NAME: **Rutupriya Purawat**

ROLL NO: **B-18**

SEAT NO: **S.19.78**

Wireless Sensor Network and Mobile Communication All Practicals

SIES (Nerul) College of Arts, Science & Commerce
RISE WITH EDUCATION

SIES (NERUL) COLLEGE OF ARTS, SCIENCE AND COMMERCE

NAAC ACCREDITATED
(ISO 9001:2008 CERTIFIED INSTITUTION)
NERUL, NAVI MUMBAI 400 706.

Certificate

Class TY BSc. CS *Roll No.* B-18

Seat No. S.19.78

Certified that Mr. / Miss. RUTUPRIYA PURAWAT
of TY BSc. CS *has duly completed the practical course in Computer Science / Information Technology during the academic year* 2021 - 2022 *as per the syllabus prescribed by the University of Mumbai.*

Prof. in charge of Practicals *Co-ordinator*

Date : _____ *Date :* _____

External Examiner

Date : _____ *Principal*

CONTENTS

SIES (Nerul) College of Arts, Science & Commerce
RISE WITH EDUCATION

RUTUPRIYA PURAWAT, B-18

PRACTICAL 1

AIM: Understanding the Sensor Node Hardware

Wireless sensor networks (WSNs) have the power of distributed communication, computing, and sensing features. They are characterized as infrastructure less, fault tolerant and self-organizing networks which provide opportunities for low-cost, easy-to-apply, rapid and flexible installations in an environment for various applications.

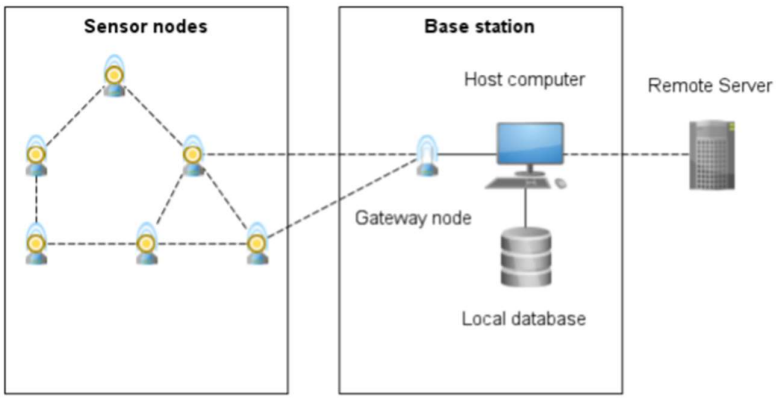
The components of sensor node hardware are as follows:

1. Sensors: Sensors in WSN are used to capture the environmental variables and which is used for data acquisition. Sensor signals are converted into electrical signals. They are capable of executing data processing, data gathering and communicating with additional associated nodes in the network. A distinctive sensor node capability is about 4-8 MHz, having 4 KB of RAM, 128 KB flash and preferably 916 MHz of radio frequency.

2. Nodes: They are used to enhance the network reliability. A relay node is a special type of field device that does not have process sensor or control equipment and as such does not interface with the 7 process itself. A distinctive rely node processor speed is about 8 MHz, having 8 KB of RAM, 128 KB flash and preferably 916 MHz of radio frequency.

3. Base Station: Besides sensor nodes, another fundamental item - a base station - can be found in WSN. In contrast to sensor nodes, the base station possesses much more computational power, larger memory and is often connected to better energy source than batteries (like power grid). One can look at the base station as an entry point to the WSN where the base station's primary goal is to gather sensed data from sensor nodes in WSN.

4. Graphical User Interface: GUI is the web-based software package, that allows the data collected by sensors to be viewed. The software is also used to set irrigation parameters.



Example of a Wireless Sensor Network (WSN)

RUTUPRIYA PURAWAT, B-18

PRACTICAL 2

AIM: Exploring and understanding TinyOS computational concepts: Events, Commands and Task, nesC model, nesC Components

TinyOS

TinyOS has a component-based programming model, codified by the NesC language, a dialect of C. TinyOS is not an OS in the traditional sense; it is a programming framework for embedded systems and set of components that enable building an application-specific OS into each application. A typical application is about 15K in size, of which the base OS is about 400 bytes; the largest application, a database-like query system, is about 64 K bytes.

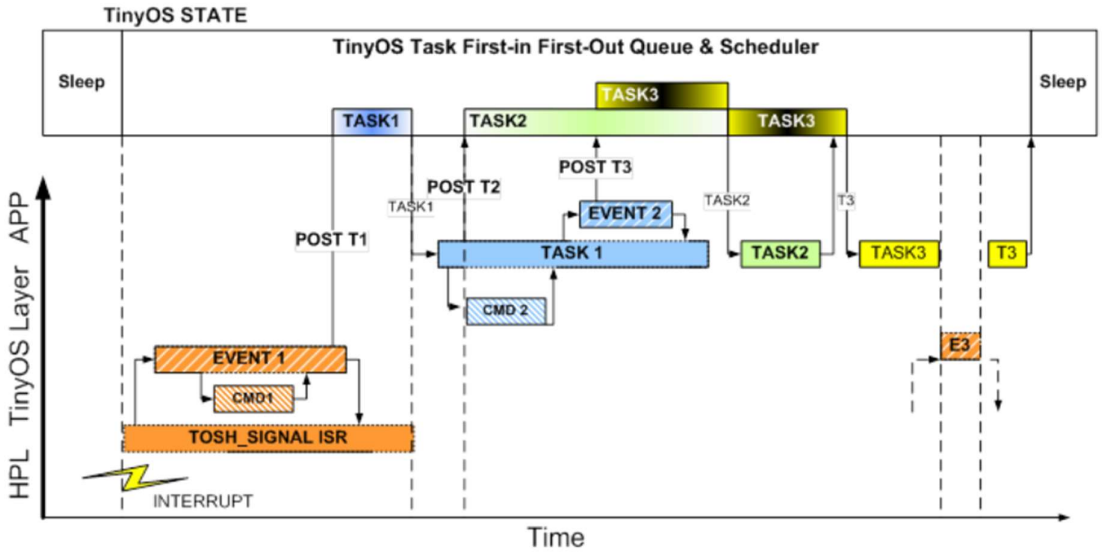
The computational concepts of TinyOS include: Events, Commands and Task.

1. Events: When a command is requested, event signals the completion of that service. Events may also be signaled asynchronously, for example, due to hardware interrupts or message arrival. From a traditional OS perspective, events are analogous to upcalls.

2. Commands: A command is typically a request to a component to perform some service, such as initiating a sensor reading. Commands are analogous to downcalls. The command returns immediately and the event signals completion at a later time.

3. Task: Rather than performing a computation immediately, commands and event handlers may post a task, a function executed by the TinyOS scheduler at a later time. This allows commands and events to be responsive, returning immediately while deferring extensive computation to tasks. While tasks may perform significant computation, their basic execution model is run-to-completion, rather than to run indefinitely; this allows tasks to be much lighter-weight than threads.

TinyOS Execution Model



nesC Model

1. nesC is a component-based, event-driven programming language used to build applications for the TinyOS platform. The name nesC is an abbreviation of "network embedded systems C".
2. TinyOS is an operating environment designed to run on embedded devices used in distributed wireless sensor networks.
3. nesC is built as an extension to the C programming language with components "wired" together to run applications on TinyOS.
4. nesC is based on the concept of components, and directly supports TinyOS's eventbased concurrency model.
5. nesC programs are subject to whole program analysis (for safety) and optimization (for performance).

nesC Components

1. nesC applications are built by writing and assembling components.
2. A component provides and uses interfaces. These interfaces are the only point of access to the component.
3. An interface generally models some service (e.g., sending a message) and is specified by an interface type.
4. Interfaces in nesC are bidirectional: they contain commands and events, both of which are essentially functions. The providers or an interface implement the commands, while the users implement the events.
5. The separation of interface type definitions from their use in components promotes the definition of standard interfaces, making components more reusable and flexible.
6. A component can provide and use the same interface type (e.g., when interposing a component between a client and service), or provide the same interface multiple times. In these cases, the component must give each interface instance a separate name.
7. Components are also a clean way to abstract the boundary between hardware and software.
8. A subtle but important point is that bidirectional interfaces make it very easy to support hardware interrupts.
9. In contrast, one-way interfaces based on procedure calls force hardware polling or having two separate interfaces for hardware operations and the corresponding interrupts.

10. There are two types of components in nesC: modules and configurations. Modules provide application code, implementing one or more interfaces. Configurations are used to wire other components together, connecting interfaces used by components to interfaces provided by others.
11. Every nesC application is described by a top-level configuration that wires together the components used.
12. Most components in TinyOS represent services or pieces of hardware (such as the LEDs) and therefore exist only in a single instance. However, it is sometimes useful to create several instances of a component.
13. In nesC, this is achieved by declaring an abstract component with optional parameters; abstract components are created at compile-time in configurations.

