Introduction to Web Science

Assignment 3

Prof. Dr. Steffen Staab
staab@uni-koblenz.de

René Pickhardt
rpickhardt@uni-koblenz.de

Korok Sengupta
koroksengupta@uni-koblenz.de

Institute of Web Science and Technologies
Department of Computer Science
University of Luxembourg

Submission until: November 16, 2016, 10:00 a.m.
Tutorial on: November 18, 2016, 12:00 p.m.

The main objective of this assignment is for you understand different concepts that are associated with the "Web". In this assignment we cover two topics: 1) DNS & 2) Internet.

These tasks are not always specific to "Introduction to Web Science". For all the assignment questions that require you to write a code, make sure to include the code in the answer sheet, along with a separate python file. Where screen shots are required, please add them in the answers directly and not as separate files.

Team Name: quebec

1. Daniel Kostic
2. Stefan Vujovic
3. Igor Fedotov

1 DIG Deeper (5 Points)

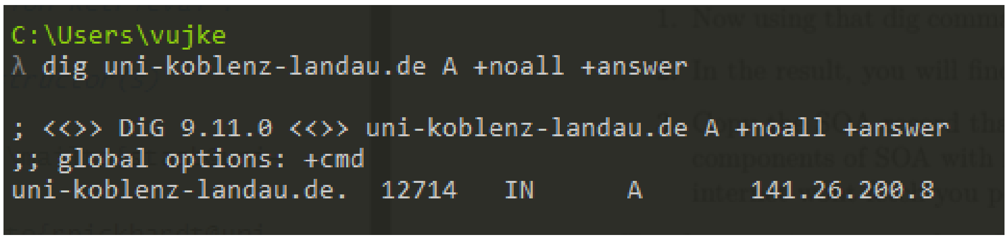
Assignment 1 started with you googling certain basic tools and one of them was "*dig*".

1. Now using that dig command, find the IP address of `www.uni-koblenz-landau.de`
2. In the result, you will find "SOA". What is SOA?
3. Copy the SOA record that you find in your answer sheet and explain each of the components of SOA with regards to your find. Merely integrating answers from the internet wont fetch you points.

Try the experiment once from University network and once from Home network and see if you can find any differences and if so, clarify why.

Answers:

1. The IP adress of www.uni-koblenz-landau.de is: 141.26.200.8



```
C:\Users\vujke
λ dig uni-koblenz-landau.de A +noall +answer

; <<>> DiG 9.11.0 <<>> uni-koblenz-landau.de A +noall +answer
;; global options: +cmd
uni-koblenz-landau.de.  12714   IN      A       141.26.200.8
```

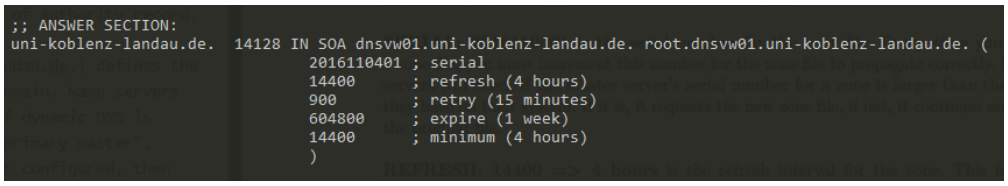
2. The Start of Authority, or SOA, record is a mandatory record in all zone files. It must be the first real record in a file (although $ORIGIN or $TTL specifications may appear above).

A start of authority (SOA) record is information stored in a domain name system (DNS) zone about that zone and about other DNS records. A DNS zone is the part of a domain for which an individual DNS server is responsible. Each zone contains a single SOA record. SOA records are defined in IETF RFC 1035, Domain Names - Implementation and Specification.

The SOA record stores information about the name of the server that supplied the data for the zone; the administrator of the zone; the current version of the data file; the number of seconds a secondary name server should wait before checking for updates; the number of seconds a secondary name server should wait before retrying a failed zone transfer; the maximum number of seconds that a secondary name server can use data before it must either be refreshed or expire; and a default number of seconds for the time-to-live file on resource records.

3. This is the SOA record for www.uni-koblenz-landau.de is:

uni-koblenz-landau.de. 14399 IN SOA dnsvw01.uni-koblenz-landau.de. root.dnsvw01.uni-koblenz-landau. de. 2016110401 14400 900 604800 14400



```
;; ANSWER SECTION:
uni-koblenz-landau.de.  14128 IN SOA dnsvw01.uni-koblenz-landau.de. root.dnsvw01.uni-koblenz-landau.de. (
                                2016110401 ; serial
                                14400      ; refresh (4 hours)
                                900        ; retry (15 minutes)
                                604800     ; expire (1 week)
                                14400      ; minimum (4 hours)
                                )
```

Going from left to right, we see the following components:

uni-koblenz-landau.de. which represents the root of the zone.

