



UE17EC351: Computer Networks

A Project Report on

Online Voting System using IPv6 addressing

by

Kaushal K Hebbar (PES1201701547)

Shreevathsa P K (PES1201701599)

under the guidance of

Prof. Rajasekar M.

Dept. of ECE,

PES University,

Bangalore.

Table of Contents

1. Problem Statement	2
2. Introduction	2
3. Configuration.....	3
3.1. Server configuration.....	3
3.2. Client configuration	3
4. IPv6 vs IPv4 Addressing	4
5. Communication Workflow	5
6. Network diagram.....	6
7. Algorithm	7
8. Result.....	8
8.1. Output Screenshots	8
8.2. Wireshark Capture	11
9. Conclusion.....	13
10. Code.....	14
10.1. Client code – ‘online_voting_client.py’	14
10.2. Server Code – ‘online_voting_server.py’	15

1. Problem Statement

The word “vote” simply means to choose/elect something/someone from a list. The main objective of voting (as far as the citizens of a county are concerned) is to come up with the representatives of peoples’ choice.

Many countries including India have problems with their voting systems. Considering the huge growth of population in our country, it is not wise to use the “traditional” polling systems where the public must physically be present at the polling booth during the time of casting their votes. The aged and the specially-abled often find it hard to commute to the poll booths and to wait in the long queues. Moreover, the time and money spent on conducting such a large scale operation must be reduced.

As our nation is thriving with digitization, this “Online Voting System” seeks to address the above issues and provides a hassle-free user experience to the electorate.

2. Introduction

We have implemented a simple online voting (e-voting) platform using socket programming. In this system, anyone who has registered is eligible to vote from his/her laptop remotely by using his/her mobile hotspot. This enables the electorate to cast his votes from wherever he/she is. A single server is run by the administrator (again using his/her mobile hotspot) and multiple clients (electorates) connect to them using IPv6 addressing This system can be used for a simple “voting for the best singer in PESU” to “voting for the best MLA in our area” !!

3. Configuration

The server and client programs are coded using Socket Programming in Python language. The system is made robust to any kind of sudden power-outage during the live voting session i.e., it does not lose track of the votes already casted.

3.1. Server configuration

The server is run by the administrator and shall have the following setup:

- 1) A laptop/PC connected wirelessly using personal mobile hotspot that is connected to an active GPRS network.
- 2) Its Public IPv6. (128 bit address)
- 3) A port number common to both server and client.
- 4) Python 3.7.x installed along with its IDE.
- 5) The python file named “online_voting_server.py”.
- 6) A database of all the registered voters and their passwords in the “Voters_database.txt”
- 7) A database of the candidate names in the “Candidates.txt”
- 8) A database “Voter_names.txt” having the list of voters already voted.
- 9) Finally, a “Votes_record.txt” file containing the polling results after each round.

3.2. Client configuration

The client is run by the electorate and shall have the following setup:

- 1) A laptop/PC connected wirelessly using personal mobile hotspot.
- 2) Server’s Public IPv6 address.
- 3) A port number common to both server and client.
- 4) Python 3.7.x installed along with its IDE.
- 5) The python file named “online_voting_client.py”.
- 6) The username and password provided to him/her.

4. IPv6 vs IPv4 Addressing

For IPv4 to communicate with the outside world we need to use Network Address Translation (NAT). This is so because, in IPv4 each device has a private IP address and a private port assigned in its LAN. If one has to connect to a laptop which is connected to the internet through a different LAN using IPv4, then it's hard to do so, as we can only get Public IP address and public port of that network but not the computer's Private IP and private port.

This could have been overcome if we had access to the Router for modifying the NAT settings by adding our customized Private IP and port to the Public port mapping. But since we didn't have the administrator access to do so, the entire procedure was found to be very involved.

So, we came up with idea of using IPv6 instead of IPv4 itself. In IPv6 addressing, we don't have Network Address Translation and thus, the concept of Private address doesn't exist. The laptop gets the same IP address as that of the phone with only the ports changed. This IP address is dynamic and changes every time you turn on your GPRS. The voting Client must be notified about the server IP address.

The IPv6 address of the PC can be found out by typing "*What is my IP*" on Google Chrome browser or simply the command "*ipconfig*" on the cmd.

An example of IPv6 (128 bit address) address is as shown below.

2409:4073:286:8aa5:b465:ae16:110f:e498

5. Communication Workflow

The following steps shall be performed by the client and server during the online voting session: (Fig 4.1 summarizes the TCP workflow)

- 1) The server shall inform its Public IP to all the clients prior to the voting session. This is done by typing *ipconfig* on the cmd.
- 2) The administrator runs the server file by the following command entered on his/her cmd.

```
>> python online_voting_server.py
```
- 3) Server uses the *socket.socket(socket.AF_INET6, socket.SOCK_STREAM)* function to create a socket. The first argument specifies the Internet address family for IPv6 and the second one indicates TCP sockets used to transport messages in the network layer.
- 4) The *bind((host, port))* function associates the socket with a specified network interface and port number on the computer.
- 5) The *listen(max)* function enables the server to accept connections and makes it a “listening socket”. The *max* value specifies the number of unaccepted connections that the system will allow before refusing new ones. (queue length)
- 6) At the same time several electorates run the client file by the typing the following command on his/her cmd.

```
>>python online_voting_client.py
```
- 7) Like the server, the client creates a socket using the *socket.socket(socket.AF_INET6, socket.SOCK_STREAM)* function and uses *connect((host, port))* method to connect to the server.
- 8) The *accept()* method in the server blocks and waits for an incoming connection. When a client connects, it returns a new socket object representing the connection and a tuple holding the address of the client. This socket is used to communicate with the client and is distinct from the listening socket that the server is using to accept new connections.
- 9) Hence, the server accepts only one client at a time queuing the rest and mutual communication happens via *send()* and *recv()* methods.
- 10) Finally, the connected client closes its connection via *close()* function and the waiting client automatically gets connected to the server. Only after all the clients close, will the server exit.

