ECE 50863
Project 2 Report Template

Yuan-Yao Lou / lou45@purdue.edu

Submit as a pdf, labeled <firstname.lastname.Project2.first.pdf>, <firstname.lastname.Project2.second.pdf> or <firstname.lastname.Project2.final.pdf>

Instructions

- Please follow this template for all milestones, but update the material with each milestone.
- Answers should be typed. Handwritten documents are not permitted.
- Your report should be submitted in **pdf format only.**
- Please keep your answers to the point.
- Graphs should be carefully plotted, with the **X and Y axis clearly labeled with appropriate legends.**
- For a slide with a graph have a 1 line **"take-home" message (what does the result show?)**
- Avoid 2 graphs on the same slide to ensure clarity.
- For all graphs, **use the file "to_send_large.txt"**

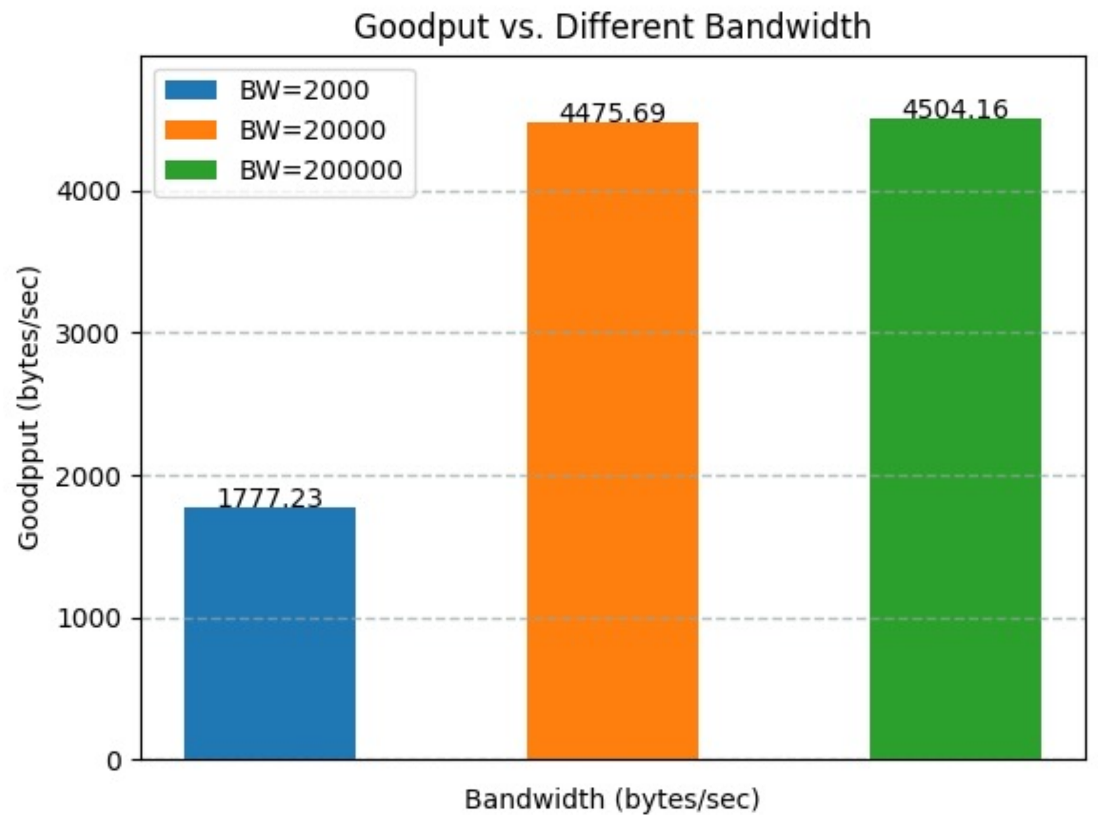
Results for Stop and Go

- Use three configurations. For all three:
  - Use one way **propagation delay of 100 ms.**
  - Set **Modest loss (2%)** and **modest reordering (2%).**
  - Vary bandwidth as: **200000 bytes/sec, 20000 bytes/sec, 2000 bytes/sec**
- Graphically show both overheads and goodput (efficiency) in two separate graphs. **Insert 2 separate slides following this.**
- Suggested format:
  - Goodput: Bar Chart with 3 bars, one for each bandwidth value.
  - Overhead: Bar Chart with 3 bars, one for each bandwidth value.