IN SOA The "IN" portion means internet. The SOA is the indicator that this is a Start of Authority record.

MNAME: dnsvw01.uni-koblenz-landau.de. defines the primary master name server for this domain. Name servers can either be master or slaves, and if dynamic DNS is configured one server needs to be a "primary master", which goes here. If dynamic DNS is not configured, then this is just one of the master name servers.

RNAME: root.dnsvw01.uni-koblenz-landau. de.

SERIAL: 2016110401 is the serial number for the zone file. Every time you edit a zone file, you must increment this number for the zone file to propagate correctly. Slave servers will check if the master server's serial number for a zone is larger than the one they have on their system. If it is, it requests the new zone file, if not, it continues serving the original file.

REFRESH: 14400 => 4 hours is the refresh interval for the zone. This is the amount of time that the slave will wait before polling the master for zone file changes.

RETRY: 900 => 15 minutes is the retry interval for this zone. If the slave cannot connect to the master when the refresh period is up, it will wait this amount of time and retry to poll the master.

EXPIRE: 604800 => 7 days is the expiry period. If a slave name server has not been able to contact the master for this amount of time, it no longer returns responses as an authoritative source for this zone.

MINIMUM: 14400 => 4 hours is the amount of time that the name server will cache a name error if it cannot find the requested name in this file. This is also known as the ”Minimum TTL”, the number of seconds that the records in the zone are valid for (time-to-live, or TTL), unless the records have a higher TTL value.

2 Exploring DNS (10 Points)

In the first part of this assignment you were asked to develop a simple TCP Client Server. Now, using **that** client server setup. This time a url should be send to the server and the server will split the url into the following:

`http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#InTheDocument`

1. Protocol
2. Domain
3. Sub-Domain
4. Port number
5. Path
6. Parameters
7. Fragment

The Protocol for sending the URL will be a string terminated with \r \n.

P.S.: You are **not** allowed to use libraries like `urlparse` for this question. You will also not use "Regular Expressions" for this.

Answer: Parsing the URL can start from the end. In the URL, we first look for the fragment. If found, it is saved and removed from the rest of the URL. Same goes for the parameters, which are saved and removed if found. After this the URL is split by colons. Three possible cases were recognized:

1. Port and protocol are missing
2. Either port or protocol are available
3. Both port and protocol are available

Each of the cases is handled separately, and all available info is saved(port, protocol, domain, path..).
Client side:

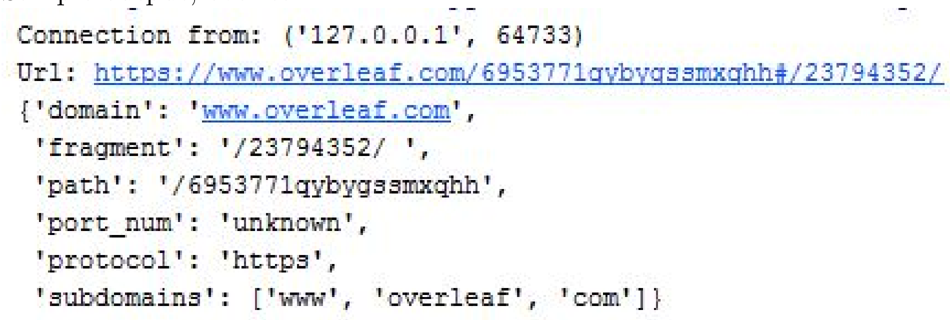
```python
import socket, json

def Main():
    host = 'localhost'
    port = 8080

    user_url = input('Insert URL for parsing: ')

    mySocket = socket.socket()
    mySocket.connect((host, port))

    # message = dict({'url': user_url})
    message = user_url + "\r \n"
    mySocket.send(message.encode())

    print(message)

    mySocket.close()


if __name__ == '__main__':
    Main()
```