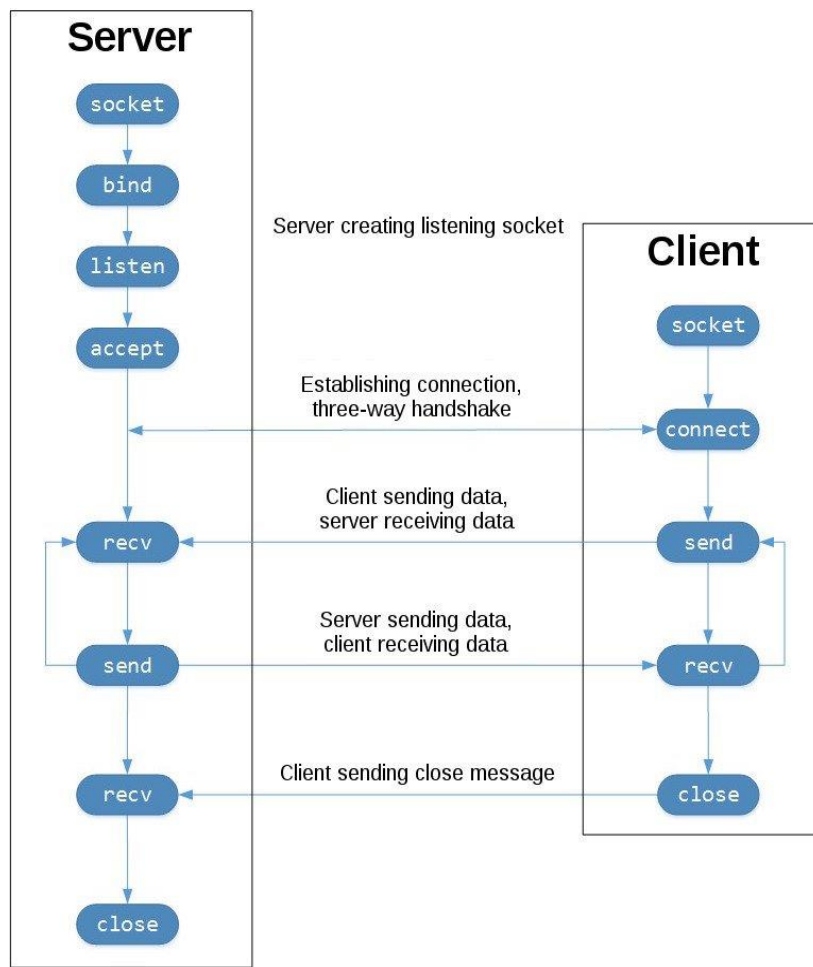


Fig. 5.1 TCP workflow

6. Network diagram

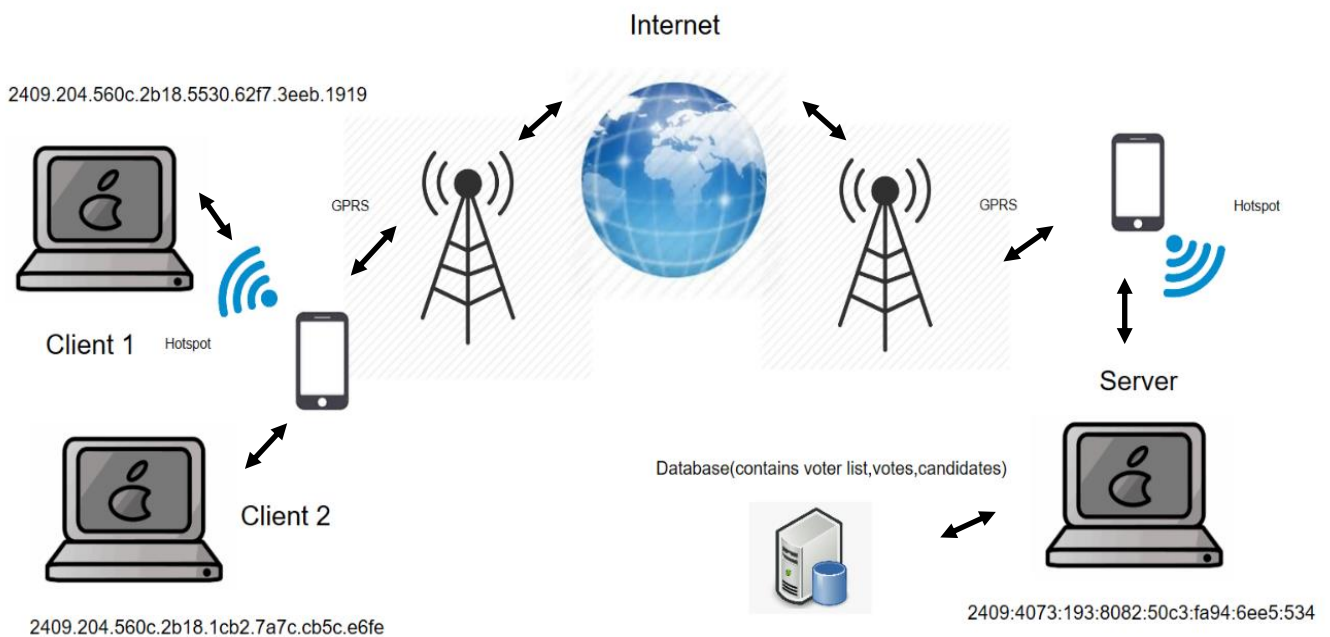


Fig. 6.1 Network diagram

7. Algorithm

The logic behind the whole system is given by the following steps of algorithm:

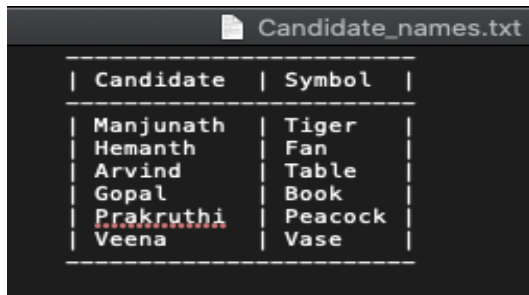
- 1) After successful interconnection connection between client and server, the administrator is asked for the number of clients connecting in the present round of polling.
- 2) The server then welcomes the client and asks for the client's username and password in the format 'usr-pwd' here 'first_name-last_4_digits_SRN'.
- 3) These credentials are then verified by the server with its database.
- 4) If electorate enters wrong credentials, one last chance is given for re-entry, failing which his/her vote is cancelled and he/she will no longer be eligible to vote.
- 5) Further, it's made sure that every person can vote only once failing which a message stating the same will be displayed on the client side.
- 6) Once the client enters the right credentials, he/she is given a list of candidates to cast his/her vote for.
- 7) The server now checks for the entered candidate is in its database. If it does not find a matching name, the server sends a message stating that the client's vote is cancelled. If the name matches then, that vote is appended.
- 8) At the back end, the server updates the current votes with the previous round results, updates the list of voters already voted, etc. stores them in different .txt files. This done to ensure that no data is lost during any sudden power-outage or any other error shutdowns.

8. Result

8.1. Output Screenshots

The following screenshots were taken during the time of an actual trial voting session. The client was running on the MS Windows 10 OS and the server on Mac 10.15 OS.

The Fig. 8.1-8.3 show the available .txt files on the server PC prior to starting the session.



Candidate	Symbol
Manjunath	Tiger
Hemanth	Fan
Arvind	Table
Gopal	Book
Prakruthi	Peacock
Veena	Vase

Fig. 8.1.1

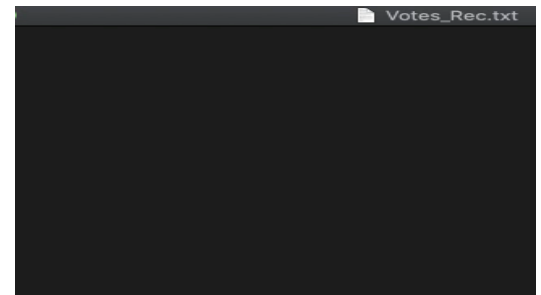
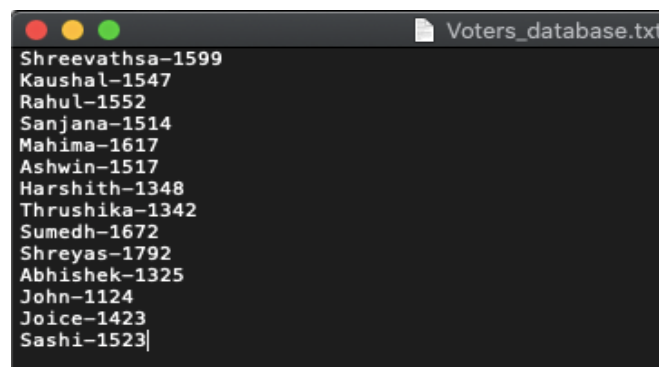


Fig. 8.1.2



```

Shreevathsa-1599
Kaushal-1547
Rahul-1552
Sanjana-1514
Mahima-1617
Ashwin-1517
Harshith-1348
Thrushika-1342
Sumedh-1672
Shreyas-1792
Abhishek-1325
John-1124
Joice-1423
Sashi-1523

```

Fig 8.1.3

Fig. 8.1.4 shows mobile hotspot connection and the server's IPv6 address. The 'host' variable in the client file stores the same.

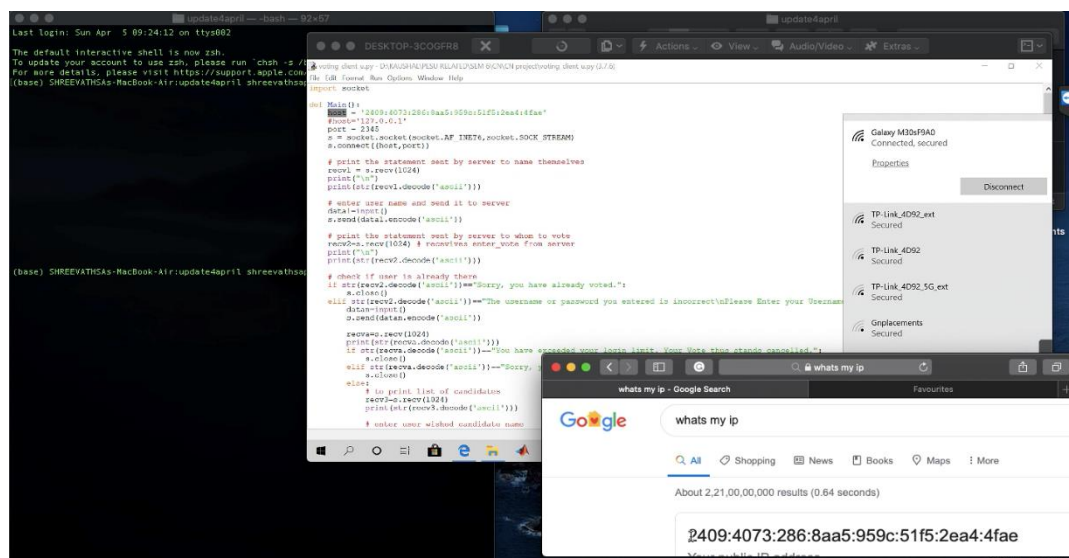


Fig 8.1.4

The client PC was shared with the server using “Team Viewer” application so that the interaction could be captured on a single screen. Fig. 8.1.5 shows the initial welcome message on the client (right) and the voter count on the server (left).

