Lab 2

Chenhao Zhang zhan4243@purdue.edu
<firstname.lastname.Project2.final.pdf>

- Note: all testing should be done with to_send_large.txt as the file being sent

- For all testing results do 5 runs and report mean and standard deviation. Use the format mean[std]
  - Ie. If your mean is 100 bytes and the standard deviation is 10 bytes, then report that as 100[10]

Stop and go results

	Bandwidth = 200,000 bytes/s	Bandwidth = 20,000 bytes/s	Bandwidth = 2,000 bytes/s
Goodput (bytes/s)	4645.37[20.21]	4566.55[81.76]	1731.164[12.17]
Overhead (% of bytes sent)	4.6[0.3]	4.76[0.9]	4.2[0.5]

Parameters should be:

- One way propagation delay : 100 ms
- Loss is 2% and reordering is 2%

Impact of window size

	Smallest window size = 10240 bytes window = 10	Second smallest window size = 15360 bytes	Medium window size = 20480 bytes	Second largest windows size = 30720 bytes	Largest window size = 40960 bytes
Goodput (bytes/s)	35336.71[2625.75]	47407.96[2527.96]	60977.79[4078.78]	73101.53[5371.53]	81536.81[9541.23]
Overhead (% of bytes sent)	2.63[0.83]	2.50[0.45]	2.13[0.55]	2.89[0.60]	3.39[0.76]

Parameters should be:

- Bandwidth 200,000 bytes/s
- One way propagation delay : 100 ms
- Loss is 2% and reordering is 2%
- This is testing your protocol, NOT stop and go

Note: Replace X in the above window sizes with the window size you used!!

Your protocol vs stop and go

	Your protocol	Stop and go
Goodput (bytes/s)	81536.81[9541.23]	4645.37[20.21]
Overhead (% of bytes sent)	3.39[0.76]	4.6[0.3] (drop on ack cause some issue to make stop and go looks worse)

Parameters should be:

- Bandwidth 200,000 bytes/s
- One way propagation delay : 100 ms
- Loss is 2% and reordering is 2%
- The window size that works best for your application

Sensitivity Testing

	5% loss and 5% reordering	2% loss and 2% reordering	No loss and no reordering
Your protocol goodput (bytes/s)	60283.85[5851.80]	81536.81[9541.2]	152042.21[121.9]
Your protocol overhead (% of bytes sent)	7.89[2.19]	3.39[0.76]	0.49[0]
Stop and go goodput (bytes/s)	4197.114[33.038]	4645.37[20.21]	4951.17[2.742]
Stop and go overhead (% of bytes sent)	10.95[0.4]	4.6[0.3]	0.4903[0]

Parameters should be:

- Bandwidth = 200,000 bytes/s
- Use whichever window size works best for your algorithm

