EE/CSE 461
Introduction to Computer Communication Networks
Winter 1998

Instructor: Tom Anderson (tom@cs)
Office: 316 Sieg Hall
Office Hours: 11-12 Wednesday and 1-2 Friday
Lectures: Wednesday and Friday, 3:30-5:00, 201 Lowe
Home Page: http://www.washington.edu/education/courses/461/CurrentQtr/

Teaching Assistant: Andy Collins (acollins@cs), office hours to be announced. Andy will lead a highly recommended but officially optional discussion section to discuss problem sets and the project. Location and time of the section is to be announced. The first section meets next week and will cover the Java primer.

Homework Grader: James Suruda (jsuruda@sanjuan.cs).

Course Description: The purpose of this course is to teach the design of networked systems, from hardware through protocols to applications.

Course Prerequisites: Regardless of what the course catalogue says, CSE/ENGR 142 and CSE 143 are **required** along with a familiarity with C or C++. CSE 326 or 373 (data structures) is recommended. There is a completely new Java-based project in this course, requiring significant programming. We do not assume any knowledge of Java, but if you are not comfortable with C or C++ programming, you are better off taking the course some other quarter. Also, since the project is new and I'm new teaching this course, we will need your patience with any glitches that might occur (and there will be some); if you want a polished version of the course, again, you should take the course some other quarter.

Enrollment: A signup sheet will be circulated for those wanting to take the course but who were not able to enroll. We will overload a small number of students from this list, with names posted on my office door by Friday noon. No one will be added after Friday, regardless of later drops.

Grading: It is important for you to realize what grades in this class reflect, and what they **don't** reflect. All we can grade you on is how well you demonstrate that you know the material this quarter. But we can't grade you on how much of a success you'll be after graduation, how smart/creative/persistent/self-motivated you are, or even how well you'll be able to apply the material in the future. And after all, it's what you do in the future, not what you do this semester, that's really important.

That said, grades will be determined roughly as follows: Final exam: 40%, midterm: 20%, problem sets: 10% and the project: 30%. Input from the TA will be used to help determine the grades of students near the borderline.

In the past, TAs, graders, and I have spent a lot of time dealing with regrading appeals, time that would be better spent helping students learn the material. Absolutely come to us if we made an arithmetic error, but realize that a few points here and there are extremely unlikely to make any difference in your final grade. To discourage appeals, we will apply the following rule: if you ask us to regrade something, we will regrade the **entire** exam, problem set, or project assignment, and we will deduct points if we find we were overly generous. Thus, your grade will be as likely to go **down** as up.

The course will be graded on a curve. However, extra credit points will not be included in establishing the curve; they will be applied afterwards, so as to have no effect on students not doing the extra credit problems. I grade on a curve rather than on an absolute scale because it protects students from stressing out if I happen to give an overly hard midterm or final. The downside of grading on a curve is that it tends to lead students to think they are competing against each other; in practice, the largest impact any individual student's performance is likely to have on your grade in a class this size is less than 0.2% -- in other words, well into the noise.

Also, I have found (in other project classes) that students often stress about whether they will be penalized because their project may not be as elaborate as that of some other group in the class; again, this is a myth. The project is hard enough without looking over your shoulder at other students. I expect most students will do quite well on the project -- the consequence is that the project has less effect than you might think on the curve. A warning, however: if you punt the project, however, you will fail the course.

I believe students in this class can learn as much from each other as they do from lectures and sections. In order to to encourage a spirit of cooperation instead of competition among students, I will give you an opportunity to tell us about those students who have helped you the most in understanding the material in the course. This information will be turned into extra credit points for the helper which will be applied after determining the curve; in other words, you can only improve your grade by helping out your classmates.

Cheating: Note that cooperation is not the same as cheating. It's OK to ask someone about the concepts needed to do homework or project assignments (provided you give them credit -- see above); however, copying other people's code or solution sets is strictly prohibited. The homework and project assignments must be the work of the students turning them in -- we will punish transgressors severely. A simple procedure to keep yourself on the right side of the line is that after a student (or the TA!) helps you with an assignment, go watch Gilligan's Island for a half hour, then go back to working on the assignment. If you can't remember what the person said after a half hour, you didn't really understand it.

Midterm and Final: There will be a single midterm. Approximate dates and times for the midterm and final are listed below. Both exams will be **closed book**, and will cover material from lecture, sections, the readings, and the project. In particular, you are likely to do poorly on the exams and in the course if you sleaze out of the project by making your project partners do all the work.

Project and Sections: The project in this course is to build a cache-coherent version of the Internet web, from ground up (e.g., starting with unreliable packets on a local Ethernet). I believe the project is the core of this course, and that most of what you learn will be from doing the project.

If you plan to do software or hardware development after graduation, you will almost certainly need to know how to work in a group. Recent CS grads almost all say that the ability to work in groups was the single most important thing they wished they had learned in college. For this project, you will need to form into groups of 2-3; the assignments will be the same no matter what size group you have. I will not permit anyone to do the project by themselves.

