

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.

Group Name : Yankee

Candidates :

1. Sabin Bhattarai - 216203590
sbhattarai@uni-koblenz.de
2. Biplov K.C. - 216203865
biplov@uni-koblenz.de
3. Syed Salman Ali. - 216203923
salmanali@uni-koblenz.de

1 DIG Deeper (5 Points)

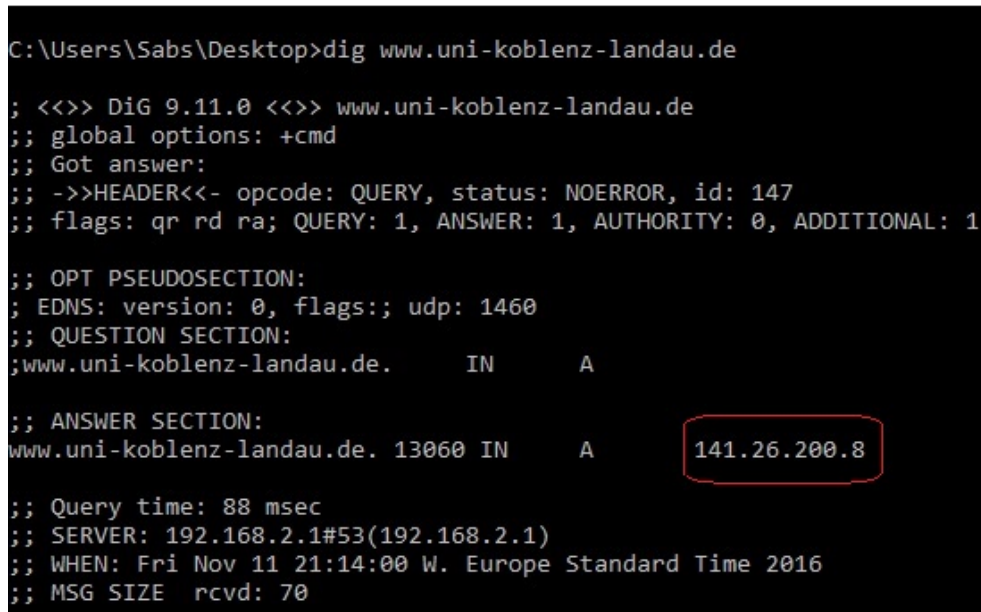
Assignment 1 started with you googling certain basic tools and one of them was "*dig*".

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 won't 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.

Answer 1.1

From Home



```
C:\Users\Sabs\Desktop>dig www.uni-koblenz-landau.de

; <<>> DiG 9.11.0 <<>> www.uni-koblenz-landau.de
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 147
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1460
;; QUESTION SECTION:
;www.uni-koblenz-landau.de.      IN      A

;; ANSWER SECTION:
www.uni-koblenz-landau.de. 13060 IN      A      141.26.200.8

;; Query time: 88 msec
;; SERVER: 192.168.2.1#53(192.168.2.1)
;; WHEN: Fri Nov 11 21:14:00 W. Europe Standard Time 2016
;; MSG SIZE rcvd: 70
```

Figure 1: IP address using Dig at home

The IP address of www.uni-koblenz-landau.de taken from home was **141.26.200.8**

From University

```
C:\Users\Sabs>dig www.uni-koblenz-landau.de

;; <<>> DiG 9.11.0 <<>> www.uni-koblenz-landau.de
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 23879
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 6, ADDITIONAL: 11

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

;; ANSWER SECTION:
www.uni-koblenz-landau.de. 4958 IN      A      141.26.200.8

;; AUTHORITY SECTION:
de.          130330 IN      NS      s.de.net.
de.          130330 IN      NS      a.nic.de.
de.          130330 IN      NS      l.de.net.
de.          130330 IN      NS      f.nic.de.
de.          130330 IN      NS      n.de.net.
de.          130330 IN      NS      z.nic.de.

;; ADDITIONAL SECTION:
a.nic.de.    92120 IN      A      194.0.0.53
a.nic.de.    92120 IN      AAAA   2001:678:2::53
f.nic.de.    92120 IN      A      81.91.164.5
f.nic.de.    92120 IN      AAAA   2a02:568:0:2::53
l.de.net.    92120 IN      A      77.67.63.105
l.de.net.    92120 IN      AAAA   2001:668:1f:11::105
n.de.net.    92120 IN      A      194.146.107.6
n.de.net.    92120 IN      AAAA   2001:67c:1011:1::53
s.de.net.    92120 IN      A      195.243.137.26
z.nic.de.    92120 IN      A      194.246.96.1

;; Query time: 0 msec
;; SERVER: 141.26.64.60#53(141.26.64.60)
;; WHEN: Tue Nov 15 15:37:13 W. Europe Standard Time 2016
;; MSG SIZE rcvd: 384
```

Figure 2: IP address using Dig at University

The IP address of `www.uni-koblenz-landau.de` taken from university was **141.26.200.8**

Answer 1.2

SOA(Start of Authority) is a special resource record that includes information about basic properties of the domain and the zone that the domain is in. It is included in every database file.