Results for Stop and Go - Goodput

Goodput increases as bandwidth increases.

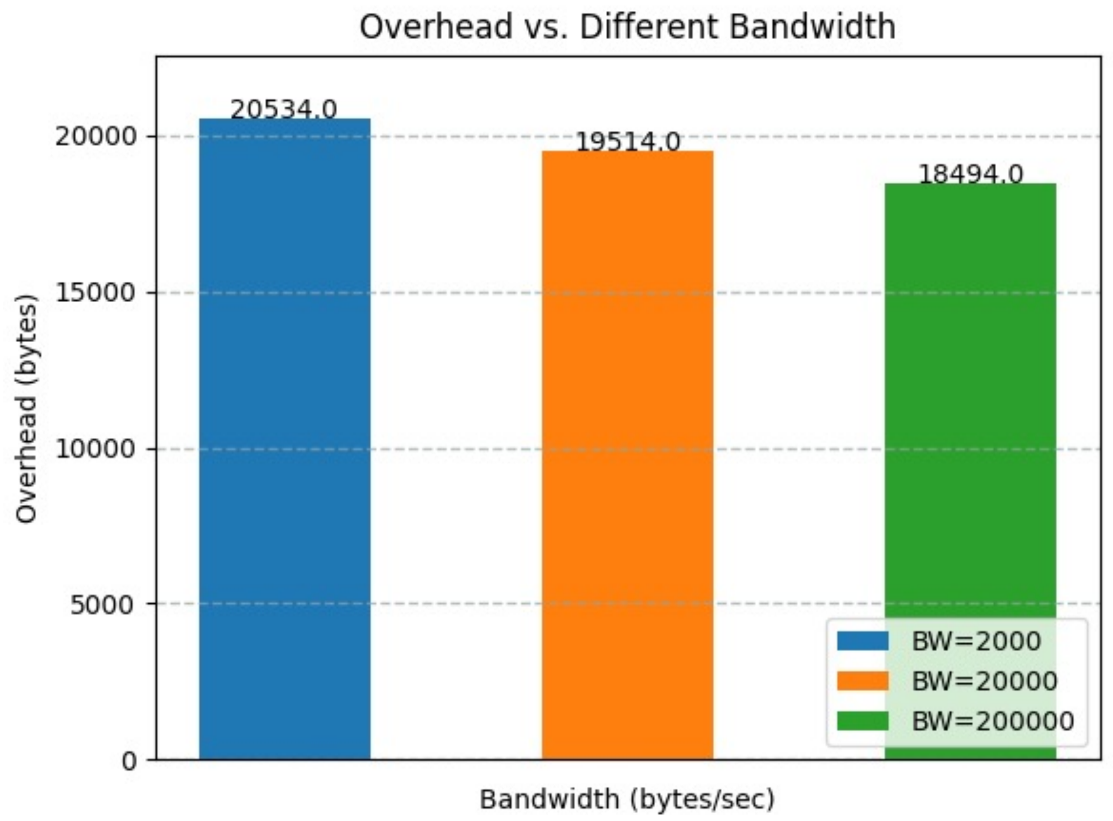
The large gap between BW=2000 and BW=20000|200000 is caused by the timeout values. Since lower bandwidth has smaller MAX_PACKETS_QUEUED value, so I set larger timeout values when BW=2000, leading to longer "Total Time". Besides, Goodput also depends on the frequency of dropping packets. Dropping packets too often leads to longer "Total Time".



Results for Stop and Go - Overhead

Overhead decreases as bandwidth increases.

Overhead is mainly affected by the retransmission times (higher number of retransmissions induce higher "Total Bytes Transmitted"). Thus, overhead value depends on the frequency of dropping packets. Properly set the timeout values could avoid extra retransmissions, reducing the Overhead.



Document your design decisions.

- Please answer each question in the following slide in 1-2 lines each.
- For initial report, answer what you propose to do and say (Proposed) to indicate this has not been implemented.
- In the interim report, update the answers to indicate what you have completed, and what you still propose to do.
- In the final report, **update the answers to indicate what you finally implemented.**

Designs Decisions – Final Milestone

- **How did you set your timeout values?**
  I fix the timeout to 0.2 due to the continuous transmission in my protocol. So generally, the timeout only happens at the end of transmission when there are still ACKs missing but the receiver does not respond timely.
- **What window size did you use? Why?**
  I set window size to 40 statically by using delay bandwidth product since it is proved to be efficient by the theory.
- **Does your code automatically adjust timeout and window size to network configuration? If so how?**
  The timeout is a fixed value. For window size, since this stage's experiment uses static link bandwidth, I fix the value to 40 in my code. However, the value is set based on the delay bandwidth product, which is affected by network configuration.

