Laboratory Journal

Submitted in partial fulfilment of the requirement

for the Laboratory

‘Communication Networks Laboratory’

(Code – EC 314)

Prepared and Submitted by:

Mr. Prakhar Dubey

(Admission No.: U19EC009)

B.Tech III (EC), Semester - VI

(2021-22)

Laboratory Teacher: Dr. Raghavendra Pal

Department of Electronics Engineering

Sardar Vallabhbhai National Institute of Technology

Surat-395007, Gujarat, INDIA.

Sardar Vallabhbhai National Institute of Technology

Surat, Gujarat, INDIA

DEPARTMENT OF ELECTRONICS ENGINEERING

2021-22

Subject- Communication Networks Lab (Code – EC314)

Certificate

This is to certify that the Laboratory Journal is prepared & submitted by **B.Tech III (Semester-VI)** student Mr. Prakhar Dubey bearing **Admission No.** U19EC009 in the partial fulfilment of the requirement for the laboratory **Communication Networks Lab (Code-EC314)** through ONLINE MODE.

We, certify that the work is comprehensive, complete and fit for evaluation.

Laboratory Teacher:

Name	Signature with Date
1. Dr. Raghavendra Pal	

Jan-May 2022

Communication Networks Lab (Code – EC314)

Academic Year (2021-22)

LIST OF EXPERIMENTS

Sr. No.	List of Experiments	Page No.
1.	Introduction to TCP/IP Networking Commands.	
2.	Cyclic Redundancy Check (CRC) Method for Error Detection.	
3.	Hamming Code for Error Detection and Correction.	
4.	Bit Stuffing.	
5.	Shortest Path Routing Algorithm.	
6.	Symmetric Key Ciphering and Deciphering using Classical Ciphers.	
7.	Asymmetric Key Ciphering and Deciphering using Modern Ciphers.	
8.	Introduction to Network Simulator (NS3)	
9.	Elementary Network Model design using NS3	
10.	Dynamic Network Model design for Optimum Routing using NS3	
11.	Local Area Network Architecture and its Performance Analysis using NS3	
12.	Wireless Network Implementation using NS3.	

Submitted By

Name: Prakhar Dubey

Admission Number: U19EC009

Class/Year/Branch: B.Tech III ECED

Experiment 1

Date:

Aim: Introduction to TCP/IP Networking Commands.

What is TCP/IP?

The full form of TCP/IP is Transmission Control Protocol/Internet Protocol. It is a series of standard rules that enable computers to speak on a **network** like WiFi. A data link protocol, TCP/IP is utilized to allow devices and computers to send and receive data over the internet.

IP and TCP are 2 distinct protocols of computer networks. IP gets the address where the data is delivered and TCP is the part that delivers the data when that particular IP address has been found.

Let's consider the IP address as your contact number given to your phone and TCP is the technology that allows the phone to ring which allows you to talk to another person on the smartphone. Both are different but they are of no use without each other.

How does the TCP/IP work?

When you send a file, photo, or message over the internet, the TCP/IP classifies the information into packets as per the four-layer process. First, the data moves across these layers in one order and later in reverse order as the information is reconstructed on the other end. This model runs as the entire procedure is standardized. Otherwise, the communication will go wild and things will slow down and fast internet services depend on efficiency.

TCP/IP commands

TCP/IP is the key part of the fundamental structure of the system. It enables you to talk with another system just by implementing a program or command. Your system is responsible for what happened next.

Here are some of the TCP/IP commands and their functions described briefly. Some of the commands are used to configure and diagnose your network. Keep in mind while writing from the command swift you can only write one command on each line and press the "Enter" button after each command to implement it.

Arp: It sets, adds, deletes, and shows the IP address, status for all entries, and physical address in the ARP table.

Hostname: It is one of the simplest TCP/IP commands. It displays or sets the name of the existing host system.

Ping: It is the basic TCP/IP command, and it works the same as calling your friend. You take your phone; call your friend expecting that your friend will say "Hello" at the other end. In the same way, computers communicate with one another over the internet with the help of a Ping command. The function of this command is to place a call to some other computer over the network and asks for the answer. Ping command has two choices that it can use to make a call to another computer over the network. It can utilize the IP address or name of the computer.

Ipconfig (windows) / ifconfig (linux): This command shows the data related to the host computer TCP/IP configuration.