Answer 1.3

SOA record for `www.uni-koblenz-landau.de` from both home and uni were the same. The result obtained was: **2016110401 14400 900 604800 14400**

From Home

```
C:\Users\Sabs>dig soa www.uni-koblenz-landau.de

; <<>> DiG 9.11.0 <<>> soa www.uni-koblenz-landau.de
; global options: +cmd
; Got answer:
; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 58056
; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1

; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1460
; QUESTION SECTION:
; www.uni-koblenz-landau.de.      IN      SOA

; AUTHORITY SECTION:
uni-koblenz-landau.de. 3600      IN      SOA      dnsvw01.uni-koblenz-landau.de. root
dnsvw01.uni-koblenz-landau.de. 2016110401 14400 900 604800 14400

; Query time: 146 msec
; SERVER: 192.168.2.1#53(192.168.2.1)
; WHEN: Tue Nov 15 21:52:11 W. Europe Standard Time 2016
; MSG SIZE rcvd: 103
```

Figure 3: IP address using Dig SOA at home

From University

```
C:\Users\Sabs>dig soa www.uni-koblenz-landau.de

; <<>> DiG 9.11.0 <<>> soa www.uni-koblenz-landau.de
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 62605
;; 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. 1799 IN SOA dnsvw01.uni-koblenz-landau.de.
2016110401 14400 900 604800 14400

;; Query time: 37 msec
;; SERVER: 141.26.64.60#53(141.26.64.60)
;; WHEN: Tue Nov 15 15:38:43 W. Europe Standard Time 2016
;; MSG SIZE rcvd: 103
```

Figure 4: IP address using Dig SOA at university

We will now define each of the section in the result obtained above

2016110401 represents serial number. This serial number shows the number of times the DNS zone has been changed.

14400 represents refresh time(in seconds). This denotes the amount of time that secondary server takes to contact the primary(other) server if it has any changes. In this case the secondary server tries to connect to primary server every 14400 seconds(4 hours).

900 shows the retry time(in seconds). This data explains the time taken by secondary server to contact primary(other) server again if it was unable to connect during above mentioned refresh time period. In this case if the secondary server cannot connect to main server in 4 hours, it will try to connect to it in every 900 seconds i.e 15 minutes.

604800 explains the expiry time(in seconds). This value explains that if the secondary server is not able to connect the main server due to any technical reason (e.g network connectivity issue) then how long should the update record stay on top of that. so, in this case the secondary server can still resolve the DNS queries from client for 604800 seconds(1 week) without being able to contact the primary or main server.

14400 represents minimum TTL(Time To Live) value. This value represents the time(in seconds) that other DNS server stores the information taken from this server in its DNS cache. In this case, other DNS server/servers stores information in their cache taken from this server for 14400 seconds(4 hours)

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 2

```
C:\WINDOWS\system32\cmd.exe
C:\Users\Sabs\Desktop\yankee\Assignment 3>python client.py
Please Enter your URL :http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#InTheDocument

Received from Server

Protocol : http
Domain : www.example.com & example.com & com
SubDomain : www.example.com & example.com
Port : 80
Path : /path/to/myfile.html
Parameter : key1=value1&key2=value2
Fragment : InTheDocument

C:\WINDOWS\system32\cmd.exe
C:\Users\Sabs\Desktop\yankee\Assignment 3>python server.py
Received URL : http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#InTheDocument
-- You are now Using Team Yankees URL SPLITTER --
```

