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.
Team :	Golf
	Mtarji Adam
Members:	Deepak Garg
	Atique Baig

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.

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".

Solution:

The checksum is calculated by summing all the 16bit words of the header expect the checksum, then applying the one's compliment to the result, the sum of all the 16bit words including the checksum shold be 0 or FFFF in the absence of any corrupt data.

By adding these Hexadecimals:

$4500 + 062A + 42A1 + 8001 + 4210 + C0A8 + 0001 + C0A8 + 0003 = D132$

The checksum value is the one's complement of 4FDD, which is done by applying subtracting FFFF - D132.
One's Complement of 4FDD = 2ECD

So, 2ECD is the checksum which needs to be sent with the packet from the sender side.

For Checking Error:
D132 + 2ECD = FFFF.
From this we find that no error occur while transferring packet from sender to receiver side.

2 Routing Algorithm (10 Points)

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:

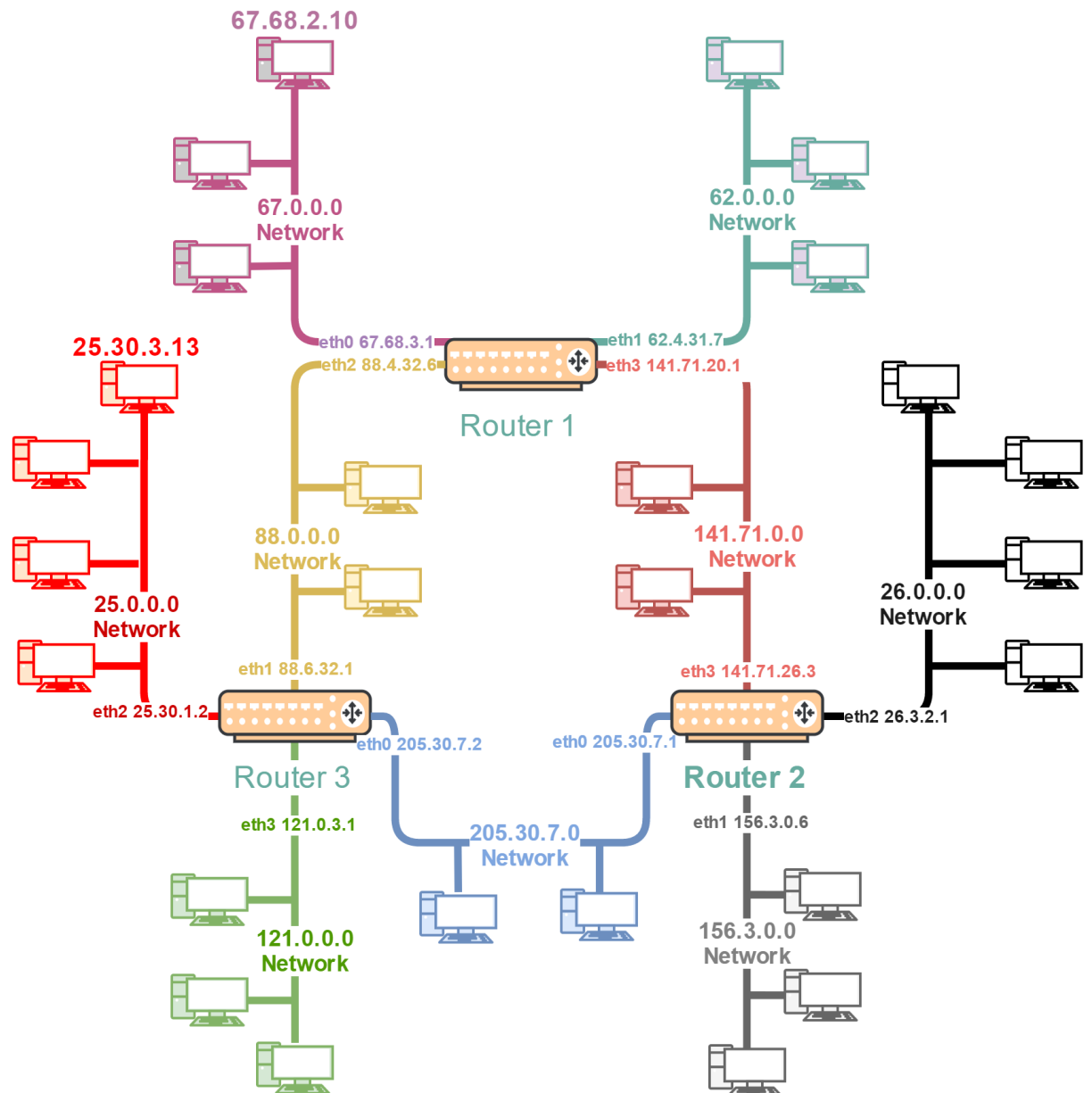1. Draw the network [6 points]
   Solution:

Figure 1: Network Diagram

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

Solution:

To find out which is the shortest path from 67.68.2.10 to 25.30.3.13, we need to evaluate each possible path and count the number of hops, the path with the lowest number of hops would be the shortest.
Our starting point is the 67.0.0.0 network and destination is 25.0.0.0 network. This means that to go out from the 67.0.0.0 network we need to go through router 1. In the other side, the destination network is behind router 3 and only accessible through this router. Now lets identify all possible routs that leads to the 25.0.0.0 network from 67.0.0.0 through router 1 and router 3.

1. hops: eth0 67.68.3.1 > eth1 88.6.32.1 > eth2 25.30.1.2 = 3 hops.

2. hops: eth0 67.68.3.1 > eth3 141.71.26.3 > eth0 205.30.7.2 > eth2 25.30.1.2 = 4 hops.

since the first path has the lowest number of hops, it is the shortest possible path from 67.68.2.10. to 25.30.3.13

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	eth2
141.0.0.0	141.30.20.1	eth3
26.0.0.0	141.71.26.3	eth 3
150.0.0.0	141.71.26.3	eth 3
205.0.0.0	141.71.26.3	eth3
25.0.0.0	84.6.32.1	eth1
121.0.0.0	84.6.32.1	eth1

Table 2: Router 2

Destination	Next Hop	Interface
141.0.0.0	141.71.26.3	eth 3
205.0.0.0	205.25.71.1	eth 0
26.0.0.0	26.3.2.1	eth2
156.0.0.0	156.3.0.6	eth 1
67.0.0.0	141.71.26.3	eth 3
62.0.0.0	141.71.26.3	eth 3
88.0.0.0	141.71.26.3	eth3
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
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
205.0.0.0	205.3.7.2	eth 0
67.0.0.0	88.4.32.6	eth 2
62.0.0.0	88.4.32.6	eth 2
141.0.0.0	88.4.32.6	eth 2
26.0.0.0	205.25.7.1	eth 0
156.0.0.0	205.25.7.1	eth 0

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`]

Solution:

1. Wide Area Network
   Bandwidth = 10MBps
   RTT = 20ms

   In 10^3ms we can send $8{\times}10^7$bits, lets calculate how much we can send in the RTT delay of 20ms :
   b1 = $20{*}8{\times}10^7$ / 10^3 = $160{\times}10^4$bits

   So in 20 ms we can send 160,0000 bits, and assuming each packet cannot exceed the size of 80,000 bits, the window size should be :

   w1 = $160{\times}10^4$bits / $80{\times}10^3$bits = 20
   In this case window size is 20.

2. Wide Area Network
   Bandwidth = 1MBps
   RTT = 30ms

   In 10^3ms we can send $8{\times}10^6$bits, lets calculate how much we can send in the RTT delay of 30ms :
   b2 = $30{*}8{\times}10^6$ / 10^3 = $240{\times}10^3$bits

   So in 30 ms we can send 240,000 bits, and assuming each packet cannot exceed the size of 80,000 bits, the window size should be :
   w2 = $240{\times}10^3$bits / $80{\times}10^3$bits = 3
   In this case window size is 3.

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**]

Solution:

1. In the Diagram below window Size of 3 packets is sent to receiver and receiver sent back the acknowledgment of packet received.

2. While sending from sender P3 lost and no acknowledgment is sent back.

3. Sender will send the P4 and P5 to receiver and will wait until acknowledgment for P3 is received and if times out(in case of packet loss, the reaction of the transmitter is delayed which results in a lower utilization of time).

