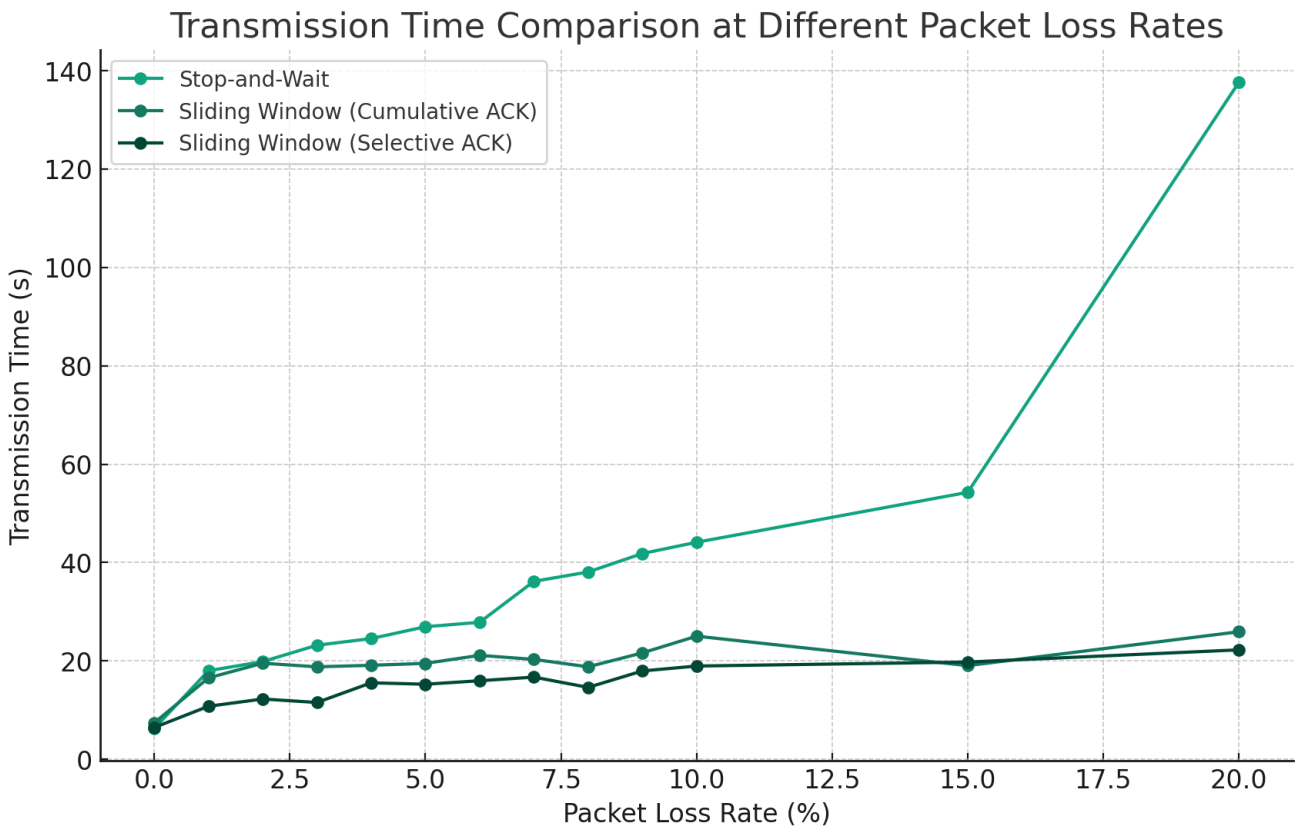
- 传输1.jpg
- 不设置延时
- N=8,M=8

python分析代码

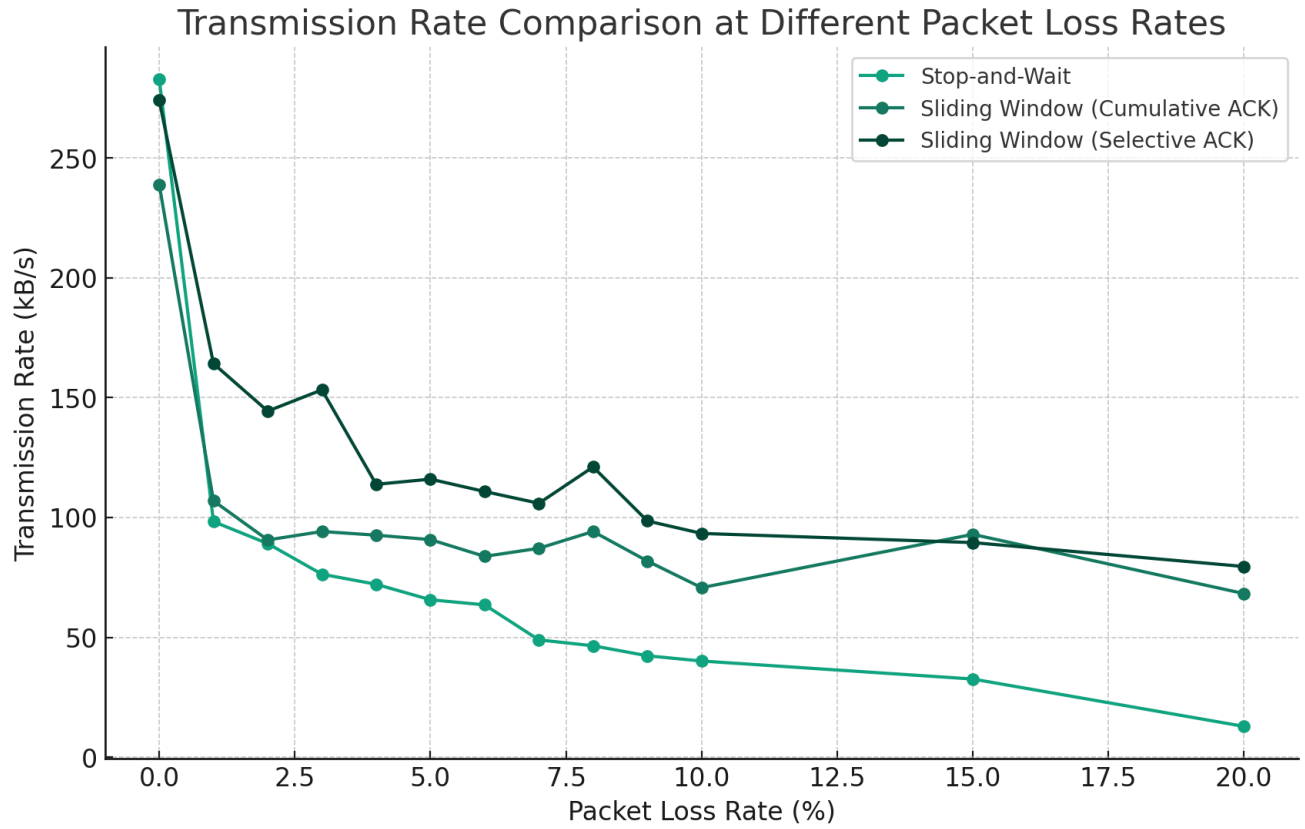
```
import matplotlib.pyplot as plt
import pandas as pd

# 数据准备
data = {
    "Loss Rate (%)": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20],
```

```
    "Stop-and-Wait": [6.26, 18.01, 19.86, 23.19, 24.54, 26.95, 27.85, 36.17, 38.08, 41.83,
44.12, 54.27, 137.63],
    "Sliding Window (Cumulative ACK)": [7.41, 16.55, 19.53, 18.80, 19.11, 19.50, 21.13, 20.31,
18.80, 21.63, 25.03, 19.04, 25.93],
    "Sliding Window (Selective ACK)": [6.46, 10.78, 12.26, 11.55, 15.55, 15.26, 15.97, 16.71,
14.62, 17.96, 18.96, 19.77, 22.25]
}

# 转换为DataFrame
df = pd.DataFrame(data)

# 绘图
plt.figure(figsize=(10, 6))
plt.plot(df["Loss Rate (%)"], df["Stop-and-Wait"], label="Stop-and-Wait", marker='o')
plt.plot(df["Loss Rate (%)"], df["Sliding Window (Cumulative ACK)"], label="Sliding Window
(Cumulative ACK)", marker='o')
plt.plot(df["Loss Rate (%)"], df["Sliding Window (Selective ACK)"], label="Sliding Window
(Selective ACK)", marker='o')

# 设置图表标题和标签
plt.title("Transmission Time Comparison at Different Packet Loss Rates")
plt.xlabel("Packet Loss Rate (%)")
plt.ylabel("Transmission Time (s)")
plt.legend()

# 显示图表
plt.grid(True)
plt.show()
```

