CSci551 Fall 2014 Project C

Assigned: 2014-10-31. Project C Due: noon, Wed. 2014-12-03.

You are welcome to discuss your project with other students at the conceptual level, and to use external libraries *after getting permission*. Any cases of plagiarism will result in an F for the entire course. **If you have any questions about policies about academic integrity, please talk to the professor.**

Changes: 2014-10-30: none yet

1 Overview

There are two *paths* for projects in CSci551 this year. The *Practical Path* project involves implementing a network-centric system. Everyone following this path solves the same problem, but of course in slightly different ways.

In the *Research Path*, students do new research working under the guidance of a mentor (usually a senior PhD student).

PhD students are required to do Research Path projects. Other students may choose either path.

In both cases the project is divided into three parts, Projects A, B and C. Each builds on the prior.

1.1 The Practical Path Project

The purpose of the CSci551 project is to get some hands on experience building a substantial distributed application running on a multi-process computing infrastructure, then to use it to study networking. It also has a secondary purpose of implementing a program using network programming (sockets) and processes (fork, in stage 1) and Unix development tools (make).

You may reuse algorithms from textbooks. You may possibly reuse functions from libraries, but if you're using anything other than the C or C++ standard library (libc, STL), or libraries mentioned on the TA's web page, you *must* check with the TA and the professor and identify it in your write-up. You need to check allowed libraries in Project Information on the class moodle. Otherwise, each student is expected to write *all* of his or her code independently. All programs will be subject to automated checking for similarity.

Please note: you should expect to spend a significant amount of time on the class projects this semester, probably at least 20 hours or more when they're all put together (or that much per project if you're new to C or C++ and network programming). Please plan accordingly. If you leave all the work until the weekend before it is due, you are unlikely to have a successful outcome. That said, Project A is a "warm-up" project. Projects B and C will be much larger. Although Project C is smaller than Project B, you should expect it will still take a significant amount of time.

For the course, you will do three separate but related projects. First (Project A), you need to demonstrate that you can read the config file, do basic process control, and submit

a complete assignment. (Project A doesn't really evaluate anything advanced, but it should confirm that everyone is on the same page and is comfortable with our software environment.) Project A has an early due date. Project A should be relatively short for students used to working on Linux; if not, it should help get you up to speed.

In Project B you will use this facility to implement *Minitor*, a simplified version of a TOR-like Onion Router as described in the paper by Dingledine et al. (see [Dingledine04a] in the class syllabus). It will probably be due just after the midterm. Project B will be *much* larger than Project A; you should plan your time accordingly. Project B will be assigned later and may overlap partially with Project A.

Project C will be assigned later. It will be smaller than Project B but bigger than Project A. It will build on Project B. We will offer a the TA's implementation of Project B for students who wish to use it instead of their own Project B implementation, but we do not promise to help you understand it. Project C will involve extending your Project B Minitor implementation with additional features, and perhaps measuring it or trying to de-anonymize it. It will probably be due the last week of class.

1.2 Research Path Projects

This year we are have an alternate *research path* for CSci551 projects. Students will conduct original research on topics related to the course material, helping to further our understanding or develop new approaches for open problems–at least, as much as one can within a semester! For these projects, you can use the language and libraries of your choice, subject to approval from your mentor (more on mentors below). As usual, you must properly credit and cite any code, text, or approaches that you borrow from others. Using and extending the research of others is a crucial part of research, but requires proper attribution.

All PhD students are *required* to do research path projects. MS students have the *option* of doing it, although MS students can cancel research path and pick up the practical project as an alternative after discussing this choice with the professor.

We will have a poster session at the end of the semester, where students can present their research path results. We encourage all students–not just research path students–to attend.

Research Projects in the Friday section are *individual* projects (not groups), but there may be some linked projects where two people work on different parts of the same problem. (Linked projects will share ideas and possibly some data or code or results, but each student will have a specific part they are responsible for and each student must do their own writeup.)

Research Projects in the Tuesday/Thursday section will allow two-person groups projects or individual projects.