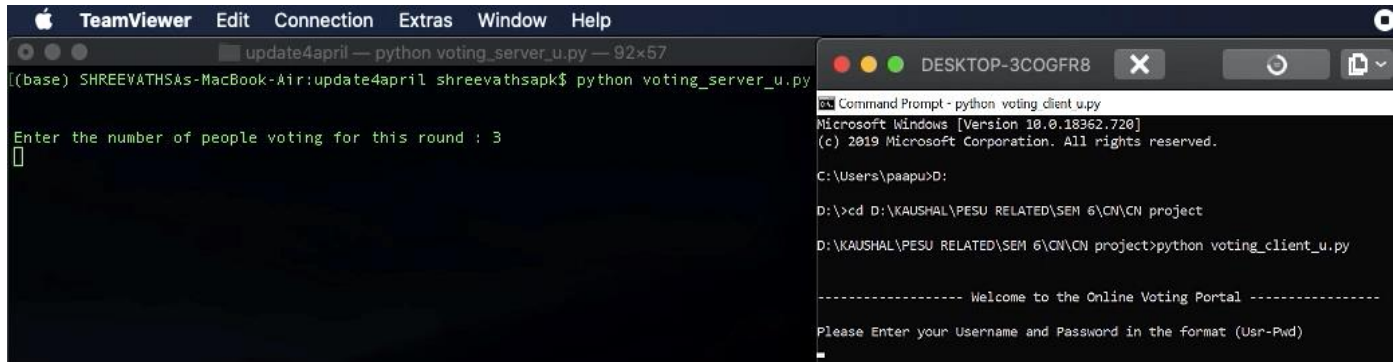


Fig. 8.1.5

Fig. 8.1.6 shows the screen after the first successful vote.

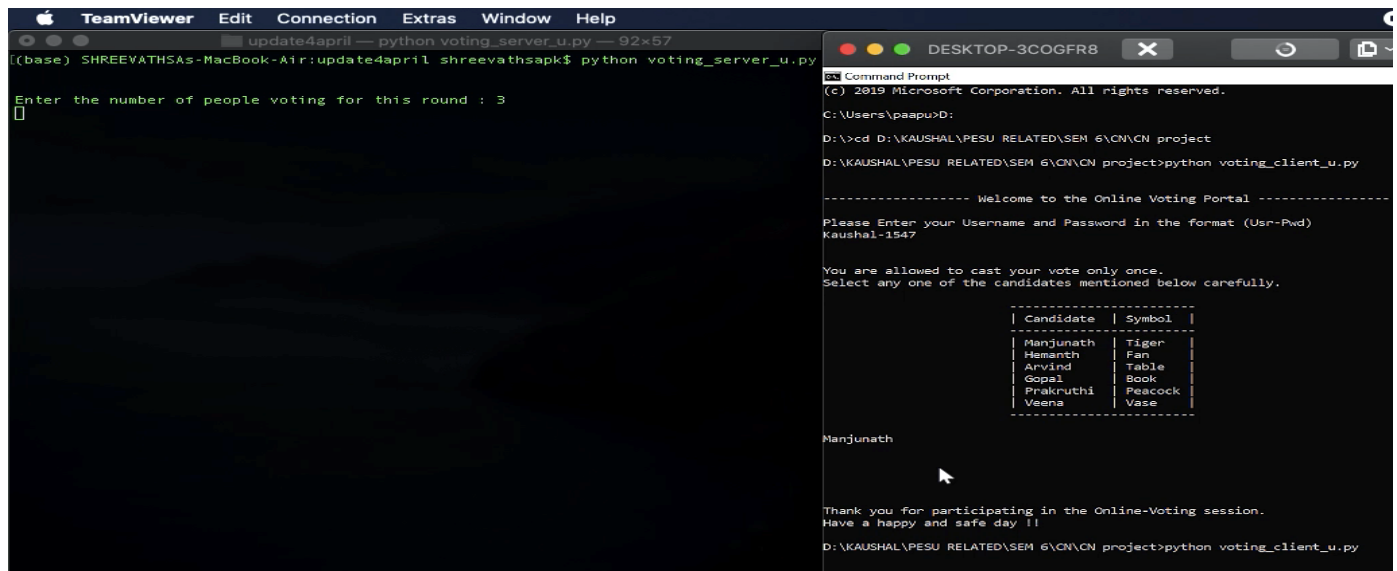


Fig. 8.1.6

Fig. 8.1.7 – Second chance to vote in case of wrong credentials, 8.1.8 – Vote cancelled since wrong credentials in the second chance, 8.1.9 – Vote cancelled since wrong candidate name.