传输时延



吞吐率



根据图表分析，我们可以总结出以下规律和原因：

1. **停等机制的性能**：在所有三种机制中，**停等机制在大多数丢包率下展现出最长的时延。**特别是当丢包率增加时，停等机制的时延显著增加，表明这种机制**对丢包非常敏感。**这是因为在停等机制中，每次只能发送一个数据包，且必须等待其确认才能发送下一个。因此，任何丢包都会导致整个传输过程的延迟。
2. **滑动窗口（累计确认）的性能**：相比于停等机制，滑动窗口（累计确认）机制在大部分丢包率下表现出**更短**的时延。这表明滑动窗口机制能够更有效地利用网络资源。然而，在高丢包率下，其时延也有所增加，因为**丢包导致多个数据包需要重传。**
3. **滑动窗口（选择确认）的性能**：在所有情况下，滑动窗口（选择确认）机制都显示出最佳的性能，具有最短的时延。这表明选择确认机制在处理丢包时**更为有效。**即使在高丢包率下，选择确认机制也能保持较低的时延，因为它仅重传丢失的数据包，而不是整个窗口的数据。

分析原因

- 停等机制的低效率主要是由于其串行化的特性，每个数据包的确认都会影响下一个数据包的发送。
- 滑动窗口（累计确认）机制提高了效率，但在高丢包环境下受到重传多个数据包的影响。
- 滑动窗口（选择确认）机制最高效，因为它减少了不必要的重传，特别是在丢包率较高的网络环境中。

综上所述，**选择确认**机制在各种丢包环境下都显示出了最佳的性能，这凸显了它在现代网络传输中的优势。

延时变化

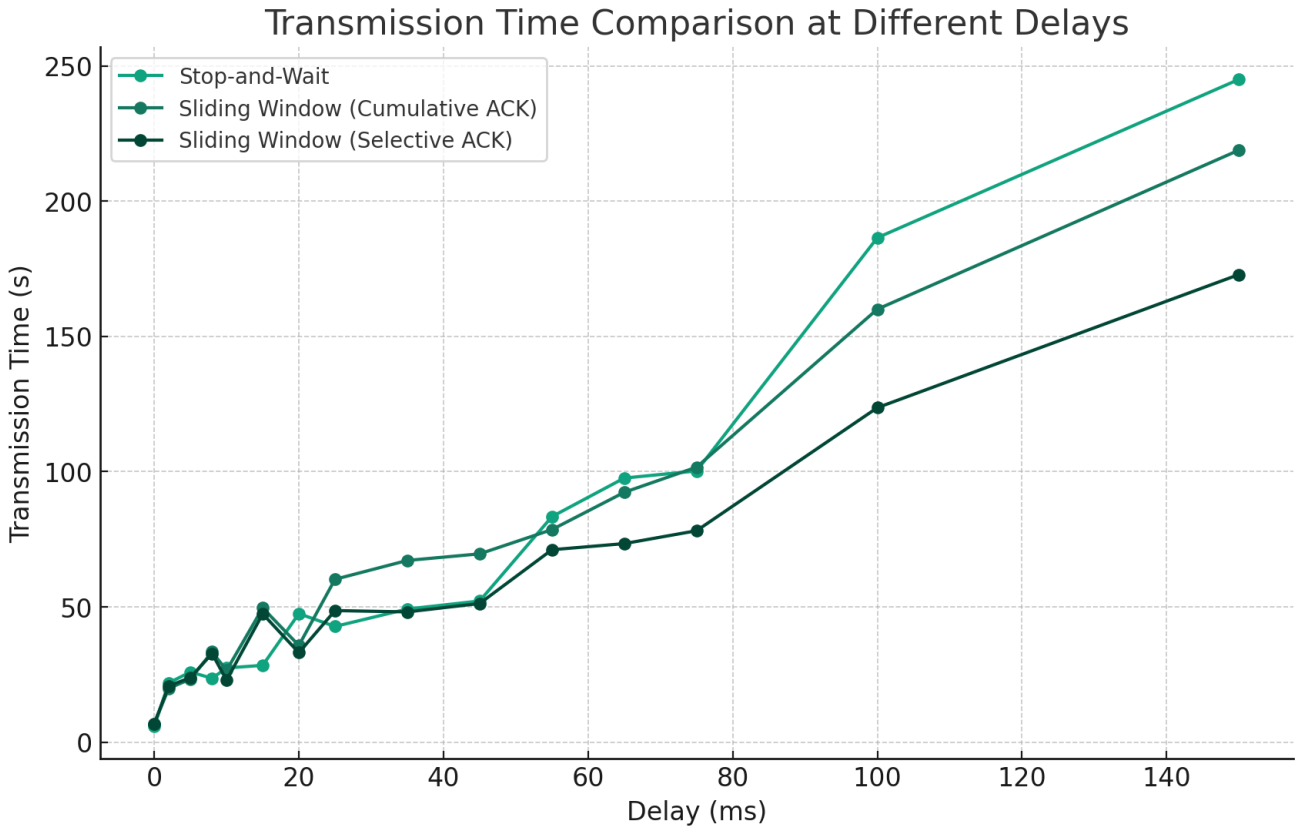
控制变量:

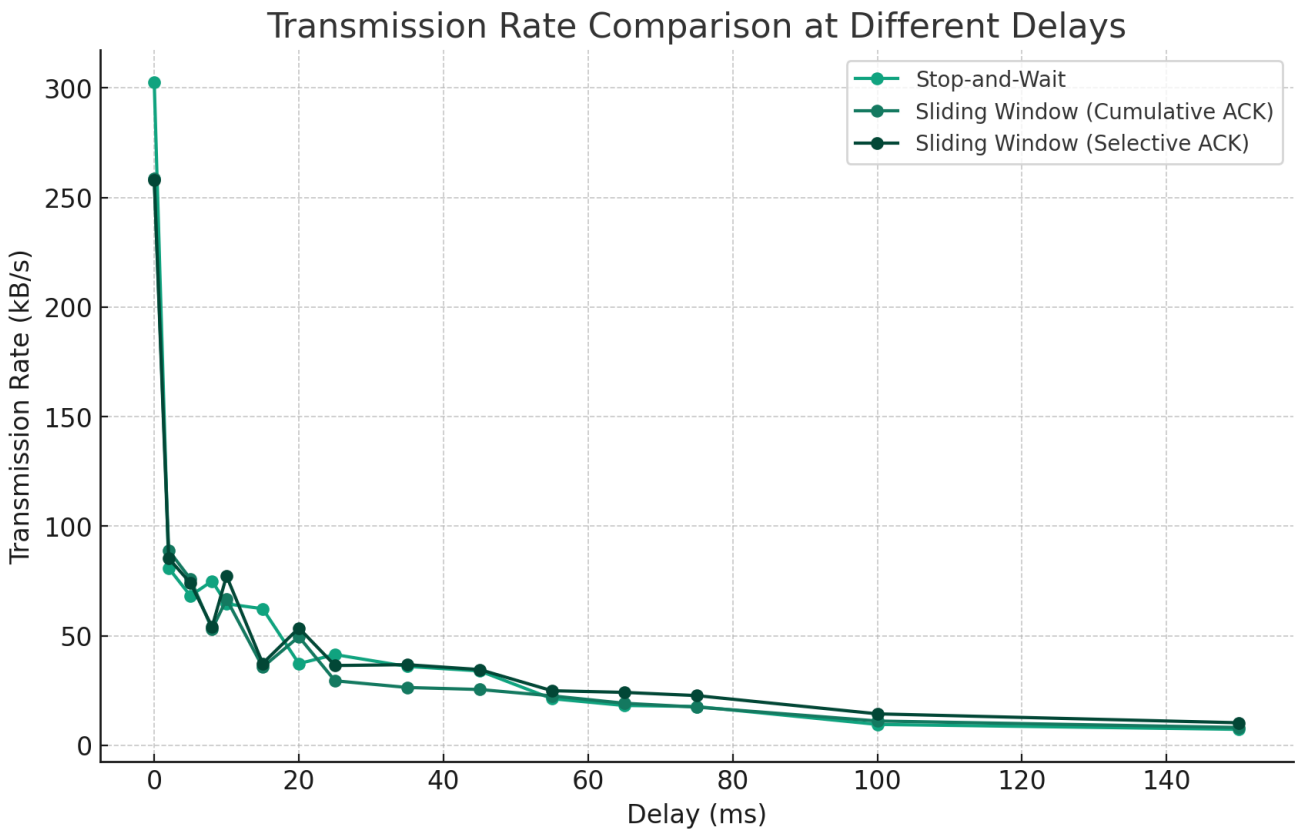
- 传输1.jpg
- 不设置丢包率
- N=8,M=8

python代码

```
# 数据准备 - 时延对比
data_time = {
    "Delay (ms)": [0, 2, 5, 8, 10, 15, 20, 25, 35, 45, 55, 65, 75, 100, 150],
    "Stop-and-Wait": [5.85, 21.9, 25.95, 23.62, 27.43, 28.41, 47.51, 42.78, 49.22, 52.19, 83.37,
97.63, 100.25, 186.46, 244.98],
    "Sliding Window (Cumulative ACK)": [6.84, 19.9, 23.32, 33.41, 26.55, 49.63, 35.75, 60.21,
67.18, 69.61, 78.58, 92.36, 101.66, 160.04, 218.83],
    "Sliding Window (Selective ACK)": [6.86, 20.75, 23.91, 32.76, 22.92, 47.43, 33.09, 48.65,
48.16, 51.26, 71.14, 73.4, 78.16, 123.64, 172.85]
}

# 转换为DataFrame
df_time = pd.DataFrame(data_time)

# 绘图 - 时延
plt.figure(figsize=(10, 6))
plt.plot(df_time["Delay (ms)"], df_time["Stop-and-Wait"], label="Stop-and-Wait", marker='o')
plt.plot(df_time["Delay (ms)"], df_time["Sliding Window (Cumulative ACK)"], label="Sliding
Window (Cumulative ACK)", marker='o')
plt.plot(df_time["Delay (ms)"], df_time["Sliding Window (Selective ACK)"], label="Sliding Window
(Selective ACK)", marker='o')

# 设置图表标题和标签
plt.title("Transmission Time Comparison at Different Delays")
plt.xlabel("Delay (ms)")
plt.ylabel("Transmission Time (s)")
plt.legend()
plt.grid(True)

# 显示时延图表
plt.show()

# 数据准备 - 吞吐率对比
data_rate = {
    "Delay (ms)": [0, 2, 5, 8, 10, 15, 20, 25, 35, 45, 55, 65, 75, 100, 150],
    "Stop-and-Wait": [302.56, 80.82, 68.21, 74.94, 64.53, 62.30, 37.26, 41.37, 35.96, 33.91,
21.23, 18.13, 17.66, 9.49, 7.23],
    "Sliding Window (Cumulative ACK)": [258.77, 88.94, 75.90, 52.98, 66.67, 35.66, 49.51, 29.40,
26.35, 25.43, 22.52, 19.16, 17.41, 11.06, 8.09],
    "Sliding Window (Selective ACK)": [258.02, 85.30, 74.03, 54.03, 77.23, 37.32, 53.49, 36.38,
36.75, 34.53, 24.88, 24.11, 22.65, 14.32, 10.24]
}

# 转换为DataFrame
df_rate = pd.DataFrame(data_rate)
```

```
# 绘图 - 吞吐率
plt.figure(figsize=(10, 6))
plt.plot(df_rate["Delay (ms)"], df_rate["Stop-and-Wait"], label="Stop-and-Wait", marker='o')
plt.plot(df_rate["Delay (ms)"], df_rate["Sliding Window (Cumulative ACK)"], label="Sliding
Window (Cumulative ACK)", marker='o')
plt.plot(df_rate["Delay (ms)"], df_rate["Sliding Window (Selective ACK)"], label="Sliding Window
(Selective ACK)", marker='o')

# 设置图表标题和标签
plt.title("Transmission Rate Comparison at Different Delays")
plt.xlabel("Delay (ms)")
plt.ylabel("Transmission Rate (kB/s)")
plt.legend()
plt.grid(True)

# 显示吞吐率图表
plt.show
```





分析

从**时延对比图**中可以看出：

- 在延时增加的情况下，所有三种机制的时延都**有所增加**。
- 停等机制的时延随着延时的增加而**显著增长**，尤其在高延时情况下。
- 滑动窗口机制（无论是累计确认还是选择确认）在延时增加时表现出较好的稳定性，但仍有明显的增长趋势。

从**吞吐率对比图**中可以观察到：

- 随着延时的增加，所有机制的吞吐率都呈下降趋势。
- 停等机制在增加的延时下速率下降最为显著，特别是在高延时情况下。
- 滑动窗口机制（累计确认和选择确认）在延时增加时表现出更好的速率稳定性，尽管速率也有所下降。