Designs Decisions – Final Milestone

- **What ACK scheme did you use? Cumulative/Selective/Both? Provide a couple of lines of detail about your ACK scheme.**
  I use a selective ACK scheme. When consecutive ACKs arrive, my protocol records the ACKs and transmits the next packet. When there is a gap between ACKs, my protocol transmits the packets that should be sent out and record the packets that do not receive the corresponding ACK for future retransmission.
- **Does the sender only retransmit on a timeout, or does any other condition trigger a retransmission in your implementation?**
  Besides retransmitting on a timeout, my protocol marks the packets that do not receive corresponding ACK and periodically check with the receiver to see whether the ACK or packet is lost, then only retransmit the necessary ones.
- **Please list any other optimizations not listed above that you implemented.**
  To ensure correctness, I pre-process the data transmitted by the sender so that sender and receiver can sync when the transmission is finished.

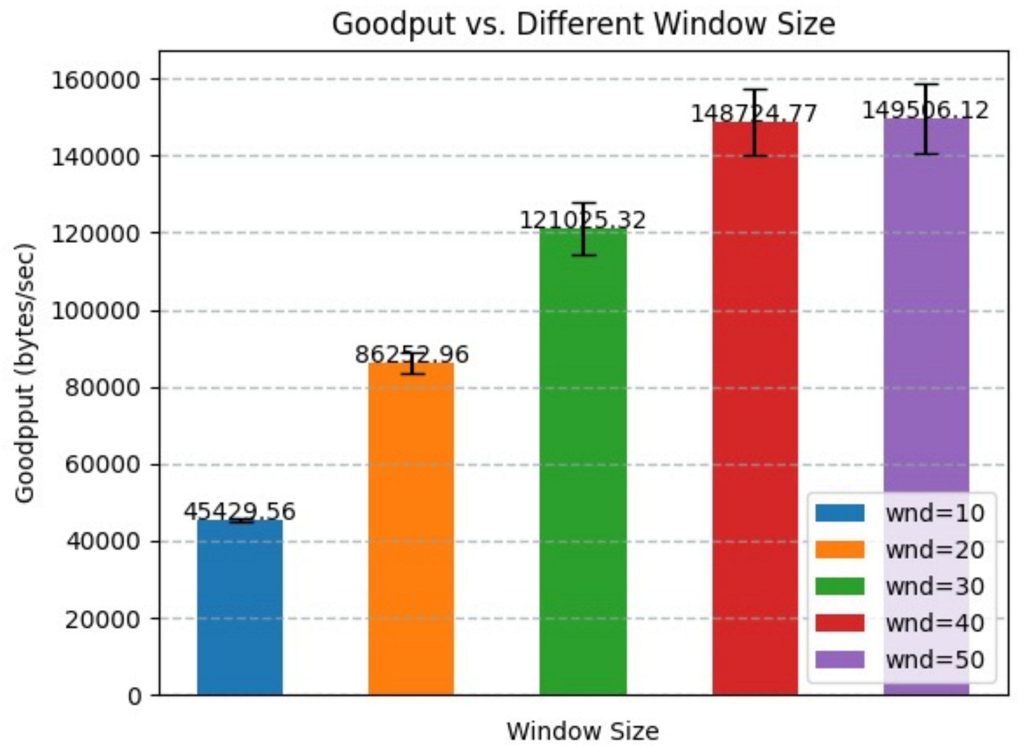
Results with your protocol: Impact of window size

- Use this configuration:
  - Bandwidth: **200000 bytes/sec**
  - One way **propagation delay of 100 ms.**
  - **Modest loss (2%)** and **modest reordering (2%)**
  - **Iterations: 100**
- Graphically **show the goodput and overheads for different window size choices.** Insert 2 slides after this one.
- Suggested format:
  - Goodput: Bar Chart: 5 different window sizes, 1 bar per window size.
  - Overhead: Bar Chart: 5 different window sizes, 1 bar per window size.
  - **Pick window sizes to cover interesting points, and show the right trend.**

Results with your protocol: Goodput

Goodput increases as window size increases.

