DAYANANDA SAGAR COLLEGE OF ENGINEERING

DEPARTMENT ELECTRONICS & COMMUNICATION ENGINEERING

CCN & EMBEDDED LAB MANUAL
VI Semester (21ECL66)
Autonomous Course

Dr. MAHESH KUMAR N
Dr. SHASHI RAJ K
Dr. DEEPA N P
PROF. SOWMYA P
PROF. VIDYASHREE K N
INSTRUCTORS
Mrs. VEENA H S
Mrs. GAYATRI H

Name of the Student	:	
Semester /Section	:	
USN	:	
Batch	:	

DAYANANDA SAGAR COLLEGE OF ENGINEERING
SHAVIGE MALLESHWARA HILLS, KUMARSWAMY LAYOUT, BANGALORE – 78
AN AUTONOMOUS INSTITUTE AFFILIATED TO VTU, APPROVED BY AICTE & UGC
ACCREDITED BY NBA & NAAC WITH 'A' GRADE

DAYANANDA SAGAR COLLEGE OF ENGINEERING

DEPARTMENT ELECTRONICS & COMMUNICATION ENGINEERING

Name of the Laboratory	**:**	CCN & EMBEDDED LAB
Semester/Year	**:**	VI / 2024 (Autonomous)
No. of Students/Batch	**:**	20
No. of Equipment's	**:**	20
Area in square meters	**:**	109 Sq. Mtrs.
Location	**:**	Level – 2
Total Cost of Lab	**:**	Rs. 37, 77,535/-
Lab In charge/s	**:**	Dr Mahesh Kumar N.
Instructors	**:**	Mrs. VEENA H S Mrs. GAYATRI H
HOD	**:**	Dr. T.C. Manjunath, Ph.D. (IIT Bombay)

ABOUT THE COLLEGE & THE DEPARTMENT

The Dayananda Sagar College of Engineering was established in 1979, was founded by Sri R. Dayananda Sagar and is run by the Mahatma Gandhi Vidya Peetha Trust (MGVP). The college offers undergraduate, post-graduates and doctoral programmes under Visvesvaraya Technological University & is currently autonomous institution. MGVP Trust is an educational trust and was promoted by Late. Shri. R. Dayananda Sagar in 1960. The Trust manages 28 educational institutions in the name of “Dayananda Sagar Institutions” (DSI) and multi – Specialty hospitals in the name of Sagar Hospitals - Bangalore, India. Dayananda Sagar College of Engineering is approved by All India Council for Technical Education (AICTE), Govt. of India and affiliated to Visvesvaraya Technological University. It has widest choice of engineering branches having 16 Under Graduate courses & 17 Post Graduate courses. In addition, it has 21 Research Centers in different branches of Engineering catering to research scholars for obtaining Ph.D under VTU. Various courses are accredited by NBA & the college has a NAAC with ISO certification. One of the vibrant & oldest dept is the ECE dept. & is the biggest in the DSI group with 70 staffs & 1200+ students with 10 Ph.D.’s & 30+ staffs pursuing their research in various universities. At present, the department runs a UG course (BE) with an intake of 240 & 2 PG courses (M.Tech.), viz., VLSI Design Embedded Systems & Digital Electronics & Communications with an intake of 18 students each. The department has got an excellent infrastructure of 10 sophisticated labs & dozen class room, R & D centre, etc…

VISION OF THE DEPARTMENT

To achieve continuous improvement in quality technical education for global competence with focus on industry, societal needs, research and professional ethics.

MISSION OF THE DEPARTMENT

Offering quality education in Electronics and Communication Engineering with effective teaching and learning process in multidisciplinary environment.

Training the students to take-up the projects in emerging technologies and work with the team spirit.

To imbibe the professional ethics, development of skills and research culture for better placement opportunities.

PROGRAM EDUCATIONAL OBJECTIVES [PEOs]

After 4 years, the students will be,

PEO-1: Ready to apply the state-of-art technology in industry and meeting the societal needs with knowledge of Electronics and Communication Engineering due to strong academic culture.

PEO-2: Competent in technical and soft skills to be employed with capability of working in multidisciplinary domains.

PEO-3: Professionals, capable of pursuing higher studies in technical, research or management programs.

PROGRAM SPECIFIC OUTCOMES [PSOs]

Students will be able to,

PSO-1: Design, develop and integrate electronics circuits and systems using current practices and standards.

PSO-2: Apply knowledge of hardware and software in designing embedded and communication systems.

INSTRUCTIONS TO THE CANDIDATES

1. Students should come with thorough preparation for the experiment to be conducted.

2. Students will not be permitted to attend the laboratory unless they bring the practical record fully completed in all respects pertaining to the experiment conducted in the previous class.

3. Practical record should be neatly maintained.

4. They should obtain the signature of the staff-in-charge in the observation book after completing each experiment.

5. Theory regarding each experiment should be written in the practical record before procedure in your own words.

6. Ask lab technician for assistance if you have any problem.

7. Save your class work, assignments in system.

8. Do not download or install software without the assistance of the laboratory technician.

9. Do not alter the configuration of the system.

10. Turnoff the systems after use.

DAYANANDA SAGAR COLLEGE OF ENGINEERING
DEPARTMENT OF ELECTRONICS AND COMMUNICATION

SYLLABUS
CCN & EMBEDDED LABORATORY

Course code: 21ECL66	Credits : 2
L:P:T:S:0:2:0:0	CIE Marks : 50
Exam Hours : 3	SEE Marks : 50
Total Hours : 26	Total Marks : 100

COURSE OBJECTIVES:

1. To build the knowledge of networking protocols and interconnections.

2. To impart the programming skill sets to implement the functionalities and responsibilities of Data link and networking layer

COURSE OUTCOMES

CO 1	Implement and simulate point-to-point and wireless networks in NS2
CO 2	Analyze the network performance by varying the network parameters
CO 3	Implement, analyze and evaluate networking protocols in NS2
CO 4	Implement interfacing of various sensors with ESP32 Development board
CO 5	Demonstrate the ability to transmit data wirelessly between different devices using different communication protocols
CO 6	Show an ability to upload/download sensor data on cloud and server

Mapping of Course Outcomes to Program Outcomes:

	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
CO1	**3**	**3**	**3**									
CO2	**3**											
CO3												
CO4												
CO5												
CO6												

SL NO	COURSE CONTENTS	COs
	PART-A	
	SIMULATION EXPERIMENTS USING NS2 / NS3 / NCTUNS	
1	Implement a point to point network with four nodes and duplex links between them. Analyze the network performance by setting the queue size and varying the bandwidth.	CO1,CO2,CO3
2	Implement a four node point to point network with links n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP between n1-n3. Apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets sent by TCP/UDP.	CO1,CO2,CO3
3	Implement Ethernet LAN using n (6-10) nodes. Compare the throughput by changing the error rate and data rate	CO1,CO2,CO3
4	Implement Ethernet LAN using n nodes and assign multiple traffic to the nodes and obtain congestion window for different sources/ destinations.	CO1,CO2,CO3
5	Implement ESS with transmission nodes in Wireless LAN and obtain the performance parameters.	CO1,CO2,CO3
6	Implementation of any routing algorithm	CO1,CO2,CO3
	PART-B	
1	To measure the Temperature, Humidity and Heat index using ESP32 Development Board and display the values in serial monitor	CO4,CO5,CO6
2	a)Wi-Fi networks and get Wi-Fi strength using ESP32 Development broad b) Configure/Set Your ESP32 Development broad as an access point c) Establish connection between IoT Node and Access point and Display of local IP and Gateway IP	CO4,CO5,CO6
3	a) Design and Implementation of HTTP based IoT Web Server to control the status of LED b) Design and Implementation of HTTP based IoT Web Server to display sensor value	CO4,CO5,CO6
4	Design and Implementation of MQTT based Controlling and Monitoring Using Ubidots MQTT Server	CO4,CO5,CO6
5	Establish a communication between client and IoT Web Server to POST and GET sensor values over HTTP	CO4,CO5,CO6

EXTRA PROGRAMS:

1. Implementation of spanning tree algorithm
2. Implementation of Stop and wait protocol
3. Implement a program to communicate between mobile nodes and suspicious nodes using NS2.
4. Implement UDP wireless communication using NS2
5. Implement TCP communication using NS2

HARDWARE AND SOFTWARE REQUIREMENTS:

Hardware Requirements:
Processor : Pentium 3 or higher
RAM : 512MB or more
Hard Disk : 16GB or more (there should be enough space to hold both Linux and Windows)

SOFTWARE REQUIREMENTS:
Operating System : Windows, Linux
Simulation Software: NS2/ NCTUns/Energia 1.8.7E21 or higher(latest)

TEXT BOOKS:
1. **Introduction to Network Simulator NS2,** Issariyakul, Teerawat, Hossain, Ekram, , Springer US, 2012.

ASSESSMENT PATTERN:

CIE –Continuous Internal Evaluation Lab (50 Marks)
SEE –Semester End Examination Lab (50 Marks)

Bloom's Category	**Performance (Day To Day)**	**Internal Test**
Marks (Out of 50)	**25**	**25**
Remember		
Understand		
Apply	05	05
Analyze	10	10
Evaluate	05	05
Create	05	05

Bloom's Category	**Marks Theory(50)**
Remember	
Understand	
Apply	15
Analyze	15
Evaluate	10
Create	10

PART -A

INTRODUCTION TO NS-2

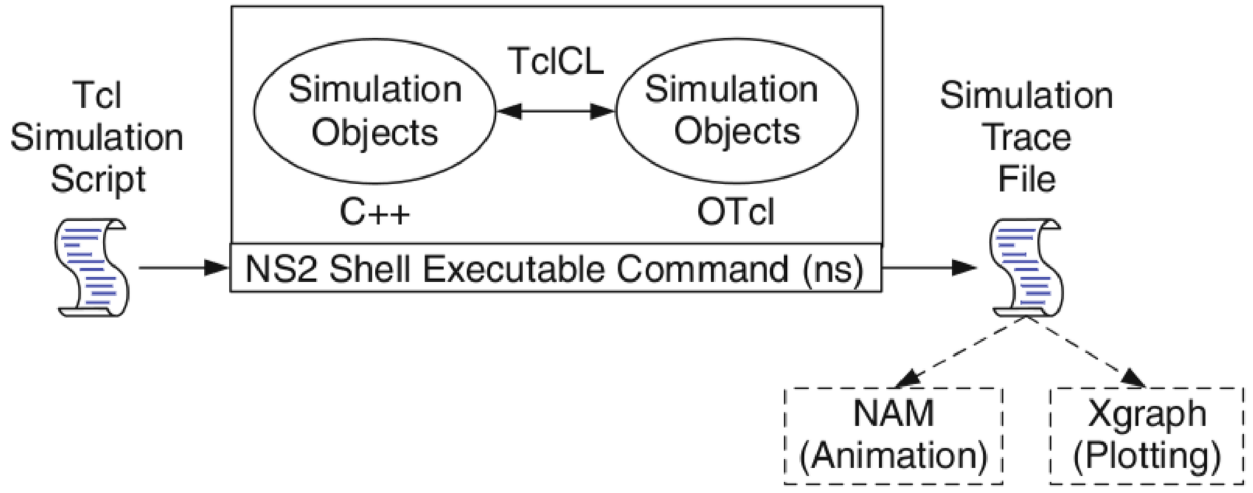
Widely known as NS2, is simply an event driven simulation tool.

Useful in studying the dynamic nature of communication networks.

Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2.

In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors.

BASIC ARCHITECTURE OF NS2:



(TOOL COMMAND LANGUAGE) TCL SCRIPTING

TCL is a general purpose scripting language. [Interpreter]

TCL runs on most of the platforms such as Unix, Windows, and Mac.

The strength of TCL is its simplicity.

It is not necessary to declare a data type for variable prior to the usage.

WIRED TCL SCRIPT COMPONENTS

Create the event scheduler

Open new files & turn on the
tracing Create the nodes

Setup the links

Configure the traffic type (e.g., TCP, UDP, etc)

Set the time of traffic generation (e.g., CBR,
FTP) Terminate the simulation

NS SIMULATOR PRELIMINARIES:

1. Initialization and termination aspects of the ns simulator.
2. Definition of network nodes, links, queues and topology.
3. Definition of agents and of applications.
4. The nam visualization tool.
5. Tracing and random variables.

INITIALIZATION AND TERMINATION OF TCL SCRIPT IN NS-

2 An ns simulation starts with the command,

set ns [new Simulator]

This is thus the first line in the TCL script. This line declares a new variable using the set command, you can call this variable as you wish, In general, it is declared as ns because, it is an instance of the Simulator class, so an object the code [new Simulator] is indeed the installation of the class Simulator using the reserved word new.
In order to have output files with data on the simulation (trace files) or files used for visualization (nam files), we need to create the files using —open command:

#Open the Trace file

set tracefile1 [open out.tr w]
$ns trace-all $tracefile1

#Open the NAM trace file

set namfile [open out.nam w]
$ns namtrace-all $namfile

The above creates a trace file called —out.tr and a nam visualization trace file called —out.nam. Within the TCL script, these files are not called explicitly by their names, but instead by pointers that are declared above and called —tracefile1 and —namfile respectively. Remark that they begin with a # symbol. The second line open the file —out.tr to be used for writing, declared with the letter —w. The third line uses a simulator method called trace-all that have as parameter the name of the file where the traces will go.

The last line tells the simulator to record all simulation traces in NAM input format. It also gives the file name that the trace will be written to later by the command $ns flush-trace. In our case, this will be the file pointed at by the pointer —$namfile ,i.e the file —out.tr. The termination of the program is done using a —finish procedure.

#Define a „finish" procedure

```
Proc finish { } {
global ns tracefile1 namfile
$ns flush-trace
Close $tracefile1
Close $namfile
Exec nam out.nam &
Exit 0
```

The word proc declares a procedure in this case called **finish** and without arguments. The word **global** is used to tell that we are using variables declared outside the procedure. The simulator method ―**flush-trace"** will dump the traces on the respective files. The TCL command ―**close"** closes the trace files defined before and **exec** executes the nam program for visualization. The command **exit** will ends the application and return the number 0 as status to the system. Zero is the default for a clean exit. Other values can be used to say that exit because something fails.
At the end of ns program we should call the procedure ―finish‖ and specify at what time the termination should occur. For example,

```
$ns at 125.0 "finish"
```

will be used to call ―**finish** at time 125sec.Indeed,the **at** method of the simulator allows us to schedule events explicitly.

The simulation can then begin using the command,

```
$ns run
```

Definition of a network of links and nodes
The way to define a node is,

```
set n0 [$ns node]
```

The node is created which is printed by the variable n0. When we shall refer to that node in the script we shall thus write $n0.

Once we define several nodes, we can define the links that connect them. An example of a definition of a link is:

```
$ns duplex-link $n0 $n2 10Mb 10ms DropTail
```

Which means that $n0 and $n2 are connected using a bi-directional link that has 10ms of propagation delay and a capacity of 10Mb per sec for each direction.

To define a directional link instead of a bi-directional one, we should replace ―duplex-link by ―simplex-link.

In NS, an output queue of a node is implemented as a part of each link whose input is that node. The definition of the link then includes the way to handle overflow at that queue. In our case, if the buffer capacity of the output queue is exceeded then the last packet to arrive is dropped. Many alternative options exist, such as the RED (Random Early Detectiom) mechanism, the FQ (Fair Queuing), the DRR (Deficit Round Robin), the Stochastic Fair Queuing (SFQ) and the CBQ (which including a priority and a round-robin scheduler).

In ns, an output queue of a node is implemented as a part of each link whose input is that node. We should also define the buffer capacity of the queue related to each link. An example would be:

```
#set Queue Size of link (n0-n2) to 20
$ns queue-limit $n0 $n2 20
```

Agents and Applications

We need to define routing (sources, destinations) the agents (protocols) the application that use them.

FTP over TCP

TCP is a dynamic reliable congestion control protocol. It uses Acknowledgements created by the destination to know whether packets are well received.

There are number variants of the TCP protocol, such as Tahoe, Reno, NewReno, Vegas. The type of agent appears in the first line:

```
set tcp [new Agent/TCP]
```

The command **$ns attach-agent $n0 $tcp** defines the source node of the tcp connection.
The command

```
set sink [new Agent /TCPSink]
```

Defines the behavior of the destination node of TCP and assigns to it a pointer called sink. ***#Setup a UDP connection***

```
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_2
```

#Setup a CBR over UDP connection

```
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packetsize_ 100
$cbr set rate_ 0.01Mb
$cbr set random_ false
```

Above shows the definition of a CBR application using a UDP agent.

The command **$ns attach-agent $n4 $sink** defines the destination node. The command **$ns connect $tcp $sink** finally makes the TCP connection between the source and destination nodes.

TCP has many parameters with initial fixed defaults values that can be changed if mentioned explicitly. For example, the default TCP packet size has a size of 1000bytes.This can be changed to another value, say 552bytes, using the command **$tcp set packetSize_ 552**.

When we have several flows, we may wish to distinguish them so that we can identify them with different colors in the visualization part. This is done by the command **$tcp set fid_ 1** that assigns to the
TCP connection a flow identification of ―1‖.We shall later give the flow identification of ―2‖ to the UDP connection.

CBR over UDP

A UDP source and destination is defined in a similar way as in the case of TCP.
Instead of defining the rate in the command $cbr set rate_ 0.01Mb, one can define the time interval between transmission of packets using the command.
$cbr set interval_ 0.005

The packet size can be set to some value using
$cbr set packetSize_ <packet size>

SCHEDULING EVENTS

NS is a discrete event based simulation. The tcp script defines when event should occur. The initializing command set ns [new Simulator] creates an event scheduler, and events are then scheduled using the format:
$ns at <time> <event>

The scheduler is started when running ns that is through the command $ns run.
The beginning and end of the FTP and CBR application can be done through the following command

```
$ns at 0.1 “$cbr start”
$ns at 1.0 “ $ftp start”
$ns at 124.0 “$ftp stop”
$ns at 124.5 “$cbr stop”
```

STRUCTURE OF TRACE FILES:

When tracing into an output ASCII file, the trace is organized in 12 fields as follows in fig shown below, The meaning of the fields are:

1	2	3	4	5	6	7	8	9	10	11	12
Event	**Time**	**From node**	**To node**	**PKT type**	**PKT Size**	**Flags**	**FID**	**Src Addr**	**Dest Addr**	**Seq No.**	**PKT ID**

1. EVENT OR TYPE IDENTIFIER

+ :a packet enque event
- :a packet deque event
r :a packet reception event
d :a packet drop (e.g., sent to dropHead_)
event c :a packet collision at the MAC level

2.TIME: Time at which the packet tracing string is created.

3-4. SOURCE AND DESTINATION NODE: Gives the input node and output node of the link at which the event occurs.

5. PACKET NAME: Gives the packet type (eg: Constant Bit Rate (CBR) or (Transmission Control Protocol) TCP).

6. PACKET SIZE: Size of packet in bytes.

7. FLAGS: digit flag string.
"-": disable
1st = "E": ECN (Explicit Congestion Notification) echo is enabled.
2nd = "P": the priority in the IP header is enabled.
3rd : Not in use
4th = "A": Congestion action
5th = "E": Congestion has occurred.
6th = "F": The TCP fast start is used.
7th = "N": Explicit Congestion Notification (ECN) is on.

8. FLOW ID: This is the flow id (fid) of IPv6 that a user can set for each flow at the input OTcl script one can further use this field for analysis purposes; it is also used when specifying stream color for the NAM display.

9-10. SOURCE AND DESTINATION ADDRESS: The format of these two fields is "a.b", where "a" is the address and "b" is the port.

11. SEQUENCE NUMBER: This is the network layer protocol's packet sequence number. Even though UDP implementations in a real network do not use sequence number, ns keeps track of UDP packet sequence number for analysis purposes

12. PACKET UNIQUE ID: The last field shows the unique ID of the packet.

XGRAPH

The xgraph program draws a graph on an x-display given data read from either data file or from standard input if no files are specified. It can display upto 64 independent data sets using different colors and line styles for each set. It annotates the graph with a title, axis labels, grid lines or tick marks, grid labels and a legend.

SYNTAX:
Xgraph [options] file-name

Options are listed here
/-bd <color> (Border)
This specifies the border color of the xgraph window.
/-bg <color> (Background)
This specifies the background color of the xgraph window.

/-fg<color> (Foreground)
This specifies the foreground color of the xgraph window.
/-lf <fontname> (LabelFont)
All axis labels and grid labels are drawn using this font.
/-t<string> (Title Text)
This string is centered at the top of the graph.
/-x <unit name> (XunitText)
This is the unit name for the x-axis. Its default is ―X‖.
/-y <unit name> (YunitText)
This is the unit name for the y-axis. Its default is ―Y‖.

AHO, WEINBERGER AND KERNIGHAN (AWK) SCRIPT:

AWK is a programmable, pattern-matching, and processing tool available in UNIX. It works equally well with text and numbers. AWK is not just a command, but a programming language too. In other words, AWK utility is a pattern scanning and processing language. It searches one or more files to see if they contain lines that match specified patterns and then perform associated actions, such as writing the line to the standard output or incrementing a counter each time it finds a match.

SYNTAX:

awk option 'selection_criteria {action}' file(s)

Here, selection_criteria filters input and select lines for the action component to act upon. The selection_criteria is enclosed within single quotes and the action within the curly braces. Both the selection_criteria and action forms an AWK program.

Example: $ awk „/manager/ {print}‟ emp.lst
Variables
Awk allows the user to use variables of there choice. You can now print a serial number, using the variable kount, and apply it those directors drawing a salary exceeding 6700:
$ awk –F"|" „$3 == "director" && $6 > 6700 {
kount =kount+1
printf " %3f %20s %-12s %d\n", kount,$2,$3,$6 }‟ empn.lst

THE –f OPTION: STORING awk PROGRAMS IN A FILE
You should holds large awk programs in separate file and provide them with the awk extension for easier identification. Let's first store the previous program in the file empawk.awk:
$ cat empawk.awk

Observe that this time we haven't used quotes to enclose the awk program. You can now use awk with the –f *filename* option to obtain the same output:
Awk –F"|" –f empawk.awk empn.lst

THE BEGIN AND END SECTIONS:

AWK statements are usually applied to all lines selected by the address, and if there are no addresses, then they are applied to every line of input. But, if you have to print something before processing the first line, for example, a heading, then the BEGIN section can be used gainfully. Similarly, the end section useful in printing some totals after processing is over. The BEGIN and END sections are optional and take the form

BEGIN {action}

END {action}

These two sections, when present, are delimited by the body of the awk program. You can use them to print a suitable heading at the beginning and the average salary at the end.

BUILT-IN VARIABLES:

Awk has several built-in variables. They are all assigned automatically, though it is also possible for a user to reassign some of them. You have already used NR, which signifies the record number of the current line. We'll now have a brief look at some of the other variable.

***The FS Variable**:* as stated elsewhere, awk uses a contiguous string of spaces as the default field delimiter. FS redefines this field separator, which in the sample database happens to be the |. When used at all, it must occur in the BEGIN section so that the body of the program knows its value before it starts processing:

BEGIN {FS=”|”}

This is an alternative to the –F option which does the same thing.

***The OFS Variable**:* when you used the print statement with comma-separated arguments, each argument was separated from the other by a space. This is awk's default output field separator, and can reassigned using the variable OFS in the BEGIN section:

BEGIN { OFS=”~” }

When you reassign this variable with a ~ (tilde), awk will use this character for delimiting the print arguments. This is a useful variable for creating lines with delimited fields.

***The NF variable**:* NF comes in quite handy for cleaning up a database of lines that don't contain the right number of fields. By using it on a file, say emp.lst, you can locate those lines not having 6 fields, and which have crept in due to faulty data entry:

$awk „BEGIN {FS = “|”}

NF! =6 { Print “Record No “, NR, “has”, “fields”}‟ empx.lst

STEPS FOR EXECUTION

Create a folder in **roots** folder. Open the terminal and type

```
[root@localhost ~]#cd foldername
```

To open editor and type program.

```
[root@localhost ~]# gedit programname.tcl
```

*Program name should have the extension " **.tcl** "
*Type the program in the editor window and then save it.

To open editor and type **awk** program.

```
[root@localhost ~]# gedit programname.awk
```

*Program name should have the extension "**.awk** "
* Type the program in the editor window and then save

it. To run the simulation program,

```
[root@localhost~]# ns programname.tcl
```

Here **"ns"** indicates network simulator. We get the topology shown in the snapshot. Now press the play button in the simulation window and the simulation will begins.

To see the trace file contents open the file as ,

```
[root@localhost~]# gedit programname.tr
```

After simulation is completed run **awk file** to see the output,

```
[root@localhost~]# awk –f programname.awk programname.tr
```

PROGRAM 1:

TO IMPLEMENT A POINT-TO-POINT NETWORK WITH FOUR NODES AND DUPLEX LINKS BETWEEN THEM. ANALYZE THE NETWORK PERFORMANCE BY SETTING QUEUE SIZE AND VARYING THE BANDWIDTH.

ALGORITHM:

STEP 1: Create the simulator object ns for designing the given simulation.
STEP 2: Open the trace file and nam file in the write mode.
STEP 3: Create the 4 nodes of the simulation using the ‘set’ command and duplex link between them.
STEP 4: Create a queue size between nodes.
STEP 5: Create UDP agent for the nodes and attach these agents to the nodes.
STEP 6: The traffic generator used is UDP and measured in terms of cbr0 and cbr1.
STEP 7: Configure node2 and node 3 as the null and attach it.
STEP 8: Connect source and destination nodes using ‘connect’ command.
STEP 9: Create the Packet size and time interval between each packet coming using the cbr object instance created.
STEP 10: Schedule the start and stop of events for UDP agent with 0.00msec and 10msec for cbr0 agent.
STEP 11: Schedule the start and stop of events for UDP agent with 2.00msec and 12 msec for cbr1 agent.
STEP 12: Schedule the simulation for 13 msec.

