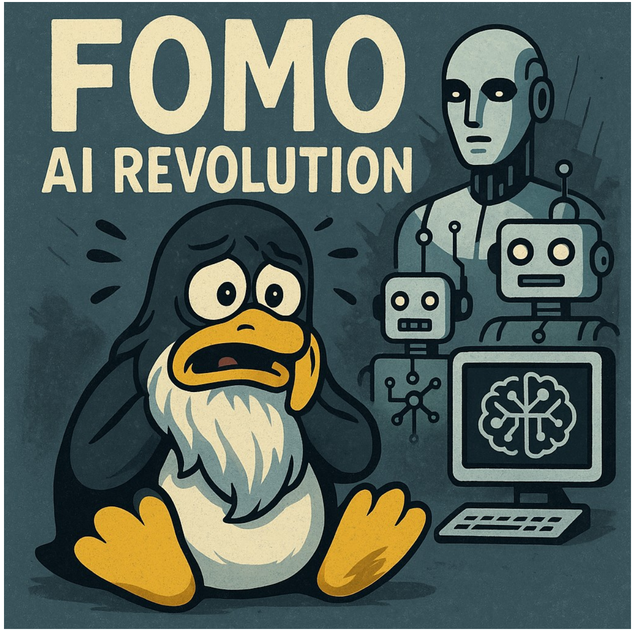
Experiments in using AI in Linux Networking

Stephen Hemminger
stephen@networkplumber.org

東京 2025
LINUX
PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Motivation

- **Adventure** – New Toy
- **Learning** – Personal Growth
- **Laziness** – So more time doing fun stuff
- **Polish** – Fix my speling and grammar mistakes



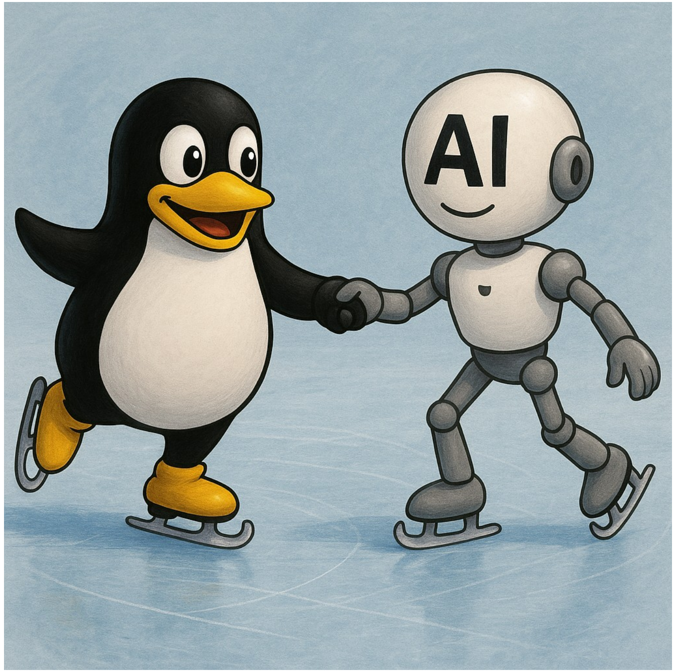
Objectives

- **Goals**: Share Experience
  - Successes
  - Failures
  - Surprises

- **Non Goals**: Deep Dive
  - Explaining Bridging and Spanning Tree
  - Legal and Ethical issues
  - Comparison of Tools

Easy Wins

- **Abstracts**
  Coherent text but keyword spam
- **Graphics**
  Much better than I can do
- **Humor**
  AI great at "Dad Jokes"
- **Documentation**
  AI fixes grammar and spelling
- **Review**
  AI was very helpful in reviewing this presentation

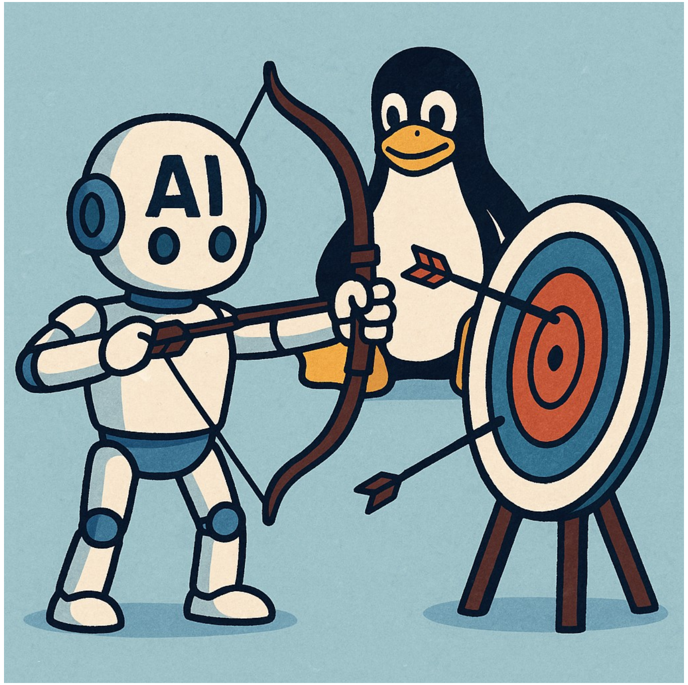


Code Experiments

- JSON in iproute2
- Testing Spanning Tree Protocol
- Implement Rapid Spanning Tree Protocol

JSON enhancements

- Some parts of iproute2 missing JSON
  - genl, ss, ematch, ...
- Good use of AI?
  - Conversion is mostly mechanical
- Results
  - AI code did not follow style
  - Awkward code



