CS 5651 Computer Networks

The Internet and its Structure

Pete Willemsen

University of Minnesota Duluth

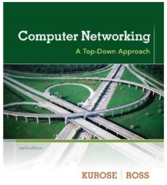
September 4 2013

Important Info

- **Lecture** - M/W/F 3:00pm - 3:50pm HH 214
- My Office Hours: M 11am-11:45am, W 1pm-2pm, F 10am-11am
  - Email: willemsn@d.umn.edu
  - Will *try* to be available via Google chat during office hours
- TA: Sarmad Siddiqui
  - Email: siddi035@d.umn.edu
  - Office Hours: TBD

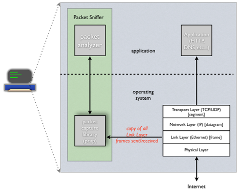
Reading Materials

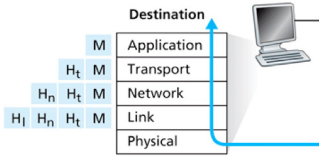
- Computer Networking: A Top-Down Approach, 6th Edition. Kurose and Ross.
  - Required Reading!



Information

- Course Requirements
  - Programming Assignments (30%) - There will be several programming assignments over the course of the semester. Programming assignments will focus on developing experience with programming the network and network applications.


Packet Sniffer
packet analyzer
application
Application (HTTP, DNS, etc...)
operating system
packet capture library (pcap)
copy of all Link Layer frames sent/received
Transport Layer (TCP/UDP) [segment]
Network Layer (IP) [datagram]
Link Layer (Ethernet) [frame]
Physical Layer
Internet



Information

- Course Requirements, cont'd.
  - Homework Assignments (10%) - Homework problems will be assigned on the Moodle on a weekly or bi-weekly basis. You are strongly encouraged to complete these homework assignments.

Information

- Course Requirements, cont'd.
    - Homework Assignments (10%) - Homework problems will be assigned on the Moodle on a weekly or bi-weekly basis. You are strongly encouraged to complete these homework assignments.
    - Lab (15%) - Labs will provides hands-on experience with computer network setup, analysis, equipment, tool use, and programming. Labs may also be used to validate your programming assignments or discuss problems with the TA. Attendance and completion of labs is required for a successful lab grade.

Information

- Course Requirements, cont'd.
  - Mid-term Exam (20%) - There will be two mid-term exam which will cover the material from roughly the first two "sections" of the course.

Information

- Course Requirements, cont'd.
  - Mid-term Exam (20%) - There will be two mid-term exam which will cover the material from roughly the first two "sections" of the course.
  - Final Exam (25%) - The final exam is comprehensive, but will also cover the material from the last third of the course. There are no excuses for missing the final! The final exam is scheduled for Wednesday, December 18 from 10:00am - 11:55am, according to the Final Exam Schedule.

Moodle

- http://moodle.umn.edu/
    - Used for course management, forums, posting of assignments, etc...
    - Login via your X.500 login information.
- Weekly summary of topics
- Programming assignments, quizzes, discussions, homeworks, labs, and even exams may use moodle.
- Forums will be used extensively for feedback and assessment on programming assignments and discussions.

Computer Networks

Concepts

Course will focus on the concepts needed to understand the organization of the Internet and computer networks, as well as how data is passed between machines across the Internet.

Implementation

The course will also heavily focus on teaching you some of the aspects involved with **HOW** to setup and use networks, as well as write applications that use the Internet and other computer networks to send data.

Useful and Necessary Tools

You will need to be familiar with the following (ASAP):

- Linux editors and C++ compiler

Useful and Necessary Tools

You will need to be familiar with the following (ASAP):

- Linux editors and C++ compiler
- CMake - build system meta tool

Useful and Necessary Tools

You will need to be familiar with the following (ASAP):

- Linux editors and C++ compiler
- CMake - build system meta tool
- Subversion - code repository software

Writing Code - Editor and Compiler

Editing your code with popular and not-so popular editors:

- gedit
- emacs
- vi
- kate (it seems really nice)
- *others???*

You will compile your code with the GNU C++ compiler for your assignments. Here's some helpful review:

- Use C++ classes well!
- Declarations go in header file (.h); Definitions go in source file (.cpp)
- What's an inline function and when to use them?
- Use the -*g* and -*Wall* flags and pay attention to it
- Use the debugger! It's called *gdb* on Linux.

Writing Code - Good Practices

Use good practices when writing your code. Some ideas follow:

- Use proper indentation - makes code easier to read and allows code blocks to be easily viewed. (**Required**)
- Comment your code - will you really remember what you did a year from now, let alone six weeks from now? (**Required**)
- Be consistent with naming conventions. Pick one you like and stick with it, but for sure, use good meaningful names. For instance, *mainData_ptr*, *float averageValue_f*, *MyClassA A;*
- Create enough classes/data structures to help you out!
- Break your code up into libraries (for reuse) as well as exportable headers and source files.
- Use exceptions when possible; they're nice!

