Lab 2 template

Junyoung Kim, kim3722@purdue.edu

- 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)	3,808.14[84.14]	3,653.14[266.39]	1,596.58[13.07]
Overhead (% of bytes sent)	25,384.20[2,744.36]	29,142.40[8,292.28]	26,373.20[1,625.09]

Parameters should be:

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

Impact of window size

	Smallest window size = 2 bytes	Second smallest window size = 5 bytes	Medium window size = 10 bytes	Second largest windows size = 15 bytes	Largest window size = 17 bytes
Goodput (bytes/s)	93,013.70 [15,117.29]	98,013.57 [15,521.50]	98,722.15 [9,829.18]	87,730.67 [1,4381.76]	106,302.34 [1,953.44]
Overhead (% of bytes sent)	14,938.40 [3,528.16]	15,727.80 [5,217.50]	12,561.20 [2,168.02]	12,759.60 [1,927.45]	15,527.00 [1,655.54]

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)	106,302.34[1,953.44]	3,808.14[84.14]
Overhead (% of bytes sent)	15,527.00[1,655.54]	25,384.20[2,744.36]

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 [optional for interim]

	5% loss and 5% reordering	2% loss and 2% reordering	No loss and no reordering
Your protocol goodput (bytes/s)	91,230.82[17,373.54]	106,302.34[1,953.44]	111,141.18[75.39]
Your protocol overhead (% of bytes sent)	30,563.40[8,997.87]	15,527.00[1,655.54]	2,272.00[0.00]
Stop and go goodput (bytes/s)	2,795.28[122.77]	3,808.14[84.14]	4,650.25[1.62]
Stop and go overhead (% of bytes sent)	70,406.00[7,779.58]	25,384.20[2,744.36]	2,835.00[0.00]

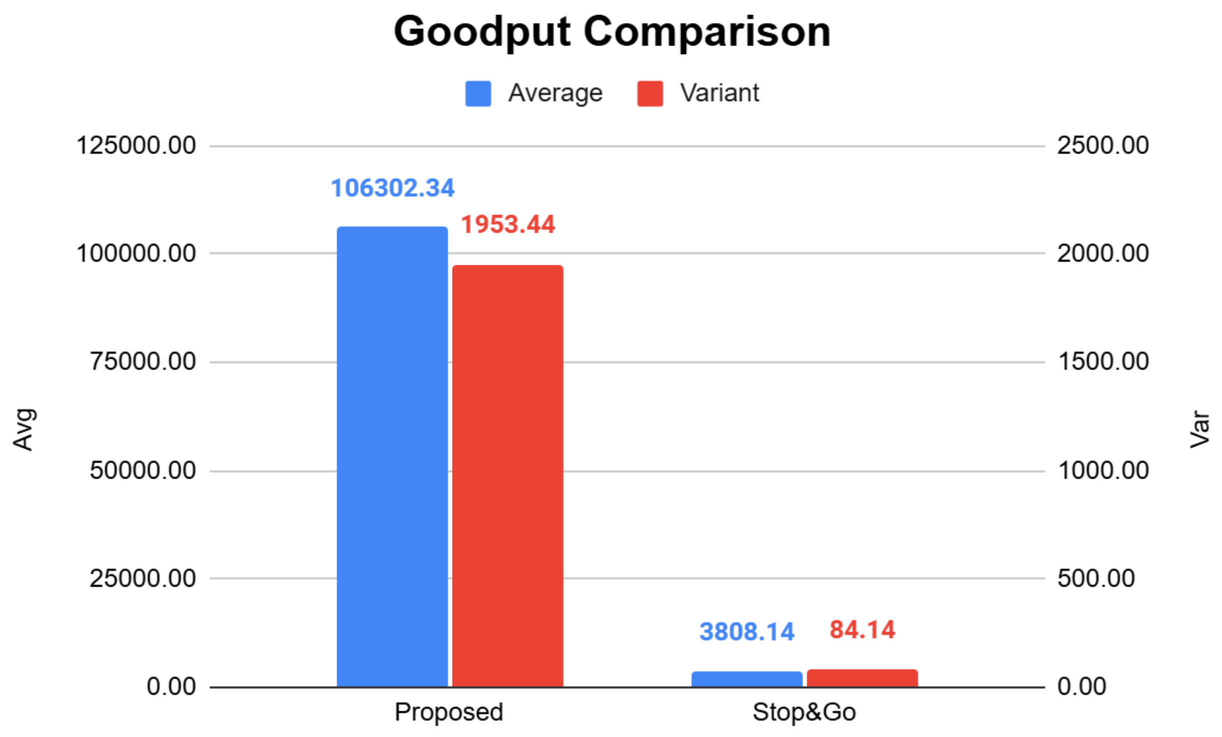
Parameters should be:

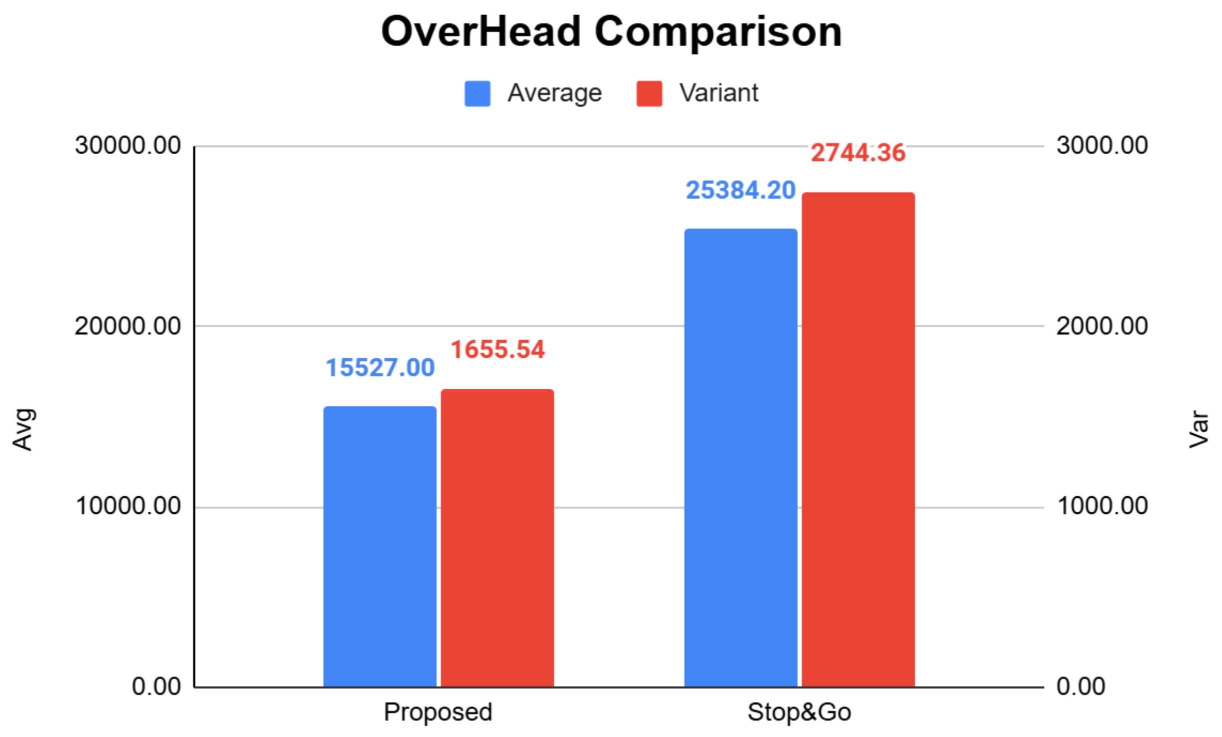
- Bandwidth = 200,000 bytes/s
- Use whichever window size works best for your algorithm

Variance graphs [optional for interim]

- 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





Protocol description (2 slides max)

- **How did you set your timeout values?**

Timeout is set the same as Stop and Go but only for end of window sender will receive NACKs.

- **What window size did you use? Why?**

Window size is changing continuously because it depends on the network situation, having a different window size will improve quality of performance.