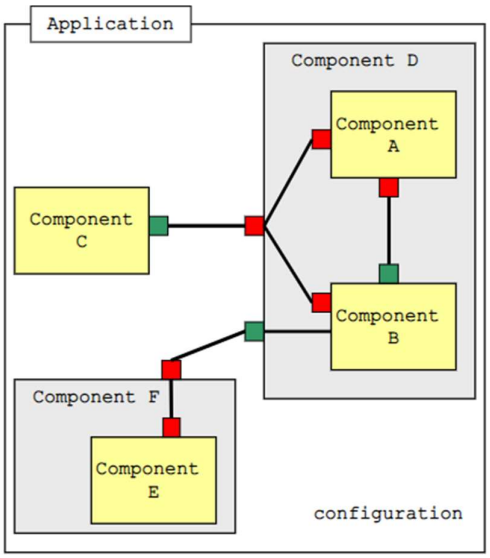
nesC model

Two key concepts

1. **Interfaces**
   - uses
   - provides
2. **Components**
   - modules
   - configurations

Application: graph of components

Interfaces: only point of access to components



RUTUPRIYA PURAWAT, B-18

PRACTICAL 3

AIM: Understanding TOSSIM for:

- Mote-mote radio communication
- Mote-PC serial communication

TOSSIM is a discrete event simulator, or more precisely emulator, of TinyOS at the bit level. An event is generated for each transmitted or received bit rather than one per packet. TOSSIM is a simulator for the execution of nesC model on TinyOS/MICA hardware. TOSSIM further works as an emulator of actual hardware through mapping hardware interrupts to discrete events. TOSSIM also has a built-in simulated radio model. In conclusion, TOSSIM is a high-fidelity emulator of a network of TinyOS/MICA motes.

TOSSIM is conceived to study the behaviour of TinyOS and its applications: it is not intended to profile the performance of new protocols. Accordingly, there is one significant drawback and that is the fact that every mote must run the same code. Consequently, the usability of TOSSIM for applications involving heterogeneous sensors is limited at best. As such, TOSSIM is not appropriate for heterogeneous applications. TOSSIM has been shown to handle simulations of a sensor network with around a thousand motes. Its ability to simulate larger mote counts is primarily limited by its bit-level granularity: its performance degrades as the traffic increases. Channel sampling is also simulated at the bit level and consequently the use of a CSMA protocol causes significant overhead particularly when compared to a TDMA-based MAC protocol.

The simulator engine provides a set of communication services for interacting with external applications. These services allow programs to connect to TOSSIM over a TCP socket to monitor or actuate a running simulation. Details of the ADC and radio models, such as readings and loss rates, can be both queried and set. Programs can also receive higher level information, such as packet transmissions and receptions or application-level events. TOSSIM supports the TinyOS tool-chain, making the transitions between simulated and real networks easy. Compiling to native code allows developers to use traditional tools such as debuggers in TOSSIM. As it is a discrete event simulation, users can set debugger breakpoints and step through what is normally real-time code (such as packet reception) without disrupting operation. It also provides mechanisms for other programs to interact and monitor a running simulation; by keeping monitoring and interaction external to TOSSIM, the core simulator engine remains very simple and efficient.

RUTUPRIYA PURAWAT, B-18

PRACTICAL 4

AIM: Create and simulate a simple adhoc network.

Steps:

- **To configure OMNET++**
  1. 1. Copy the OMNeT++ archive to the directory where you want to install it. Choose a directory whose full path does not contain any space; for example, do not put OMNeT++ under Program Files.
  2. Extract the zip file.
  3. Start mingwenv.cmd in the omnetpp-4.2.2 directory by double-clicking it in Windows Explorer.
  4. Then enter the following commands:
  5. $ ./configure
  6. $ make

- **Starting the IDE**
  OMNeT++ comes with an Eclipse-based Simulation IDE. You should be able to start

  1. Start the Omnet++ IDE, and import the project via File -> Import -> Existing Projects to the Workspace. A project named inet should appear.
  2. Build with Project -> Build, or hit ctrl+b
  3. Now you should be able to launch example simulations.
  4. Then open inet/examples/
  5. Right click on adhoc -create new folder as SimpleAdhoc.
  6. Right click on your newly created folder and select NED file. Give name as Net1.
  7. In Initial Contents, click on new manages mobility wireless network wizard.
  8. Keep the options pane as it is then click on finish.

Below is the code that will be available in source part of net1.ned once configured:

```
package inet.examples.adhoc.SimpleAdhoc;
// numOfHosts: 5
import inet.networklayer.autorouting.ipv4.IPv4NetworkConfigurator;
import inet.nodes.inet.WirelessHost;
import inet.nodes.wireless.AccessPoint;
import inet.world.radio.ChannelControl;
network Net
{
parameters:
int numOfHosts;
submodules:
host[numOfHosts]: WirelessHost
{
@display("r=,,#707070");
}
ap: AccessPoint
{
@display("p=213,174;r=,,#707070");
}
channelControl: ChannelControl
{
numChannels = 2;
@display("p=61,46");
}
configurator: IPv4NetworkConfigurator
{
```

```
@display("p=140,50");
}
}
```

On design part you will find components appearing according to the code as the above snapshot.

Same as do this in omnetpp.ini file :

Source code for omnetpp.ini:

```
[General]
network = Net1
*.numOfHosts = 5
#debug-on-errors = true
tkenv-plugin-path = ../../../etc/plugins
**.constraintAreaMinX = 0m
**.constraintAreaMinY = 0m
**.constraintAreaMinZ = 0m
**.constraintAreaMaxX = 600m
**.constraintAreaMaxY = 400m
**.constraintAreaMaxZ = 0m
**.debug = true
**.coreDebug = false
**.host*.**.channelNumber = 0
# channel physical parameters
*.channelControl.carrierFrequency = 2.4GHz
*.channelControl.pMax = 2.0mW
*.channelControl.sat = -110dBm
*.channelControl.alpha = 2
# mobility
**.host*.mobilityType = "MassMobility"
```

```
**.host*.mobility.initFromDisplayString = false
**.host*.mobility.changeInterval = truncnormal(2s, 0.5s)
**.host*.mobility.changeAngleBy = normal(0deg, 30deg)
**.host*.mobility.speed = truncnormal(20mps, 8mps)
**.host*.mobility.updateInterval = 100ms
# ping app (host[0] pinged by others)
*.host[0].numPingApps = 0
*.host[*].numPingApps = 2
*.host[*].pingApp[*].destAddr = "host[0]"
**.pingApp[0].startTime = uniform(1s,5s)
**.pingApp[1].startTime = 5s+uniform(1s,5s)
**.pingApp[*].printPing = true
# nic settings
**.wlan[*].bitrate = 2Mbps
**.wlan[*].mgmt.frameCapacity = 10
**.wlan[*].mac.address = "auto"
**.wlan[*].mac.maxQueueSize = 14
**.wlan[*].mac.rtsThresholdBytes = 3000B
**.wlan[*].mac.retryLimit = 7
**.wlan[*].mac.cwMinData = 7
**.wlan[*].radio.transmitterPower = 2mW
**.wlan[*].radio.thermalNoise = -110dBm
**.wlan[*].radio.sensitivity = -85dBm
**.wlan[*].radio.pathLossAlpha = 2
**.wlan[*].radio.snirThreshold = 4dB
[Config Ping1]
description = "host1 pinging host0"
[Config Ping2] # __interactive__
```

```
description = "n hosts"
# leave numHosts undefined here
**.mobility.constraintAreaMinZ = 0m
**.mobility.constraintAreaMaxZ = 0m
**.mobility.constraintAreaMinX = 0m
**.mobility.constraintAreaMinY = 0m
**.mobility.constraintAreaMaxX = 600m
**.mobility.constraintAreaMaxY = 400m
**.debug = false
**.coreDebug = false
**.channelNumber = 0
# channel physical parameters
*.channelControl.carrierFrequency = 2.4GHz
*.channelControl.pMax = 20.0mW
*.channelControl.sat = -110dBm
*.channelControl.alpha = 2
# mobility
**.host[*].mobilityType = "MassMobility"
**.host[*].mobility.changeInterval = truncnormal(2s, 0.5s)
**.host[*].mobility.changeAngleBy = normal(0deg, 30deg)
**.host[*].mobility.speed = truncnormal(20mps, 8mps)
**.host[*].mobility.updateInterval = 100ms
# nic settings
**.bitrate = 2Mbps
**.mac.address = "auto"
**.mac.maxQueueSize = 14
**.mac.rtsThresholdBytes = 3000B
**.wlan[*].mac.retryLimit = 7
```