2 Practical Project A: Getting Started

(We don't reproduce Project A here, but you may want to refer to it for background. You are required to use the same VM development environment from Project A in Projects B and C, and you should assume similar infrastructure, with a proxy and routers and log files.)

3 Research Project A: Getting Started

(See the original project A assignment for details about Research Project A.)

4 Practical Project B: Minitor

We don't reproduce it here, but you may wish to refer back to your Project B for hints and background. In fact, Project C is required to use the same messages, message formats, and logging rules as Project B.

In general, you should assume that each stage includes all prior stages, with later stages superseding earlier ones where they conflict. (For example, encryption in stage 6 succeeds non-encryption.)

5 Research Project B

We don't reproduce it here, but you may wish to refer back to your Project B.

6 Practical Project C: Extending Minitor

6.1 Stage 7: TCP to the Real World

In stage 3 you reached out to the real world. And as much as your professor loves ping, there is more to the Internet than that. In this stage you will extend Minitor to support TCP connections that pass through the proxy and router.

You will need to continue with all the setup of prior stages, and you should continue to support ICMP. But now your router must also rewrite TCP packets, not just ICMP.

Conceptually, in our case, proxying TCP is just like proxying ICMP: you need to update the IP address, recompute the packet header checksum, and send it through. We can get away with this simple treatment because we're not doing full proxying, and we know packets come from a single computer, so there's need to remap ports.

We also require that you log each TCP packet as it comes from tunnel in the proxy, as `TCP from tunnel, src IP/port: SI:SP, dst IP/port: DI:DP, seqno: N, ackno: M`. and also as it comes from routers, as `incoming TCP packet, circuit incoming: 0xIDi, src IP/port: SI:SP, dst IP/port: DI:DP, seqno: N, ackno: M`.

We also require that you log each TCP packet as it comes from the other routers in the last hop router, as `outgoing TCP packet, circuit incoming: 0xIDi, incoming src IP/port: Si/Sp, outgoing src IP/port: SI:SP, dst IP/port: DI:DP, seqno: N, ackno: M` and also as it comes from raw socket, as `incoming TCP packet, src IP/port: SI:SP, dst IP/port: DI:DP, seqno: N, ackno: M, outgoing circuit: 0xIDo`.

As in prior stages, proxy and all routers should also log the packet in hex format and intermediate routers also log relay information, as in prior stages, and packets should be sent using onion routing with encryption.

For this stage we must make sure the Linux kernel stays out fo the way for TCP. To avoid this problem, we'll filter outgoing RST packets with this command[1]:

```
sudo iptables -A OUTPUT -p tcp --tcp-flags RST RST -j DROP
```

In addition, we will use ant.isi.edu (128.9.160.91) as one of our targets, so you need to add one route:

```
sudo ip route add table 9 to 128.9.160.91 dev tun1
```

For this stage (only) you can assume we will send *only* TCP traffic to a single destination IP address, and only one TCP connection at a time. While that IP address can be anywhere, we won't send to multiple IP addressees, nor will not send ICMP traffic. However, you should allow for it to be on different ports so we can send multiple connections to that destination sequentially. (We will relax this limitation in the next stage.)

Sample input: Sample input will be identical to stage 6, except for the stage number.

```
stage 7
num_routers 4
minitor_hops 2
```

Sample output: After running your program and the above route commands, `echo "GET /" | nc -s $IP_OF_ETH ant.isi.edu 80 >index.html`. where `IP_OF_ETH` is set as described in stage 3, you should see TCP traffic in your sample output.

Please check the sample output for stage 7 is in the assignments folder.

Writeup: Please prepare a README for stage 7, called `README.stage7.txt`.

a) Reused Code: Did you use code from anywhere for your project? If not, say so. If so, say what functions and where they're from. (Also identify this with a comment in the source code.) If you use the class timer code, you must say so here and describe any changes you had to make to it.

b) Complete: Did you complete this stage? If not, what works and what doesn't?

c) What would happen to your TCP connection if the kernel's RST packets were not filtered out?

d) Why does the Linux kernel generate a RST packet for TCP, while it ignored ICMP and UDP packets in prior stages?