Figure 5: Result for Server Client communication

Class Library for splitting URL given to its function

```
1: # Class for URL to parts
2: class URLParts():
3:     def __init__(self):
4:         print ('-- You are now Using Team Yankees URL SPLITTER -- ')
5:
6:     def splitURL(self, urlFromClient):
7:
8:         url = urlFromClient # [:urlFromClient.find('\r')]
9:         PROTOCOLCHARS = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ '
10:        protocol = port = domain = subdomain = path = parameter = fragment = ''
11:
12:        protocolEnd = url.find('/:/')
13:
14:        if (protocolEnd > 0):
15:            for urlChar in url[:protocolEnd]:
16:                if urlChar not in PROTOCOLCHARS:
17:                    break
18:            else:
19:                protocol = url[:protocolEnd].lower()
20:                restUrl = url[protocolEnd:].lstrip('/:/')
21:
22:        if not protocol:
23:            restUrl = url
24:
25:
26:        posPath = restUrl.find('/')
27:        posParameter = restUrl.find('?')
28:        posFrag = restUrl.find('#')
29:
30:        if posPath > 0:
31:            if posParameter > 0 and posFrag > 0:
32:                domainSubPort = restUrl[:posPath]
33:                path = restUrl[posPath:min(posParameter, posFrag)]
34:            elif posParameter > 0:
35:                if posParameter > posPath:
36:                    domainSubPort = restUrl[:posPath]
37:                    path = restUrl[posPath:posParameter]
38:                else:
39:                    domainSubPort = restUrl[:posParameter]
40:                    path = ''
41:            elif posFrag > 0:
42:                domainSubPort = restUrl[:posPath]
43:                path = restUrl[posPath:posFrag]
44:            else:
45:                domainSubPort = restUrl[:posPath]
46:                path = restUrl[posPath:]
47:        else:
48:            if posParameter > 0:
```

```
49:         domainSubPort = restUrl[:posParameter]
50:     elif posFrag > 0:
51:         domainSubPort = restUrl[:posFrag]
52:     else:
53:         domainSubPort = restUrl
54:
55:     if posParameter > 0:
56:         if posFrag > 0:
57:             parameter = restUrl[posParameter+1:posFrag]
58:         else:
59:             parameter = restUrl[posParameter+1:]
60:     if posFrag > 0:
61:         fragment = restUrl[posFrag+1:]
62:     if not protocol:
63:         path = domainSubPort + path
64:         domainSubPort = ''
65:
66:     if (domainSubPort.find(':') != -1):
67:         port = domainSubPort[domainSubPort.find(':').:].lstrip(':')
68:     else:
69:         port = 'None'
70:
71:     if (domainSubPort.find(':') > 0):
72:         domainSub = domainSubPort[:domainSubPort.find(':')]
73:     else:
74:         domainSub = domainSubPort
75:
76:     domain = [domainSub]
77:     while domainSub.find('.') != -1 :
78:         domainSub = domainSub[domainSub.find('.').:].lstrip('.')
79:         domain.append(domainSub)
80:
81:     # Removing last element in domain list as all subdomain are domains.
82:     subdomain = domain[:]
83:     subdomain.pop()
84:     domainString = ''
85:     for domainVal in domain :
86:         if domainVal != domain[-1]:
87:             domainString = domainString + domainVal + ' & '
88:         else:
89:             domainString = domainString + domainVal
90:
91:     subDomainString = ''
92:     for subDomainVal in subdomain :
93:         if subDomainVal != subdomain[-1]:
94:             subDomainString = subDomainString + subDomainVal + ' & '
95:         else:
96:             subDomainString = subDomainString + subDomainVal
97:
```



```
98:         returnUrlParts = {}
99:         returnUrlParts['Protocol'] = protocol
100:        returnUrlParts['Domain'] = domainString
101:        returnUrlParts['SubDomain'] = subDomainString
102:        returnUrlParts['Port'] = port
103:        returnUrlParts['Path'] = path
104:        returnUrlParts['Parameter'] = parameter
105:        returnUrlParts['Fragment'] = fragment
106:        return (returnUrlParts)
```

Client Code

```
1:
2: from jsonSocket import Client
3: import json
4:
5: host = 'localhost'
6: port = 8080
7:
8: # Client code:
9: client = Client() # Client Class is obtained from jsonSocket Library
10: data = {}
11: url = input("Please Enter your URL :")
12: data['URL'] = url
13:
14: #collects data and converts to Json and sends to the Server
15: client.connect(host, port).send(json.dumps(data).encode())
16:
17: #Gets response from server
18: response = client.recv()
19:
20: print ("\n \n Received from Server \n \n" , response.decode())
21: client.close()
```