**.wlan[*].mac.cwMinData = 7

**.wlan[*].mac.cwMinMulticast = 31

**.radio.transmitterPower = 20.0mW

**.radio.carrierFrequency = 2.4GHz

**.radio.thermalNoise = -110dBm

**.radio.sensitivity = -85dBm

**.radio.pathLossAlpha = 2

**.radio.snirThreshold = 4dB

relay unit configuration

**.relayUnitType = "MACRelayUnitNP"

**.relayUnit.addressTableSize = 100

**.relayUnit.agingTime = 120s

**.relayUnit.bufferSize = 1MiB

**.relayUnit.highWatermark = 512KiB

**.relayUnit.pauseUnits = 300 # pause for 300*512 bit (19200 byte) time

**.relayUnit.addressTableFile = ""

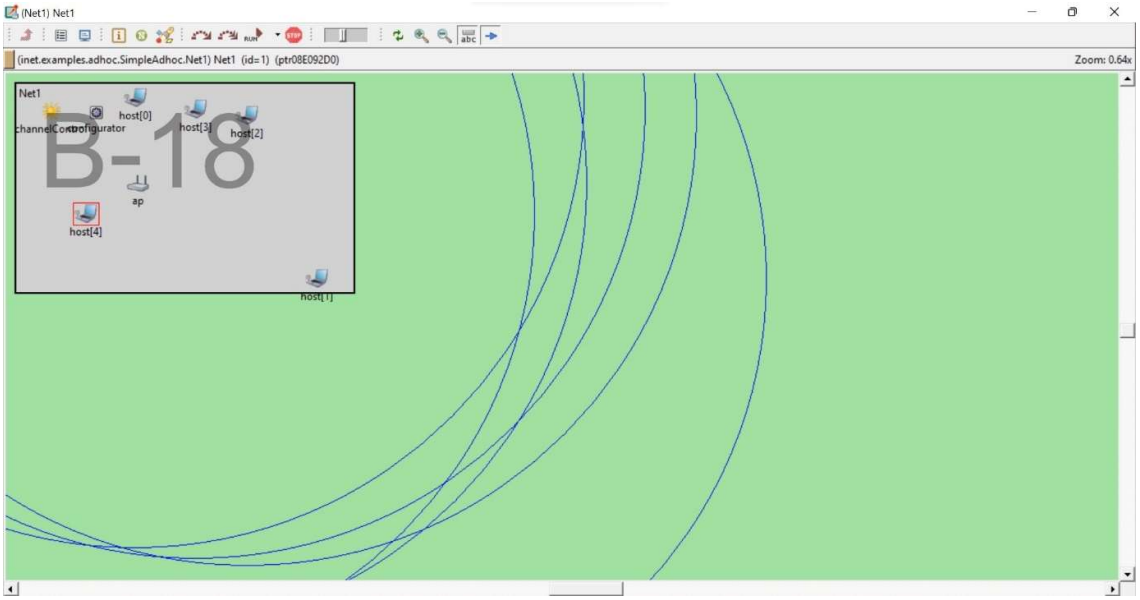
**.relayUnit.numCPUs = 2

**.relayUnit.processingTime = 2us

EXECUTION:

Now try to execute by right click on ned file Run as-1-Omnet++ simulation.

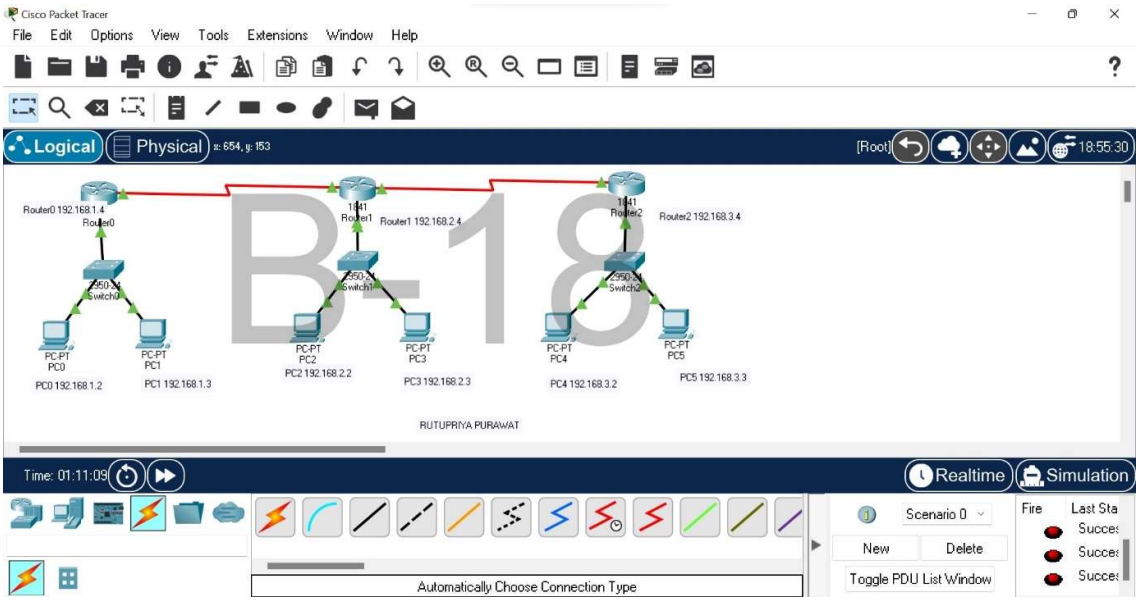
OUTPUT:


(Net1) Net1
(inet.examples.adhoc.SimpleAdhoc.Net1) Net1 (id=1) (ptr08E092D0)
Zoom: 0.64x
Net1
host[0]
host[3]
host[2]
channelController
configurator
B-18
ap
host[4]
host[1]

RUTUPRIYA PURAWAT, B-18

PRACTICAL 5

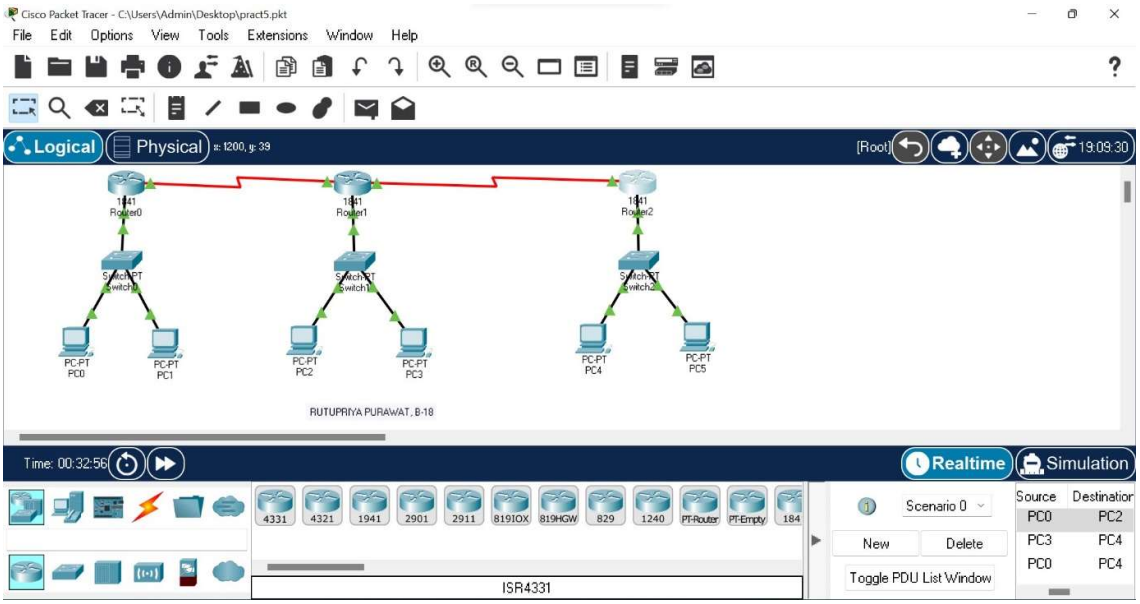
AIM: Understanding, Reading and Analyzing the Routing Table of a network