[1] If this approach to proxying TCP seems somewhat less than elegant, I think you're correct. It does, however, *work*, unlike a number of alternatives. We recommend you use it for that reason. If someone would like to demonstrate an alternative, please let us know—you still need to use this approach for your project and it won't change your grade, but we would like to know cleaner approaches.

6.2 Stage 8: Multiple Circuits

Next we consider handling multiple active circuits.

Real TOR has support for multiplexing multiple TCP connections as streams in circuits. We're not going to re-create that, but we can put one flow in each circuit.

In this stage, we relax the limitation of a single TCP destination IP address, and we re-allow ICMP traffic. We define a *flow* as TCP traffic with the standard 5-tuple: a unique source and destination IP address and port, plus the protocol number. We will not actually send UDP traffic. We may send ICMP; you can treat it as have port 0 at both ends when defining its flow.

Each flow should go to a unique circuit. The proxy should maintain a flow cache. When a packet arrives from the tunnel, it should identify the flow. For each new flow, it should construct a unique circuit. Each circuit will have a different, randomly computed list of hops, each circuit of the length given by `minitor_hops` in the configuration file. The proxy should remember the flow-to-circuit mapping so subsequent packets reuses the same circuit. (Note that Project B required circuit IDs to be $i * 256 + s$, where i is the router id (1 to N), or 0 if it's the proxy, and s is a sequential number, starting at 1. You are now going to increment s for each new flow.)

All routers must keep track of multiple circuits, and egress routers must be able to map incoming packets to the correct return circuit.

Sample input: Sample input will be identical to prior stages, except for the stage number.

```
stage 8
num_routers 4
minitor_hops 2
```

Sample output: Now you should be able to send to multiple destinations:

```
echo "GET /" | nc -s $IP_OF_ETH ant.isi.edu 80 >a.html;
echo "GET /" | nc -s $IP_OF_ETH www.csail.mit.edu 80 >b.html;
ping -c 1 -I $IP_OF_ETH www.csail.mit.edu.
```

In your output you should see that each of these create a different circuit.

Writeup: Please prepare a README for stage 8, called `README.stage8.txt`.

a) Reused Code: Did you use code from anywhere for your project? If not, say so. If so, say what functions and where they're from. (Also identify this with a comment in the source code.) If you use the class timer code, you must say so here and describe any changes you had to make to it.

b) Complete: Did you complete this stage? If not, what works and what doesn't?

c) In this stage we are careful to make sure all packets for a given flow take the same path. What bad thing might happen if different packets from one flow took different paths of different lengths?

d) Continuing (c), suppose all paths were the same length and they were sent slowly (say, one packet every millisecond), but each packet went over a different circuit. Would the problem you identified in (c) be likely to occur in our simple test network on one machine?

e) Continuing (d), now suppose paths were the same length and packets were sent every millisecond, but now Minitor nodes were anywhere in the Internet, not all one test machine. Now, would the problem you identified in (c) be likely?

6.3 Stage 9: Router Failure and Recovery

In addition to the challenge of handling many flows (which we only partly emulated), TOR has to deal with node failures. We next expand Minitor to deal with simple node failures.

We will add a new input command `die_after N`, which means that you should kill the second router in each TOR path after N packets have been sent on that path. However, each path should loose its second router after N packets on that path (it's a tough world). Each path will lose one router once, and then be stable. So that you can form paths, after the number of remaining routers is equal to `minitor_hops`, you must stop killing routers. (As said before, we promise that `num_routers` will exceed `minitor_hops` in the initial configuration file.)

Ideally we would reach in and kill your routers ourselves, but for consistency we're going to have the proxy do it for us in a controlled way. The proxy should count up to N packets for a flow. Just after forwarding the Nth packet, it should kill the second router on that path. It will determine router hosts the second hop of that flow's circuit and send *kill-router*, control message type 0x91, directly to the mortal router. On receiving that message, the router should log "router X killed" and then terminate (the Unix process should exit). The proxy and other routers should not do anything else—they don't get to do any cleanup, so we simulate an uncontrolled failure.