Server Code

```
1: from jsonSocket import Server
2: import json
3: from URLParts import URLParts
4:
5: host = 'localhost'
6: port = 8080
7:
8: server = Server(host, port) # Server Class is obtained from jsonSocket Library
9: server.accept()
10:
11: #Receives data from client
12: data = server.recv()
13: data = json.loads(data.decode())
14:
15: print ('Received URL : ' , data['URL'])
16:
17: #Here we use our URLParts class to use the function splitURL to get split data
18: urlToParts = URLParts()
19: dataURLParts = urlToParts.splitURL(data['URL'])
20:
21: #Editing the obtained Json formatted data from client
22: formattedText = "\n"
23: formattedText = formattedText + \
24:     'Protocol' + " : " + dataURLParts['Protocol'] + "\n" + \
25:     'Domain' + " : " + dataURLParts['Domain'] + "\n" + \
26:     'SubDomain' + " : " + dataURLParts['SubDomain'] + "\n" + \
27:     'Port' + " : " + dataURLParts['Port'] + "\n" + \
28:     'Path' + " : " + dataURLParts['Path'] + "\n" + \
29:     'Parameter' + " : " + dataURLParts['Parameter'] + "\n" + \
30:     'Fragment' + " : " + dataURLParts['Fragment'] + "\n";
31:
32: # Please note the following code could be done to omit hard coding. However
33: # with the below code the order was not achieved while printing
34:
35: #formattedText = ""
36: #for key, value in data.items():
37: #    formattedText = formattedText + key + ":" + value + "\n";
38:
39: # Now we encode the string and send to client for printing
40: server.send(formattedText.encode()).close()
```

Class Library for Client and Server

```
1: import socket
2:
3:
4: # Class for Server
5: class Server(object):
6:
7:     backlog = 5
8:     client = None
9:
10:    def __init__(self, host, port):
11:        self.socket = socket.socket()
12:        self.socket.bind((host, port))
13:        self.socket.listen(self.backlog)
14:
15:    def __del__(self):
16:        self.close()
17:
18:    def accept(self):
19:        # if a client is already connected, disconnect it
20:        if self.client:
21:            print('3')
22:            self.client.close()
23:        self.client, self.client_addr = self.socket.accept()
24:        return self
25:
26:    def send(self, data):
27:        if not self.client:
28:            raise Exception('Cannot send data, no client is connected')
29:        self.client.send(data)
30:        return self
31:
32:    def recv(self):
33:        if not self.client:
34:            raise Exception('Cannot receive data, no client is connected')
35:        return self.client.recv(30000)
36:
37:    def close(self):
38:        if self.client:
39:            self.client.close()
40:            self.client = None
41:        if self.socket:
42:            self.socket.close()
43:            self.socket = None
44:
45:
46:
47: #Class for Client
48: class Client(object):
```

```
49:
50:     socket = None
51:
52:     def __del__(self):
53:         self.close()
54:
55:     def connect(self, host, port):
56:         self.socket = socket.socket()
57:         self.socket.connect((host, port))
58:         return self
59:
60:     def send(self, data):
61:         if not self.socket:
62:             raise Exception('You have to connect first before sending data')
63:         self.socket.send(data)
64:         return self
65:
66:     def recv(self):
67:         if not self.socket:
68:             raise Exception('You have to connect first before receiving data')
69:         return self.socket.recv(30000)
70:
71:     def recv_and_close(self):
72:         data = self.recv()
73:         self.close()
74:         return data
75:
76:     def close(self):
77:         if self.socket:
78:             self.socket.close()
79:             self.socket = None
```

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

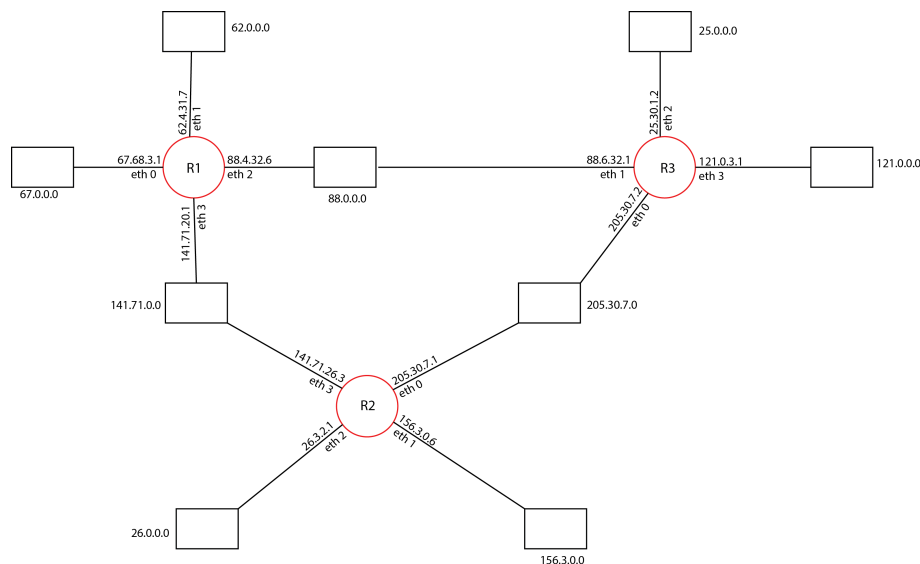


Figure 6: DNS Routing Network

Let us assume a client with the following ip address 67.4.5.2 wants to resolve the following domain `subdomain.webscienceexampdomain.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 `webscienceexampdomain.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 3:

Assumption:

1. The DNS requests and responses will fit inside one IP packet.
2. Our Internet Service Provider is R1.

Firstly, 67.4.5.2 creates an IP packet with the source address 67.4.5.2 and destination address 67.68.3.1 inside there is the DNS request. This IP packet is sent as an ethernet frame to R1 (ISP) i.e. 67.69.3.1 via eth0 as shown in figure below.