4. Then sender will resent the P3 and receive the ack.

5. The sender now moves its window to send next packets after receiving ack. for every packet.
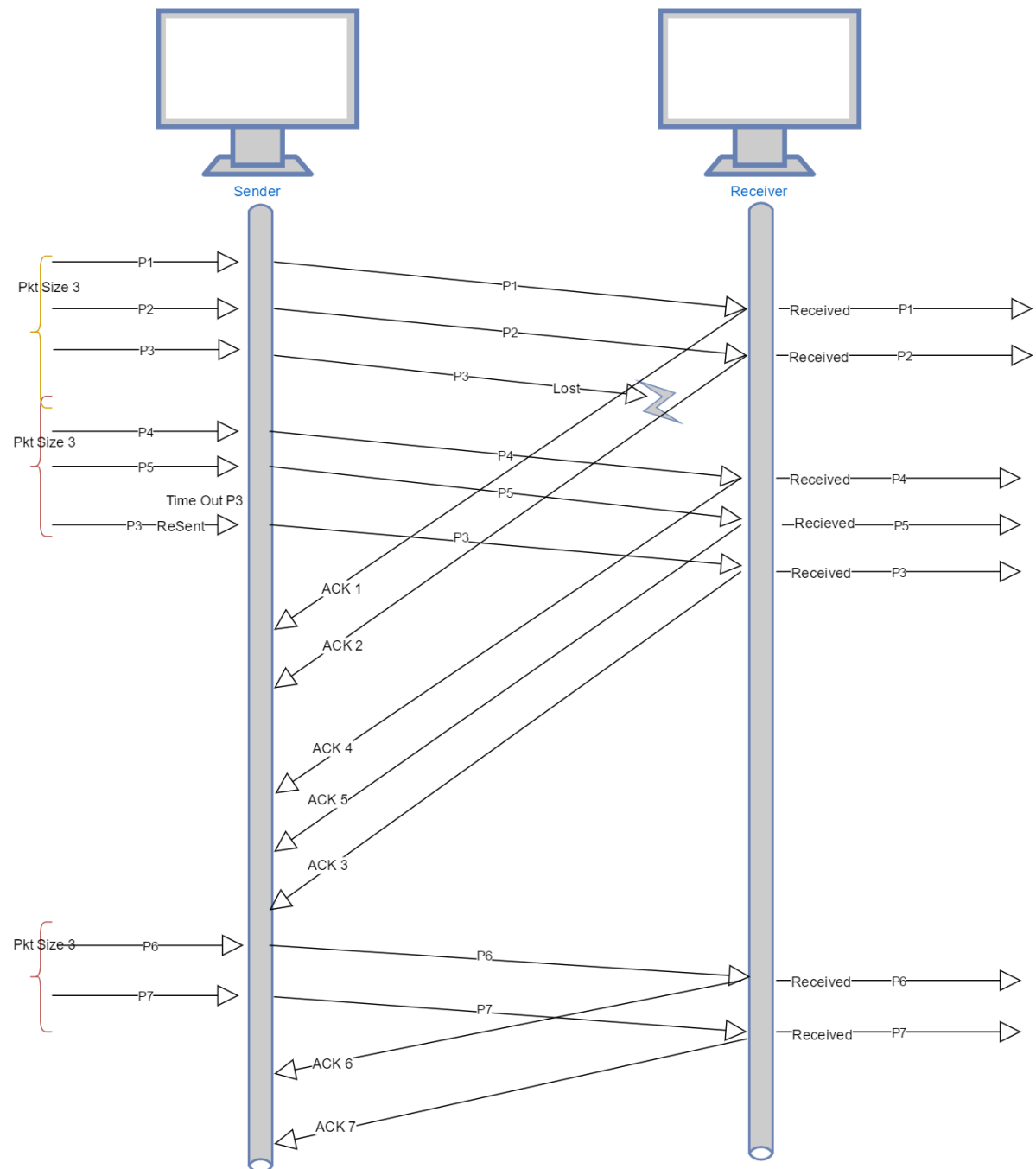
Figure 2: Selective Repeat Diagram