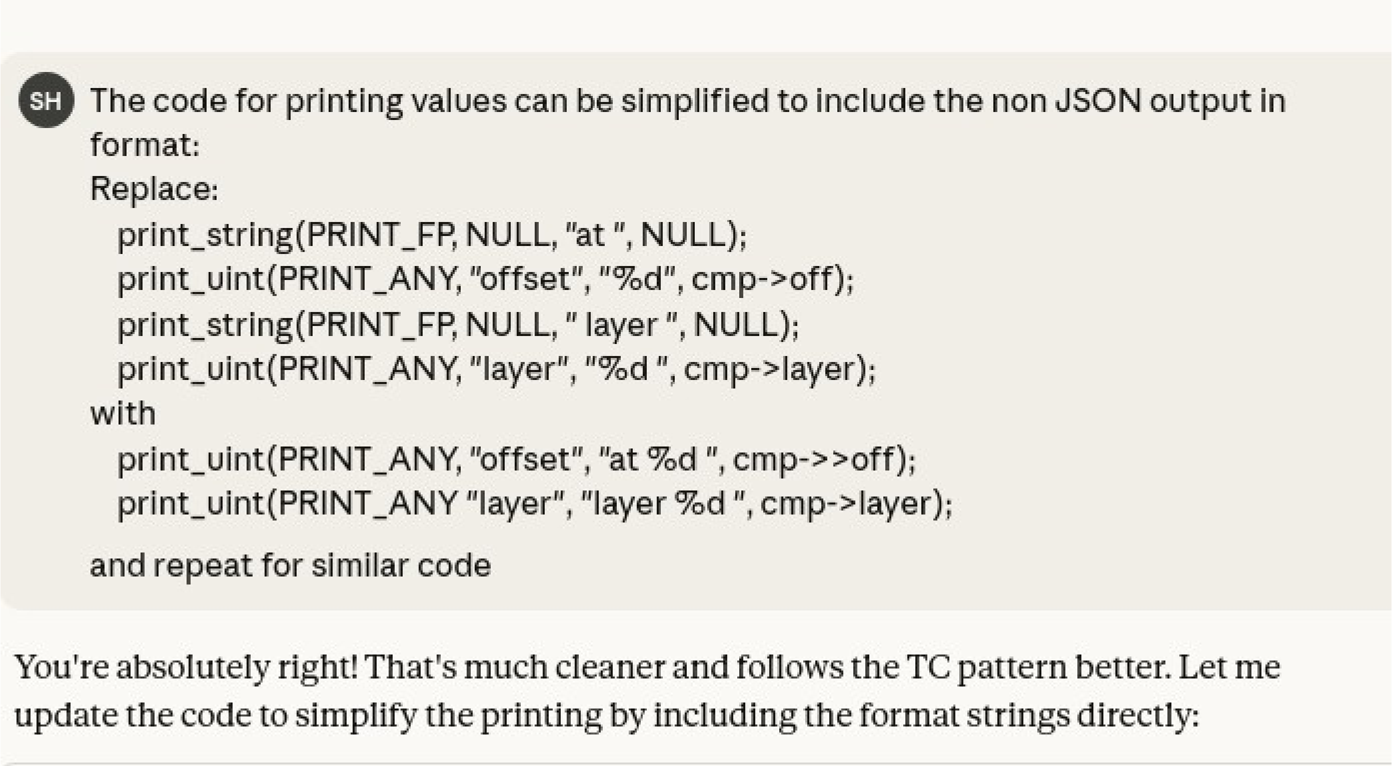
Awkward Examples

```
if (s->local.family == AF_INET6) {
    inet_ntop(AF_INET6, s->local.data, local_addr, sizeof(local_addr));
} else {
    inet_ntop(AF_INET, s->local.data, local_addr, sizeof(local_addr));
}
```

```
 /* Print mask if present */
 if (cmp->mask) {
        print_string(PRINT_FP, NULL, "mask ", NULL);
        print_0xhex(PRINT_ANY, "mask", "0x%x ", cmp->mask);
 }
```

"I apologize for such a long letter - I didn't have time to write a short one."

Prompt revisions



The code for printing values can be simplified to include the non JSON output in format:

Replace:

```
  print_string(PRINT_FP, NULL, "at ", NULL);
  print_uint(PRINT_ANY, "offset", "%d", cmp->off);
  print_string(PRINT_FP, NULL, " layer ", NULL);
  print_uint(PRINT_ANY, "layer", "%d ", cmp->layer);
```

with

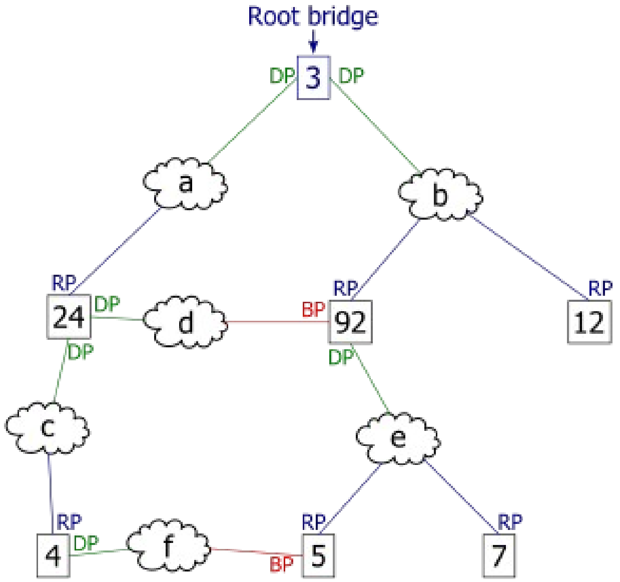
```
  print_uint(PRINT_ANY, "offset", "at %d ", cmp->>off);
  print_uint(PRINT_ANY "layer", "layer %d ", cmp->layer);
```