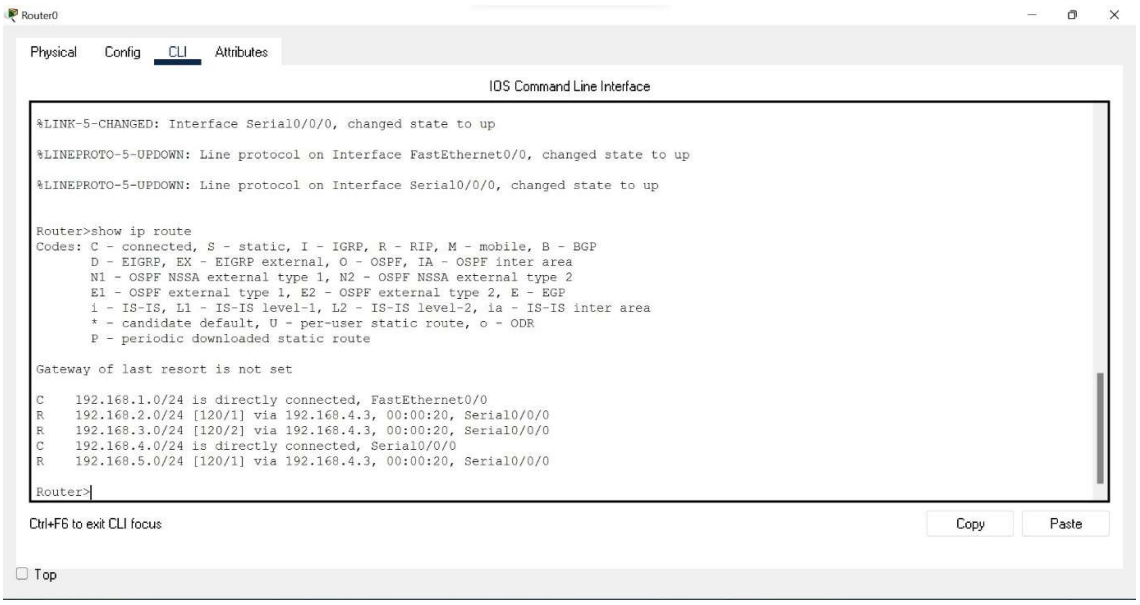


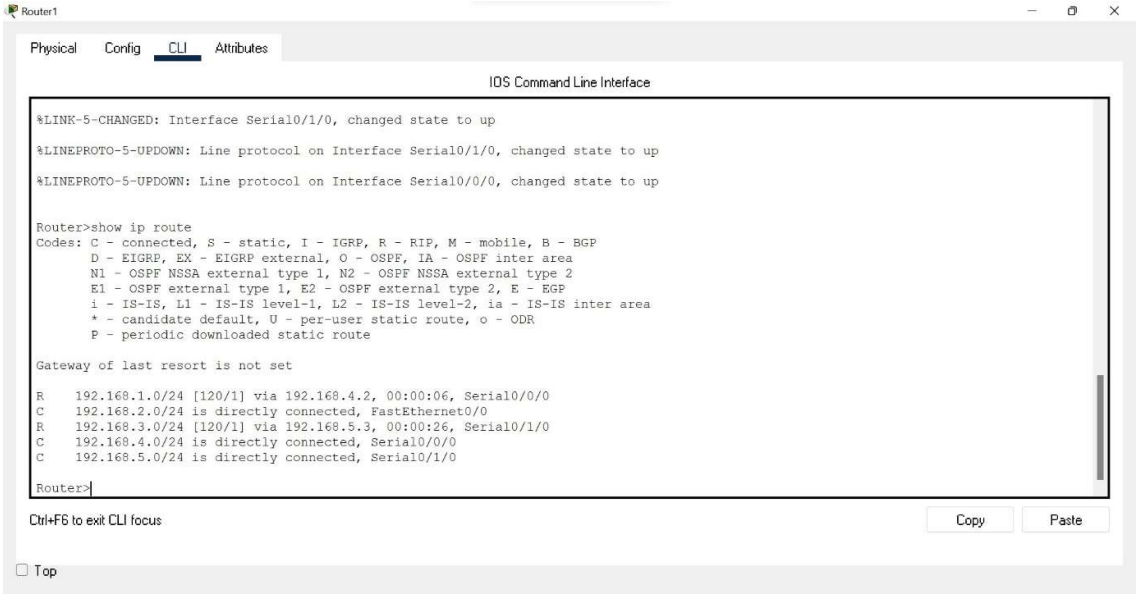
RUTUPRIYA PURAWAT, B-18

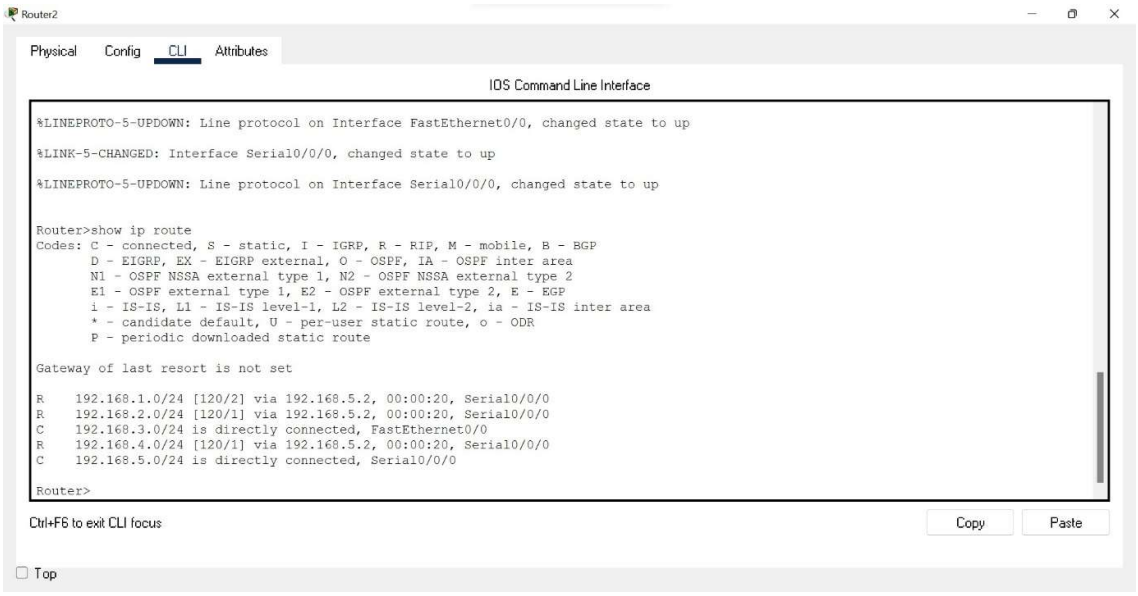
PRACTICAL 5

AIM: Understanding, Reading and Analyzing the Routing Table of a network






Router1
Physical
Config
CLI
Attributes
IOS Command Line Interface
%LINK-5-CHANGED: Interface Serial0/1/0, changed state to up
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/1/0, changed state to up
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/0/0, changed state to up
Router>show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
* - candidate default, U - per-user static route, o - ODR
P - periodic downloaded static route
Gateway of last resort is not set
R 192.168.1.0/24 [120/1] via 192.168.4.2, 00:00:06, Serial0/0/0
C 192.168.2.0/24 is directly connected, FastEthernet0/0
R 192.168.3.0/24 [120/1] via 192.168.5.3, 00:00:26, Serial0/1/0
C 192.168.4.0/24 is directly connected, Serial0/0/0
C 192.168.5.0/24 is directly connected, Serial0/1/0
Router>
Ctrl+F6 to exit CLI focus
Copy
Paste
Top


Router2
Physical
Config
CLI
Attributes
IOS Command Line Interface
%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to up
%LINK-5-CHANGED: Interface Serial0/0/0, changed state to up
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/0/0, changed state to up
Router>show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
* - candidate default, U - per-user static route, o - ODR
P - periodic downloaded static route
Gateway of last resort is not set
R 192.168.1.0/24 [120/2] via 192.168.5.2, 00:00:20, Serial0/0/0
R 192.168.2.0/24 [120/1] via 192.168.5.2, 00:00:20, Serial0/0/0
C 192.168.3.0/24 is directly connected, FastEthernet0/0
R 192.168.4.0/24 [120/1] via 192.168.5.2, 00:00:20, Serial0/0/0
C 192.168.5.0/24 is directly connected, Serial0/0/0
Router>
Ctrl+F6 to exit CLI focus
Copy
Paste
Top

RUTUPRIYA PURAWAT, B-18

PRACTICAL 6

AIM: Create a basic MANET implementation simulation for Packet animation and Packet Trace.

Steps:

- **To configure OMNET++**
  1. 1. Copy the OMNeT++ archive to the directory where you want to install it. Choose a directory whose full path does not contain any space; for example, do not put OMNeT++ under Program Files.
  2. Extract the zip file.
  3. Start mingwenv.cmd in the omnetpp-4.2.2 directory by double-clicking it in Windows Explorer.
  4. Then enter the following commands:
  5. $ ./configure
  6. $ make

- **Starting the IDE**
  OMNeT++ comes with an Eclipse-based Simulation IDE. You should be able to start

  1. Start the Omnet++ IDE, and import the project via File -> Import -> Existing Projects to the Workspace. A project named inet should appear.
  2. Build with Project -> Build, or hit ctrl+b
  3. Now you should be able to launch example simulations.
  4. Then open inet/examples/
  5. Right click on adhoc -create new folder as SimpleAdhoc.
  6. Right click on your newly created folder and select NED file. Give name as Net1.
  7. In Initial Contents, click on new manages mobility wireless network wizard.
  8. Keep the options pane as it is then click on finish.