PROGRAM: COMMENTS

```
set ns [new Simulator]                ;#creating a new variable ns


set tf [open p1.tr w]                 ;#open trace file p1.tr in writable mode
$ns trace-all $tf                     ;#main class object ns linked with trace file object tf


set nf [open p1.nam w]                ;#open nam file p1.nam in writable mode
$ns namtrace-all $nf                  ;#main class object ns linked with nam file object nf


set n0 [$ns node]                     ;#creation of nodes
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
```

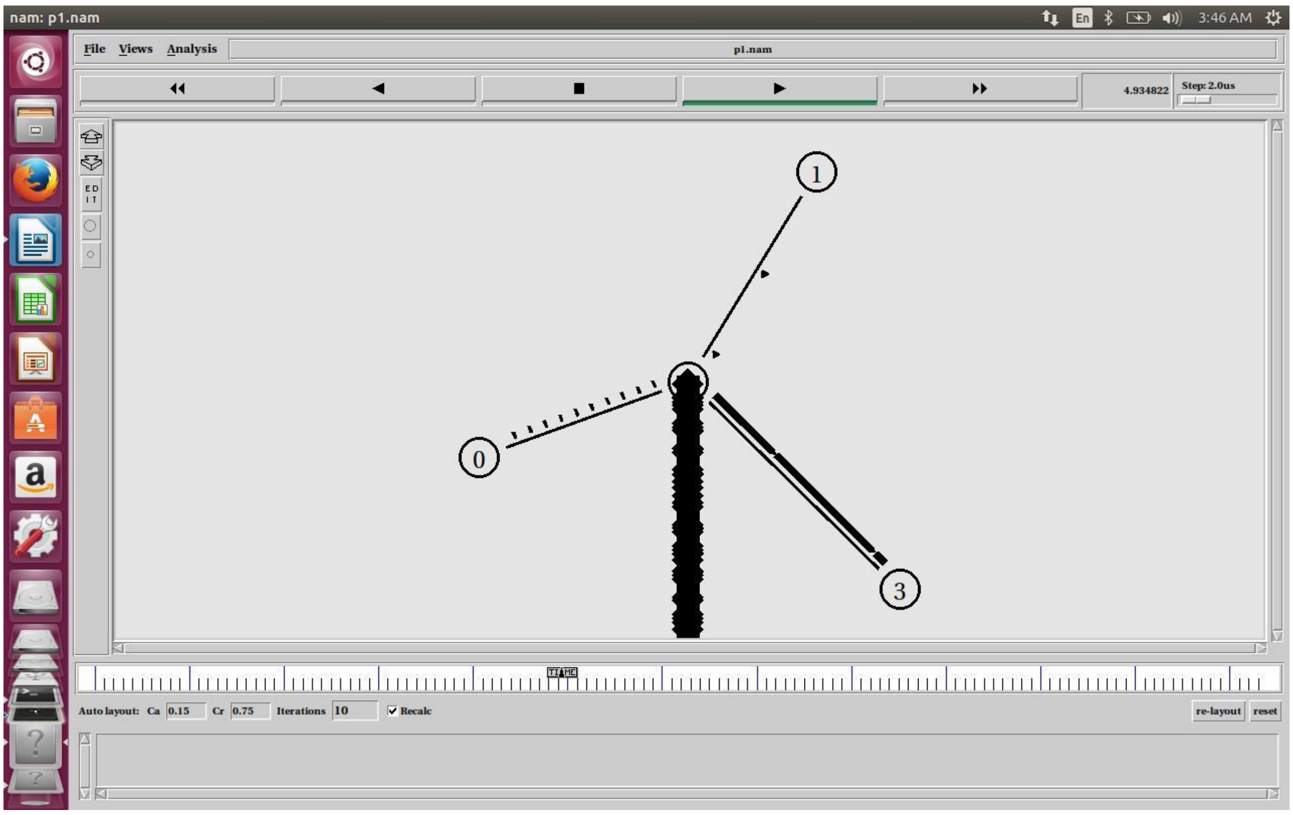
```
$ns duplex-link $n0 $n2 20Mb 10ms DropTail          ;#forming duplex connection between the
$ns duplex-link $n1 $n2 10Mb 10ms DropTail         nodes ;#and vary the bandwidth in this link

$ns duplex-link $n2 $n3 0.7Mb 10ms DropTail


$ns set queue-limit $n0 $n2 10                  ;#Assigning Queuesize between the nodes

$ns set queue-limit $n1 $n2 10

$ns set queue-limit $n2 $n3 5


set udp0 [new Agent/UDP]                        ;#creating object(agent) for the class  Agent/UDP

set udp1 [new Agent/UDP]                        ;#creating object(agent) for the class  Agent/UDP

set null [new Agent/Null]               ;#creating Destination object(agent) for the class  Agent/Null

set cbr0 [new Application/Traffic/CBR]        ;#creating object(agent) for the class
                                                ;#Application/Traffic/CBR

set cbr1 [new Application/Traffic/CBR]


$ns attach-agent $n0 $udp0                      ;#Attaching agents

$ns attach-agent $n1 $udp1

$ns attach-agent $n2 $null

$ns attach-agent $n3 $null

$cbr0 attach-agent $udp0

$cbr1 attach-agent $udp1


$ns connect $udp0 $null                       ;#connecting source and destination node agents

$ns connect $udp1 $null


$cbr0 set packetSize_ 512                      ;#assigning values of packet size and time interval

$cbr0 set interval_ 0.001


$cbr1 set packetSize_ 512

$cbr1 set interval_ 0.005
```

```
proc finish { } {                                    ;#terminate Simulation
global ns nf tf
$ns flush-trace
close $tf
close $nf
exec nam p1.nam &
exit 0
}

$ns at 0.0 "$cbr0 start"                           ;#Setting time for start and stop the packet transmission.
$ns at 10.0 "$cbr0 stop"
$ns at 2.0 "$cbr1 start"
$ns at 12.0 "$cbr1 stop"

$ns at 13.0 "finish"                               ;#Setting time to stop the simulation

$ns run
```

(AWK) SCRIPT:

```
BEGIN{
        count=0
}

{
        if($1=="d")
                count++;
}

END {
        printf("Number of packets dropped: %d\n",count);
}
```

TOPOLOGY AND NAM RESULT:



OUTPUT:

Case 1:

Source Node	Destination Node	Bandwidth	Delay	Queue Limit	No. of packets Dropped
n0	n2	20Mb	10ms	10	9902
n1	n2	10Mb	10ms	10	
n2	n3	0.7Mb	10ms	5	

Case 2:

Source Node	Destination Node	Bandwidth	Delay	Queue Limit	No. of packets Dropped
n0	n2	200Mb	10ms	100	0
n1	n2	100Mb	10ms	100	
n2	n3	70Mb	10ms	50	

RESULT: Network Performance analyzed by setting queue size and varying bandwidth using NS-2.

PROGRAM 2:

TO IMPLEMENT FOUR NODE POINT TO POINT NETWORK WITH LINKS N0-N1,N1-N2 AND N2-N3. APPLY TCP AGENT BETWEEN N0-N3 AND UDP BETWEEN N1-N3. APPLY RELEVANT APPLICATION OVER TCP AND UDP AGENTS CHANGING THE PARAMETER AND DETERMINE THE NUMBER OF PACKETS SENT BY TCP/UDP.

ALGORITHM:

STEP 1: Create the simulator object ns for designing the given simulation.
STEP 2: Open the trace file and nam file in the write mode.
STEP 3: Create the 4 nodes of the simulation using the ‘set’ command and duplex link between them.
STEP 4: Create a queue size between nodes.
STEP 5: Create UDP agent between the nodes n0 & n3 and attach these agents to the nodes.
STEP 6: Create TCP agent between the nodes n1 & n3 and attach these agents to the nodes.
STEP 7: The traffic generator used between n0 & n3 is UDP and measured in terms of cbr.
STEP 8: The traffic generator used between n0 & n3 is FTP and measured in terms of ftp.
STEP 9: Configure node 3 as the null and sink in UDP and TCP protocol respectively and attach it.
STEP 10: Connect source and destination nodes using ‘connect’ command.
STEP 11: Create the Packet size and time interval between each packet coming using the cbr & ftp object instance created.
STEP 12: Schedule the start and stop of events for FTP agent with 0.00msec and 10msec for ftp agent.
STEP 13: Schedule the start and stop of events for UDP agent with 2.00msec and 12 msec for cbr agent.
STEP 14: Schedule the simulation for 13 msec.

PROGRAM:

```
set ns [new Simulator]

set tf [open p2.tr w]
$ns trace-all $tf

set nf [open p2.nam w]
$ns namtrace-all $nf


set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$ns duplex-link $n0 $n2 20Mb 10ms DropTail
$ns duplex-link $n1 $n2 10Mb 10ms DropTail
$ns duplex-link $n2 $n3 0.7Mb 10ms DropTail

$ns set queue-limit $n0 $n2 10
```

```
$ns set queue-limit $n1 $n2 10
$ns set queue-limit $n2 $n3 5

set tcp [new Agent/TCP]
set udp [new Agent/UDP]

set tcpsink [new Agent/TCPSink]
set null [new Agent/Null]

set cbr [new Application/Traffic/CBR]
set ftp [new Application/FTP]


$ns attach-agent $n0 $tcp
$ns attach-agent $n1 $udp
$ns attach-agent $n3 $tcpsink
$ns attach-agent $n3 $null

$ftp attach-agent $tcp
$cbr attach-agent $udp


$ns connect $udp $null
$ns connect $tcp $tcpsink

$cbr set packetSize_ 512
$cbr set interval_ 0.005

$ftp set packetSize_ 512
$ftp set interval_ 0.005


proc finish {} {
global ns nf tf
$ns flush-trace
close $tf
close $nf
exec nam p2.nam &
exit 0
}


$ns at 0.0 "$ftp start"
$ns at 10.0 "$ftp stop"
$ns at 2.0 "$cbr start"
$ns at 12.0 "$cbr stop"

$ns at 13.0 "finish"
$ns run
```

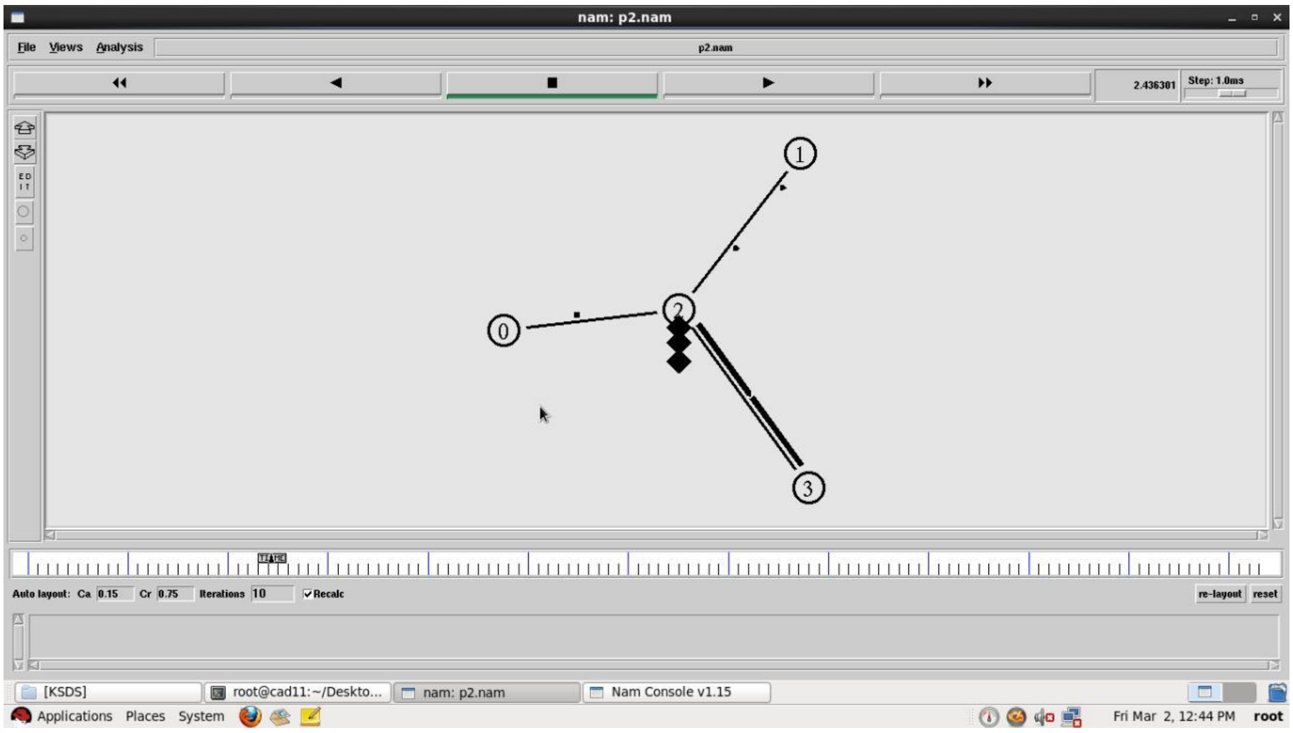
AWK SCRIPT:

```
BEGIN{
tcp=0;
cbr=0;
}
{
if($1=="-" && $5=="tcp")
tcp++
if($1=="-" && $5=="cbr")
cbr++
}
END{
printf("\n no.of tcp packets sent= %d \n",tcp);
printf("\n no.of cbr packets sent= %d \n",cbr);
}
```

TO RUN:

```
ns filename.tcl
awk -f awkfilename.awk tracefilename.tr
```

TOPOLOGY AND NAM RESULT:



Auto layout: Ca 0.15 Cr 0.75 Iterations 10 ✓Recalc

PROGRAM 3:

SIMULATE AN ETHERNET LAN USING N NODES (6-10), CHANGE THE ERROR RATE AND DATA RATE AND COMPARE THE THROUGHPUT.

ALGORITHM:

STEP 1: Create the simulator object ns for designing the given simulation.
STEP 2: Open the trace file and nam file in the write mode.
STEP 3: Create the nodes of the simulation using the 'set' command.
STEP 4: Create Ethernet LAN using make-LAN command and connect the nodes to the LAN.
STEP 5: Create TCP agent for the nodes and attach these agents to the nodes.
STEP 6: The traffic generator used is FTP for both node1 and node5.
STEP 7: Add error node between the nodes 3 and 6.
STEP 8: Configure node5 as the sink and attach it.
STEP 9: Connect node1 and node5 agents using 'connect' command.
STEP 10: Setting colour for the TCP packets.
STEP 11: Schedule the events for FTP agent 0.1msec.
STEP 12: Schedule the simulation for 5 msec.

```
set ns [new Simulator]

set tf [open p3.tr w]
$ns trace-all $tf

set nf [open p3.nam w]
$ns namtrace-all $nf

$ns color 1 "blue"

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]

$n1 label "Source/UDP"
$n3 label "Error Node"
$n5 label "Destination"



;#The below code is used to create a two Lans (Lan1 and #Lan2).


$ns make-lan "$n0 $n1 $n2 $n3" 100Mb 10ms LL Queue/DropTail Mac/802_3
```

```
$ns make-lan "$n4 $n5 $n6 " 100Mb 10ms LL Queue/DropTail Mac/802_3

$ns duplex-link $n3 $n6 100Mb 10ms DropTail

set udp1 [new Agent/UDP]
set cbr1 [ new Application/Traffic/CBR]
set null5 [new Agent/Null]

$ns attach-agent $n1 $udp1
$cbr1 attach-agent $udp1
$ns attach-agent $n5 $null5

$ns connect $udp1 $null5

$cbr1 set packetSize_ 1000
$cbr1 set interval_ 0.0001 ;# This is the data rate.Change ;#
                              this to vary the throughput.

set err [new ErrorModel] ;# The code is used to add an error model
$ns lossmodel $err $n3 $n6 ;#between the nodes n3 and n6. $err set
rate_ 0.1 ;# This is the error rate. Change this
                             ;#rate to add errors between n3 and n6.

$udp1 set class_ 1

proc finish { } {
global nf ns tf
close $nf
close $tf
exec nam p3.nam &
exit 0
}

$ns at 0.1 "$cbr1 start"
$ns at 5.0 "finish"
$ns run
```

AWK SCRIPT:

```
BEGIN {
  Rpacketsize=0;
  Rtimeinterval=0;
}
{
if ($1=="r" && $3=="3" && $4=="6")
Rpacketsize = Rpacketsize+$6;
Rtimeinterval=$2;
}
END {
printf ("throughput:%f Mbps\n",(Rpacketsize/Rtimeinterval)*(8/1000000));
}
```

OUTPUT:

The throughput can be analysed by changing the data rate and error rate as shown below.

Case 1: Fix the data rate to 0.0001 and vary the error rate then throughput decreases as shown below.

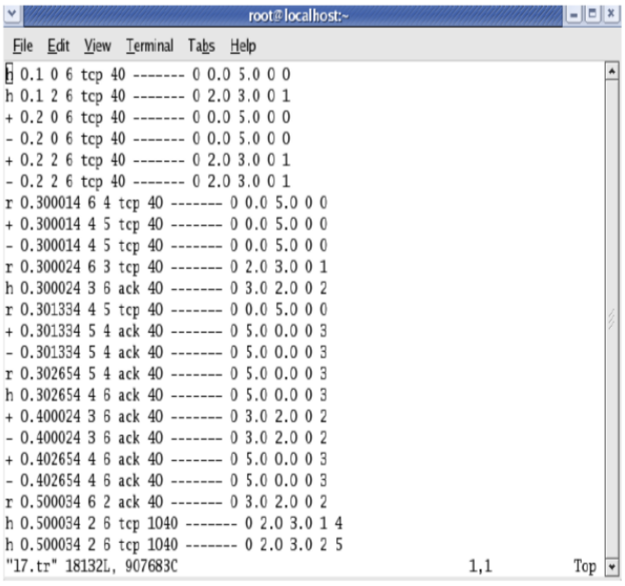
Error Rate	Data Rate	Throughput
0.1	0.0001	
0.2	0.0001	
0.3	0.0001	
0.4	0.0001	

Case 2: Fix the error rate to 0.1 and vary the data rate then throughput increases as shown below.

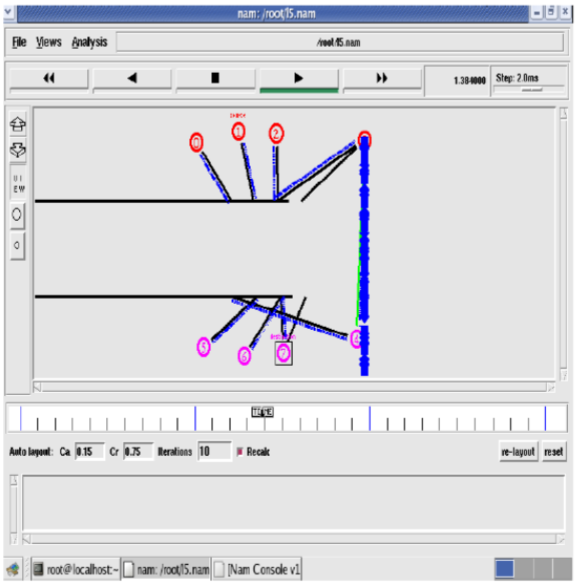
Error Rate	Data Rate	Throughput
0.1	0.1	
0.1	0.01	
0.1	0.001	
0.1	0.0001	

RESULT: Simulation of an Ethernet lan using 7 nodes, the Network throughput calculated by varying the error rate and data rate using NS-2.

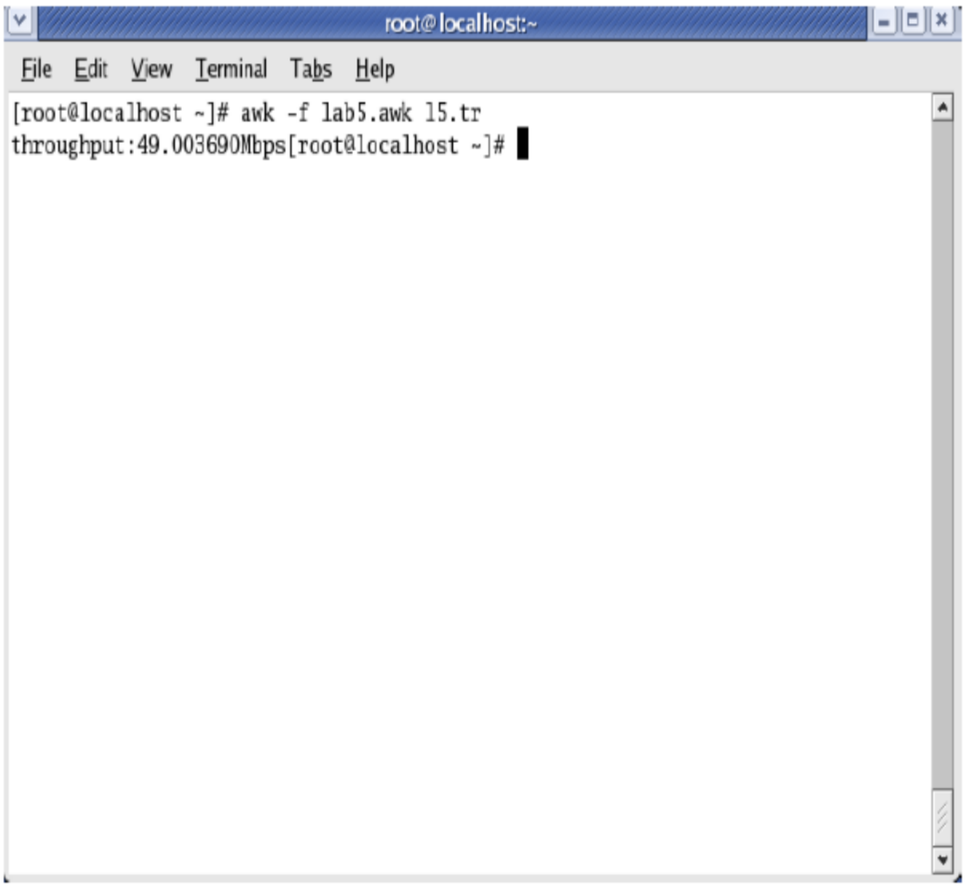
Here “**h**” indicates host.



```
h 0.1 0 6 tcp 40 ------- 0 0.0 5.0 0 0
h 0.1 2 6 tcp 40 ------- 0 2.0 3.0 0 1
+ 0.2 0 6 tcp 40 ------- 0 0.0 5.0 0 0
- 0.2 0 6 tcp 40 ------- 0 0.0 5.0 0 0
+ 0.2 2 6 tcp 40 ------- 0 2.0 3.0 0 1
- 0.2 2 6 tcp 40 ------- 0 2.0 3.0 0 1
r 0.300014 6 4 tcp 40 ------- 0 0.0 5.0 0 0
+ 0.300014 4 5 tcp 40 ------- 0 0.0 5.0 0 0
- 0.300014 4 5 tcp 40 ------- 0 0.0 5.0 0 0
r 0.300024 6 3 tcp 40 ------- 0 2.0 3.0 0 1
h 0.300024 3 6 ack 40 ------- 0 3.0 2.0 0 2
r 0.301334 4 5 tcp 40 ------- 0 0.0 5.0 0 0
+ 0.301334 5 4 ack 40 ------- 0 5.0 0.0 0 3
- 0.301334 5 4 ack 40 ------- 0 5.0 0.0 0 3
r 0.302654 5 4 ack 40 ------- 0 5.0 0.0 0 3
h 0.302654 4 6 ack 40 ------- 0 5.0 0.0 0 3
+ 0.400024 3 6 ack 40 ------- 0 3.0 2.0 0 2
- 0.400024 3 6 ack 40 ------- 0 3.0 2.0 0 2
+ 0.402654 4 6 ack 40 ------- 0 5.0 0.0 0 3
- 0.402654 4 6 ack 40 ------- 0 5.0 0.0 0 3
r 0.500034 6 2 ack 40 ------- 0 3.0 2.0 0 2
h 0.500034 2 6 tcp 1040 ------- 0 2.0 3.0 1 4
h 0.500034 2 6 tcp 1040 ------- 0 2.0 3.0 2 5
"17.tr" 18132L, 907683C                              1,1          Top
```



Topology **Output**



```
[root@localhost ~]# awk -f lab5.awk 15.tr
throughput:49.003690Mbps[root@localhost ~]#
```

PROGRAM 4:

IMPLEMENT ETHERNET LAN USING N NODES AND ASSIGN MULTIPLE TRAFFIC TO THE NODES AND OBTAIN CONGESTION WINDOW FOR DIFFERENT SOURCES AND DESTINATIONS.

ALGORITHM:

STEP 1: Create the simulator object ns for designing the given simulation.

STEP 2: Open the trace file and nam file in the write mode.

STEP 3: Create the nodes of the simulation using the ‘set’ command.

STEP 4: Create Ethernet LAN using make-LAN command and connect the nodes to the LAN.

STEP 5: Create TCP agent for the nodes and attach these agents to the nodes.

STEP 6: The traffic generator used is FTP for both node0 and node2.

STEP 7: Configure node3 and node1 as the sink and attach it.

STEP 8: Connect node0 and node3, node2 and node1 agents using ‘connect’ command.

STEP 9: Set color for the TCP packets.

STEP10: Define congestion window maximum value.

STEP 11: Schedule the events for FTP agent.

STEP 12: Schedule the simulation for 5 msec.