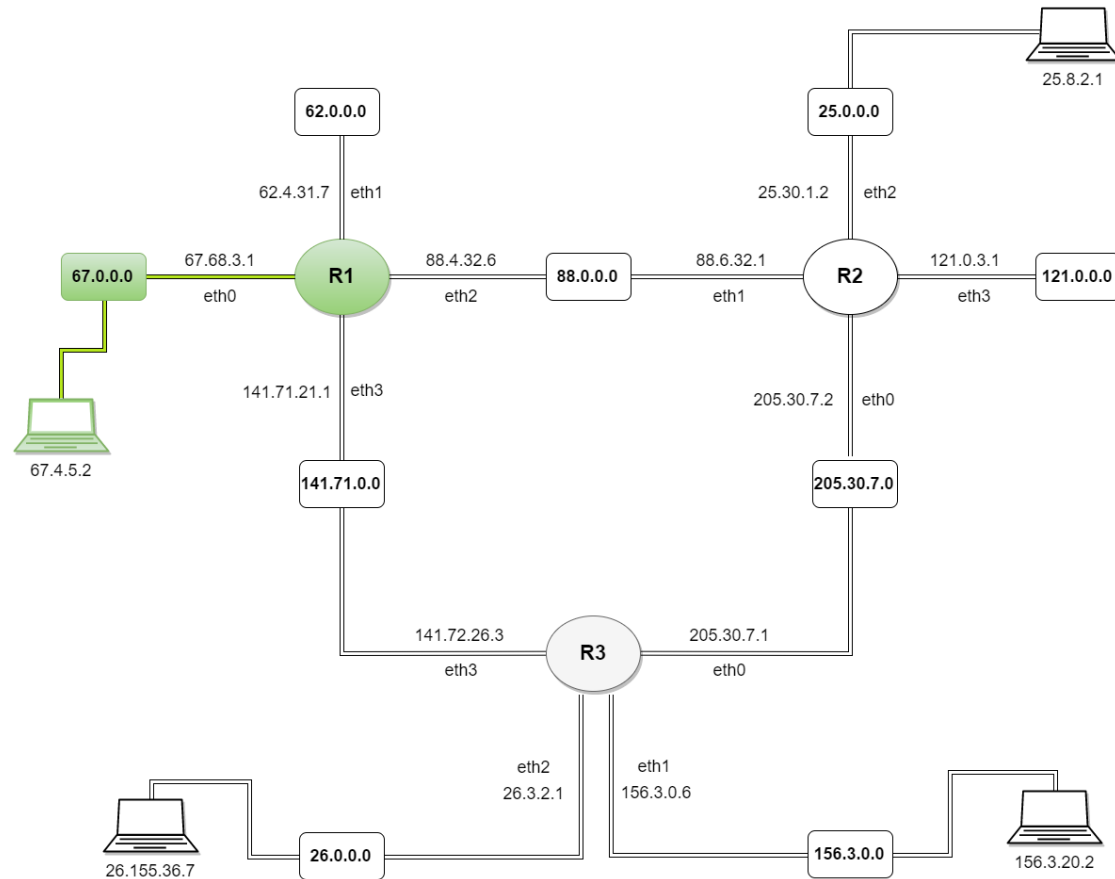


Figure 7: DNS request sent to R1 acting as Internet Service Provider

Secondly, R1 receives the frame and the encapsulated IP packet is sent to R2 with IP address 88.6.32.1 via eth1. R2 then forwards the request to root nameserver at ip address 25.8.2.1 as shown in the figure below. The ISP (R1) gets back the response from the root that the requested nameserver will be dealt at the next destination with nameserver websciencexampldomain.com associated to ip address 156.3.20.2.