Below is the code that will be available in source part of net1.ned once configured:

```
package inet.examples.adhoc.mob;

import inet.world.radio.ChannelControl;

network mobnet

{

parameters :

double hosts;

submodules :

channelControl: ChannelControl;

host[hosts]: Host1;

}
```

Add another NED file As Host1 in a same manner. And add the source code as follows:

```
package inet.examples.adhoc.mob;

import inet.networklayer.autorouting.ipv4.HostAutoConfigurator;

import inet.nodes.inet.WirelessHost;

//

// Wireless-enabled Host

//

module Host1 extends WirelessHost

{

parameters :

@display ( "i=device/cellphone" );

submodules :

ac_wlan: HostAutoConfigurator {

@display ( "p=171,335" );

}

}
```

On design part you will find components appearing according to the code as the below

snapshot.

Same as do this in omnetpp.ini file :

Source code for omnetpp.ini:

```
[General]

debug-on-errors = true

network = mobnet

sim-time-limit = 60 min

cmdenv-express-mode = true

*.hosts = 3

**.constraintAreaMinX = 0 m

**.constraintAreaMinY = 0 m

**.constraintAreaMinZ = 0 m

**.constraintAreaMaxX = 600 m

**.constraintAreaMaxY = 400 m

**.constraintAreaMaxZ = 0 m

**.debug = true

**.coreDebug = false

**.host*.**.channelNumber = 0

# channel physical parameters

*.channelControl.carrierFrequency = 2 . 4 GHz

*.channelControl.pMax = 2 . 0 mW

*.channelControl.sat = - 110 dBm

*.channelControl.alpha = 2

*.channelControl.numChannels = 1

# mobility

**.host*.mobilityType = "MassMobility"
```

.host*.mobility.initFromDisplayString = **false

**.host*.mobility.changeInterval = truncnormal (2 s, 0 . 5 s)

**.host*.mobility.changeAngleBy = normal (0 deg, 30 deg)

**.host*.mobility.speed = truncnormal (20 mps, 8 mps)

**.host*.mobility.updateInterval = 100 ms

**.host*.ac_wlan.interfaces = "wlan0"

 # UDPBasicApp / UDPSink

**.numUdpApps = 1

**.udpApp[0]. typename = "UDPBasicApp"

**.udpApp[0].destAddresses = "host[0]"

**.udpApp[0].localPort = 9001

**.udpApp[0].destPort = 9001

**.udpApp[0].messageLength = 100 B

**.udpApp[0].startTime = uniform (10 s, 30 s)

**.udpApp[0].sendInterval = uniform (10 s, 30 s)

 # nic settings

**.wlan[*].mgmtType = "Ieee80211MgmtAdhoc"

**.wlan[*].bitrate = 2 Mbps

**.wlan[*].mgmt.frameCapacity = 10

**.wlan[*].mac.address = "auto"

**.wlan[*].mac.maxQueueSize = 14

**.wlan[*].mac.rtsThresholdBytes = 3000 B

**.wlan[*].mac.retryLimit = 7

**.wlan[*].mac.cwMinData = 7

**.wlan[*].radio.transmitterPower = 2 mW

**.wlan[*].radio.thermalNoise = - 110 dBm

**.wlan[*].radio.sensitivity = - 85 dBm

**.wlan[*].radio.pathLossAlpha = 2

**.wlan[*].radio.snirThreshold = 4 dB

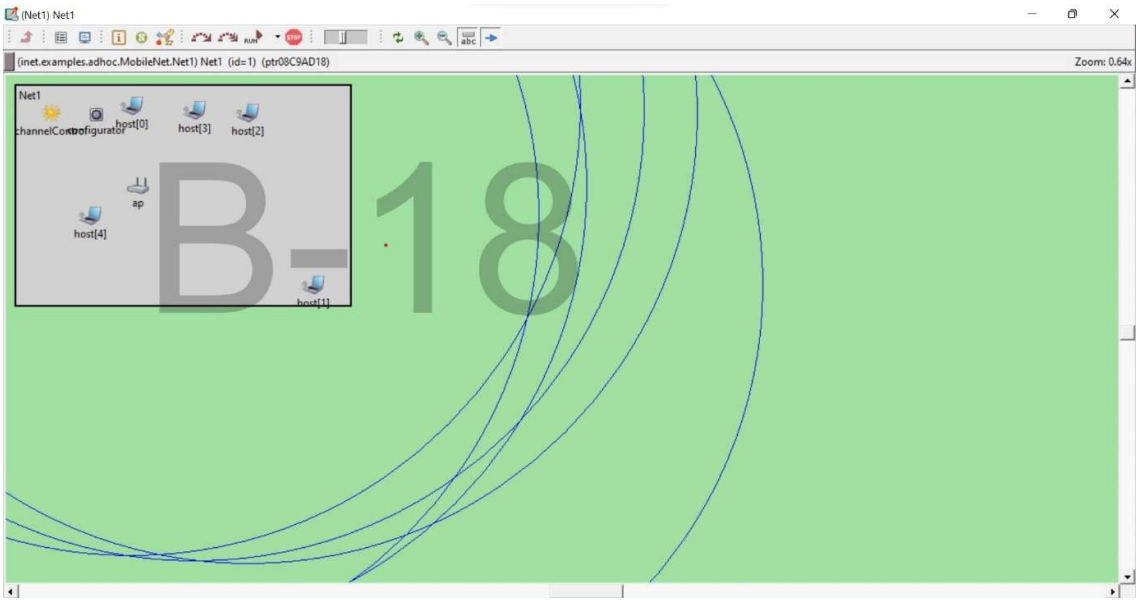
.udpapp.*. vector-recording = **true

. vector-recording = **true

EXECUTION:

Now try to execute by right click on ned file Run as-1-Omnet++ simulation.

OUTPUT:



RUTUPRIYA PURAWAT, B-18

PRACTICAL 7

AIM: Implement a Wireless sensor network simulation.

STEPS:

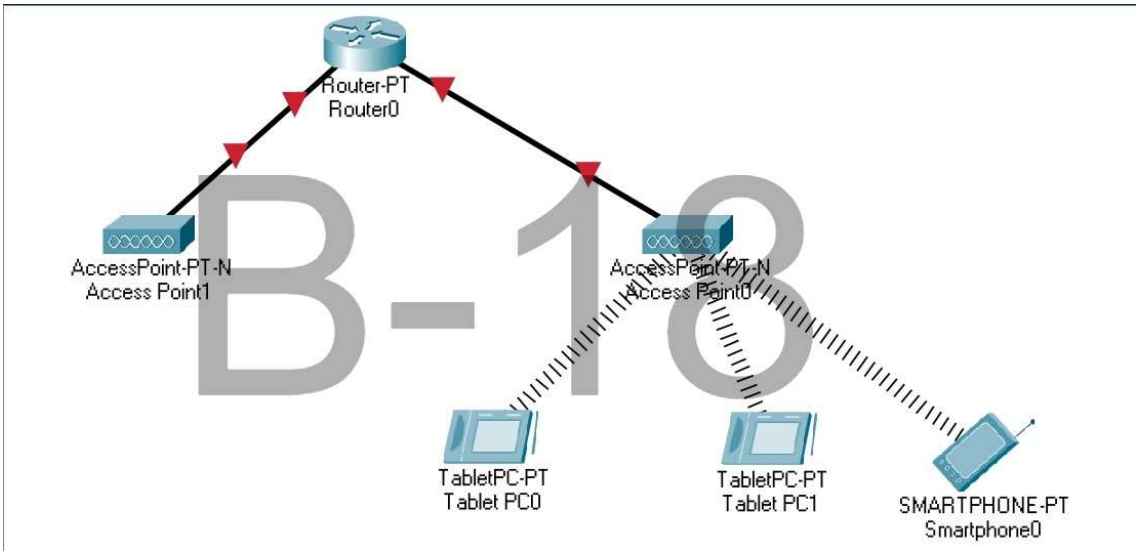
1)Create a network using 1 Router-PT, 2 AccessPoint-PT-N and some end devices like tablets, smartphones and laptop.

2)Connect access points and router using coaxial cable.

3) In laptop connect Linksys WPC300N (The Linksys-WPC300N module provides one 2.4GHz wireless interface suitable for connection to wireless networks. The module supports protocols that use Ethernet for LAN access.).