Although window size 50 has the highest goodput, it does not exceed much compared with window size 40. Moreover, it has the tradeoff between goodput and overhead (see next slide).

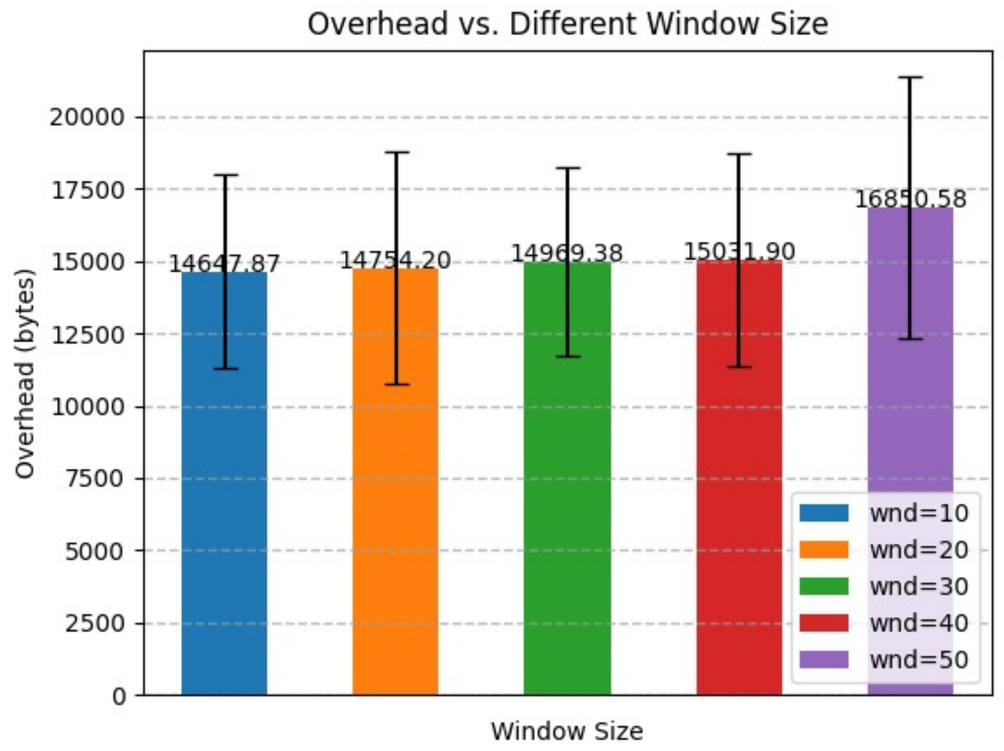


Results with your protocol: Overhead

Overhead remains almost the same except the window size is 50.

I retransmit the packets that do not receive corresponding ACK by sending small overhead sync packet while transmitting the subsequent packets, so the overhead strongly depends on the probability of dropping packets (=2%).

On the other hand, when the window size is larger than 40, it might cause the buffer to be full and lead to extra packet drop and higher overhead. Thus, the standard deviation is also larger.

