Datanet 2013 Assignment 2

Danni Dromi Dídac Rodríguez Arbonès Hans Ienasescu Radu Dragusin
Marcos Vaz Salles

Deadline: 23:55 on May 12, 2013

This assignment is the second of four assignments to be handed in during this course. It is split into two parts, the first containing a set of theoretical exercises and the second detailing a practical assignment.

The assignment is due on May 12, 2013 at 23:55. It is to be handed in on the Absalon course page. It is your responsibility to make sure in time that the upload of your files succeeds. Email or paper submissions will not be accepted.

The assignment may be solved in groups of 2 or 3 students. We **strongly** encourage you to work in groups, and we have balanced the work load accordingly. Please contact your TA if you intend on solving it alone!

The assignment will be scored on a scale of 0-10 points. There will be **no resubmission** for this assignment. In order to participate in the exam, you must obtain a total of 24 points over the four assignments.
A well-formed solution to this assignment should include a PDF or plaintext file with answers to all exercises as well as questions posed in the programming part of the assignment, other formats will not be accepted. In addition, you must submit your code along with your written solution. The source code that is handed in should *only* contain `.py` files, no compiled python (`.pyc`) files. Your code should be compatible with Python version $\geq$ 2.6 && $<$ 3.0. If you have more than one source file please submit your assignment as a compressed folder, containing all source files.

Evaluation of the assignment will take both parts into consideration. Do not get hung up in a single question. It is best to make an effort on every question and write down all your solutions than to get a perfect solution for only one or two of the questions.

If you have questions about the exercises or the practical assignment, please use the discussion forum on Absalon so your fellow students may benefit from your questions.

1 Theoretical part

Each of the following sections deals with topics covered by the lectures and, to some extent, the TA sessions.

Each section contains a number of questions that should be answered **briefly** and **precisely**. Most of the questions can be answered in 2 sentences or less. We have annotated questions that demand longer answers, or figures with a proposed answer format.

Miscalculations are more likely to be accepted, if you account for your calculations in your answers.

1.1 Domain Name System

1.1.1 DNS provisions

Three of the most important goals of DNS[1] are to ensure fault tolerance, scalability and efficiency. Explain how these ensurances can (and are) met in practice. *Hint: Look at figure 2.23 of the K&R book and the resource record (RR) format — (Answer with 2-4 sentences)*

1.1.2 DNS lookup and format

Part 1: Explain the advantages of the `CNAME` type records. Explain how DNS may provide simple load balancing among servers. *(Answer with 2-4 sentences)*

Part 2: Many DNS servers, especially *root* and *top level domain* (TLD) servers, respond with 'iterative' replies to recursive requests. Explain the differences between iterative and recursive lookups and when and why recursive lookups are justified. *(Answer with 4-8 sentences)*

Part 3: In this question, you are asked to give a detailed description of a DNS lookup, performed as a recursive request by a client to a recursive DNS resolver. You may use `dig` to supply ressource records or make them up. However it is important, that your answer is consistent.

Assume the aforementioned DIKU student wants to browse the website of `grooveshark.com`. Assume that neither the client nor recursive resolver has any cache RR on the domain. Now assume the student types in `grooveshark.com` into the URL bar and the browser sends a DNS request. As recursive resolver, she uses `ns2.censurfridns.dk` at 89.104.194.142. You may assume that it will contact the root server `a.root-servers.net` at 198.41.0.4.

Show all DNS requests and responses taking place. Be sure to include **at least one of each** different ressource record transmitted – be sure to understand the different types of ressource records contained in a response (see figure 2.23 of K&R). *(Answer with a diagram like fig. 2.21 of K&R. You may condense the packet contents as you see fit – e.g. no header fields are necessary – but be sure not to leave anything of importance out.)*

1.2 Transport protocols

The questions relating to TCP in this section does not assume knowledge of the flow and congestion control parts of TCP.