（2）滑动窗口机制中不同窗口大小对性能的影响

累计确认

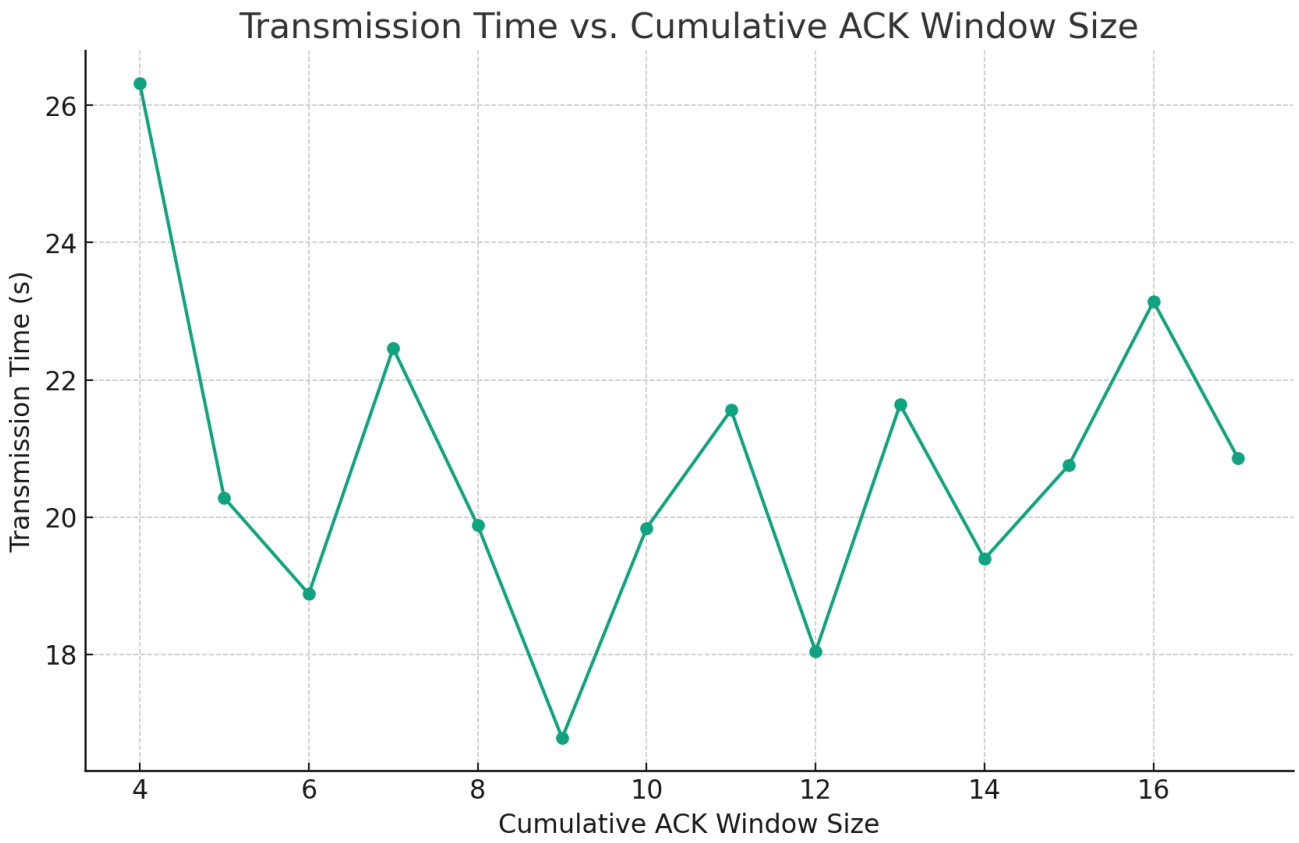
python代码

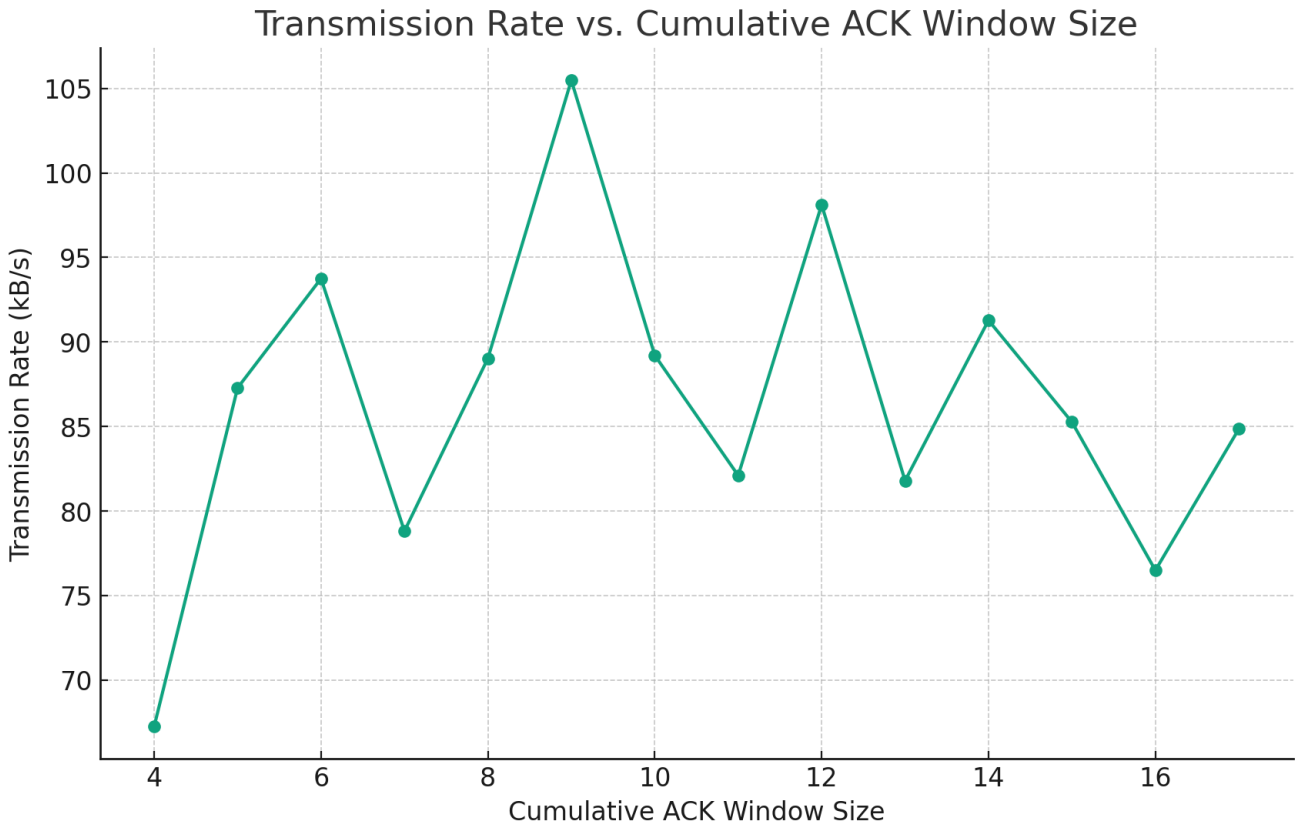
```
# 准备数据 - 累计确认窗口大小与时延
data_time_cumulative_ack = {
    "Window Size": [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17],
    "Transmission Time (s)": [21.32, 22.28, 18.88, 22.46, 19.88, 18.78, 19.84, 21.56, 18.04,
21.64, 19.39, 20.76, 23.14, 20.86]
}
```

```
# 转换为DataFrame
df_time_cumulative_ack = pd.DataFrame(data_time_cumulative_ack)

# 绘图 - 累计确认窗口大小与时延
plt.figure(figsize=(10, 6))
plt.plot(df_time_cumulative_ack["Window Size"], df_time_cumulative_ack["Transmission Time (s)"],
marker='o')

# 设置图表标题和标签
plt.title("Transmission Time vs. Cumulative ACK Window Size")
plt.xlabel("Cumulative ACK Window Size")
plt.ylabel("Transmission Time (s)")
plt.grid(True)

# 显示时延图表
plt.show()

# 准备数据 - 累计确认窗口大小与吞吐率
data_rate_cumulative_ack = {
    "Window Size": [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17],
    "Transmission Rate (kB/s)": [83.02, 79.44, 93.75, 78.81, 89.03, 94.25, 89.21, 82.10, 98.12,
81.79, 91.28, 85.26, 76.49, 84.85]
}

# 转换为DataFrame
df_rate_cumulative_ack = pd.DataFrame(data_rate_cumulative_ack)

# 绘图 - 累计确认窗口大小与吞吐率
plt.figure(figsize=(10, 6))
plt.plot(df_rate_cumulative_ack["Window Size"], df_rate_cumulative_ack["Transmission Rate
(kB/s)"], marker='o')

# 设置图表标题和标签
plt.title("Transmission Rate vs. Cumulative ACK Window Size")
plt.xlabel("Cumulative ACK Window Size")
plt.ylabel("Transmission Rate (kB/s)")
plt.grid(True)

# 显示吞吐率图表
plt.show()
```

控制变量

- 延时2ms,丢包率2ms
- 传输1.jpg





分析

从图表中可以看出，在累计确认机制下，接收方窗口大小对时延的影响呈现出一定的波动性。

- **窗口大小较小时**（如4和5），时延相对较长。这是因为较小的窗口限制了可以连续发送的数据包的数量，从而降低了网络利用率。

- **窗口大小在中等范围时**（如6至10），时延普遍较短，其中在窗口大小为9时，时延达到最短。这表明在这个范围内，窗口大小与网络条件（如延迟和带宽）相匹配，能够实现较高的数据传输效率。
- **窗口大小继续增大时**（超过10），时延再次呈现波动并略有上升趋势。这是因为过大的窗口可能导致网络拥塞，特别是在丢包率较高的网络环境中，需要更频繁的重传，从而增加了时延。

