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: Golf

Members: Atique Baig
Mtarji Adam
Deepak Garg

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. IP address of `www.uni-koblenz-landau.de` using Dig command:

```
adam@adam-VirtualBox:~$ dig SOA +multiline www.uni-koblenz-landau.de

; <<>> DiG 9.10.3-P4-Ubuntu <<>> SOA +multiline www.uni-koblenz-landau.de
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 14798
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.uni-koblenz-landau.de. IN SOA

;; AUTHORITY SECTION:
uni-koblenz-landau.de.  1617 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)
                              )

;; Query time: 17 msec
;; SERVER: 127.0.1.1#53(127.0.1.1)
;; WHEN: Sat Nov 12 15:08:36 CET 2016
;; MSG SIZE  rcvd: 103
```

Figure 1: dig command from University Network

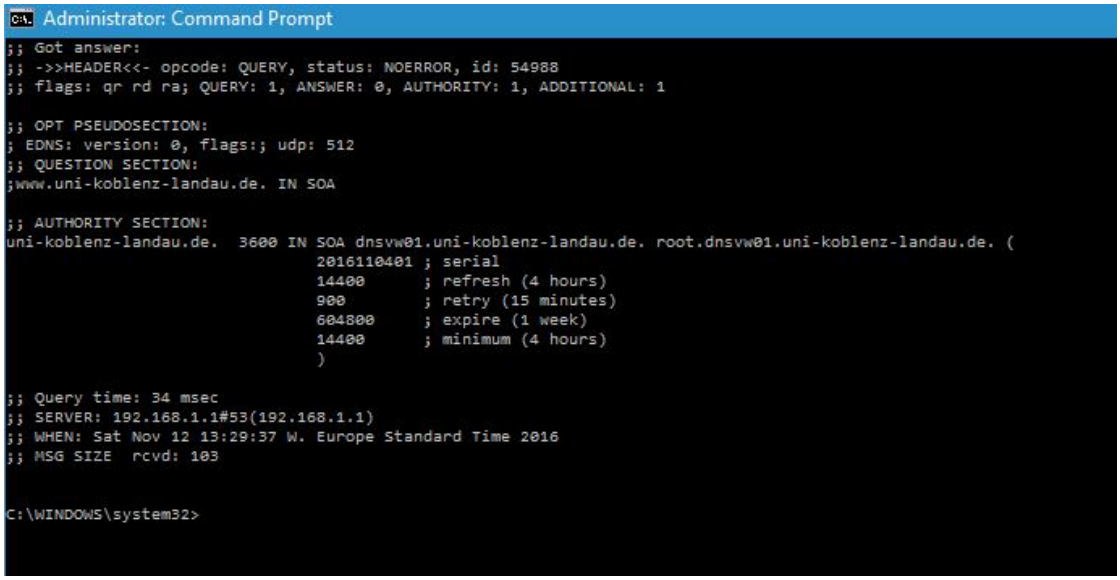

Figure 2: dig command from Home Network

2. SOA(Start of Authority)

   SOA stands for Start of Authority, a type of resource record stored by every DNS server in its database, it indicates that this DNS is the best source of information to look-up data in the requested domain/zone, as it holds basic properties of the domain and the zone he is part of. Every domain must have a Start of Authority record at the cut-over point where the domain is delegated from its parent domain.

3. components of SOA :

   Source host where the zone file is located:
   dnsvw01.uni-koblenz-landau.de.
   nameserver : dnsvw01
   place : uni-koblenz-landau
   top level domaine : de

   Contact e-mail of the administrator of the domain zone file :
   root.dnsvw01.uni-koblenz-landau.de
   postmaster : root.dnsvw01
   place : uni-koblenz-landau
   top level domain : de

   Serial number : 2016110401

The serial number is used as a revision number for the zone file, every change in the file increments the serial number and this allows other DNS servers to check the serials and know if they need to request an update of there copy of the file.

Refresh : 14400 (4 hours)
The time other DNS server waits before querying the DNS server for changes and compares the serials to know if he need to request a copy of the zone file.

