Command-Line to Codebase: Learning Software to Expand Your Engineering Toolkit

Thomas Kjær Aabo

You can start small (and still learn a lot)

Use evidence, not guesses

Start from a workflow you already trust

Automate it in tiny steps

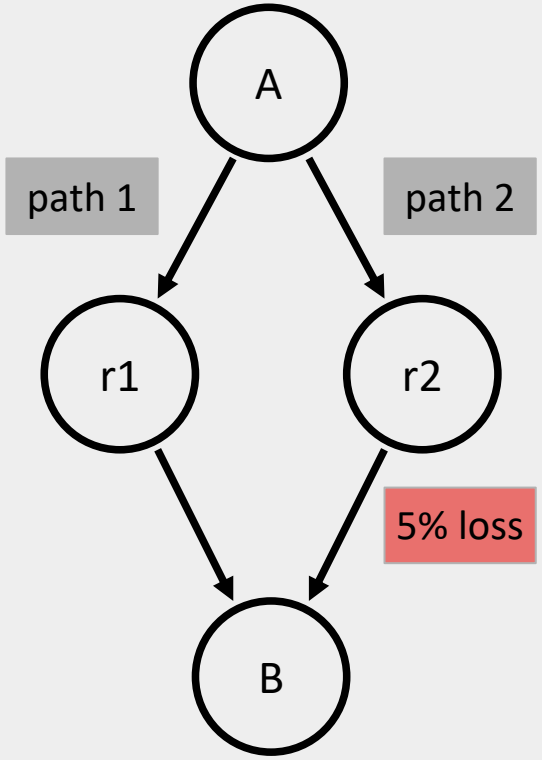
Start with one repeated workflow

- Pick a repeated workflow
- Make the first step safe and small

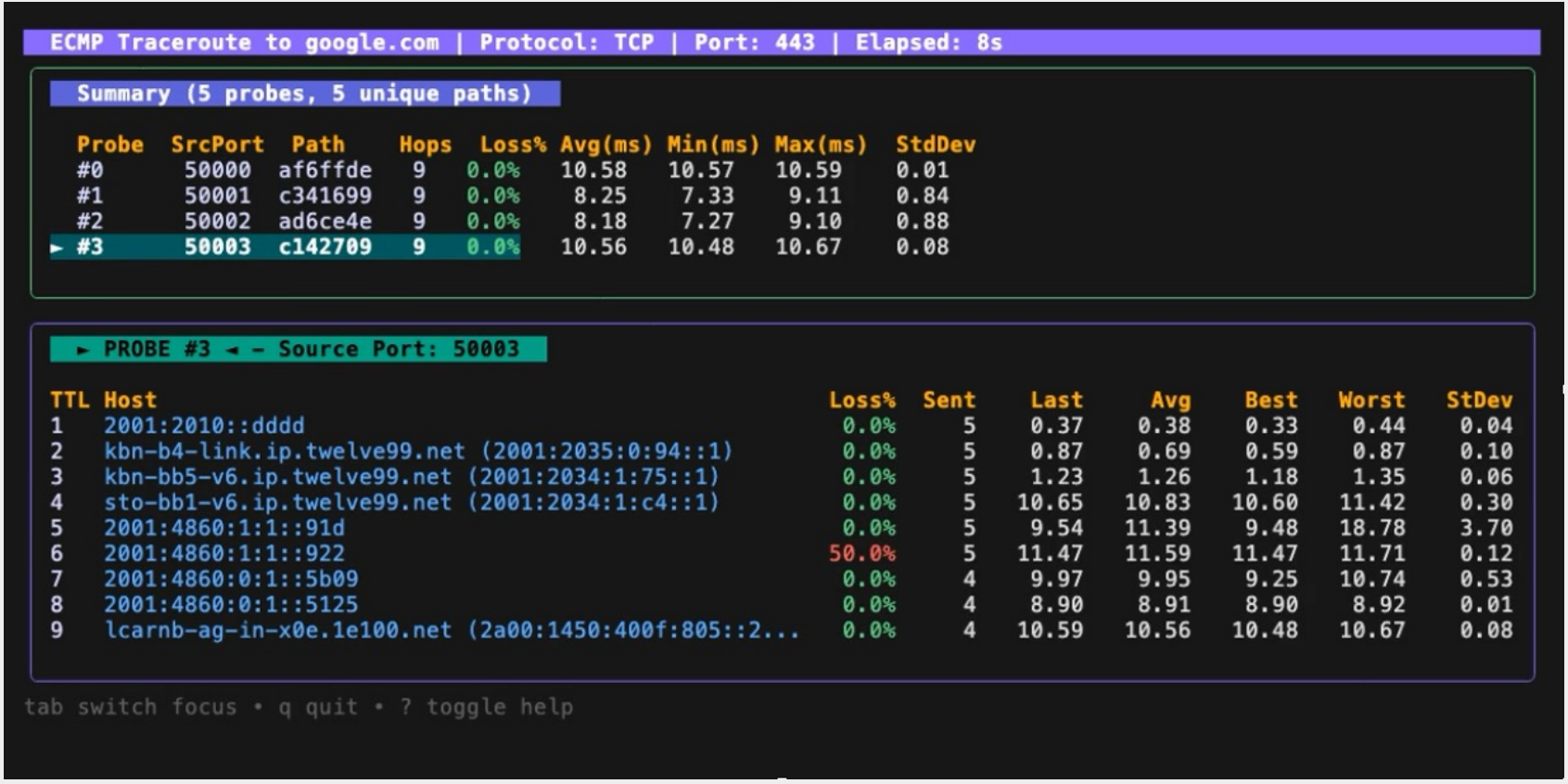
My example: ECMP troubleshooting

- ECMP can be frustrating to troubleshoot during incidents
- Especially if there are many paths and loss is intermittent

Why I needed a tool



Observable output (paths over time)



Define success (before coding)

In my ECMP example, “success” meant:

- Lightweight enough to run during incidents
- Scriptable for automation and dashboards

A method you can reuse

Change one thing at a time