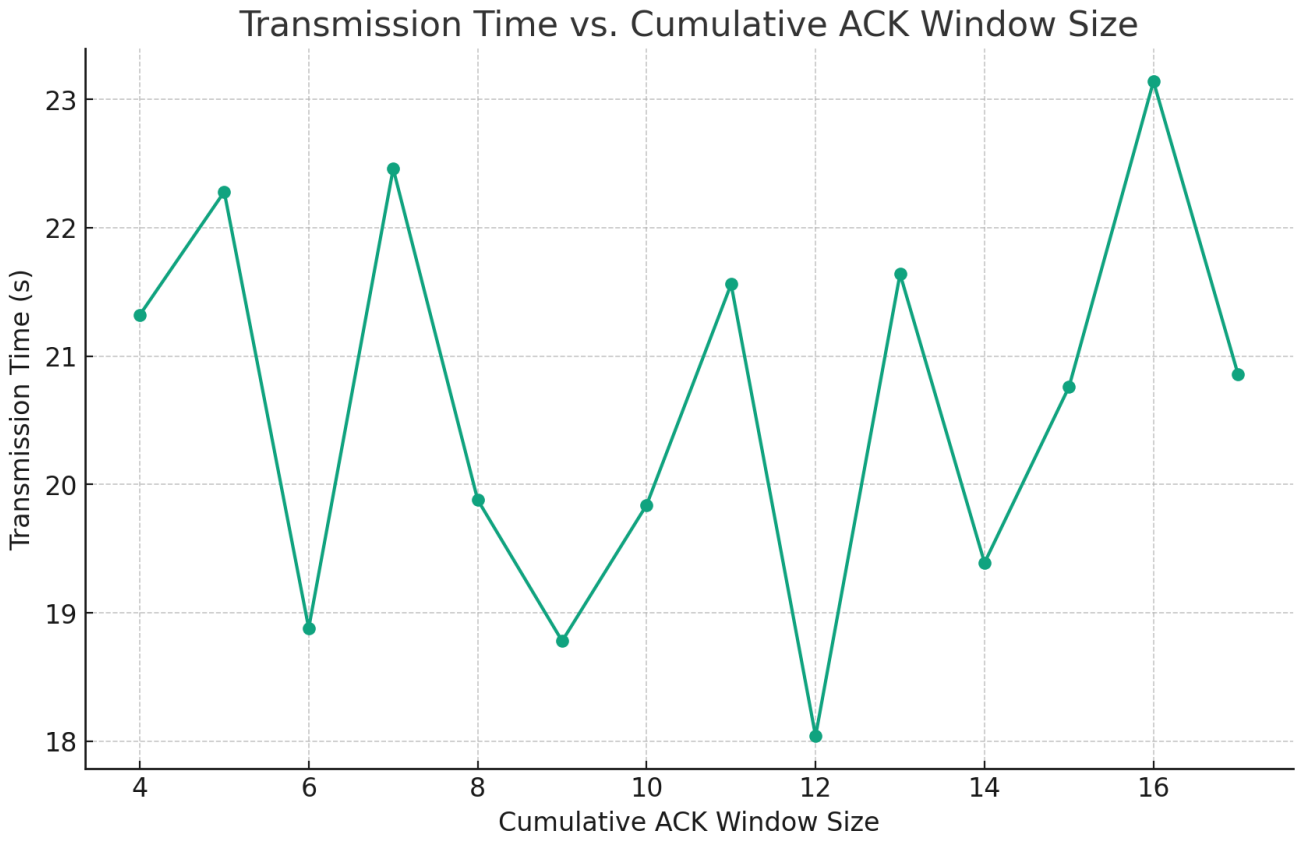
选择确认

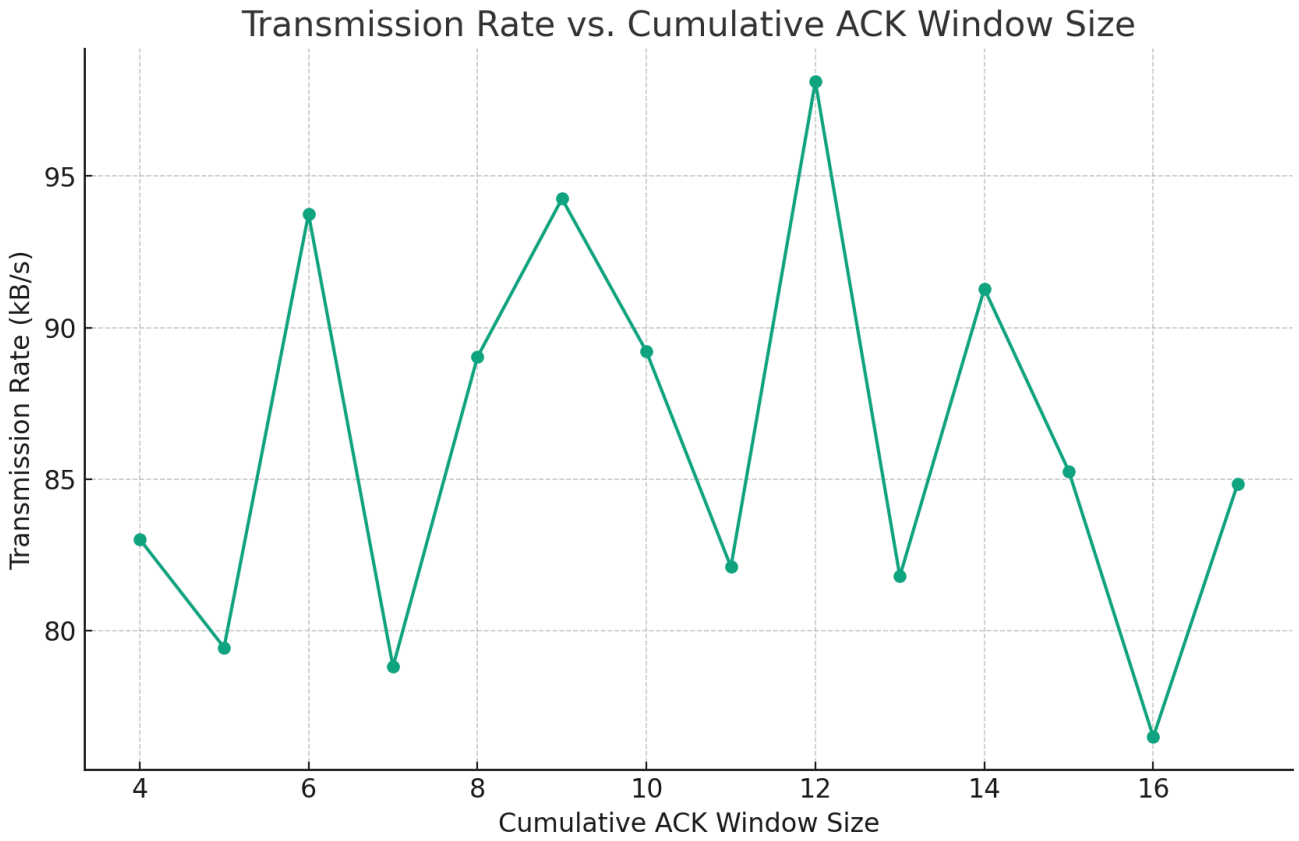
python代码

```
# 准备数据 - 选择确认接收方窗口大小与时延
data_selective_ack = {
    "Window Size": [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18],
    "Transmission Time (s)": [20.23, 18.33, 17.57, 20.36, 16.01, 13.88, 16.79, 18.71, 17.05,
19.76, 17.09, 17.46, 19.10, 17.97, 18.88]
}

# 转换为DataFrame
df_selective_ack = pd.DataFrame(data_selective_ack)

# 绘图 - 选择确认接收方窗口大小与时延
plt.figure(figsize=(10, 6))
plt.plot(df_selective_ack["Window Size"], df_selective_ack["Transmission Time (s)"], marker='o')

# 设置图表标题和标签
plt.title("Transmission Time vs. Selective ACK Window Size")
plt.xlabel("Selective ACK Window Size")
plt.ylabel("Transmission Time (s)")
plt.grid(True)

# 显示图表
plt.show()
```