Retry : 900 (15min)
The retry time is the time the server need to wait before retrying a failed zone transfer.The Retry value will tell it when to get back. This value is not very important and can be a fraction of the refresh value.

Expire time : 604800 (1week)
The total time a server will keep trying a zone transfer, after it expires, the server considers that the data is too old to be reliable.

Minimum TTL: 14400 (4hours)
The minimum time to live is the time that all servers with a copy of the zone file can keep it in there cache.

Comparison between home network and university network:
After using the dig command on both networks, we notice that we get the same result, this means that both queries goes to the same DNS server.

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:

First, we start by reusing the client/server setup we had last assignment, as a reminder, we made 2 python scripts. The server script uses a class named 'Receiver' which plays the role of a server, it will have a function to listen and read packets, another function that will parse the received packets(in this case the url) and print the information. We also added another class named 'URLobject' that will hold each of the information we are after as a member variable, and defines 2 other functions, one to parse a received url string and store the values in its member variables, and another function to print these variables. The second script uses a class named 'Sender' to prepare and send a url using a socket.

The concept we put in place, by using the python string function 'split()' only, we are splitting the url string once (giving us 2 parts) each time using a separator and by following the standard url structure to know which part to expect. We also take into consideration the absence of certain element of the url that is not always present (for example: the absence of parameters or fragments).

Screenshots:

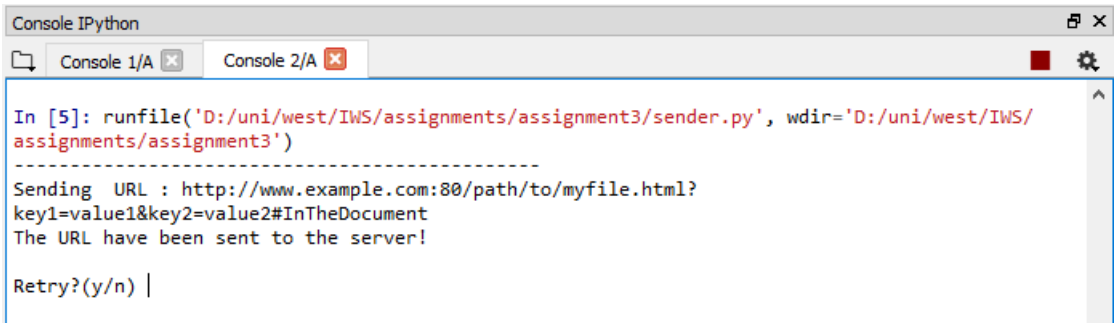

Figure 3: Information sent by the client

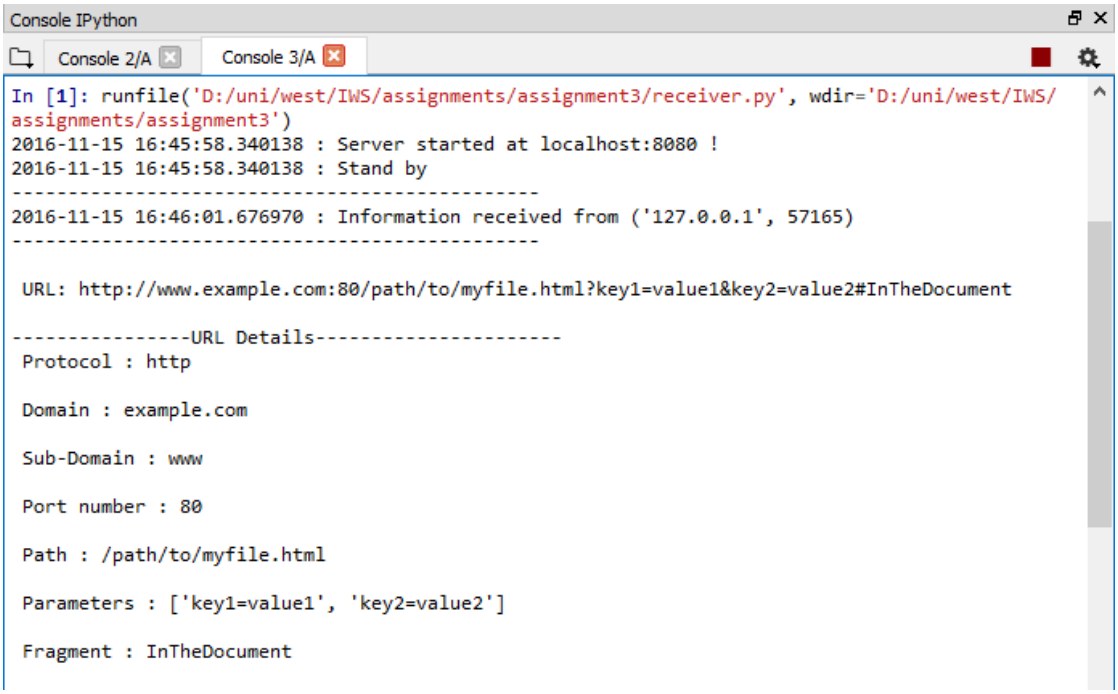

Figure 4: Information received by the server

sender.py:

```
# -*- coding: utf-8 -*-
"""
Introduction to Web Science
Assignment 3
Question 2
Team : golf
```

```
Script description :
    This script declares a 'Sender' class which will be used as a client
"""

# modules
import socket

"""
the Sender class will play the role of a client object
"""
class Sender():

    # nothing special to initialize for this object
    def __init__(self):
        pass

    # the send_packet function allows the Sender object to send a message to the server
    # First we create a socket, then we attempt to connect to the server
    # we encode the message so we can send it as a byte stream
    # After sending the packet we close the used socket
    def send_packet(self,message):
        try:
            self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.socket.connect(('localhost',8080))
            self.socket.send(message.encode('utf8'))
            self.socket.close()
            print('The URL have been sent to the server!')
        except socket.error as exc:
            print ("Error sending data to locolhost:8080 : %s" % exc)

    # client function prepares the url and sends it through the socket using
    # the send_packet function;
    # the chain argument allows subsequent use of the client if the user wishes to
    def client(self,chain='y'):
        if chain=='y':
            print('-------------------------------------------------')
            print('Sending  URL : http://www.example.com:80/path/to/myfile.html?key1=valu
            self.send_packet('http://www.example.com:80/path/to/myfile.html?key1=value1&k
            self.chain_query()

    # chain_query prompts the user for input regarding the client,
    # if he wishes to reuse the client he needs to input 'y'
    def chain_query(self):
        input_chain=input('Retry?(y/n) ').strip()
```

```
        if input_chain == 'y' or input_chain =='n':
            self.client(input_chain)
        else:
            print('Please provide an answer with y or n')
            self.chain_query()


# main script to start the client,
# we create a sender object and call the client function
sender = Sender()
sender.client()
```

