Computer Networks X_400487
Lab Guide
Vrije Universiteit Amsterdam

2022-04-06

This document was compiled on 2022-04-06 16:27:11Z, and may be updated during the course.
Keep an eye on the Canvas page for notifications of such updates.

This lab guide is created for the 2021 edition of Computer Networks at the Vrije Universiteit Amsterdam. It is written and edited by Nour Saffour, Yann Regev, and Jesse Donkervliet. The mandatory socket assignment is based on a lab assignment created by Sebastian Österlund. The other assignments are created and/or edited by Dennis Andriesse, Giulia Frascaria, Leon Overweel, Mitchel Olsthoorn, Sebastian Österlund, and Jesse Donkervliet.

Contents

1 Lab Overview

Part of becoming a Computer Scientist is equipping yourself with the practical skills you will need to solve real-world problems, and computer programming is one of these skills. In this lab, you learn how to build networked applications and program using sockets. This guide describes all available lab assignments. Completing a lab assignment earns you points for the course. Every assignment lists the number of points it is worth in its title.

In the mandatory lab assignment, you will implement a simple chat application. You can find the description of this assignment in Section 3.1, and a full description of the chat-protocol in Appendix A. After you complete this assignment, you can create your own simple networked applications using sockets and TCP/IP.

You must pass the mandatory assignment to pass the course.

2 Setup

This section helps you set up the tools you will need to complete the lab assignments. If you run into problems installing the required software, you can use the virtual machine (VM) provided on Canvas. The VM comes with the required software preinstalled.

The assignments are to be completed using the Python 3 programming language. Please download and install Python 3 via its website[1] or your package manager. Once you have Python installed, you can test that it works by typing `python` on the command line (Unix, MacOS), or opening the IDLE Python interpreter (Windows).

Once you have Python3 installed, you may want to install a text editor or integrated development environment (IDE). We recommend Visual Studio Code,[2] but many other good editors exist for Python.

Python in a interpreted language, which means you do not need to compile your program to machine instructions before running it. You can run your code directly from your IDE, or by passing it as an argument to Python by running "`python3 path-to-python-script.py`."

3 Assignments

This section describes the Computer Networks lab assignments. The first assignment is mandatory, and *must* be completed to pass the course. The other assignments are optional. Completing assignments earns you points. Every assignment lists its reward in the title. Some assignments have a variable reward. Read their description for more details.

Each assignment must be approved by a TA during a lab session, and handed in on Canvas. The deadline for the assignments can be found on Canvas and in the course syllabus. The TAs are present during the lab sessions to provide support for the mandatory assignment. You are expected to complete the other assignment without external help, but TAs are there to provide support in extreme cases.

It is not allowed to use third-party packages, such as those found on PyPI, for the lab assignments.

[1] `https://www.python.org/`
[2] `https://code.visualstudio.com/`

3.1 Mandatory Assignment: Chat Client (0-150 points)

In this assignment, you implement a text-based chat client. For this, you use Python, sockets, and the Transmission Control Protocol (TCP). Once you are comfortable using the socket interface, using sockets in other programming languages should be straightforward.[3] After completing this assignment, you will be able to exchange messages with your fellow students using your own client.

The chat client and chat server are built on top of TCP and use their own *application-layer protocol*. This means the client and server can request the other to perform actions by sending predefined types of messages. For example, your client can ask the server to forward a message to another user's client by sending the string "`SEND username insert your message here\n`" to the server, where "username" is the user to whom you want to send your message. If your message is correct, the server will send two messages: one to the destination user, forwarding your message, and one back to you that says "`SEND-OK\n`", to notify you that your message was forwarded successfully.[4] The full details of the protocol are listed in Appendix A.

Similar to Web browsers and other modern applications, your chat client does not expose the protocol it uses to the user. Instead, it provides a user-friendly text-based interface that makes it easy for users to chat with others without knowing the protocol specifications. The specifications of this interface, and the requirements of this assignment, are listed below.