Ipconfig

Ipconfig/all: It shows the complete configuration data related to TCP/IP connection including gateway, router, DHCP, DNS, and Ethernet adapter in the system.

Ipconfig/renew: This command is used to renew or change the existing IP addresses that you are getting from the DHCP server. The renew command is a fast problem solver in case you are facing connection problems.

Chprtsv: It transforms a print service structure on a server or client machine.

Chnamsv: It changes TCP/IP-based name service structure on the host.

Hostent: This command straightforwardly changed address-mapping records in the structure configuration database

Nslookup: This command is utilized to diagnose DNS issues. If you can reach a source by sating an IP address and not its DNS, you will have a DNS issue.

Netstat: This command displays the TCP/IP protocol sessions, connections and open port connection information

Tracert: This command allows you to examine your TCP/IP system to check if a remote node is accessible or not and the trail utilized to reach it.

Ipconifg /release: The release command enables you to release the IP lease from the DHCP server.

Debug: Interrogates, enables, and disables the TCP/IP debug choice. Debug command has 2 choices i.e. TRACE, which follows the information flows across the TCP/IP software, and the DUMP, which inspects the information of different data schemes. The environment to be examined can be specified.

Display: This commands sets when specified or all ICMP messages are stated in the SUMLOG and at the ODT. You will have 3 message reporting choices i.e. FIRST, ALWAYS, or NEVER message at a particular time.

Display interval: This command defines the time that ICMP, Error, and Reset table data is mentioned to the SUMLOG. The default time is two hours.

Pathping: It is a unique command to windows and is generally a combo of Tracert and Ping commands. This command traces the route to the targeting address then initiates a 25-second test of each router throughout the way, collecting stats on the rate of lost data through each hop.

Route: This command shows the routing table of the computer. A representative computer having one network interface associated with a LAN, with a router is simple and usually doesn’t have any network issues. However, if you are having issues reaching your PC on the network, you can use this command to ensure that the entries in the routing table are right.

Conclusion:

Experiment 2

Date:

Aim: To perform cyclic Redundancy Check (CRC) Method for Error Detection.

Tools Required: MATLAB

Theory:

CRC or Cyclic Redundancy Check is a method of detecting accidental changes/errors in the communication channel.
CRC uses **Generator Polynomial** which is available on both sender and receiver side. An example generator polynomial is of the form like $x^3 + x + 1$. This generator polynomial represents key 1011. Another example is $x^2 + 1$ that represents key 101.

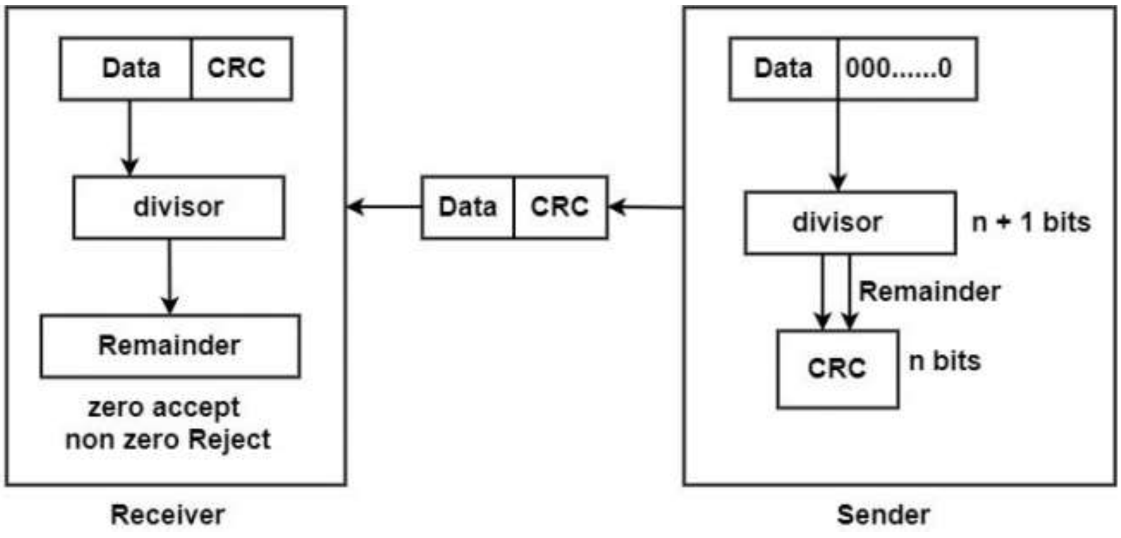
n : Number of bits in data to be sent from sender side.

k : Number of bits in the key obtained from generator polynomial.

