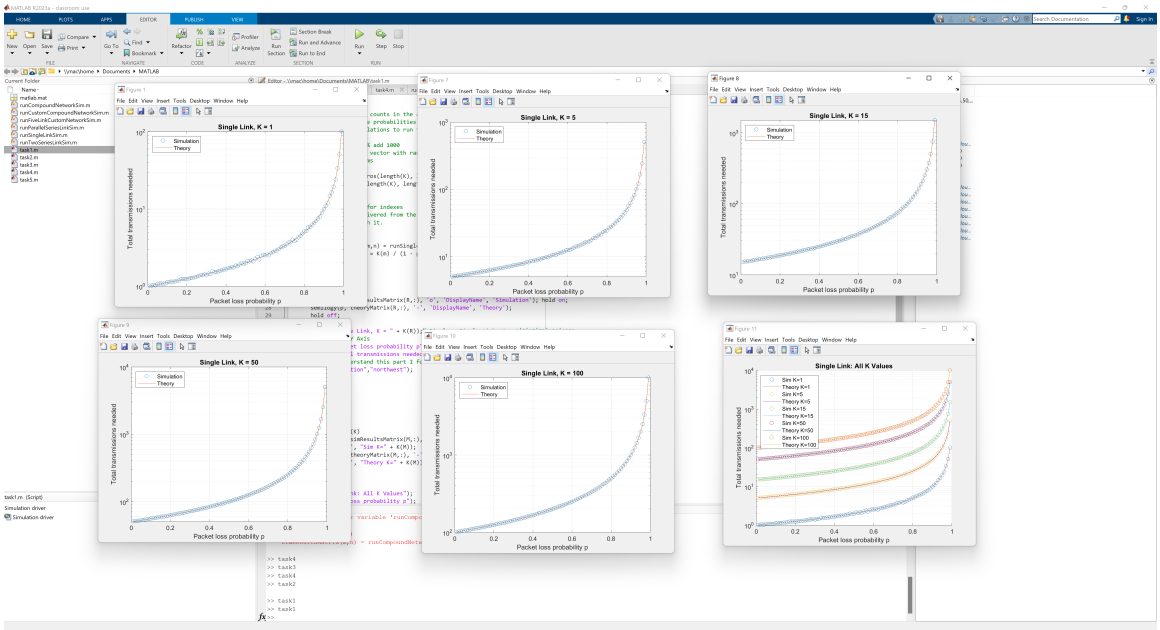
LAN Reliability Project

GITHUB: https://github.com/SleepiestFool01/LAN-Reliability-Project/

This project shows how different network setups affect reliability when packets have a chance of failure. Each task uses the same idea: send K packets, record how many transmissions it takes for all of them to make it across, and then graph the results. Everything was run 1,000 times to get an accurate average. Each task adds one more layer of complexity to the network, from one single link up to a compound network with different probabilities on each link. The figures basically show how adding redundancy improves reliability, and how adding more failure points makes it worse.

Task 1 Single Link

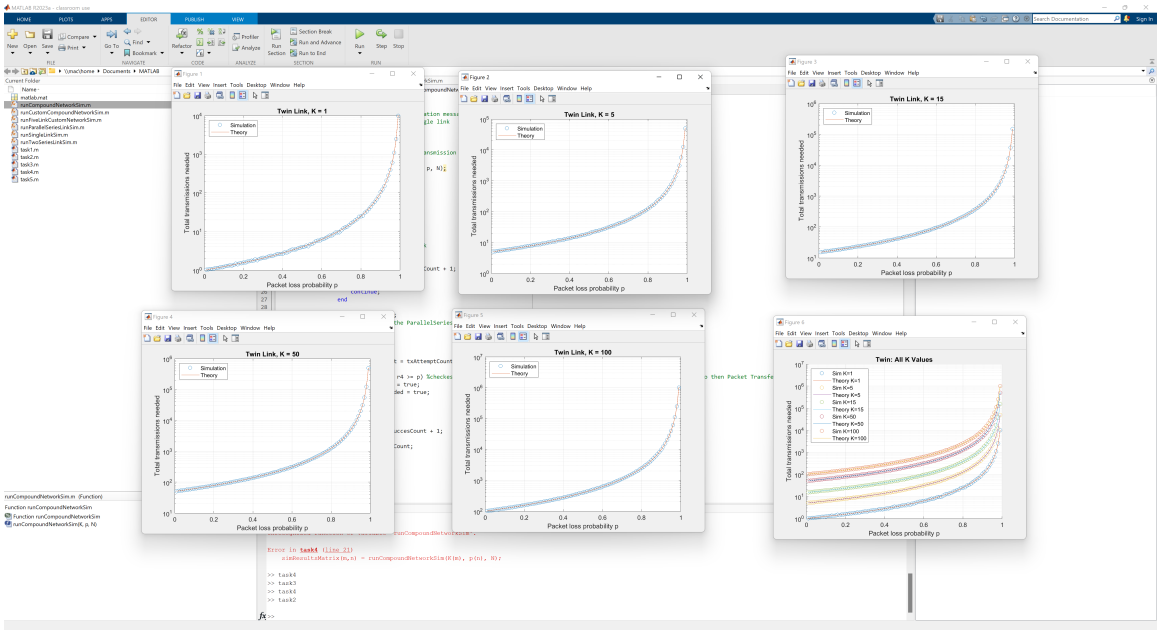
This first setup is just one direct connection. The code calls runSingleLinkSim(), which generates a random number and checks if it's greater than p. If it is, that packet succeeds. The graph shows the number of transmissions increasing quickly once p gets high (around 0.8 or so). The calculated line using K / (1 - p) matches the simulation line almost exactly, so the code works the way it should. Reliability clearly drops fast when failure chances rise.