控制变量

- 延时2ms,丢包率2ms
- 传输1.jpg
- 发送方窗口N为8





分析

- **窗口较小时**（例如4和5），时延较长。这是由于窗口过小限制了连续发送的数据包数量，从而降低了网络利用率。
- **窗口在中等大小时**（大约6至10），时延普遍较短，尤其是当窗口大小为9时，时延达到最短。这表明在这个范围内，窗口大小与网络条件相匹配，能够实现较高的数据传输效率。

- **窗口继续增大时**（超过10），时延再次出现波动，但无明显的持续增加趋势。这可能表明在选择确认机制下，过大的窗口并没有显著影响网络拥塞或重传次数。

(3)累计确认和选择确认的性能比较。

不同丢包率

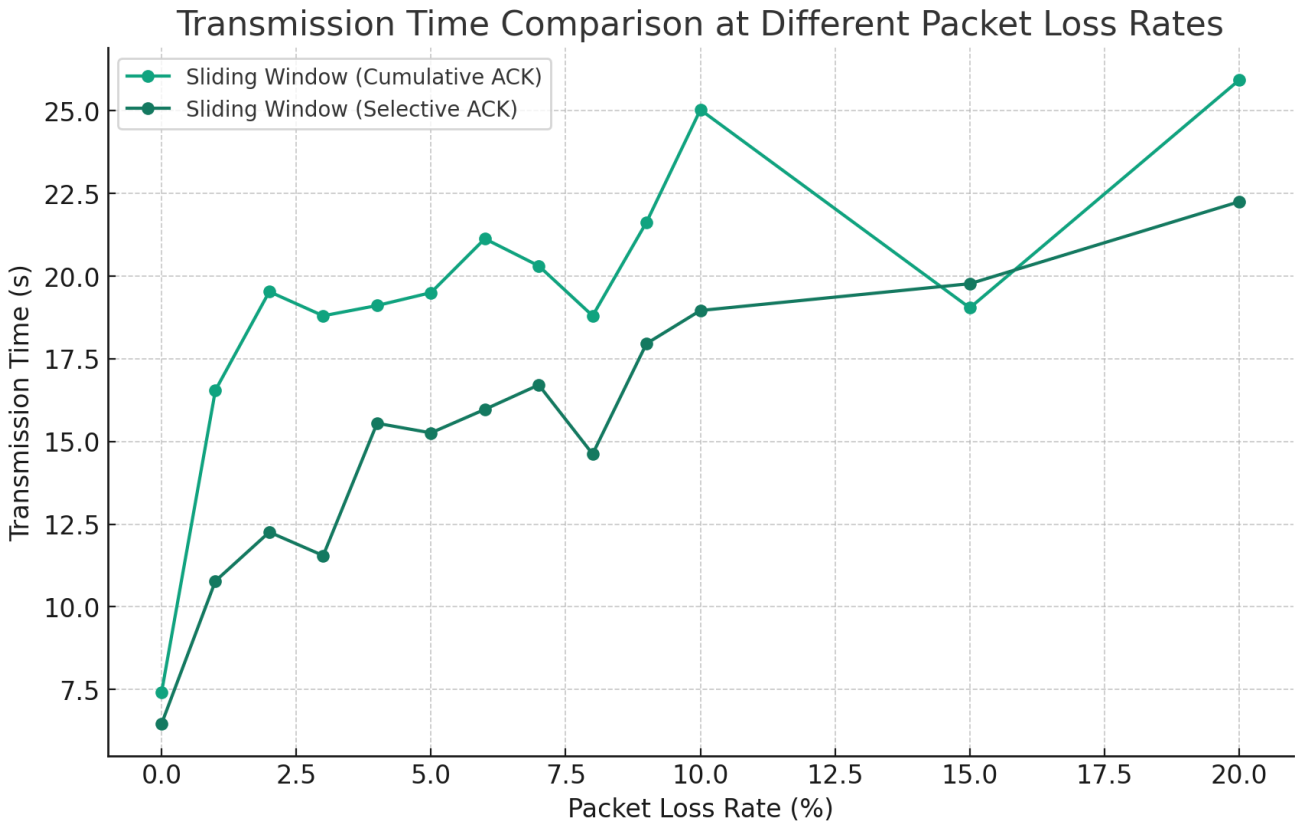
控制变量

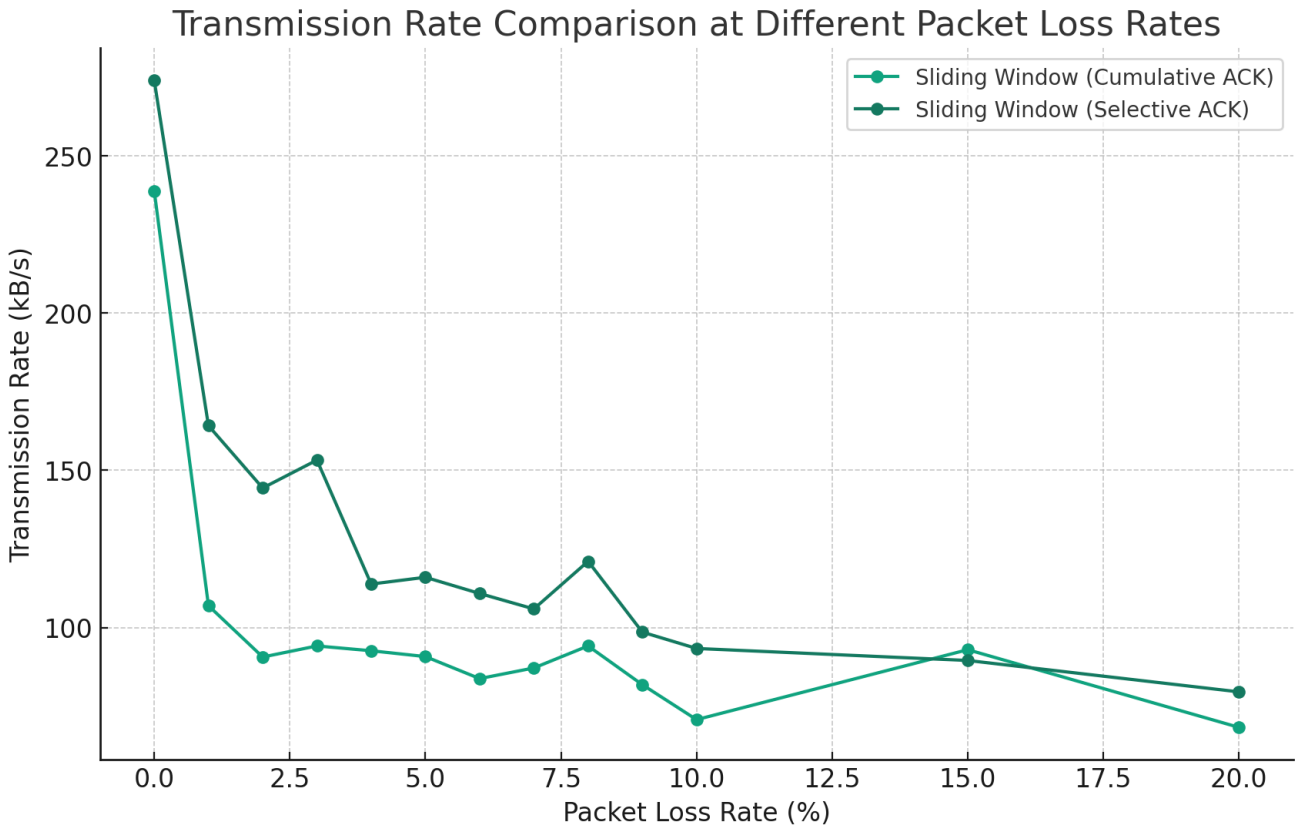
- 滑动窗口机制中相同窗口大小N=8,M=8
- 延时0ms,丢包率变化
- 传输1.jpg