1.2.1 TCP reliability and utilization

Part 1: Is the 3-way-handshake in TCP really needed? Can you suffice with less? Explain why or why not.

Part 2: How does TCP facilitate a *full-duplex*[2] connection?

[1]As specified in RFC 1034 and RFC 1035, superseeding RFC 882 and RFC 883

[2]Full speed bidirectional data transfer

1.2.2 Reliability vs overhead

Part 1: Considering the data transferred over a network, how does a TCP connection add overhead relative to UDP?

Part 2: Explain the TCP notion of reliable data transfer. In which way are TCP connections reliable? Are packets **always** delivered?

1.2.3 Use of transport protocols

DNS part 1: Given your answer in question 1.1.2 above as well as your current knowledge of transport protocols, explain what transport protocol DNS employs and why.

DNS part 2: A number of attacks on DNS have been proposed, among others the *cache-poisoning* attack[3], where information guessed by the attacker is used to send fraudulent responses to recursive DNS resolvers. Explain how the use of TCP could have eliminated this attack. Explain also what issues may arise from this and how adverse the effects can be if for instance a DNS server of an ISP is succesfully attacked. *(Answer with 3-8 sentences, possibly including a scenario)*

Simple HTTP requests and responses Take a look at the HTTP request and response below in figure 1 and 2, and answer the following questions

```
GET / HTTP/1.1
Host: erdetsnartfredag.dk
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.7; en-US; rv:1.9.2.3)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip,deflate
Connection: close
If-Modified-Since: Thu, 22 Sep 2011 08:40:09 GMT
If-None-Match: "1b5801c-519-4ad83a3e72c40"
Cache-Control: max-age=0
```

Figure 1: Reload request

```
HTTP/1.1 304 Not Modified
Date: Fri, 27 Apr 2012 05:13:41 GMT
Server: Apache/2.2.3 (CentOS)
Connection: close
ETag: "1b5801c-519-4ad83a3e72c40"
```

Figure 2: 304 Response

The request contains 390 bytes and the response contains 146. Assume that *maximum transmission unit* (MTU) is 1460 B.

[3]Explained in great detail here: `http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html`

1. What is the minimal number of packets involved in exchanging the request and the response, looking from the TCP perspective?
2. What is the proportion of packets containing data to packets needed for connection handling? What could justify the use of TCP as transport protocol for HTTP?
3. Could UDP be used as a transport protocol for HTTP? Explain why or why not.

1.3 TCP: Principles and practice

Some of the questions below refer to one or more *Request For Comments* (RFC) memorandums posted by the *Internet Engineering Task Force* (IETF). In most cases, you can find answers to the question in the book, but we recommend that you browse through these RFCs to see how internet standards are formulated and distributed in practice.

Be careful when answering the questions below. If any ambiguity may arise, remember to specify the point of view of your answers – is it from the server or the client or both?

1.3.1 TCP headers

Part 1: The TCP segment structure is shown in figure 3.29 in K&R (or section 1.3 of RFC793[4]).

1. What is the purpose of the RST bit? Give an example of when a TCP stack may return a RST packet.
2. What does the sequence number and acknowledgement number headers refer to for a given connection? Is there any relation between server and client acknowledgement and sequence numbers?
3. What is the purpose of the window? What does a positive window size signify?
4. If the window size of a TCP receiver is 0, what happens then at the sender? How can the sender find out when the receiver window is not 0?

Part 2: In the previous assignment, you were asked why the 3-way handshake was necessary. One of the practical reasons for this 3-way-handshake was the synchronization of sequence numbers. In some situations, the client can initiate its transfer of data after having received the SYN-ACK-packet. Explain when this is possible. *(Hint: What does the server need to tell the client in the SYN-ACK-packet?)*

Part 3: Give a transmission diagram for a client connecting to a server, where the client first sends some data, then the server sends some data back to the client, then the client disconnects. Be sure to specify all relevant header fields for all segments as well as any connection handling segments. Your diagram should look like figure 3.37 of K&R. Include at least one, but non-disruptive, packet drop.