Server side:

```
import socket
import json
from pprint import pprint


def Main():
    host = "localhost"
    port = 8080

    mySocket = socket.socket()
    mySocket.bind((host, port))

    mySocket.listen(1)
    conn, addr = mySocket.accept()
    print("Connection from: " + str(addr))
    data = conn.recv(1024).decode()
    if not data:
        return
    url = data.strip("\r \n")
    print("Url: " + url)
    url_dict = {}
    if '#' in url:  # contains fragment?
        url_dict['fragment'] = url.split('#')[1]
        url = url.split('#')[0]
    if '?' in url:  # contains params?
        url_dict['params'] = url.split('?')[1].split('&')
        url = url.split('?')[0]
    colon = url.split(':')
    if (len(colon) == 1):  # no port, no protocol
        url_dict['protocol'] = 'unknown'
        url_dict['domain'] = colon[0].split('/')[0]
        url_dict['path'] = colon[0][len(url_dict['domain']):]
    elif (len(colon) == 2):
        # no port number, or protocol
        if (colon[1][0] == "/"):  # protocol available
            url_dict['protocol'] = colon[0]
            url_dict['port_num'] = 'unknown'
            url_dict['domain'] = colon[1].split('/')[2]
            url_dict['path'] = colon[1][len(url_dict['domain']) + 2:]
        else:  # port num available
            url_dict['port_num'] = colon[1].split('/')[0]
            url_dict['domain'] = colon[0]
            url_dict['path'] = colon[1][len(url_dict['port_num'])]
    elif (len(colon) == 3):
        # with port number
        url_dict['protocol'] = colon[0]
        url_dict['port_num'] = colon[2].split('/')[0]
```

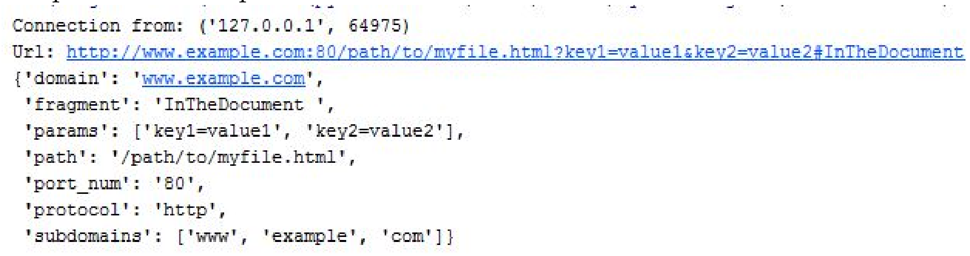
```
            url_dict['domain'] = colon[1][2:]
            url_dict['path'] = colon[2][len(url_dict['port_num']):]
        url_dict['subdomains'] = url_dict['domain'].split('.')
        pprint(url_dict)
        conn.close()


if __name__ == '__main__':
    Main()
```

Sample output, overleaf link:



```
Connection from: ('127.0.0.1', 64733)
Url: https://www.overleaf.com/6953771qybygssmxqhh#/23794352/
{'domain': 'www.overleaf.com',
 'fragment': '/23794352/ ',
 'path': '/6953771qybygssmxqhh',
 'port_num': 'unknown',
 'protocol': 'https',
 'subdomains': ['www', 'overleaf', 'com']}
```

Output with example link:



```
Connection from: ('127.0.0.1', 64975)
Url: http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#InTheDocument
{'domain': 'www.example.com',
 'fragment': 'InTheDocument ',
 'params': ['key1=value1', 'key2=value2'],
 'path': '/path/to/myfile.html',
 'port_num': '80',
 'protocol': 'http',
 'subdomains': ['www', 'example', 'com']}
```

3 DNS Recursive Query Resolving (5 Points)

You have solved the ”Routing Table” question in Assignment 2. We updated the routing tables once more. resulting in the following tables creating the following topology

Table 1: Routing Table

Router1			**Router2**			**Router3**		
Destination	Next Hop	Interface	Destination	Next Hop	Interface	Destination	Next Hop	Interface
67.0.0.0	67.68.3.1	eth 0	205.30.7.0	205.30.7.1	eth 0	205.30.7.0	205.30.7.2	eth 0
62.0.0.0	62.4.31.7	eth 1	156.3.0.0	156.3.0.6	eth 1	88.0.0.0	88.6.32.1	eth 1
88.0.0.0	88.4.32.6	eth 2	26.0.0.0	26.3.2.1	eth 2	25.0.0.0	25.03.1.2	eth 2
141.71.0.0	141.71.20.1	eth 3	141.71.0.0	141.71.26.3	eth 3	121.0.0.0	121.0.3.1	eth 3
26.0.0.0	141.71.26.3	eth3	67.0.0.0	141.71.20.1	eth 3	156.3.0.0	205.30.7.1	eth 0
156.3.0.0	88.6.32.1	eth 2	62.0.0.0	141.71.20.1	eth 3	26.0.0.0	205.30.7.1	eth 0
205.30.7.0	141.71.26.3	eth 3	88.0.0.0	141.71.20.1	eth 3	141.71.0.0	205.30.7.1	eth 0
25.0.0.0	88.6.32.1	eth 2	25.0.0.0	205.30.7.2	eth 0	67.0.0.0	88.4.32.6	eth 1
121.0.0.0	88.6.32.1	eth 2	121.0.0.0	205.30.7.2	eth 0	62.0.0.0	88.4.32.6	eth 1