PROGRAM:

```
set ns [new Simulator]

set tf [open p4.tr w]
$ns trace-all $tf

set nf [open p4.nam w]
$ns namtrace-all $nf

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$ns make-lan "$n0 $n1 $n2 $n3" 10Mb 10ms LL Queue/DropTail Mac/802_3

set tcp1 [new Agent/TCP]
set ftp1 [new Application/FTP]
set sink1 [new Agent/TCPSink]
set tcp2 [new Agent/TCP]
set ftp2 [new Application/FTP]
set sink2 [new Agent/TCPSink]

$ns attach-agent $n0 $tcp1
$ftp1 attach-agent $tcp1
$ns attach-agent $n3 $sink1
$ns connect $tcp1 $sink1

$ns attach-agent $n2 $tcp2
$ftp2 attach-agent $tcp2
$ns attach-agent $n1 $sink2
$ns connect $tcp2 $sink2

set file1 [open file1.tr w]
$tcp1 attach $file1
$tcp1 trace cwnd_
$tcp1 set maxcwnd_ 10

set file2 [open file2.tr w]
$tcp2 attach $file2
$tcp2 trace cwnd_

$ns color 1 "red"
$ns color 2 "blue"

$tcp1 set class_ 1
$tcp2 set class_ 2

proc finish { } {
global nf tf ns
$ns flush-trace
exec nam p4.nam &
close $nf
close $tf
exit 0
}

$ns at 0.1 "$ftp1 start"
$ns at 1.5 "$ftp1 stop"

$ns at 2 "$ftp1 start"
$ns at 3 "$ftp1 stop"

$ns at 0.2 "$ftp2 start"
$ns at 2 "$ftp2 stop"

$ns at 2.5 "$ftp2 start"
$ns at 4 "$ftp2 stop"

$ns at 5.0 "finish"
$ns run
```

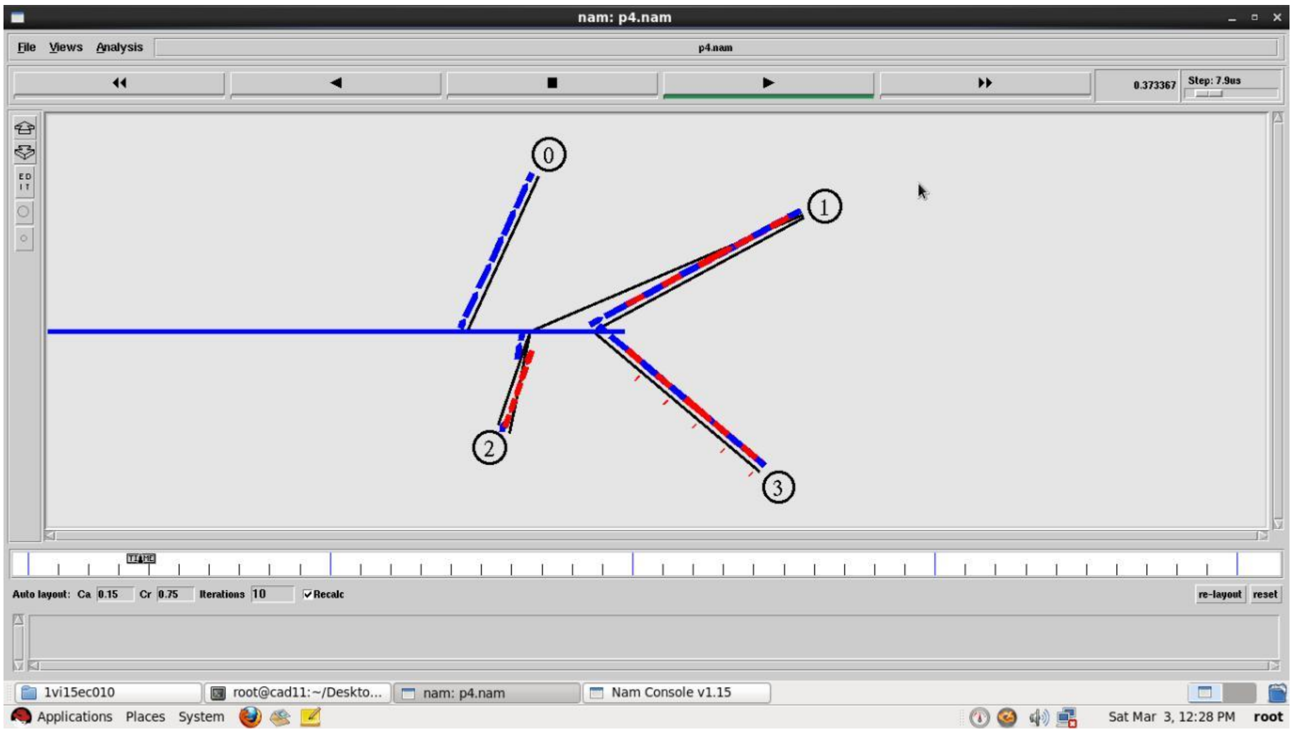
AWK SCRIPT:

```
BEGIN{
}
{
if($6=="cwnd_")
printf("%f\t%f\t\n",$1,$7);
}
END{
}
```

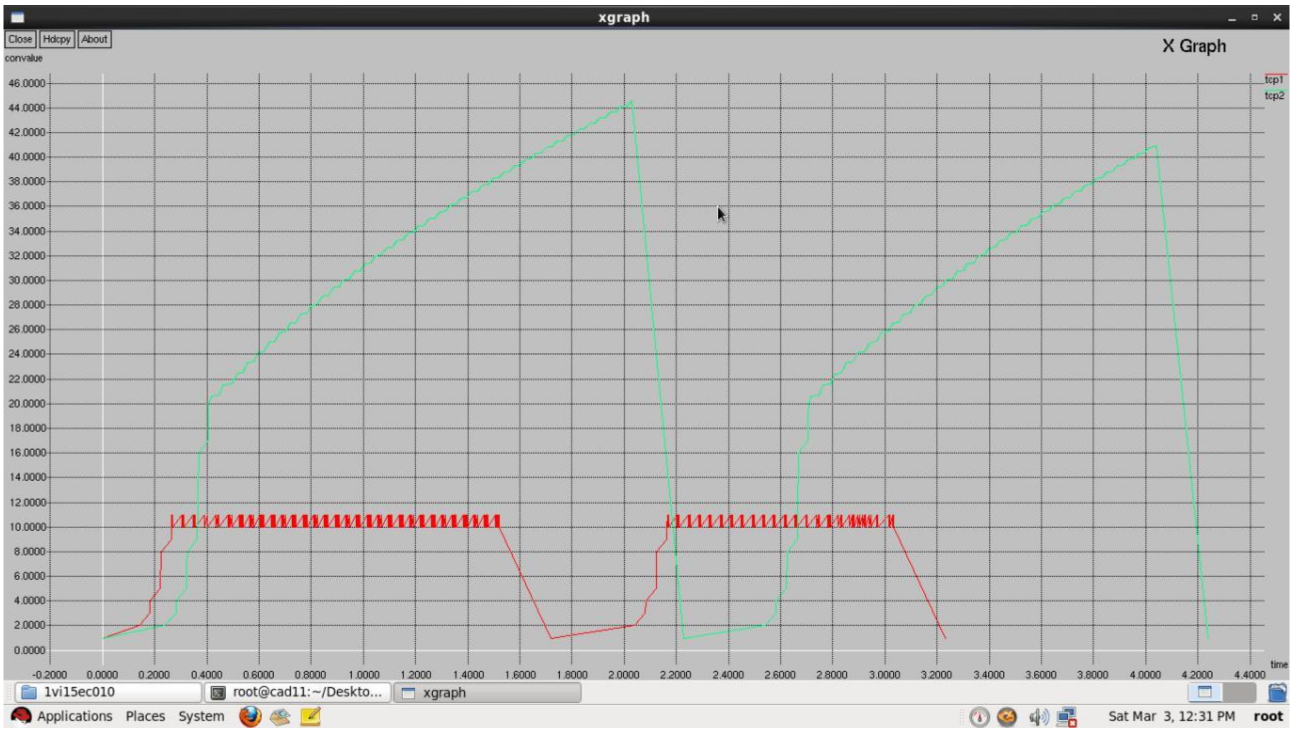
TO RUN:

```
ns filename.tcl
awk -f p4.awk file1.tr > file1
awk -f p4.awk file2.tr > file2
xgraph -x “time” -y “convalue” file1 file2
```

TOPOLOGY AND NAM RESULT:



Plot of congestion window for different sources and destination (tcp1 and tcp2)



RESULT: Ethernet LAN using n(four) nodes implemented by creating multiple traffic to the nodes and observed congestion window for different sources and destinations.

PROGRAM 5:

TO IMPLEMENT ESS WITH TRANSMISSION NODES IN WIRELESS LAN AND OBTAIN THE PERFORMANCE PARAMETERS.

ALGORITHM:

STEP 1: Create GOD for wireless nodes.

STEP 2: Create the simulator object ns for designing the given simulation.

STEP 3: Open the trace file and nam file in the write mode.

STEP 4: Create the 3 nodes of the simulation using the ‘set’ command.

STEP 5: Create a queue size between nodes.

STEP 6: Create TCP agent for the nodes and attach these agents to the nodes.

STEP 7: The traffic generator used is TCP and measured in terms of FTP0 and FTP1.

STEP 8: Create data transmission and node movements.

STEP 9: Schedule the simulation for 250 milisec.

PROGRAM:

```
set ns [new Simulator]

set tf [open p5.tr w]
$ns trace-all $tf

set nf [open p5.nam w]
$ns namtrace-all-wireless $nf 1000 1000

set topo [new Topography]
$topo load_flatgrid 1000 1000

$ns node-config -adhocRouting DSDV \
-llType LL \
-macType Mac/802_11 \
-ifqType Queue/DropTail \
-ifqLen 50 \
-phyType Phy/WirelessPhy \
-channelType Channel/WirelessChannel \
-propType Propagation/TwoRayGround \
-antType Antenna/OmniAntenna \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON

create-god 3
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

$n0 label "tcp0"
$n1 label "sink1/tcp1"
$n2 label "sink2"
$ns initial_node_pos $n0 50
$ns initial_node_pos $n1 50
$ns initial_node_pos $n2 50

#The below code is used to give the initial node positions.
$n0 set X_ 50
$n0 set Y_ 50
$n1 set X_ 200
$n1 set Y_ 200
$n2 set X_ 700
$n2 set Y_ 700

$ns at 0.1 "$n0 setdest 50 50 15"
$ns at 0.1 "$n1 setdest 200 200 25"
$ns at 0.1 "$n2 setdest 700 700 25"
set tcp0 [new Agent/TCP]
set ftp0 [new Application/FTP]
set sink1 [new Agent/TCPSink]
set tcp1 [new Agent/TCP]
set ftp1 [new Application/FTP]
set sink2 [new Agent/TCPSink]
```

```
$ns attach-agent $n0 $tcp0
$ftp0 attach-agent $tcp0
$ns attach-agent $n1 $sink1
$ns connect $tcp0 $sink1
$ns attach-agent $n1 $tcp1
$ftp1 attach-agent $tcp1
$ns attach-agent $n2 $sink2
$ns connect $tcp1 $sink2

$ns at 0.1 "$ftp0 start"
$ns at 0.1 "$ftp1 start"

#The below code is used to provide the node movements.
$ns at 100 "$n1 setdest 550 550 15"
$ns at 190 "$n1 setdest 100 100 15"

proc finish {} {
global ns nf tf
$ns flush-trace
exec nam p5.nam &

close $tf
exit 0
}

$ns at 250 "finish"
$ns run
```

AWK SCRIPT:

```
BEGIN{
count1=0
count2=0
pack1=0
pack2=0
time1=0
time2=0
}
{
if($1=="r"&& $3=="_1_" && $4=="AGT")
{
count1++
pack1=pack1+$8
time1=$2
}
if($1=="r"&& $3=="_2_"&& $4=="AGT")
{
count2++
pack2=pack2+$8
time2=$2
}
}
END
```

```
{
printf("The Throughput from n0 to n1:
%fMbps\n",((count1*pack1*8)/(time1*1000000))) Printf("The Throughput from n1 to
n2:%f Mbps", ((count2* pack2 * 8) /(time2*1000000)))
}
```

OUTPUT:

```
The Throughput from n0 to n1:_______Mbps
The Throughput from n1 to n2:_______Mbps
```

Trace file contents and format for wireless network

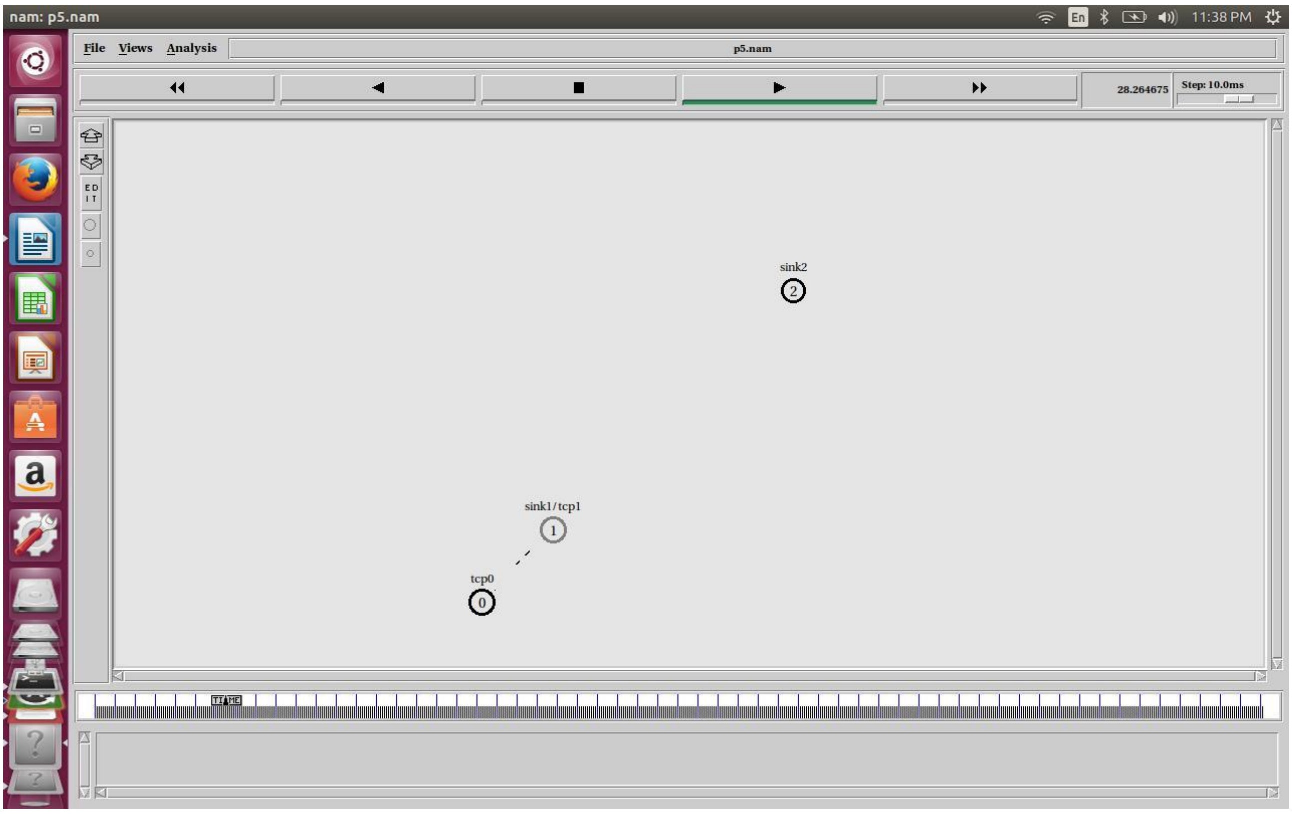
ACTION:	[s|r|D]: s -- sent, r -- received, D -- dropped
WHEN:	the time when the action happened
WHERE:	the node where the action happened
LAYER:	AGT -- application,
	RTR -- routing,
	LL -- link layer (ARP is done here)
	IFQ -- outgoing packet queue (between link and mac layer)
	MAC -- mac,
	PHY -- physical
flags:	
SEQNO:	the sequence number of the packet
TYPE:	the packet type
	cbr -- CBR data stream packet
	DSR -- DSR routing packet (control packet generated by routing)
	RTS -- RTS packet generated by MAC 802.11
	ARP -- link layer ARP packet
SIZE:	the size of packet at current layer, when packet goes down, size increases, goes up size decreases
[a b c d]:	a -- the packet duration in mac layer header
	b -- the mac address of destination
	c -- the mac address of source
	d -- the mac type of the packet body
flags:	
[......]:	[
	source node ip : port_number
	destination node ip (-1 means broadcast) : port_number
	ip header ttl
	ip of next hop (0 means node 0 or broadcast)
	]

Example:

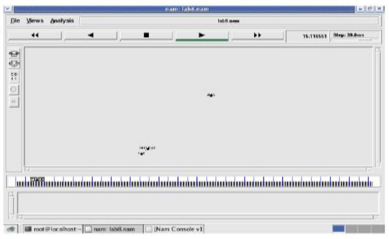
s 0.032821055 _1_ RTR --- 0 message 32 [0 0 0 0] ------- [1:255 -1:255 32 0]

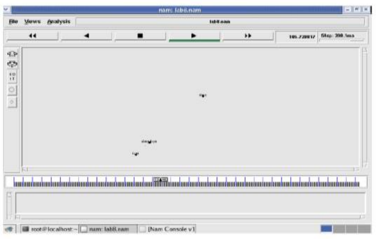
- It represent packet is sent at time .032821055 second by node 1.
- RTR represent that this packet is routing packet.
- Next three dashes are for flags.
- Sequence number of packet is 0. Type of packet is message.
- Size of this packet is 32 bytes.
- [0 0 0 0] first zero in bracket represent packet duration in mac layer header, second 0 is for mac address for destination, third zero for mac address for source and fourth zero for mac type of the packet body.
- Next dashes "-------" are flags, one of the flags is used for energy information of node.
- [1:255 -1:255 32 0] source node IP and port are 1 and 255 respectively, destination IP and port are -1 and 255.
- next field 32 is ttl field in IP header

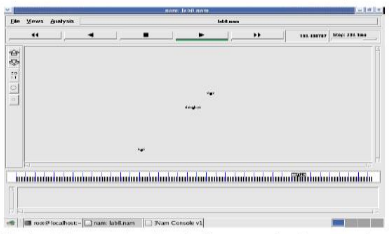
TOPOLOGY AND NAM RESULT:

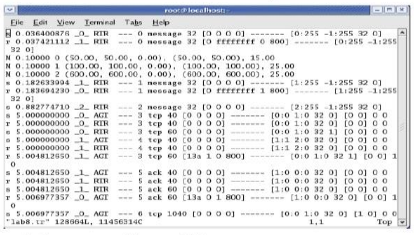


Packet transmission by tcp from n0 to n1


Node 1 and 2 are communicating


Node 2 is moving towards node 3

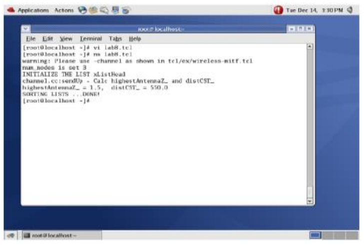

Node 2 is coming back from node 3 towards node1



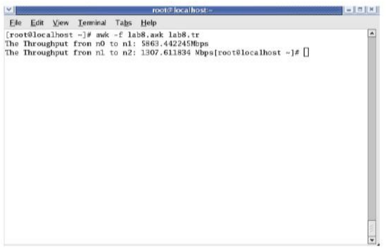
```
s 0.036400876 _0_ RTR  --- 0 message 32 [0 0 0 0] ------- [0:255 -1:255 32 0]
r 0.037421112 _1_ RTR  --- 0 message 32 [0 ffffffff 0 800] ------- [0:255 -1:255
 32 0]
M 0.10000 0 (50.00, 50.00, 0.00), (50.00, 50.00), 15.00
M 0.10000 1 (100.00, 100.00, 0.00), (100.00, 100.00), 25.00
M 0.10000 2 (600.00, 600.00, 0.00), (600.00, 600.00), 25.00
s 0.182633994 _1_ RTR  --- 1 message 32 [0 0 0 0] ------- [1:255 -1:255 32 0]
r 0.183694230 _0_ RTR  --- 1 message 32 [0 ffffffff 1 800] ------- [1:255 -1:255
 32 0]
s 0.882774710 _2_ RTR  --- 2 message 32 [0 0 0 0] ------- [2:255 -1:255 32 0]
s 5.000000000 _0_ AGT  --- 3 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
r 5.000000000 _0_ RTR  --- 3 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
s 5.000000000 _0_ RTR  --- 3 tcp 60 [0 0 0 0] ------- [0:0 1:0 32 1] [0 0] 0 0
s 5.000000000 _1_ AGT  --- 4 tcp 40 [0 0 0 0] ------- [1:1 2:0 32 0] [0 0] 0 0
r 5.000000000 _1_ RTR  --- 4 tcp 40 [0 0 0 0] ------- [1:1 2:0 32 0] [0 0] 0 0
r 5.004812650 _1_ AGT  --- 3 tcp 60 [13a 1 0 800] ------- [0:0 1:0 32 1] [0 0] 1
 0
s 5.004812650 _1_ AGT  --- 5 ack 40 [0 0 0 0] ------- [1:0 0:0 32 0] [0 0] 0 0
r 5.004812650 _1_ RTR  --- 5 ack 40 [0 0 0 0] ------- [1:0 0:0 32 0] [0 0] 0 0
s 5.004812650 _1_ RTR  --- 5 ack 60 [0 0 0 0] ------- [1:0 0:0 32 0] [0 0] 0 0
r 5.006977357 _0_ AGT  --- 5 ack 60 [13a 0 1 800] ------- [1:0 0:0 32 0] [0 0] 1
 0
s 5.006977357 _0_ AGT  --- 6 tcp 1040 [0 0 0 0] ------- [0:0 1:0 32 0] [1 0] 0 0
"lab8.tr" 128664L, 11456314C                                      1,1          Top
```

Trace File

Here **"M"** indicates mobile nodes, **"AGT"** indicates Agent Trace, **"RTR"** indicates Route Trace



```
[root@localhost ~]# vi lab8.tcl
[root@localhost ~]# ns lab8.tcl
warning: Please use -channel as shown in tcl/ex/wireless-mitf.tcl
num_nodes is set 3
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5,  distCST_ = 550.0
SORTING LISTS ...DONE!
[root@localhost ~]#
```



```
[root@localhost ~]# awk -f lab8.awk lab8.tr
The Throughput from n0 to n1: 5863.442245Mbps
The Throughput from n1 to n2: 1307.611834 Mbps[root@localhost ~]# []
```

PROGRAM 6:

IMPLEMENTATION OF LINK STATE ROUTING ALGORITHM ALGORITHM:

STEP 1: Create the simulator object ns for designing the given simulation

STEP 2: Open the trace file and nam file in the write mode

STEP 3: Create the 12 nodes of the simulation using the 'set' command and duplex link between them

STEP 4: Create a queue size between nodes

STEP 5: Create UDP agent for the nodes and attach these agents to the nodes

STEP 6: The traffic generator used is UDP and measured in terms of cbr0 and cbr1

STEP 7: Configure node2 and node 3 as the null and attach it

STEP 8: Connect duplex links between the nodes

STEP 9: Create rtmodel and link between 5 and 11 is disconnected

STEP 10: Schedule the simulation for 50 milisec

#TO CREATE SIMULATOR OBJECT, TRACE FILE AND NAM FILE

```
set ns [new Simulator]
set nr [open p6.tr w]
$ns trace-all $nr
set nf [open p6.nam w]
$ns namtrace-all $nf

#Using routing protocol
$ns rtproto LS


#Creation of Nodes
for {set i 0} {$i < 12} {incr i} {
   set n($i) [$ns node]
   }

#Creation of links
$ns duplex-link $n(0) $n(1) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(2) 1Mb 10ms DropTail
$ns duplex-link $n(2) $n(3) 1Mb 10ms DropTail
$ns duplex-link $n(3) $n(4) 1Mb 10ms DropTail
$ns duplex-link $n(4) $n(5) 1Mb 10ms DropTail
$ns duplex-link $n(5) $n(6) 1Mb 10ms DropTail
$ns duplex-link $n(6) $n(7) 1Mb 10ms DropTail
```

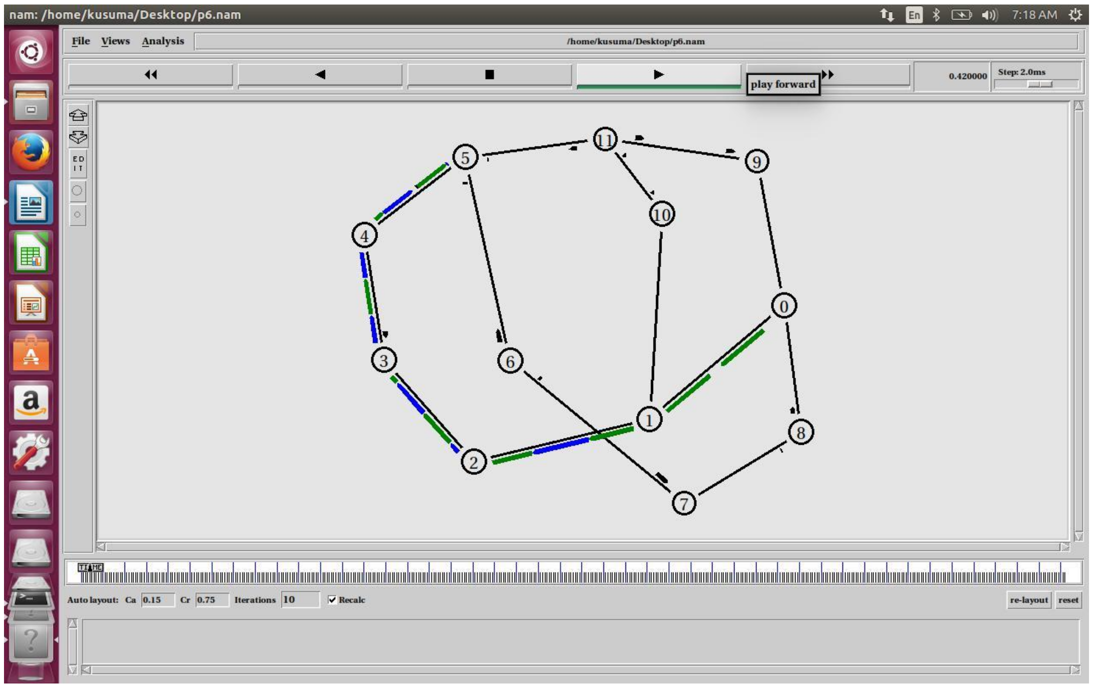
```
$ns duplex-link $n(7) $n(8) 1Mb 10ms DropTail
$ns duplex-link $n(8) $n(0) 1Mb 10ms DropTail
$ns duplex-link $n(0) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(10) 1Mb 10ms DropTail
$ns duplex-link $n(9) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(10) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(11) $n(5) 1Mb 10ms DropTail

#Set up UDP connection
set udp0 [new Agent/UDP]
set cbr0 [new Application/Traffic/CBR]
set null0 [new Agent/Null]
set udp1 [new Agent/UDP]
set cbr1 [new Application/Traffic/CBR]
set null1 [new Agent/Null]

$ns attach-agent $n(0) $udp0
$cbr0 attach-agent $udp0
$ns attach-agent $n(5) $null0
$ns connect $udp0 $null0


$ns attach-agent $n(1) $udp1
$cbr1 attach-agent $udp1
$ns attach-agent $n(5) $null1
$ns connect $udp1 $null1

#Set up CBR over UDP connection


$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005


$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005


#Schedule events for CBR and FTP agents

$ns at 0.1 "$cbr0 start"
$ns at 0.1 "$cbr1 start"

$ns rtmodel-at 10.0 down $n(11) $n(5)    ;# communication link between node 11
                                    ;#to 5 disconnected
$ns rtmodel-at 30.0 up $n(11) $n(5)      ;# communication link between node 11
```