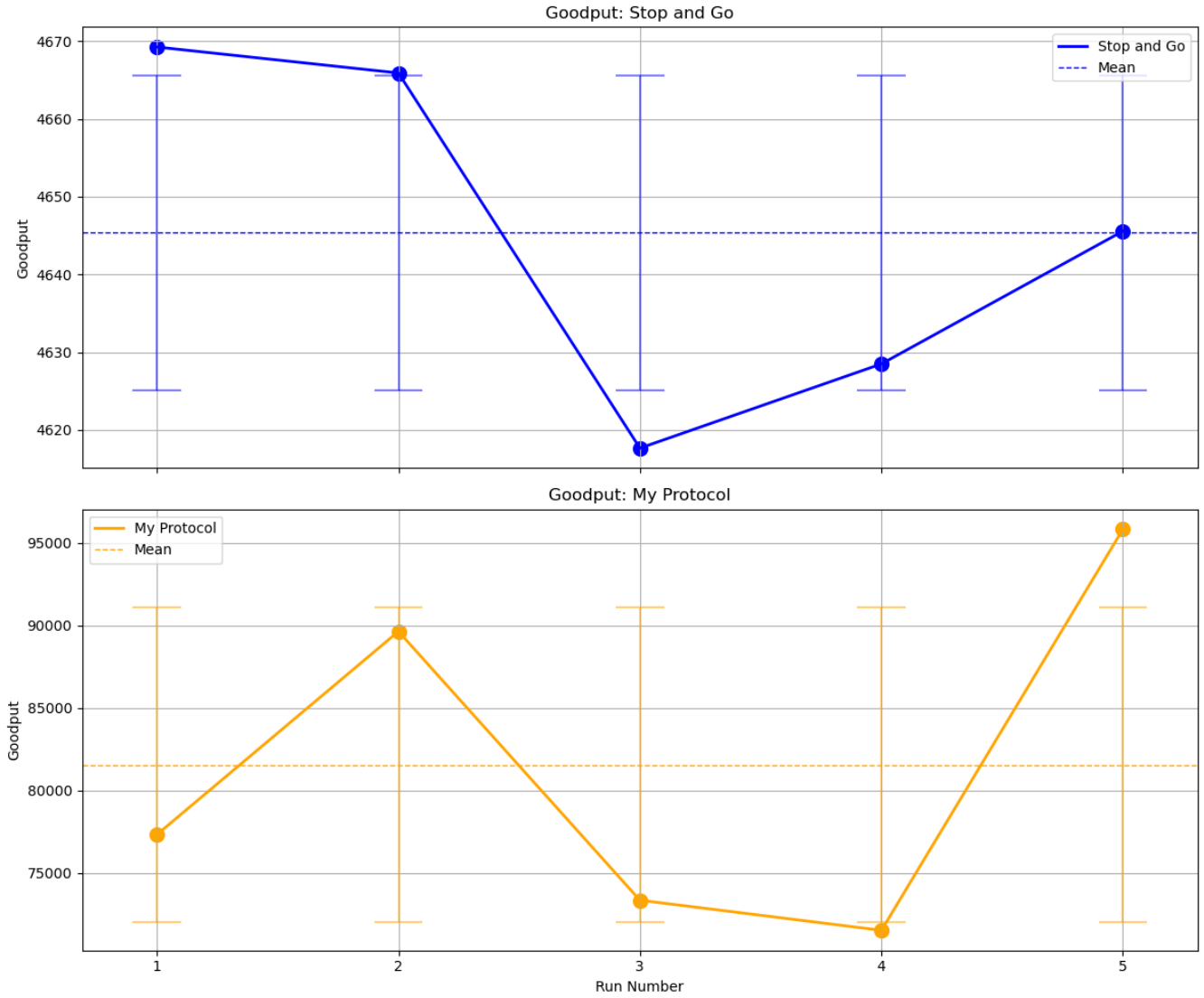
Variance graphs

- You need graphs for the goodput over 5 runs with your protocol vs stop and go (including variance in the graphs)

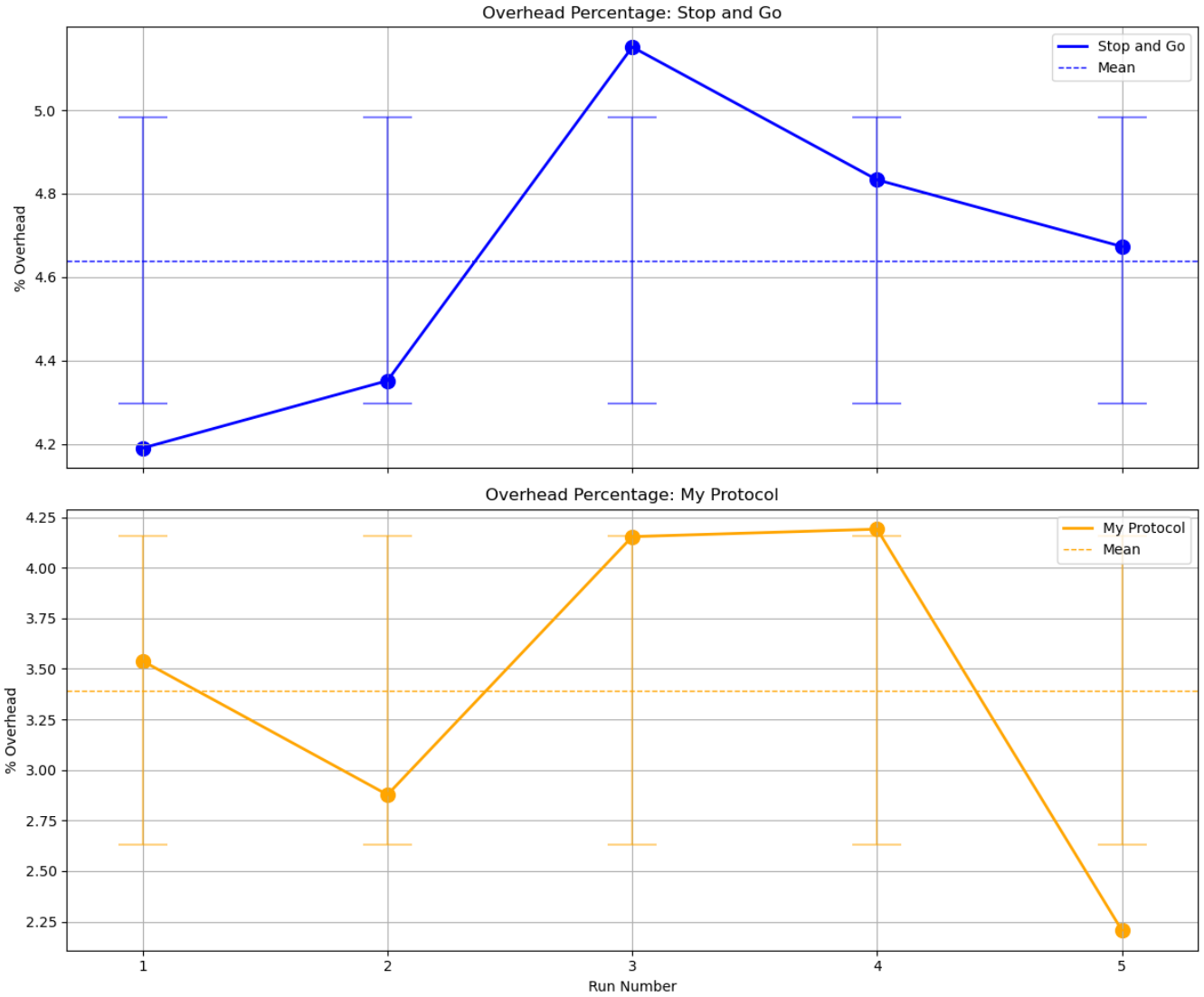
- You need graphs for the overhead over 5 runs with your protocol vs stop and go (including variance in the graphs)

