

CSCI 1675

Designing High Performance Network Systems

Fall 2026
Akshay Narayan
CIT 368 TuTh 13:00 - 14:20
<https://cs.brown.edu/courses/csci1675/>

About me: Akshay Narayan

- Third year at Brown
- Postdoc at [Berkeley](#), PhD from [MIT](#), Undergrad from [Berkeley](#)
- Focus throughout: systems and networking
- Recent work on eBPF, Jupyter Notebooks, Cloud Microservice Applications, LLM training, etc.

Today's class session

- What is this class about?
- Administrivia

Class Goals

- Understand performance for network systems
 - How do we measure performance?
 - How does system design affect performance?

What is the class **not** about?

- Non-network systems' performance (consider 1952Y)
- Learning about network protocols
- Interface between applications and hardware
- Building applications that use the network

These topics are addressed in this class's prerequisites, **CS 1680 / CS 1670 / CS 1380**

(We may cover some topics you have previously seen in greater depth)

~10 minute Discussion:

1. Why care about performance?
2. If system A has “better performance” than system B, what does that mean?

(most lectures will involve student discussion)

Meme 1: Specialization



Dave Patterson (UC Berkeley/Google)
(Turing award winner):

- Scenario with 100M users speaking to phones 3 minutes per day: If only CPUs, double whole data center fleet!
- Goal: **Custom** Domain Specific Architecture (DSA) to reduce the Total Cost of Ownership (TCO) of DNN inference phase by 10X

One meta-strategy to improve performance:
specialization

Meme 2: Premature Optimization



Don Knuth (Stanford), 1974 Turing Award lecture:

The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times; **premature optimization is the root of all evil** (or at least most of it) in programming.

Trying to improve performance
might introduce bugs

Counterpoint: Costs of specialization

Can we optimize without specializing?

A plea for generality in research systems

Aurojit Panda (NYU), HotEthics Workshop 2024

- Building specialized hardware clusters takes a lot of resources, and they can only do one thing
- So, not many entities can afford to invest in specialized deployments
- Ultra-fast performance within a specific deployment, or decent performance across a wide range of deployments?

Questions to keep in mind

- When is improving performance worth the costs?
 - What type of costs are they (societal? complexity?)?
 - How can we measure those costs?
- This class: It's important to understand performance, so we know not just **how** but **when** to improve it

Class Topics

- Module 1: Measuring performance
 - ~September
- Module 2: IO
 - ~Early October
- Module 3: Concurrency and Scheduling
 - ~Late October
- Module 4: Cloud Computing
 - ~November

Administrivia

HTA: Ting He

- 2nd year Master's student
- Research on extensible operating systems
- Office hours:
 - Tuesday, Thursday 16:00 - 17:00 (most weeks)
 - CIT 201

UTA: Ethan Park

- 5th year Master's student
- Research on measuring service mesh overheads
- Office hours:
 - Monday 16:30-17:30, Wednesday 13:30-14:30 (most weeks)
 - CIT 201

Enrollment

- Capacity won't increase (you just met the entire staff).
- In past offerings, everyone who wanted a spot got one
- If you are going to drop, please do so sooner rather than later as a courtesy to people on the waitlist
- We will resolve the waitlist on a rolling basis - please submit an override request.

Class Components + Grading

- This class has two major components:
 - Reading + writing **homeworks (35%)**
 - Implementation + evaluation **projects (65%)**
 - Projects 0 and 1: focus on network APIs
 - Project 2: focus on microservices

Reading + Writing Homework Assignments

- Goals: Get practice reading technical papers about content from lectures, and writing technical responses.
- We will read ~6 research papers across 4 homework assignments
- HW0 out today: <https://cs.brown.edu/courses/csci1675/fall2026/hw/hw0.html>
- **Due 17 Sept. (before the end of shopping period)**

Please don't use AI tools (LLMs) to write homework responses. The whole point is practicing your own technical writing skills.