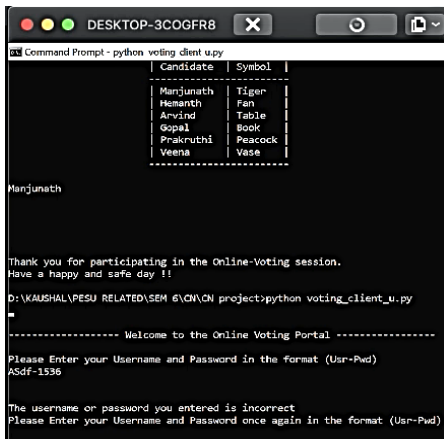


Fig. 8.1.7

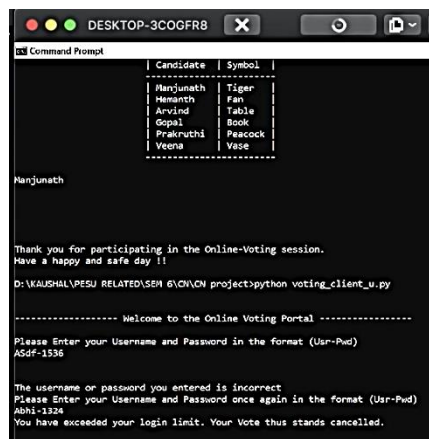


Fig. 8.1.8

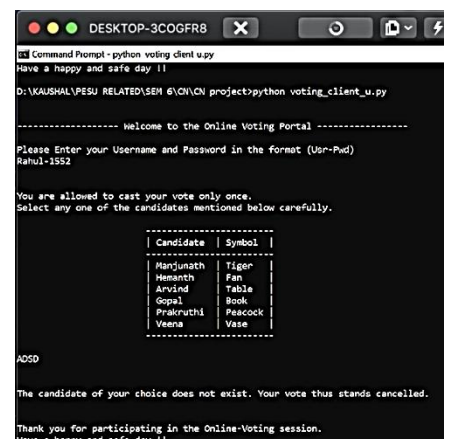


Fig. 8.1.9

Fig. 8.1.10 and 8.1.11 shows the server messages for continuing the polling rounds and for declaring the results respectively.

```

Terminal Shell Edit View Window Help
update4april — bash — 92x57
((base) SHREEVATHSAs-MacBook-Air:update4april shreevathsapk$ python voting_server_u.py

Enter the number of people voting for this round : 3
Should the final results be declared now? (Y/N) : N
The final results will be declared after further rounds.
(base) SHREEVATHSAs-MacBook-Air:update4april shreevathsapk$ python voting_server_u.py

DESKTOP-3COGFR8
Command Prompt

Candidate | Symbol
-----|-----
Manjunath | Tiger
Hemanth   | Fan
Arvind    | Table
Gopal     | Book
Prakruthi | Peacock
Veena     | Vase

Thank you for participating in the Online-Voting session.
Happy and safe day !!

D:\PESU RELATED\SEM 6\ON\ON project>python voting_client_u.py

----- Welcome to the Online Voting Portal -----
Enter your Username and Password in the format (Usr-Pwd)
:

Name or password you entered is incorrect
Enter your Username and Password once again in the format (Usr-Pwd)
:
exceeded your login limit. Your Vote thus stands cancelled.

Thank you for participating in the Online-Voting session.
Happy and safe day !!

D:\PESU RELATED\SEM 6\ON\ON project>_

```

Fig. 8.1.10

```

Terminal Shell Edit View Window Help
update4april — bash — 80x52
Last login: Sun Apr 5 10:21:47 on ttys000
The default interactive shell is now zsh.
To update your account to use zsh, please run 'chsh -s /bin/zsh'.
For more details, please visit https://support.apple.com/kb/HT200050.
(base) SHREEVATHSAs-MacBook-Air:update4april shreevathsapk$ python voting_server_u.py

Enter the number of people voting for this round : 2
Should the final results be declared now? (Y/N) : Y
-----
RESULTS DECLARED
-----
Candidate | Symbol | Votes
-----|-----|-----
Manjunath | Tiger  | 3
Hemanth   | Fan    | 0
Arvind    | Table  | 0
Gopal     | Book   | 0
Prakruthi | Peacock| 0
Veena     | Vase   | 0
-----

***** Online Voting Portal Closing *****
***** Thanks all for voting !! *****

The Winner is:Manjunath
(base) SHREEVATHSAs-MacBook-Air:update4april shreevathsapk$

DESKTOP-3COGFR8
Command Prompt

The candidate of your choice does not exist. Your vote thus stands cancelled.

Thank you for participating in the Online-Voting session.
Have a happy and safe day !!

D:\KAUSHAL\PESU RELATED\SEM 6\ON\ON project>python voting_client_u.py

----- Welcome to the Online Voting Portal -----
Please Enter your Username and Password in the format (Usr-Pwd)
Sanjana-1514

You are allowed to cast your vote only once.
Select any one of the candidates mentioned below carefully.

Candidate | Symbol
-----|-----
Manjunath | Tiger
Hemanth   | Fan
Arvind    | Table
Gopal     | Book
Prakruthi | Peacock
Veena     | Vase

Manjunath

Thank you for participating in the Online-Voting session.
Have a happy and safe day !!

D:\KAUSHAL\PESU RELATED\SEM 6\ON\ON project>_

```

Fig. 8.1.11

Fig. 8.1.12 and 8.1.13 show the people already casted their vote and the votes recorded text files on the servers PC.

```

Voter_names.txt
Ashwin-1517
Kaushal-1547
Sanjana-1514
Mahima-1617

```

Fig. 8.1.12

```

Votes_Rec.txt
Manjunath
Manjunath
Manjunath

```

Fig. 8.1.13

8.2. Wireshark Capture

The following pictures show the screenshots taken by running Wireshark during the voting session. In Fig. 8.2.1 the yellow and green highlighted addresses are the IPv6 addresses.