4) if connecting PC then connect PT-HOST-NM-1W (The PT-HOST-NM-1W module provides one 2.4GHz wireless interface suitable for connection to wireless networks. The module supports protocols that use Ethernet for LAN access.)

OUTPUT:



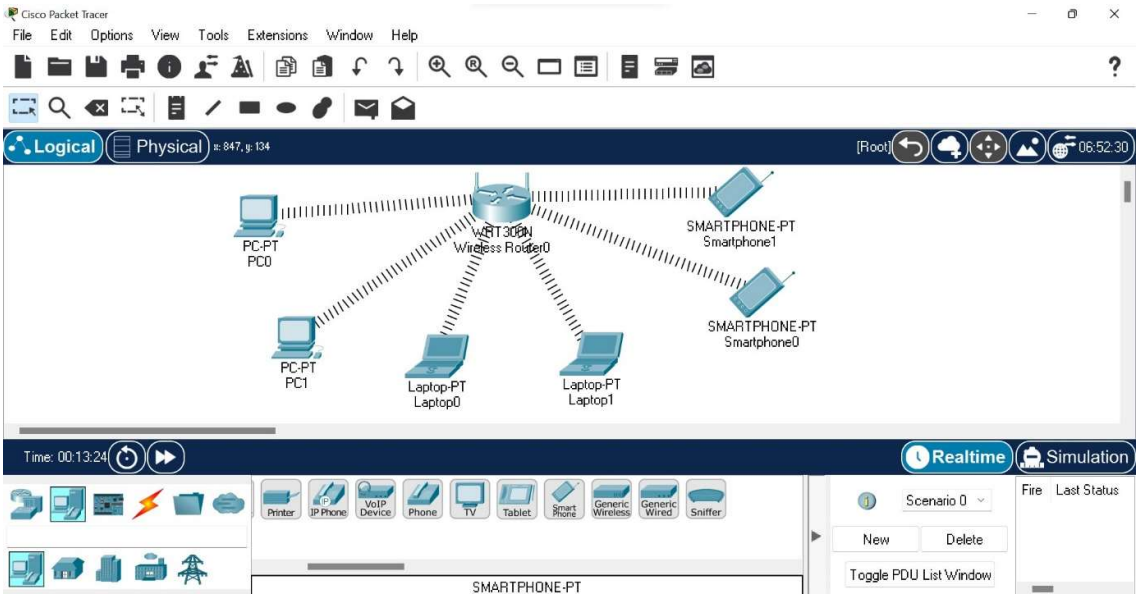
RUTUPRIYA PURAWAT, B-18

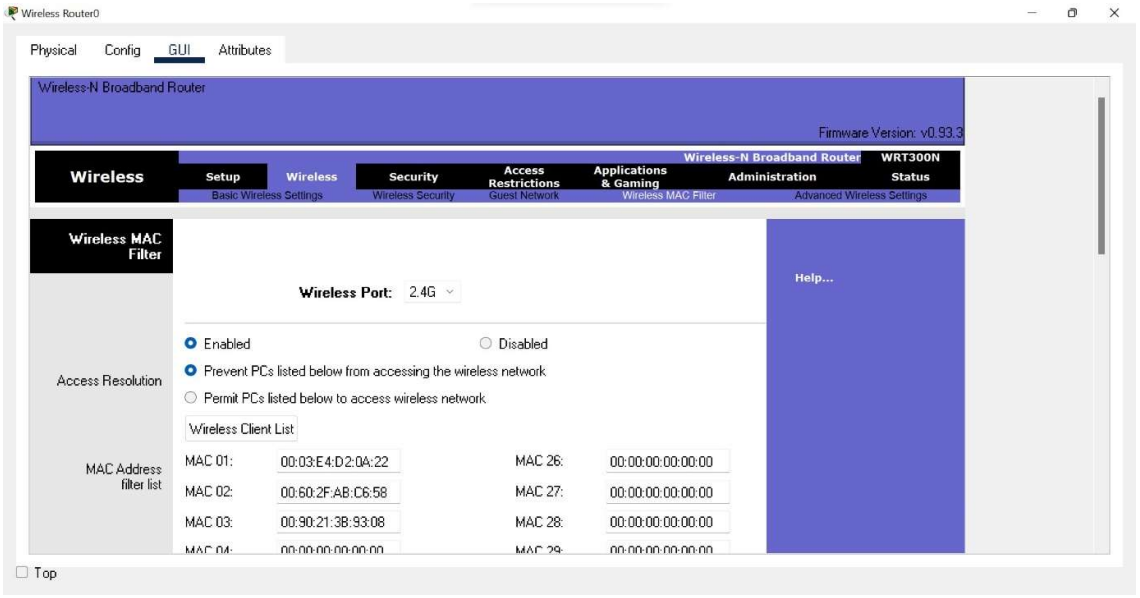
PRACTICAL 8

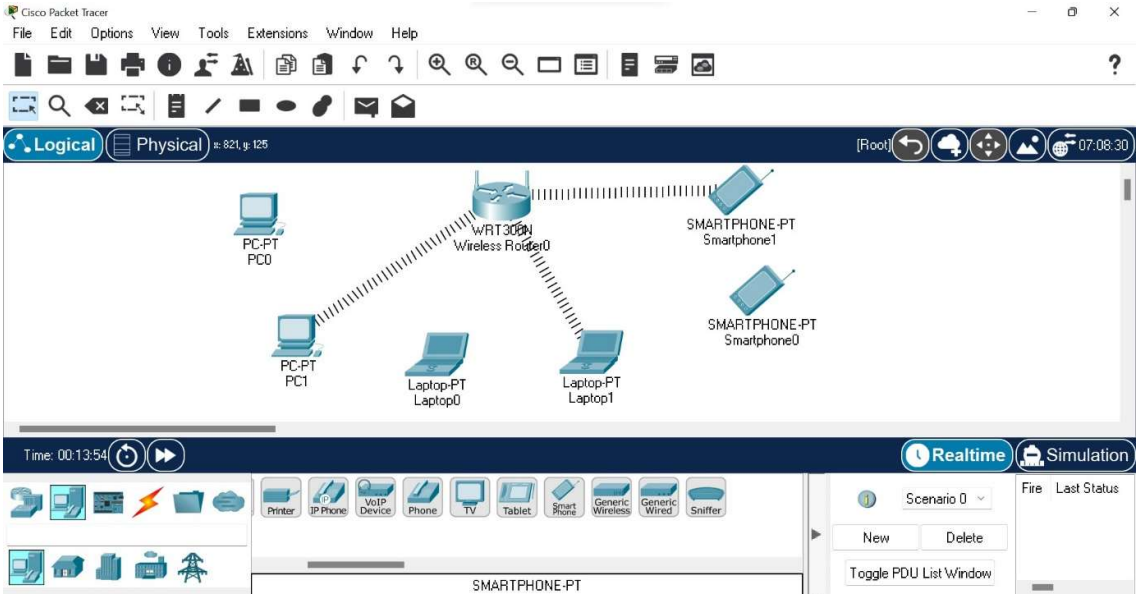
AIM: Create MAC protocol simulation implementation for wireless sensor Network.

STEPS:

1. Create a network using Linksys WRT300N and some end devices like smart phones, laptop PC.
2. In laptop connect Linksys WPC300N (The Linksys-WPC300N module provides one 2.4GHz wireless interface suitable for connection to wireless networks. The module supports protocols that use Ethernet for LAN access.)
3. In PC connect PT-HOST-NM-1W (The PT-HOST-NM-1W module provides one 2.4GHz wireless interface suitable for connection to wireless networks. The module supports protocols that use Ethernet for LAN access.)
4. In the GUI of wireless router go to Wireless MAC Filter and enable it.
5. For the MAC address go to the config tab of the end device under fast Ethernet copy the MAC address put colon after 2 digits and save it in textbox below the end devices.
6. Now enter the MAC Address in the MAC address filter list of the end devices to prevent end devices from accessing the wireless routers.




