

Introduction to Web Science

Assignment 2

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 Koblenz-Landau

Submission until: November 9, 2016, 10:00 a.m.

Tutorial on: November 11th, 2016, 12:00 p.m.

The main objective of this assignment is for you to use different tools with which you can understand the network that you are connected to or you are connecting to in a better sense. 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 Bravo

Daniyal Akbari (akbari@uni-koblenz.de)

Shriharsh Ambhore (ashriharsh@uni-koblenz.de)

Kandhasamy Rajasekaran (kandhasamy@uni-koblenz.de)

1 IP Packet (5 Points)

Consider the IPv4 packet that is received as:

4500 062A 42A1 8001 4210 XXXX C0A8 0001 C0A8 0003

Consider XXXX to be the check sum field that needs to be sent with the packet.

Please provide a step-by-step process for calculating the "Check Sum".

1. Add up all the 16 bits together except the checksum $4500 + 062A + 42A1 + 8001 + 4210 + C0A8 + 0001 + C0A8 + 0003 = 2\ D130$
2. If there is a carry then add it
2 is the carry. So $2 + D130 = D132$
3. If there is still a carry then add that as well.
there is no carry now
4. Get the One's complement - 2ECD

2 Routing Algorithm (10 Points)

UPDATE. The bold fonted numbers have been updated on Monday Nov. 7th. (If you already have done so feel free to use the old numbers. But the solution with the old version will be more complex than the solution with the updated numbers.)

You have seen how routing tables can be used to see how the packets are transferred across different networks. Using the routing tables below of Router 1, 2 and 3:

1. Draw the network [6 points]
2. Find the shortest path of sending information from 67.68.2.10 network to 25.30.3.13 network [4 points]

Table 1: Router 1

Destination	Next Hop	Interface
67.0.0.0	67.68.3.1	eth 0
62.0.0.0	62.4.31.7	eth 1
88.0.0.0	88.4.32.6	eth 2
141. 71 .0.0	141. 71 .20.1	eth 3
26.0.0.0	141.71.26.3	eth 3
156.3 .0.0	141.71.26.3	eth 3
205. 30.7 .0	141.71.26.3	eth 3
25.0.0.0	88.6.32.1	eth 2
121.0.0.0	88.6.32.1	eth 2

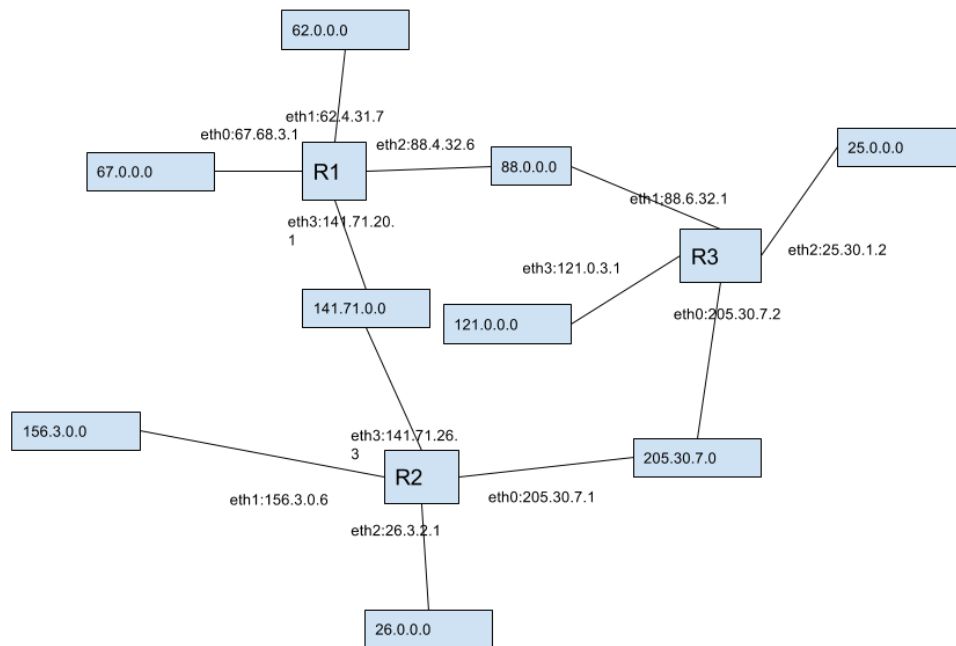
Table 2: Router 2

Destination	Next Hop	Interface
141. 71 .0.0	141.71.26.3	eth 3
205. 30.7 .0	205. 30.7 .1	eth 0
26.0.0.0	26.3.2.1	eth 2
156. 3 .0.0	156.3.0.6	eth 1
67.0.0.0	141. 71 .20.1	eth 3
62.0.0.0	141. 71 .20.1	eth 3
88.0.0.0	141. 71 .20.1	eth 3
25.0.0.0	205.30.7.2	eth 0
121.0.0.0	205.30.7.2	eth 0