Requirements
Your application must:

1. Implement the chat protocol described in Appendix A.
2. Connect to the chat server and let the user log in using a unique name.
3. Ask for another name if the chosen name is already taken.
4. Let the user shutdown the client by typing `!quit`.
5. Let the user list all currently logged-in users by typing `!who`.
6. Let the user send messages to other users by typing `@username message`.
7. Receive messages from other users and display them to the user.

The chat server is hosted by the teaching team. Its address can be found on Canvas. For testing purposes, a special user called *echobot* is always online. Echobot is a chat bot that replies to all your messages by sending back the same message it receives.

Evaluation
A teaching assistant will ask you questions about your code and check its correctness. The reward obtained for this exercise is based on when you complete the assignment.

150 points	Complete the assignment before or during week 3.
100 points	Complete the assignment during week 4.
50 points	Complete the assignment during week 6.

Resources
You can find the full protocol used by server and clients in Appendix A. References for programming with sockets and threads can be found in Appendices B and C respectively.

[3] Famous last words.
[4] Make sure that you receive this OK before asking the user for the next message to send.

3.2 Chat Server (250 points)

In the chat client assignment, you used a server hosted by the teaching team. This server connects multiple clients and forwards messages between them. In this assignment, you implement your own chat server using the same protocol.

Unlike the client, the server is likely to have multiple open connections at the same time—one for each client that is connected to it. Because it is impossible to predict when a client will send a request or message, your server needs to keep checking all connections for incoming data. Both polling and multi-threading are allowed as solutions to this problem.

Requirements

1. Support the full protocol specified in Appendix A.
2. Support at least 64 simultaneous clients.

Evaluation
The TAs will ask you questions about your code and check its correctness, possibly by chatting with each other using your server.

Resources
See the resources listed in Section 3.1.

3.3 Chat Client for Unreliable Networks (500 points)

In the mandatory assignment, you implemented a simple chat client. Because the interfaces and protocols of computer networking are already well-established, there are many challenges that you do not need to take into account. For example, once a connection is established, TCP provides a stream of bytes in the order they were sent and without error. This assignment asks you to give up such conveniences and instead provide solutions to some of these challenges yourself.

Assignment Description
The Internet Protocol (IP) is an unreliable datagram protocol. Fortunately for many application programmers, the Transport Control Protocol (TCP) runs on the hosts and hides the unreliability of the network and underlying protocols.

Implement a chat client that uses UDP and the protocol shown in Appendix A. This chat client must be able to communicate to similar clients via the Unreliable Chat Server, whose address can be found on Canvas. You can configure the Unreliable Chat Server to simulate an unreliable network by letting it drop a fraction of the received messages, insert errors into messages, or even change their order. Use this to test the correctness of your chat client. The commands used to configure the server are listed in Appendix D.

Your chat client should meet all the requirements listed in Section 3.1, as well as the requirements below.

Requirements

1. Use UDP instead of TCP.
2. Guarantee delivery of messages using acknowledgments.
3. Protect against errors in the message. The client must detect at least single, double, and triple bit errors.
4. Messages are delivered in the order they are sent.
5. The interface shown to the user does not differ from the one in the mandatory assignment.

Assessment
Show that your chat client hides the unreliability of the network from the user. Configure the Unreliable Chat Server and send messages between two instances of your client.

3.4 Botnet Reverse-Engineering (250—750 points)

A botnet is a network of malware-infected computers. The malware makes the computer respond to remote commands send by an attacker. This attacker could, for example, instruct these computers to all at once request a resource-intensive service from a single provider, executing a Distributed Denial of Service Attack (DDoS).

Assignment Description

For clarity, we introduce two new terms. Firstly, we use "botnet executable" for the malicious code that is running on an infected computer. Secondly, we use "botnet control server" for the machine that sends malicious commands to these computers.

In this assignment, we give you a botnet executable. It is your task to reverse engineer the protocol this executable uses to communicate with the botnet control server. Fortunately, this particular botnet was made by us, and is not malicious, so you can safely run it on your machine. Your task is to run the botnet executable, capture its traffic while it is connected to the server, and then analyze the captured traffic to figure out how the botnet protocol works. This is a common task for real-world malware analysts.