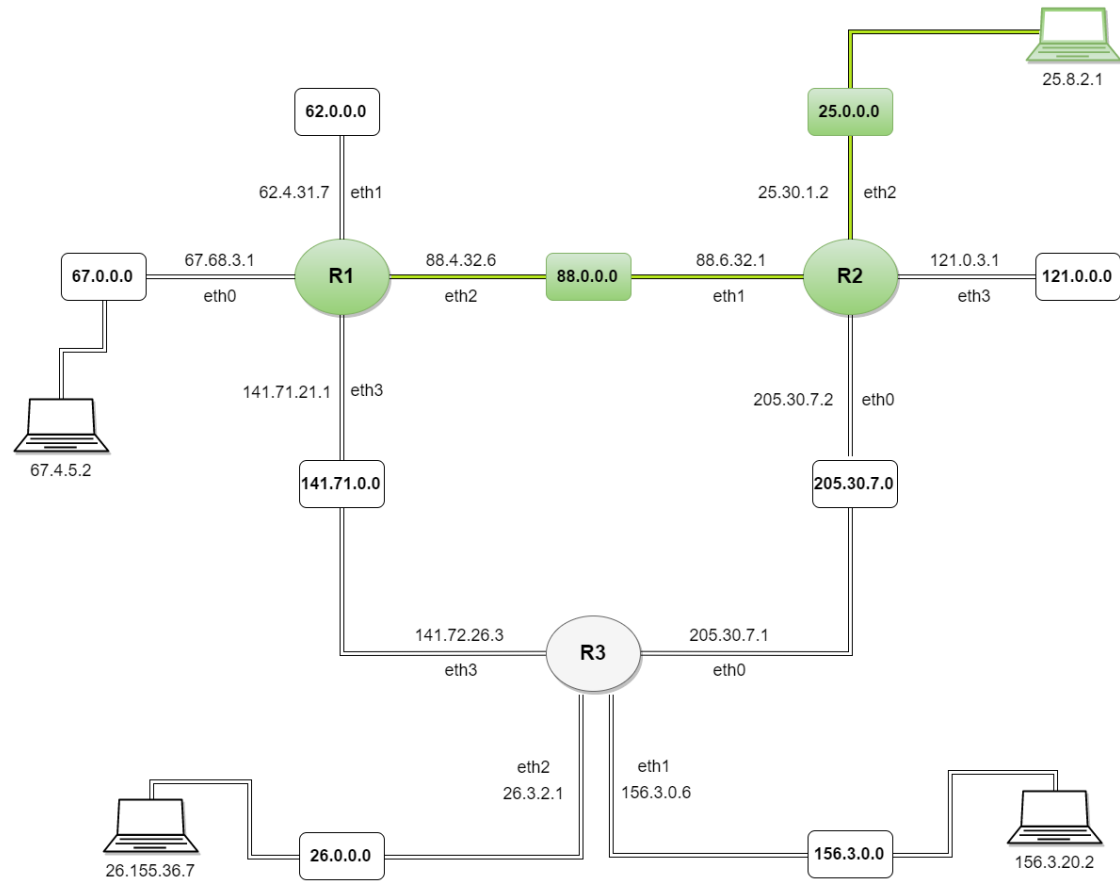


Figure 8: Request sent to root name server 25.8.2.1 and response received

Thirdly, R1 creates and sends the encapsulated IP packet with destination - 156.3.20.2 to R3 with IP address 141.72.26.3 via eth3. R3 then forwards the request to domain nameserver at ip address 156.3.20.2 as shown in the figure below. The ISP (R1) again gets the response from the domain nameserver that the requested nameserver will be dealt at the next destination with nameserver subdomain.webscienceexampledomain.com associated to ip address 26.155.36.7.