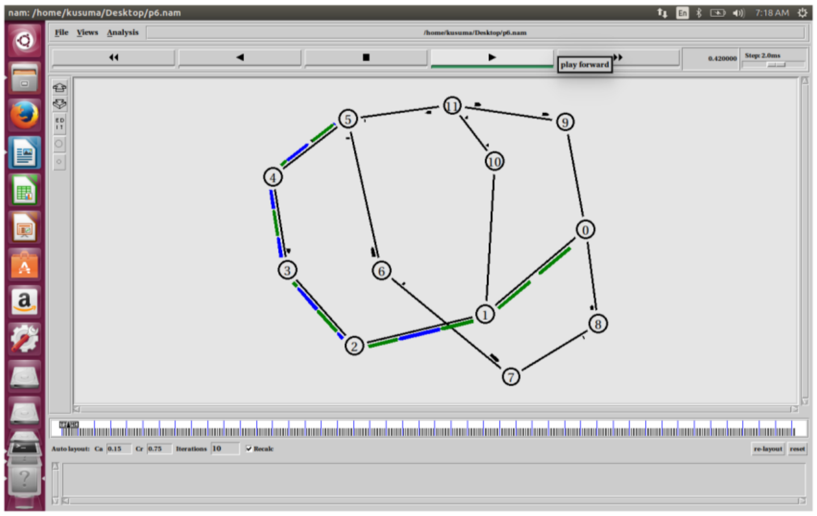
```
;#to 5 reconnected

$ns rtmodel-at 15.0 down $n(7) $n(6)
$ns rtmodel-at 20.0 up $n(7) $n(6)

$ns color 1 "green"
$ns color 2 "blue"

$udp0 set class_ 1
$udp1 set class_ 2

#Define procedure to clean up
proc finish { } {
    global ns nr nf
    $ns flush-trace
    close $nf
    close $nr
    exec nam p6.nam &
    exit 0
}

$ns at 45.0 "$cbr0 stop"
$ns at 45.0 "$cbr1 stop"
$ns at 50.0 "finish"

#Run the simulation
$ns run
```

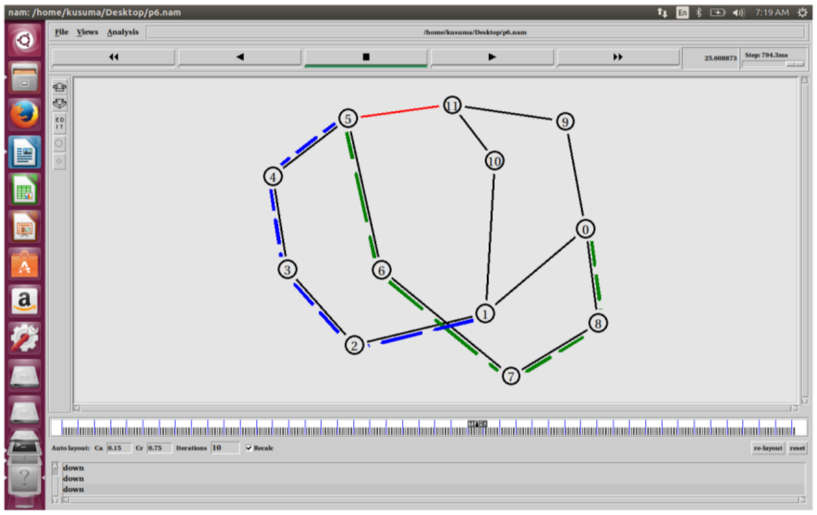
AWK SCRIPT:

```
BEGIN {
pksend = 0
pkreceive = 0
pkdrop = 0
pkrouting = 0
}

#Executed for each line of input file thru.tr
{
#For udp packets
if ( $1=="+" && ($3=="0" || $3=="11") && $5=="cbr" )
{
pksend++;
}
if ( $1=="r" && $4=="5" && $5=="cbr" )

{
pkreceive++;
}
if ( $1=="d" )
{
pkdrop++;
}
if ( $1=="r" && ($5=="rtProtoDV" || $5=="rtProtoLS") )
{
pkrouting++;
}
}
END {
print "No of send packets = " pksend
print "No of received packets = " pkreceive
print "No of dropped packets = " pkdrop
print "No of routing packets = " pkrouting
print "Normalized Overhead (routing/received), NOH = " pkrouting/pkreceive
print "Packet Delivery Ratio (received/send), PDR = " pkreceive/pksend }
```

<u>TOPOLOGY AND NAM:</u>







```
down
down
down
```

OUTPUT:

No of Send Packets= 18959

No of Received Packets =13543

No of Dropped Packets= 4349

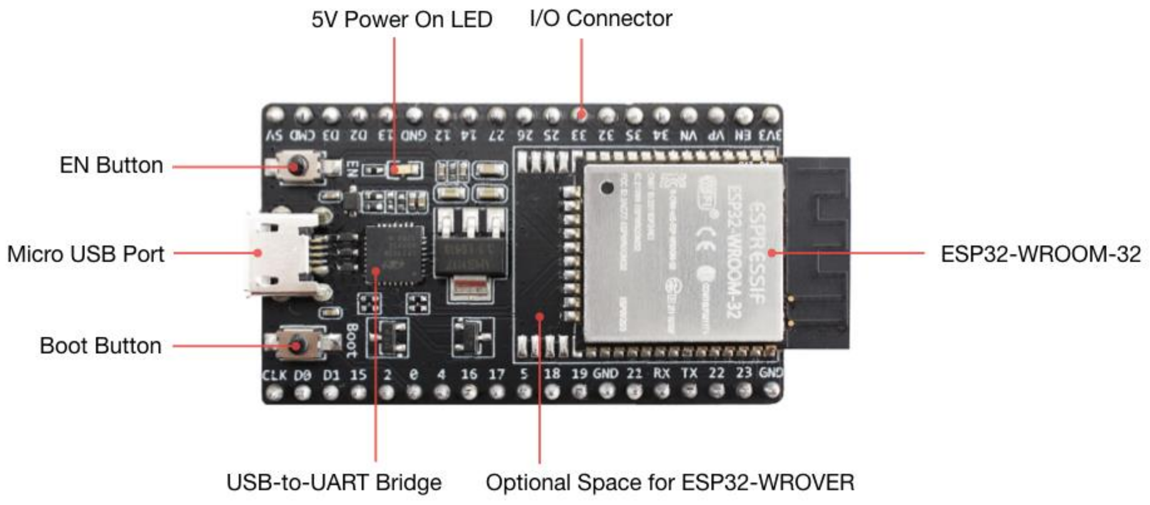
No of Routing Packets=656

Part B

ESP32

ESP32 is a series of low-cost, low-power system on a chip microcontroller with integrated Wi-Fi and dual-mode Bluetooth. At the core of this module is the ESP32-D0WDQ6 chip*. The chip embedded is designed to be scalable and adaptive. There are two CPU cores that can be individually controlled, and the clock frequency is adjustable from 80 MHz to 240 MHz. The user may also power off the CPU and make use of the low-power co-processor to constantly monitor the peripherals for changes or crossing of thresholds. ESP32 integrates a rich set of peripherals, ranging from capacitive touch sensors, Hall sensors, SD card interface, Ethernet, high-speed SPI, UART, I2S and I2C.

- The ESP32 is dual core.
- It has Wi-Fi and Bluetooth built-in.
- It runs 32-bit programs.
- The clock frequency can go up to 240MHz and it has a 512 kB RAM.
- This particular board has 30 or 36 pins, 15 in each row.
- It also has wide variety of peripherals available, like: capacitive touch, ADCs, DACs, UART, SPI, I2C and much more.
- It comes with built-in hall effect sensor and built-in temperature sensor.



Key Component	Description
ESP32-WROOM-32	A module with ESP32 at its core. For more information
EN	Reset button.
Boot	Download button. Holding down **Boot** and then pressing **EN** initiates Firmware Download mode for downloading firmware through the serial port.
USB-to-UART Bridge	Single USB-UART bridge chip provides transfer rates of up to 3 Mbps.
Micro USB Port	USB interface. Power supply for the board as well as the communication interface between a computer and the ESP32-WROOM-32 module.

5V Power On LED	Turns on when the USB or an external 5V power supply is connected to the board.
I/O	Most of the pins on the ESP module are broken out to the pin headers on the board. You can program ESP32 to enable multiple functions such as PWM, ADC, DAC, I2C, I2S, SPI, etc.

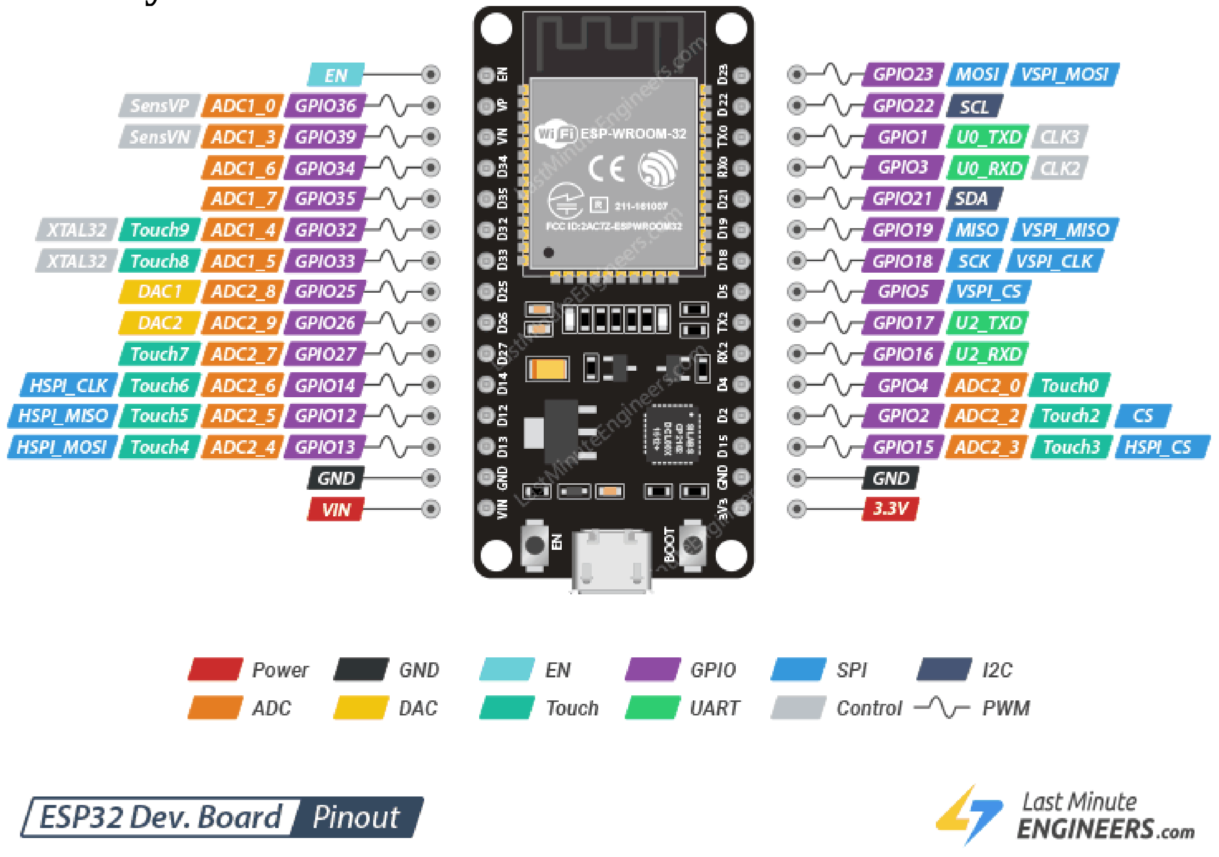
Note: The pins D0, D1, D2, D3, CMD and CLK are used internally for communication between ESP32 and SPI flash memory. They are grouped on both sides near the USB connector. Avoid using these pins, as it may disrupt access to the SPI flash memory / SPI RAM.

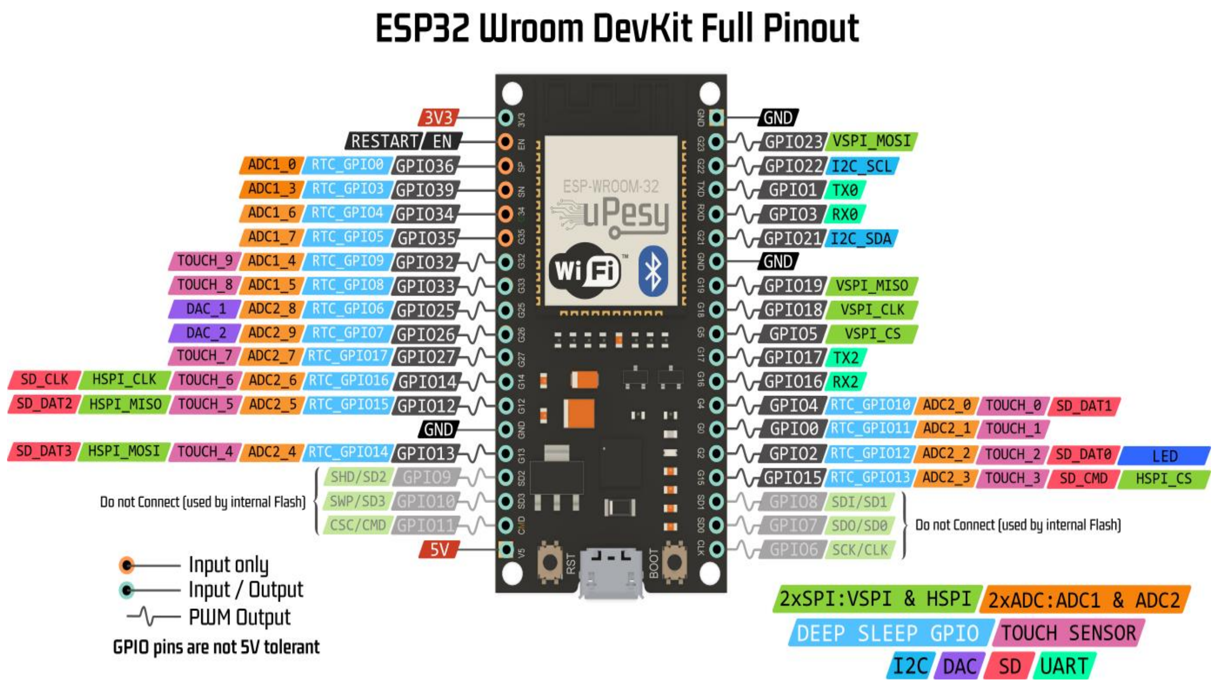
Features of the ESP32 include the following:

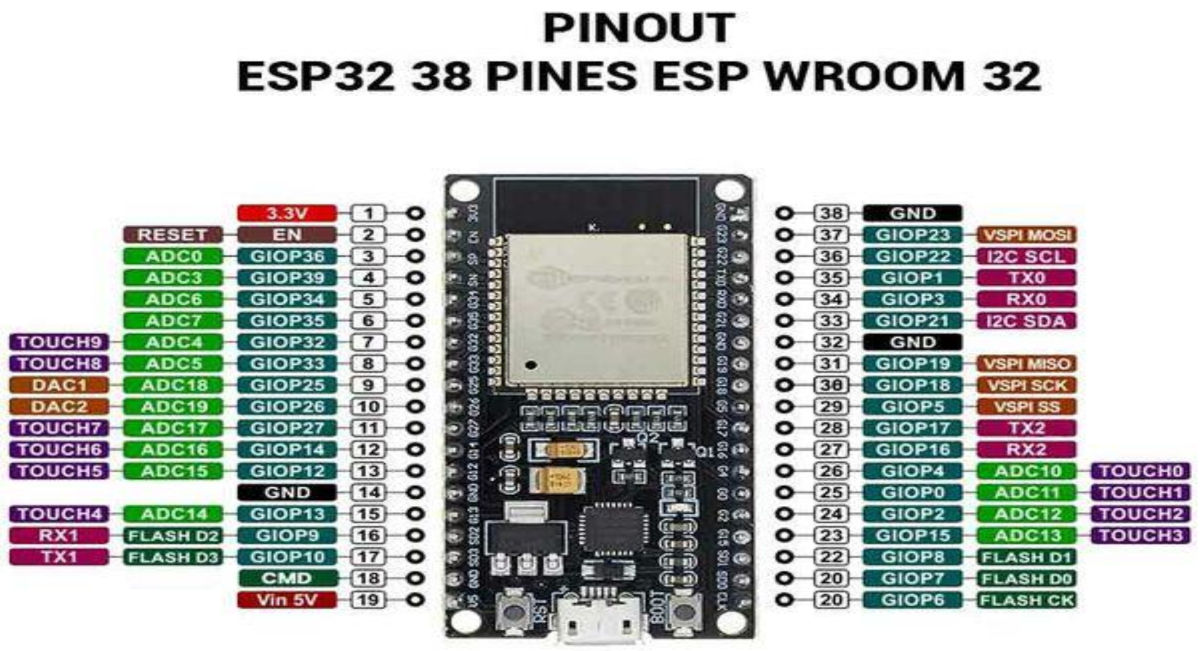
- **Processors:**
  - CPU: Xtensa dual-core (or single-core) 32-bit LX6 microprocessor, operating at 160 or 240 MHz and performing at up to 600 DMIPS
  - Ultra-low power (ULP) co-processor
- **Memory:** 320 KiB RAM, 448 KiB ROM
- **Wireless connectivity:**
  - Wi-Fi: 802.11 b/g/n
  - Bluetooth: v4.2 BR/EDR and BLE (shares the radio with Wi-Fi)
- **Peripheral interfaces:**
  - 34 × programmable GPIOs
  - 12-bit SAR ADC up to 18 channels
  - 2 × 8-bit DACs
  - 10 × touch sensors (capacitive sensing GPIOs)
  - 4 × SPI
  - 2 × I²S interfaces
  - 2 × I²C interfaces
  - 3 × UART
  - SD/SDIO/CE-ATA/MMC/eMMC host controller
  - SDIO/SPI slave controller
  - Ethernet MAC interface with dedicated DMA and planned IEEE 1588 Precision Time Protocol support[4]
  - CAN bus 2.0
  - Infrared remote controller (TX/RX, up to 8 channels)
  - Motor PWM
  - LED PWM (up to 16 channels)
  - Hall effect sensor
  - Ultra-low power analog pre-amplifier
- **Security:**
  - IEEE 802.11 standard security features all supported, including WPA, WPA2, WPA3 (depending on version)[5] and WLAN Authentication and Privacy Infrastructure (WAPI)
  - Secure boot
  - Flash encryption
  - 1024-bit OTP, up to 768-bit for customers

  - Cryptographic hardware acceleration: AES, SHA-2, RSA, elliptic curve cryptography (ECC), random number generator (RNG)
- **Power management:**
  - Internal low-dropout regulator
  - Individual power domain for RTC
  - 5 µA deep sleep current
  - Wake up from GPIO interrupt, timer, ADC measurements, capacitive touch sensor interrupt

Pin Layout:




ESP32 Wroom DevKit Full Pinout
3V3
RESTART / EN
ADC1_0 RTC_GPIO0 GPIO36
ADC1_3 RTC_GPIO3 GPIO39
ADC1_6 RTC_GPIO4 GPIO34
ADC1_7 RTC_GPIO5 GPIO35
TOUCH_9 ADC1_4 RTC_GPIO9 GPIO32
TOUCH_8 ADC1_5 RTC_GPIO8 GPIO33
DAC_1 ADC2_8 RTC_GPIO6 GPIO25
DAC_2 ADC2_9 RTC_GPIO7 GPIO26
TOUCH_7 ADC2_7 RTC_GPIO17 GPIO27
SD_CLK HSPI_CLK TOUCH_6 ADC2_6 RTC_GPIO16 GPIO14
SD_DAT2 HSPI_MISO TOUCH_5 ADC2_5 RTC_GPIO15 GPIO12
GND
SD_DAT3 HSPI_MOSI TOUCH_4 ADC2_4 RTC_GPIO14 GPIO13
SHD/SD2 GPIO9
SWP/SD3 GPIO10
CSC/CMD GPIO11
5V
Do not Connect (used by internal Flash)
GND
GPIO23 VSPI_MOSI
GPIO22 I2C_SCL
GPIO1 TX0
GPIO3 RX0
GPIO21 I2C_SDA
GND
GPIO19 VSPI_MISO
GPIO18 VSPI_CLK
GPIO5 VSPI_CS
GPIO17 TX2
GPIO16 RX2
GPIO4 RTC_GPIO10 ADC2_0 TOUCH_0 SD_DAT1
GPIO0 RTC_GPIO11 ADC2_1 TOUCH_1
GPIO2 RTC_GPIO12 ADC2_2 TOUCH_2 SD_DAT0 LED
GPIO15 RTC_GPIO13 ADC2_3 TOUCH_3 SD_CMD HSPI_CS
GPIO8 SDI/SD1
GPIO7 SDO/SD0
GPIO6 SCK/CLK
Do not Connect (used by internal Flash)
ESP-WROOM-32
uPesy
RST
BOOT
Input only
Input / Output
PWM Output
GPIO pins are not 5V tolerant
2xSPI:VSPI & HSPI
2xADC:ADC1 & ADC2
DEEP SLEEP GPIO
TOUCH SENSOR
I2C
DAC
SD
UART



For more details : **https://www.upesy.com/blogs/tutorials/esp32-pinout-reference-gpio-pins-ultimate-guide**
https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf

Install the Arduino Desktop IDE:

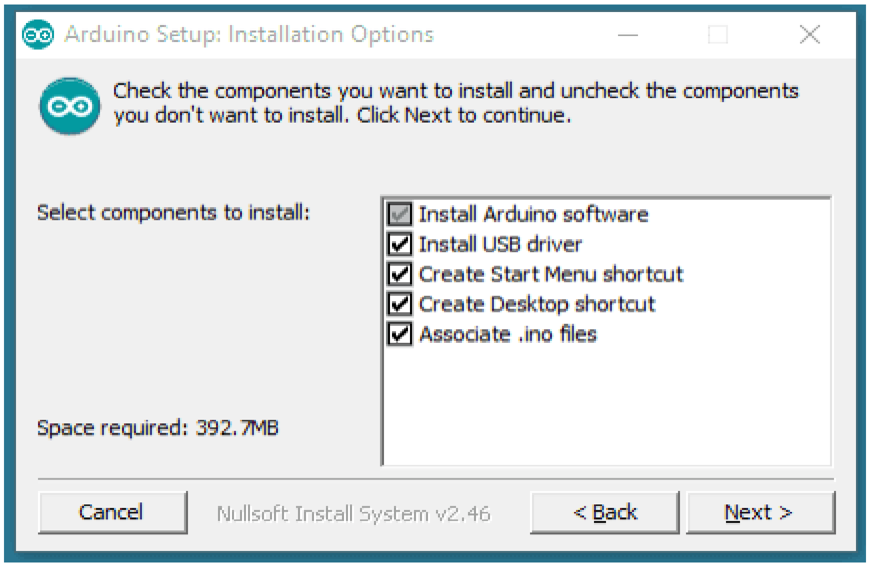
Arduino IDE Installation (Windows):

Step 1: Download the Arduino Software (IDE):

https://www.arduino.cc/en/software

Step 2:

When the download finishes, proceed with the installation and please allow the driver installation process when you get a warning from the operating system.



Arduino Setup: Installation Options

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

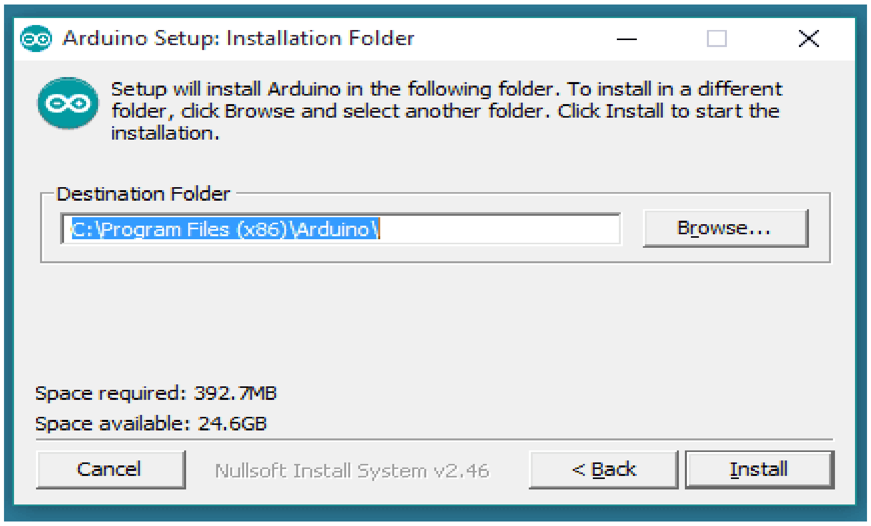
Select components to install:

- [x] Install Arduino software
- [x] Install USB driver
- [x] Create Start Menu shortcut
- [x] Create Desktop shortcut
- [x] Associate .ino files

Space required: 392.7MB

Choose the components to install.

Step 3:



Arduino Setup: Installation Folder

Setup will install Arduino in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

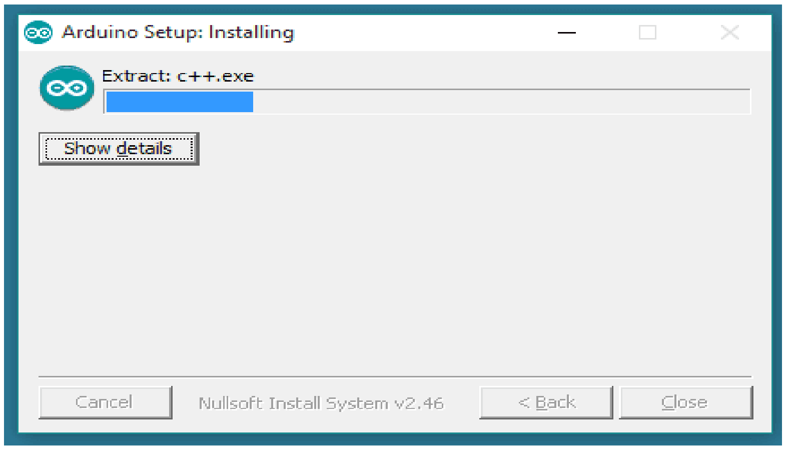
Destination Folder: C:\Program Files (x86)\Arduino\

Space required: 392.7MB

Space available: 24.6GB

Choose the installation directory.

Step 4:



Installation in progress.

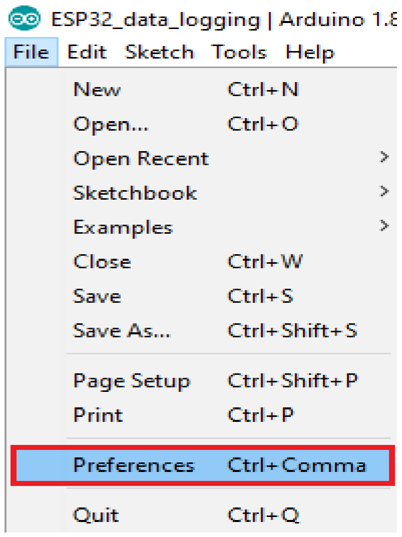
The process will extract and install all the required files to execute properly the Arduino Software (IDE)

Installing the ESP32 Board in Arduino IDE (Windows, Mac OS X, Linux)

There's an add-on for the Arduino IDE that allows you to program the ESP32 using the Arduino IDE and its programming language.