Requirements

1. Capture network traffic from the botnet executable.
2. Reverse engineer the protocol used.
3. Answer the following questions about the botnet in your report:
   (a) Which IP address and port number are used by the command server?
   (b) What transport layer protocol is used by the botnet?
   (c) What is the version number of the given bot client?
   (d) The botnet supports 4 different commands (excluding the `hidden` command, which you don't need to describe here). What are they?
   (e) (250 points, *difficult puzzle*) The protocol includes a single encrypted message type, which is sent to the server after a certain kind of command is received. You may have to run the bot multiple times to see the encrypted message. Try to find out how this message is encrypted, and then decrypt it. Which encryption algorithm is used, and what is used as the key? Please report to us a single decrypted message (copied verbatim, including the checksum field), and the key that you used to decrypt it. Briefly describe the structure of the message, and how you went about cracking the encryption.
   (f) (250 points, *difficult puzzle*) The protocol also includes a special kind of command which contains a hidden message. This message is sent to the bot by the server after a `COMMAND` request. You may have to run the bot multiple times to see the message. The message includes a mysterious payload in which the server has hidden secret data (hint: the secret data is in plaintext). What does this payload represent, and how is the data hidden within it? Please describe how you figured this out. Also send us the hidden data (plaintext) which you recovered from the payload.

Evaluation
The TA will first ask you to run the original executable, capture its network traffic, and explain the messages that are exchanged. Afterwards, the TA will discuss the answers you provided to the questions listed in the requirements.

Resources

Botnet executable Download the executable from Canvas. We provide both a Windows (32-bit/64-bit) and an Ubuntu Linux (32-bit/64-bit) binary. Choose the one that matches your platform. If there is no binary matching your platform, you can run the 64-bit Ubuntu Linux executable using the Virtual Machine provided on Canvas.

Wireshark Download and install Wireshark from `http://www.wireshark.org/download.html`. Get familiar with its basic workings. Try capturing packets from an interface by navigating to the "Capture" menu and choosing "Interfaces". Click "Start" to begin capturing packets. Make sure there is some network activity so that there are packets for Wireshark to capture. Click "Stop" in the "Capture" menu to stop capturing packets. You should see several lines of captured packets. Click some of the packets and examine them. Once you are familiar with the basic workings of Wireshark, move on to the assignment.

3.5 DIY DNS Server (750 points)

How many web pages do you visit in a day? Would you be able to remember all their IP addresses? Probably not. It turns out that humans are bad at remembering arbitrary sequences of numbers, but reasonably good at remembering names. By assigning names to IP addresses, web browsing becomes doable for humans. Instead of having to remember the sequence "216.58.211.110," you only need to remember "www.google.com." Your browser automatically translates this into the correct IP address.[5]

However, this automated translation increases the complexity of the system, which now needs to translate a name into an IP address before it can establish a connection. It would be infeasible for every computer to keep a local, up-to-date copy, of all name-to-address mappings. Instead, computers depend on a globally distributed system called the Domain Name System (DNS) to look up these mappings dynamically.

This system contains a large hierarchy of servers called *DNS servers*. A DNS server is a computer that keeps track of IP addresses and their associated domain names. Other DNS servers can ask it for the IP address matching a certain domain name. It then resolves, or translates, this name into an IP address by looking it up in its local database, or by contacting other DNS servers higher up in the hierarchy.

Assignment Description
DNS servers communicate with each other using their own protocol. It is your job to implement your own DNS server that adheres to this protocol and performs recursive queries. Start by reading the official specification, RFC 1035.[6] The RFC mentions in detail the request formats, the queries that you will receive, and many other valuable information.

Requirements
Typically, a client application forwards a domain name to the operating system, which in turn forwards it to a DNS server that performs recursive queries. Performing a recursive query means that the DNS server will query other DNS servers until it finds the address that belongs to the given domain name. This address is then returned to the application via the operating system.

Your task is to implement your own DNS server with the following requirements:

1. Perform recursive DNS queries.
2. Handle requests from multiple operating systems.
3. Implement the RTT (Round Trip Time) algorithm to choose a name server.
4. Implement the caching policy specified in the RFC section "7.4. Using the cache"
5. Handles mail exchange servers requests.

