PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

100-ft Ring Road, Bengaluru – 560 085, Karnataka, India

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGG

PROGRAM B.TECH

Computer Communication Networks (UE20EC301)

Project Report on

Distributed Data Storage and Computing

Using Socket Programming

Submitted By

– B Gautham PES1UG20EC044

– Sai Pranay Chennamsetti PES1UG20EC048

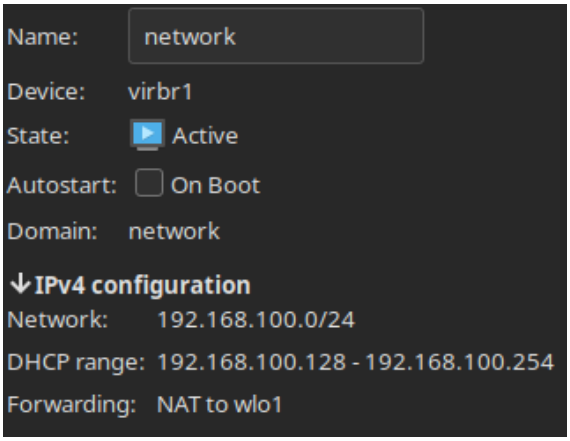
– Dheemanth R Joshi PES1UG20EC059

Problem Statement:

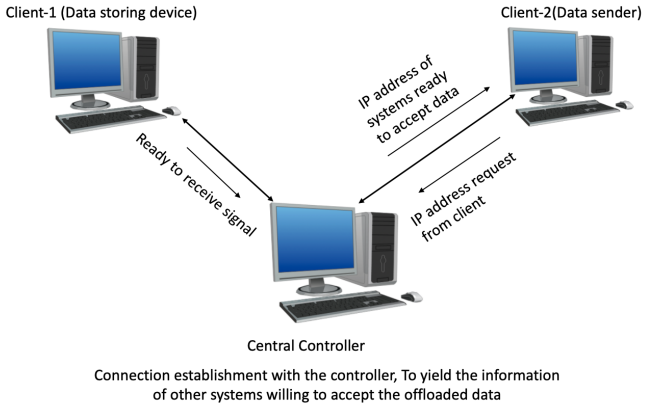
Decentralized control based offloading to store data using socket programming

Introduction:

Distributed storage is a software-defined storage system that enables access to data - when you want, where you want and whom you want to access. In this distributed computing environment, client server applications (consisting of client program and server program) are designed by using socket programming. Socket is a two-way communication link between client and server programs that are running in a network environment. We have used the Julia programming language to program socket ports.



Client Running on Virtual Machine Using NAT (Note that the NAT'ed router is virtual(being emulated by the host machine))



In our system model, we have a central controller overseeing/managing the end systems connected to the same LAN. To identify all the end systems present in the LAN we utilized the NMAP function which floods the subnet with ARP requests to identify the users present in the subnet. To simulate we have used 2 systems with one running server as well as client in a virtual machine. The client program running on the virtual machine is connected to a virtual NAT'ed router which forwards all traffic to the WiFi interface card (wlo1) of the host machine (the host machine is running the server(control) program), due to this the server and client appear to have the same IP address on Wireshark but with different MAC addresses.