Confidence grows from evidence (not perfection)

- Start in a domain you trust
- Use observation loops you already know (captures, counters, RFCs)
- Build in tiny slices, and let the feedback teach you

Use a loop

Change → measure → explain

One variable at a time

Impostor feelings are normal

- I didn’t feel like a “real software engineer”
- Evidence helped me move anyway
- Small progress, every week

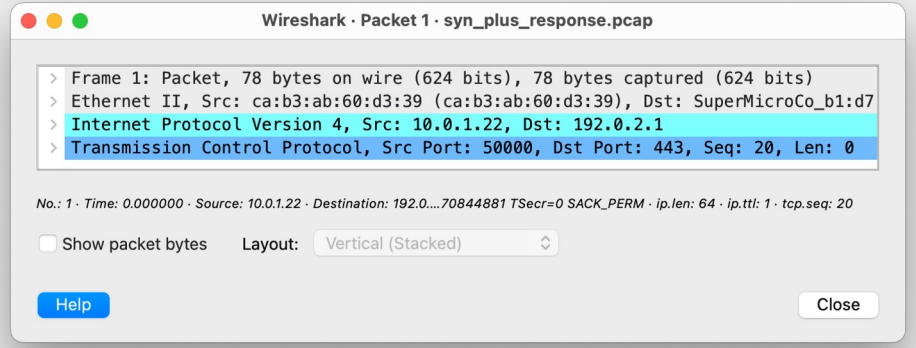
Momentum comes and goes (that’s normal)

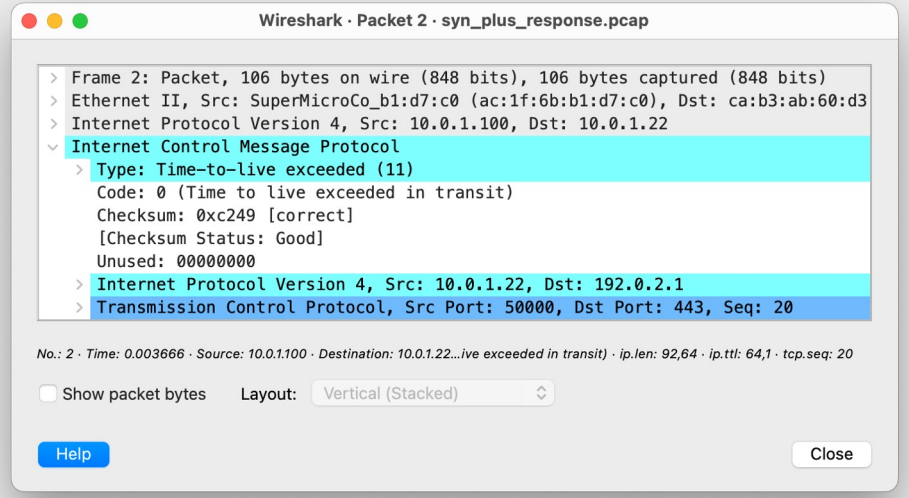
- It took me ~2 years to reach a release
- Life changes (sometimes pain points disappear)
- Make it easy to restart

Start with one packet (control one variable)

First milestone: craft one packet. Validate it in Wireshark.

Control one variable at a time. Familiar tools make the new parts feel safe.