Wireless Router0
Physical
Config
GUI
Attributes
Wireless-N Broadband Router
Firmware Version: v0.93.3
Wireless
Setup
Wireless
Security
Access Restrictions
Applications & Gaming
Administration
Status
Basic Wireless Settings
Wireless Security
Guest Network
Wireless MAC Filter
Advanced Wireless Settings
Wireless MAC Filter
Wireless Port: 2.4G
Help...
Enabled
Disabled
Access Resolution
Prevent PCs listed below from accessing the wireless network
Permit PCs listed below to access wireless network
Wireless Client List
MAC Address filter list
MAC 01: 00:03:E4:D2:0A:22
MAC 02: 00:60:2F:AB:C6:58
MAC 03: 00:90:21:3B:93:08
MAC 26: 00:00:00:00:00:00
MAC 27: 00:00:00:00:00:00
MAC 28: 00:00:00:00:00:00
Top


Cisco Packet Tracer
File
Edit
Options
View
Tools
Extensions
Window
Help
Logical
Physical
[Root]
07:08:30
PC-PT PC0
WRT300N Wireless Router0
SMARTPHONE-PT Smartphone1
SMARTPHONE-PT Smartphone0
PC-PT PC1
Laptop-PT Laptop0
Laptop-PT Laptop1
Time: 00:13:54
Realtime
Simulation
Printer
IP Phone
VoIP Device
Phone
TV
Tablet
Smart Phone
Generic Wireless
Generic Wired
Sniffer
Scenario 0
New
Delete
Toggle PDU List Window
Fire
Last Status
SMARTPHONE-PT

RUTUPRIYA PURAWAT, B-18

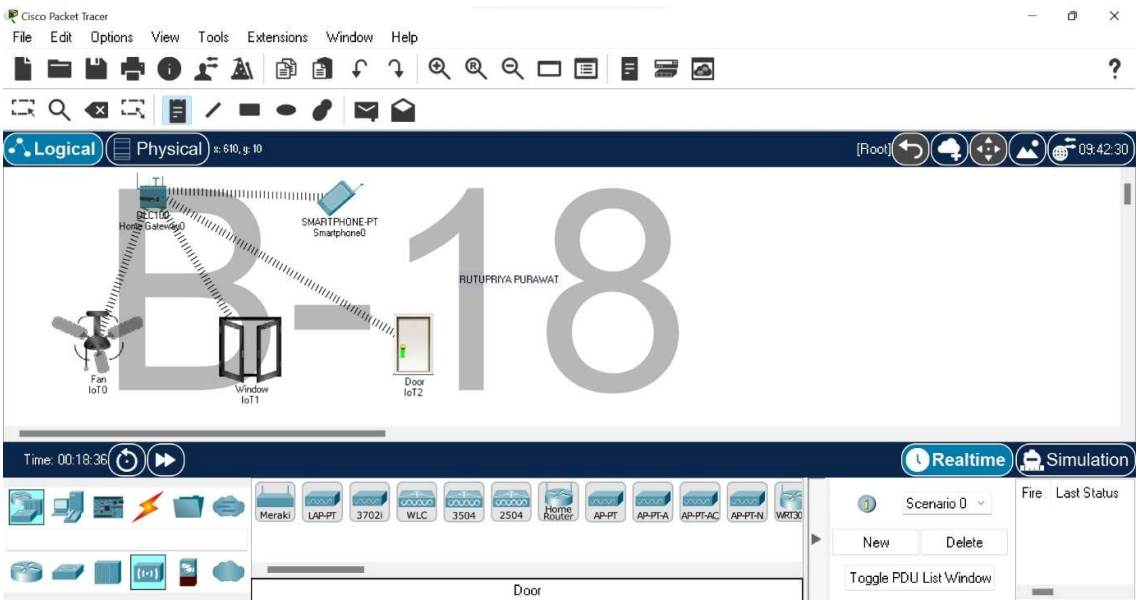
PRACTICAL 9

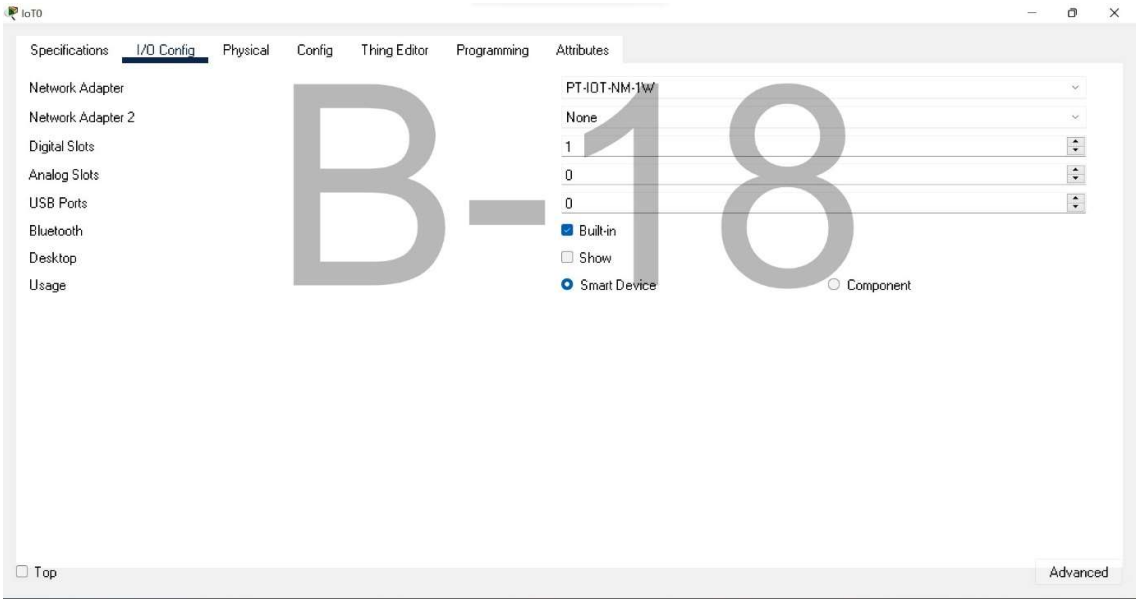
AIM: Simulate Mobile Adhoc Network with Directional Antenna.

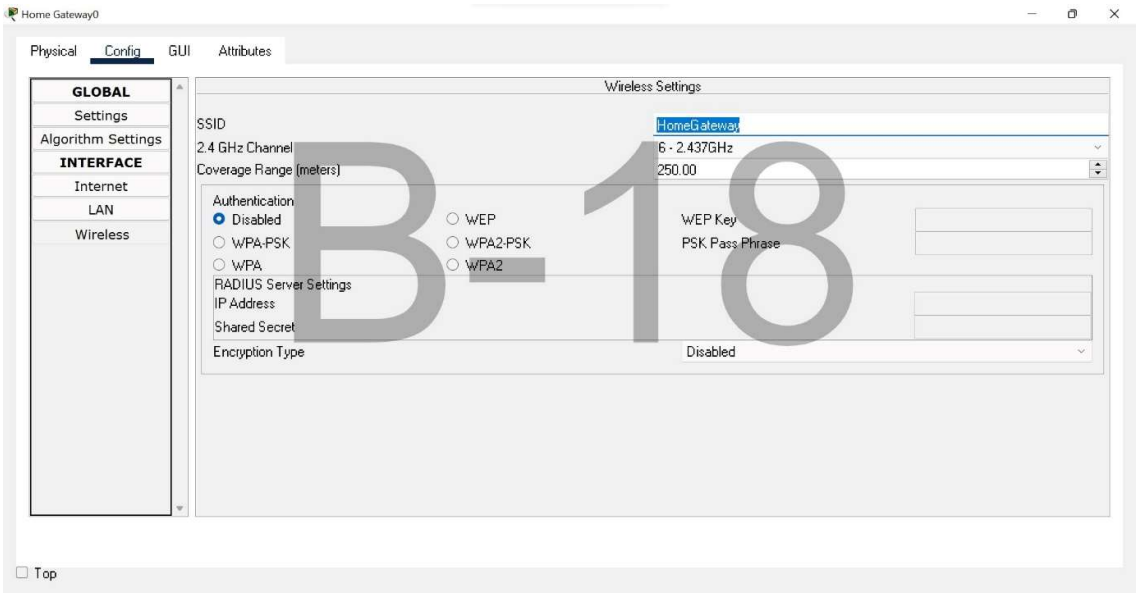
STEPS:

1. Create a network using Home gateway, and IOT devices
2. The Home Gateway provides 4 ethernet ports as well as a wireless access point configured with the "HomeGateway" ssid on channel 6. WEP / WPA-PSK / WPA2 enterprise can be configured to secure wireless connections. The picture below shows 4 IOE Things attached to a Home Gateway The Home Gateway is connected to the Internet through it's Internet WAN ethernet port.
3. For fan go to config -> Advance -> IO config -> network adapter:1W
4. Copy the SSID of home gateway (config -> wireless -> SSID)
5. In fan and window select home gateway in IO config
6. Go to smartphone -> desktop -> IOT monitor -> control the IOT devices.