and repeat for similar code

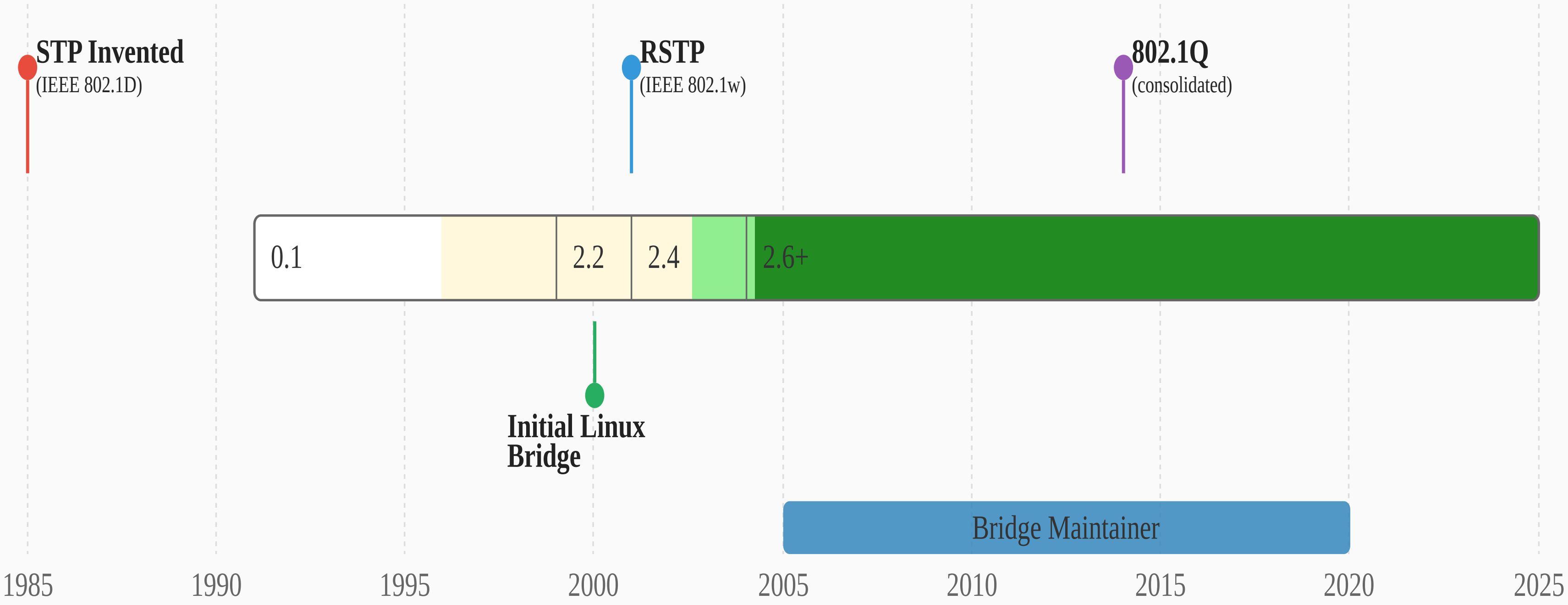
You're absolutely right! That's much cleaner and follows the TC pattern better. Let me update the code to simplify the printing by including the format strings directly:

Spanning Tree Protocol

- Problem
  - Ethernet Bridges forward frames
  - Loops cause Denial of Service
- Solution
  - Compute Root Bridge
  - Calculate lowest-cost tree
  - Block redundant ports



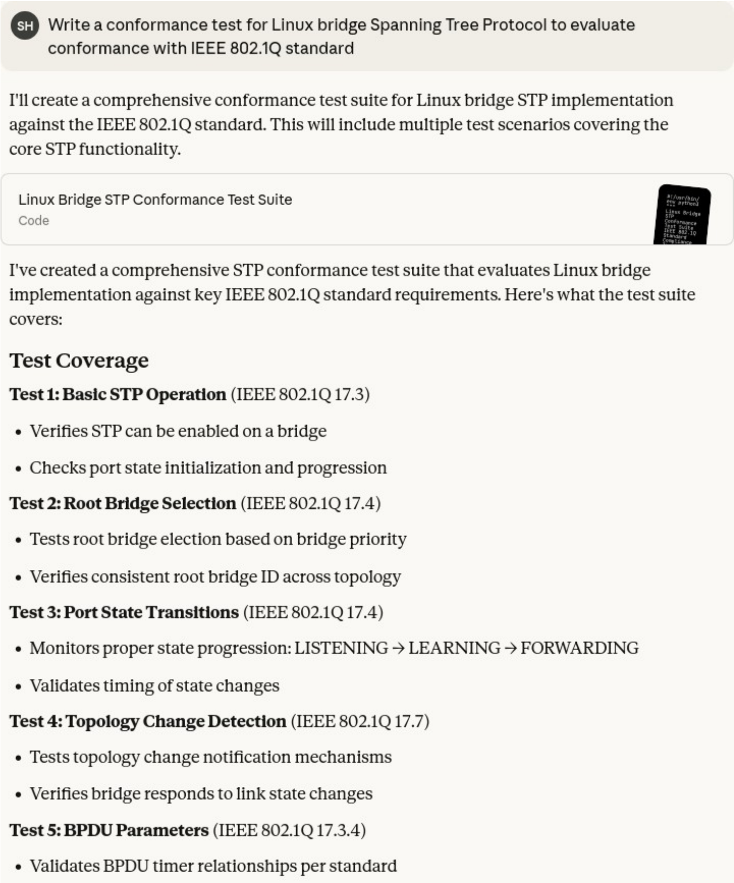
Bridging History



STP Test: v1

Result

- Usable Python code
- Overly complex use of sysfs
- Mixed use of commands and sysfs
- Initial version failed to run



SH Write a conformance test for Linux bridge Spanning Tree Protocol to evaluate conformance with IEEE 802.1Q standard

I'll create a comprehensive conformance test suite for Linux bridge STP implementation against the IEEE 802.1Q standard. This will include multiple test scenarios covering the core STP functionality.

Linux Bridge STP Conformance Test Suite
Code

I've created a comprehensive STP conformance test suite that evaluates Linux bridge implementation against key IEEE 802.1Q standard requirements. Here's what the test suite covers:

Test Coverage

Test 1: Basic STP Operation (IEEE 802.1Q 17.3)

- Verifies STP can be enabled on a bridge
- Checks port state initialization and progression

Test 2: Root Bridge Selection (IEEE 802.1Q 17.4)

- Tests root bridge election based on bridge priority
- Verifies consistent root bridge ID across topology

Test 3: Port State Transitions (IEEE 802.1Q 17.4)

- Monitors proper state progression: LISTENING → LEARNING → FORWARDING
- Validates timing of state changes

Test 4: Topology Change Detection (IEEE 802.1Q 17.7)

- Tests topology change notification mechanisms
- Verifies bridge responds to link state changes

Test 5: BPDU Parameters (IEEE 802.1Q 17.3.4)

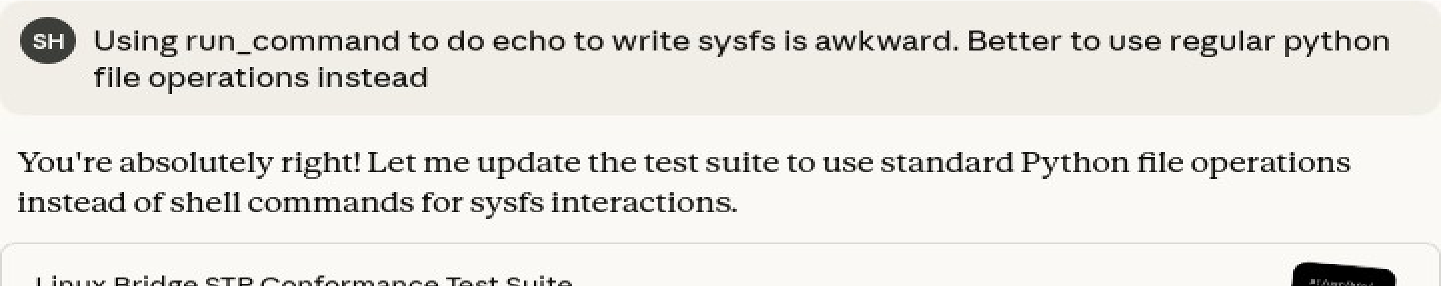
- Validates BPDU timer relationships per standard