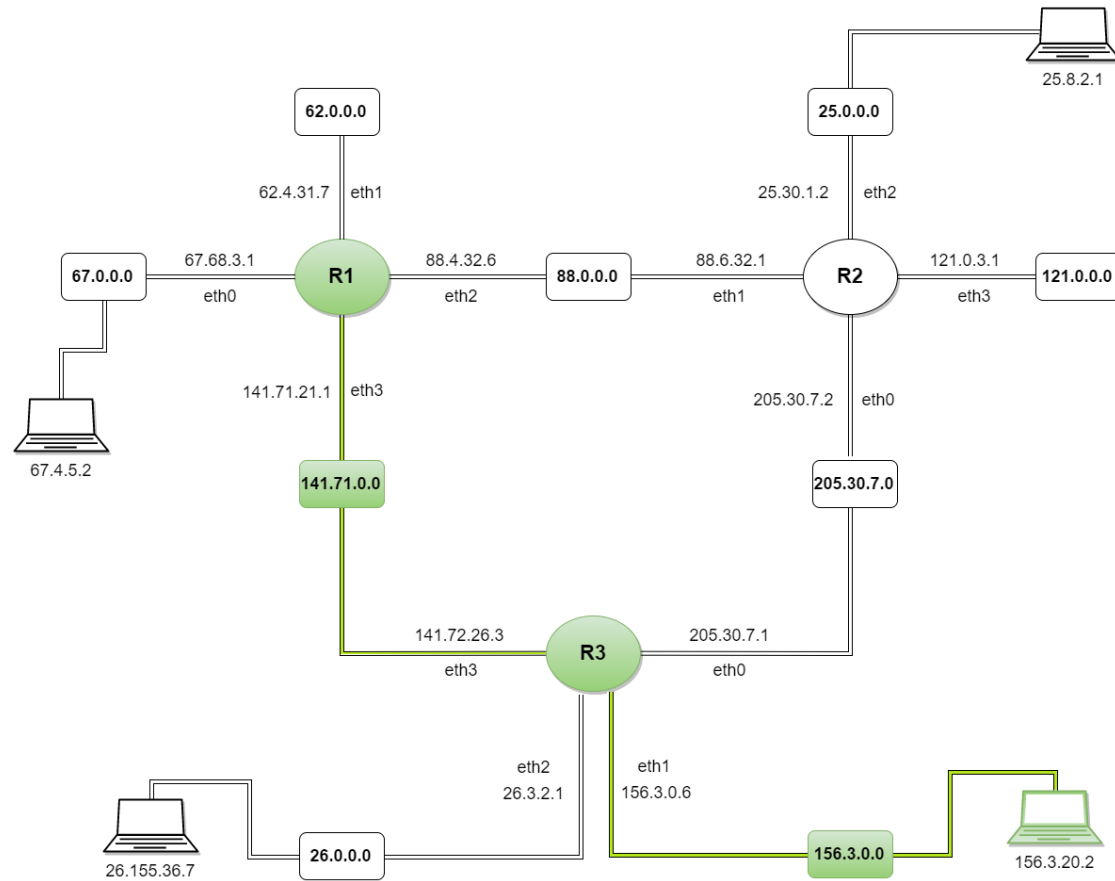


Figure 9: Request sent to domain name server 156.3.20.2 and response received

Fourthly, R1 creates the ip packet with destination address 26.155.36.7 and sends it via eth3 to R3 at ip address 141.72.26.3. R3 forwards the packet to the next hop at 26.155.36.7 (subdomain.webscienceexampledomain.com) which is the final destination in this case. A response is then sent back to R1 with the requested ip address as shown in the figure below.

