

Lab 0: networking warmup

Welcome to NJU: Computer Networking. In this warmup lab, you will set up an installation of Linux on your computer, learn how to perform some tasks over the Internet by hand, write a small program in C++ that fetches a Web page over the Internet, and implement (in memory) one of the key abstractions of networking: a reliable stream of bytes between a writer and a reader. We expect this warmup to take you between 2 and 6 hours to complete (future labs will take more of your time).

1 Set up Linux on your computer

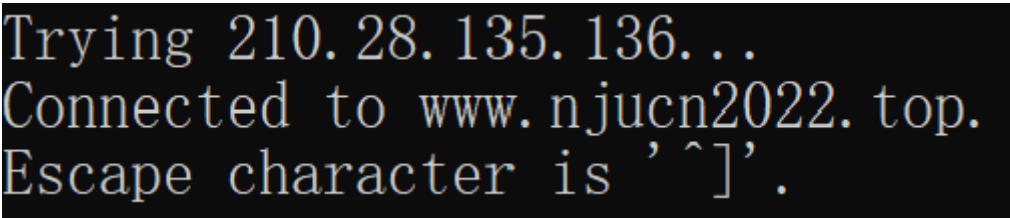
Our assignments require the GNU/Linux operating system and a recent C++ compiler that supports the C++ 2017 standard. We will provide a document to guide you to set up the environment, you can find the document in our QQ group.

2 Networking by hand


Let's get started with using the network. You are going to do two tasks by hand: **retrieving a Web page** (just like a Web browser) and **telnet on your PC**. Both of these tasks rely on a networking abstraction called a **reliable bidirectional byte stream** you'll type a sequence of bytes into the terminal, and the same sequence of bytes will eventually be delivered, in the same order, to a program running on another computer (a server). The server responds with its own sequence of bytes, delivered back to your terminal.




2.1 Fetch a Web page


1. In a Web browser, visit www.njucn2022.top and observe the result.
2. Now, you'll do the same thing the browser does, by hand.
 - **(a)** On your VM, run `telnet www.njucn2022.top http`. This tells the telnet program to open a reliable byte stream between your computer and another computer (named www.njucn2022.top), and with a particular service running on that computer: the "http" service, for the Hyper-Text Transfer Protocol, used by the World Wide Web. If your computer has been set up properly and is on the Internet, you will see:



```
Trying 210.28.135.136...
Connected to www.njucn2022.top.
Escape character is '^]'.
```

If you need to quit, hold down `ctrl` and press `]`, and then type `close` .

- **(b)** Type `GET /ALPHA HTTP/1.1` . This tells the server the path part of the URL.
- **(c)** Type `Host: www.njucn2022.top` . This tells the server the host part of the URL.
- **(d)** Type `Connection: close` . This tells the server that you are finished making requests, and it should close the connection as soon as it finishes replying.


- **(e)** Hit the Enter key one more time: . This sends an empty line and tells the server that you are done with your HTTP request.
- **(f)** If all went well, you will see the same response that your browser saw, preceded by HTTP headers that tell the browser how to interpret the response.

2.2 Listening and connecting

You've seen what you can do with telnet: a **client** program that makes outgoing connections to programs running on other computers. Now it's time to experiment with being a simple **server**: the kind of program that waits around for clients to connect to it.

1. In one terminal window, run `netcat -v -l -p 9090` on your VM. You should see:

```
xun@xun-virtual-machine:~$ netcat -v -l -p 9090
Listening on [0.0.0.0] (family 0, port 9090)
```

2. Leave netcat running. In another terminal window, run `telnet localhost 9090`.
3. If all goes well, the netcat will have printed something like "Connection from localhost 53500 received!".
4. Now try typing in either terminal window---the netcat (server) or the telnet (client). Notice that anything you type in one window appears in the other, and vice versa. You'll have to hit  for bytes to be transferred.
5. In the netcat window, quit the program by typing `ctrl -C`. Notice that the telnet program immediately quits as well.