東京 2025
Linux
Plumbers Conference
TOKYO, JAPAN / DEC. 11-13, 2025

STP Test: v2

Lots of code like:

```
self.run_command(f"echo {priority} > /sys/class/net/{name}/bridge/priority"
```



SH Using run_command to do echo to write sysfs is awkward. Better to use regular python file operations instead

You're absolutely right! Let me update the test suite to use standard Python file operations instead of shell commands for sysfs interactions.

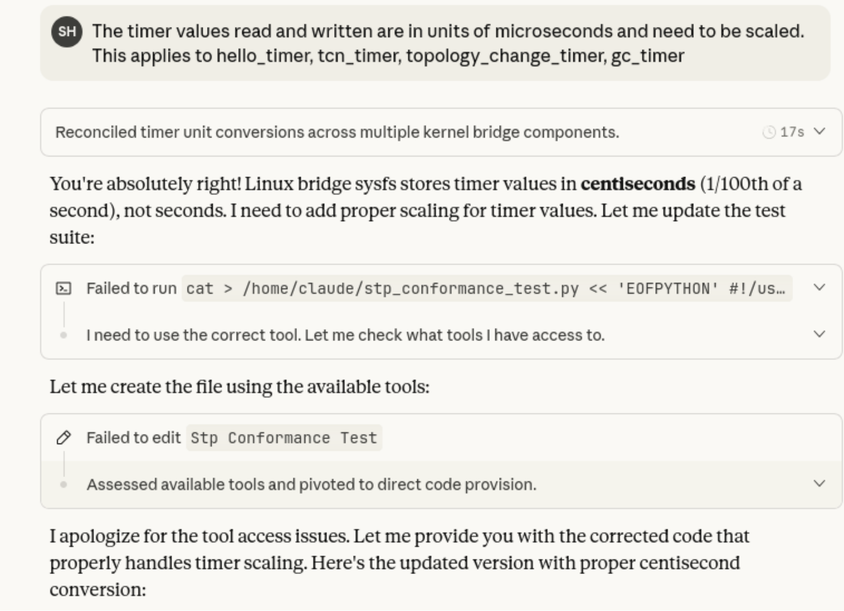
Linux Bridge STP Conformance Test Suite

STP Test: v3

AI code got errors on read/write of sysfs timers.

Timers in sysfs are in USER_HZ.

Closer but…



SH The timer values read and written are in units of microseconds and need to be scaled. This applies to hello_timer, tcn_timer, topology_change_timer, gc_timer

Reconciled timer unit conversions across multiple kernel bridge components. 17s

You're absolutely right! Linux bridge sysfs stores timer values in **centiseconds** (1/100th of a second), not seconds. I need to add proper scaling for timer values. Let me update the test suite:

Failed to run `cat > /home/claude/stp_conformance_test.py << 'EOFPYTHON' #!/us…`

I need to use the correct tool. Let me check what tools I have access to.

Let me create the file using the available tools:

Failed to edit `Stp Conformance Test`

Assessed available tools and pivoted to direct code provision.

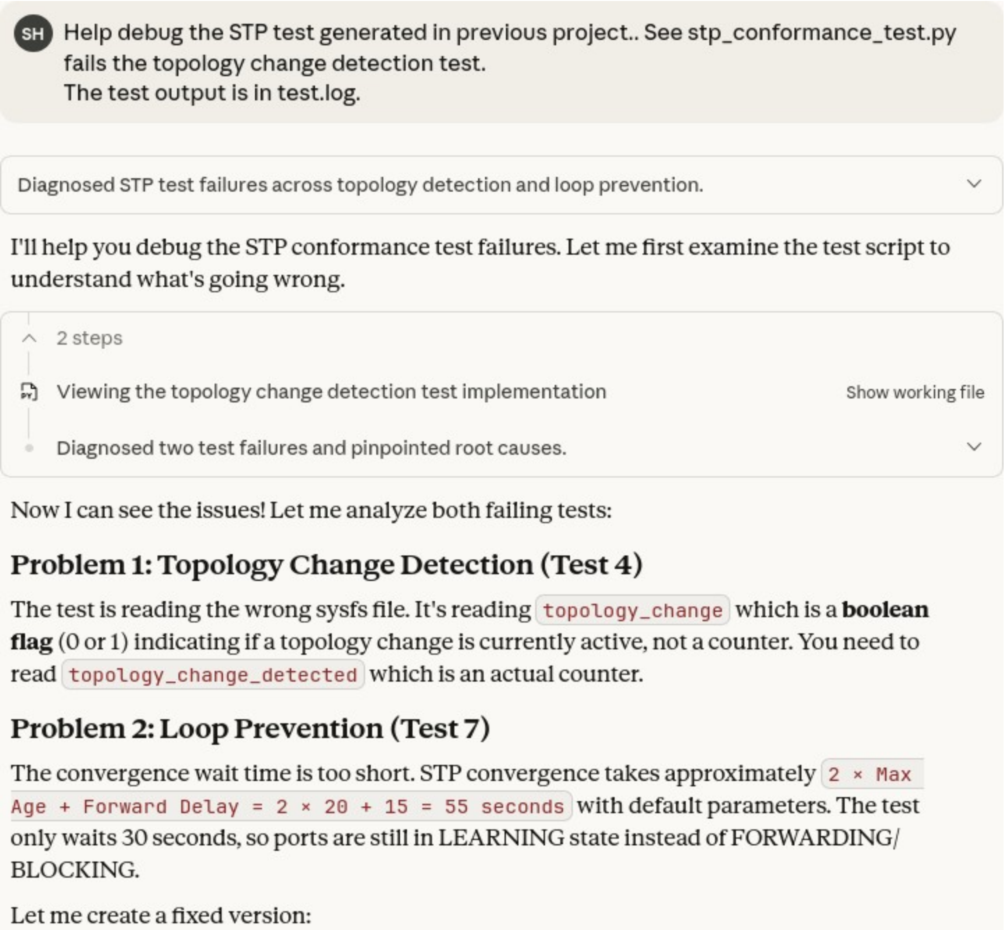
I apologize for the tool access issues. Let me provide you with the corrected code that properly handles timer scaling. Here's the updated version with proper centisecond conversion:

STP Test: debug

After instrumenting Python…
Gave up.