EXECUTION AND OUTPUTS:




IoT0
Specifications
I/O Config
Physical
Config
Thing Editor
Programming
Attributes
Network Adapter
PT-IOT-NM-1W
Network Adapter 2
None
Digital Slots
1
Analog Slots
0
USB Ports
0
Bluetooth
Built-in
Desktop
Show
Usage
Smart Device
Component
Top
Advanced


Home Gateway0
Physical
Config
GUI
Attributes
GLOBAL
Settings
Algorithm Settings
INTERFACE
Internet
LAN
Wireless
Wireless Settings
SSID
HomeGateway
2.4 GHz Channel
6 - 2.437GHz
Coverage Range (meters)
250.00
Authentication
Disabled
WEP
WEP Key
WPA-PSK
WPA2-PSK
PSK Pass Phrase
WPA
WPA2
RADIUS Server Settings
IP Address
Shared Secret
Encryption Type
Disabled
Top


IoT0
Specifications
Physical
Config
Attributes
GLOBAL
Settings
Algorithm Settings
Files
INTERFACE
Wireless0
Bluetooth
Gateway/DNS IPv6
Automatic
Static
Default Gateway
DNS Server
IoT Server
None
Home Gateway
Remote Server
Server Address
User Name
Password
Refresh
Top
Advanced


Smartphone0
Physical
Config
Desktop
Programming
Attributes
IoT Monitor
IoT Server - Devices
Home | Conditions | Editor | Log Out
IoT0 (PTT0810823I-)
Ceiling Fan
Status
Off
Low
High
IoT1 (PTT0810WOAX-)
Window
On
IoT2 (PTT0810P1PZ-)
Door
Open
Lock
Unlock
Lock
Top

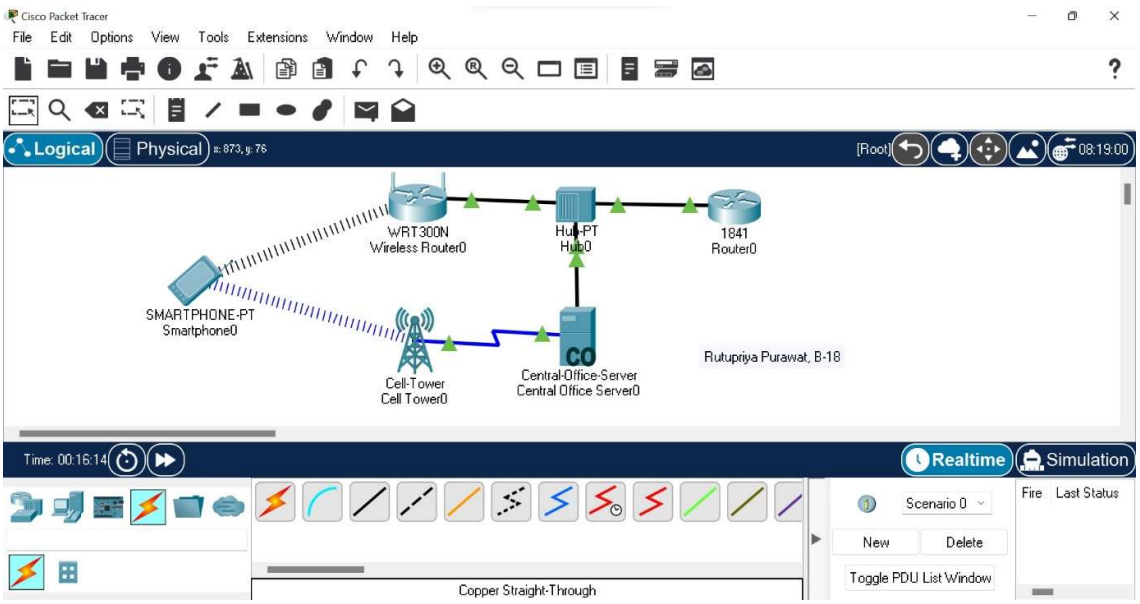
RUTUPRIYA PURAWAT, B-18

PRACTICAL 10

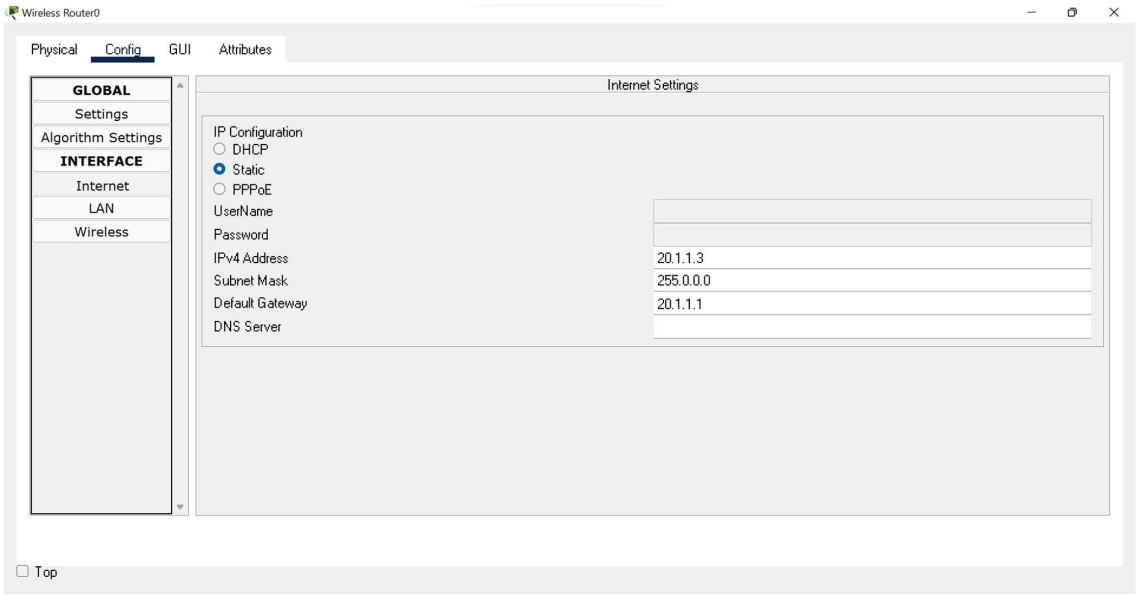
AIM: Create a mobile network using Cell Tower, Central Office Server, Web browser and Web Server. Simulate connection between them.

Steps:

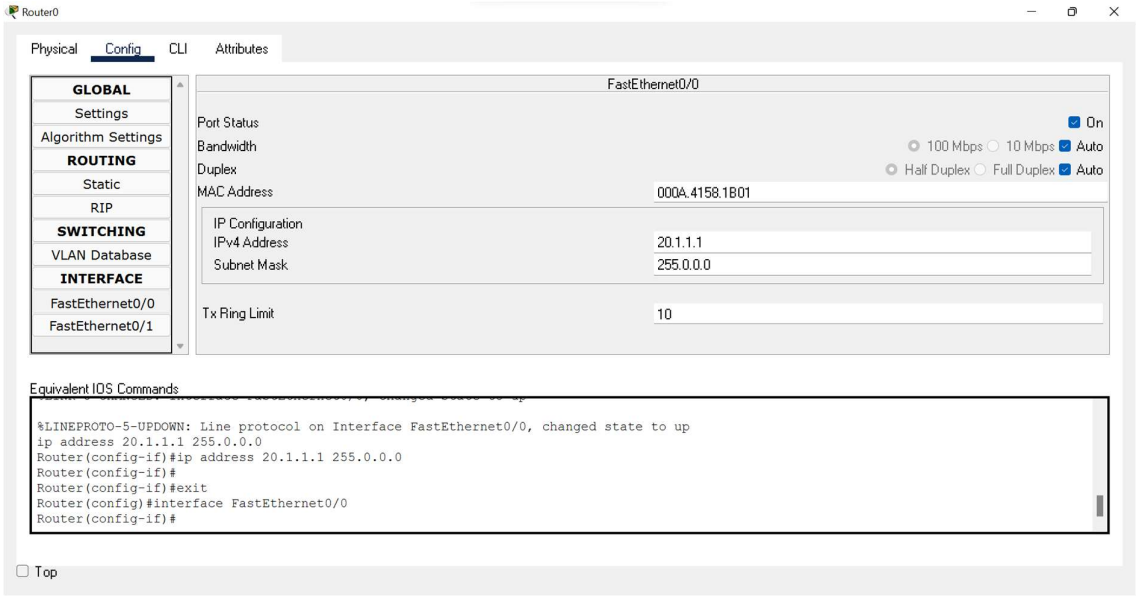
1. Create a network using smartphone, wireless router WRT300N, Hub-pt, 1841 Router, central-office-server, Cell-Tower.
2. Connect cell tower and central office server using coaxial cable.
3. Connect wireless router WRT300N, Hub-pt, 1841 Router, central-office-server using copper straight through wire.



4. Click on wireless router. In config tab, select internet. In internet, choose IP configuration as static and set IP address and default gateway.



5. Click on router 1841. In config tab select interface and give IP address.



6. Click on smartphone and ping router1841.