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: hotel

Sebastian Blei 211100448 sblei@uni-koblenz.de

Andrea Mildes 212100201 mildes@uni-koblenz.de

Johannes Kirchner 213200317 jkirchner@uni-koblenz.de

Abdul Afghan Guest Student(No Uni Credentials yet)

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 is 141.26.200.8
2. SOA is the "Start of Authority" and is part of the Zonefile. It consists (among other things) of a Serialnumber and of timers defining the refresh, retry, expire and time to live intervals. The SOA is thus responsible of controlling the Zonetransfer, which is the synchronization of data within a cluster of DNS-Nameservers.
3. SOA Record: "uni-koblenz-landau.de. 1340 IN SOA dnsvw01.uni-koblenz-landau.de. root.dnsvw01.uni-koblenz-landau.de. 2016110401 14400 900 604800 14400"
   - uni-koblenz-landau.de: the name of the zone
   - 1340: indicates the time to live (the time this entry remains cached) remaining
   - IN: represents the zoneclass. IN is the abbreviation for Internet
   - SOA: indicates that this is a SOA record
   - dnsvw01.uni-koblenz-landau.de: the primary master of the zone. Changes to the zonefile will only be done on this particular server. Secondary nameservers will create their zonefile by transferring the content of the zonefile from this (primary) server.
   - root.dnsvw01.uni-koblenz-landau.de : the e-mail-adress of the person responsible for the zone. The first period indicates the "@" symbol. So the corresponding e-mail-adress would be root@dnsvw01.uni-koblenz-landau.de
   - 2016110401: the serialnumber, which will increase after each change. 2016 is the year, 11 the month, 04 the day and 01 the vesion-number, which will be increased after every change. This is used by the secondary nameservers to identify changes made to the primary nameserver.

- 14400: represents the refresh-interval in seconds. Every secondary nameserver will request the serialnumber of its primary nameserver after this interval to see if any changes took place. (in this case every 4 hours)
- 900: represents the retry-interval. If the secondary nameserver requested the serialnumber of its primary nameserver but did not receive a response, it will try again after the given amount of seconds. (After 15 Minutes in this case)
- 604800: represents the expire-interval. If the primary nameserver still does not respond after this interval, the secondary nameserver will no longer respond to requests regarding the zone. (After 1 Week in this case)
- 14400: represents the TTL, the Time to live for negative caching. Negative caching is the caching of requested domains that can't be matched with an dns-entry, so the next time the nameserver receives a request containing the domain, it can look up the negative entry in it's cache. Time to live defines the time a negative cache entry is valid. As soon as the TTL is over, the entry will be removed from the cache.

4. When performing the experiment inside the University network, the dig command also returns an authority section and an additional section. This is not the case when using dig outside the university network. This is because you are addressing different DNS Servers depending on the network you are in. It seems like the DNS Server outside the university is a recursive DNS server, which cached the IP Adress for www.uni-koblenz-landau.de and therefor returns only the "Answer" IP Adress. However, there is no recursive DNS server inside the university network, so the dig command also returns the authority nameservers in the authority section of the dig command as well as the authority nameservers combined with their IP-Addresses in the additional section.

   dig output when **inside** the university network:

```
; «» DiG 9.8.3-P1 «» +multiline www.uni-koblenz-landau.de
;; global options: +cmd
;; Got answer:
;; -»HEADER«- opcode: QUERY, status: NOERROR, id: 47103
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 6, ADDI-
TIONAL: 10

;; QUESTION SECTION:
;www.uni-koblenz-landau.de. IN A

;; ANSWER SECTION:
www.uni-koblenz-landau.de. 889 IN A 141.26.200.8

;; AUTHORITY SECTION:
de. 126261 IN NS n.de.net.
de. 126261 IN NS s.de.net.
de. 126261 IN NS a.nic.de.
de. 126261 IN NS l.de.net.
de. 126261 IN NS f.nic.de.
de. 126261 IN NS z.nic.de.

;; ADDITIONAL SECTION:
a.nic.de. 88051 IN A 194.0.0.53
a.nic.de. 88051 IN AAAA 2001:678:2::53
f.nic.de. 88051 IN A 81.91.164.5
f.nic.de. 88051 IN AAAA 2a02:568:0:2::53
l.de.net. 88051 IN A 77.67.63.105
l.de.net. 88051 IN AAAA 2001:668:1f:11::105
n.de.net. 88051 IN A 194.146.107.6
n.de.net. 88051 IN AAAA 2001:67c:1011:1::53
s.de.net. 88051 IN A 195.243.137.26
z.nic.de. 88051 IN A 194.246.96.1

;; Query time: 450 msec
;; SERVER: 141.26.64.60#53(141.26.64.60)
;; WHEN: Tue Nov 15 16:45:02 2016
;; MSG SIZE rcvd: 373
```