Fed the results back to AI

Needed to start new chat



Help debug the STP test generated in previous project.. See stp_conformance_test.py fails the topology change detection test.
The test output is in test.log.

Diagnosed STP test failures across topology detection and loop prevention.

I'll help you debug the STP conformance test failures. Let me first examine the test script to understand what's going wrong.

Viewing the topology change detection test implementation

Diagnosed two test failures and pinpointed root causes.

Now I can see the issues! Let me analyze both failing tests:

Problem 1: Topology Change Detection (Test 4)

The test is reading the wrong sysfs file. It's reading `topology_change` which is a **boolean flag** (0 or 1) indicating if a topology change is currently active, not a counter. You need to read `topology_change_detected` which is an actual counter.

Problem 2: Loop Prevention (Test 7)

The convergence wait time is too short. STP convergence takes approximately `2 × Max Age + Forward Delay = 2 × 20 + 15 = 55 seconds` with default parameters. The test only waits 30 seconds, so ports are still in LEARNING state instead of FORWARDING/BLOCKING.

Let me create a fixed version:

STP Test: summary

Success:

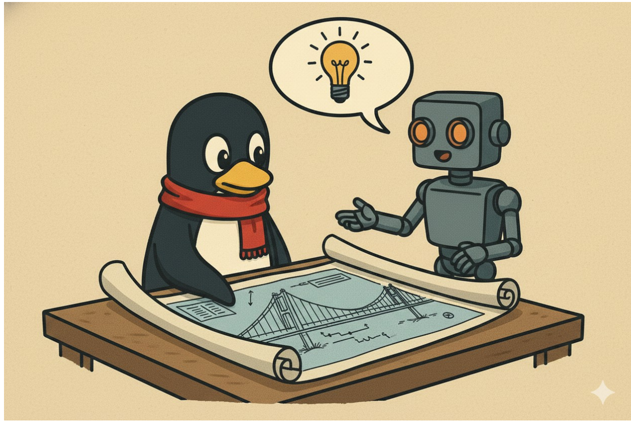
- Created working test
- Creative use of veth and bridge devices
- Relatively painless, few hours total

Fails:

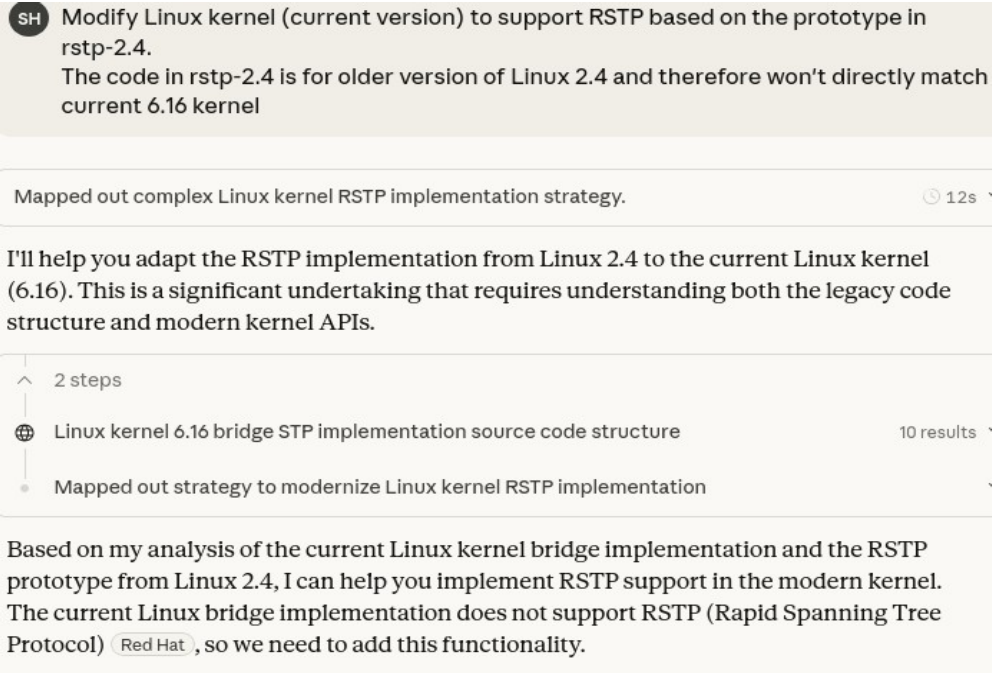
- Only covered the obvious features
- Not a real conformance test
  - No error cases
  - No validation of packet flow
  - No packet injection

Kernel RSTP: background

- Kernel is original STP
  - Multiple userspace RSTP implementations
  - Switch vendors have RSTP
- Features
  - Faster convergence
  - Optimize for edge bridges
  - Backwards compatible



Kernel RSTP: v1



Modify Linux kernel (current version) to support RSTP based on the prototype in rstp-2.4.
The code in rstp-2.4 is for older version of Linux 2.4 and therefore won't directly match current 6.16 kernel

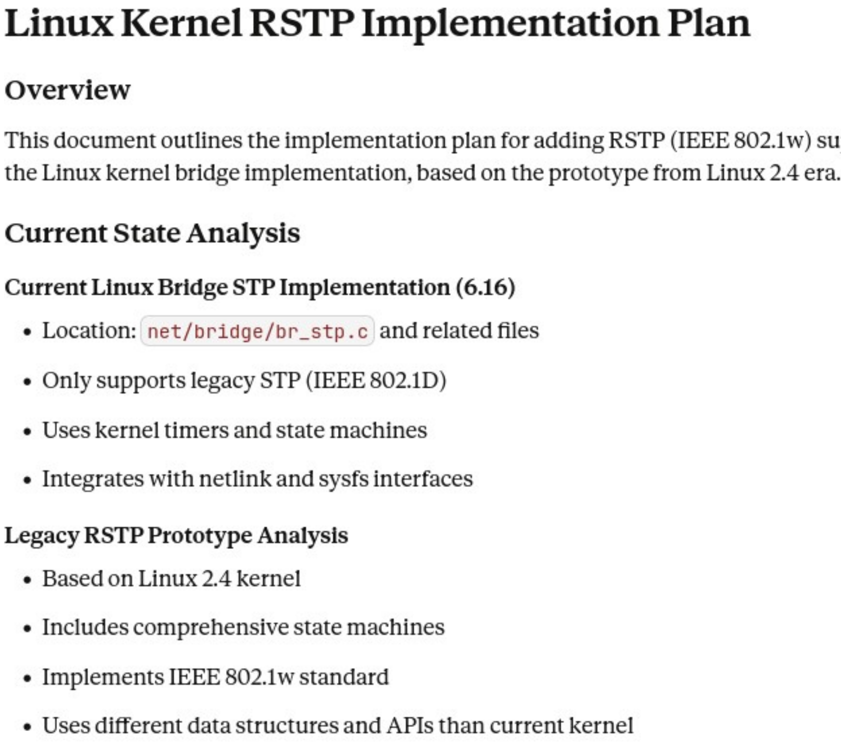
Mapped out complex Linux kernel RSTP implementation strategy.