I will deduct points if your code is hard to read and/or poorly commented.

Building your Code

Use CMake!
CMake is a high level build system that will let you cross compile your code on numerous build environments.
Why you want to learn about it and use it:

- Bigger projects; no messing around typing extra stuff to compile
- Build libraries of your code!
- Cross-platform development

All CMake-based project will contain a file, called `CMakeLists.txt`:

- CMakeLists.txt contains the build instructions for your project.
- Specifies which files are part of the project and any dependencies.

Simple CMake Example

You have one file set that contains your class and a file that contains your main.

```
main.cpp  ClassTest.h  ClassTest.cpp
```

Normally, you could use a Makefile to *build* this into your executable. But Makefiles are not native to Visual Studio (Windows) and Xcode (OS X) and you want to develop there also.

CMake can help. Here is what the CMakeLists.txt file looks like to build your project:

```
CMAKE_MINIMUM_REQUIRED (VERSION 2.8)
PROJECT (ClassTest)

ADD_EXECUTABLE (classTest
                classTest.cpp classTest.h
                main.cpp)
```

Simple CMake Example Cont'd

Once you have your CMakeLists.txt file, you can build your project. A couple notes:

- Keep your sources separate from the build directories. I create a build directory to hold the "build" files. This is similar to what VS and other IDEs do.
- Only commit your sources, not your build directories to your software versioning repository (svn).

How to build:

```
mkdir build
cd build
cmake ..
make
```

- If your build directory is already there, you don't need to re-create it, nor run the cmake command again.
- You only need run the make command
- You can clean and build all with the *make clean* command provided in your auto-generated Makefile (CMake did this for you)

Keeping your Code

Use a software repository and versioning system!
Subversion is good, but there are others that are useful if you know how to use them:

- git
- mercurial
- *numerous others...*

Keeps a backup of your code!
Let's your code be accessed from many places!
For multi-person projects, this is a MUST!

subversion

Documentation - http://svnbook.red-bean.com/

- Revision control system
- Store your code at various states in time
- Tag your code with easy to remember names
- Branch your code for explorative dev
- Possible to setup with an Apache server
- But even simpler to setup without a dedicated server!
    - Requires that your machine has *ssh* and *subversion* installed
    - Machine should also be “online”

subversion Setup

First thing to do

- Create directory in your account to house your projects

Next,

- Create subversion repository for your project
  - *cd your_directory*
  - *svnadmin create project1*
  - *chmod -R go-rwx ..*
- This creates the "database" that subversion uses to keep track of your project.

Checking out your Project

Use the SSH protocol do to this securely

- svn co svn+ssh://*userid@machine*/path_to_repo_here *your local name*

Example

- svn co svn+ssh://willemsn@marengo/home/vol/svnrepo/project1 project1

Creating the Initial Project Structure

- I tend to like having a *trunk* (mainline of development), a *tags*, and a *branches* directory structure to organize my subversion projects. After creating project with svnadmin, follow the example below to get setup this way:
  - svn co svn+ssh://willemsn@marengo/home/vol/svnrepo/project1 project1
  - cd project1
  - mkdir -p trunk tags branches
  - svn add trunk tags branches
  - svn commit -m "Added initial dir structure." trunk tags branches
  - cd ..
  - /bin/rm -fr project1
  - svn co svn+ssh://willemsn@marengo/home/vol/svnrepo/project1/trunk project1
- You'll then be ready to go with your main trunk of development!

Important SVN Commands

- Add new files or directories (recursively) to the repository
    - svn add [files or dirs]
- Update your copy from the repository
    - svn update
- Commit your updates
    - svn commit -m "My comment about updates" [dirs or specific files]
- Differences between your copy and repository
    - svn diff -rHEAD [my file or directory]
- Local status of your repository
    - svn status

git

Documentation - http://git-scm.com/documentation

- Revision control system
- Store your code at various states in time
- Manages changes to a tree of files
- Optimized for distributed development and large file set
- Also good at merges and branching
- Originally developed by Linus Torvalds

Stanford has a nice basic overview of git commands:
http://www-cs-students.stanford.edu/~blynn/gitmagic/ch02.html

git Setup

We'll use a directory on a CS machine to hold your git repository (marengo would be good to use):

- Create directory in your account to house your projects

Next,

- Create git repository for your project
  - *cd your_git_directory*
  - *GIT_DIR=project1.git git init*
  - *cd project1.git*
  - *cp hooks/post-update.sample hooks/post-update*
- This creates the main git master repository to keep track of your project.

Cloning the Project with git

On a remote machine

- git clone username@marengo.d.umn.edu: /your_git_directory/project1.git
- cd project1
- emacs README
- git commit -m "Added the readme file"
- git push origin master

I suggest you read up on git.