dig output when **outside** the university network:

```
; «» DiG 9.8.3-P1 «» +multiline www.uni-koblenz-landau.de
;; global options: +cmd
;; Got answer:
;; -»HEADER«- opcode: QUERY, status: NOERROR, id: 61559
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDI-
TIONAL: 0

;; QUESTION SECTION:
;www.uni-koblenz-landau.de. IN A

;; ANSWER SECTION:
www.uni-koblenz-landau.de. 416 IN A 141.26.200.8

;; Query time: 46 msec
;; SERVER: 192.168.2.1#53(192.168.2.1)
;; WHEN: Tue Nov 15 17:06:04 2016
;; MSG SIZE rcvd: 59
```

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:
client:

```
# assignment 3 task2
# Andrea Mildes - mildes@uni-koblenz.de
# Sebastian Blei - sblei@uni-koblenz.de
# Johannes Kircher - jkircher@uni-koblenz.de
# Abdul Afghan - abdul.afghan@outlook.de

import socket
import json

s = socket.socket()
s.connect(('localhost', 8080))

url = input('Please insert url: ')

jsonObj = {'url': url}
jsonStr = json.dumps(jsonObj)

s.send(jsonStr.encode())

s.close()
```

Server:

```
# assignment 3 task2
# Andrea Mildes - mildes@uni-koblenz.de
# Sebastian Blei - sblei@uni-koblenz.de
# Johannes Kircher - jkircher@uni-koblenz.de
# Abdul Afghan - abdul.afghan@outlook.de

import socket
import json

s = socket.socket()
s.bind(('localhost', 8080))

s.listen(5)

while True:
        c, addr = s.accept()
        jsonData = c.recv(1024)
        jsonObj = json.loads(jsonData.decode())
        urlString = jsonObj['url']

        # find protocol
        prot = 'standard assumed: http'
        protPos = urlString.find("://")
        if protPos != -1:
                prot = urlString[:protPos]
        else:
                protPos = -3

        # determine domains and port
        port = 'standard port assumed: 80'
        portPos = urlString.find(":", protPos+1)
        pathPos = urlString.find("/", protPos+3)
        if portPos != -1:
                domain = urlString[protPos+3: portPos]
                port = urlString[portPos+1: pathPos]
        else:
                if pathPos != -1:
                        domain = urlString[protPos+3: pathPos]
                else:
                        domain = urlString[protPos+3:]
        subdomain = 'none'
        lastDot = domain.rfind(".")
        subdomaintmp = domain[:lastDot]
        subDot = subdomaintmp.rfind(".")
        if subDot != -1:
                subdomain = subdomaintmp[:subDot]

        # find path ?
        valPos = urlString.find("?")
        path = 'none given'
        if pathPos != -1:
                path = urlString[pathPos+1:valPos]

        # find value
        fragPos = urlString.find("#")
        val = 'none given'
        if valPos != -1:
                val = urlString[valPos+1: fragPos]

        # determine fragment
        frag = 'no fragment'
        if fragPos != -1:
                frag = urlString[fragPos+1:]


        print('Url: ', jsonObj['url'])
        print('Protocol: ', prot)
        print('Domiain: ', domain)
        print('Sub-Domain: ', subdomain)
        print('Port Number: ', port)
        print('Path: ', path)
        print('Parameters: ', val)