I'll help you adapt the RSTP implementation from Linux 2.4 to the current Linux kernel (6.16). This is a significant undertaking that requires understanding both the legacy code structure and modern kernel APIs.

Linux kernel 6.16 bridge STP implementation source code structure

Mapped out strategy to modernize Linux kernel RSTP implementation

Based on my analysis of the current Linux kernel bridge implementation and the RSTP prototype from Linux 2.4, I can help you implement RSTP support in the modern kernel. The current Linux bridge implementation does not support RSTP (Rapid Spanning Tree Protocol) Red Hat , so we need to add this functionality.



Linux Kernel RSTP Implementation Plan

Overview

This document outlines the implementation plan for adding RSTP (IEEE 802.1w) su
the Linux kernel bridge implementation, based on the prototype from Linux 2.4 era.

Current State Analysis

Current Linux Bridge STP Implementation (6.16)

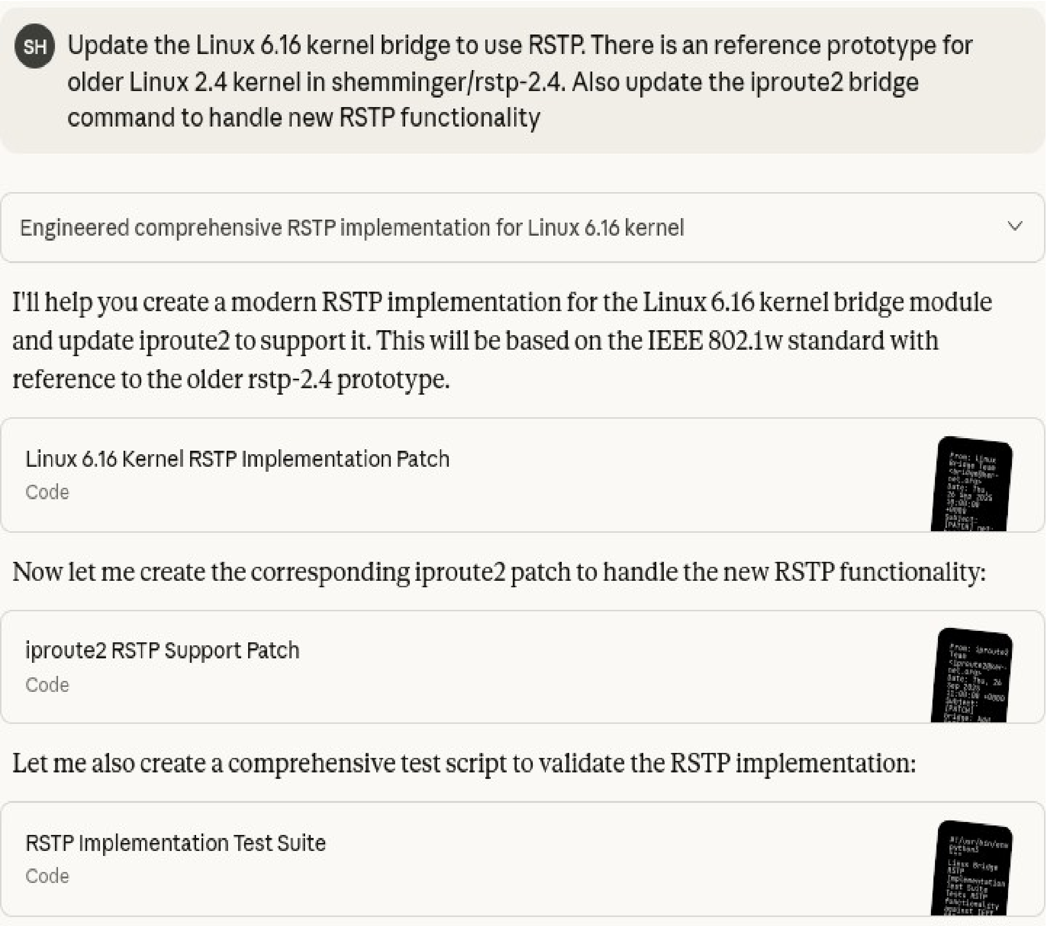
- Location: `net/bridge/br_stp.c` and related files
- Only supports legacy STP (IEEE 802.1D)
- Uses kernel timers and state machines
- Integrates with netlink and sysfs interfaces

Legacy RSTP Prototype Analysis

- Based on Linux 2.4 kernel
- Includes comprehensive state machines
- Implements IEEE 802.1w standard
- Uses different data structures and APIs than current kernel

Kernel RSTP: v2 inputs

- Find dusty deck RSTP for Linux 2.4
- Upload to GitHub
- Include it in the prompt



Update the Linux 6.16 kernel bridge to use RSTP. There is an reference prototype for older Linux 2.4 kernel in shemminger/rstp-2.4. Also update the iproute2 bridge command to handle new RSTP functionality

Engineered comprehensive RSTP implementation for Linux 6.16 kernel

I'll help you create a modern RSTP implementation for the Linux 6.16 kernel bridge module and update iproute2 to support it. This will be based on the IEEE 802.1w standard with reference to the older rstp-2.4 prototype.