The TA for your section will grade all phases of your project. The TA's have been instructed to grade in part on design and implementation style and to be increasingly strict about this as the semester proceeds. In other words, it is **not** enough to get a working solution; you must implement the solution in a clean way that would simplify making further enhancements. (Many employers have said that new graduates don't know how to program well -- it will really benefit you in the long run to work on your software engineering skills.)

For each project phase, your TA will meet with your group for 10 minutes the week **before** the phase is due, to conduct a **graded design review**. These reviews serve several purposes: (i) to encourage you to get an early start on the assignments (a key to success in this course), (ii) to catch design errors early, **before** you spend a lot of time debugging, and (iii) to give you is chance to explain and defend your approach (this is an important skill to learn as engineers).

The project will be implemented in Java. For those of you who don't know Java, don't panic -- we will give you a handout that describes what you need to know. Beyond its hype value, learning to program in Java is incredibly useful even if you eventually do most of your programming in C++. C++ is powerful and convenient, but it is also very easy to misuse -- there are lots of features in C++ that might occasionally be useful, but more often do more harm than good (see for example, Meyers book, "Effective C++). Java is essentially a subset of C++, carefully designed to enforce what I would call "structured object-oriented programming". Those very familiar with C++ may find the subset constraining, but it preserves many of the advantages of C++ while eliminating most of its complexity.

We will use flexible slip dates for the project, and documentation will be due the day after you turn in the project -- the goal is to encourage you write the documentation after having had a good night's sleep! Further details about the project will be covered in later handouts.

Problem Sets: I will assign a bunch of short written problem sets, at a rate of roughly one per week. Unlike the project, the problem sets are to be done **individually**. The due dates for the problem sets will be announced during the semester.

Textbook: Peterson and Davie, "*Computer Networks: A Systems Approach*." This textbook is extremely well-written and easy to read. It has one major flaw and a few omissions. The flaw (one shared with almost all other networking textbooks) is that it is organized around climbing up a protocol stack; since many issues can and must be addressed at multiple levels, this organization leads it to discuss the same topics in multiple chapters. I've organized the lectures thematically, to avoid repeating myself as much as possible; this means the lectures jump around in the textbook. (Another way of viewing the course lectures is that they attack the protocol stack starting in the middle, then going down, followed by up.) Among the omissions are (i) quantitative analysis of network performance, (ii) comparison of computer networks with telephone networks, (iii) comparison of the Internet and local area networks (LANs) with "machine-area" multiprocessor interconnection networks, and (v) the design of distributed applications on computer networks. We will attempt to cover these issues in lecture and in the project.

We also recommend that you buy and read Maguire, "*Writing Solid Code*". Every undergraduate CS and EE major should read this book before graduation. By reading it now, you'll find you'll save lots of time debugging later in the quarter (and for the rest of your professional life). Getting any complex piece of software to work for real requires taking every step available to find bugs as early in the development process as possible. My own experience is that many students have come to me with intractable bugs that they've spent days trying to find, where the bug could have been avoided (or found much more quickly) by following Maguire.

You may also find it convenient to have a copy of "*The Java Programming Language*" by Arnold and Gosling.

Course Outline and Approximate Dates

Jan. 6 - 8: Introduction: pipes, protocols, layering, the end-to-end arg, and how the web works

Peterson Ch. 1

Phase 1: From unreliable Internet messages to application-to-application communication

Jan. 13 - 15: Reliable delivery: stop and wait, sliding window, selective repeat, RTT est.
Peterson 3.5, 6.2

Jan. 20 - 22: End-to-end: fragmentation, ports, sockets, send/receive, RPC, T/TCP
Peterson 6.1, 6.3 - 6.5, 7.1, 9.1

Phase 2: From bits to messages across the Internet

Jan. 27 - 29: The hardware zoo: Bits, frames, CRC's, topologies, switches and host adaptors
Peterson 3.1 - 3.4, 3.8, 4.1, 4.4., 9.2

Feb. 3, 5pm: Project 1 due

Feb. 4 - 6: Routing: IP, ATM, and multiprocessor networks
Peterson 4.2-4.3, 5.1-5.4

Feb. 11 - 20: Congestion control from Ethernet to TCP
Peterson 3.6, 8

Feb. 13: Midterm

Phase 3: Using networks to build applications

Feb. 25 - 27: Retrieving data: Web, NFS, DNS and cache coherence
Peterson 5.6

Feb. 26, 5pm: Project 2 due

March 4 : Reliable applications: transactional RPC and group communication

March 6: Push-based applications: multicast
Peterson 5.5

March 11 - 13: Secure applications and course summary
Peterson 7.3

March 16, 5pm: Project 3 due

March 19, 2:30 - 4:30: Final Exam