Table 3: Router 3

Destination	Next Hop	Interface
205. 30 .7.0	205.30.7.2	eth 0
88.0.0.0	88.6.32.1	eth 1
25.0.0.0	25.30.1.2	eth 2
121.0.0.0	121.0.3.1	eth 3
156. 3 .0.0	205. 30 .7.1	eth 0
26.0.0.0	205. 30 .7.1	eth 0
141.0.0.0	205. 30 .7.1	eth 0
67.0.0.0	88.4.32.6	eth 1
62.0.0.0	88.4.32.6	eth 1

1)



2) Shortest path from 67.68.2.10 to 25.30.3.13

67.68.2.10 to 67.68.3.1 (R1 eth0)

R1 to 88.6.32.1 (R3 eth1)

R3 to 25.30.3.13

Reason - When sent from 67.68.2.10, the routing table in that particular machine will have default configuration to 67.68.3.1 and from there it goes to 88.6.32.1 (based on the routing table configuration for that router) and shares the message with all the machines in network 25.0.0.0 which will reach the recipient.

It takes only two hops through this way than the path R1-R2-R3.

3 Sliding Window Protocol (10 Points)

Sliding window algorithm, which allows a sender to have more than one unacknowledged packet "in flight" at a time, improves network throughput.

Let us consider you have 2 Wide Area Networks. One with a bandwidth of 10 Mbps (Delay of 20 ms) and the other with 1 Mbps (Delay of 30 ms) . If a packet is considered to be of size 10kb. Calculate the window size of number of packets necessary for Sliding Window Protocol. [5 points]

The transmission time of a data packet in first WAN is $(10 \text{ Kb}) / (10 \text{ Mbps}) = 1 \text{ msec}$.

The transmission time of a data packet in second WAN is $(10 \text{ Kb}) / (1 \text{ Mbps}) = 10 \text{ msec}$.

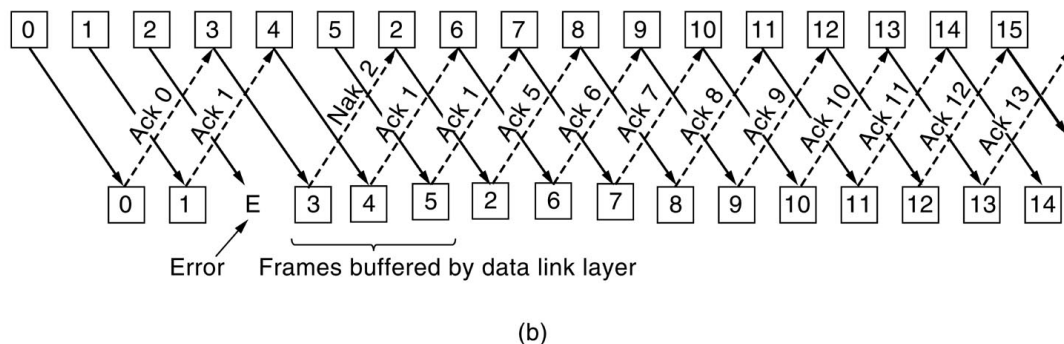
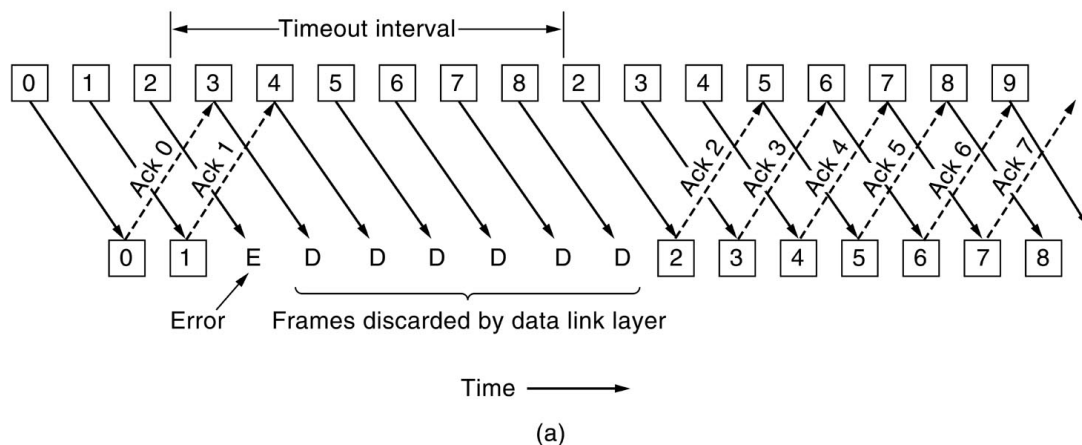
The total delay time is $20 + 30 = 50 \text{ msec}$. Thus an RTT for sending a data packet and receiving an acknowledgment for it is $50 + 1 + 10 = 61 \text{ msec}$.