Qualities of CRC

- It should have accurately one less bit than the divisor.
- Joining it to the end of the data unit should create the resulting bit sequence precisely divisible by the divisor.

CRC generator and checker



Sender Side (Generation of Encoded Data from Data and Generator Polynomial (or Key)):

1. The binary data is first augmented by adding k-1 zeros in the end of the data
2. Use ***modulo-2 binary division*** to divide binary data by the key and store remainder of division.

3. Append the remainder at the end of the data to form the encoded data and send the same

Receiver Side (Check if there are errors introduced in transmission)
Perform modulo-2 division again and if the remainder is 0, then there are no errors.
In this article we will focus only on finding the remainder i.e. check word and the code word.

Modulo 2 Division:
The process of modulo-2 binary division is the same as the familiar division process we use for decimal numbers. Just that instead of subtraction, we use XOR here.

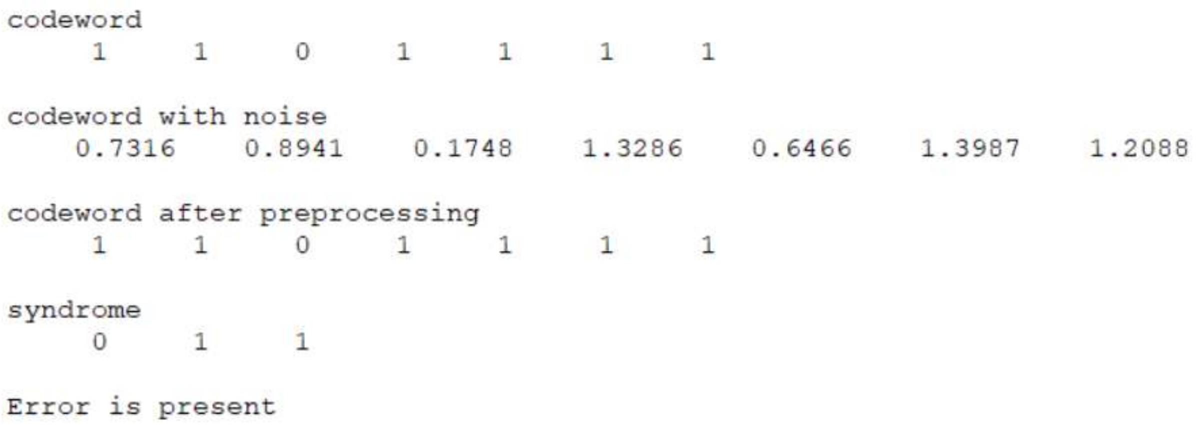
- In each step, a copy of the divisor (or data) is XORed with the k bits of the dividend (or key).
- The result of the XOR operation (remainder) is (n-1) bits, which is used for the next step after 1 extra bit is pulled down to make it n bits long.
- When there are no bits left to pull down, we have a result. The (n-1)-bit remainder which is appended at the sender side.

Code:

```
clear all;
close all;
clc;

d_w=[1 1 0 1];
div=[1 0 1 0];

%% Transmitter

d_w_length=length(d_w);
div_length=length(div);
remainder=d_w(1:div_length);

for i=(div_length):(d_w_length)
    if (remainder(1) == div(1))
        remainder=bitxor(remainder,div);
    end
    remainder = remainder(2:div_length);
    if (i ~= d_w_length)
        remainder = [remainder d_w(i+1)];
    end
end

c_w= [d_w remainder];
disp('codeword')
disp(c_w);

%% channel noise

c_w=awgn(c_w,10);
disp('codeword with noise')
disp(c_w);
%% reciever preprocess
```

```
c_w=round(c_w);
c_w= c_w > 0;
disp('codeword after preprocessing')
disp(c_w);
%% reciever check

c_w_length=length(c_w);
syndrome=c_w(1:div_length);

for i=(div_length):(c_w_length)
    if (syndrome(1) == div(1))
        syndrome=bitxor(syndrome,div);
    end
    syndrome = syndrome(2:div_length);
    if (i ~= c_w_length)
        syndrome = [syndrome c_w(i+1)];
    end
end
disp('syndrome')
disp(syndrome);

for i=1:length(syndrome)
    if(syndrome(i)==1)
        disp('Error is present');
        break;
    elseif (i==length(syndrome))
        disp('Error is not present');
    end
end
```