To install the ESP32 board in your Arduino IDE, follow these Steps:

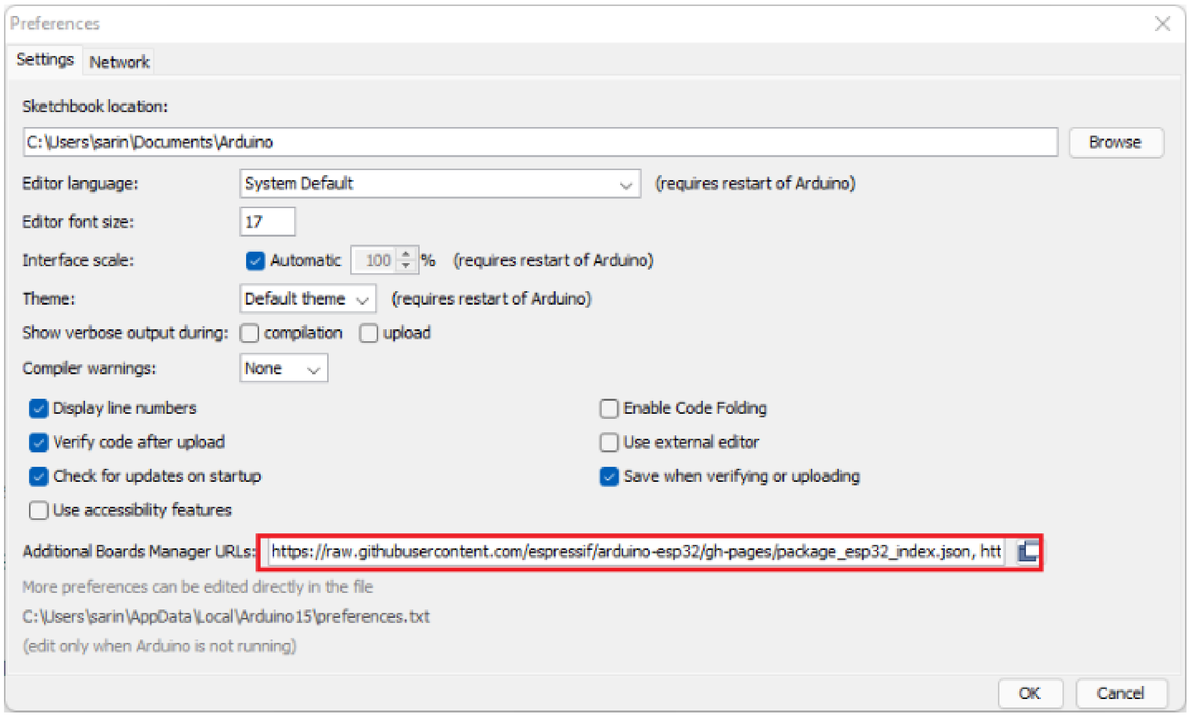
Step 1: In your Arduino IDE, go to File> Preferences



Step 2: Enter the following into the "Additional Board Manager URLs" field:

```
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json
```

Then, click the "OK" button:



Preferences

Settings | Network

Sketchbook location: C:\Users\sarin\Documents\Arduino

Editor language: System Default (requires restart of Arduino)

Editor font size: 17

Interface scale: ✓ Automatic 100 % (requires restart of Arduino)

Theme: Default theme (requires restart of Arduino)

Show verbose output during: ☐ compilation ☐ upload

Compiler warnings: None

✓ Display line numbers
☐ Enable Code Folding
✓ Verify code after upload
☐ Use external editor
✓ Check for updates on startup
✓ Save when verifying or uploading
☐ Use accessibility features

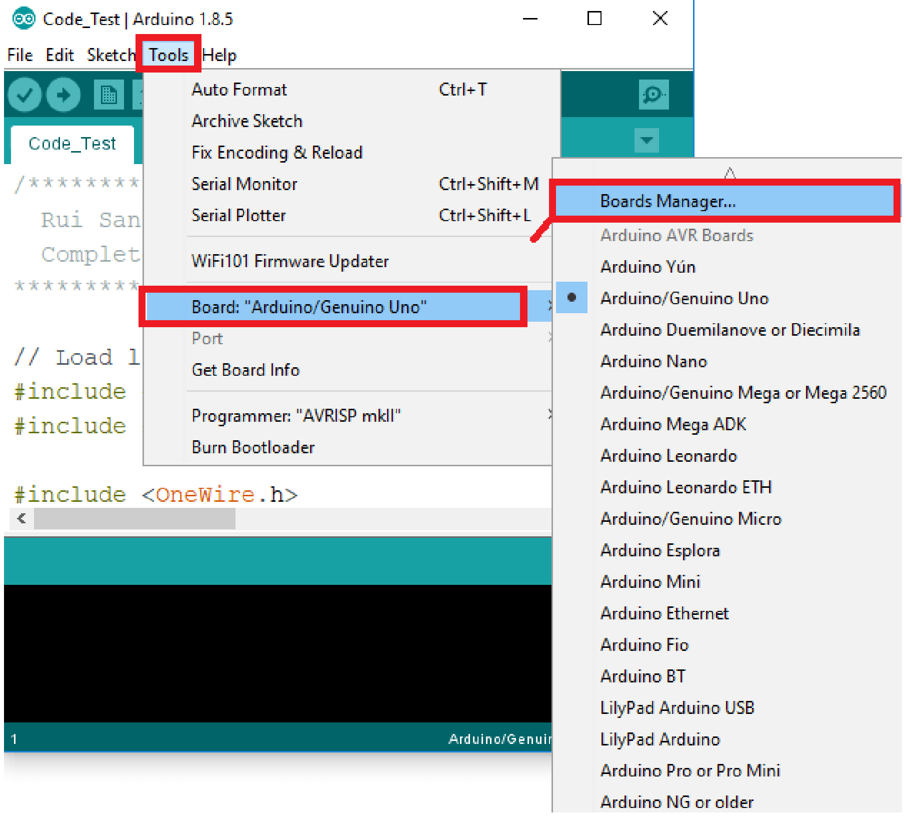
Additional Boards Manager URLs: https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json, htt

More preferences can be edited directly in the file
C:\Users\sarin\AppData\Local\Arduino15\preferences.txt
(edit only when Arduino is not running)

Note: if you already have the ESP8266 boards URL, you can separate the URLs with a comma as follows:

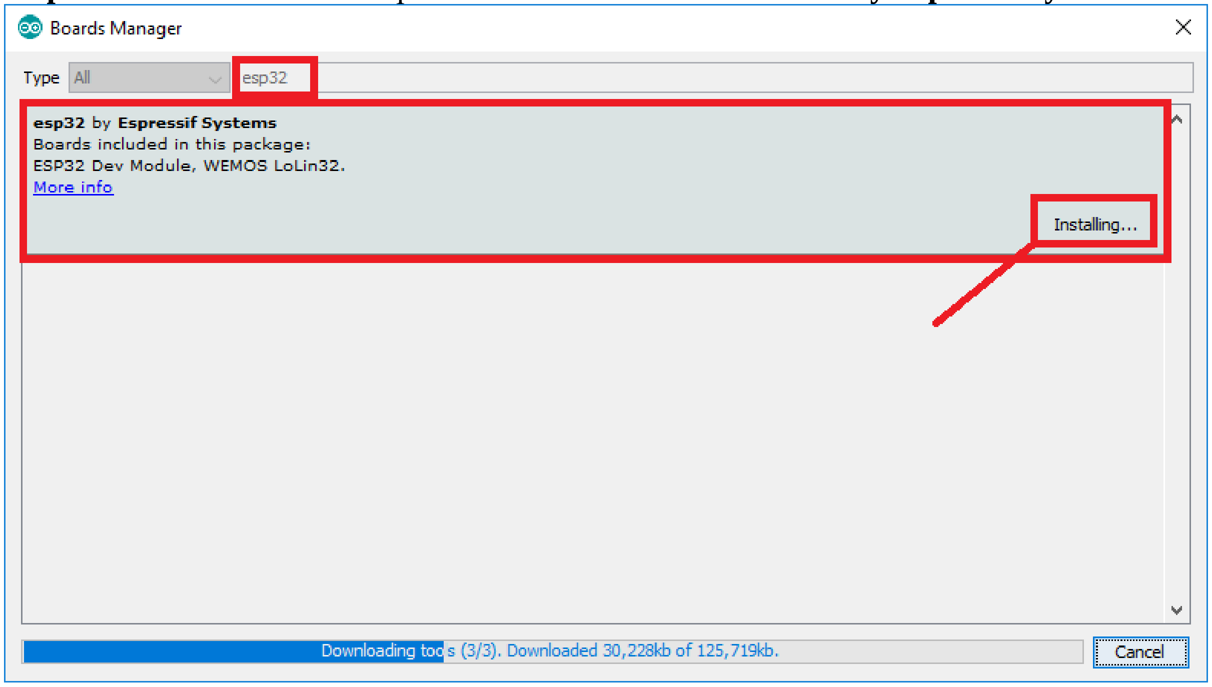
`https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json,` [`http://arduino.esp8266.com/stable/package_esp8266com_index.json`](http://arduino.esp8266.com/stable/package_esp8266com_index.json)

Step 3: Open the Boards Manager. **Go to Tools > Board > Boards Manager…**



```
/********
  Rui San
  Complet
*********

// Load l
#include
#include

#include <OneWire.h>
```

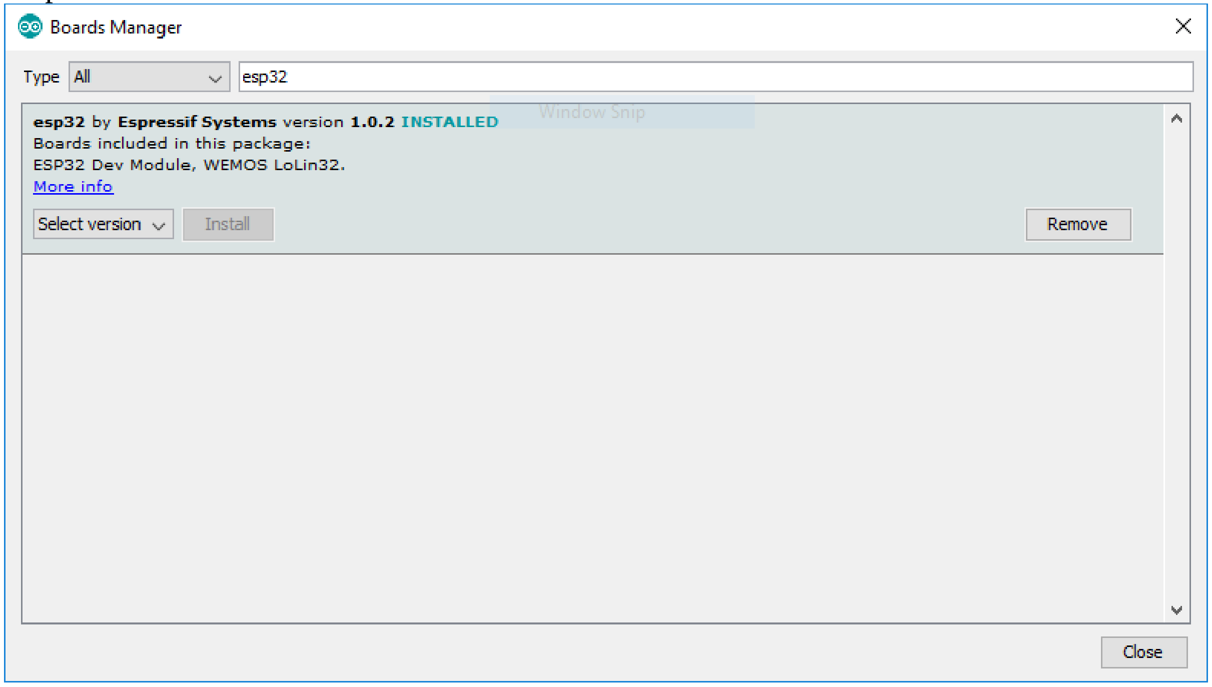
Step 4: Search for ESP32 and press install button for the "**ESP32 by Espressif Systems**":



esp32 by **Espressif Systems**
Boards included in this package:
ESP32 Dev Module, WEMOS LoLin32.
More info

Downloading tools (3/3). Downloaded 30,228kb of 125,719kb.

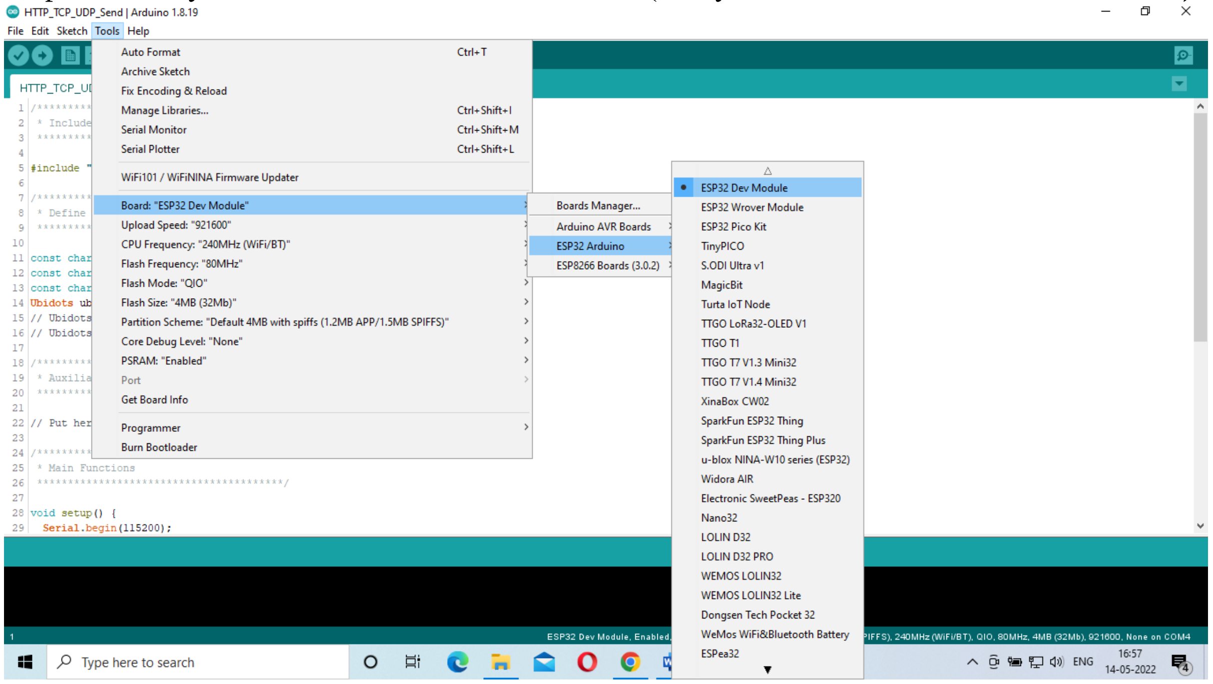
Step 5: That's it. It should be installed after a few seconds.



esp32 by **Espressif Systems** version **1.0.2** **INSTALLED**
Boards included in this package:
ESP32 Dev Module, WEMOS LoLin32.
More info

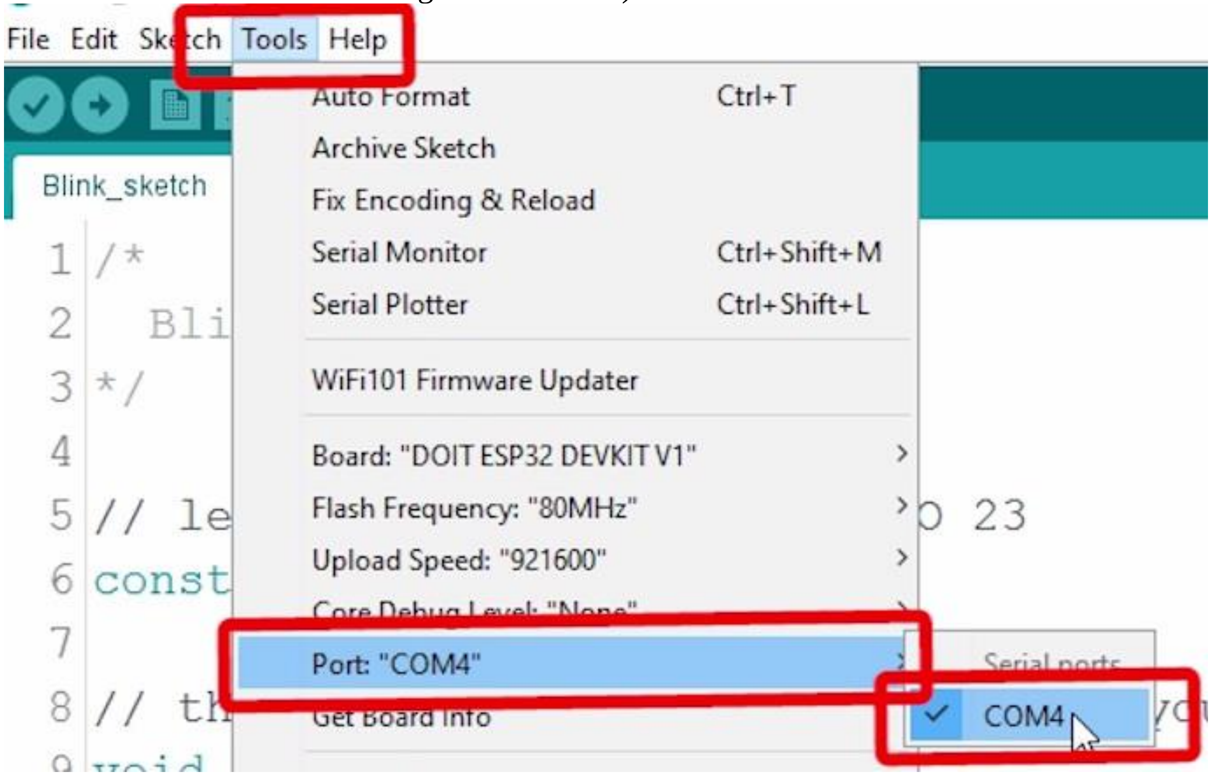
Testing the Installation:

Plug the ESP32 board to your computer. With your Arduino IDE open, follow these steps:
Step 1: Select your Board in Tools > Board menu (in my case it's the "ESP32 Dev Module")



Board: "ESP32 Dev Module"
Upload Speed: "921600"
CPU Frequency: "240MHz (WiFi/BT)"
Flash Frequency: "80MHz"
Flash Mode: "QIO"
Flash Size: "4MB (32Mb)"
Partition Scheme: "Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS)"
Core Debug Level: "None"
PSRAM: "Enabled"

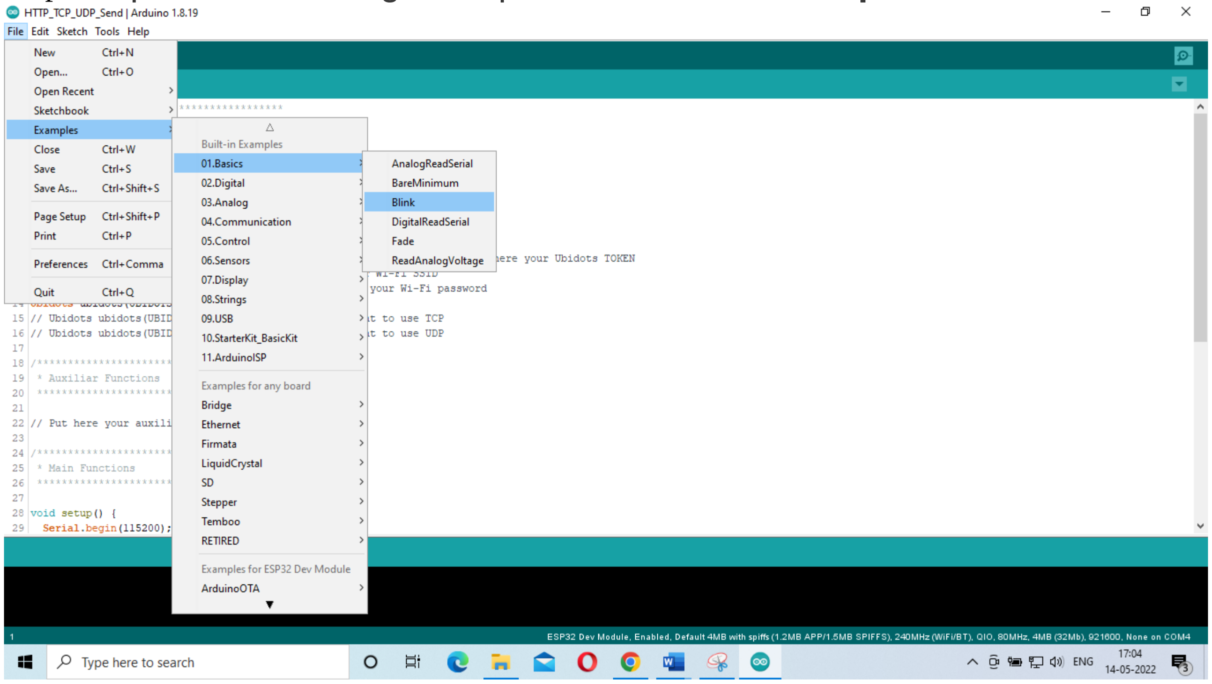
Step 2: Select the Port (if you don't see the COM Port in your Arduino IDE, you need to install the CP210x USB to UART Bridge VCP Drivers):



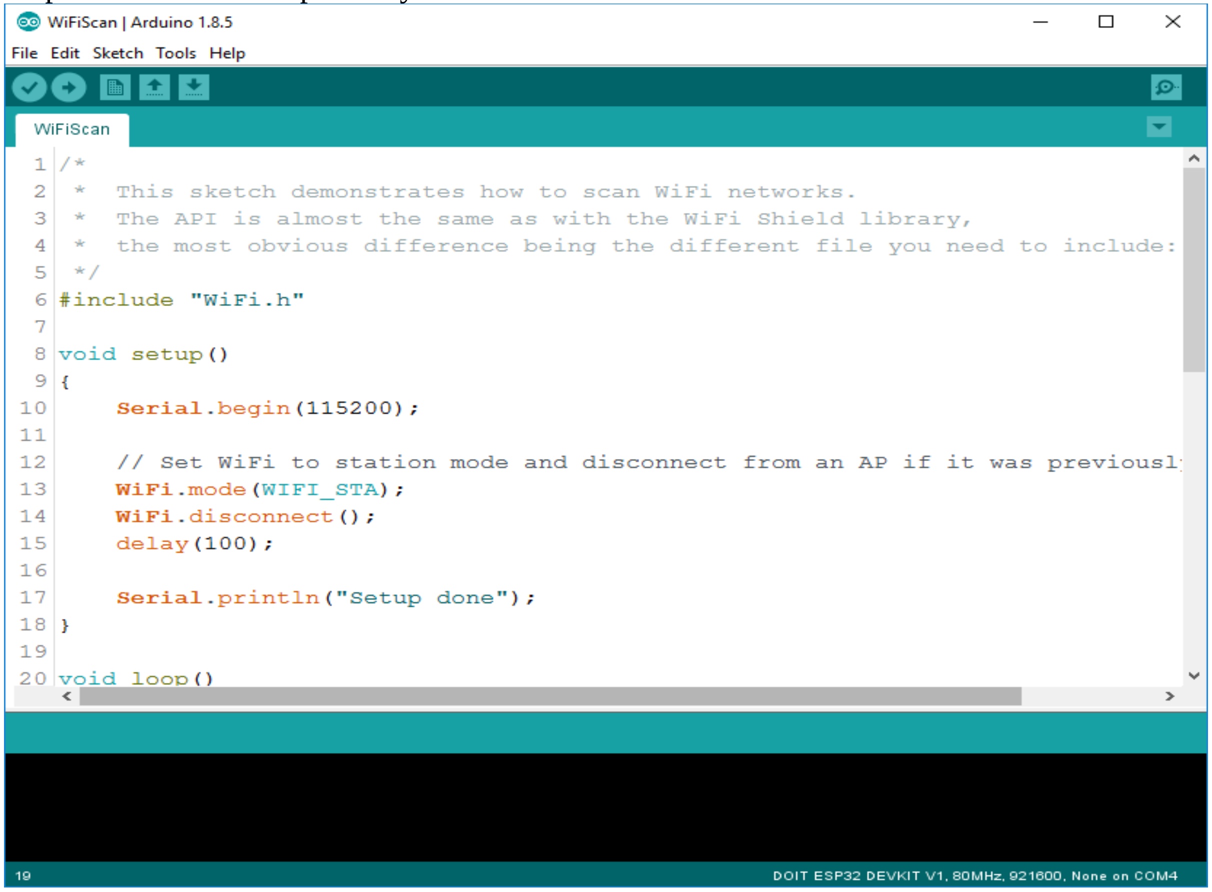
Board: "DOIT ESP32 DEVKIT V1"
Flash Frequency: "80MHz"
Upload Speed: "921600"
Core Debug Level: "None"
Port: "COM4"

[Type text]

Step 3: Open the following example under **File** > **Examples >Basics> Blink**



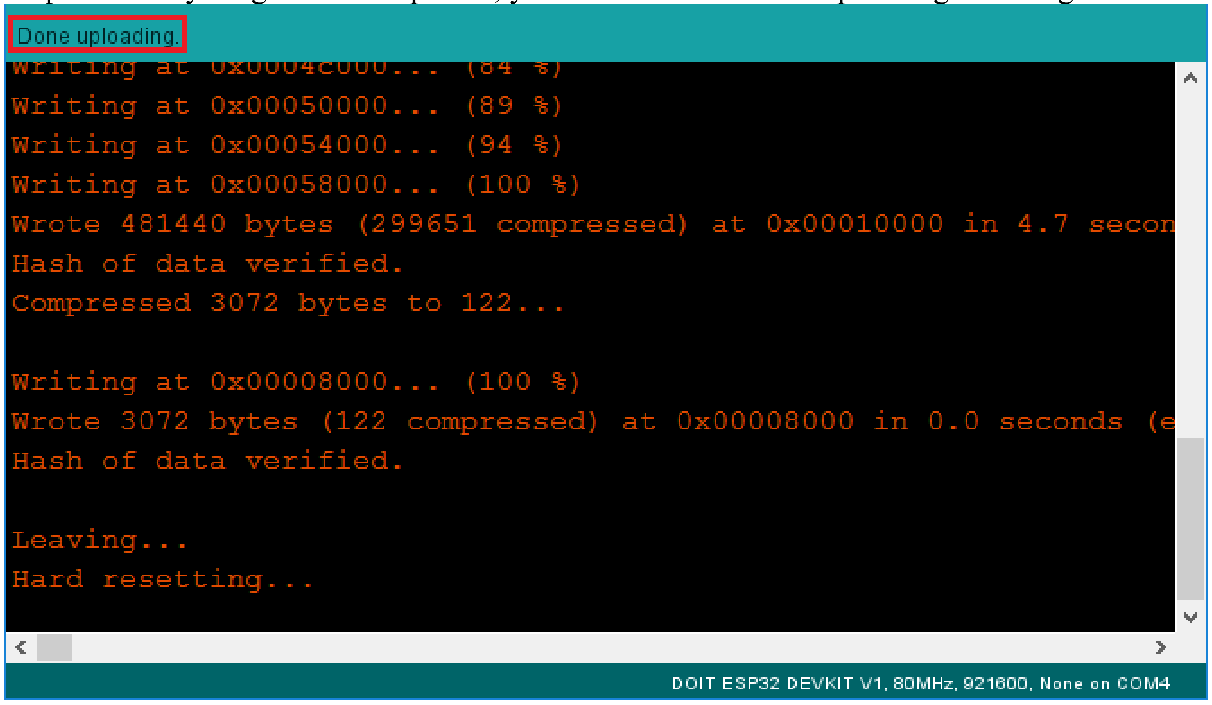
Step 4: A new sketch opens in your Arduino IDE:


WiFiScan | Arduino 1.8.5
File Edit Sketch Tools Help
WiFiScan
19
DOIT ESP32 DEVKIT V1, 80MHz, 921600, None on COM4

```
/*
 *  This sketch demonstrates how to scan WiFi networks.
 *  The API is almost the same as with the WiFi Shield library,
 *  the most obvious difference being the different file you need to include:
 */
#include "WiFi.h"

void setup()
{
    Serial.begin(115200);

    // Set WiFi to station mode and disconnect from an AP if it was previousl
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    delay(100);

    Serial.println("Setup done");
}

void loop()
```

Step 5: Press the Upload button in the Arduino IDE. Wait a few seconds while the code compiles and uploads to your board.