Task 2 Two Links in Series

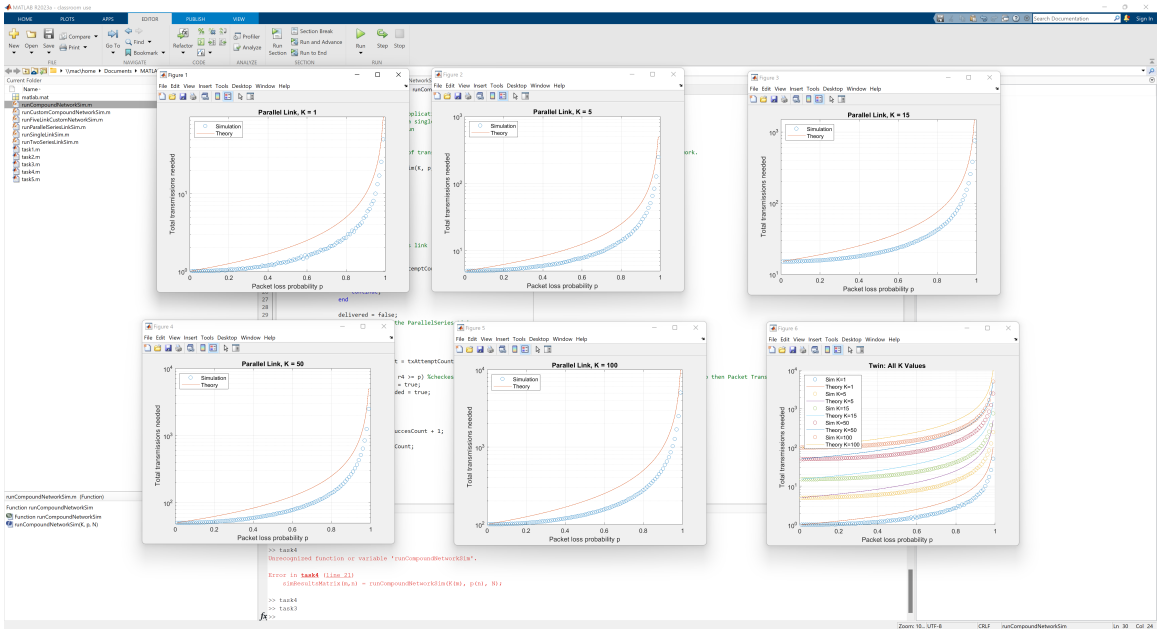
Task 2 uses two links in a row, meaning both have to succeed for the packet to get through. In the code, it just adds another random test before counting the transmission as good. Because the packet has to make it through both links, the reliability is worse than

Task 1. The plot climbs faster, and around p = 0.5 the average transmissions are roughly four times higher than before. This one also takes longer to run since it checks more conditions every loop.