- Parameters should be:
  - Bandwidth 200,000 bytes/s
  - One way propagation delay : 100 ms
  - Loss is 2% and reordering is 2%
  - The window size that works best for your application

Variance graphs (goodput)



Variance graphs (% of overhead)



Interim Protocol Description

- How did you set your timeout values?

Timeout = (max_packet_size/ link_bandwidth + 2 * propagation delay) * 1.5, the multipler 1.5 can be other number since I found that if timeout on this value without multipler can cause some issue.

And my timeout also has the exp backoff for it. The max timeout time is going to be 4

- What window size did you use? Why?

Window = int(2*propagation delay*(link_bandwidth/max_packet_size)) + 1

Link_bandwidth / max_packet_size = number of packet per sec link can take.

RTT * number of packet per sec link can take = window that can send to avoid congestion on emulator

Also in order to avoid 0 window size on low link_bandwidth, +1 is for it.

- Does your code automatically adjust timeout and window size to network configuration? If so how?

Yes, following the equation above. But it is not dynamic since it is calcaulating from the config. The dynamic windows size will be done in the final version.

Interim Protocol Description

- Provide a couple of lines of detail about your ACK scheme.

My ack scheme is really interesting, because I tried to optimize on Go back N already, it acts like a cumulative ACK when the packet is dropped or reordered, the receiver will buffer the received_seq_num > expected_seq and drop any packet exceed the buffer size and keep sending ack to sender about the expected_seq. And when the expected_seq comes, it writes all continuous seq_num in buffer and only send one ack till the seq_num it stops. And other time, the behavior is just one packet send, and then ack back.

- Does the sender only retransmit on a timeout, or does any other condition trigger a retransmission in your implementation?

Yes, if sender see two same ack, it will resend the packet of windows (this is going to be my optimization later since sender does not need to resend the whole windows)

- Please list any other optimizations not listed above that you implemented.

I also tried to suppress the resend request ack sent by the receiver. It has a counter inside the count the packet received. When the num_packet from the sender exceeds the timeout count, it will ack the expected_seq again at that time. It avoids receiver spamming the ack over the sender. Right now I set to same size of the windows size.
Also for the sender, I have used the exp back off for the timeout.