Step 6: If everything went as expected, you should see a "Done uploading." message.



Done uploading.

```
Writing at 0x0004c000... (84 %)
Writing at 0x00050000... (89 %)
Writing at 0x00054000... (94 %)
Writing at 0x00058000... (100 %)
Wrote 481440 bytes (299651 compressed) at 0x00010000 in 4.7 secon
Hash of data verified.
Compressed 3072 bytes to 122...

Writing at 0x00008000... (100 %)
Wrote 3072 bytes (122 compressed) at 0x00008000 in 0.0 seconds (e
Hash of data verified.

Leaving...
Hard resetting...
```

DOIT ESP32 DEVKIT V1, 80MHz, 921600, None on COM4

More details : https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/

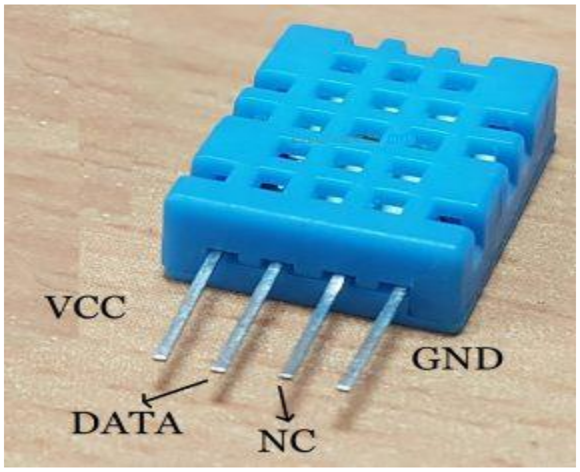
Experiment -1

Aim: To measure the Temperature, Humidity and Heat index using ESP32 Development Board and display the values in serial monitor.
APPARATUS:

S No	Name of the Equipment	Quantity
1	ESP32 Development Board	1
2	DHT11 or DHT22 temperature and humidity sensor	1
3	Jumper wires	3
4	Micro USB cable	1

The DHT11 and DHT22 sensors are used to measure temperature and relative humidity.

- Temperature range: 0 to 50ºC +/- 2ºC
- Relative humidity range: 20 to 90% +/-5%
- Temperature resolution: 1ºC
- Humidity resolution: 1%
- Operating voltage: 3 to 5.5 V DC
- Current supply: 0.5 to 2.5 mA
- Sampling period: 1 second



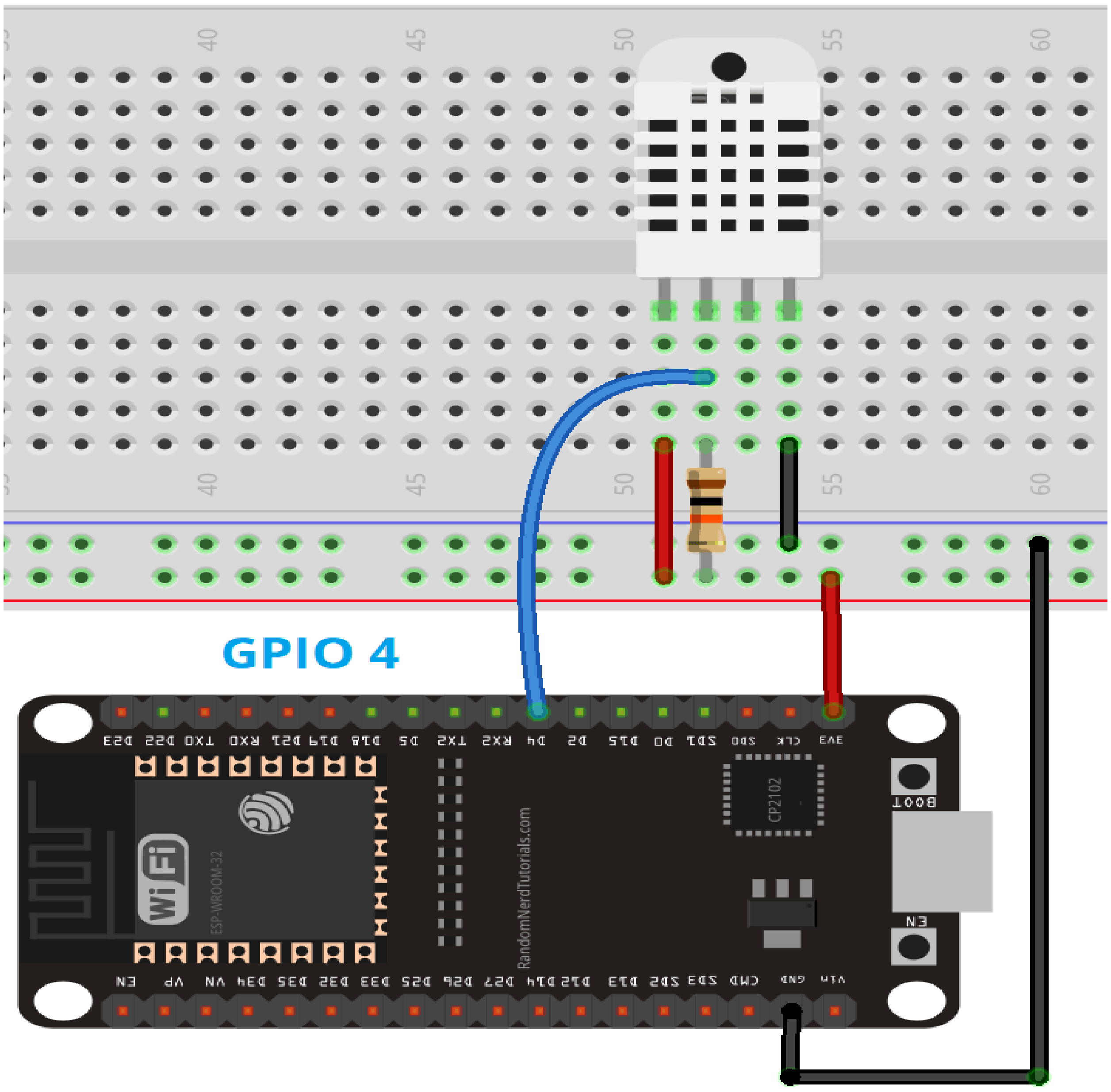


Figure.1 Circuit Diagram for measurement of Temperature and Humidity.

PROCEDURE:

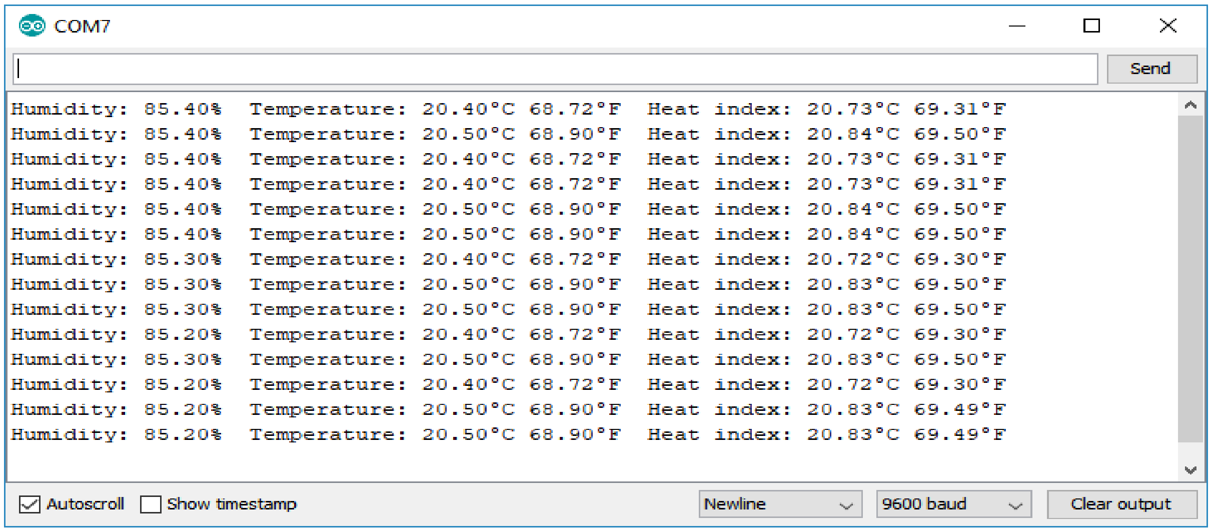
1. Plug the ESP32 development board to your PC
2. Make the connection as per the circuit diagram
3. Open the Arduino IDE in computer and write the program. Save the new sketch in your working directory

   Note: **Make sure you have the right board and COM port selected in your Arduino IDE settings**.
4. Compile the program and upload it to the ESP32 Development Board. If everything went as expected, you should see a “Done uploading” message. (You need to hold the ESP32 on-board Boot button while uploading).
5. After uploading the code, open the Serial Monitor at a baud rate of 115200.

CODE:

```
#include "DHT.h"

#define DHTPIN 4     // Digital pin connected to the DHT sensor
// Feather HUZZAH ESP8266 note: use pins 3, 4, 5, 12, 13 or 14 --
// Pin 15 can work but DHT must be disconnected during program upload.

// Uncomment whatever type you're using!
#define DHTTYPE DHT11   // DHT 11
//#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
//#define DHTTYPE DHT21   // DHT 21 (AM2301)

// Connect pin 1 (on the left) of the sensor to +5V
// NOTE: If using a board with 3.3V logic like an Arduino Due connect pin 1
// to 3.3V instead of 5V!
// Connect pin 2 of the sensor to whatever your DHTPIN is
// Connect pin 4 (on the right) of the sensor to GROUND
// Connect a 10K resistor from pin 2 (data) to pin 1 (power) of the sensor

// Initialize DHT sensor.
// Note that older versions of this library took an optional third parameter to
// tweak the timings for faster processors.  This parameter is no longer needed
// as the current DHT reading algorithm adjusts itself to work on faster procs.
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(115200);
  Serial.println(F("DHTxx test!"));

  dht.begin();
}

void loop() {
  // Wait a few seconds between measurements.
  delay(2000);

  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
  float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  float f = dht.readTemperature(true);

  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }
  // Compute heat index in Fahrenheit (the default)
  float hif = dht.computeHeatIndex(f, h);
  // Compute heat index in Celsius (isFahreheit = false)
  float hic = dht.computeHeatIndex(t, h, false);

  Serial.print(F("Humidity: "));
  Serial.print(h);
  Serial.print(F("%  Temperature: "));
  Serial.print(t);
  Serial.print(F("\xC2\xB0 C, "));
  Serial.print(f);
  Serial.print(F("\xC2\xB0 F,  Heat index: "));
  Serial.print(hic);
  Serial.print(F("\xC2\xB0 C, "));
  Serial.print(hif);
  Serial.println(F("\xC2\xB0 F."));
}
```

Result:



```
Humidity: 85.40%  Temperature: 20.40°C 68.72°F  Heat index: 20.73°C 69.31°F
Humidity: 85.40%  Temperature: 20.50°C 68.90°F  Heat index: 20.84°C 69.50°F
Humidity: 85.40%  Temperature: 20.40°C 68.72°F  Heat index: 20.73°C 69.31°F
Humidity: 85.40%  Temperature: 20.40°C 68.72°F  Heat index: 20.73°C 69.31°F
Humidity: 85.40%  Temperature: 20.50°C 68.90°F  Heat index: 20.84°C 69.50°F
Humidity: 85.40%  Temperature: 20.50°C 68.90°F  Heat index: 20.84°C 69.50°F
Humidity: 85.30%  Temperature: 20.40°C 68.72°F  Heat index: 20.72°C 69.30°F
Humidity: 85.30%  Temperature: 20.50°C 68.90°F  Heat index: 20.83°C 69.50°F
Humidity: 85.30%  Temperature: 20.50°C 68.90°F  Heat index: 20.83°C 69.50°F
Humidity: 85.20%  Temperature: 20.40°C 68.72°F  Heat index: 20.72°C 69.30°F
Humidity: 85.30%  Temperature: 20.50°C 68.90°F  Heat index: 20.83°C 69.50°F
Humidity: 85.20%  Temperature: 20.40°C 68.72°F  Heat index: 20.72°C 69.30°F
Humidity: 85.20%  Temperature: 20.50°C 68.90°F  Heat index: 20.83°C 69.49°F
Humidity: 85.20%  Temperature: 20.50°C 68.90°F  Heat index: 20.83°C 69.49°F
```

Experiment -2

2 a: Scan Wi-Fi networks and get Wi-Fi strength using ESP32 Development broad.

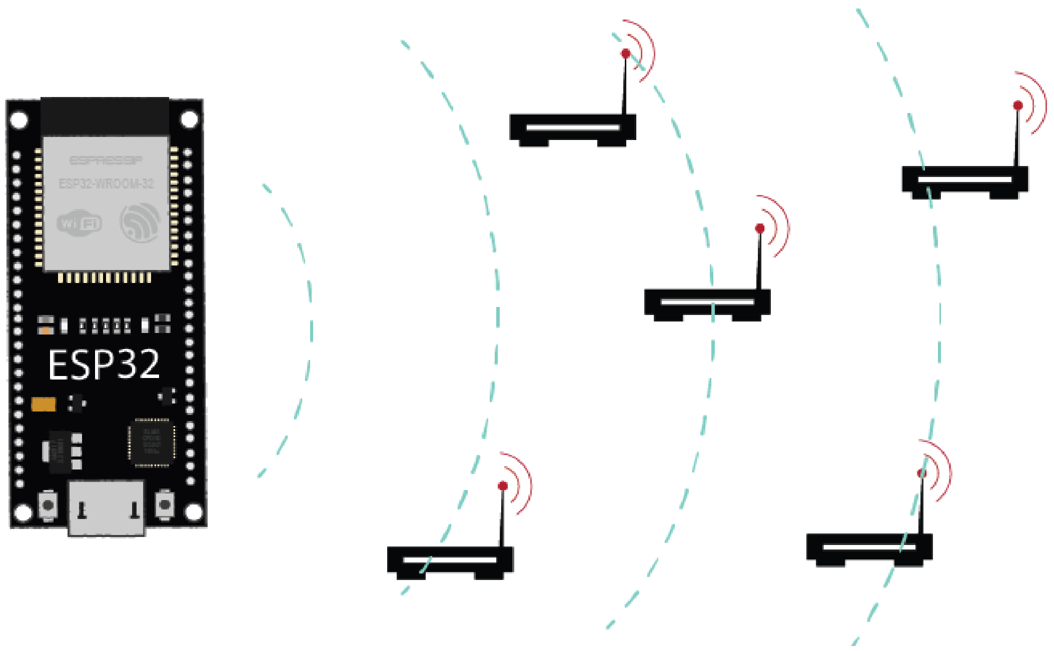
ESP32 Wi-Fi Modes:

The ESP32 board can act as Wi-Fi Station, Access Point or both. To set the Wi-Fi mode, use WiFi.mode() and set the desired mode as argument:

WiFi.mode(WIFI_STA)	**station mode:** the ESP32 connects to an access point
WiFi.mode(WIFI_AP)	**access point mode:** stations can connect to the ESP32
WiFi.mode(WIFI_STA_AP)	**access point and a station** connected to another access point

APPARATUS:

S No	Name of the Equipment	Quantity
1	ESP32 Development Board	1
2	Micro USB cable	1



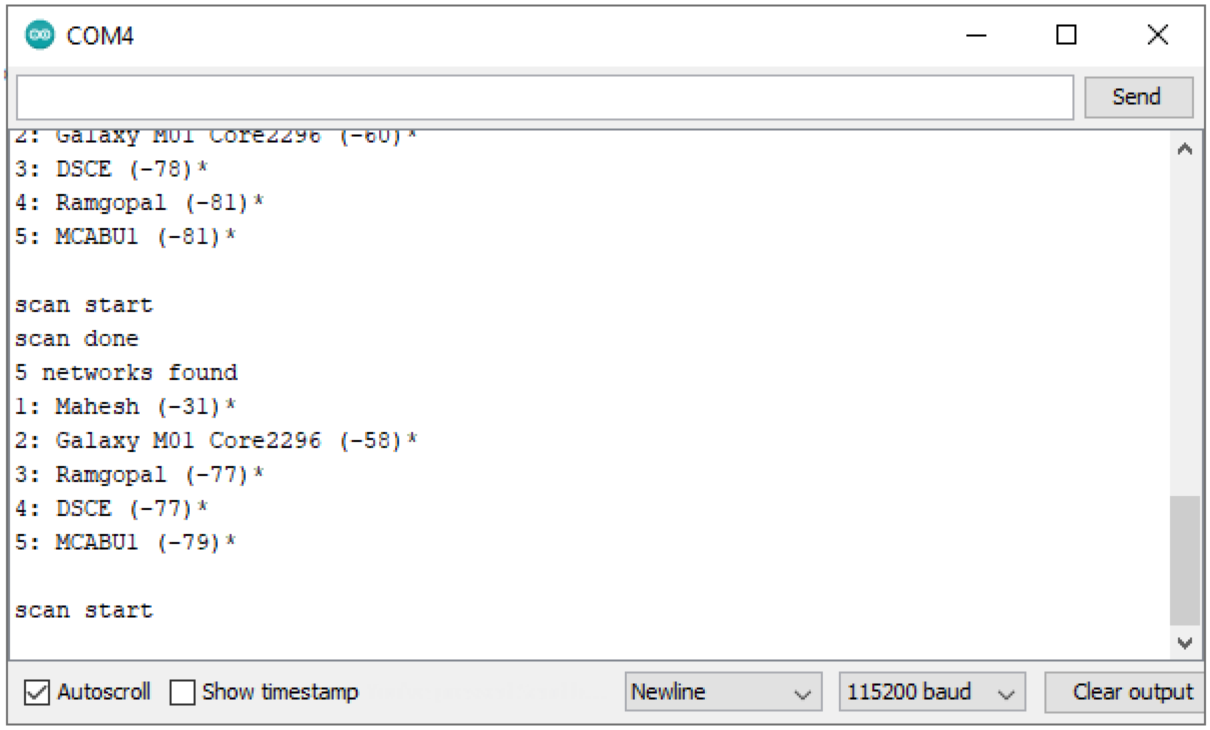
PROCEDURE:

1. Plug the ESP32 development board to your PC
2. Make the connection as per the circuit diagram
3. Open the Arduino IDE in computer and write the program. Save the new sketch in your working directory
   Note: **Make sure you have the right board and COM port selected in your Arduino IDE settings**.
4. Compile the program and upload it to the ESP32 Development Board. If everything went as expected, you should see a “Done uploading” message. (You need to hold the ESP32 on-board Boot button while uploading).
5. After uploading the code, open the Serial Monitor at a baud rate of 115200.

CODE:

```
#include "WiFi.h"

void setup()
{
   Serial.begin(115200);

   // Set WiFi to station mode and disconnect from an AP if it was previously connected
   WiFi.mode(WIFI_STA);
   WiFi.disconnect();
   delay(100);

   Serial.println("Setup done");
}
void loop()
{
   Serial.println("scan start");

   // WiFi.scanNetworks will return the number of networks found
   int n = WiFi.scanNetworks();
   Serial.println("scan done");
   if (n == 0) {
      Serial.println("no networks found");
   } else {
      Serial.print(n);
      Serial.println(" networks found");
      for (int i = 0; i < n; ++i) {
         // Print SSID and RSSI for each network found
         Serial.print(i + 1);
         Serial.print(": ");
         Serial.print(WiFi.SSID(i));
         Serial.print(" (");
         Serial.print(WiFi.RSSI(i));
         Serial.print(")");
         Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN)?" ":"*");
         delay(10);
      }
   }
   Serial.println("");

   // Wait a bit before scanning again
   delay(5000);
}
```

Result:

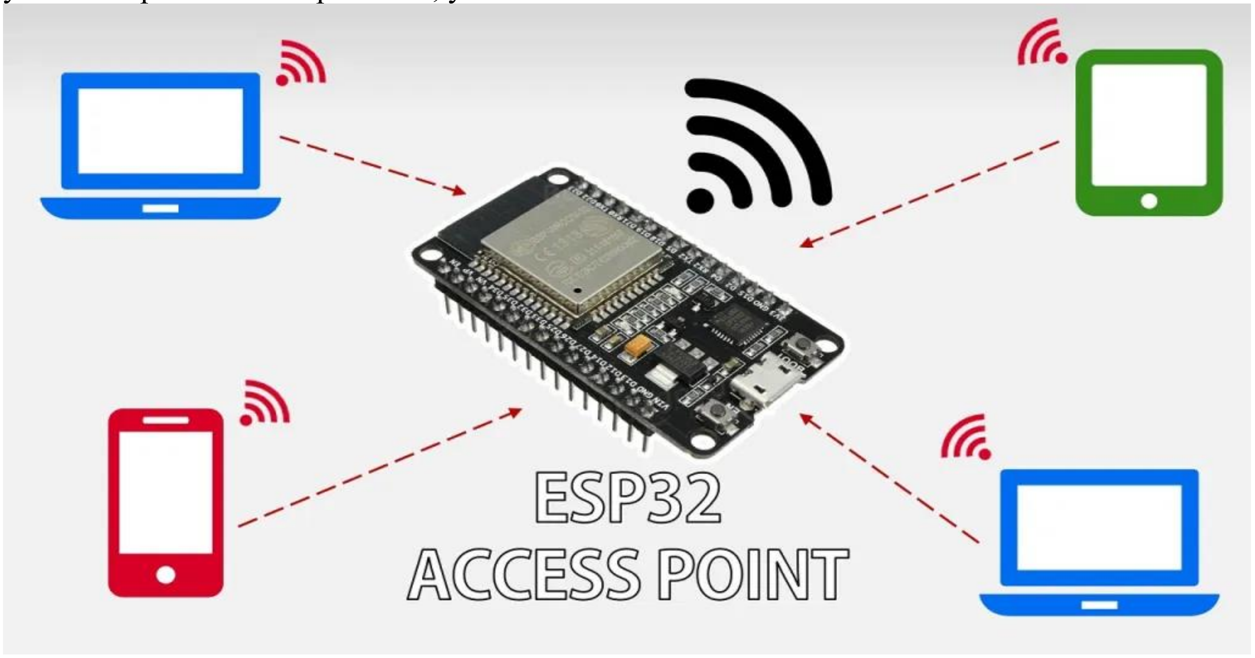


```
2: Galaxy M01 Core2296 (-60)*
3: DSCE (-78)*
4: Ramgopal (-81)*
5: MCABU1 (-81)*

scan start
scan done
5 networks found
1: Mahesh (-31)*
2: Galaxy M01 Core2296 (-58)*
3: Ramgopal (-77)*
4: DSCE (-77)*
5: MCABU1 (-79)*

scan start
```

☑ Autoscroll ☐ Show timestamp   Newline   115200 baud   Clear output

2 b: Configure/Set Your ESP32 Development broad as an access point

When you set your ESP32 board as an access point, you can be connected using any device with Wi-Fi capabilities without connecting to your router. When you set the ESP32 as an access point, you create its own Wi-Fi network, and nearby Wi-Fi devices (stations) can connect to it, like your smartphone or computer. So, you don't need to be connected to a router to control it.



PROCEDURE:

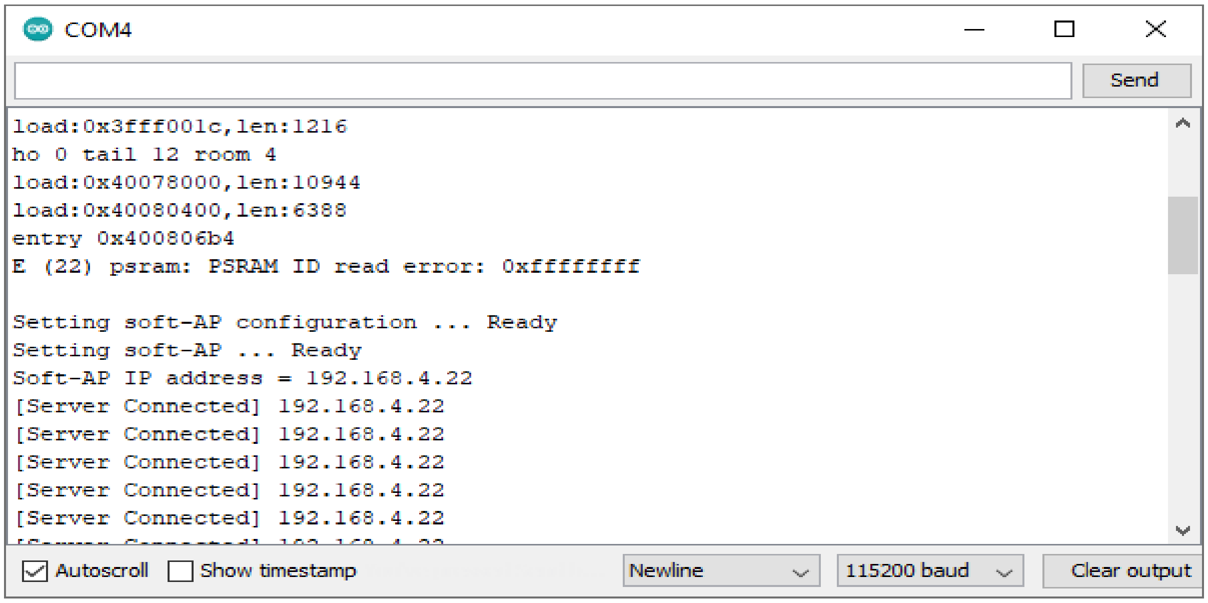
1. Plug the ESP32 development board to your PC

2. Open the Arduino IDE in computer and write the program. Save the new sketch in your working directory
   Note: **Make sure you have the right board and COM port selected in your Arduino IDE settings**.
3. Define a SSID name and a password to access the ESP32 development board
4. Compile the program and upload it to the ESP32 Development Board. If everything went as expected, you should see a “Done uploading” message. (You need to hold the ESP32 on-board Boot button while uploading).
5. After uploading the code, open the Serial Monitor at a baud rate of 115200.