137	51.674422881	AzureWav_e9:cf:1b	Broadcast	ARP	42 Who has 192.168.86.164? Tell 192.168.86.17
138	51.674543268	AzureWav_e9:cf:1b	Broadcast	ARP	42 Who has 192.168.86.165? Tell 192.168.86.17
139	51.674638040	AzureWav_e9:cf:1b	Broadcast	ARP	42 Who has 192.168.86.166? Tell 192.168.86.17
140	51.674723868	AzureWav_e9:cf:1b	Broadcast	ARP	42 Who has 192.168.86.167? Tell 192.168.86.17
141	51.674791718	AzureWav_e9:cf:1b	Broadcast	ARP	42 Who has 192.168.86.168? Tell 192.168.86.17
142	51.674872870	AzureWav_e9:cf:1b	Broadcast	ARP	42 Who has 192.168.86.169? Tell 192.168.86.17
143	51.674939790	AzureWav_e9:cf:1b	Broadcast	ARP	42 Who has 192.168.86.170? Tell 192.168.86.17
144	51.675002742	AzureWav_e9:cf:1b	Broadcast	ARP	42 Who has 192.168.86.171? Tell 192.168.86.17
145	51.675071229	AzureWav_e9:cf:1b	Broadcast	ARP	42 Who has 192.168.86.172? Tell 192.168.86.17
146	51.675143099	AzureWav_e9:cf:1b	Broadcast	ARP	42 Who has 192.168.86.173? Tell 192.168.86.17
147	51.675202192	AzureWav_e9:cf:1b	Broadcast	ARP	42 Who has 192.168.86.174? Tell 192.168.86.17

Central Controller (Server) Flooding the Subnet with ARP requests

NMAP: Scans the network that a computer is connected to and outputs a list of ports, device names, operating systems, and several other identifiers that help the user understand the details behind their connection status.

```
julia> include("pool_respond.jl")
Awaiting for controller's signal....
```

Client on Start-up Waiting for Controller's signal

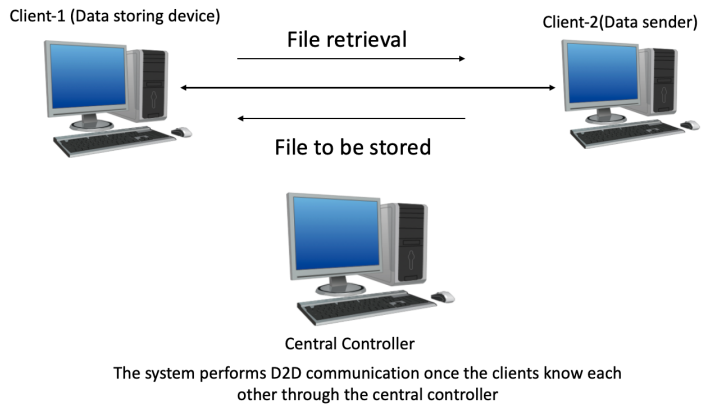
To identify the client systems (Trustworthy) which are running the client code, we scan the network to find all the systems that are listening to a particular predefined port number (in our case 2000). This port number is hard coded into the client program. After identifying all the systems, the server (Central Controller) sends ID numbers which are sequential and the total number of users connected to the server to the systems. The server when communicating the ID and other details connects to port 2000 for communication.

```
┌─[gautham@gauthamHP:~/D/CCN]─[02:03:14 PM]─[I]─[G:main=]
└─>$ julia server.jl
Machine ip is 192.168.86.17
Getting network details : Subnet ID is 192.168.86.0/24 and Default gateway @ 192.168.86.153
Getting hosts on network......
Host @ 192.168.86.41 is up.
Host @ 192.168.86.136 is up.
Host @ 192.168.100.140 is up.
******************
Finding listeners on port 2000....
Found 192.168.100.140 !
Found 192.168.86.136 !
Dict{Any, Any}(2 => ip"192.168.100.140", 1 => ip"192.168.86.136")
Sent Pool info and device id to 2 with ip address 192.168.100.140)
Sent Pool info and device id to 1 with ip address 192.168.86.136)
Waiting for requests.....
```

Server Finding Users listening to port 2000 and assigning ID number

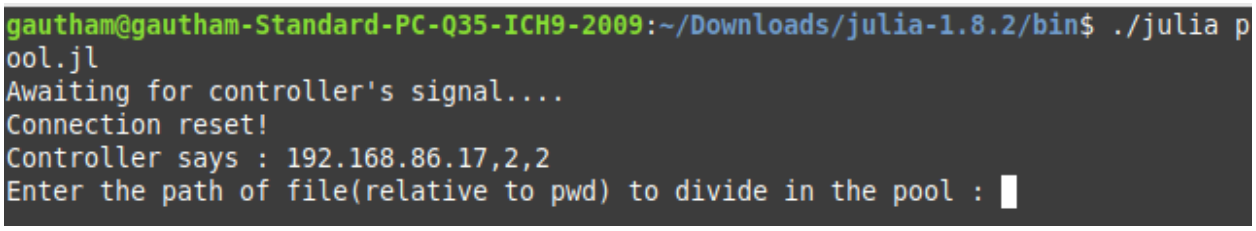
```
Awaiting for controller's signal....
Connection reset!
Controller says : 192.168.86.17,1,2
Waiting for queries.....
```