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

I used the PID controller to adjust the window size based on the number of missing packets. This PID controller will aim to achieve the given ratio (total packet delivered / total packet sent) by increasing or decreasing next window size after each transmission window.

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

Algorithm uses a cumulative NACK + max packet ID. After each cycle of window transmission, sender will send EOW to indicate End of Window signal. Receiver will respond to this with cumulative NACK. This cumulative NACK is missing packets from packet ID 0 to whatever the maximum ID the receiver received. Sender will assume anything that has higher ID than given max packet ID is dropped or has not been transmitted. This way the sender can find out missing packets without providing additional information about the total number of packets to receive but also reduces the amount of data transmitted by not ACK all received packets. EOW and NACK + max packet ID will operate with Stop and Go methods to ensure perfect communication.

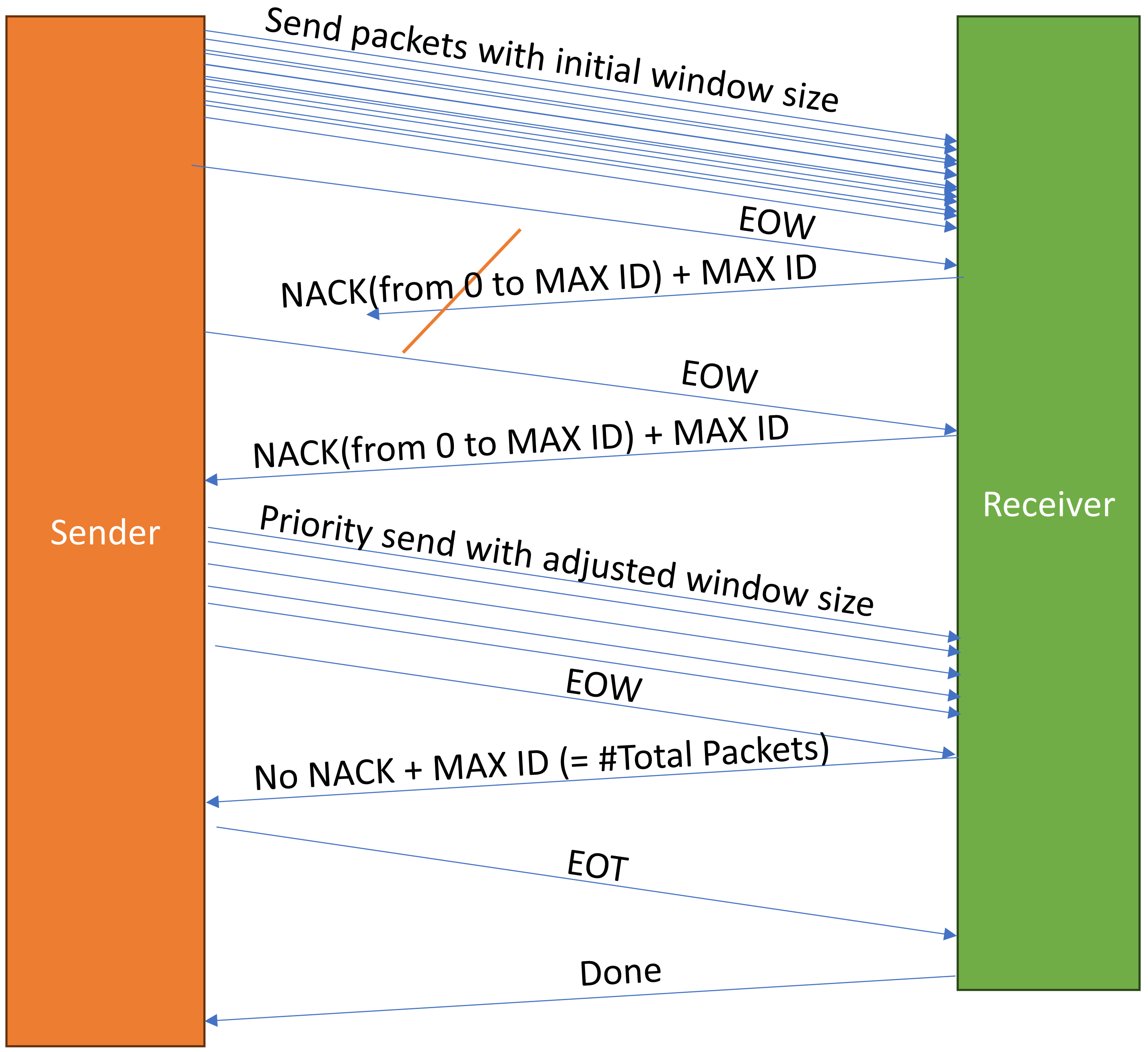
- **Does the sender only retransmit on a timeout, or does any other condition trigger a retransmission in your implementation?**

