Datanet 2013 Assignment 1

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

Deadline: 23:55 on May 5, 2013

This assignment is the first 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 5, 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 ≥ 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 Store and Forward

The answers to the questions in this section should not make any assumptions on specific protocols or details pertaining to the different layers. You can answer these questions after having read chapter one.

1.1.1 Processing and delay

Explain, with one or two sentences, the key reason for delays in typical packet switched networks, besides physical constraints such as the propagation speed of different transmission medium.

1.1.2 Transmission speed

Consider the setup below in figure 1. A DIKU student is using a laptop at home, browsing the `diku.dk` website. The upstream connection speed is 2 Mb/s from the DSL modem at home to the DSLAM[1].

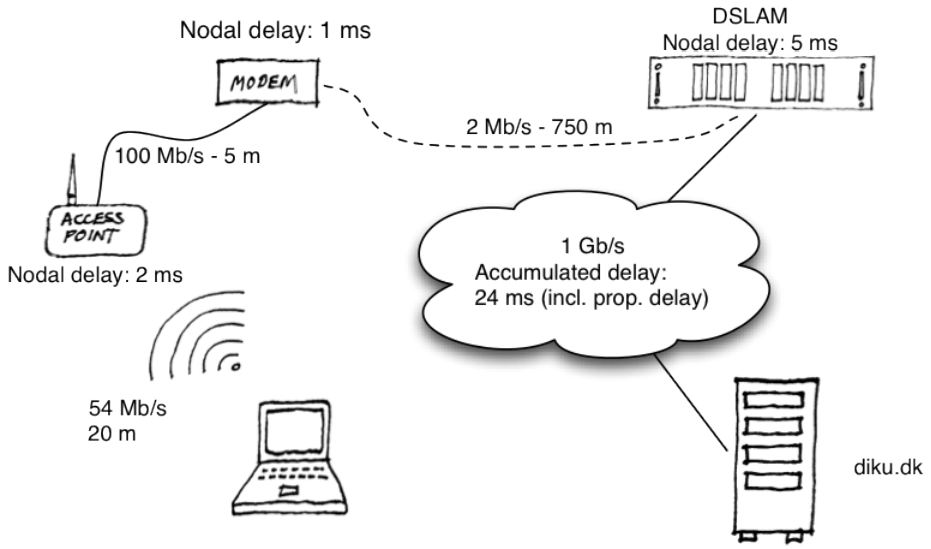

Figure 1: A typical DSL setup

Part 1 Given the information in figure 1, calculate the *round trip time* (RTT). For calculating propagation delay, assume that the propagation speed in all links visible is $2.4 \cdot 10^8$ m/s and the queuing delay is contained in the noted node delays. You may leave out the propagation delay, but explain why, if you do.

Part 2 Assume 640 KB of data is sent to the `diku.dk` webserver, including any overhead. Assume that the server acknowledges the upload when all bytes have been transfered with a single packet. Calculate the total transmission time, given the RTT calculated above.

[1] Digital Subscriber Line Access Multiplexer, used by Internet Service Providers to provide DSL connectivity over phone lines

1.2 Buffers and latency

Packet queuing happens when a network link is at capacity, i.e. when a router or switch that is forwarding packets is receiving packets faster than it can retransmit them. Routers and switches are equipped with memory for buffering, so that bursts of packages can be absorbed and transmitted later, if the output link is at capacity.

Part 1 In the above setup, assume that the DSL modem has a buffer of 512 KB and the wireless router has a buffer exceeding that. Thus we introduce a *variable* buffer delay. Given the upload speed of the link, what is the maximum buffer delay that a package can experience in this setup? What does this mean if the student in the above scenario sends a request when the buffer is almost full.

Part 2 The term *bandwidth-delay product* (BDP) is a rule of thumb for general buffer size, where delay is given as the RTT. Why is the BDP a good measure for adequate buffer size? What are the consequences of having a buffer size that does not match the BDP? *(Answer with 3-5 sentences)*

1.3 HTTP

1.3.1 HTTP semantics

HTTP employs a message format divided into header and body sections. The initial header of a request consists of a method field, a resource identifier field and a protocol version field. Likewise, the initial header of a response consists of a protocol version field, a status code and an optional message.

Part 1: What is the purpose of the method field in requests? How does `POST` and `GET` requests differ in practice?

Part 2: An additional (and mandatory) header is the `Host` header. Why is this header in general necessary?

Part 3: Try opening a `telnet` or `nc` connection to `google.com` on port 80. Make a simple `GET` request for `/`, including a protocol version and a `Host` header. How should the result be interpreted and what do you see, when you open `google.com` in a browser? Why is the response status code important? *(Answer with 2-4 sentences)*

1.3.2 HTTP headers and fingerprinting

Part 1: One of the additional header fields is `Set-cookie` (for responses) and `cookie`. What is the reason for these header fields and to what degree may they be used as an unique identifiers?

Part 2: Using Chrome developer tools, Firefox with Firebug, or another similar tool, or just browsing through section 2.2.3 of K&R, note some other headers that might give away sensitive information. Can anything be done to prevent spilling these identifiers?