[4]`https://tools.ietf.org/html/rfc793#section-3.1`

1.3.2 High performance TCP

1. What is the maximum possible value of the receive window header? Which constraints does this impose on the throughput of a connection? *(Answer with a number and sentence)*

2. Given a 100Mb/s network, calculate the latency (RTT) at which the maximum standard receive window starts limiting the throughput. *(You should account for your calculations)*

3. RFC 1323[5] lists a number of features to improve the performance of TCP, among other the Window Scale option, that is set during connection setup. List a RFC1323-compliant option header that allows for full utilization of a 100Mb/s connection with an RTT of 72 ms.

1.3.3 Flow and Congestion Control

Part 1: Explain the type of congestion control that modern TCP implementations employ. Is it network assisted congestion control?

Part 2: Congestion control translates directly to limiting the transmission rate. How does TCP determine the transmission rate when doing congestion control?

Part 3: How does a TCP sender infer the value of the congestion window, that is, what is the basis for calculating the congestion window? *(Answer with at most 10 sentences)*

Part 4: The fast retransmit extension, as described in RFC 2581[6] allows for retransmission of packets after receiving 3 duplicate ACKs for the same segment. What does the sender need to implement in order to allow for fast retransmit? What does the receiver need to implement? If a sender supports fast retransmit and needs to do fast retransmit for a specific segment, how many ACKs should the sender receive for the previous packet?

2 Programming part

For the practical part of this assignment, you will continue to build on your work from the first assignment. The final goal is to implement a simple peer-to-peer chat service that uses a central name server to look up the addresses for each peer.

For this second assignment, you will be working on a new skeleton code, however you will be using the knowledge you gained from the first assignment.
You will have to implement a working basic centralised name server as well as a client that supports registering with the name server and looking up other clients.
The protocols for this system aren't the same as last time. They can be found in Appendix A.The second assignment, as well as third assignment, will be using the TCP model (ie. sockets are kept alive and if the die we assume the user has left the service and will have to reconnect).

The clients will connect to the name server by preforming a handshake, which requires the user to provide a nick name. For this assignment, it is not required for the name server to connect clients to one another, neither is it required for the clients to communicate with one another.

[5]TCP Extensions for High Performance: `http://www.ietf.org/rfc/rfc1323.txt`
[6]`https://tools.ietf.org/html/rfc2581`

For your convenience we will run a functional name server on one of the DIKU systems (further information will be made available on Absalon). This server is not constantly monitored, so, should it crash it may take some time for it to be restarted. However, since implementing the name server is one of your tasks, you should be able to complete the assignment even if the one we provide goes out of service.

For the practical part of assignment 2 you are expected to hand your Python source code and a document containing the answers to the questions as well as a short explanation of how you implemented your code (if your code has any shortcomings you should document them here).

2.1 Handout Code

The code for this assignment consists of the two files `peer.py` and `server.py`. Some of the functionality is already implemented (or partially implemented) in the handout code, however all of the tasks related to networking are up to you to solve. There are comments throughout the code explaining what you should implement and where.

For this assignment (like the last) we will be using Python's *low-level* socket interface.

3 The Client

The client you will be implementing this week isn't technically a peer (since beeing a peer requires you to act as both a client and a server), however it will be referenced as a peer in the code (for reasons that will become clear in the following assignment).

Even though this programming task focuses on the name server you will still have to implement a handfull of functions on the peer.

This client uses a simple IRC like input protocol shown in Appendix A.2. The client shouldn't try to connect to the name server before the user gives it the appropriate input. When connecting to the name server, a special handshake has to be preformed. If this handshake fails the connection will close and the user will have to try connecting again.
When connected, the user can ask the name server for a list of online users. The name server also support functionality for looking up user information, however this function shouldn't be avaliable to the user.

You will also need to implement minor changes to several methods in order for them to utilize sockets. As stated earlier, all of the programming tasks related to networking are up to you to solve.

