Assignment 1: Make it Work.

Your team runs an independent network company. You own the switches and your customers' traffic runs through them. You will write one program that runs every switch you own and enables your network to function: packets arrive, broken links get routed around.

Your program will be dropped into a network it has never seen. It wakes up knowing only: "*you are switch 7, and you have cables on ports 1, 3, and 4.*" Who its neighbors are, what the network looks like, where destinations live; it must learn all of that by sending and receiving packets. This assignment requires the network to be able to **deliver and recover**:

1. **Deliver:** Discover your neighbors and the topology, install forwarding, and get every packet to its destination.
2. **Recover:** Links die and come back on schedules you can't see. There is no alert, a dead link just goes silent. Notice the silence and route around it.

You get practice worlds(networks) and the exact scoring code, with unlimited runs. Grading uses worlds you've never seen. After submission there will be a short in-class quiz of your own network's graded run.

How does your switch work?

Every switch has two halves, just like real hardware:

- **Data Plane:** Touches every packet, so it must be dumb and quick. it looks things up in tables or other precomputed information ("if destination starts with 10.0.90 → send out port 2") and does what the table says. It cannot think. When it encounters a packet for which it does not know what to do it punts the packet to the control plane.
- **Control plane:** This is the brain of the switch. Real code that can remember things and fill in those tables the data plane depends on. But it's slow, and it talks to the dataplane through a limited access, it cannot touch every packet.

Your program is both halves. Most of this assignment is learning to live with the split: the brain decides, the tables forward, and the brain only learns something when a packet gets punted to it. Resource limits(table sizes, compute) are real but generous, you'll only feel them if you are wasteful. The exact numbers are in *docs/LIMITS.md* in your kit.

Addresses

IP addresses belong to customer apps. Switches know only its own ID and its ports. How switches name each other inside your control messages is your design choice. (There are no MAC addresses in this world)

*Note: Real routers usually have an IP address on each interface. But those addresses are not needed to forward packets. They are used so hosts can send packets **to the router itself**: for example, to reach a gateway, run a routing protocol with a neighbour, or respond to `ping`. We removed all of those functions, so switches do not need addresses.*

What remains is the core forwarding job of a router: look at a packet's **destination**, match it against the forwarding table, and choose an outgoing port. We simplified everything around forwarding, not forwarding itself.

Control messages also do not need addresses. A switch sends one through a port, and it reaches the switch connected to the other end of that link. This is enough to discover neighbours. What goes inside these messages, including how switches identify themselves, is up to your design. MAC addresses are useful on shared Ethernet, where many devices hear the same transmission. Here, every link has exactly two ends, so choosing a port already determines who receives the message.

Every customer prefix is a `/24`, and its app responds at the first host address. For example, `10.0.90.0/24` is reachable at `10.0.90.1`.

A switch is **not** told which customer prefix is behind it. That mapping changes in every world and follows no pattern. Your switches must discover it!

Part 1- Deliver

A simple network(s) with 5-15 switches, one customer app on each, light traffic. "Light" means the offered load never exceeds link capacity: congestion will not drop packets in this assignment, so every lost packet is caused by your program. Discover your neighbours, learn the network, so every app can reach every other app.

You pass if: >= 99.9% of packets arrive(after an established warmup window). Delivery is counted as unique packets delivered ÷ unique packets sent, scoring starts after a 200 ms warmup (**W**) and stops 100 ms before the run ends. You are not punished for knowing nothing at the start, or for packets still in flight at the end. 99.9% is a floor, not a target. A packet delivered to the **wrong** app is a misdelivery, not a delivery. It counts against you exactly like a loss.

A warning about loops: We can see the true path of every packet, and your report card will show any packets that travelled in circles. Loops won't fail you here, but a looping packet burns capacity on every link it revisits, and capacity is what is graded in A2. A design that loops passes today and pays later!

Part 2- Recover

The same networks, but now links die and come back on a randomized schedule. The five Part 2 worlds are the same five topologies as Part 1; same switches, same cables, same customer prefixes, just running longer with links dying under them. So when recovery misbehaves, run the Part 1 twin first: if that one is clean, your routing is fine and the bug is in how you notice and route around silence.

There is no alert; the link just goes silent. Notice the silence (for example, neighbours send "hello" messages periodically and you look for the silence), route around it, and use the link again when it returns. We promise the network always stays connected, and that failures never land close enough together to pile up. You get 1000 ms of simulated time to recover from each one. That is the **recovery budget**.

You pass if: every app can still reach every other app, at least 98.0% of packets arrive over the same scored window as Part 1, and you recover from every single failure inside the 1000 ms budget.