Client Receiving IP address, ID number and Number of devices connected from the server



When the user wishes to offload a file, their system would divide the file into N subfiles (N being the number of pool devices connected to the server) using the divide function in the client code, then the system sends a request to the server to acquire the IP address of other users connected to the server. After

exchange of information, the local system would initiate simultaneous D2D TCP connection to the other (N-1) systems listening on port 2001 through the use of asynchronous tasks (@async macro in the code). Then the system initiates the file transfer sending N-1 subfiles to the N-1 users by sending the **STORE** command. The local system keeps 1 subfile with itself.



Client Dividing the file and distributing it with the connected devices

Connected device receiving file to store

Now to retrieve the file the client will request for the IP address(s) of devices in the pool. The client then connects to all the devices on the pool listening on port 2001 simultaneously through the use of asynchronous tasks. The client sends the **FWD** command to tell the pool devices to send the stored file back to the client. The client then collects all the segments and combines it into one file.

2nd system responding to FWD command to return the partitioned file

```
gautham@gautham-Standard-PC-Q35-ICH9-2009:~/Downloads/julia-1.8.2/bin$ ./julia p
ool.jl
Awaiting for controller's signal....
Connection reset!
Controller says : 192.168.86.17,2,2
Enter the path of file(relative to pwd) to divide in the pool : 5g_comms.txt
Press any key to retrieve files back from the pool.
Retrieving from 1...
Retrieved file from the pool successfully!
```

Wireshark captures:

832	65.46633…	192.168.86.17	192.168.86.136	TCP	74	46078 → 2000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3104936910 TSecr=0 WS=128
833	65.74586…	192.168.86.136	192.168.86.17	TCP	78	2000 → 46078 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=64 TSval=3919839549 TSecr=3104936910 SACK_PERM=1
834	65.74590…	192.168.86.17	192.168.86.136	TCP	66	46078 → 2000 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3104937190 TSecr=3919839549
835	65.74600…	192.168.86.17	192.168.86.136	TCP	83	46078 → 2000 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=17 TSval=3104937190 TSecr=3919839549
836	65.74609…	192.168.86.17	192.168.86.136	TCP	66	46078 → 2000 [FIN, ACK] Seq=18 Ack=1 Win=64256 Len=0 TSval=3104937190 TSecr=3919839549