        print('Fragment: ', frag)

        c.close()

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

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

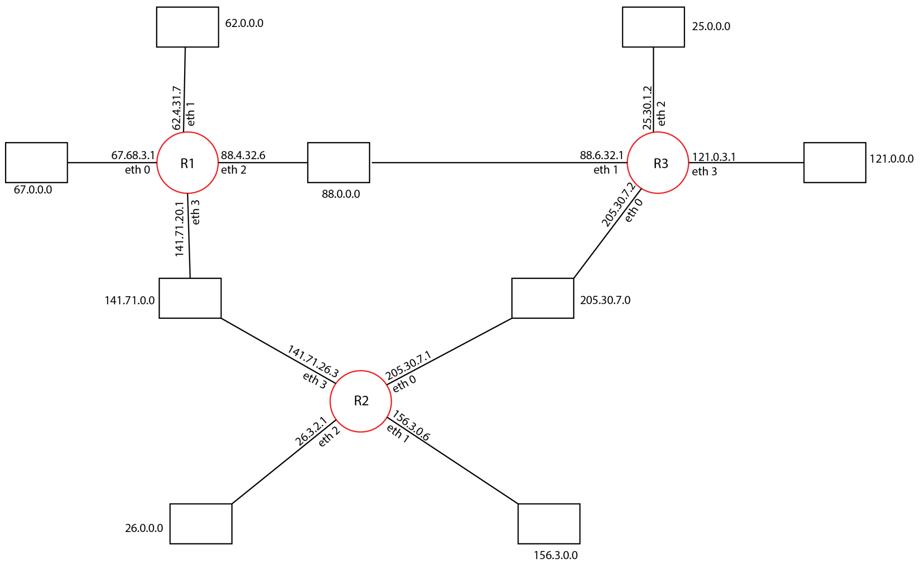

Figure 1: DNS Routing Network

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:
We assume the DNS Server for the client is the router of the network, in this case Router 1.

67.4.5.2 creates an IP packet with the source address 67.4.5.2 and destination address 67.68.3.1 (Router 1). Inside there is the DNS request. This IP packet is send as an ethernet frame to 67.68.3.1. 67.68.3.1 receives the frame and forwards the encapsulated IP packet to 88.6.32.1 which sends the packet to the root name server 25.8.2.1.

The root name server creates an IP packet with the source address 25.8.2.1 and the destination address 67.68.3.1.
Inside there is the DNS response which tells the DNS Server (Router 1) to send its request to the nameserver responsible for the webscienceexampledomain.com. zone and the IP address for said nameserver (156.3.20.2).
This IP packet is send as an ethernet frame to 25.30.1.2. 25.30.1.2 receives the frame and forwards the encapsulated IP packet to Router 1 / the DNS Server at 88.4.32.6.

Router 1 creates an IP packet with the source address 141.71.20.1 and the destination address 156.3.20.2. Inside there is the DNS request from the client. This IP packet is send as an ethernet frame to 141.71.26.3. 141.71.26.3 receives the frame and forwards the encapsulated IP packet to the nameserver at 156.3.20.2.
156.3.20.2 creates an IP packet with the source address 156.3.20.2 and the destination address 67.68.3.1. Inside there is the DNS response which tells the DNS Server (Router 1) to send its request to the nameserver responsible for the subdomain.webscienceexampledomain.com. zone and the IP address for said nameserver (26.155.36.7).
This IP packet is send as an ethernet frame to 156.3.0.6. 156.3.0.6 receives the frame and forwards the encapsulated IP packet to Router 1 at 141.71.20.1.

Router 1 creates an IP packet with the source address 141.71.20.1 and the destination address 26.155.36.7. Inside there is the DNS request. This packet is send as an ethernet frame to 141.71.26.3. 141.71.26.3 receives the packet and forwards the encapsulated IP packet to 26.155.36.7.
26.155.36.7 creates an IP packet with the source address 26.155.36.7 and the destination address 141.71.20.1. Inside there is the DNS response including the IP address of the requested domain.
This packet is send as an ethernet frame to 26.3.2.1. 26.3.2.1 receives the frame and forwards the encapsulated IP packet to 141.71.20.1.

Router 1 caches the IP address of the DNS request for future requests and creates an IP packet with the source address 67.68.3.1 and the destination address 67.4.5.2. Inside there is the DNS response including the requested IP address. The IP packet is send as an ethernet frame to 67.4.5.2. 67.4.5.2 receives the frame and now knows the IP address of subdomain.webscienceexample.com

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`.