Recovery is measured one event at a time, never averaged. When a link dies we watch the traffic that link was carrying and measure how long it takes to start flowing again. Every event has to clear the budget on its own. Eleven clean recoveries and one destination black-holed forever is still a fail. Links coming back up are scored the same way as links going down. A program that routes around a dead link but never notices its return is leaving capacity on the floor, and capacity is what A2 grades. For reference, our own solution recovers from every failure comfortably under 200 ms.
Why no alert? Real switches often do get an electrical warning when a cable dies. Yours does not, on purpose. Detecting the silence yourself is the skill being graded here, and the failures get quieter in later assignments: links that slow down, or drop some traffic and forward the rest. Nothing will ever warn you about those. Build the habit now.

Your Starter Kit

- **NetworkArena simulator:** a Rust SDK(any language that compiles to wasm is allowed, but we offer technical support only for Rust).
- **Hello_Switch:** A working example switch. It compiles, it runs, it forwards nothing, and it scores 0%. It is your hello-world.
- **Practice worlds:** Five networks, each given to you twice, once without failures for Part 1, once with three failure schedules for Part 2.
- **Learning_Switch:** 55 lines. Installs one route on the first punt. A step past hello-world, not a solution.
- **Replay viewer:** Any run produces a log you can open and step through in your browser.
- **Report Card:** The same grader we use :)
- **README.md**

The Quiz

Eventually, you'll take a short quiz about your own team's graded run.

On Quiz day: A one page design card with other notes is only allowed. No laptops or AI allowed. Every question that refers to a moment in your run shows you that moment, a picture of your topology at that instant, a switch's table, an excerpt of the log. You never have to remember a timestamp; you have to read the evidence and explain it. We are not testing whether you memorized your code. Every question can be answered by someone who knows the choices their team made and can interpret evidence from their own run. That is the skill being graded.

Three kinds of questions:

1) **What happened?** (can you explain what your design does?): We show a moment from your run, switch's table, relevant log excerpt and ask; which path is the traffic on?
2) **Why did it happen?** (rational behind your design)
3) **What would happen if?** (does your design extend across networks): "if link 2 had died instead of 5, what would have happened?", "how would you patch your design here?", "you are an adversary to this network, how will you break it?"; During grading we re-run your program with that one change, so we're holding the answer. A memorized replay doesn't help here; a model of how your design behaves does.

Every MCQ includes an "I don't know" option worth partial credit - a confident wrong answer scores less. Guessing is a bad strategy; knowing your network is the only good one.

Using AI

Use it as much as you want, for anything: writing code, debugging, explaining concepts, even generating practice questions from your own designs and logs (a great way to prepare for this course). We assume your code is partly or fully written by AI; that's the modern programmer.

Important: AI can write your routing protocol; it cannot sit your quiz. Every point of individual credit in this course sits where AI cannot go. Delegate the coding; don't delegate the reasoning and understanding.

The Report Card

Every run prints a report card. It is the same tool you use at home, so grading day holds no surprises. The card has four checks:

- **C1, delivery.** Did the packets arrive? At least 99.9% in Part 1, 98.0% in Part 2. One packet under the floor is a fail, so aim for zero loss.
- **C2, reachability.** Could every app reach every other app? One pair that never delivered a packet is a failure.

- **C3, loops.** Did any packets go in circles, or die of TTL exhaustion? **This won't fail you**, but will hurt you in A2 and further. Read it anyway; a loop is a bug waiting for a busier network.
- **C4, recovery.** Part 2 only. After every link goes down, and every link comes back up, packets must stop dying within 1000 ms. One slow recovery is a failure. In Part 1 this line says "skipped".

A run is OVERALL: PASS when C1 and C2 pass and C4 passes or is skipped. When a run fails, the card names the failed check, the moment it happened, and follows one dead packet hop by hop. Start there.

From cards to a grade. Each hidden Part 1 world is one run. Each hidden Part 2 world is three runs, one per failure schedule. Your score for each part is the fraction of its runs that pass. There is no partial credit inside a run. You get three graded attempts and we keep your best.

Grading Rubric

Component	Share	What It Measures
Parts 1–2 Team	~50%	Your network's behavior on hidden network topologies
Quiz; Individual	~50%	Whether you understand how your network works

You will have unlimited practice runs and three graded attempts. The best of your graded attempts will count.

Logistics

- Teams of max 2 people, single is fine.
- **Due date:** Sept 18th
- **Deliverables:** switch_<group_name/number>.wasm, design-card.pdf (the one page you'll be using in the quiz)
- **Graded Attempts:** Details on how this will work will be announced on Ed soon.