Controller send device ID and other details to the clients

6494	667.043590	192.168.86…	192.168.86…	TCP	74	38684 → 2001 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=301735838 TSecr=0 WS=128
6495	667.044087	192.168.86…	192.168.86…	TCP	78	2001 → 38684 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=64 TSval=4255944461 TSecr=301735838
6496	667.247402	192.168.86…	192.168.86…	TCP	66	38684 → 2001 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=301736105 TSecr=4255944461
6497	667.247603	192.168.86…	192.168.86…	TCP	66	[TCP Window Update] 2001 → 38684 [ACK] Seq=1 Ack=1 Win=131712 Len=0 TSval=4255944665 TSecr=301736105
6498	667.302812	192.168.86…	192.168.86…	TCP	90	38684 → 2001 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=24 TSval=301736158 TSecr=4255944665
6499	667.302813	192.168.86…	192.168.86…	TCP	15…	38684 → 2001 [ACK] Seq=25 Ack=1 Win=64256 Len=1448 TSval=301736158 TSecr=4255944665
6500	667.302815	192.168.86…	192.168.86…	TCP	15…	38684 → 2001 [PSH, ACK] Seq=1473 Ack=1 Win=64256 Len=1448 TSval=301736158 TSecr=4255944665
6501	667.302816	192.168.86…	192.168.86…	TCP	15…	38684 → 2001 [ACK] Seq=2921 Ack=1 Win=64256 Len=1448 TSval=301736158 TSecr=4255944665
6502	667.302817	192.168.86…	192.168.86…	TCP	15…	38684 → 2001 [PSH, ACK] Seq=4369 Ack=1 Win=64256 Len=1448 TSval=301736158 TSecr=4255944665
6503	667.302818	192.168.86…	192.168.86…	TCP	15…	38684 → 2001 [ACK] Seq=5817 Ack=1 Win=64256 Len=1448 TSval=301736158 TSecr=4255944665
6504	667.302819	192.168.86…	192.168.86…	TCP	15…	38684 → 2001 [PSH, ACK] Seq=7265 Ack=1 Win=64256 Len=1448 TSval=301736158 TSecr=4255944665
6505	667.303002	192.168.86…	192.168.86…	TCP	66	2001 → 38684 [ACK] Seq=1 Ack=8713 Win=123008 Len=0 TSval=4255944720 TSecr=301736158
6506	667.305656	192.168.86…	192.168.86…	TCP	15…	38684 → 2001 [ACK] Seq=8713 Ack=1 Win=64256 Len=1448 TSval=301736158 TSecr=4255944665
6507	667.305657	192.168.86…	192.168.86…	TCP	15…	38684 → 2001 [PSH, ACK] Seq=10161 Ack=1 Win=64256 Len=1448 TSval=301736158 TSecr=4255944665
6508	667.305657	192.168.86…	192.168.86…	TCP	15…	38684 → 2001 [ACK] Seq=11609 Ack=1 Win=64256 Len=1448 TSval=301736158 TSecr=4255944665
6509	667.305755	192.168.86…	192.168.86…	TCP	66	2001 → 38684 [ACK] Seq=1 Ack=13057 Win=118656 Len=0 TSval=4255944723 TSecr=301736158
6510	667.315383	192.168.86…	192.168.86…	TCP	15…	38684 → 2001 [ACK] Seq=13057 Ack=1 Win=64256 Len=1448 TSval=301736170 TSecr=4255944720
6511	667.315385	192.168.86…	192.168.86…	TCP	15…	38684 → 2001 [PSH, ACK] Seq=14505 Ack=1 Win=64256 Len=1448 TSval=301736170 TSecr=4255944720