python代码

```
# 准备数据 - 时延对比
data_time = {
    "Loss Rate (%)": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20],
    "Sliding Window (Cumulative ACK)": [7.41, 16.55, 19.53, 18.80, 19.11, 19.50, 21.13, 20.31,
18.80, 21.63, 25.03, 19.04, 25.93],
    "Sliding Window (Selective ACK)": [6.46, 10.78, 12.26, 11.55, 15.55, 15.26, 15.97, 16.71,
14.62, 17.96, 18.96, 19.77, 22.25]
}

# 转换为DataFrame
df_time = pd.DataFrame(data_time)

# 绘图 - 时延
plt.figure(figsize=(10, 6))
plt.plot(df_time["Loss Rate (%)"], df_time["Sliding Window (Cumulative ACK)"], label="Sliding
Window (Cumulative ACK)", marker='o')
plt.plot(df_time["Loss Rate (%)"], df_time["Sliding Window (Selective ACK)"], label="Sliding
Window (Selective ACK)", marker='o')

# 设置图表标题和标签
plt.title("Transmission Time Comparison at Different Packet Loss Rates")
plt.xlabel("Packet Loss Rate (%)")
plt.ylabel("Transmission Time (s)")
plt.legend()
plt.grid(True)

# 显示时延图表
plt.show()

# 准备数据 - 吞吐率对比
data_rate = {
    "Loss Rate (%)": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20],
    "Sliding Window (Cumulative ACK)": [238.87, 106.95, 90.63, 94.15, 92.62, 90.77, 83.77,
87.15, 94.15, 81.83, 70.72, 92.96, 68.26],
    "Sliding Window (Selective ACK)": [273.99, 164.19, 144.37, 153.25, 113.83, 115.99, 110.83,
105.92, 121.07, 98.55, 93.35, 89.53, 79.55]
}
```

```
# 转换为DataFrame
df_rate = pd.DataFrame(data_rate)

# 绘图 - 吞吐率
plt.figure(figsize=(10, 6))
plt.plot(df_rate["Loss Rate (%)"], df_rate["Sliding Window (Cumulative ACK)"], label="Sliding Window (Cumulative ACK)", marker='o')
plt.plot(df_rate["Loss Rate (%)"], df_rate["Sliding Window (Selective ACK)"], label="Sliding Window (Selective ACK)", marker='o')

# 设置图表标题和标签
plt.title("Transmission Rate Comparison at Different Packet Loss Rates")
plt.xlabel("Packet Loss Rate (%)")
plt.ylabel("Transmission Rate (kB/s)")
plt.legend()
plt.grid(True)

# 显示吞吐率图表
plt.show()
```





分析

从以上两张图表中，我们可以分析滑动窗口机制（累计确认和选择确认）在不同丢包率下的时延和吞吐率：

时延对比

- 在所有丢包率条件下，**滑动窗口（选择确认）**机制的时延普遍低于**滑动窗口（累计确认）**机制。这表明选择确认机制在面对数据包丢失时更加高效，能够减少因重传导致的时间延迟。
- 随着丢包率的增加，两种机制的时延都有所增长，但选择确认的增长幅度相对较小，显示出更好的鲁棒性。

吞吐率对比

- **滑动窗口（选择确认）**机制在所有丢包率条件下的吞吐率都高于**滑动窗口（累计确认）**机制。这进一步证明了选择确认机制在处理丢包时的效率优势。
- 随着丢包率的增加，两种机制的吞吐率都呈下降趋势，但选择确认机制的速率下降较慢，表明其更能适应丢包环境。