The algorithm has a buffer where it stores packets to send. The buffer will pop the packet when the sender needs to send. When the receiver responds with cumulative NACK, missing packets will be inserted in front of other packets to have higher priority. Once the sender starts send again, the sender automatically retransmits the missing packet first.

Protocol Drawing

- **Additional features?**

Added XOR encryption and decryption to add small security in the transmission. XOR was chosen to ensure network performance as it has a small header and fast encryption and decryption.


Sender
Receiver
Send packets with initial window size
EOW
NACK(from 0 to MAX ID) + MAX ID
EOW
NACK(from 0 to MAX ID) + MAX ID
Priority send with adjusted window size
EOW
No NACK + MAX ID (= #Total Packets)
EOT
Done

Summary of features

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

Summary of changes: final report only

- Compactly summarize changes since your interim report.

1. Added new acknowledgement system.

   - No more initial setting to let the receiver know the total number of packets. Receiver will respond with cumulative NACK + biggest packet ID received to indicate the highest order of packets it received.

2. Added Window size adjustment
   - Used PID controller to achieve given “total packet delivered / total packet sent” ratio.
   - Every window sent, the controller will evaluate the current ratio and adjust the next window size.

3. Added buffer & priority stacking
   - Used deque as a sender buffer. When sender sends the message, it will pop the packet and send it to the receiver.
   - When receiver answers with NACK, it will prioritize the missing packet and insert to front of the deque.

4. Added security feature
   - Used XOR encryption to keep simple header and fast performance.

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 (final report only)

Design Alternative Short description E.g., "Disabling selective ACK." "Adding feature X".	Goodput (bytes/s)	Overhead (bytes)
Proposed PID setting: target_ratio=0.50, kp=30.0, ki=0.5, kd=0.05	106,302.34[1,953.44]	15,527.00[1,655.54]
PID setting 1 target_ratio=0.70, kp=5.0, ki=0.1, kd=0.1	70,080.38[4,927.83]	12,193.80[2,709.86]
PID setting 2 target_ratio=0.90, kp=10.0, ki=0.2, kd=0.05	59,933.59[3,302.68]	14,976.80[4,042.75]
Fixed window size + Cumulative NACK	49,591.19[2,871.45]	14,420.60[1,625.28]
Adjustable PID window size + Non-timeout ACK	154,180.97[4,964.74]	89,395.40[21,887.86]

Final report only: Ablation results discussion

- What features were most crucial to achieving good performance? Why?

  Having correct methods to acknowledge the packet was very important to reduce the Overhead value. In the proposed model, cumulative NACK helped the most to reduce overhead value.

- Were there features that you thought should help but didn’t work out? Why do you think they didn’t improve performance?

  I thought non-timeout based retransmit will help the most as the sender will not wait to send the next packet. However, this did not work well as sending ACK for every packet increased unnecessary transmit. It also sent many duplicate packets when there are few missing packets as ACK not arrived yet, but sender must keep send until ACK is received for all the packets.

- What ideas do you have for the future (that you did not implement)?

  I would like to add a different controller other than PID because the performance of this controller depends on the parameter tuning. It will be nice to add a learning type controller such as a reinforcement learning controller.

Interim Report only: Discussion

- If this is your interim report, what do you plan on implementing between now and the final report?
  - Current version is optimized by reducing the required number of Stop and Go communications. However, it still preserves the original problems of Stop and Go. I will work on more effective communication that covers problems of the current Stop and Go design.
- [For final report, don’t delete this slide, but instead move to the end as reference].