CODE:

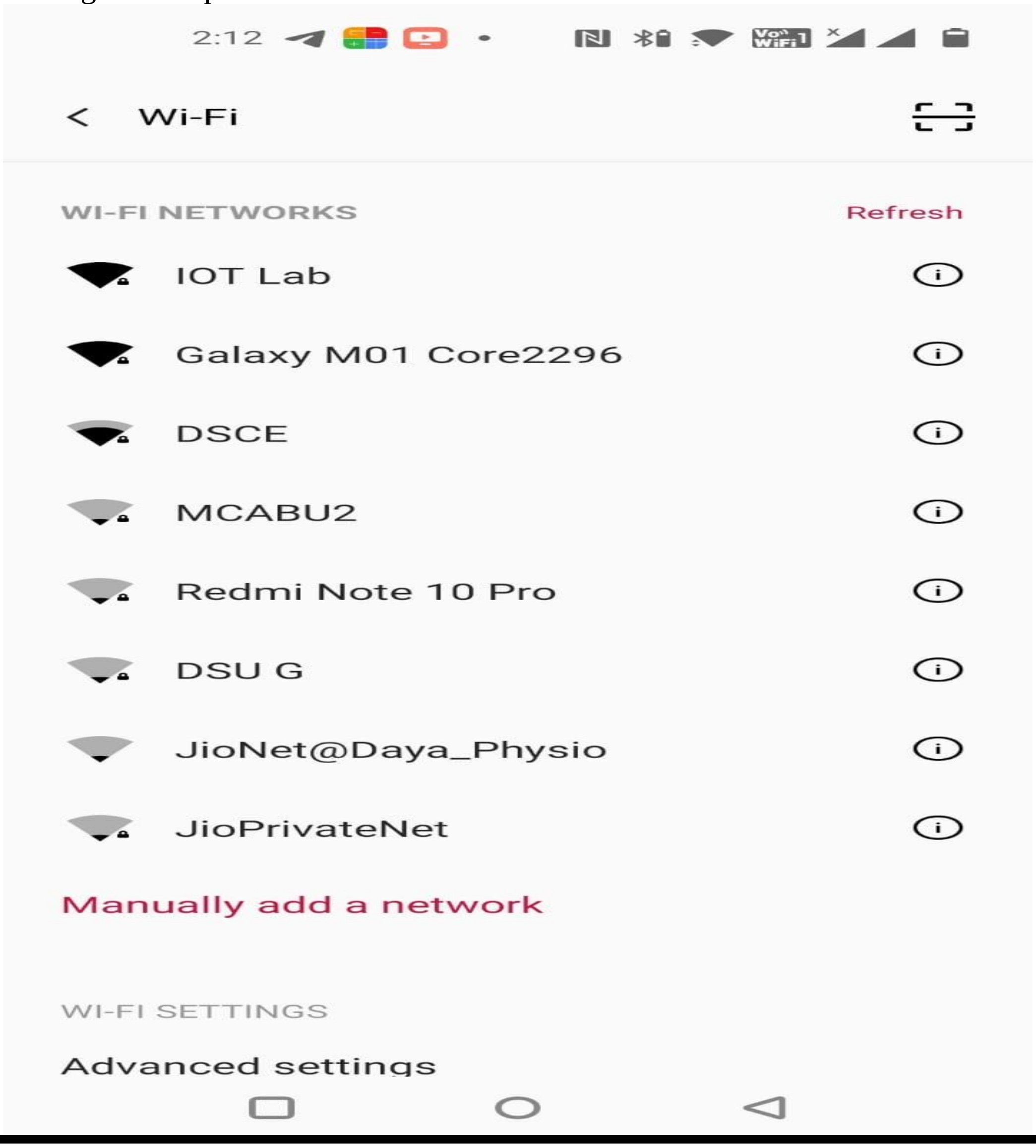
```
#include <WiFi.h>
const char *ssid = "SSID_Name_your_Choice";
const char *password = " Your_Choice(min 6 character) ";
IPAddress local_IP(192,168,4,22);
IPAddress gateway(192,168,4,9);
IPAddress subnet(255,255,255,0);
void setup()
{
 Serial.begin(115200);
 Serial.println();
 Serial.print("Setting soft-AP configuration ... ");
 Serial.println(WiFi.softAPConfig(local_IP, gateway, subnet) ? "Ready" : "Failed!");
 Serial.print("Setting soft-AP ... ");
 Serial.println(WiFi.softAP(ssid,password) ? "Ready" : "Failed!");
 //WiFi.softAP(ssid);
 //WiFi.softAP(ssid, password, channel, ssdi_hidden, max_connection)

 Serial.print("Soft-AP IP address = ");
 Serial.println(WiFi.softAPIP());
}
void loop() {
 Serial.print("[Server Connected] ");
 Serial.println(WiFi.softAPIP());
 delay(500);
}
```

Result:



```
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:10944
load:0x40080400,len:6388
entry 0x400806b4
E (22) psram: PSRAM ID read error: 0xffffffff

Setting soft-AP configuration ... Ready
Setting soft-AP ... Ready
Soft-AP IP address = 192.168.4.22
[Server Connected] 192.168.4.22
[Server Connected] 192.168.4.22
[Server Connected] 192.168.4.22
[Server Connected] 192.168.4.22
[Server Connected] 192.168.4.22
```

Through Smart phone:



2 c: Establish connection between IoT Node and Access point and Display of local IP and Gateway IP

The goal of this experiment is to understand the configuration of IoT Node to connect to an Access Point. Any device when connected to a network is assigned an IP address, which is a unique number for each device on a network. We shall also see how to identify the IP network of our IoT Node. The Access Point also has an IP address, known as the Gateway IP. These concepts are fundamentals to understand the architecture of an IoT Network.

Smartphone generates the WIFI network, acts like an Access Point and is assigned a Gateway IP

ESP32 acts as IOT Node, connects to a WIFI network and is assigned a Local IP Address

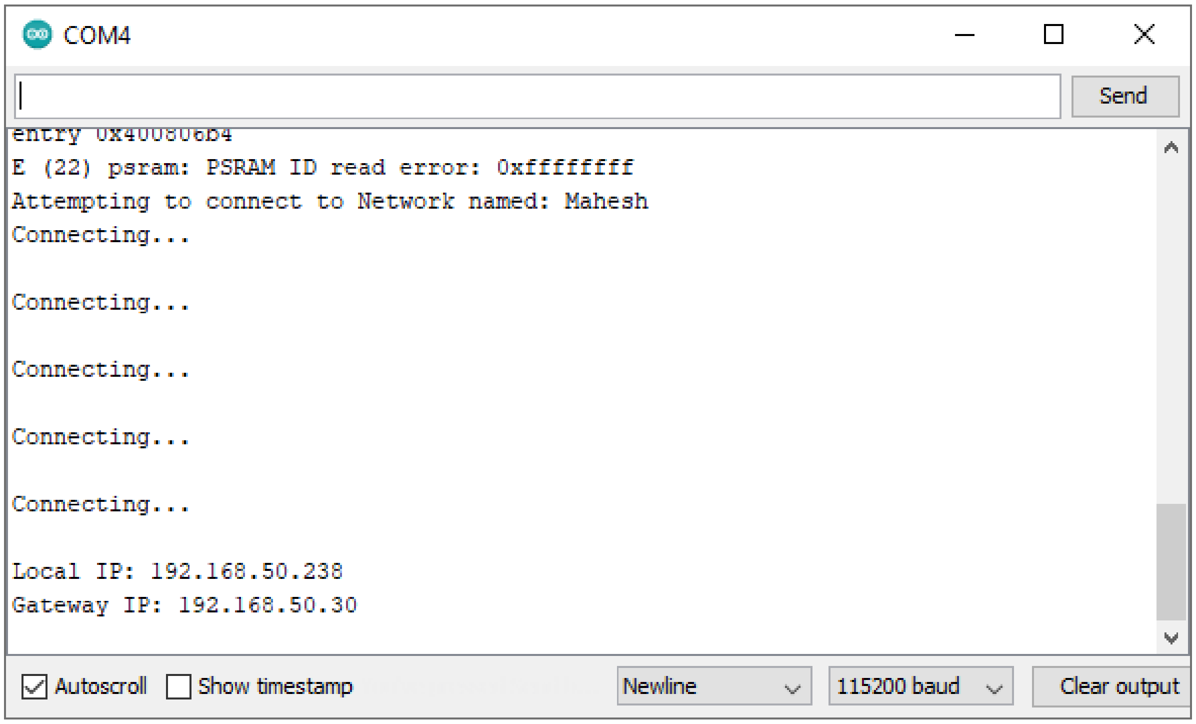
PROCEDURE:

1. Plug the ESP32 development board to your PC
2. Open the Arduino IDE in computer and write the program. Save the new sketch in your working directory
   Note: **Make sure you have the right board and COM port selected in your Arduino IDE settings**.
3. Define a SSID name and a password to access the ESP32 development board
4. Compile the program and upload it to the ESP32 Development Board. If everything went as expected, you should see a “Done uploading” message. (You need to hold the ESP32 on-board Boot button while uploading).
5. After uploading the code, open the Serial Monitor at a baud rate of 115200.

CODE:

```
#include <WiFi.h>

char ssid[]="REPLACE_WITH_YOUR_SSID";
char password[]="REPLACE_WITH_YOUR_PASSWORD";

IPAddress ip;
IPAddress gateway;

void setup()
{
  Serial.begin(115200); //Initialize Serial Port
  //attempt to connect to wifi
  Serial.print("Attempting to connect to Network named: ");
  // print the network name (SSID);
  Serial.println(ssid);

  //Connect to WiFI
  WiFi.begin(ssid, password);

  //Wait untill wifi is connected
  while ( WiFi.status() != WL_CONNECTED)
  {
    // print dots while we wait to connect
    Serial.print(".");
    delay(300);
  }

  //If you are connected print the IP Address
  Serial.println("\nYou're connected to the network");

  //wait untill you get an IP address
  while (WiFi.localIP() = INADDR_NONE) {
  // print dots while we wait for an ip addresss
  Serial.print(".");
  delay(300);
  }

  ip=WiFi.localIP();
  Serial.println(ip);
  gateway=WiFi.gatewayIP();
  Serial.println("GATEWAY IP:");
  Serial.println(gateway);
}

void loop()
{
  // put your main code here, to run repeatedly:

}
```

Result:



COM4

```
entry 0x400806b4
E (22) psram: PSRAM ID read error: 0xffffffff
Attempting to connect to Network named: Mahesh
Connecting...

Connecting...

Connecting...

Connecting...

Connecting...

Local IP: 192.168.50.238
Gateway IP: 192.168.50.30
```

☑ Autoscroll ☐ Show timestamp   Newline   115200 baud

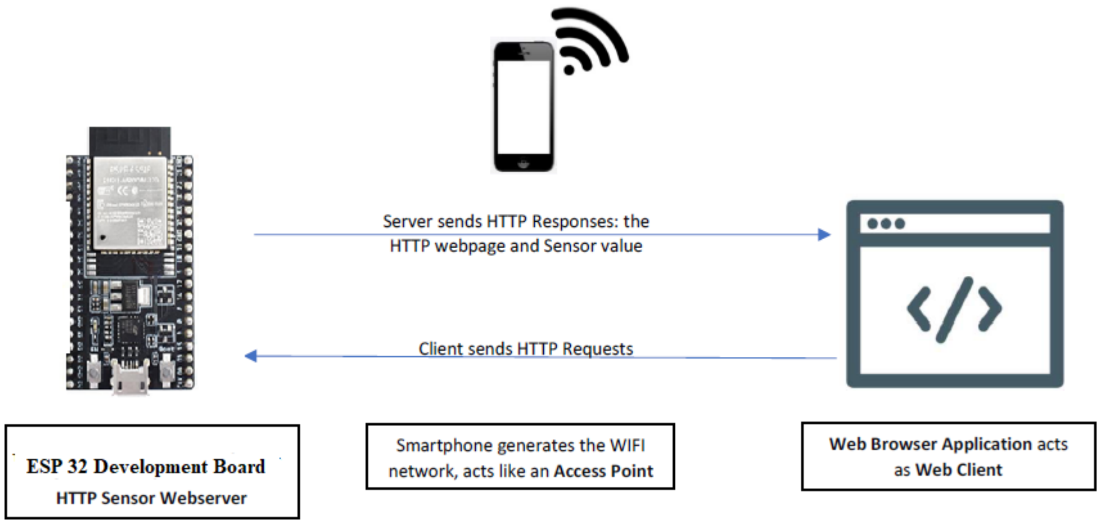
Experiment -3

3 a) Design and Implementation of HTTP based IoT Web Server to control the status of LED

APPARATUS:

S No	Name of the Equipment	Quantity
1	ESP32 Development Board	1
2	DHT11 or DHT22 temperature and humidity sensor	1
3	Jumper wires	3
4	Micro USB cable	1

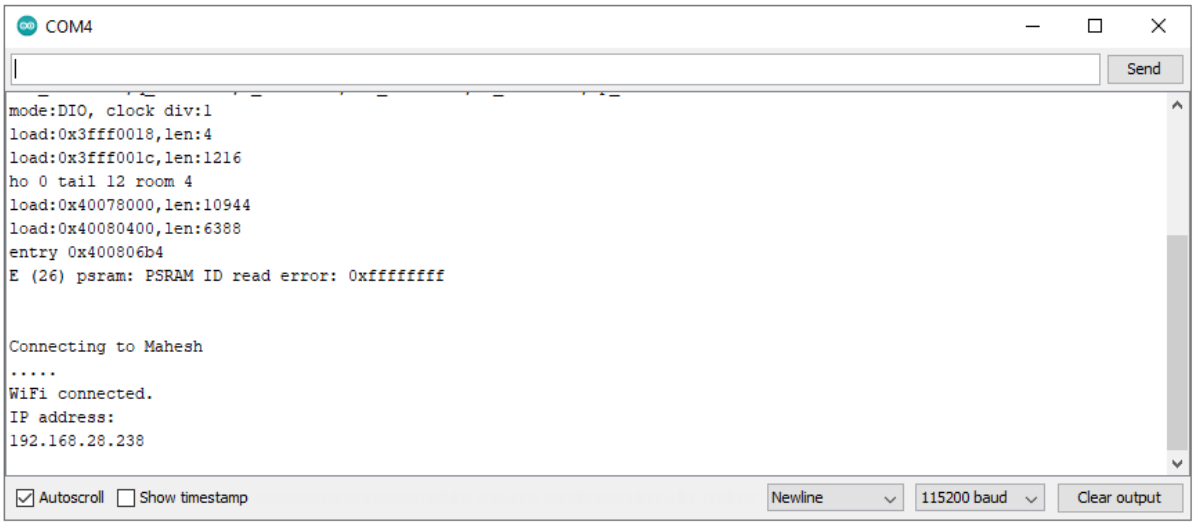
An HTTP based webserver provides access to data to an HTTP client such as web browser. In IoT applications, the IoT Nodes are connected to sensors and actuators. The sensor data can be accessed using a web browser and also the actuators can be controlled from the web browser. This facilitates a wide variety of applications in IoT domain. The underlying protocol used for this communication between a Client and a Server is HTTP. The goal of this lab is to understand the configuration and working principle of an IoT web server. Using a browser, such as google chrome, we will send some HTTP based commands to the web server. Based on the type of command, the IoT Node web server will toggle the LEDs.


Server sends HTTP Responses: the HTTP webpage and Sensor value
Client sends HTTP Requests
ESP 32 Development Board
HTTP Sensor Webserver
Smartphone generates the WIFI network, acts like an Access Point
Web Browser Application acts as Web Client

CODE:

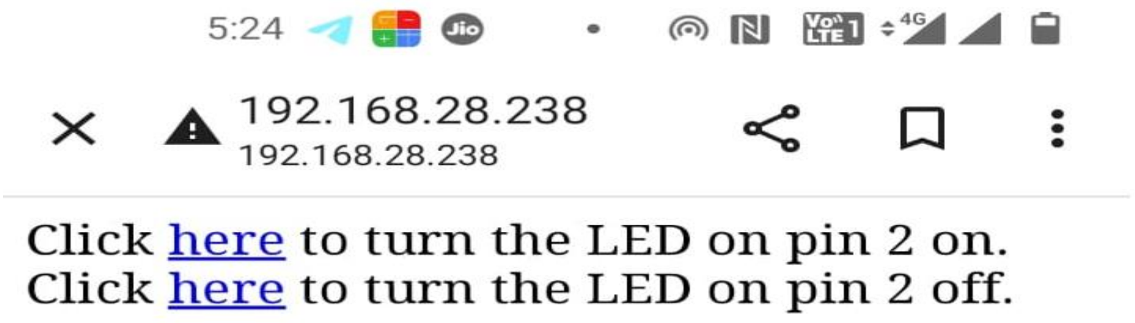
```
#include <WiFi.h>

const char* ssid     = "Mahesh";
const char* password = "012345678";

WiFiServer server(80);

void setup()
{
    Serial.begin(115200);
    pinMode(2, OUTPUT);      // set the LED pin mode

    delay(10);

    // We start by connecting to a WiFi network

    Serial.println();
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.println("WiFi connected.");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());

    server.begin();

}

int value = 0;

void loop(){
 WiFiClient client = server.available();   // listen for incoming clients

  if (client) {                             // if you get a client,
    Serial.println("New Client.");           // print a message out the serial port
    String currentLine = "";                // make a String to hold incoming data from the client
    while (client.connected()) {            // loop while the client's connected
      if (client.available()) {             // if there's bytes to read from the client,
        char c = client.read();             // read a byte, then
        Serial.write(c);                    // print it out the serial monitor
        if (c == '\n') {                    // if the byte is a newline character

          // if the current line is blank, you got two newline characters in a row.
          // that's the end of the client HTTP request, so send a response:
```

```
          if (currentLine.length() == 0) {
            // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
            // and a content-type so the client knows what's coming, then a blank line:
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println();

            // the content of the HTTP response follows the header:
            client.print("Click <a href=\"/H\">here</a> to turn the LED on pin 2 on.<br>");
            client.print("Click <a href=\"/L\">here</a> to turn the LED on pin 2 off.<br>");

            // The HTTP response ends with another blank line:
            client.println();
            // break out of the while loop:
            break;
          } else {    // if you got a newline, then clear currentLine:
            currentLine = "";
          }
        } else if (c != '\r') {  // if you got anything else but a carriage return character,
          currentLine += c;      // add it to the end of the currentLine
        }

        // Check to see if the client request was "GET /H" or "GET /L":
        if (currentLine.endsWith("GET /H")) {
          digitalWrite(2, HIGH);               // GET /H turns the LED on
        }
        if (currentLine.endsWith("GET /L")) {
          digitalWrite(2, LOW);                // GET /L turns the LED off
        }
      }
    }
    // close the connection:
    client.stop();
    Serial.println("Client Disconnected.");
  }
}
```



```
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:10944
load:0x40080400,len:6388
entry 0x400806b4
E (26) psram: PSRAM ID read error: 0xffffffff


Connecting to Mahesh
.....
WiFi connected.
IP address:
192.168.28.238
```

On Web browser/Web Client:



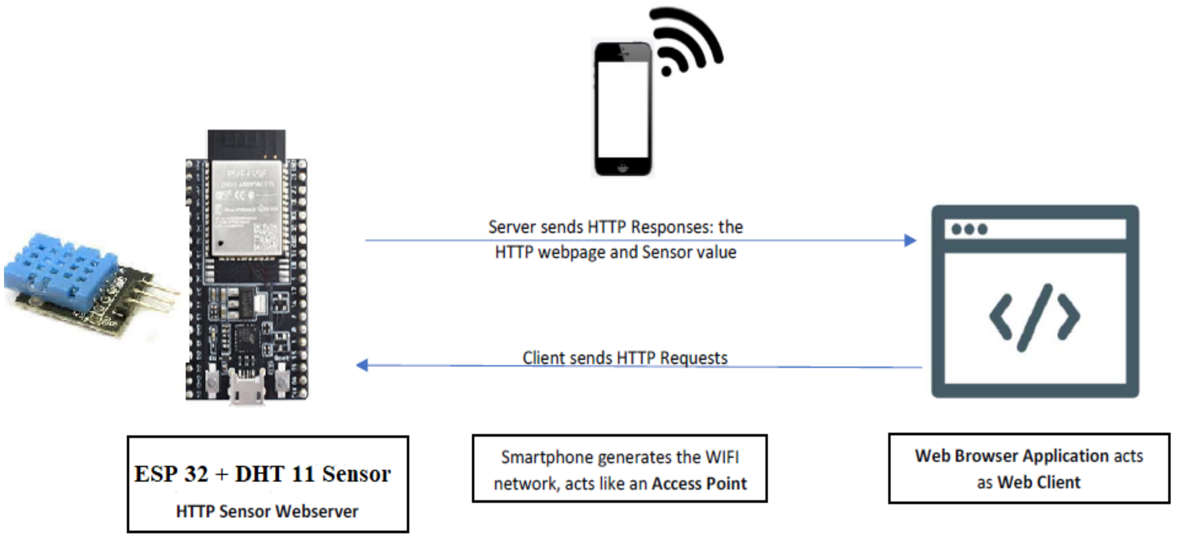
5:24

192.168.28.238
192.168.28.238

Click here to turn the LED on pin 2 on.
Click here to turn the LED on pin 2 off.

3 b) Design and Implementation of HTTP based IoT Web Server to display sensor value

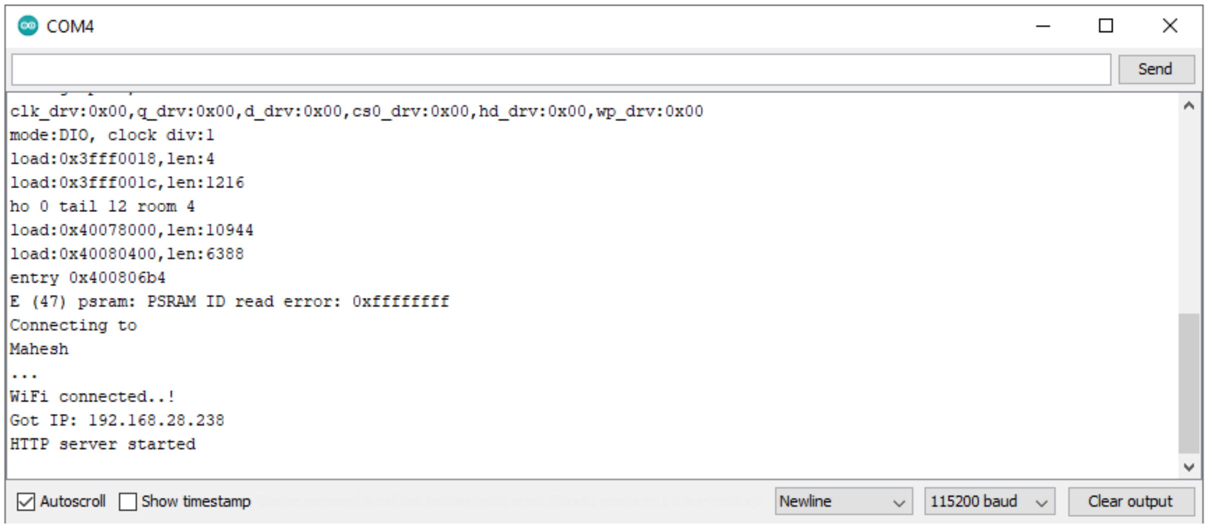
The goal of this experiment is to integrate the HTTP webserver with Sensor. As we know, most IoT Nodes are equipped with sensors and the IoT systems should provide these sensor data to users. In this experiment, we used DHT11 Temperature and Humidity sensor connected to an IoT Node/ ESP32 development board. The IoT Node is configured as a Webserver and the sensor data can be accessed via a web browser.