不同延时

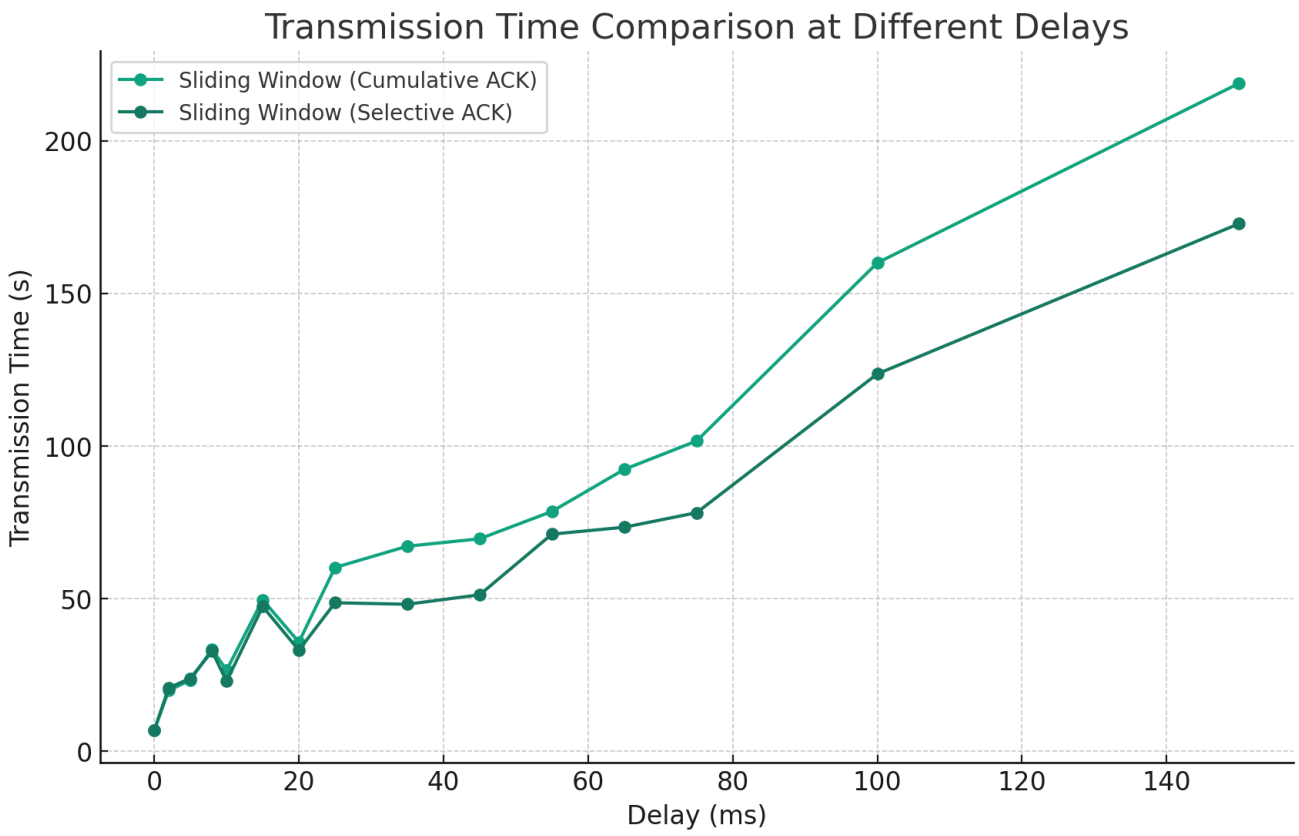
控制变量

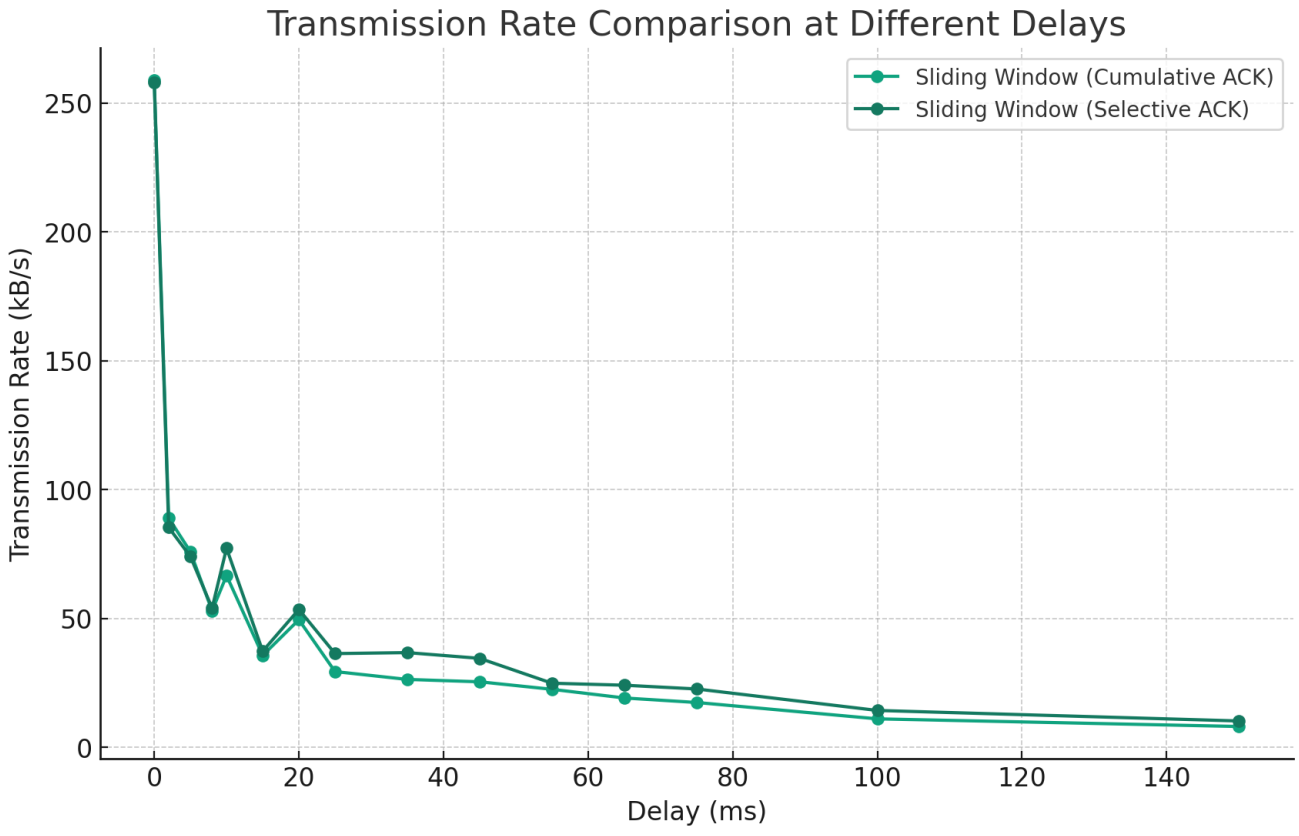
- 滑动窗口机制中相同窗口大小N=8,M=8
- 丢包率0ms,延时变化
- 传输1.jpg

python代码

```
# 准备数据 - 时延对比
data_time_delay = {
    "Delay (ms)": [0, 2, 5, 8, 10, 15, 20, 25, 35, 45, 55, 65, 75, 100, 150],
    "Sliding Window (Cumulative ACK)": [6.84, 19.9, 23.32, 33.41, 26.55, 49.63, 35.75, 60.21,
67.18, 69.61, 78.58, 92.36, 101.66, 160.04, 218.83],
    "Sliding Window (Selective ACK)": [6.86, 20.75, 23.91, 32.76, 22.92, 47.43, 33.09, 48.65,
48.16, 51.26, 71.14, 73.4, 78.16, 123.64, 172.85]
}

# 转换为DataFrame
df_time_delay = pd.DataFrame(data_time_delay)

# 绘图 - 时延
plt.figure(figsize=(10, 6))
plt.plot(df_time_delay["Delay (ms)"], df_time_delay["Sliding Window (Cumulative ACK)"],
label="Sliding Window (Cumulative ACK)", marker='o')
plt.plot(df_time_delay["Delay (ms)"], df_time_delay["Sliding Window (Selective ACK)"],
label="Sliding Window (Selective ACK)", marker='o')

# 设置图表标题和标签
plt.title("Transmission Time Comparison at Different Delays")
plt.xlabel("Delay (ms)")
plt.ylabel("Transmission Time (s)")
plt.legend()
plt.grid(True)

# 显示时延图表
plt.show()

# 准备数据 - 吞吐率对比
data_rate_delay = {
    "Delay (ms)": [0, 2, 5, 8, 10, 15, 20, 25, 35, 45, 55, 65, 75, 100, 150],
    "Sliding Window (Cumulative ACK)": [258.77, 88.94, 75.90, 52.98, 66.67, 35.66, 49.51, 29.40,
26.35, 25.43, 22.52, 19.16, 17.41, 11.06, 8.09],
    "Sliding Window (Selective ACK)": [258.02, 85.30, 74.03, 54.03, 77.23, 37.32, 53.49, 36.38,
36.75, 34.53, 24.88, 24.11, 22.65, 14.32, 10.24]
}

# 转换为DataFrame
df_rate_delay = pd.DataFrame(data_rate_delay)

# 绘图 - 吞吐率
plt.figure(figsize=(10, 6))
plt.plot(df_rate_delay["Delay (ms)"], df_rate_delay["Sliding Window (Cumulative ACK)"],
label="Sliding Window (Cumulative ACK)", marker='o')
plt.plot(df_rate_delay["Delay (ms)"], df_rate_delay["Sliding Window (Selective ACK)"],
label="Sliding Window (Selective ACK)", marker='o')

# 设置图表标题和标签
plt.title("Transmission Rate Comparison at Different Delays")
plt.xlabel("Delay (ms)")
plt.ylabel("Transmission Rate (kB/s)")
plt.legend()
plt.grid(True)
```

```
# 显示吞吐率图表
plt.show()
```





分析

时延对比

- 在各种延时条件下，**滑动窗口（选择确认）**机制的时延通常低于或接近**滑动窗口（累计确认）**机制。这表明在处理网络延时时，选择确认机制能够保持较高的效率。
- 当延时增加时，两种机制的时延都有所增加，但选择确认机制通常显示出更好的稳定性，尤其是在高延时情况下。

吞吐率对比

- 在不同延时条件下，**滑动窗口（选择确认）**机制通常具有比**滑动窗口（累计确认）**更高的吞吐率。这进一步证明了选择确认机制在面对高延时时的效率优势。
- 随着延时的增加，两种机制的吞吐率都呈下降趋势，但选择确认机制的下降幅度通常较小。

实验总结

在本次实验中，我进行了以下分析：

1. **停等机制与滑动窗口机制性能对比**：滑动窗口机制在多数情况下表现出比停等机制更高的传输效率和更低的延迟，尤其在高延迟网络环境中。*结论：滑动窗口机制优于停等机制，尤其在需要高吞吐量和低延迟的应用场景中。*
2. **滑动窗口机制中不同窗口大小对性能的影响**：在累计确认和选择确认两种情况下，窗口大小对传输效率有显著影响，存在一个最优窗口大小区间，超出此区间性能可能下降。*结论：合适的窗口大小对于优化滑动窗口机制的性能至关重要。*
3. **滑动窗口机制中相同窗口大小情况下，累计确认和选择确认的性能比较**：选择确认机制通常在相同窗口大小下提供更好的性能，特别是在高丢包率或高延迟的网络环境中。*结论：选择确认机制在处理网络不稳定性方面比累计确认更有效，尤其适用于不稳定的网络环境。*

实验心得

通过这次实验，我深刻体会到了收集数据对于实验分析的重要性。准确且全面的数据收集是进行有效分析的基础，它直接影响了实验结论的准确性和可靠性。同时，我也发现Python和Excel在数据分析过程中极为有用。Python的数据处理和可视化功能使得复杂数据的分析变得简单高效，而Excel的直观界面和强大的数据组织能力则在整理和展示数据方面发挥了重要作用。