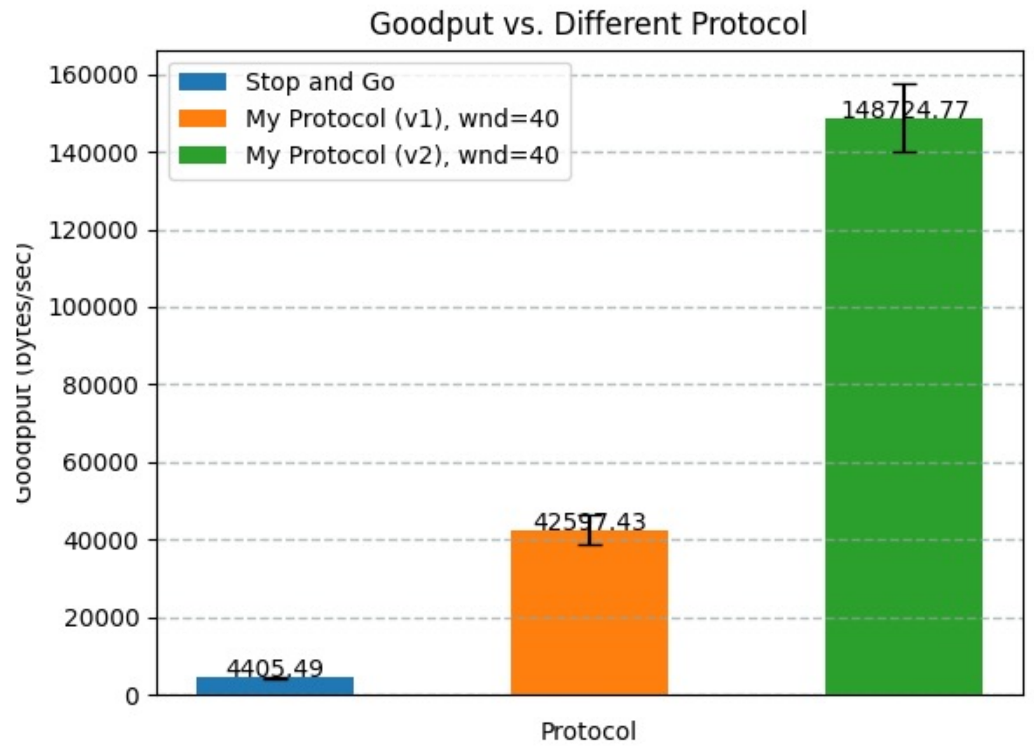


Results: Your protocol Vs. Stop and Go

- Use this configuration:
  - Bandwidth: **200000 bytes/sec**
  - One way **propagation delay of 100 ms.**
  - **Modest loss (2%)** and **modest rebuffering (2%)**
  - **Iterations: 100**
  - **Pick what you think is the best window choice for your scheme: 40**
- Graphically compare (a) your final protocol (b) your interim protocol and (c) Stop and Go with respect to goodput and overheads (insert 2 slides, one for each metric).
- Suggested format:
  - Bar Chart: **3 bars. The bars correspond to the 3 protocols.**
  - Interim report need have only 2 bars (for Stop and Go and your interim protocol), and you can update with additional bar (for final protocol) in the final report.
  - For the best credit on this, **please run each experiment multiple times** and include the **mean and error bar.** This is because performance can be variable across multiple experiments.

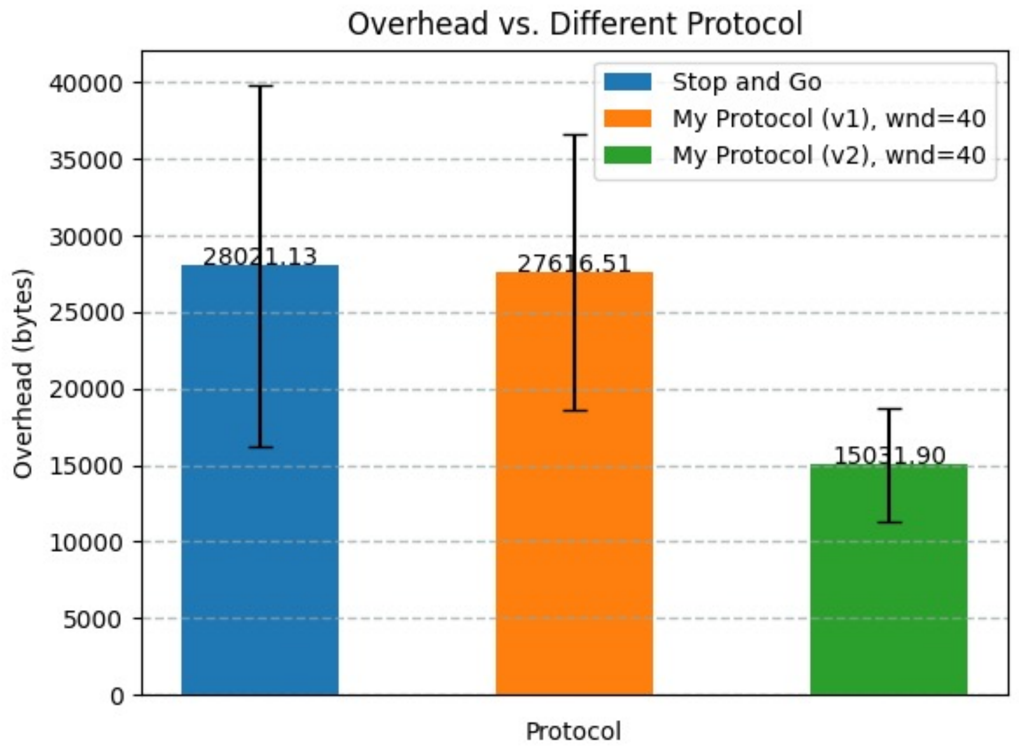
Results: Your protocol Vs. Stop and Go: Goodput

My protocol is significantly better than the protocol "Stop and Go" (around 33.75x) and better than my protocol v1 (around 3.49x).