No.	Time	Source	Destination	Protocol	Length	Info
10	44.125201	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	TCP	98	2345 → 19977 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=
12	44.205203	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	TCP	86	[TCP Window Update] 2345 → 19977 [ACK] Seq=1 Ack=1 Win=1
14	44.205464	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	KNET	225	AppData (45)
22	53.750299	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	TCP	86	2345 → 19977 [ACK] Seq=140 Ack=13 Win=131712 Len=0 TSval=
23	53.750427	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	KNET	191	AppData (103), AppData (99), AppData (97)
25	53.848604	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	KNET	366	AppData (101), AppData (110)
30	58.870299	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	TCP	86	2345 → 19977 [ACK] Seq=525 Ack=18 Win=131648 Len=0 TSval=
31	58.870773	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	TCP	86	2345 → 19977 [FIN, ACK] Seq=525 Ack=18 Win=131648 Len=0
36	62.966272	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	TCP	98	2345 → 25845 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=
38	63.374858	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	TCP	86	[TCP Window Update] 2345 → 25845 [ACK] Seq=1 Ack=1 Win=1
40	65.218986	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	TCP	86	2345 → 19977 [ACK] Seq=526 Ack=19 Win=131648 Len=0 TSval=
50	74.230580	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	KNET	225	AppData (45)
60	82.831472	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	TCP	86	2345 → 25845 [ACK] Seq=140 Ack=12 Win=131712 Len=0 TSval=
61	82.831567	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	KNET	191	AppData (103), AppData (99), AppData (97)
63	82.937010	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	KNET	366	AppData (101), AppData (110)
66	85.903234	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	TCP	86	2345 → 25845 [ACK] Seq=525 Ack=17 Win=131648 Len=0 TSval=
67	85.903708	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	TCP	86	2345 → 25845 [FIN, ACK] Seq=525 Ack=17 Win=131648 Len=0
72	92.661647	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	TCP	86	2345 → 25845 [ACK] Seq=526 Ack=18 Win=131648 Len=0 TSval=
9	44.125127	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:5530:62f7:3eeb:1919	TCP	98	2345 → 14037 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=
15	44.205464	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:5530:62f7:3eeb:1919	TCP	86	[TCP Window Update] 2345 → 14037 [ACK] Seq=1 Ack=1 Win=1
32	58.870933	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:5530:62f7:3eeb:1919	KNET	225	AppData (45)
42	70.953273	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:5530:62f7:3eeb:1919	TCP	86	2345 → 14037 [ACK] Seq=140 Ack=13 Win=131712 Len=0 TSval=
43	70.953439	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:5530:62f7:3eeb:1919	KNET	191	AppData (103), AppData (99), AppData (97)
45	71.030520	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:5530:62f7:3eeb:1919	KNET	366	AppData (101), AppData (110)
48	74.229925	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:5530:62f7:3eeb:1919	TCP	86	2345 → 14037 [ACK] Seq=525 Ack=18 Win=131648 Len=0 TSval=
49	74.230385	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:5530:62f7:3eeb:1919	TCP	86	2345 → 14037 [FIN, ACK] Seq=525 Ack=18 Win=131648 Len=0
59	82.831471	2409:4073:193:8082:50c3:fa94:6ee5:534	2405:204:560c:2b18:5530:62f7:3eeb:1919	TCP	86	2345 → 14037 [ACK] Seq=526 Ack=19 Win=131648 Len=0 TSval=
7	44.124754	2405:204:560c:2b18:5530:62f7:3eeb:1919	2409:4073:193:8082:50c3:fa94:6ee5:534	TCP	94	14037 → 2345 [SYN] Seq=0 Win=42470 Len=0 MSS=1370 SACK_F
8	44.124760	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	2409:4073:193:8082:50c3:fa94:6ee5:534	TCP	94	19977 → 2345 [SYN] Seq=0 Win=42470 Len=0 MSS=1370 SACK_F
11	44.205071	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	2409:4073:193:8082:50c3:fa94:6ee5:534	TCP	86	19977 → 2345 [ACK] Seq=1 Ack=1 Win=45056 Len=0 TSval=183
13	44.205382	2405:204:560c:2b18:5530:62f7:3eeb:1919	2409:4073:193:8082:50c3:fa94:6ee5:534	TCP	86	14037 → 2345 [ACK] Seq=1 Ack=1 Win=45056 Len=0 TSval=183
16	44.277478	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	2409:4073:193:8082:50c3:fa94:6ee5:534	TCP	86	19977 → 2345 [ACK] Seq=1 Ack=140 Win=45056 Len=0 TSval=1
21	53.750186	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	2409:4073:193:8082:50c3:fa94:6ee5:534	TCP	98	19977 → 2345 [PSH, ACK] Seq=1 Ack=140 Win=45056 Len=12 T
24	53.848528	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	2409:4073:193:8082:50c3:fa94:6ee5:534	TCP	86	19977 → 2345 [ACK] Seq=13 Ack=245 Win=45056 Len=0 TSval=
26	53.926440	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	2409:4073:193:8082:50c3:fa94:6ee5:534	TCP	86	19977 → 2345 [ACK] Seq=13 Ack=525 Win=45056 Len=0 TSval=
29	58.870174	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	2409:4073:193:8082:50c3:fa94:6ee5:534	TCP	91	19977 → 2345 [PSH, ACK] Seq=13 Ack=525 Win=45056 Len=5 T
33	58.937917	2405:204:560c:2b18:5530:62f7:3eeb:1919	2409:4073:193:8082:50c3:fa94:6ee5:534	TCP	86	14037 → 2345 [ACK] Seq=1 Ack=140 Win=45056 Len=0 TSval=1
34	58.974479	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	2409:4073:193:8082:50c3:fa94:6ee5:534	TCP	86	19977 → 2345 [ACK] Seq=18 Ack=526 Win=45056 Len=0 TSval=1
35	62.966046	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	2409:4073:193:8082:50c3:fa94:6ee5:534	TCP	94	25845 → 2345 [SYN] Seq=0 Win=42470 Len=0 MSS=1370 SACK_F
37	63.374784	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	2409:4073:193:8082:50c3:fa94:6ee5:534	TCP	86	25845 → 2345 [ACK] Seq=1 Ack=1 Win=45056 Len=0 TSval=183
39	65.218845	2405:204:560c:2b18:1cb2:7a7c:cb5c:e6fe	2409:4073:193:8082:50c3:fa94:6ee5:534	TCP	86	19977 → 2345 [FIN, ACK] Seq=18 Ack=526 Win=45056 Len=0 TSval=

Fig. 8.2.1

Here, both the clients try to establish the TCP connection when they are run simultaneously.