Summary of features

Protocol question	**Answer**
Window size (bytes)?	40960
Automatic window size adjustment?	No
Automatic timeout adjustment?	No
Cumulative ack?	Yes
Selective ack?	Yes
Non-timeout based retransmit?	Yes

Interim Report Discussion

- If this is your interim report, what do you plan on implementing between now and the final report?
- Sender does not need to resend the whole window size when it sees multiple ack, basically, implement the selective ack.
- Cumulative ack improvement, right now I am still acking back every single packet only when packet reorder or drop will activate my cumulative ack scheme, but it is not needed. I can ack they cumulatively back even in a normal case. Just improving my half-implemented cumulative ack implementation.
- Fast retransmit feature improvement together with cumulative ack to help deal with the packet reordering and goodput because retransmit based on the timeout is expensive and packet can arrive late if reordered. So it can trigger retransmit event unintended.
- Dynamic windows size for because sender can know the packet drop, so I will keep sender at max windows of 50 first, and if it sees the drop which is duplicate ack + sack from receiver, then it adjust the windows by half and then increase the windows size slowly. Like AIMD that is discussed in the class. (this is just an idea, not tried yet)
- [For final report, don’t delete this slide, but instead move to the end as reference].

Summary of changes: final report

1. Cumulative Ack: Make the cumulative ack to be real cumulative ack (not acking back every single time), right now it acks back when it receive 10 counts or timeout receiving sender packet. Note: I remove the ack suppression from the interim report since it is a half implemented cumulative ack. And when the Cumulative ack count set to be 1, there will be no suppression on ack that sender will receive, the receiver will spam the ack.
2. Fast Retransmit: the fast retransmit will transmit all of the unack packet from current packet ack to offset of 0 – 2 * cumulative ack count and it will fast retransmit based on the 3 duplicate ack (this speed up the multiple dropping in close range. Example. packet 100 dropped and packet 105 dropped while it can send back packet 100 and packet 105 together instead of only packet 100 to avoid another retransmit event at packet 105. Also, the fast retransmit mechanism also help with the packet of reordering issue since the reordered packet can just arrive late (not immediately but in the next 2 ack)
3. Selective Ack: Added selective ack to avoid sender resending the whole window. The receiver will send sender the received packet in the sack list with the sack list length. And sender can see it to remove the acked item in sender buffer. And when sequence number in the receiver buffer connects, it will just write into the file and clear the whole buffer. And receiver buffer will not exceed the size of the sender windows size because in my config I set windows size of 40 and cumulative ack count to be 10, so before it fills the buffer to windows size of sender, it already triggers fast retransmit/timeout from sender and sender will only send unacked packet until receiver acks back (means that receiver writes to the file and clears its buffer)

it is a bit hard to explain in context, maybe in example it will be clear:
Sender: sending packet 1 – 40: dropped packet 31 Receiver received packet 1 – 30, 32 – 40 and write 1 – 30 into the file and buffer packet 32 – 40 and ack back 31 (ack number). Sender see the ack 31 and know 1 – 30 are acked and sending 41 – 70 (30 packet instead of 40) while sending 41 -70, the receiver acked 31 at end of receiving 40 and 50. and trigger fast retransmit. (when it receive the unexpected seq, it immediately acks back one time already, so at 40 and 50 that is 2 time) 3 time to trigger fast retransmit. And sender resend only the packet 31 in this case (it will check 31 – 50 have any other dropped packet from sack list and send it together with retransmit) and receiver will receive 31 and write to file, clear buffer to the next missing packet.

Ablation table (final report only)

- Compare your final design with at least two other alternatives that may have performed less well.
- Example alternatives involve:
  - Turning off a feature (e.g., selective ACK) that was important for performance.
  - A design feature that you tried which (maybe surprisingly) hurt performance

- Feel free to add more rows to the table if necessary

- Parameters should be:
  - Bandwidth 200,000 bytes/s
  - One way propagation delay : 100 ms
  - Loss is 2% and reordering is 2%
  - The window size that works best for your application

Ablation table

Design Alternative Short description E.g., ”Disabling selective ACK.” “Adding feature X”.	Goodput (bytes/s)	Overhead (% of bytes)
Selective ACK: Off Cumulative ACK: On Fast Retransmit: On	80347.91[4270.07]	59.32[1.73]
Selective ACK: On Cumulative ACK: Off Fast Retransmit: On	105663.03[13158.12]	30.36[6.60]
Selective ACK: On Cumulative ACK: On Fast Retransmit: Off	86659.12[1412.89]	12.10[4.20]

Final report only: Ablation results discussion

- What features were most crucial to achieving good performance? Why?
- Were there features that you thought should help but didn’t work out? Why do you think they didn’t improve performance?
- What ideas do you have for the future (that you did not implement)?