Results: Your protocol Vs. Stop and Go: Overhead

My protocol is significantly better “Stop and Go” and my protocol v1 (both around 0.53x) and with a much more stable standard deviation.

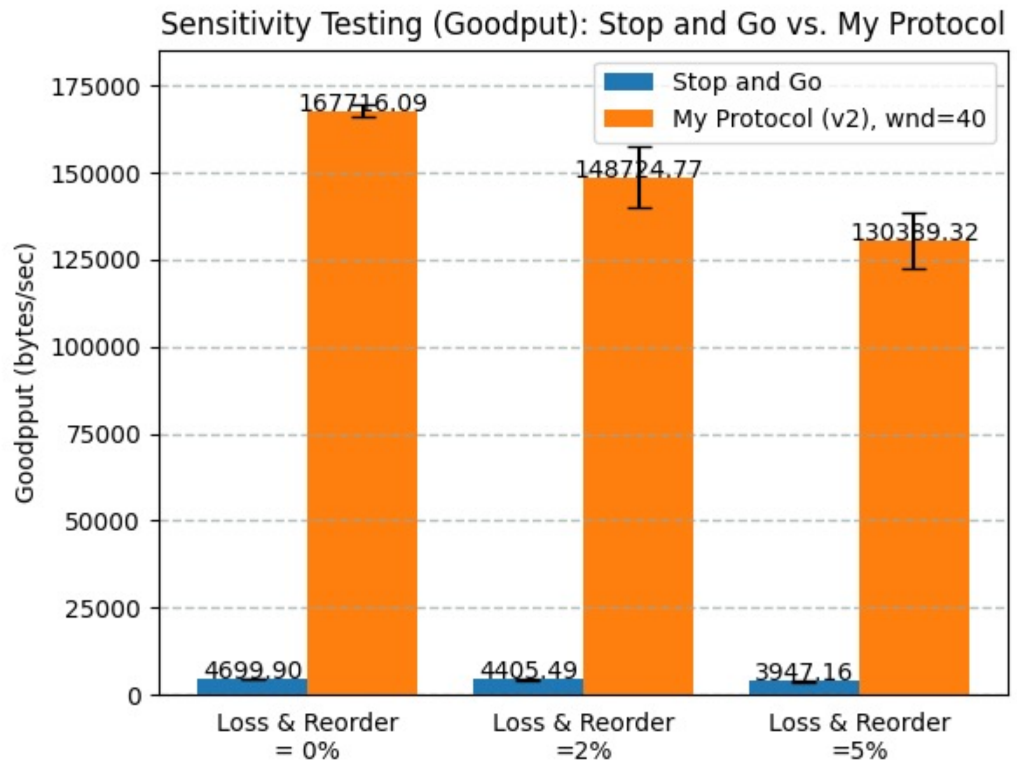


Sensitivity Testing [Final Report only]

- Use this configuration:
  - Bandwidth: **200000 bytes/sec**
  - One way **propagation delay of 100 ms.**
  - **Iterations: 100**
  - **Pick what you think is the best window choice for your scheme: 40**
  - **Pick 3 configurations:**
    - Higher loss (5%) and higher reordering (5%)
    - Modest loss (2%) and modest reordering (2%)
    - No loss, and no reordering
- Graphically **compare your final protocol and Stop and Go** with respect to goodput and overheads (insert 2 slides, one for each metric).
- Suggested format:
  - Bar Chart: 3 groups of 2 bars each. Each group corresponds to a configuration. In each group, the 2 bars correspond to Your Protocol, and Stop and Go respectively.