[5] Another, more compelling, reason not to use IP addresses to identify Web pages is that it makes the assumption a Web page is tied to a single machine, or network device. More generally, it creates a dependency between an entity on one layer, and the implementation of a lower layer. Naming entities on their own layer solves this problem, but the Domain Name System (DNS) does not do this. Instead, it translates a human-readable domain name into an Internet address. I.e., it simply provides global, mutable, and easy-to-remember aliases for network devices. The Web browser has to guess the right transport-layer address (port number) to find the right entity. This system works because it relies on hard-coded port numbers. If you want to run your Web server on a different port number, users have to enter it in their browser manually. This shows that, although we have DNS, their is still a dependency between implementations across layers. This is not so much a shortcoming of DNS, as a shortcoming of the design of the upper layers of the Internet.

[6] See `https://www.ietf.org/rfc/rfc1035.txt`.

Evaluation
Your implementation is evaluated by the TAs. To test your server implementation, you can configure your Web browser or operating system to use it as its DNS server. Make sure that your server is able to handle requests from different operating systems. Your implementation must resolve requests by communicating with the root server and the servers it lists in its replies. You cannot pass the assignment if you simply forward requests to another DNS server that performs recursive queries.

Resources
Below is a list of free, popular, and public DNS servers. You can analyze their responses to learn more about how to implement your own server.

Google (`8.8.8.8` and `8.8.4.4`)

Quad9 (`9.9.9.9` and `149.112.112.112`)

OpenDNS (`208.67.222.222` and `208.67.220.220`)

You can find the addresses of the DNS root servers at `https://www.iana.org/domains/root/servers`.

3.6 Online-Multiplayer Game (1000 points)

Online games can be highly demanding of computer networks. They require both high bandwidth and low latency communication, often between large groups of players.

Assignment Description
In this assignment, you design, implement, and demo your own online-multiplayer game. To participate in this assignment, you first need to write a plan for your game and get it approved. Your plan must at least contain the following three parts:

1. A description of the game itself, and how it is played.
2. A description of the requirements of the game, and the resulting requirements on the network. To get your plan approved, you must show that these requirements pose sufficient networking challenges. Examples of such challenges include supporting large numbers of concurrent users, automatically recovering from network failures, and latency compensation techniques for real-time games. We especially appreciate advanced and new ideas. Do not be afraid to be creative!
3. A description of your approach, and how this will meet your requirements.

Discuss your plan with the teacher before getting started. Plans are only approved if their requirements are sufficiently complex. If you plan to complete this assignment, submit your game plan several weeks before the deadline and assume that it will require several rounds of feedback to be approved. If your plan is not approved, you cannot complete this assignment.

Requirements

1. Submit your plan on Canvas.
2. Your plan must be approved by the teacher.
3. Build your game such that it meets the agreed-upon requirements.

Evaluation
Demo your game to, or play your game with(!), the TA. Show the TA the source code and explain how your game works, how your game meets its requirements, and how you solved challenges encountered along the way.

A Chat Application Protocol

Message	Sent by	Description
`HELLO-FROM <name>\n`	Client	First hand-shake message.
`HELLO <name>\n`	Server	Second hand-shake message.
`WHO\n`	Client	Request for all currently logged-in users.
`WHO-OK <name1>,...,<namen>\n`	Server	A list containing all currently logged-in users.
`SEND <user> <msg>\n`	Client	A chat message *for* a user. Note that the message cannot contain the newline character, because it is used as the message delimiter.
`SEND-OK\n`	Server	Response to a client if their 'SEND' message is processed successfully.
`UNKNOWN\n`	Server	Sent in response to a SEND message to indicate that the destination user is not currently logged in.
`DELIVERY <user> <msg>\n`	Server	A chat message *from* a user.
`IN-USE\n`	Server	Sent during handshake if the user cannot log in because the chosen username is already in use.
`BUSY\n`	Server	Sent during handshake if the user cannot log in because the maximum number of clients has been reached.
`BAD-RQST-HDR\n`	Server	Sent if the last message received from the client contains an error in the header.
`BAD-RQST-BODY\n`	Server	sent if the last message received from the client contains an error in the body.