The loop I already trusted

- Observe → measure → explain

- Run traceroute/MTR and collect statistics
- Capture packet dumps for analysis and evidence

Automating the same loop

- Observe → measure → explain

- Run parallel probes and collect statistics (automatically)
- Capture packet dumps for analysis and evidence

- Same loop, just faster and repeatable

Break the problem into smaller parts

Inputs → Probing → Results

Inputs: CLI arguments → settings

Probing: send test packets → receive replies → decode

Results: stats → output (TUI or JSON)

Start where you’re comfortable.

Constraints can keep you moving

- Constraints create direction and keep you moving

- I chose a tight constraint as a fun challenge
- That small win kept me going

ICMP errors echo part of the original packet

- ICMP errors echo the original IP header
- Plus at least 8 protocol bytes (TCP/UDP)

My constraint/puzzle

ICMP only echoes a tiny part of the packet

Where can an ID live?

- Reuse fields the protocol already echoes

TCP first 8 bytes: src port (2) | dst port (2) | sequence (4)

UDP first 8 bytes: src port (2) | dst port (2) | length (2) | checksum (2)

Design decision

- TCP: encode in sequence number
- UDP: encode in length field
- ICMP errors will echo enough to recover TTL + probe

Next bottleneck: UI (get a first draft)

- For me: probing was comfortable; UI wasn’t
- A rough draft helped me move forward

GenAI is part of the toolbox now

- I used GenAI to get a rough UI starting point
- The value was speed to something I could iterate on

How to use it

- GenAI is a highly productive junior; you’re the senior guiding it
- Review the output like you would a production change
- If you can’t explain it, you don’t use it

Keep ownership

- Slow down: verify, test and make it yours
- Use it to learn – not to outsource thinking

Understanding beats speed

- The goal isn’t speed – it’s learning and ownership
- If it doesn’t make sense, don’t use it
- In the end you’re the one accountable

Make work visible so your team can resume

- Break work into small features
- Write down what you learned (for future you)
- Make it easy to resume and share

Replace “I think” with “I measured”

- As network engineers we already do this:
  - Find *why* something doesn’t work and explain it with data
- Apply that same habit to your own learning
- You gain proof that you can learn hard things

Key takeaways

- Start where you’re strong
- Use observation to stay grounded
- Keep scope small, then layer structure
- Momentum comes and goes. That is okay
- Ask for help, then make it yours

Try one small project (then share what you learn)

- Find a pain point in your daily work
- Build one tiny feature
- Share the journey so others can follow

Thanks for listening!

- Happy to answer questions
- Or hear what constraint helped you move forward

- Take-home slides next: a starter plan + templates

My ECMP tool: https://github.com/tkjaer/etr

References

Encoding design notes:

- https://github.com/tkjaer/etr/blob/main/docs/probe-encoding-design.md

ICMP quoting RFCs:

- RFC 792 – ICMPv4 quoting minimum - Time exceeded message
- RFC 1812 – Router guidance (“as much as possible”) §4.3.2.3
- RFC 4443 – ICMPv6 quoting minimum §2.4

Take-home slides

A simple starter plan

- Pick one workflow you repeat often, or one task that has friction
- Define success (what output you want)
- Pick the lowest-friction language (shell, Python, PowerShell) — you can switch later
- Automate the smallest step first
- Timebox it (30–60 minutes) and stop after the first working result
- Test one change at a time (keep notes)

First 60 minutes (a template)

- Pick one real input you already have (a file, a command output, a packet capture)
- Write the smallest program that prints one useful line
- Run it on that real input
- Save the command you ran (so you can repeat it)
- Commit the result as a checkpoint

Version control (Git basics)

- Why: keep a history you can trust and roll back
- Work in branches (a separate line of work)
- Commit small changes (save a snapshot)
- Commit when it works (don’t batch a week of changes)
- Push to a remote (backup + share)

Full disclosure: I work for GitHub.

Collaboration (even solo)

- Why: make work visible and easier to resume later
- Write down small tasks in your Git platform (issues) or a short TODO list
- Write a 3-line README (what it does, how to run it, what “done” means)
- Use pull requests (even when you work alone)
- Use pull requests as a checkpoint: explain what you changed and why

Quality checks

- Why: catch mistakes early, not in production
- Run formatting/lint checks automatically (optional)
- Write a small end-to-end test (run the tool and verify the output)
- Keep one “golden” command you can re-run (same args, same expected output)
- Run tests locally and in CI/automation (repeatable checks)

Small next experiments

- Why: small wins build momentum
- Add a JSON export for one workflow (save results, reuse later, avoid re-runs)
- Build a tiny dashboard or a graph (see trends quickly)
- Write a 5-line script that saves time (reduce daily friction)
- Add one doc page for future you (capture the why)