Routers must now detect when their peers die. To detect failure, each router should keep a timer when it sends proxied packet down a circuit. The timer should count down 5 seconds. If the router gets a return packet on the circuit, it should reset the timer. If the timer expires with no return traffic before 5 seconds, the router should log "router SELF-NAME worried about NEXT-NAME on circuit circuit-ID". (Self-name and next-name are the UDP ports of the current next next-hop routers.)

Then it should generate a *router-worried*, control message type 0x92, with the circuit ID, SELF-NAME and NEXT-NAME, to indicate that itself (router ID SELF-NAME) thinks router NEXT-NAME may be done. It should onion-route this back to the proxy up the circuit.

When the proxy gets a router-worried message, it should mark the NEXT-NAME as down. It should then discard the circuit from its list of active circuits. As a result, the next time traffic for that circuit arrives at the proxy it will be forced to rebuild a new circuit.

You are only require to support one circuit in stage 9.

Sample input: Finally, a slightly different sample input (like the good old stages).

```
stage 9
num_routers 6
minitor_hops 3
die_after 5
```

Sample output: If you run `ping -c 1 -I $IP_OF_ETH www.csail.mit.edu`. You should see just regular traffic. But then `ping -c 6 -I $IP_OF_ETH www.csail.mit.edu` should show a router failure in the logs.

And

```
ping -c 6 -I $IP_OF_ETH www.csail.mit.edu;
sleep 6;
ping -c 2 -I $IP_OF_ETH www.csail.mit.edu.
```

Should show failure and recovery on a new path.

Writeup: Please prepare a README for stage 8, called `README.stage8.txt`.

a) Reused Code: Did you use code from anywhere for your project? If not, say so. If so, say what functions and where they're from. (Also identify this with a comment in the source code.) If you use the class timer code, you must say so here and describe any changes you had to make to it.

b) Complete: Did you complete this stage? If not, what works and what doesn't?

7 Research Project C

For Research Project C, you will be expected to work on your research project regularly throughout the period of the assignment. In particular, you will be responsible for the following deliverables, as described in your "next steps" part of your writeup for Research Project B.

1. Regular meetings with your mentor. We expect these will be about weekly, so you should have about 4 of them (and at least 3) over this period. (You may have more if you want.)

2. Written weekly notes about your progress, sent to your mentor the day before your weekly meeting, commented on by the mentor at/after the meeting, and updated by you after the meeting.

3. Code, data, and analysis results, provided when you submit this portion of the project.

4. With the other files you submit, you should include a README.txt file describing what you submitted. This README should also include a URL or filename for the weekly notes about your progress and meetings with your mentor.

5. A Project C report. This document should be about 4–8 pages summarizing your research project work over the semester. This report should be a mini-paper, providing

a self-contained summary of everything you did on your research project. Whereas Project A and Project B could be more informal, Project C should be a final report similar to a conference paper in content and format. The papers we read this semester may be useful as a guide for how to write a good research paper. (It is due at the same time as the Practical Project C, although the poster has a different deadline (see below).)

It must include the following numbered sections:

Section 1 An introduction, giving an overview of what you're doing, why is this work important (interesting to the field), and what is new about it?

Section 2 A description of the relationship to other work. As you review related work, include citations in your text. Make sure each piece of related work identifies how it is alike and different from your work.

Section 3 A section listing what your goals were from Research Projects A and B with a statement about which goals you completed, and for those you didn't complete, why you didn't complete them. Possible Reasons may vary from "ran out of time" or "unable to get access to data" to "decided with mentor that result Y was more interesting than result X"; you may have other reasons as well.

(This section is the only one that would not appear in a real paper.)

Section 4 A discussion of your work. You will likely want to use multiple sections or subsections here, and you should identify your goal, methodology/approach, data collection, and results.