receiver.py:

```
# -*- coding: utf-8 -*-
"""
Introduction to Web Science
Assignment 3
Question 2
Team : golf

Script description :
    This script declares a 'Receiver' class which will be used as a receiving server
    This script should be run first
"""

# modules
import socket
import datetime as dt

"""
Receiver class will be able to play the role of a server object
"""
class Receiver():
    # in the __init__ function we assign a server socket
    # and bind it to the localhost at the port N 8080
    def __init__(self):
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server_socket.bind(('localhost', 8080))
        self.server_socket.listen(5)

    # the receiv_packet function will be listening to any connection request
    # on the socket and accept it, then it reads the packet sent
    # and decode it using the utf8 encoding;
```

```
    # we feed the received message to the defined class URLobject
    # and call the function parse_url, we call the print_detail function
    # of the urlobject to print the information in the required format
    def receiv_packet(self):
        c,addr= self.server_socket.accept()
        p=c.recv(2048).decode('utf8')
        print('-----------------------------------------------')
        print('{0} : Information received from {1}' .format(dt.datetime.now(),addr))
        print('-----------------------------------------------')
        print('\n URL: {0}'.format(p))
        url_object=URLobject()
        url_object.parse_url(p)
        url_object.print_detail()
        c.close()
"""
URL_object is a class that will hold the url values and provide the function
to parse the url
"""
class URLobject:
    # we initialize all the member variables by N/A which will be printed
    # in case the url doesn't have one of the elements
    def __init__(self):
        self.protocol='N/A'
        self.domain='N/A'
        self.sub_domain='N/A'
        self.port='N/A'
        self.path='N/A'
        self.parameters='N/A'
        self.fragment='N/A'

    # print_detail simply prints the member variables in the required format
    def print_detail(self):
        print('----------------URL Details----------------------')
        print(' Protocol : {0} \n'.format(self.protocol))
        print(' Domain : {0} \n'.format(self.domain))
        print(' Sub-Domain : {0} \n'.format(self.sub_domain))
        print(' Port number : {0} \n'.format(self.port))
        print(' Path : {0} \n'.format(self.path))
        print(' Parameters : {0} \n'.format(self.parameters))
        print(' Fragment : {0} \n'.format(self.fragment))
        print('-----------------------------------------------')

    """
    Our main function, it takes a url_string as an argument and parses it.
```

```
    We use split function on multiple separators to get the information we are after;
    The concept is to continuously split the url in 2 each time using a separator,
    the first element splited is generally what we are looking for next,
    the second element splited is then spliten in turn in 2 and we repeat the process
    in addition, every time before splitting, we check if the remaining string
    contains the operator we are using next

    we are also taking in consideration the absence of certain element from the url
    for example: the port number can be absent, so is the parameters and fragment
    The function should be able to parse any valid url
    """
    def parse_url(self,url_string):
        # checks if the url contains the ':' separator
        if(':' in url_string):
            # we split the url in 2, giving us the protocol as a fist element
            s1_url = url_string.split(':',1)
            self.protocol = s1_url[0]
            # we will store the domain part of the url in this variable
            d_url=''
            # we will store the path, parameters and fragment part of the url
            # in this variable
            s3_url=''

            """
            from here we check if there is a second ':' separator indicating
            the existence of a port number in the url, it its the case
            we continue splitting using that separator, if the port is absent
            we move to the next separator to get the domain url and the rest
            """
            # we check if the 1st remaining sub-url contains another ':'
            if(':' in s1_url[1]):
                # we split the 1st sub-url in 2, the first element contains
                # the domain and subdomain
                s2_url=s1_url[1].split(':',1)
                d_url=s2_url[0]
                # we check the 2nd element of the 2nd sub-url
                # for any '/' separator
                if('/' in s2_url[1]):
                    # we split  once using the '/' separator,
                    # this isolates the port number
                    _url=s2_url[1].split('/',1)
                    self.port=_url[0]
                    s3_url = _url[1]
```

```
        else:
            # in case there is no '/' separator we consider whats left to be
            # the port number
            self.port=s2_url[1]

    # else if there is no port number, we continue splitting to get the domain
    elif ('/' in (s1_url[1])[2:]):
        # we skip the first 2 characters as they are both '/'
        # then we split by '/' once
        s2_url=(s1_url[1])[2:].split('/',1)
        # the first element contains the domain
        d_url=s2_url[0]
        # the second element contains whats left of the url
        s3_url=s2_url[1]
    else:
        # if no '/' separator is found, this means that whats left is
        # the domain and subdomain part of the url
        d_url=(s1_url[1])[2:]

    # now we parse the d_url containing the domain part of the url
    # we check if any '.' separator exists
    if( '.' in d_url):
        # we split the 1st element of the 2nd sub-url in 3
        # we consider that we should have 2 '.' separator in most cases
        site_url = d_url.split('.',3)
        # the first 2 elements from the left are
        # the top level domain and the domain
        self.domain = site_url[-2]+'.'+site_url[-1]
        # the remaining element is the subdomain,
        # we remove the extra '/'
        self.sub_domain = site_url[-3].replace('/','')
    else:
        # if the domain part of the url does not contain any '.'
        # it is not a valid url
        print(' Not a valid url')

    # we check the remaining sub url for any '?' separator
    if('?' in s3_url):
        # we split using '?' separator to get
        # the path as a first element
        s4_url=s3_url.split('?',1)
        self.path='/'+s4_url[0]
        # we check the remaining sub-url for any '#'
        if( '#' in s4_url[1]):
```

```
                    # we split the sub-url using the '#' separator
                    # to get the parameters as first element and
                    # fragment as second element
                    s5_url=s4_url[1].split('#',1)
                    self.parameters=s5_url[0].split('&')
                    self.fragment=s5_url[1]
                else:
                    # in this case, this means that the url has no fragment
                    self.parameters=s4_url[1].split('&')
            else:
                # in this case, this means the url has no parameters
                # but might have a fragment
                # we check the remaining sub-url for any '#'
                if('#' in s3_url):
                    # we split the sub-url using the '#' separator
                    # to get the parameters as first element and
                    # fragment as second element
                    s4_url=s3_url.split('#',1)
                    self.path='/'+s4_url[0]
                    self.fragment=s4_url[1]
                elif len(s3_url) > 0:
                    # in this case, this means the url has neither parameters
                    # nor fragments, whats left is the path
                    self.path='/'+s3_url


        else:
            # in the absence of a protocol, the url is not valid
            print(' Not a valid url')

# Main script to start the server, we create a receiver object first
try:
    receiver = Receiver()
    print('{0} : Server started at localhost:8080 !'.format(dt.datetime.now()))

    # using a loop while 1 will keep the server listening to connection attempts
    while 1:
        print('{0} : Stand by' .format(dt.datetime.now()))
        receiver.receiv_packet()
except socket.error as exc:
    print ("Error starting the server on the port 8080 : %s" % exc)
```