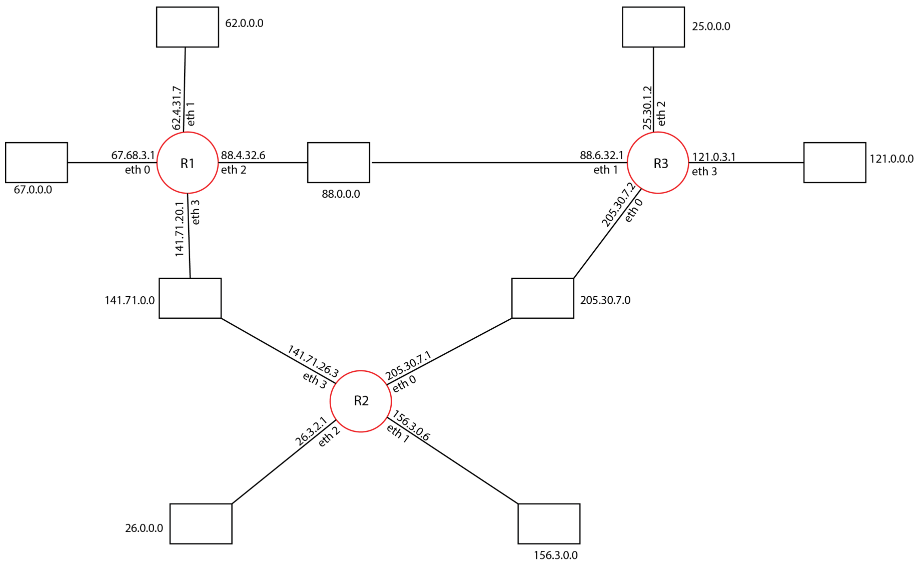


Figure 1: DNS Routing Network

Let us asume a client with the following ip address 67.4.5.2 wants to resolve the following domain `subdomain.webscienceexampledomain.com` using the DNS.

You can further assume the root name server has the IP address of 25.8.2.1 and the name-server for `webscienceexampledomain.com` has the IP address 156.3.20.2. Finally the sub-domain is handled by a name server with the IP of 26.155.36.7.

Please explain how the traffic flows through the network in order to resolve the recursive DNS query. You can assume ARP tables are cached so that no ARP-requests have to be made.

Answer: **An assumption has been made that the root supports recursive DNS queries** 67.4.5.2 creates an IP packet with the source address 67.4.5.2 an destination address 25.8.2.1 inside there is a DNS recursive query. This IP packet is sent as an Ethernet frame to Router 1 (R1) with the IP address of 67.68.3.1 on eth0. R1 receives the frame and looks inside it's routing table,and forwards the encapsulated IP packet to Router 3 (R3) with an address 88.6.32.1 on eth2. R3 receives the frame and routes it according to its table to 25.8.2.1 on eth2. After receiving the recursive request, the root server begins an iterative query. Lets assume that the webscience.com is inside root's zone. He will then create an IP packet with a DNS query for the 156.3.20.2 DNS server. This packet is sent as an Ethernet frame to R3, who forwards it to Router 2 (R2) via eth 0. R2 can forward this packet directly to the destination, so he makes an Ethernet frame and sends it to 156.3.20.2. This DNS sever knows about subdomain.webscience.com which is in his zone, so he sends a response to root with an IP of the sub domain DNS. Traffic flows trough the network the same way as previously. Finally, the root server asks the sub domain server to resolve the query, and he gets an IP address as a response. This IP address is sent from the root back to the client, and he can now use this information to make new requests.

Important Notes

Submission

- Solutions have to be checked into the github repository. Use the directory name `groupname/assignment3/` in your group's repository.
- The name of the group and the names of all participating students must be listed on each submission.
- Solution format: all solutions as *one* PDF document. Programming code has to be submitted as Python code to the github repository. Upload *all* `.py` files of your program! Use `UTF-8` as the file encoding. *Other encodings will not be taken into account!*
- Check that your code compiles without errors.
- Make sure your code is formatted to be easy to read.
  - Make sure you code has consistent indentation.
  - Make sure you comment and document your code adequately in English.
  - Choose consistent and intuitive names for your identifiers.
- Do *not* use any accents, spaces or special characters in your filenames.

Acknowledgment

This latex template was created by Lukas Schmelzeisen for the tutorials of ”Web Information Retrieval”.

LaTeX

Currently the code can only be build using LuaLaTeX, so make sure you have that installed. If on Overleaf, there's an error, go to settings and change the LaTeXengine to `LuaLaTeX`.