2.3 Try some simple commands

- ping

*Ping is a computer network administration software utility used to test the **reachability** of a host on an Internet Protocol (IP) network. It is available for virtually all operating systems that have networking capability, including most embedded network administration software.*

*Ping measures the **round-trip time** for messages sent from the originating host to a destination computer that are echoed back to the source. The name comes from active sonar terminology that sends a pulse of sound and listens for the echo to detect objects under water.*

In windows cmd, run `ping www.njucn2022.top`, you will see the result like the picture below:

```
C:\Users\123>ping www.njucn2022.top

正在 Ping www.njucn2022.top [210.28.135.136] 具有 32 字节的数据:
来自 210.28.135.136 的回复: 字节=32 时间=12ms TTL=60
来自 210.28.135.136 的回复: 字节=32 时间=10ms TTL=60
来自 210.28.135.136 的回复: 字节=32 时间=11ms TTL=60
来自 210.28.135.136 的回复: 字节=32 时间=11ms TTL=60

210.28.135.136 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
    最短 = 10ms, 最长 = 12ms, 平均 = 11ms
```

- traceroute

In computing, **traceroute** and **tracert** are computer network diagnostic commands for displaying possible routes (paths) and measuring transit delays of packets across an Internet Protocol (IP) network. The history of the route is recorded as the round-trip times of the packets received from each successive host (remote node) in the route (path); the sum of the mean times in each hop is a measure of the total time spent to establish the connection. Traceroute proceeds unless all (usually three) sent packets are lost more than twice; then the connection is lost and the route cannot be evaluated. Ping, on the other hand, only computes the final round-trip times from the destination point.

You can run **traceroute** www.njucn2022.top in **Ubuntu** terminal or run **tracert** www.njucn2022.top in **Windows** cmd(in Ubuntu you should run **sudo apt-get install traceroute** to install it before using it). You will see the result like the picture below:

```
C:\Users\XUN>tracert www.baidu.com

通过最多 30 个跃点跟踪
到 www.a.shifen.com [14.215.177.38] 的路由:

  1  269 ms    8 ms    6 ms  192.168.15.254
  2   1 ms     4 ms    3 ms  10.10.10.1
  3  11 ms     4 ms    5 ms  118.121.168.1
  4   4 ms     3 ms    5 ms  89.163.213.222.broad.dy.sc.dynamic.163data.com.cn [222.213.163.89]
  5   *        13 ms   *      61.139.122.117
  6   *        *      *      请求超时。
  7  42 ms     48 ms   43 ms  113.96.4.186
  8  44 ms     46 ms   47 ms  113.96.4.209
  9  48 ms     48 ms   86 ms  113.96.11.74
 10  47 ms     50 ms   48 ms  14.29.121.186
 11   *        *      *      请求超时。
 12   *        *      *      请求超时。
 13  53 ms     49 ms   48 ms  14.215.177.38

跟踪完成。
```

tracert in Windows

```
xun@DESKTOP-60F52VQ: $ traceroute www.baidu.com
traceroute to www.baidu.com (14.215.177.38), 30 hops max, 60 byte packets
 1  172.24.240.1 (172.24.240.1)  0.306 ms  0.261 ms  0.227 ms
 2  192.168.15.254 (192.168.15.254)  10.526 ms  10.510 ms  9.947 ms
 3  10.10.10.1 (10.10.10.1)  1.619 ms  1.961 ms  1.942 ms
 4  118.121.168.1 (118.121.168.1)  5.152 ms  5.144 ms  5.124 ms
 5  89.163.213.222.broad.dy.sc.dynamic.163data.com.cn (222.213.163.93)  4.039 ms  4.032 ms  89.163.213.222.broad.dy.sc.dynamic.163data.com.cn (222.213.163.89)  3.987 ms
 6  61.139.122.141 (61.139.122.141)  9.308 ms  61.139.122.117 (61.139.122.117)  19.377 ms  61.139.122.137 (61.139.122.137)  7.628 ms
 7  202.97.30.150 (202.97.30.150)  37.525 ms  37.857 ms  202.97.101.73 (202.97.101.73)  38.910 ms
 8  113.96.5.98 (113.96.5.98)  44.586 ms  113.96.5.70 (113.96.5.70)  42.207 ms  113.96.5.110 (113.96.5.110)  44.479 ms
 9  94.96.135.219.broad.fs.gd.dynamic.163data.com.cn (219.135.96.94)  74.287 ms  73.991 ms  102.96.135.219.broad.fs.gd.dynamic.163data.com.cn (219.135.96.102)  47.090 ms
10  102.96.135.219.broad.fs.gd.dynamic.163data.com.cn (219.135.96.102)  50.989 ms  14.215.32.110 (14.215.32.110)  37.794 ms  121.14.67.130 (121.14.67.130)  40.822 ms
11  *  *  *
12  *  *  *
13  *  *  *
```