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 Client

```python
# modules
import socket

#we create a connection-oriented sockets (stream socket) that uses TCP
tcp_server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#we bind it to localhost:8080
tcp_server_socket.bind(('localhost', 8080))
tcp_server_socket.listen(5)
#we listen and accept incoming connections
c,addr= tcp_server_socket.accept()
#we read received data (as a packet of 1024)
print(c.recv(1024))
```

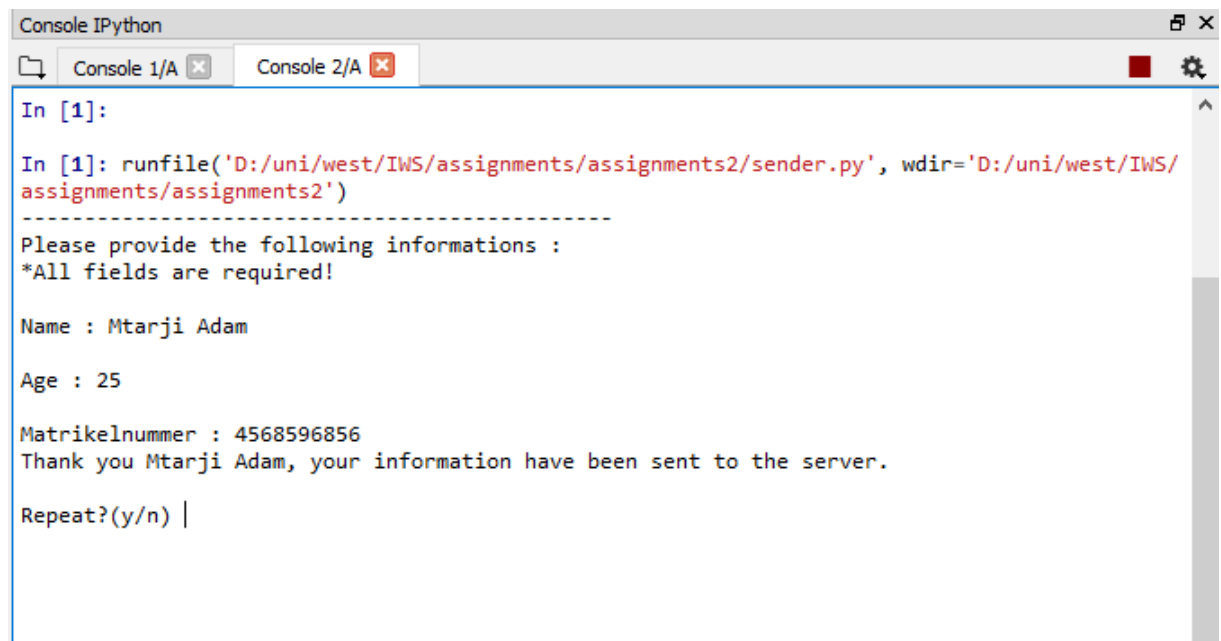**Figure 3:** Simple TCP server

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.

   Solution:
   We create a class named 'Sender' which we will use to prompt the user for input of his name, age and matrikelnummer.

Figure 4: Providing input to the client

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.

<u>Solution</u>:
We want to reliably send the input to the server, so we use a connection oriented socket to stream the data via TCP. We also want to be able to identify each input, so we use the 'Sender' class to store the input in a json format and encode it to utf8 so we can send it via the socket stream, we then receive the packet on the server side using a 'Receiver' class and decode the stream, we load the json to get a dictionary containing the user inputs, then we format and output them.

<u>sender.py</u>:

```python
# modules
import socket
import json

"""
the Sender class will play the role of a client object
"""
```

```python
class Sender():

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

    # the send_packet function allows the Sender objec to send a message to the serv
    # First we create a socket, then we attempt to connect to the server
    # The message is dumped using json library and encoded in utf8 to be sent trough
    # 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(json.dumps(message).encode('utf8'))
            self.socket.close()
            print('Thank you {0}, your information have been sent to the server.'.fo
        except socket.error as exc:
            print ("Error sending data to locolhost:8080 : %s" % exc)

    # The client function allows to prompt the use for input and prepare a message
    # to send 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':
            name=''
            age=''
            matrikelid=''
            print('---------------------------------------------')
            print('Please provide the following informations :')
            print('*All fields are required!')
            while not(self.field_validation(name)):
                name=input('Name : ').strip()

            while not(self.field_validation(age,True)):
                age=input('Age : ').strip()


            while not(self.field_validation(matrikelid)):
                matrikelid=input('Matrikelnummer : ').strip()

            message = {'name':name,'age':age,'matrikelid':matrikelid}
            self.send_packet(message)
            self.chain_query()
```