To go as fast as possible, First WAN must use a window that is at least as large as the bandwidth-delay product, which is $(10 \text{ Mbps})(61 \text{ msec}) = 610000 \text{ Bits}$.

The question asks for how many data packets, which is given by:

$$\text{Ceiling}[610000 \text{ b} / (10 \text{ Kb/pkt})] = 61 \text{ pkts}$$

Since you now understand the concept of Window Size for Sliding Window Protocol and how to calculate it, consider a window size of 3 packets and you have 7 packets to send. Draw the process of **Selective Repeat Sliding Window Protocol** where in the 3rd packet from the sender is lost while transmission. Show diagrammatically how the system reacts when a packet is not received and how it recuperates from that scenario. [5 points]



Receiver B window size = 1 Discard all frames after error No ack - they will eventually get re-sent Can be lot of re-sends Wastes lot of bandwidth if error rate high.

DA needs buffers in memory to hold frames waiting for ack (might need to re-transmit them). Acks may get lost/damaged. "ack n" is interpreted by A as "ack everything up to n". Each frame has its own timer. Easy to simulate multiple timers in software.

Selective repeat (SR) (b) above

Receiver B window size = n Good frames after error are buffered (because can't send them to Network Layer NB yet, can't get rid of them) Bad frame 2 times out and is re-sent. At a timeout, sender A only sends the oldest un-ack-ed frame, not a sequence.

Or (in figure) B sends a Negative Ack (NAK) to provoke a re-send of frame 2 instead of waiting for timeout - may improve performance. If you know it will timeout, why wait? Notice B doesn't ack 3-5 while it is buffering them. It sends back "Ack 1" which means "Everything ok up to and including 1". Once 2 gets through, B can send a whole bunch (2-5) to NB layer and can then jump to "Ack 5" ("Everything ok up to and including 5")

4 TCP Client Server (10 Points)

Use the information from the [socket](#) documentation and create: [4 points]

1. a simple TCP Server that listens to a
2. Client

Note: Please use port 8080 for communication on `localhost` for client server communication.

Given below are the following points that your client and server must perform: [6 points]

1. The *Client* side asks the user to input their name, age & *matrikelnummer* which is then sent to the server all together.
2. Develop a protocol for sending these three information and subsequently receiving each of the information in three different lines as mentioned in the below format. Provide reasons for the protocol you implemented.
3. Format the output in a readable format as:
Name: Korok Sengupta;
Age: 29;
Matrikelnummer: 2122356

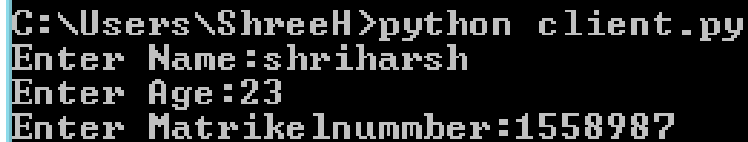
Provide a snapshot of the results along with the code.

5 Python

```
1: >>> """
2: Client Code
3: Created on Mon Nov 7 03:36:00 2016
4: @author: Shriharsh Ambhore
5: @author: Kandhasamy Rajasekaran
6: @author: Daniel Akbari
7: """
8:
9:
10: import socket
11: import json
12:
13: dict={'Name':'','Age':'','Matrikelnummer':''}
14:
15: dict['Name']=input('Enter Name:')
16: dict['Age']=input('Enter Age:')
17: dict['Matrikelnummer']=input('Enter Matrikelnummer:')
18:
19: s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
20: s.connect(('localhost', 8080))
21: s.send(json.dumps(dict).encode())
22: data = s.recv(4096)
23: s.close()
24: print('server response:\n',data.decode('utf-8'))
```