Table 1: Chat Application Protocol. Angular brackets (<>) indicate variable content.

B Socket Interface Reference

This appendix lists multiple functions from Python's socket library that can help you get started. You can find more extensive documentation at `https://docs.python.org/3/library/socket.html`.

socket

You can create a new socket using the The `socket` function. You can use this socket to either initiate, or respond to, a connection request. The parameters specify the network-layer and transport-layer protocol.

```
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

connect

Establish a connection using a socket. The parameter is a tuple specifying the network-layer and transport-layer address. The first parameter is a string containing an IP-address or a domain name. The second parameter is an integer containing the port number.

```
host_port = ("127.0.0.1", 4321)
sock.connect(host_port)
```

send

`send` sends bytes over a socket. Depending on the number of bytes that need to be sent, the available buffer space at the sender, and number of bytes that can be accepted at the receiver, it may not be possible to send the complete buffer at once. Therefore, the function sends whatever it can and returns the number of bytes it sent. It is up to the programmer to keep calling `send` until all bytes are sent. Alternatively, you can use `sendall` to let Python do this for you.

```
string_bytes = "Sockets are great!".encode("utf-8")
bytes_len = len(s)
num_bytes_to_send = bytes_len
while num_bytes_to_send > 0:
    # Sometimes, the operating system cannot send everything immediately.
    # For example, the sending buffer may be full.
    # send returns the number of bytes that were sent.
    num_bytes_to_send -= sock.send(string_bytes[bytes_len-num_bytes_to_send:])

# sendall calls send repeatedly until all bytes are sent.
sock.sendall(string_bytes)
```

recv

`recv` receives bytes over a socket and returns a buffer containing these bytes. This function is blocking, meaning the function will wait until there are bytes available.

```
# Waiting until data comes in
# Receive at most 4096 bytes.
data = sock.recv(4096)
if not data:
    print("Socket is closed.")
else:
    print("Socket has data.")
```

Exceptions

Python uses exceptions to handle errors. You can handle errors by placing function calls which can return an error in try-except blocks.

```
try:
    sock.send("how to handle errors?".encode("utf-8"))
    answer = sock.recv(4096)
except OSError as msg:
    print(msg)
```

C Threading Interface Reference

This reference shows multiple functions from Python's threading library. For more extensive documentation, see the Python Docs website: `https://docs.python.org/3/library/threading.html`

thread

Creating a new thread is done using the `threading.Thread` function. When creating a new thread, you need to pass the function that needs to be executed in a separate thread, and the arguments that function requires. In the example below, we create a `Thread` object that will run the `print` function in a separate thread.

```
import threading
t = threading.Thread(target=print, args=("hello", "world"))
```

The args parameter must be a tuple. If the function only takes one argument, you can add an additional "," to create it.

```
>>> a = ("hello", "world")
>>> b = ("hello")
>>> c = ("hello",)
>>> type(a)
<class 'tuple'>
>>> type(b)
<class 'str'>
>>> type(c)
<class 'tuple'>
```

start

Run the function you specified in a new thread by calling `start`.

```
t.start()
```

join

Wait for the function and thread to fining by calling `join`.

```
t.join()
```

D Commands to Configure the Unreliable Chat Server

Message	Sent by	Description
`SET DROP <value>\n`	Client	Set message drop probability. Value between 0 and 1.
`SET FLIP <value>\n`	Client	Set bit flip probability. Value between 0 and 1.
`SET BURST <value>\n`	Client	Set burst error probability.
`SET BURST-LEN <lower> <upper>\n`	Client	Configure the burst error length. Default is 3.
`SET DELAY <value>\n`	Client	Set message delay probability.
`SET DELAY-LEN <lower> <upper>\n`	Client	Configure the delay length in seconds. Default is 5.
`GET <name>\n`	Client	Get the current value of a setting.
`VALUE <name> <value> (<upper>)\n`	Server	The name and value(s) of a setting.
`RESET\n`	Client	Set all values to their defaults.
`SET-OK\n`	Server	Value received and set.

Table 2: Chat Application Protocol Extension for Unreliable Chat Server. Default value is 0 unless indicated otherwise. Angular brackets (<>) indicate variable content.