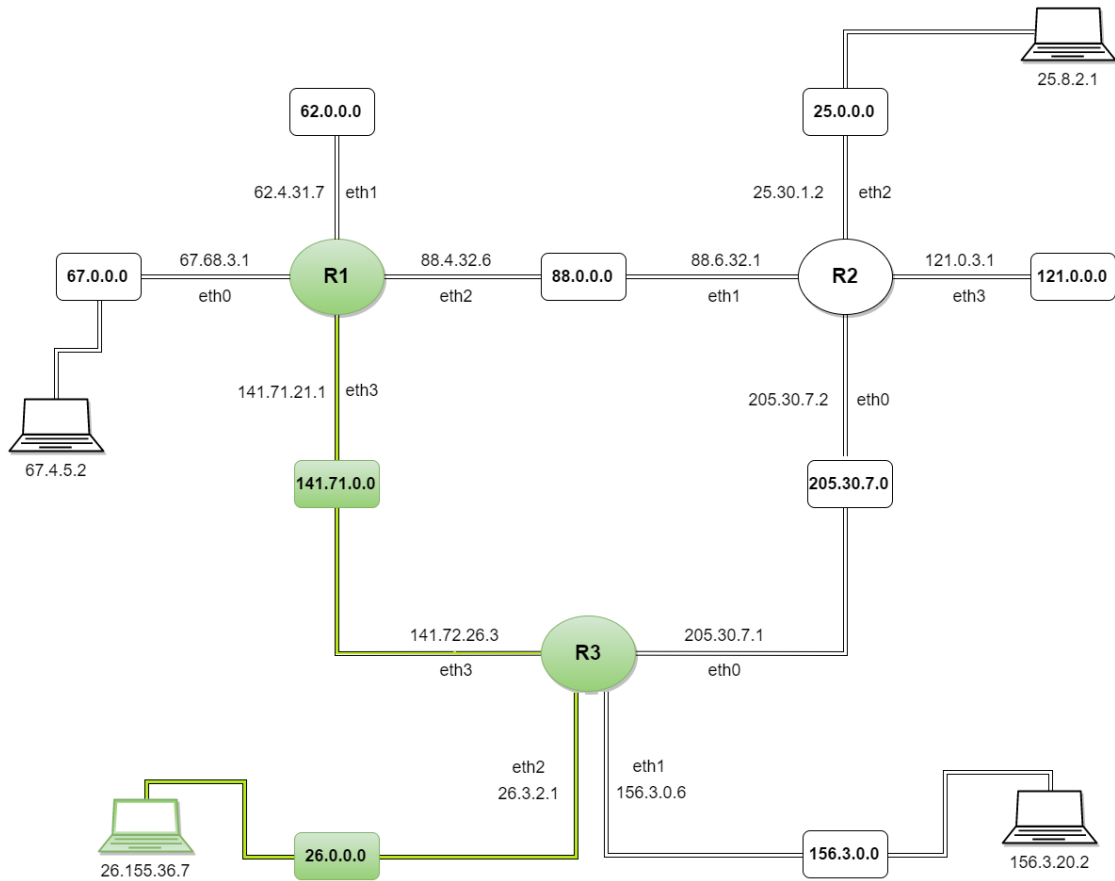


Figure 10: Request sent to sub-domain name server 26.155.36.7 and response received

Finally, R1 (ISP) sends the resolved query back to the client (ip 67.4.5.2) as shown in the figure below.

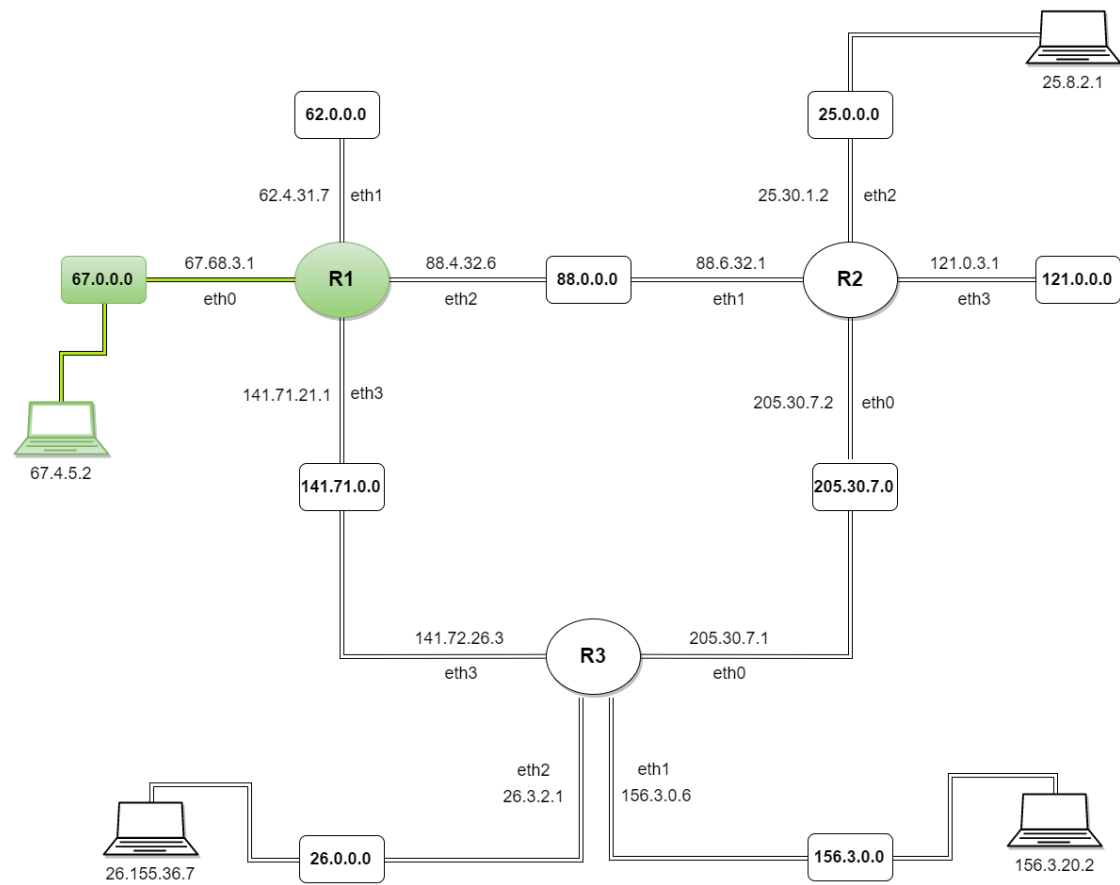


Figure 11: Response sent back to client

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 \LaTeX engine to LuaLaTeX.