Section 5 A description of possible future work following Project C. You are not required to actually do this work, but it's always good to identify where the work might go, especially since the project work might not have perfectly aligned with the semester deadline.

Section 6 A bibliography, with at least what you cite in related work

In addition, make sure you provide a title, list you as the author, identify your mentor, and include a URL to your weekly notes, all before the first section.

(We encourage you to write your document in LaTeX and build a PDF, but you can use whatever tool you prefer.)

6. A poster summarizing your work. The poster session is tentitively scheduled for 11am–12:30pm 2014-12-09. (We will provide pizza for lunch!) We will confirm the date, time, and place as soon as we can. Making a poster is required. Attending the poster session is strongly encouraged (but not required). While it is not required, we strongly hope you will attend the poster session to describe your work to us, to your classmates, and to other attendees. *All* CSci551 students are encouraged to attend as well.

   We will print posters for you, but *only* if a copy is uploaded to the course moodle 48 hours before the presentation date. Alternatively, you may get it printed yourself (it should be about $50 at Kinkos, or you may have access to your own large format printer).

As with project B, since research is not always predictable, you will not be graded on executing exactly the plan in your Research Project B proposal, but it will serve as a guideline. However, the most important factor is to conduct quality work that adapts to what you discover about the problem as you go. Your mentor especially (as an expert on the problem), but also your TAs and professor, can help focus you on interesting directions throughout the semester.

7.1 About Posters

There are two kinds of posters: (a) ones you stand by and talk about when people come up, and (b) ones that you leave alone and people walk up and read without you.

Medical doctors do type (b), where you go to a conference with 2000 people and have poster sessions with 300 posters. Those posters are little mini-papers with lots of text.

Computer science poster sessions that I've seen are almost *always* like type (a). With type (a), since you're standing there, the poster ends up more like PowerPoint slides, but with enough text that you CAN read them, but lots of pictures and the details are spoken by you standing next to it.

I recommend that posters follow (roughly) NABC format.

A rough format is:

```
.......TITLE.......
......authors......

abstract      problem (need)


              benefits
approach


competition
              conclusion
```

in two column format.

Other general poster advice:

- A poster should *never* just be an array of 8.5x11" power point slides (in a grid). That's like writing slides by pasting a copy of your paper into PowerPoint.
- To make it look non-grid-like, it's usually good to keep the column section at different locations. (see the examples we provide below).
- write interpretation of the graphs directly on the graphs, using arrows to point at what about the graph supports your claim. (Don't just put bullet points below the graph.)
- Always include a date on the poster (typically in gray in the lower margin), otherwise in 2 years you'll look back and forget the context.

- For a public poster, it's great to include a pointer to your paper or technical report or website for more information.

I recommend you do posters in PowerPoint on a 24x36" single page. You can also use Adobe Illustrator or Apple Keynote or Microsoft Viseo or LibreOffice Impress, or whatever you prefer. There are LaTeX templates for posters, although they can be somewhat difficult to get custom layout. (All electronic posters are *much* easier and neater than physically cutting and pasting, as one of your professors did when he was in graduate school :-)

Some sample posters are on the Moodle.

8 Submitting and Evaluating Practical Projects

To submit each part the assignment, put *all* the files needed (Makefile, README, all source files, and source to any libraries) in a gzip'ed tar file and upload it to the class moodle with the filename `projc.tar.gz`. *Warning:* when you upload to the moodle, please be careful in that you must both do "Upload a file" *and* do "Save changes"! When you are done you should get a message "File uploaded successfully", and you should see a list with your file there and an option to "update this file". If you do *not* see "file uploaded successfully", then you have *not* completed uploading!

We *strongly* recommend that you confirm that you have included everything needed to build your project by extracting your tarfile in a different directory and building it yourself. (It's easy to miss something if you don't check.)

It is a project requirement that your project come with a Makefile and build with just the make command. To evaluate your project we will type the following command:

```
% make
```

Structure the Makefile such that it creates an executable called `projc`. (Note: *not* `a.out`.) For more information please read the make(1) man page. (The Linux we use supports GNU make; you may use GNU make extensions if you wish.)