traceroute in Ubuntu

2.4 Send yourself an email

Now that you know how to fetch a Web page, it's time to send an email message, again using a reliable byte stream to a service running on another computer. Below will use QQ mailbox to study.

1. Preparatory work: Log in to QQ Mailbox and enable the POP3/SMTP service as well as record the authorization code as Mosaic overlays. Convert QQ account and authorization code into Base64 code. This step can be completed through Baidu search online Base64 encoding conversion.



2. In one terminal window, run `telnet smtp.qq.com smtp` or `telnet smtp.qq.com 25`. The "smtp" service refers to the Simple Mail Transfer Protocol, used to send email messages. If all goes well, you will see:

```
cs144@cs144vm:~$ telnet smtp.qq.com smtp
Trying 203.205.232.7...
Connected to smtp.qq.com.
Escape character is '^]'.
220 newxmesmtplogicsvrszc8.qq.com XMail Esmtp QQ Mail Server.
```

3. First step: identify your computer to the email server. `HELO mycomputer` . Wait to see something like "250-newxmesmtplogicsvrszc8.qq.com-9.46.31.207-6318557"
4. Next step: Log in to QQ mailbox during the session. Type `auth login` , wait to see "334 VXNlcm5hbWU6", "334 VXNlcm5hbWU6" means "Username:". Type the base64 encoded QQ account, wait to see "334 UGFzc3dvcmQ6". Type the based64 encoded authorization code. Wait to see "235 Authentication successful".
5. Next step: who is sending the email? Type `mail from:<YourQQaccount@qq.com>` . If all goes well, you will see "250 OK".
6. Next: who is the recipient? For starters, try sending an email message to yourself. Type `rcpt to:<YourQQaccount@qq.com>` . If all goes well, you will see "250 OK".
7. It's time to upload the email message itself. Type `data` to tell the server you're ready to start. If all goes well, you will see "354 End data with <CR><LF>.<CR><LF>.".
8. Now you are typing an email message to yourself. First, start by typing the headers that you will see in your email client. Leave a blank line at the end of the headers.

From: YourQQaccount@qq.com

To: YourQQaccount@qq.com ↵

Subject: Hello computer network! ↵

9. Type the body of the email message—anything you like. When finished, end with a dot on a line by itself: . ↵. Expect to see something like: "250 Ok: queued as"
10. Type **QUIT** ↵ to end the conversation with the email server. Check your inbox and spam folder to make sure you got the email.

3 Writing a network program using an OS stream socket

In the next part of this warmup lab, you will write a short program that fetches a Web page over the Internet. You will make use of a feature provided by the Linux kernel, and by most other operating systems: the ability to create a reliable bidirectional byte stream between two programs, one running on your computer, and the other on a different computer across the Internet (e.g., a Web server such as Apache or nginx, or the netcat program).

This feature is known as a stream socket. To your program and to the Web server, the socket looks like an ordinary file descriptor (similar to a file on disk, or to the stdin or stdout I/O streams). When two stream sockets are connected, any bytes written to one socket will eventually come out in the same order from the other socket on the other computer.