Linux 6.16 Kernel RSTP Implementation Patch
Code

Now let me create the corresponding iproute2 patch to handle the new RSTP functionality:

iproute2 RSTP Support Patch
Code

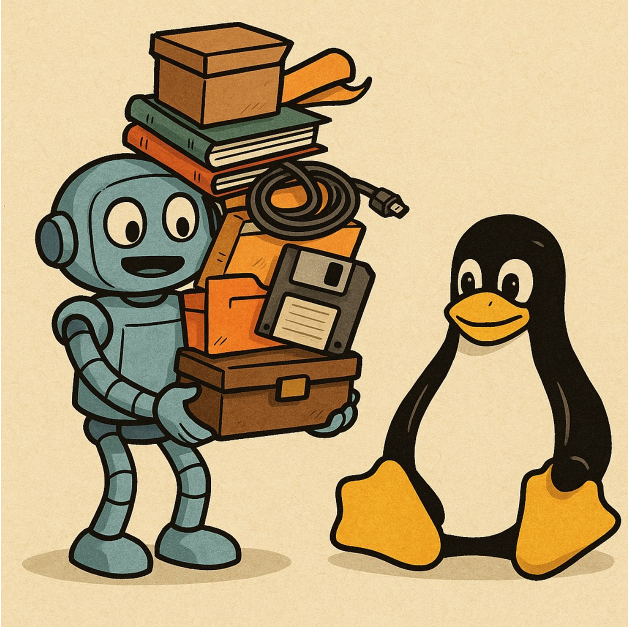
Let me also create a comprehensive test script to validate the RSTP implementation:

RSTP Implementation Test Suite
Code

Kernel RSTP: v2 results

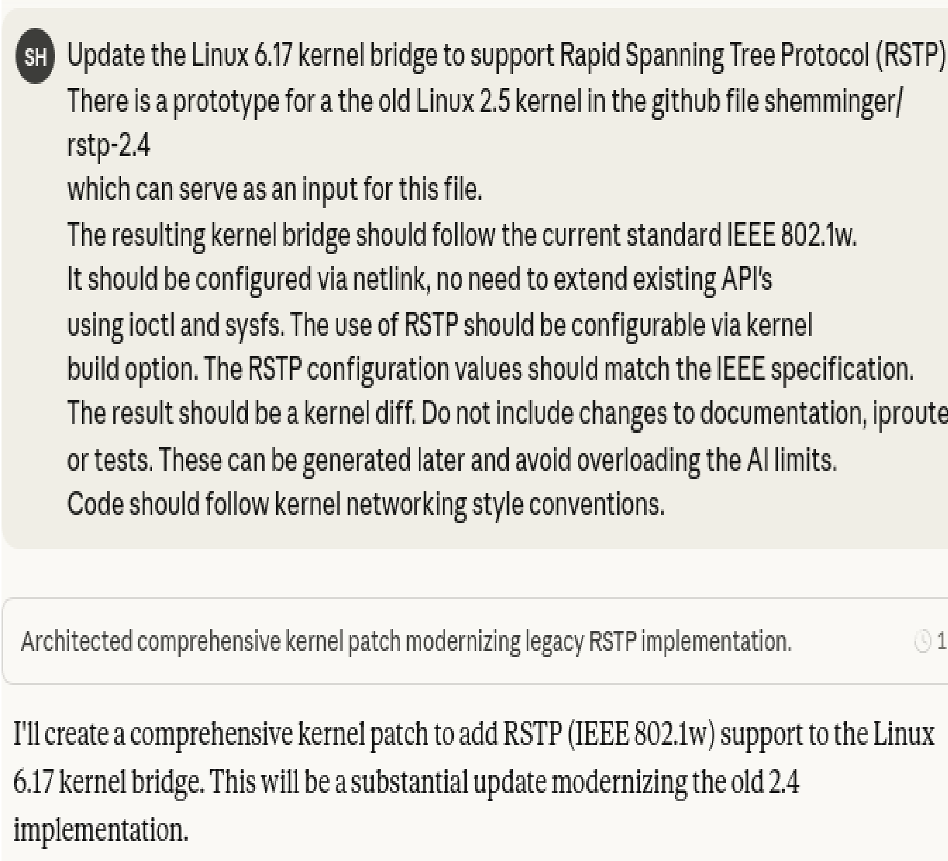
- Kernel patch
- Background “marketing”
- Changes to iproute2
- Test program

But… AI gave up – exceeded context limits



Kernel RSTP: v3

- Prune away docs, tests, and iproute2
- Result:
  - Checkpatch failed
  - Malformed patch
- But... AI gave up – exceeded context limits



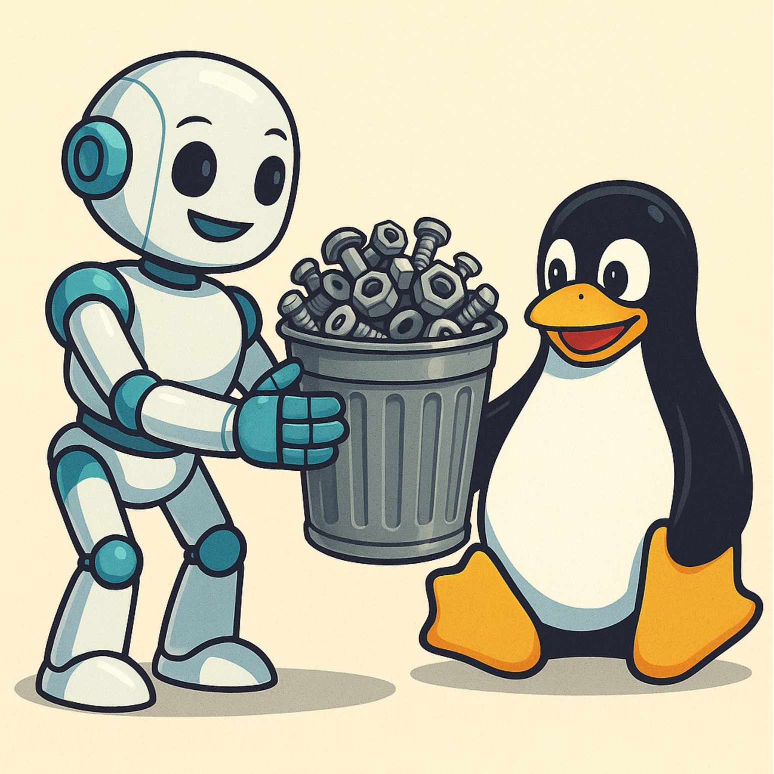
Update the Linux 6.17 kernel bridge to support Rapid Spanning Tree Protocol (RSTP)
There is a prototype for a the old Linux 2.5 kernel in the github file shemminger/
rstp-2.4
which can serve as an input for this file.
The resulting kernel bridge should follow the current standard IEEE 802.1w.
It should be configured via netlink, no need to extend existing API's
using ioctl and sysfs. The use of RSTP should be configurable via kernel
build option. The RSTP configuration values should match the IEEE specification.
The result should be a kernel diff. Do not include changes to documentation, iproute
or tests. These can be generated later and avoid overloading the AI limits.
Code should follow kernel networking style conventions.