Device response to STORE Command

8986	813.475018	192.168.86…	192.168.86…	TCP	74	38686 → 2001 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=301882310 TSecr=0 WS=128
8987	813.475451	192.168.86…	192.168.86…	TCP	78	2001 → 38686 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=64 TSval=1979475902 TSecr=301882310
8988	813.681662	192.168.86…	192.168.86…	TCP	66	38686 → 2001 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=301882573 TSecr=1979475902
8989	813.681663	192.168.86…	192.168.86…	TCP	88	38686 → 2001 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=22 TSval=301882574 TSecr=1979475902
8990	813.681834	192.168.86…	192.168.86…	TCP	66	2001 → 38686 [ACK] Seq=1 Ack=23 Win=131712 Len=0 TSval=1979476108 TSecr=301882574
8991	813.684430	192.168.86…	192.168.86…	TCP	15…	2001 → 38686 [ACK] Seq=1 Ack=23 Win=131712 Len=1448 TSval=1979476111 TSecr=301882574
8992	813.684439	192.168.86…	192.168.86…	TCP	15…	2001 → 38686 [ACK] Seq=1449 Ack=23 Win=131712 Len=1448 TSval=1979476111 TSecr=301882574
8993	813.684444	192.168.86…	192.168.86…	TCP	15…	2001 → 38686 [ACK] Seq=2897 Ack=23 Win=131712 Len=1448 TSval=1979476111 TSecr=301882574
8994	813.684450	192.168.86…	192.168.86…	TCP	15…	2001 → 38686 [ACK] Seq=4345 Ack=23 Win=131712 Len=1448 TSval=1979476111 TSecr=301882574
8995	813.684455	192.168.86…	192.168.86…	TCP	15…	2001 → 38686 [ACK] Seq=5793 Ack=23 Win=131712 Len=1448 TSval=1979476111 TSecr=301882574
8996	813.684459	192.168.86…	192.168.86…	TCP	15…	2001 → 38686 [ACK] Seq=7241 Ack=23 Win=131712 Len=1448 TSval=1979476111 TSecr=301882574
8997	813.684467	192.168.86…	192.168.86…	TCP	15…	2001 → 38686 [ACK] Seq=8689 Ack=23 Win=131712 Len=1448 TSval=1979476111 TSecr=301882574
8998	813.684469	192.168.86…	192.168.86…	TCP	15…	2001 → 38686 [ACK] Seq=10137 Ack=23 Win=131712 Len=1448 TSval=1979476111 TSecr=301882574
8999	813.696244	192.168.86…	192.168.86…	TCP	66	38686 → 2001 [ACK] Seq=23 Ack=2897 Win=63488 Len=0 TSval=301882589 TSecr=1979476111
9000	813.696245	192.168.86…	192.168.86…	TCP	66	38686 → 2001 [ACK] Seq=23 Ack=7241 Win=60544 Len=0 TSval=301882589 TSecr=1979476111
9001	813.696246	192.168.86…	192.168.86…	TCP	66	38686 → 2001 [ACK] Seq=23 Ack=8689 Win=59392 Len=0 TSval=301882590 TSecr=1979476111
9002	813.696360	192.168.86…	192.168.86…	TCP	15…	2001 → 38686 [ACK] Seq=11585 Ack=23 Win=131712 Len=1448 TSval=1979476122 TSecr=301882589
9003	813.696364	192.168.86…	192.168.86…	TCP	15…	2001 → 38686 [ACK] Seq=13033 Ack=23 Win=131712 Len=1448 TSval=1979476122 TSecr=301882589
9004	813.696369	192.168.86…	192.168.86…	TCP	15…	2001 → 38686 [ACK] Seq=14481 Ack=23 Win=131712 Len=1448 TSval=1979476122 TSecr=301882589

Device Response to FWD Command

Project files (codes) can be found @ https://github.com/bgautham4/CCN

Server code:

```
#testing server code
include("GetInfo.jl")
#using Sockets
print("Getting network details : ")
default_route_gen!()
default_gateway,subnet_id = get_default_route()
print("Subnet ID is $(subnet_id) and Default gateway @ $(default_gateway)\n")
print("Getting hosts on network......\n")
hosts_gen!(subnet_id)
arp_table_gen!()
devices = get_hosts_arp(my_ip,default_gateway)
#devices = get_hosts(my_ip,default_gateway)
for dev in devices
        sleep(2)
        print("Host @ $(dev) is up.\n")
end
print("******************\n")
print("Finding listeners on port 2000....\n")
pool = get_open_ports(devices,2000)
for host in values(pool)
        print("Found $(host) !\n")
end
println("$(pool)")
N_pool = length(pool) # find out the number of devices in the pool.
@sync for device_id in keys(pool)
        @async begin
                info_sock = connect(pool[device_id], 2000)
                write(info_sock, "$(my_ip),$(device_id),$(N_pool)")
                println("Sent Pool info and device id to $(device_id) with ip address $(pool[device_id]))")
                close(info_sock)
```

```
        end
end
control = listen(my_ip,2000)
println("Waiting for requests.....")
while true
        conn = accept(control)
        @async begin  # @async allows the simultaneous handling of multiple connections.
                try
                        request = readline(conn)  #Request of the form GET,src_dev,needed_dev\n
                        cmd,src_dev,needed_dev = split(request,",")
                        needed_id = parse(Int,needed_dev)
                        write(conn,"$(pool[needed_id])")
                        close(conn)
                        println("Device $(src_dev) @ $(pool[parse(Int,src_dev)]) asked for $(needed_dev)")
                catch e
                        println("An error has occured! An invalid query was possibly made.")
                        write(conn,"An invalid query was possibly made!")
                        close(conn)
                end
        end
end
```