In reality, however, the Internet doesn't provide a service of reliable byte-streams. Instead, the only thing the Internet really does is to give its "best effort" to deliver short pieces of data, called Internet datagrams, to their destination. Each datagram contains some metadata (headers) that specifies things like the source and destination addresses--what computer it came from, and what computer it's headed towards--as well as some payload data (up to about 1,500 bytes) to be delivered to the destination computer.

Although the network tries to deliver every datagram, in practice datagrams can be (1) lost, (2) delivered out of order, (3) delivered with the contents altered, or even (4) duplicated and delivered more than once. It's normally the job of the operating systems on either end of the connection to turn "best effort datagrams" (the abstraction the Internet provides) into "reliable byte streams" (the abstraction that applications usually want).

The two computers have to cooperate to make sure that each byte in the stream eventually gets delivered, in its proper place in line, to the stream socket on the other side. They also have to tell each other how much data they are prepared to accept from the other computer, and make sure not to send more than the other side is willing to accept. All this is done using an agreed-upon scheme that was set down in 1981, called the Transmission Control Protocol, or TCP.

In this lab, you will simply use the operating system's pre-existing support for the Transmission Control Protocol. You'll write a program called "webget" that creates a TCP stream socket, connects to a Web server, and fetches a page—much as you did earlier in this lab. In future labs, you'll implement the other side of this abstraction, by implementing the Transmission Control Protocol yourself to create a reliable byte-stream out of not-so-reliable datagrams.

3.1 Let's get started—fetching and building the starter code

1. The lab assignments will use a starter codebase called "Sponge." In your workspace of ubuntu, open a terminal and run **git clone your own repository url** to fetch the source code for the lab.(every one

of you will be invited into our [github classroom](#), and you will get a copy of the project as a repository of your own. More details can be found in [git_tutorial.md](#) in our QQ group.)

2. Enter the Lab 0 directory: `cd sponge`
3. Create a directory to compile the lab software: `mkdir build`
4. Enter the build directory: `cd build`
5. Set up the build system: `cmake ..`
6. Compile the source code: `make` (you can run `make -j4` to use four processors).

3.2 Modern C++: mostly safe but still fast and low-level

The lab assignments will be done in a contemporary C++ style that uses recent (2011) features to program as safely as possible. This might be different from how you have been asked to write C++ in the past. For references to this style, please see the C++ Core Guidelines (<http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines>).

The basic idea is to make sure that every object is designed to have the smallest possible public interface, has a lot of internal safety checks and is hard to use improperly, and knows how to clean up after itself. We want to avoid “paired” operations (e.g. malloc/free, or new/delete), where it might be possible for the second half of the pair not to happen (e.g., if a function returns early or throws an exception). Instead, operations happen in the constructor to an object, and the opposite operation happens in the destructor. This style is called “Resource acquisition is initialization,” or RAII.

In particular, we would like you to:

- Use the language documentation at <https://zh.cppreference.com/w/cpp> as a resource.
- Never use malloc() or free().
- Never use new or delete.
- Essentially never use raw pointers (*), and use “smart” pointers (unique ptr or shared ptr) only when necessary. (You will not need to use these in Our labs.)
- Avoid templates, threads, locks, and virtual functions. (You will not need to use these in Our labs)
- Avoid C-style strings (char *str) or string functions (strlen(), strcpy()). These are pretty error-prone. Use a std::string instead.
- Never use C-style casts (e.g., (FILE *)x). Use a C++ static cast if you have to (you generally will not need this in Our labs).
- Prefer passing function arguments by const reference (e.g.: const Address & address).
- Make every variable const unless it needs to be mutated.
- Make every method const unless it needs to mutate the object.
- Avoid global variables, and give every variable the smallest scope possible.
- Before handing in an assignment, please run make format to normalize the coding style.