We will then run your program using a test configuration file. You can assume that the topology description will be syntactically correct. After running the program, we will grade your project based on the output files created by your program's processes.

It is a project requirement that your implementation be somewhat modular. This means that you should follow good programming practices—keep functions relatively short, use descriptive variable names. You must use at least one header file, and multiple files for different parts of your program code. (The whole project should be broken up into *at least* two C/C++ files. If you have a good file hierarchy in mind you can break up into more files but the divisions should be logical and not just spreading functions into many files.) Indicate in a comment at the front of each file what functions that file contains.

We will consider external libraries, but **it is a project requirement that all external libraries must be explicitly approved by the professor**. Any libraries in the default VM image are suitable (including libc and STL). A list of approved libraries will be on the TA's webpage on the moodle.

Computer languages other than C or C++ will be considered but *must be approved ahead of time*; please contact the professor and TA if you have an alternative preference. The deadline for approving new computer languages is *one week* before the project due date, so get requests in early. The language must support sockets and process creation. (Please ask *before* you start, we don't want you waste your work.)

Although we provide a complete sample input file and output, final evaluation of your program will include other input sources. We therefore advise you to test your program with a range of inputs.

Although the exact output from your program may be different from the sample output we provide (due to events happening in different orders), your output should match ours in format.

It is a project requirement that your code build without warnings (when compiled with -Wall). We will be responsible for making sure there are no warnings in class-provided code (any warnings in class-provided code are our bugs and will not count against you).

9 Hints

The structure of the project is designed to help you by breaking it up into smaller chunks (compared to the size of the whole project). We strongly encourage you to follow this in your implementation, and do the stages in order, testing them as you go. In the past, some students have tried to read and implement the whole assignment all at once, almost always resulting in an unhappy result.

9.1 Common pitfalls

Please do not hardcode any directory path in your code! If you hardcode something like `//home/csci551/...` in your code to access something in your home directory and the grader cannot access these directories during grading, your code will not work (and this will be your fault)! If your code does not work, you get no credit! Instead, assume paths are given external to your program, and that you read and write files in the current directory (wherever that is).

9.2 Doing multiple things at once

Later stages of the project may require you to handle both timers and I/O at the same time. (Stage 1 is not complex enough to require timers.) One approach would be to use threads, but most operating systems and many network applications don't actually use threads because thread overhead can be quite large (not context switch cost, but more often memory cost—most threads take at least 8–24KB of memory, and on a machine with 1000s of active connections that adds up, and always in debugging time, in that you have to deal with synchronization and locking). Instead of threads, we strongly encourage you to use timers and event driven programming with a single thread of control. (See the talk "Why

Threads Are A Bad Idea (for most purposes)" by John Ousterhout, `http://www.stanford.edu/~ouster/cgi-bin/papers/threads.pdf` for a more careful explanation of why.)

Creating a timer library from scratch is interesting, but non-trivial. We will provide a timer library that makes it easy to schedule timers in a single-threaded process. You may download this code from the TA web page. There is *no* requirement to use this code, but you may if you want. If you want to use it, download it from the class web page. There is no external documentation, but please read the comments in the `timers.hh` and look at `test-app.cc` as an example. If you do use the code, you must add it to your Makefile and you must document how you used it in your README.

9.3 Other sources of help

You should see the Unix man pages for details about socket APIs, fork, and Makefiles. Try `man foo` where foo is a function or program.

The TAs can provide *some* help about Unix APIs and functionality, but it is not their job to read the manual page for you, nor is it their job to teach how to log in and use vi or emacs.

You may wish to get the book *Unix Network Programming*, Volume 1, by W. Richard Stevens, as a reference for how to use sockets and fork (it's a great book). We will *not* cover this material in class.

The README file should not just be a few sentences. You need to take some time to describe what you did and especially anything you didn't do. (Expect the grader to take off more points for things they have to figure out are broken than for known deficiencies that you document.)