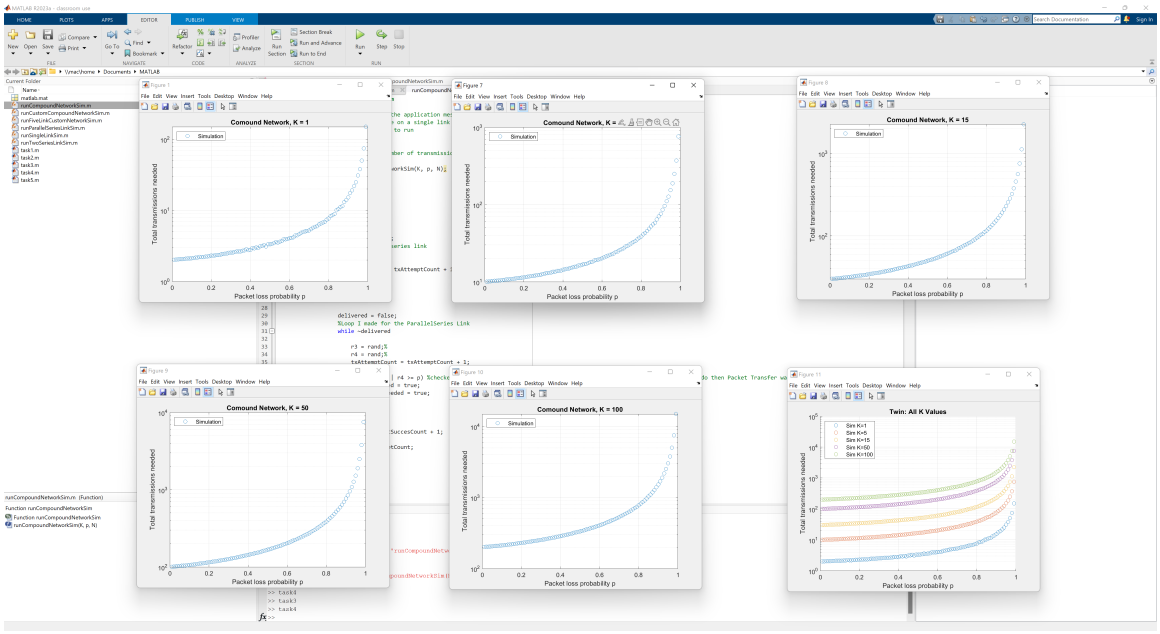
Task 3 Two Links in Parallel

For Task 3, I switched the logic so that the packet only needs to succeed on either link instead of both. The code still generates two random numbers but checks with an "or" instead of "and." You can tell right away in the figure that this version is way more reliable. The average transmissions stay low even as p increases. It's the same idea as adding a backup route, if one path fails, the other one might still work.



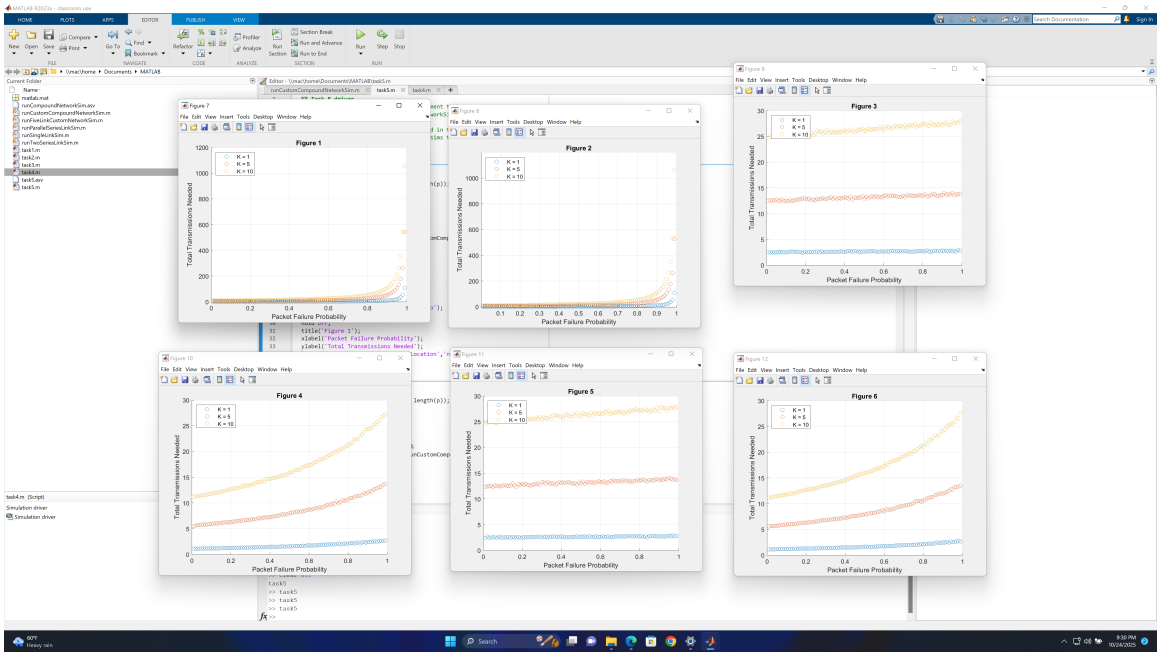
Task 4 Compound Network

This one mixes the two ideas. The packet can go across either link to reach the middle node, but then it still has to go through one final link to finish. So it's parallel first, then series at the end. It's kind of the "in-between" network. Such that its more reliable than the pure series setup but not as good as the pure parallel one. The graph lines sit right between the previous two tasks, which makes sense. It also takes a little longer to run since it runs both checks.



Task 5 Compound Network with Different Probabilities

Task 5 keeps the same compound layout but lets each link have its own probability of failure (p1, p2, p3). I wrote a new function called runCustomCompoundNetworkSim() that takes those three values separately. The driver runs six figures total. Each one changes which link is being swept from 1%–99% while the others stay fixed. The K values are 1, 5, and 10 like the table listed. When you look at the results, it's easy to see which link is the weakest because that's where the whole network performance drops. If one link is unreliable, the other two can't really make up for it. When all three are solid (low p values), the curves flatten out and the transmissions stay low across all K values.



Conclusion

Overall, every task shows the main idea of network reliability:

- Series links make things worse since all parts must work.
- Parallel links make things better by giving extra chances for success.
- Compound networks sit in the middle, and the weakest link usually decides how it performs.

All five tasks lined up with what I expected. The MATLAB results match the logic of real networks. More redundancy in a system means better reliability, and high failure probabilities cause huge spikes in transmissions.