IP address of Client_1 → 2409.204.560c.2b18.5530.62f7.3eeb.1919

IP address of Client_2 → 2409.204.560c.2b18.1cb2.7a7c.cb5c.e6fe

IP address of Server → 2409:4073:193:8082:50c3:fa94:6ee5:534

The server acknowledges both the connections initially. Hence, there is now a competition among the clients and eventually Client_2 wins. The server sends the message to Client_2 via the kNet Protocol. Meanwhile, Client_1 keeps sending the messages but doesn't get any response from the server. Thus, Client_1 is made to wait until Client_2 is done.

The Client_2 then provides its credentials via a TCP protocol in its payload. (29th segment) The server again sends a message via the kNet Protocol.

The Client_2 finishes its voting session and exits. Only then, does the Client_1 get its chance to vote. The similar workflow happens for Client_1. If any other client tries to connect, TCP connection is established but the client put in the waiting queue until Client_1 is done.

1



1

1

1



1

9. Conclusion

We have successfully designed an online-voting system using Socket programming and IPv6 addressing. By using this Python application, the electorate can cast his vote from anywhere in the world by simply using his/her laptop and mobile hotspot. This makes it a much faster and a more efficient (cost and time effective) way of conducting elections. Additionally, all the unexpected events like sudden load-shedding, etc. are handled and no vote is lost track off. A dashboard displaying the polling results and the declaration of the winner is shown on the server's (Admin's) side.

This system can further be improved by conducting the whole system using web protocols such as 'https'. In order to store and manage large amounts of data in a systematic way, a DBMS software can be implemented. System generated unique id and password can be provided to make the login more secure. Further, GUIs can be used to enhance the user experience of voting session as a whole.

10. Code

10.1. Client code – ‘online_voting_client.py’

```
import socket

def Main():
    host = '2409:4073:2016:96a2:a4e0:bf29:8dd6:2313' #give the ip address of server pc
    port = 2345 #assign a port
    s = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)
    s.connect((host, port)) #to Establish TCP connection with the host

    # print the statement sent by server to name themselves
    recv1 = s.recv(1024) #receives first message from server
    print("\n")
    print(str(recv1.decode('ascii'))) #decodes the received message by converting from bytes to ascii format

    # enter user name and password and send it to server
    data1 = input()
    data1 = data1.lower() #convert to lower case to avoid caps error
    s.send(data1.encode('ascii')) #send username and password to the server encode the ascii to bytes

    # print the statement sent by server to whom to vote
    recv2 = s.recv(1024) # receives enter_vote from server
    print("\n")
    print(str(recv2.decode('ascii'))) #receive a message from server

    # check if user is already there
    if str(recv2.decode('ascii')) == "Sorry, you have already voted.": #client shutdowns without any further operations
        s.close()
    elif str(recv2.decode('ascii')) == "The username or password you entered is incorrect\nPlease Enter your Username or Password":
        data1 = input()
        data1 = data1.lower()
        s.send(data1.encode('ascii')) #sends second time the credentials

        recv2 = s.recv(1024)
        print(str(recv2.decode('ascii')))
        if str(recv2.decode('ascii')) == "You have exceeded your login limit. Your Vote thus stands cancelled.": #even though user has entered correct credentials but has exceeded login limit
            s.close()
        elif str(recv2.decode('ascii')) == "Sorry, you have already voted.": #if already voted close
            s.close()
        else:
            #valid credentials are passed, and hasn't voted before, allow to take vote
            # to print list of candidates
            recv3 = s.recv(1024)
            print(str(recv3.decode('ascii')))

            # enter user wished candidate name
            data2 = input()

            s.send(data2.encode('ascii')) #send users vote

            # if user enters non-existent candidate name
            recv4 = s.recv(1024) #response for vote from server
            print("\n")
            print(str(recv4.decode('ascii')))
            if str(recv4.decode('ascii')) == "The candidate of your choice does not exist. Your vote thus stands cancelled.":
                s.close()

            #s.close()
        else:
            #if the user is voting for the first time
            # to print list of candidates
            recv3 = s.recv(1024)
            print(str(recv3.decode('ascii')))

            # enter user wished candidate name
            data2 = input() #Cast your vote

            s.send(data2.encode('ascii'))

            # if user enters non-existent candidate name
            recv4 = s.recv(1024)
            print("\n")
            print(str(recv4.decode('ascii')))
            if str(recv4.decode('ascii')) == "The candidate of your choice does not exist. Your vote thus stands cancelled.":
                s.close()

    s.close()

if __name__ == '__main__':
    Main()
```