Server sends HTTP Responses: the HTTP webpage and Sensor value
Client sends HTTP Requests
ESP 32 + DHT 11 Sensor
HTTP Sensor Webserver
Smartphone generates the WIFI network, acts like an Access Point
Web Browser Application acts as Web Client

CODE:

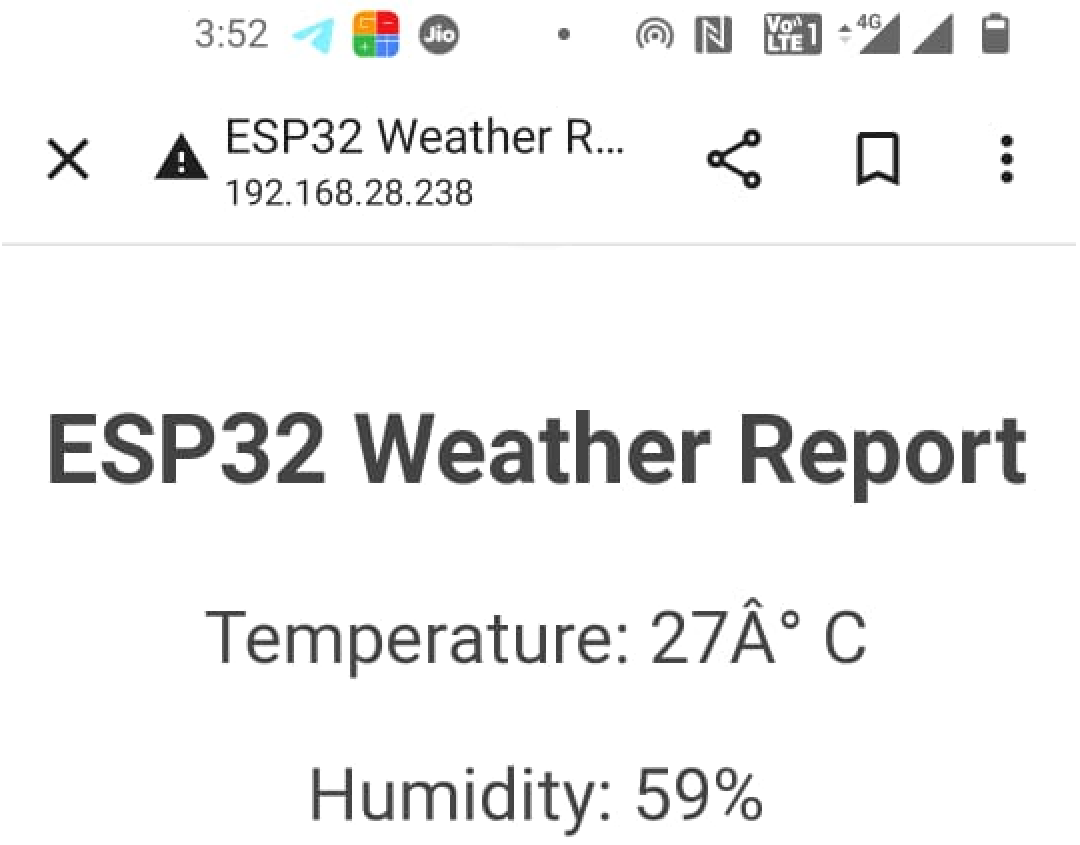
```
#include <WiFi.h>
#include <WebServer.h>
#include "DHT.h"

// Uncomment one of the lines below for whatever DHT sensor type you're using!
#define DHTTYPE DHT11   // DHT 11
//#define DHTTYPE DHT21   // DHT 21 (AM2301)
//#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321

/*Put your SSID & Password*/
const char* ssid = " REPLACE_WITH_YOUR_SSID ";      // Enter SSID here
const char* password = " REPLACE_WITH_YOUR_PASSWORD";      //Enter Password here

WebServer server(80);

// DHT Sensor
uint8_t DHTPin = 4;

// Initialize DHT sensor.
DHT dht(DHTPin, DHTTYPE);

float Temperature;
float Humidity;

void setup() {
  Serial.begin(115200);
  delay(100);

  pinMode(DHTPin, INPUT);

  dht.begin();

  Serial.println("Connecting to ");
  Serial.println(ssid);

  //connect to your local wi-fi network
  WiFi.begin(ssid, password);

  //check wi-fi is connected to wi-fi network
  while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected..!");
  Serial.print("Got IP: ");  Serial.println(WiFi.localIP());
server.on("/", handle_OnConnect);
  server.onNotFound(handle_NotFound);

  server.begin();
  Serial.println("HTTP server started");

}
void loop() {

  server.handleClient();

}

void handle_OnConnect() {

 Temperature = dht.readTemperature(); // Gets the values of the temperature
  Humidity = dht.readHumidity(); // Gets the values of the humidity
  server.send(200, "text/html", SendHTML(Temperature,Humidity));
}

void handle_NotFound(){
  server.send(404, "text/plain", "Not found");
}

String SendHTML(float Temperaturestat,float Humiditystat){
  String ptr = "<!DOCTYPE html> <html>\n";
  ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-
scalable=no\">\n";
  ptr +="<title>ESP32 Weather Report</title>\n";
  ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align:
center;}\n";
  ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;}\n";
  ptr +="p {font-size: 24px;color: #444444;margin-bottom: 10px;}\n";
  ptr +="</style>\n";
  ptr +="</head>\n";
  ptr +="<body>\n";
  ptr +="<div id=\"webpage\">\n";
  ptr +="<h1>ESP32 Weather Report</h1>\n";
   ptr +="<p>Temperature: ";
  ptr +=(int)Temperaturestat;
  ptr +="\xC2\xB0 C</p>";
  ptr +="<p>Humidity: ";
  ptr +=(int)Humiditystat;
  ptr +="%</p>";

  ptr +="</div>\n";
  ptr +="</body>\n";
  ptr +="</html>\n";
  return ptr;
}
```

RESULT:



```
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:10944
load:0x40080400,len:6388
entry 0x400806b4
E (47) psram: PSRAM ID read error: 0xffffffff
Connecting to
Mahesh
...
WiFi connected..!
Got IP: 192.168.28.238
HTTP server started
```

On Web browser/Web Client:



ESP32 Weather Report

Temperature: 27Â° C

Humidity: 59%

Experiment -4

Design and Implementation of MQTT based Controlling and Monitoring Using Ubidots MQTT Server.

Theory:

MQTT protocol

MQTT stands for Message Queuing Telemetry Transport. MQTT is a machine-to-machine internet of things connectivity protocol. It is an extremely lightweight and publish-subscribe messaging transport protocol. This protocol is useful for the connection with the remote location where the bandwidth is a premium. These characteristics make it useful in various situations, including constant environment such as for communication machine to machine and internet of things contexts. It is a publish and subscribe system where we can publish and receive the messages as a client. It makes it easy for communication between multiple devices. It is a simple messaging protocol designed for the constrained devices and with low bandwidth, so it's a perfect solution for the internet of things applications. Some of the features of an MQTT are given below:

- It is a machine-to-machine protocol, i.e., it provides communication between the devices.
- It is designed as a simple and lightweight messaging protocol that uses a publish/subscribe system to exchange the information between the client and the server.
- It does not require that both the client and the server establish a connection at the same time.
- It provides faster data transmission, like how WhatsApp/messenger provides a faster delivery. It's a real-time messaging protocol.
- It allows the clients to subscribe to the narrow selection of topics so that they can receive the information they are looking for.

MQTT Architecture:

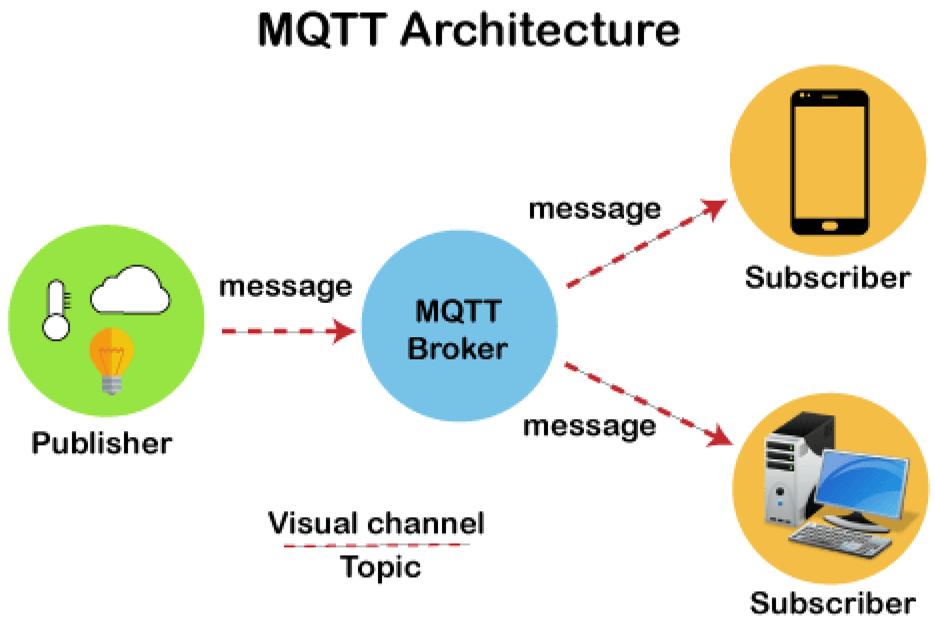
To understand the MQTT architecture, we first look at the components of the MQTT.

- Message
- Client
- Server or Broker
- TOPIC

- **Message:** Message: The message is the data that is carried out by the protocol across the network for the application. When the message is transmitted over the network, then the message contains the following parameters: Payload data, Quality of Service (QoS), Collection of Properties, Topic Name

- **Client:** In MQTT, the subscriber and publisher are the two roles of a client. The clients subscribe to the topics to publish and receive messages. In MQTT, the client performs two operations:
    1. **Publish:** When the client sends the data to the server, then we call this operation as a publish.

    2. **Subscribe:** When the client receives the data from the server, then we call this operation a subscription.

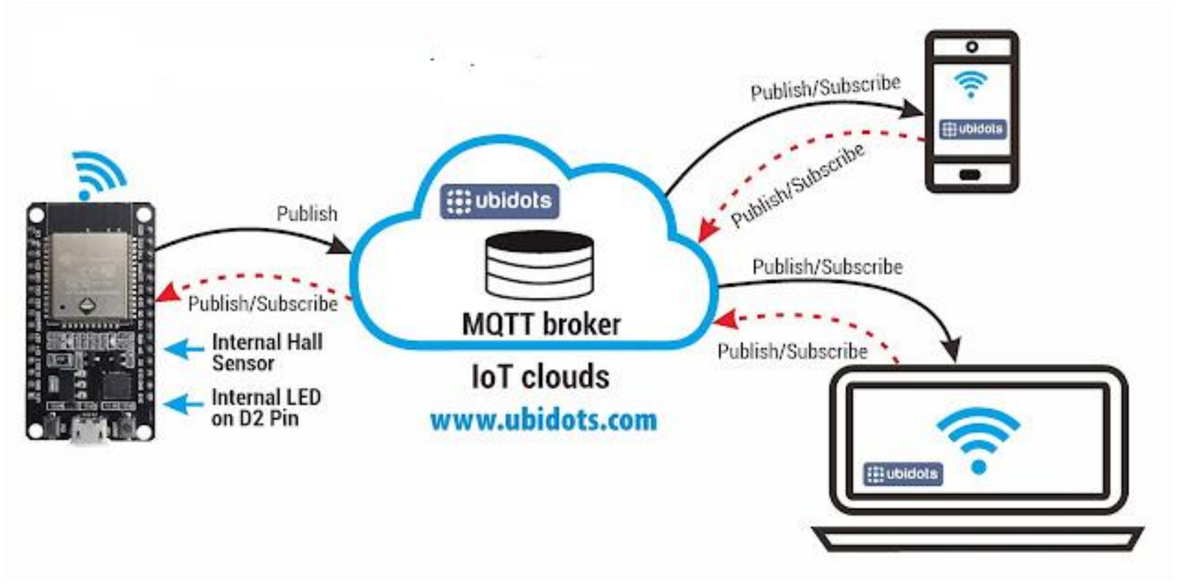


For more details: https://www.javatpoint.com/mqtt-protocol

Design and Implementation of MQTT based Controlling and Monitoring Using Ubidots MQTT Server.

APPARATUS:

S No	Name of the Equipment	Quantity
1	ESP32 Development Board	1
2	LED	1
3	Jumper wires	2
4	Micro USB cable	1



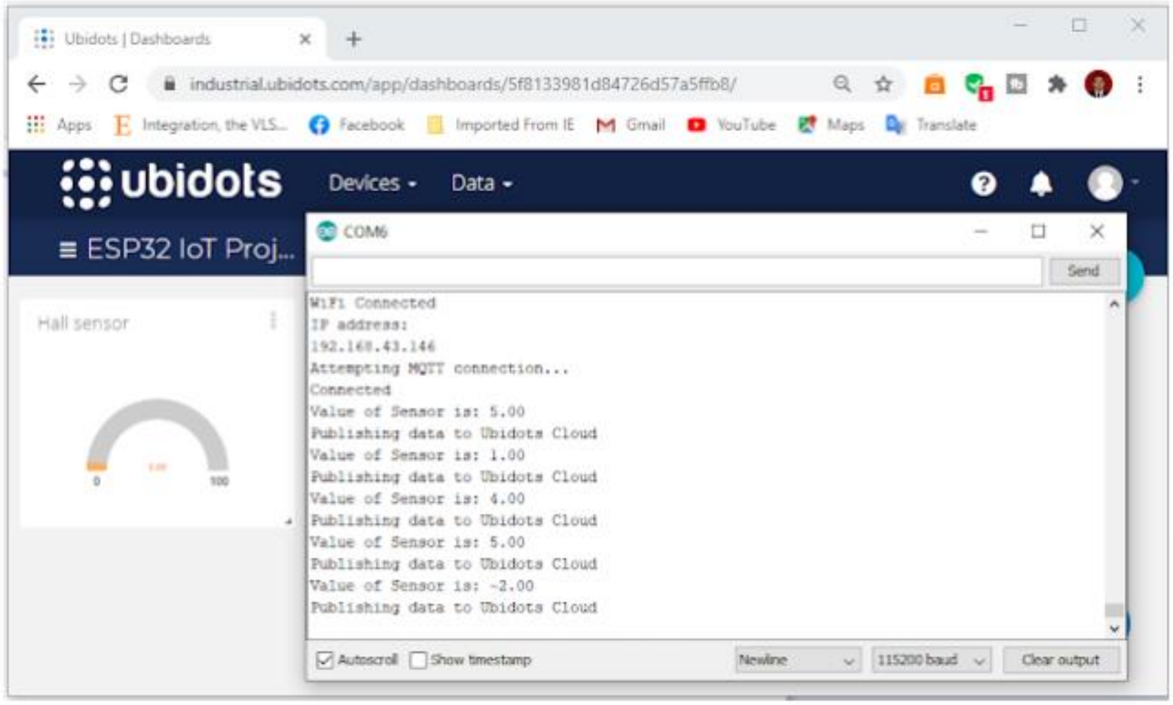
Create Account on Ubidots Website:

1. https://www.robotics-university.com/2019/05/internet-of-things-monitoring-sensor-and-controlling-led-on-ubidots-dashboard-using-mqtt-protocol.html
2. https://ubidots.com/
3. https://help.ubidots.com/en/articles/854333-ubidots-basics-devices-variables-dashboards-and-alerts

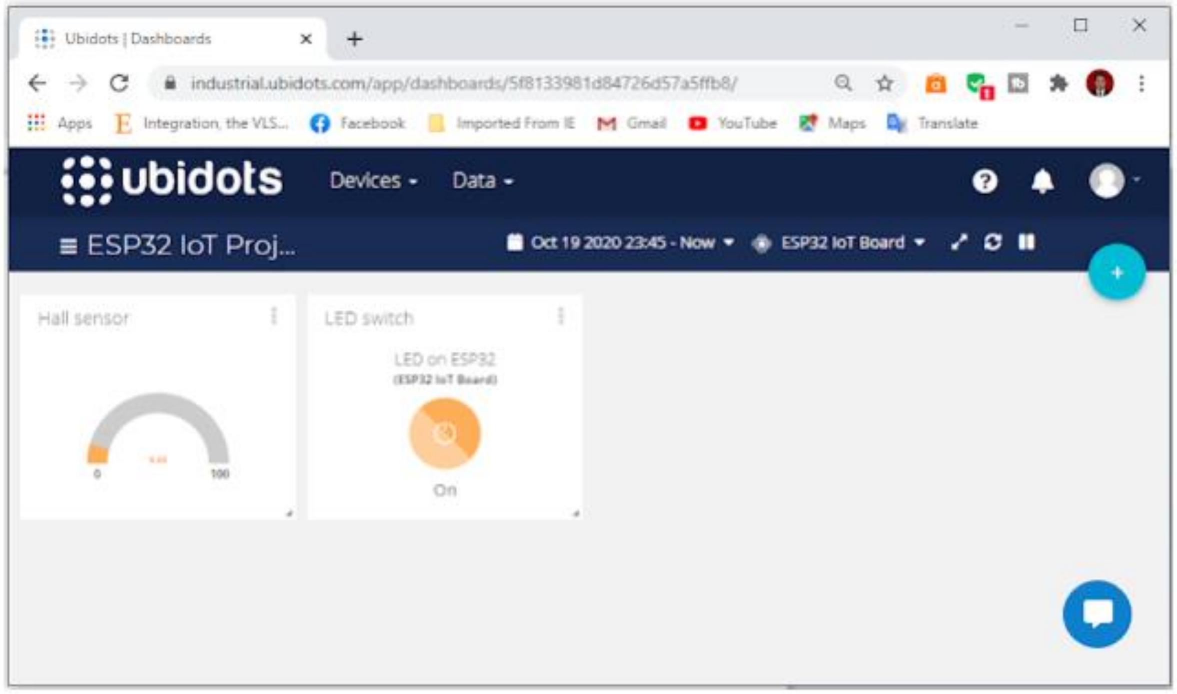
CODE:

```
#include <WiFi.h>
#include <PubSubClient.h>

#define WIFISSID " Replace_With_Your_Ssid " // Put your WifiSSID here
#define PASSWORD " Replace_With_Your_Passward" // Put your wifi password here
#define TOKEN "YOUR_TOKEN" // Put your Ubidots' TOKEN
#define MQTT_CLIENT_NAME "ESP32" // MQTT client Name, please enter your own 8-12
alphanumeric character ASCII string;
                                    //it should be a random and unique ascii string and different from all
other devices

/****************************************
 * Define Constants
 ****************************************/
#define VARIABLE_LABEL "Variable Name 1" // Assing the variable label
#define VARIABLE_LABEL_SUBSCRIBE " Variable Name 2" // Assing the variable label
#define DEVICE_LABEL "Device Name" // Assig the device label

#define led 26 // Set the GPIO26 as LED

char mqttBroker[]  = "things.ubidots.com";
char payload[100];
char topic[150];
char topicSubscribe[100];
// Space to store values to send
char str_sensor[10];

/****************************************
 * Auxiliar Functions
 ****************************************/
WiFiClient ubidots;
PubSubClient client(ubidots);

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.println("Attempting MQTT connection...");

    // Attemp to connect
    if (client.connect(MQTT_CLIENT_NAME, TOKEN, "")) {
      Serial.println("Connected");
      client.subscribe(topicSubscribe);
    } else {
      Serial.print("Failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 2 seconds");
      // Wait 2 seconds before retrying
      delay(2000);
    }
  }
}
void callback(char* topic, byte* payload, unsigned int length) {
  char p[length + 1];
  memcpy(p, payload, length);
  p[length] = NULL;
  String message(p);
  if (message == "0.0") {
    digitalWrite(led, LOW);
  } else {
    digitalWrite(led, HIGH);
  }

  Serial.write(payload, length);
  Serial.println();
}

/****************************************
 * Main Functions
 ****************************************/
void setup() {
  Serial.begin(115200);
  WiFi.begin(WIFISSID, PASSWORD);
  // Assign the pin as INPUT
  pinMode(led, OUTPUT);

  Serial.println();
  Serial.print("Wait for WiFi...");

  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }

  Serial.println("");
  Serial.println("WiFi Connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  client.setServer(mqttBroker, 1883);
  client.setCallback(callback);
  sprintf(topicSubscribe,"/v1.6/devices/%s/%s/lv",
DEVICE_LABEL,VARIABLE_LABEL_SUBSCRIBE);

  client.subscribe(topicSubscribe);
}

void loop() {
  if (!client.connected()) {
    client.subscribe(topicSubscribe);
    reconnect();
  }

  sprintf(topic, "%s%s", "/v1.6/devices/", DEVICE_LABEL);
  sprintf(payload, "%s", ""); // Cleans the payload
  sprintf(payload, "{\"%s\":", VARIABLE_LABEL); // Adds the variable label

  float sensor = hallRead();
  Serial.print("Value of Sensor is:- ");Serial.println(sensor);

  /* 4 is mininum width, 2 is precision; float value is copied onto str_sensor*/
  dtostrf(sensor, 4, 2, str_sensor);

  sprintf(payload, "%s {\"value\": %s}}", payload, str_sensor); // Adds the value
  Serial.println("Publishing data to Ubidots Cloud");
  client.publish(topic, payload);
  client.loop();
  delay(1000);
}
```

RESULT:



```
WiFi Connected
IP address:
192.168.43.146
Attempting MQTT connection...
Connected
Value of Sensor is: 5.00
Publishing data to Ubidots Cloud
Value of Sensor is: 1.00
Publishing data to Ubidots Cloud
Value of Sensor is: 4.00
Publishing data to Ubidots Cloud
Value of Sensor is: 5.00
Publishing data to Ubidots Cloud
Value of Sensor is: -2.00
Publishing data to Ubidots Cloud
```



Realtime monitoring Hall sensor & control LED on Ubidots dashboard

Experiment -5

Establish a communication between client and IoT Web Server to POST and GET sensor values over HTTP, TCP or UDP

APPARATUS:

S No	Name of the Equipment	Quantity
1	ESP32 Development Board	1
2	Micro USB cable	1

PROCEDURE:

1. Plug the ESP32 development board to your PC
2. Open the Arduino IDE in computer
3. Select the ESP32 board from Tools > Board > ESP32 Dev module.
4. Download the Ubidots library https://github.com/ubidots/ubidots-esp32
5. Now, click on Sketch -> Include Library -> Add .ZIP Library.
6. Select the .ZIP file of Ubidots and then "Accept".
7. Close the Arduino IDE and open it again and write the program. Save the new sketch in your working directory
   Note: **Make sure you have the right board and COM port selected in your Arduino IDE settings**.
8. Compile the program and upload it to the ESP32 Development Board. If everything went as expected, you should see a "Done uploading" message. (You need to hold the ESP32 on-board Boot button while uploading).
9. After uploading the code, open the Serial Monitor at a baud rate of 115200.

POST values to Web Server /Ubidots:

CODE:

```
#include "Ubidots.h"

/****************************************
 * Define Instances and Constants
 ****************************************/

const char* UBIDOTS_TOKEN = " YOUR_TOKEN ";  // Put here your Ubidots
TOKEN
const char* WIFI_SSID = " Replace_With_Your_Ssid "; // Put here your Wi-Fi SSID
const char* WIFI_PASS = " Replace_With_Your_password " // Put here your Wi-Fi
password
Ubidots ubidots(UBIDOTS_TOKEN, UBI_HTTP);
// Ubidots ubidots(UBIDOTS_TOKEN, UBI_TCP); // Uncomment to use TCP
// Ubidots ubidots(UBIDOTS_TOKEN, UBI_UDP); // Uncomment to use UDP

/****************************************
```

```
 * Auxiliar Functions
 ****************************************/

// Put here your auxiliar functions

/****************************************
 * Main Functions
 ****************************************/

void setup() {
  Serial.begin(115200);
  ubidots.wifiConnect(WIFI_SSID, WIFI_PASS);
  // ubidots.setDebug(true);  // Uncomment this line for printing debug messages
}

void loop() {
  float value1 = random(0, 9) * 10;
  float value2 = random(0, 9) * 100;
  float value3 = random(0, 9) * 1000;
  ubidots.add("counter", value1);  // Change for your variable name
  //ubidots.add("Variable_Name_Two", value2);
 // ubidots.add("Variable_Name_Three", value3);

  bool bufferSent = false;
  bufferSent = ubidots.send();  // Will send data to a device label that matches the device
Id

  if (bufferSent) {
    // Do something if values were sent properly
    Serial.println("Values sent by the device");
  }

  delay(5000);
}
```

GET values from Web Server /Ubidots:

CODE:

```
#include "Ubidots.h"

/****************************************
 * Define Constants
 ****************************************/

const char* UBIDOTS_TOKEN = " YOUR_TOKEN ";  // Put here your Ubidots TOKEN
const char* WIFI_SSID = "_Your_SSID";      // Put here your Wi-Fi SSID
const char* WIFI_PASS = "Your_Password";  // Put here your Wi-Fi password
const char* DEVICE_LABEL_TO_RETRIEVE_VALUES_FROM = " your device label ";
// Replace with your device label
const char* VARIABLE_LABEL_TO_RETRIEVE_VALUES_FROM = " your variable ";       //
Replace with your variable label

Ubidots ubidots(UBIDOTS_TOKEN, UBI_HTTP);
// Ubidots ubidots(UBIDOTS_TOKEN, UBI_TCP); // Uncomment to use TCP

/****************************************
 * Auxiliar Functions
 ****************************************/

// Put here your auxiliar functions

/****************************************
 * Main Functions
 ****************************************/

void setup() {
  Serial.begin(115200);
  ubidots.wifiConnect(WIFI_SSID, WIFI_PASS);
  // ubidots.setDebug(true); //Uncomment this line for printing debug messages
}

void loop() {
  /* Obtain last value from a variable as float using HTTP */
  float value = ubidots.get(DEVICE_LABEL_TO_RETRIEVE_VALUES_FROM,
VARIABLE_LABEL_TO_RETRIEVE_VALUES_FROM);

  // Evaluates the results obtained
  if (value != ERROR_VALUE) {
    Serial.print("Value:");
    Serial.println(value);
  }
  delay(5000);
}
```

PROBABLE/SUGGESTED QUESTION BANK

1. What are functions of different layers in OSI model?

2. Differentiate between TCP/IP Layers and OSI Layers.

3. Why header is required?

4. What is the use of adding header and trailer to frames?

5. What is encapsulation?

6. Why fragmentation is required?

7. Name the protocols in the TCP/IP protocol suite?

8. What is bit stuffing and its applications?

9. What is byte stuffing and its applications?

10. Differentiate between flow control and congestion control.

10. Differentiate between Point-to-Point Connection and End-to-End connections.

11. What is HDLC protocol?

12. Explain the frame format of I, U and S frames in detail.

13. What is CRC? Explain in detail the procedure to find CRC? In which CRC is used?

13. Differentiate between TCP and UDP.

14. What is Sliding window protocol? Name the two protocols comes under Sliding window protocol.

15. Explain different types of routing protocols?

16. Explain the difference between Distance vector and Link state algorithm.

17. Explain the procedure to calculate the shortest path in Dijkstra’s algorithm.

18. Explain different types of Congestion control algorithms?

19. Explain the difference between Leaky bucket and token bucket algorithm in detail.

20. Explain different types of Encryption and Decryption algorithms in detail.

21. Explain the procedure to calculate the Encryption and Decryption procedure in Substitution method.

22. Explain the procedure to calculate the Encryption and Decryption procedure in Transposition method.

23. What are the other error detection algorithms?

29. What are drawbacks in distance vector algorithm?
30. What is cryptography?
31. How do you classify cryptographic algorithms?
32. What are public key and private key?
33. What is NS2?
34. Explain the architecture of NS2.
35. Explain the frame format of Trace file for wired node in detail.
36. Which are scripts used in NS2?
37. What is awk?
36. What is an agent and traffic descriptors?
37. Name the agents and its associated traffic descriptors used in NS2 in detail.
38. For TCP and UDP the destination nodes are called as?
39. What are the applications of TCP and UDP?
40. What is an Ethernet LAN? Which protocol is used?
41. What is an error rate and data rate? Explain in detail.
42. How the throughput is calculated when the error rate and data rates are changed.
43. What is congestion window?
44. If the congestion window is not set what will happen?
45. Which are commands used to check the particular node congestion?
46. Explain the congestion window graph in detail.
47. What is ESS and BSS?
48. Explain the frame format of Trace file for wireless nodes in detail.
49. Which protocol is used in Wireless LAN?
50. How the throughput is calculated in the Wireless LAN?