Part 3: The `ETag` response header works in conjunction with the `If-None-Match` and `If-Match` request headers to prevent unnecessary page fetches, if enabled. How can ETags work as cookies[2]?

[2] You may want to consult section 13.3.2-3 and section 14.19 `http://www.ietf.org/rfc/rfc2616.txt`

1.3.3 The case of Deep Packet Inspection

Recently, the mobile carrier and ISP '3' (`http://3.dk/`) was ordered by the danish courts to block `grooveshark.com`. '3' chose to block the site using a filtering system known as *Deep Packet Inspection* (DPI). The packet inspection works by looking into the TCP payload (i.e. the HTTP packages) and filtering or modifying the packets before retransmitting them.

Part 1: Explain how '3' could have filtered the `grooveshark.com` HTTP requests.

Part 2: Why is this a violation of layering in networks? How would you enforce the layering? *(Answer with 2-5 sentences)*

2 Practical part

During this course you will have three assignments with a practical part. 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. The service is based on a simple text-based protocol.

For this first assignment, you will be implementing an initial client and server using low level socket programming in Python. Your client and server will communicate in accordance with a simple protocol which is presented in Appendix A. Starting from the provided skeleton code you will have to implement a functioning client and server. In subsequent assignments, you will have to build on top of your client and server to make them communicate in accordance to an extended protocol.

For your convenience we will run a functional 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 server is one of your tasks, you should be able to complete the assignment even if the one we provide goes out of service.

2.1 Socket Programming

For this assignment we will be using Python's *low-level* socket interface. This interface is very close to the interface provided by the operating system and gives you a lot of control. It is *not* allowed to use any high-level socket or network classes available for Python.

In this assignment you will be implementing a client and server communicating in accordance to a very simple protocol. The server side should offer a listener socket that can capture an incoming client and initiate a new socket connection to that client. The server also needs to implement the server side of the protocol described in A.1.

The client application needs to connect first to the server side listener socket to initiate a new socket connection. When a client is started it provides a very simple command-line interface for the user described in A.2. For this assignment, your server is required to handle three types of client requests:

- PING: Perform a simple ping to the server
- CALC: Ask the server to solve a simple arithmetic problem
- ECHO: Sends a message to the server which will reply with the original message

Inspired by HTTP, each request above only requires the server to send back a response through the socket established by the client. The socket does not need to be kept alive after a response is sent by the server, but it could be kept alive as an optimization.

The complete protocol is explained in appendix A.

For this assignment, your server is only required to handle a single client at a time, however we encourage you to try and extend your server to handle multiple simultaneous clients, since this will be important for future assignments. However, make sure you have the basic server working before you start experimenting with handling several clients.

In addition to implementing the client and server you should answer the following questions:

Question 1

The skeleton code restricts messages to `BUFFER_SIZE` characters. Is this a smart choice or would it be better to use a unique token that is used to mark the end of a message? What would be a good token to use (i.e. one or more characters)? Could you use a combination of both strategies? Explain your answer in *one* paragraph.

Question 2

If a socket connection dies, should anything happen, or should we just remove the socket and continue functioning? Should the client try to reconnect to the address belonging to the dead socket? Explain your answer in *one* paragraph.

Question 3

Explain in one paragraph how you tested your implementation. How robust would you judge your client and server to be? How do they behave to malformed responses/requests? What would be some basic steps you could take to avoid a malicious party to crash your client or server?

Question 4

Is the protocol described in Appendix A stateful or stateless? If the answer is stateful, what do you consider part of the state? If the answer is stateless, give an argument why you consider the protocol stateless.

A Protocol

This appendix describes the `client ↔ server ↔ user` protocols.

A.1 CLIENT ↔ SERVER

`PING`

Perform a simple ping-like request from the client to the server.
The server's response to a well formed request is:

`100 PONG <address> <sys time>` The server responds with the address of the client (obtained from the socket), and the system time local to the server.

`CALC <num1> <op> <num2>`

Perform an arithmetic calculation request from the client to the server. The `num` can be any integer or real number. The `op` operator can be either addition, subtraction, multiplication or division (represented by `+-*/`). An example of such a query could be
`CALC 20.4 - 10.2`
The server will need to catch illegal arithmetic expressions.
The server's response to a well formed request is:

`200 EQUALS <num>` The expression was calculated successfully and returned as `num`.

`201 NAN` The expression could not be determined, or result is undefined (not-a-number).

`ECHO <message>`

Send a message to the server which, in response, will send the original message back to the client. There is exactly one blank space between blank `ECHO` and `<message>`.
If `<message>` starts, contains or ends with blank spaces or tabs, they will be included in the server response.
The server's response to a well formed request is:
`300 <message>`
where `<message>` is the original message sent by the client.

Note: The emphasis on the format of ECHO has been made because for ECHO we need to receive everything we send, including blank spaces or tabs. This not the case of CALC since the most important part of CALC is that the expression is a correct arithmetic expression. Thus the protocol is not so restrictive with CALC in this regard.

A.2 CLIENT ↔ USER

- /ping — as described in A.1
- /calc <num1> <op> <num2> — as described in A.1
- /echo <message> — as described in A.1

Note: Given the response from the server, all that the client does is to print all responses to the console as received, so that the user can inspect the entire content.