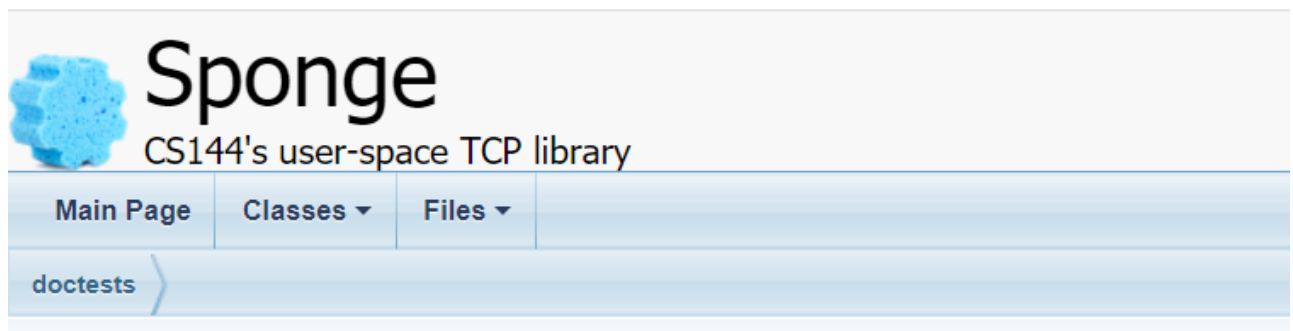
On using Git: The labs are distributed as Git (version control) repositories—a way of documenting changes, checkpointing versions to help with debugging, and tracking the provenance of source code. Please make frequent small commits as you work, and use commit messages that identify what changed and why. The Platonic ideal is that each commit should compile and should move steadily towards more and more tests passing. Making small “semantic” commits helps with debugging (it’s much easier to debug if each commit

compiles and the message describes one clear thing that the commit does) and protects you against claims of cheating by documenting your steady progress over time—and it's a useful skill that will help in any career that includes software development. The graders will be reading your commit messages to understand how you developed your solutions to the labs. If you haven't learned how to use Git, please do ask for help at Our course office hours or consult a tutorial (e.g., <https://guides.github.com/introduction/git-handbook> or <https://www.liaoxuefeng.com/wiki/896043488029600>). Finally, you are welcome to store your code in a private repository on GitHub, GitLab, Bitbucket, etc., but please make sure your code is not publicly accessible.

3.3 Reading the Sponge documentation

To support this style of programming, Sponge's classes wrap operating-system functions (which can be called from C) in "modern" C++.

1. You can find the documentation of [sponge](https://cs144.github.io/doc/lab0/index.html) from <https://cs144.github.io/doc/lab0/index.html>. You will see:



Click [Classes](#) and you will get:

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

▼ C Address	Wrapper around IPv4 addresses and DNS operations
C Raw	Wrapper around sockaddr_storage
C Buffer	A reference-counted read-only string that can discard bytes from the front
C BufferList	A reference-counted discontinuous string that can discard bytes from the front
C BufferViewList	A non-owning temporary view (similar to <code>std::string_view</code>) of a discontinuous string
C ByteStream	An in-order byte stream
▼ C EventLoop	Waits for events on file descriptors and executes corresponding callbacks
C Rule	Specifies a condition and callback that an EventLoop should handle
▼ C FileDescriptor	A reference-counted handle to a file descriptor
C FDWrapper	A handle on a kernel file descriptor
C gai_error_category	Error category for <code>getaddrinfo</code> and <code>getnameinfo</code> failures
C InternetChecksum	The internet checksum algorithm
C LocalStreamSocket	A wrapper around Unix-domain stream sockets
C NetParser	
C NetUnparser	
C Socket	Base class for network sockets (TCP, UDP, etc.)
C tagged_error	<code>Std::system_error</code> plus the name of what was being attempted

You need to use some of these APIs in your **webget** assignment, so you can read this document to learn how to use them, and for more information, please refer to the source code which can be found in **path_to_sponge/lib.sponge/util** directory.

2. Pay particular attention to the documentation for the **FileDescriptor**, **Socket**, **TCPSocket**, and **Address** classes. (Note that a **Socket** is a type of **FileDescriptor**, and a **TCPSocket** is a type of **Socket**.)
3. Now, find and read over the header files that describe the interface to these classes in the **lib.sponge/util** directory: **file_descriptor.hh**, **socket.hh**, and **address.hh**.