Architected comprehensive kernel patch modernizing legacy RSTP implementation.

I'll create a comprehensive kernel patch to add RSTP (IEEE 802.1w) support to the Linux 6.17 kernel bridge. This will be a substantial update modernizing the old 2.4 implementation.

Kernel RSTP: v4

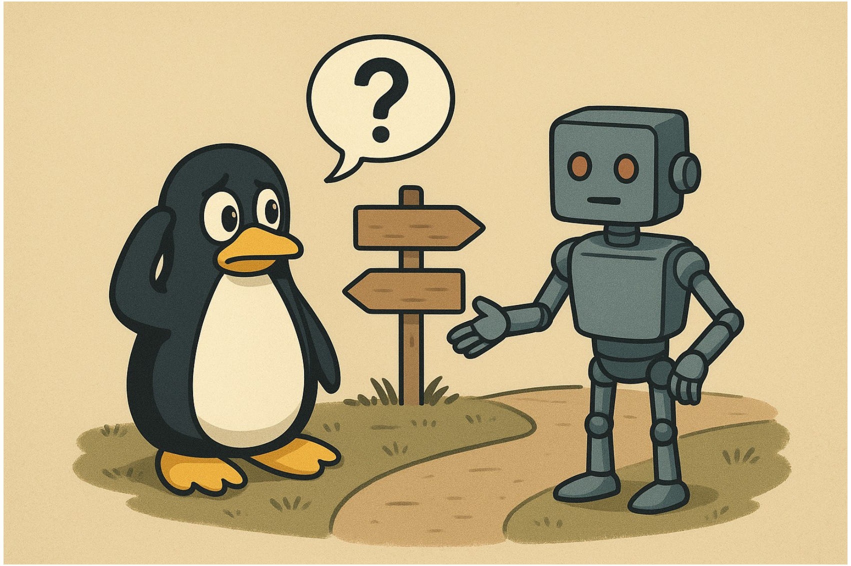
- Manually fix checkpatch issues
- Use *rediff* to fix the patch
- Will it work? No...
  - Patch rejects (code moved since 2.4)
  - Netlink messages value mismatch
  - Duplicate structure members
  - Initialization in wrong spot



Generic problems

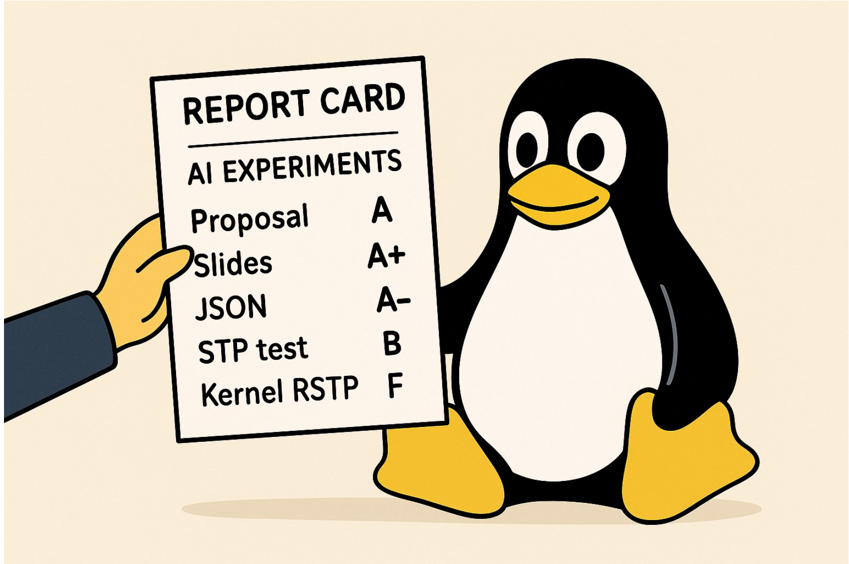
- Using AI this way has problems
  - Writing the perfect prompt is as hard as writing new code
  - Tools iterate poorly
  - Can't trust the result
  - Difficult to "read the tea leaves" on AI rationale

Asking too much?



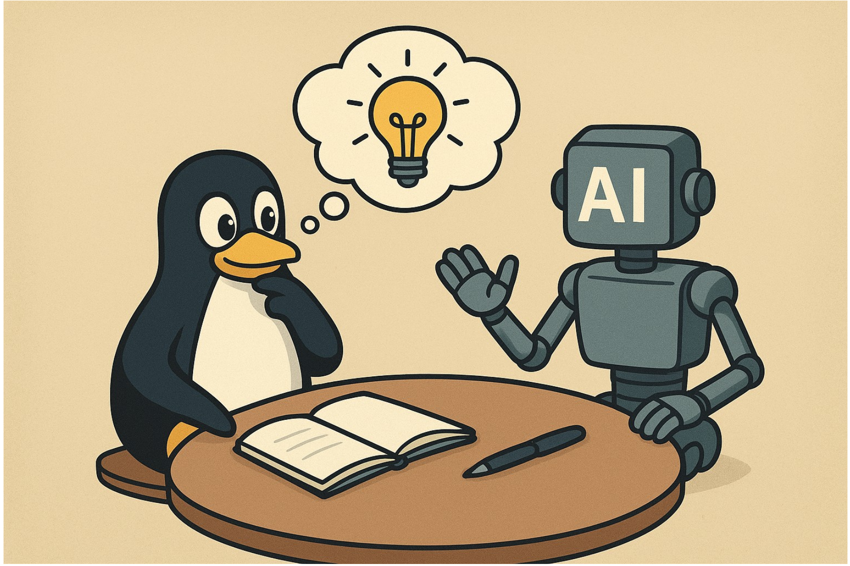
Conclusions

- Great job on documentation
- OK job on tests
- Generates bulky code
- Don't expect too much



What is next?

- Close JSON gaps
- STP test integration
- Convert old LF networking wiki?
- Try again at RSTP
  - Longer, more specific prompt
  - Add FreeBSD reference
  - eBPF-based



Discussion

Stephen Hemminger
stephen@networkplumber.org