10.2. Server Code – ‘online_voting_server.py’

```
import socket
from collections import Counter
import time

candidate_vs_vote={'Manjunath':0,'Hemanth':0,'Arvind':0,'Gopal':0,'Prakruthi':0,'Veena':0} #globally initialise the dictionary

def threaded(c,name_list,database_list):

    enter_name="----- Welcome to the Online Voting Portal -----\n\nPlease Enter your Username and
    c.send(enter_name.encode('ascii'))
    f_v=open("Voter_names.txt","a")
    f_c=open("Candidate_names.txt","r")

    name = c.recv(1024)
    name=str(name.decode('ascii'))

    if name not in database_list:
        wrong="The username or password you entered is incorrect\nPlease Enter your Username and Password once again ir
        c.send(wrong.encode('ascii'))
        name1 = c.recv(1024)
        name1=str(name1.decode('ascii'))
        if name1 not in database_list:
            closeit="You have exceeded your login limit. Your Vote thus stands cancelled."
            c.send(closeit.encode('ascii'))
        else:
            if name1 not in name_list:
                name_list.append(name1)
                f_v.write(name1)
                f_v.write("\n")
                enter_vote=("You are allowed to cast your vote only once.\nSelect any one of the candidates mentioned i
                c.send(enter_vote.encode('ascii'))

                all_candidates=""
                for candidate_name in f_c:
                    all_candidates+=candidate_name
                c.send(all_candidates.encode('ascii'))

                vote=c.recv(1024)
                vote=str(vote.decode('ascii'))
                if vote in candidate_vs_vote:
                    # print("Checkcounter candidate_vs_vote before{} ".format(candidate_vs_vote[vote]))
                    candidate_vs_vote[vote]+=1
                    #print("Checkcounter candidate_vs_vote after{} ".format(candidate_vs_vote[vote]))
                else:
                    no_candidate="The candidate of your choice does not exist. Your vote thus stands cancelled."
                    c.send(no_candidate.encode('ascii'))
            else:
                enter_vote="Sorry, you have already voted."
                c.send(enter_vote.encode('ascii'))

    else:

        if name not in name_list:
            name_list.append(name)
            f_v.write(name)
            f_v.write("\n")
            enter_vote=("You are allowed to cast your vote only once.\nSelect any one of the candidates mentioned below
            c.send(enter_vote.encode('ascii'))

            all_candidates=""
            for candidate_name in f_c:
                all_candidates+=candidate_name
            c.send(all_candidates.encode('ascii'))

            vote=c.recv(1024)
            vote=str(vote.decode('ascii'))

            if vote in candidate_vs_vote:
                # print("Checkcounter candidate_vs_vote before{} ".format(candidate_vs_vote[vote]))
                candidate_vs_vote[vote]+=1
                #print("Checkcounter candidate_vs_vote after{} ".format(candidate_vs_vote[vote]))
            else:
                no_candidate="The candidate of your choice does not exist. Your vote thus stands cancelled."
                c.send(no_candidate.encode('ascii'))
            else:
                enter_vote="Sorry, you have already voted."
                c.send(enter_vote.encode('ascii'))

    f_v.close()
    f_c.close()
    c.close()
```



```

def Main():
    name_list=[]      #list that holds already voted persons names read from a file
    Votes_buff=[]     #list that holds number of votes that was casted in the previous session of server
    symbol_list=['Tiger', 'Fan', 'Table', 'Book', 'Peacock', 'Vase']

    #creating voters database list
    f_d=open("Voters_database.txt","r")
    database_list=[]  #from database you read the names of eligible candidates
    for data in f_d:
        database_list.append(data)
    database_list=[x[:-1] for x in database_list]

    f_vot_rec=open("Votes_Rec.txt","r")    #this is used to keep track of no of votes
    for i in f_vot_rec:
        Votes_buff.append(i.split("\n")[0])    #read votes from file into a buffer
    for key in Votes_buff:
        candidate_vs_vote[key]+=1    #increment the candidates votes depending on the number of occurrences of his name i

    old_Votes_buff=Votes_buff    #save the initial state of votes before starting this session.needed to find number vot
    old_candidate_vs_vote=Counter(old_Votes_buff)
    host = ""
    port = 2345    #select any port this must be same at the client side
    s = socket.socket(socket.AF_INET6, socket.SOCK_STREAM) #####
    s.bind((host, port))
    #print("socket binded to port", port)
    s.listen(3)
    #print("socket is listening")

    #to copy voter names file to num_list from previous session
    f_v=open("Voter_names.txt","r")
    for i in f_v:
        name_list.append(i)
    name_list = [x[:-1] for x in name_list]

    print("\n")
    cnt=int(input("Enter the number of people voting for this round : "))
    while (cnt>=1):
        c, addr = s.accept()
        #print('Connected to :', addr[0], ':', addr[1])
        #function calling
        threaded(c,name_list,database_list)
        cnt-=1
    yorn=input("Should the final results be declared now? (Y/N) : ")
    if yorn=="Y":

        sym_cnt=0
        #to print the results
        print("-----")
        print("                      RESULTS DECLARED")
        print("-----")
        print("-----")
        print("          | Candidate | Symbol | Votes |")
        print("-----")
        for key,val in candidate_vs_vote.items():
            print("          | {:<12} | | {:<7} | | {:^3} | |".format(key,symbol_list[sym_cnt],val))
            sym_cnt+=1
        print("-----")
        print("\n***** Online Voting Portal Closing *****\n*****")
        print("\n The Winner of the election is : {}".format(max(candidate_vs_vote, key=lambda key: candidate_vs_vote[key])))
        print("\n")
        s.close()
    else:
        print("The final results will be declared after further rounds.")
        s.close()

    update_votes=open("Votes_Rec.txt","a")
    for key in candidate_vs_vote:
        counts=candidate_vs_vote[key]-old_candidate_vs_vote[key]
        while (counts>0):
            update_buffer=key
            update_votes.write(update_buffer)
            update_votes.write("\n")
            counts-=1

if __name__ == '__main__':
    Main()

```