3.4 Writing webget

It's time to implement **webget**, a program to fetch Web pages over the Internet using the operating system's TCP support and stream-socket abstraction—just like you did by hand earlier in this lab.

1. From the build directory, open the file **../apps/webget.cc** in a text editor or IDE.
2. In the `get URL` function, find the comment starting `“// Your code here.”`
3. Implement the simple Web client as described in this file, using the format of an HTTP (Web) request that you used earlier. Use the **TCPSocket** and **Address** classes.
4. Hints:
 - Please note that in HTTP, each line must be ended with `“\r\n”` (it's not sufficient to use just `“\n”` or `endl`).

- Don't forget to include the "Connection: close" line in your client's request. This tells the server that it shouldn't wait around for your client to send any more requests after this one. Instead, the server will send one reply and then will immediately end its outgoing bytestream (the one from the server's socket to your socket). You'll discover that your incoming byte stream has ended because your socket will reach "EOF" (end of file) when you have read the entire byte stream coming from the server. That's how your client will know that the server has finished its reply.
 - Make sure to read and print all the output from the server until the socket reaches "EOF" (end of file)—**a single call to read is not enough**.
 - We expect you'll need to write about ten lines of code.
5. Compile your program by running `make`. If you see an error message, you will need to fix it before continuing.
 6. Test your program by running `./apps/webget www.njucn2022.top /ALPHA`. How does this compare to what you see when visiting `www.njucn2022.top` in a Web browser? How does it compare to the results from Section 2.1? Feel free to experiment—test it with any http URL you like!
 7. When it seems to be working properly, run `make check_webget` to run the automated test. Before implementing the get URL function, you should expect to see the following:


```
1/1 Test #28: t_webget .....***Failed    0.01 sec

Function called: get_URL(www.njucn2022.top, /check_ans/ans).

Warning: get_URL() has not been implemented yet.

ERROR: webget returned output that did not match the tests expectations
After completing the assignment, you will see:
1/1 Test #28: t_webget ..... Passed    0.10 sec

100% tests passed, 0 tests failed out of 1
```
 8. The graders will run your webget program with a different hostname and path than make check runs—so make sure it doesn't only work with the hostname and path used by make check.

4 Submit

1. In your submission, please only make changes to `webget.cc`. Please don't modify any of the tests or the helpers in `libsponge/util`.
2. Before handing in any assignment, please run these in order:
 - (a) `make format` (to normalize the coding style)
 - (b) `make` (to make sure the code compiles)
 - (c) `make check_webget` (to make sure the automated tests pass)
3. Write a report in `writeups/lab0.md`, then export it as `pdf` format, so finally there should be two files in `writeups` directory: `lab0.md` and `lab0_report.pdf`. This file should be a roughly 3-4 pages document with no more than 80 characters per line to make it easier to read. The report should contain the following sections:
 - **Program Structure and Design:** Describe the high-level structure and design choices embodied in your code. You do not need to discuss in detail what you inherited from the starter code. Use this as an

opportunity to highlight important design aspects and provide greater detail on those areas for your grading TA to understand. You are strongly encouraged to make this writeup as readable as possible by using subheadings and outlines.

- **Implementation Challenges:** Describe the parts of code that you found most troublesome and explain why. Reflect on how you overcame those challenges and what helped you finally understand the concept that was giving you trouble. How did you attempt to ensure that your code maintained your assumptions, invariants, and preconditions, and in what ways did you find this easy or difficult? How did you debug and test your code?
 - **Remaining Bugs: Submit your test screenshots** from `make check_webget`. Point out and explain as best you can any bugs (or unhandled edge cases) that remain in the code.
4. Please also fill in the number of hours the assignment took you and any other comments.
 5. When ready to submit, please follow the instructions in Our QQ group. Please make sure you have committed everything you intend before submitting. We can only grade your code if it has been committed.
 6. Please let the course staff know ASAP of any problems at the Friday-afternoon lab session, or by posting a question on QQ group. Good luck and welcome to NJU networking lab!