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

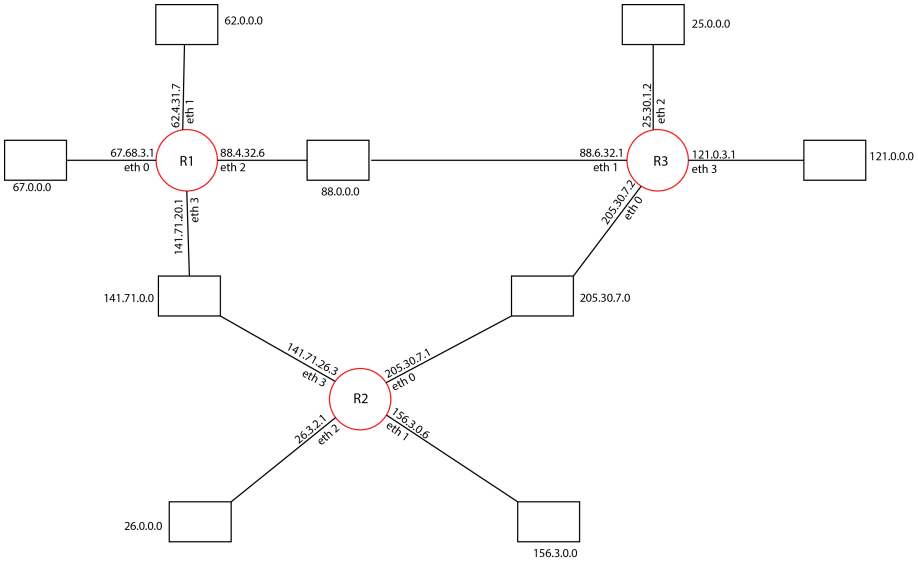

Figure 5: DNS Routing Network

Let us assume 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.

Hint: You can start like this:

67.4.5.2 creates an IP packet with the source address XXXXXX an destination address YYYYY inside there is the DNS request. This IP packet is send as an Ethernet frame to ZZZZZ. ZZZZZ receives the frame and forwards the encapsulated IP packet to ....

Also you can assume the DNS requests and responses will fit inside one IP packet. You also don't have to write down the specific DNS requests and responses in hex.