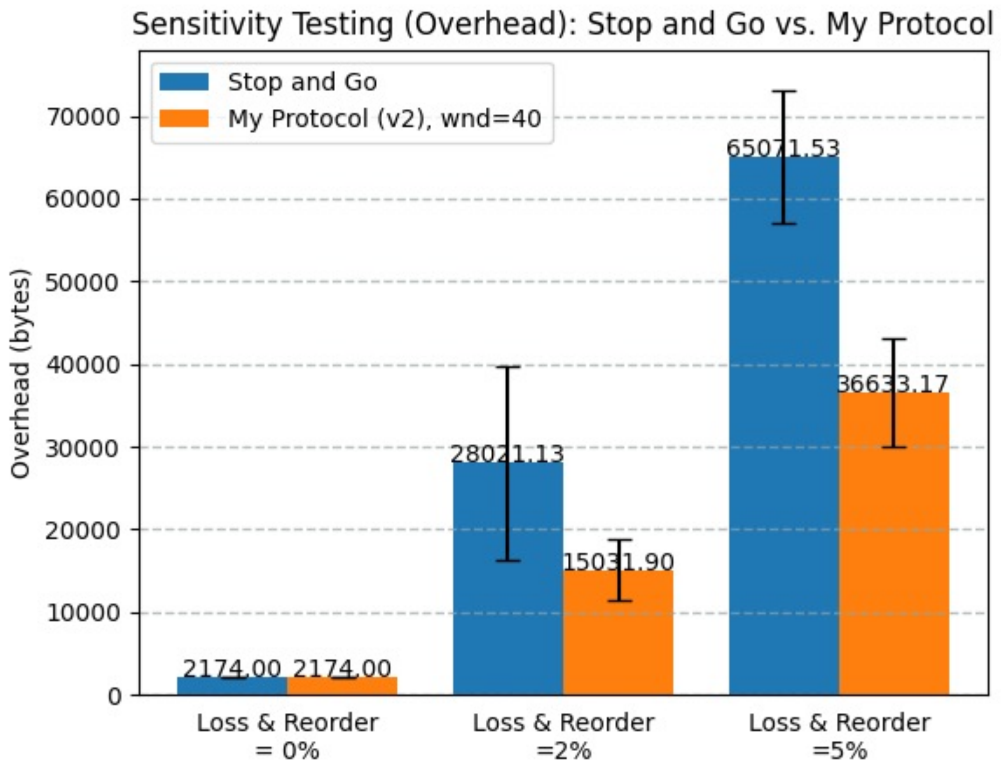
Sensitivity Testing: Goodput

Goodput of my protocol and “Stop and Go” both decreases as the probability of loss and reordering increases. My protocol performs better in all conditions.



Sensitivity Testing: Overhead

Overhead of my protocol and “Stop and Go” both increases as the probability of loss and reordering increases. My protocol performs better in all conditions.



Discussion/Conclusion – Interim Milestone

- **For Interim Milestone, do you think your results are satisfactory? How do you plan to improve results for the final report?**

  I strongly believe that my results could be improved, for both goodput and overhead. Waiting for all packets in the current window to be ACKed wastes time. I plan to keep moving my window once receiving ACK from the receiver.

  However, for the packets that do not receive corresponding ACK, I need to make sure receiver does receive them correctly.

Discussion/Conclusion – Final Milestone

- **Any interesting findings? Were particular optimizations particularly useful? Were some optimizations not as useful?**

  My initial approach assumes the receiver has an infinite buffer. Thus, I have two phases in my protocol. The first phase is transmitting the packets without worrying about the packet loss situation. The second phase is to review the packets that do not receive corresponding ACKs and retransmit them.

  I fix this issue by sending periodic sync messages from sender to receiver to check whether the missing ACK is due to data packet loss or ACK packet loss, then only retransmit the necessary packets. **This mechanism ensures the lost packets are retransmitted within 2x of window size (i.e., the receiver buffer size is bounded within 2x of window size).** Thus, there are no longer two phases in my protocol since sync messages are transmitted while transmitting the data packets.

Discussion/Conclusion – Final Milestone

- **Optimizations you should try in the future but could not get to?**

  One of the optimizations I still want to try is to **set timeout values dynamically according to the RTT values of each transmission iteration.** Currently, the timeout is a fixed value in my protocol by observing the network condition in local machines. However, this is not a realistic case. Thus, I think dynamic adjustment of timeout values might help my protocol be useful in the real world.

  On the other hand, **my protocol assumes that the size of the receiver's buffer is within 2x of the sender's window size** (based on log tracing, the time interval from a packet loss to receive the missing one is around 0.4s). This assumption may be valid for certain applications. However, My protocol **might need to be refined to accommodate applications with different requirements,** instead of sticking to this assumption.