Projects

- Goals: Learn evaluation techniques. Gain experience with modern network APIs and tools.
- Project 0 will build and evaluate a synthetic work app
- Project 1 will build and evaluate a key-value store
- Project 2 will build and evaluate a microservice application
 - Project 2 only: optionally in groups of up to 2
- Projects 0 and 1 will be implemented in **Rust**, Project 2 any language

Projects

- Implementation:
 - Write a performance-oriented system feature
 - Mostly focused on IO (network systems)
- Evaluation:
 - Measure and interpret your system's performance
 - Write a **report (primary deliverable)** explaining your system's performance using instrumentation data.
 - Run on course VMs to get consistent results

Projects

- Different than projects in many other classes. Focus on **understanding behavior** over implementation correctness. You are responsible for testing and demonstrating your implementation's correctness.
- Grading format: “research meeting”
 - Meet with course staff to briefly present your report
 - Questions about the performance trends you observed
 - Answers using quantitative evidence from your report
- **Regrade policy (no late days):**
 - Fix any problems by the next grading meeting to regain points

LLM Use in Projects

Project evaluation will focus on your understanding of the report and code you submit.

- Using LLM tools is not strictly banned, but we encourage writing project code by hand to ensure you understand it.

Course Feedback about 1675 Projects

This course is more difficult than you expect. Having taken project heavy courses, I thought this course would take a lot less time.

Making sure your metrics meet the requirements can be difficult if you take the wrong approach, and the assignments will not tell you what approach to take.

The class's smallest focus is on the implementation, and the majority of it is on the data/graphs you produce, how you analyze them, and how you justify them.

Project Implementation Flexibility

- Project 0 will build and evaluate a synthetic work app
 - No stencil / starter code (mandatory Rust crate + runner script)
- Project 1 will build and evaluate a key-value store
 - Client-only starter code, server from scratch
- Project 2 will build and evaluate a microservice application
 - No stencil / starter code (use standard microservice tools)
 - Design flexibility: Propose and build an application of your choice

Why Rust?

- Modern systems programming language
 - Great tooling and documentation, easy to work with
 - Compiler can detect memory safety bugs
 - If you're familiar with C/C++, Rust is easier
 - If you're familiar with Golang, its runtime bakes-in features that project 1 will explore implementing

Rust Lifetimes

Original C version (real bug from my code, 2022)

```
int make_match_rule(struct rte_flow_item *pattern_out, u16 dst_port) {  
    struct rte_flow_item_udp udp_flow =  
        { .hdr.dst_port = RTE_BE16(dst_port) };  
    struct rte_flow_item patterns = {  
        { .type = RTE_FLOW_ITEM_TYPE_UDP,  
          .mask = &udp_dst_port_mask,  
          .spec = &udp_flow, .last = NULL } };  
    memcpy(pattern_out, &patterns, sizeof(struct rte_flow_item));  
    return 0;  
}
```

Bug: DPDK flow steering isn't working.

Rust Lifetimes

Rust version

```
fn make_match_rule(pattern_out: &mut[rte_flow_item],
    dst_port: u16) -> i32 {
    let udp_flow = rte_flow_item_udp { dst_port };
    let patterns = [ rte_flow_item{ udp_flow: &udp_flow } ];
    pattern_out.copy_from_slice(&patterns);
    return 0;
}
```

Rust Lifetimes

```
error[E0597]: `udp_flow` does not live long enough
  --> src/lib.rs:11:45
    |
10 |   let udp_flow = rte_flow_item_udp { dst_port: 42 };
    |           ----- binding `udp_flow` declared here
11 |   let patterns = [ rte_flow_item{ udp_flow: &udp_flow } ];
    |                                           ^^^^^^^^^^^
    |                                           |
    |                                   borrowed value does not live long enough
    |                                   this usage requires that `udp_flow` is borrowed for `'static`
...
14 | }
    | - `udp_flow` dropped here while still borrowed
```

Learning Rust

- Brown's Rust book: <https://rust-book.cs.brown.edu/>
 - Chapter 4, Lifetimes: <https://rust-book.cs.brown.edu/ch04-00-understanding-ownership.html>
- “Traditional” book, TRPL: <https://doc.rust-lang.org/stable/book/>
- Rust by Example: <https://doc.rust-lang.org/stable/rust-by-example/>
- std library documentation: <https://doc.rust-lang.org/std/index.html>

Action Items

- Submit an override request if not registered
- Start working on HW 0
- Project 0 will be released soon
- Take some time to practice with Rust if needed