Client Code:

```
using Sockets
include("Divide_files.jl")
my_ip = getipaddr()
get_id_sock = listen(my_ip,2000)
print("Awaiting for controller's signal....\n")
n_attempts = 0
data = ""
#Acquire details from controller.
```

```
while n_attempts < 3
    try
        conn = accept(get_id_sock)
        global data = read(conn,String)
        println("Controller says : $(data)")
        close(conn)
        break
    catch e
        if isa(e,Base.IOError)
            println("Connection reset!")
            global n_attempts+=1
        else
            println("An error has occured!")
            close(conn)
        end
    end
end
data_arr = split(data,",")
server_ip = IPv4(data_arr[1])
my_id = parse(Int,data_arr[2])
N_pool = parse(Int,data_arr[3])
pool_ids = [i for i in 1:N_pool if i!=my_id]
#Initialize some directories..
for i in 1:N_pool
    run(`mkdir -p pool_storage/$(i)`)
end
print("Enter the path of file(relative to pwd) to divide in the pool : ")
file_name = readline(stdin)
run(`mkdir segments`)
open("$(file_name)") do f
    divide_file(f,N_pool,"segments/")
```

```
end
run(`rm $(file_name)`)
run(`mv segments/$(my_id).txt pool_storage/$(my_id)/`)
@sync for device in pool_ids
    @async begin
        get_ip = connect(server_ip,2000)
        write(get_ip,"GET,$(my_id),$(device)\n") #Request for pool device ip.
        device_ip = IPv4(read(get_ip,String))
        close(get_ip)
        write_sock = connect(device_ip,2001)
        write(write_sock,"STORE,$(my_ip),$(my_id)\n")  #Init command to the storage pool device of the form
STORE,dev_ip,dev_id\n
        open("segments/$(device).txt") do f
            #=while !eof(f)
                write(write_sock,read(f,Char))
            end=#
            write(write_sock,read(f,String))
        end
        close(write_sock)
    end
end

run(`rm -rf segments/`)
print("Press any key to retrieve files back from the pool.")
readline(stdin)
@sync for device in pool_ids
    println("Retrieving from $(device)...")
    @async begin
        get_ip = connect(server_ip,2000)
        write(get_ip,"GET,$(my_id),$(device)\n") #Request for pool device ip.
        device_ip = IPv4(read(get_ip,String))
```

```
        close(get_ip)
        read_sock = connect(device_ip,2001)
        write(read_sock,"FWD,$(my_ip),$(my_id)\n")  #Init command to the storage pool device of the form
STORE/FWD,dev_ip,dev_id\n
        open("pool_storage/$(my_id)/$(device).txt","w") do f
           #=while !eof(f)
              write(write_sock,read(f,Char))
           end=#
           text = read(read_sock,String)
           write(f,text)
        end
        close(read_sock)
    end
end
combine_files(N_pool,"pool_storage/$(my_id)/")
println("Retrieved file from the pool successfully!")
```

File Division Code:

```
function divide_file(file::IO,n_segs::Int,dest::String="")
    size = position(seekend(file))
    seg_size = size ÷ n_segs
    seekstart(file)
    for i in 1:n_segs
        if i<n_segs
            open("$(dest)$(i).txt","w") do f
                while position(file) != i*seg_size
                    write(f,read(file,Char))
                end
            end
        else
            open("$(dest)$(i).txt","w") do f
                while !eof(file)
                    write(f,read(file,Char))
                end
            end
        end
    end
end
function combine_files(n_segs::Int,path::String="",dest::String="")
    open("$(dest)combine.txt","w") do f
        for i in 1:n_segs
            open("$(path)$(i).txt") do f2
                while !eof(f2)
                    write(f,read(f2,Char))
                end
            end
        end
    end
```