11

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

    # field_validation function allows the client to validate the input
    def field_validation(self,field,is_numeric=False):
        if len(field) ==0:
            return False
        elif is_numeric:
            if field.isdigit():
                return True
            else:
                print('Please provide a valid number')
                return False
        else :
            return True

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

receiver.py:

```python
# modules
import socket
import json
import datetime as dt

"""
Receiver class will be able to play the role of a server object
"""
class Receiver():
    # in the __init__ function we asign a server socket and bind it to the localhos
    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)
```
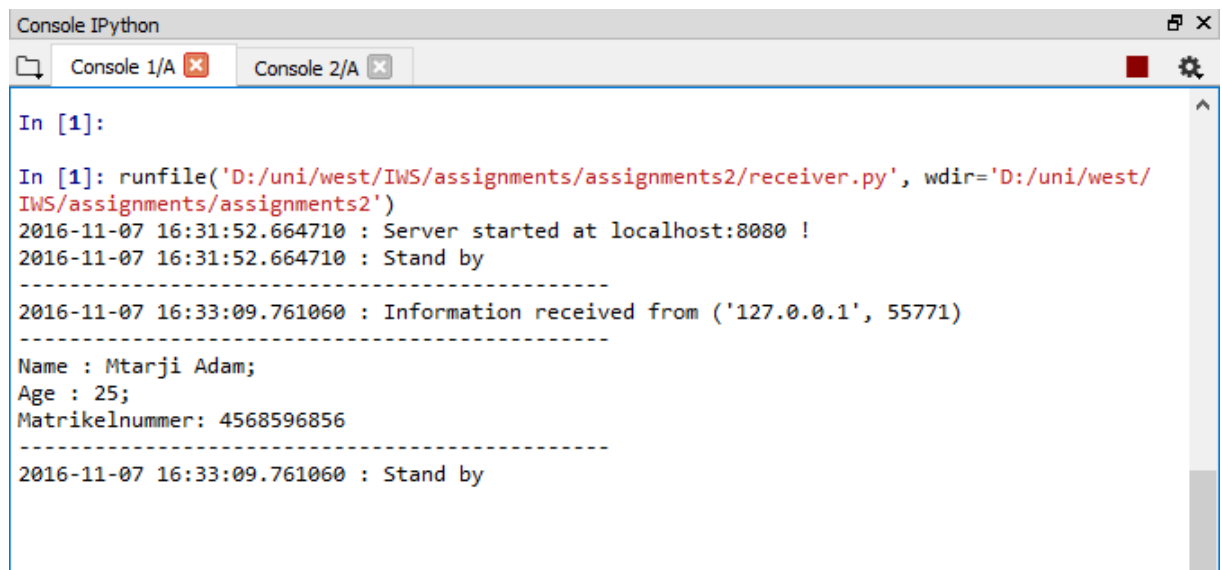
```python
    # the receiv_packet function will be listening to any connection request on the
    # then it reads the packet sent and decode it using the utf8 encoding
    # the decoded packets can then be loaded using json library to rebuild the dict
    # the information is the represented in the required format, we also add a times
    def receiv_packet(self):
        c,addr= self.server_socket.accept()
        p=json.loads(c.recv(1024).decode('utf8'))
        print('-----------------------------------------------')
        print('{0} : Information received from {1}' .format(dt.datetime.now(),addr))
        print('-----------------------------------------------')
        print('Name : {0};'.format(p['name']))
        print('Age : {0};' .format(p['age']))
        print('Matrikelnummer: {0}' .format(p['matrikelid']))
        print('-----------------------------------------------')
        c.close()


# 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 keap 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. Format the output in a readable format as:
   Name: Korok Sengupta;
   Age: 29;
   Matrikelnummer: 21223ert56

Figure 5: Information received by the server

Provide a snapshot of the results along with the code.

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".