Answer:

67.4.5.2 creates an IP packet with the source address 67.4.5.2 and destination address 25.8.2.1, inside there is the DNS request. This IP packet is sent as an Ethernet frame to 67.68.3.1 (eth0). 67.68.3.1 (eth0) receives the frame and forwards the encapsulated IP packet to 88.6.32.1 (eth1), which in turn will forward the IP packet to 25.30.1.2 (eth2). Then 25.30.1.2 (eth2) will forward the IP packet to its final destination, the root DNS with the address 25.8.2.1.

The root DNS will read the request and send back a DNS referral to the client containing information about the DNS server for the webscienceexampledomain.com, he creates an IP packet with the source address 25.8.2.1 and destination address 67.4.5.2, inside is the DNS referral. This IP packet is sent as Ethernet frame to 25.30.1.2 (eth2). 25.30.1.2 (eth2) receives the frame and forwards the encapsulated IP packet to 88.4.32.6 (eth1), which in turn will forward the IP packet to 67.68.3.1 (eth0). Then 67.68.3.1 (eth0) will forward the IP packet to the client with the IP 67.4.5.2.

The client reads the DNS referral and sends a new DNS request to the IP address of webscienceexampledomain.com using the information received by the root DNS server. 67.4.5.2 creates an IP packet with the source address 67.4.5.2 and destination address 156.3.20.2(webscienceexampledomain.com), inside there is the DNS request. This IP packet is sent as an Ethernet frame to 67.68.3.1 (eth0). 67.68.3.1 (eth0) receives the frame and forwards the encapsulated IP packet to 88.6.32.1 (eth1), which in turn will forward the IP packet to 205.30.7.1 (eth0). then 205.30.7.1 (eth0) will forward the IP packet to 156.3.0.6 (eth1). Finally 156.3.0.6 (eth1) will forward the IP packet to its final destination, the webscienceexampledomain.com DNS with the address 156.3.20.2.

The webscienceexampledomain.com DNS will read the request and send back a DNS referral to the client containing information about the DNS server for the subdomain.webscienceexampledomain.com, he creates an IP packet with the source address 156.3.20.2 and destination address 67.4.5.2, inside is the DNS referral. This IP packet is sent as Ethernet frame to 156.3.0.6 (eth1). 156.3.0.6 (eth1) receives the frame and forwards the encapsulated IP packet to 141.71.20.1 (eth3), which in turn will forward the IP packet to 67.68.3.1 (eth0). Then 67.68.3.1 (eth0) will forward the IP packet to the client with the IP 67.4.5.2.

The client reads the DNS referral and sends a new DNS request to the IP address of subdomain.webscienceexampledomain.com using the information received by the web-

scienceexampledomain.com DNS server. 67.4.5.2 creates an IP packet with the source address 67.4.5.2 and destination address 26.155.36.7(subdomain.webscienceexampledomain.com), inside there is the DNS request. This IP packet is sent as an ethernet frame to 67.68.3.1 (eth0). 67.68.3.1 (eth0) receives the frame and forwards the encapsulated IP packet to 141.71.26.3 (eth3), which in turn will forward the IP packet to 26.3.2.1 (eth2). then 26.3.2.1 (eth2) will forward the IP packet to its final destination, the subdomain.webscienceexampledomain.com DNS with the address 26.155.36.7.

The subdomain.webscienceexampledomain.com DNS will read the request and send back a DNS answer to the client containing information about the target receiver, he creates an IP packet with the source address 26.155.36.7 and destination address 67.4.5.2, inside is the DNS answer. This IP packet is sent as Ethernet frame to 26.3.2.1 (eth2). 26.3.2.1 (eth2) receives the frame and forwards the encapsulated IP packet to 141.71.20.1 (eth3), which in turn will forward the IP packet to 67.68.3.1 (eth0). Then 67.68.3.1 (eth0) will forward the IP packet to the client with the IP 67.4.5.2.

The client, having a DNS answer, will proceed by sending an IP packet to the target receiver.

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`.