4 The Name Server

The name server will be the main focus of this assignment. You will need to implement a name server that can handle the protocol shown in Appendix A.1

You should make sure the name server sets up the needed sockets at initialization (hint: make the address of the listening socket reusable. That way you won't have to wait for the socket to time out if the server crashes).

When the name server is running it should listen for new peers trying to connect to the chat service and existing peers sending queries to the name server. Furthermore it will need to keep an eye out for dead sockets. If a client program is killed or a socket dies, the name server should detect this and remove that socket from memory and close it.

Furthermore you will need to implement the name server's side of the handshake (shown in Appendix A.1).

Aside from implementing the peer/name server system you should answer the following questions:

Question 1
Is it easy to reverse engineer the protocol by just listening to traffic (packet sniffing)? Is there a (socket based) technique that could be used to make this more difficult to reverse engineer?

Question 2
Is your implementation robust? Could it easily be crashed? What if someone implements a 'malicious' peer, for example, a peer that never adds the message end-marker or tries to connect thousands of times? How would you prevent your peer from crashing?

Note: we do not expect you to submit a fully robust implementation, but it is an important aspect to consider.

Question 3
Is there a way for the name server to keep track of which peers are active and which are not? should the name server disconnect idle peers and if so, what should the peers do in order to let the name server know they are still active?

Question 4
Our current setup will be using a centralized name server even for the final peer-2-peer chat system. Is this a good idea? What advantages and disadvantages are there of running a centralized name server? How could we change the service so that the name server would be distributed?

A Protocol

This appendix describes the `peer ↔ name server` and `peer ↔ user` protocols.

A.1 PEER ↔ NAME SERVER

`HELLO <nick>`
To identify itself, a peer has to perform a simple 'handshake' protocol with the name server. The peer sends the command `HELLO` with the user's *nickname*
The name server may respond with:

`100 CONNECTED` Handshake successful, the peer is now connected to the name server.

`101 TAKEN` Handshake unsuccessful, the nickname was already taken. The name server directly closes the socket.

`102 HANDSHAKE EXPECTED` Received some information but not enough information for a proper handshake.

`LOOKUP <nick>`
Look up the address information for another peer identified by nickname.
The server may respond with:

`200 INFO <ip>` The user was found and the address information is given.

`201 USER NOT FOUND` The user was not found.

Note: This request is not directly accessed by the user in this assignment. It will be used in the third assignment for peer identification. Feel free to implement your own testing code for the user interface (i.e. /lookup NICK).

`USERLIST`
Request a list of all users currently online according to the server.
The server may respond with:

`300 INFO <num peers> <nickname> <ip addr>, ...` The first piece of information is the number of users that have registered. Following that is a comma separated list of *n* users. This list should not contain the peer making the query.

`301 ONLY USER` You are the only peer currently registered.

`LEAVE`
Tell the server that the connection can be closed and the user is no longer available for chatting.
The server may respond with:

`400 BYE` Connection closed, user logged out.

`500 BAD FORMAT`

The name server should give this reply to any request sent that doesn't match the protocol.

A.2 PEER ↔ USER

- /connect <IP> <PORT> - connect to name server and preform the handshake. IP should be the ip-address of the server and PORT the port that the server is listening on.The handshake is described in A.1
  The user should be prompted a message telling whether or not the connection succeded.
- /nick <NAME> - select a nick name. The user should get a confirming message. This command does not invoke any communication with the server. The nickname should not contain commas (',‘), because they will interfere with the protocol for USERLIST.
- /userlist - get a list of online users A.1. The user should get a list looking like:

```
Online Users:

Eric - You
Pete - 192.32.5.1
Shawn - 64.128.10.56
Lisa - 56.0.23.2.1
```

- /leave - close the connection with the name server. The user should get a confirming message A.1.
- /quit - close the peer application. The user should get a confirming message. This command does not invoke any communication with the server.