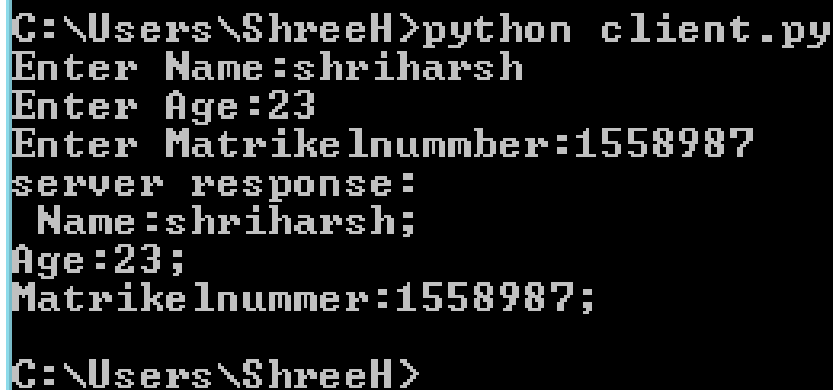
```
1: """
2: Server Code
3: Created on Mon Nov 7 03:36:00 2016
4: @author: Shriharsh Ambhore
5: @author: Kandhasamy Rajasekaran
6: @author: Daniel Akbari
7: """
8:
9: import socket
10: import json
11:
12: server_details=('localhost',8080)
13:
14: server_socket=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
15: server_socket.bind(server_details)
16: server_socket.listen(5)
17:
18: conn,address=server_socket.accept()
19: print ("connected on",address)
20: while True:
21:     print('listening')
22:     dataFromClient=conn.recv(4096)
```

```
23:     print('Data from client:',dataFromClient.decode())
24:
25:     recvdDict=json.loads(dataFromClient.decode())
26:
27:     returnString="Name:"+recvdDict['Name']+";\n"+"Age:"+recvdDict['Age']+";\n"+"Matrikelnummer:"+recvdDict['Matrikelnummer']+";\n"
28:
29:     conn.send(returnString.encode())
30:     conn.close()
```

A terminal window with a black background and white text. The prompt is 'C:\Users\ShreeH>'. The user enters 'python client.py'. The program then prompts 'Enter Name:' and the user enters 'shriharsh'. It then prompts 'Enter Age:' and the user enters '23'. Finally, it prompts 'Enter Matrikelnummer:' and the user enters '1558987'.

```
C:\Users\ShreeH>python client.py
Enter Name:shriharsh
Enter Age:23
Enter Matrikelnummer:1558987
```

Figure 1: Client Input

A terminal window with a black background and white text. The prompt is 'C:\Users\ShreeH>'. The user enters 'python client.py'. The program then prompts 'Enter Name:' and the user enters 'shriharsh'. It then prompts 'Enter Age:' and the user enters '23'. Finally, it prompts 'Enter Matrikelnummer:' and the user enters '1558987'. After the input, the program outputs 'server response:' followed by 'Name:shriharsh;', 'Age:23;', and 'Matrikelnummer:1558987;'. The prompt 'C:\Users\ShreeH>' is shown again.

```
C:\Users\ShreeH>python client.py
Enter Name:shriharsh
Enter Age:23
Enter Matrikelnummer:1558987
server response:
  Name:shriharsh;
Age:23;
Matrikelnummer:1558987;

C:\Users\ShreeH>
```

Figure 2: Server Response to client

Protocol implemented for sending Name,age,matrikelnummer is as follows: Input entered by user is added to the dict data structure where Name, Age, Matrikelnummer is the key and the input entered by the user is the value respectively. The dict data structure formed is transformed to a light weight JSON format and send to server which ensures that 'Key'- 'Value' mappings is retained. On the server side all this information is again converted to a dict data structure and value of each individual element is accessed to do

the further processing and the result is send back to the client. This way no delimiter is required to identify each input, first: as there is a high probability that the data entered by the use may contain any of the delimiter used, secondly: a value entered by the user is always associated with the key and we do not need to worry about the sequence in which the data is entered by the user to differentiate among other input.

Important Notes

Submission

- Solutions have to be checked into the github repository. Use the directory name `groupname/assignment2/` 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".

L^AT_EX

Currently the code can only be build using **LuaLaTeX**, so make sure you have that installed. If on Overleaf, go to settings and change the **L^AT_EX**engine to **LuaLaTeX** in case you encounter any error