Output:



```
codeword
     1     1     0     1     1     1     1

codeword with noise
    0.7316    0.8941    0.1748    1.3286    0.6466    1.3987    1.2088

codeword after preprocessing
     1     1     0     1     1     1     1

syndrome
     0     1     1

Error is present
```

Conclusion:

Experiment 3

Date:

Aim: Use Hamming Codes for Error Detection and Correction.

Tools Required: MATLAB

Theory:

When bits are transmitted over the computer network, they are subject to get corrupted due to interference and network problems. The corrupted bits leads to spurious data being received by the receiver and are called errors.

Error-correcting codes (ECC) are a sequence of numbers generated by specific algorithms for detecting and removing errors in data that has been transmitted over noisy channels. Error correcting codes ascertain the exact number of bits that has been corrupted and the location of the corrupted bits, within the limitations in algorithm.

ECCs can be broadly categorized into two types −

- **Block codes** − The message is divided into fixed-sized blocks of bits, to which redundant bits are added for error detection or correction.
- **Convolutional codes** − The message comprises of data streams of arbitrary length and parity symbols are generated by the sliding application of a Boolean function to the data stream.

Hamming Code

Hamming code is a block code that is capable of detecting up to two simultaneous bit errors and correcting single-bit errors. It was developed by R.W. Hamming for error correction.

In this coding method, the source encodes the message by inserting redundant bits within the message. These redundant bits are extra bits that are generated and inserted at specific positions in the message itself to enable error detection and correction. When the destination receives this message, it performs recalculations to detect errors and find the bit position that has error.

Hamming code is a set of error-correction codes that can be used to **detect and correct the errors** that can occur when the data is moved or stored from the sender to the receiver. It is **technique developed by R.W. Hamming for error correction**.

Redundant bits –
Redundant bits are extra binary bits that are generated and added to the information-carrying bits of data transfer to ensure that no bits were lost during the data transfer. The number of redundant bits can be calculated using the following formula:

$2^r \geq m + r + 1$

where, r = redundant bit, m = data bit

Parity bits –
A parity bit is a bit appended to a data of binary bits to ensure that the total number of 1's in the data is even or odd. Parity bits are used for error detection. There are two types of parity bits:

1. **Even parity bit:**
   In the case of even parity, for a given set of bits, the number of 1's are counted. If that count is odd, the parity bit value is set to 1, making the total count of occurrences of 1's an even number. If the total number of 1's in a given set of bits is already even, the parity bit's value is 0.
2. **Odd Parity bit –**
   In the case of odd parity, for a given set of bits, the number of 1's are counted. If that count is even, the parity bit value is set to 1, making the total count of occurrences of 1's an odd number. If the total number of 1's in a given set of bits is already odd, the parity bit's value is 0.

General Algorithm of Hamming code –
The Hamming Code is simply the use of extra parity bits to allow the identification of an error.

1. Write the bit positions starting from 1 in binary form.
2. All the bit positions that are a power of 2 are marked as parity bits.
3. All the other bit positions are marked as data bits.
4. Each data bit is included in a unique set of parity bits, as determined its bit position in binary form.

   In general, each parity bit covers all bits where the bitwise AND of the parity position and the bit position is non-zero.
5. Since we check for even parity set a parity bit to 1 if the total number of ones in the positions it checks is odd.
6. Set a parity bit to 0 if the total number of ones in the positions it checks is even.

Code:

```
clc;
close all;
clear all;

% message bits
m = [0 1 0 1];
n = length(m);
r = 1;

while (power(2,r) < (n+r+1))
    r = r+1;
end
```

```
% Final result array
res = zeros(1, n+r);

% Find positions of redundant bits
for i = 0:1:r-1
    res(power(2, i)) = -1;
end

j=1;

for i=1:1:r+n-1
    if(res(i) ~= -1)
        res(i) = m(j);
        j = j+1;
    end
end

for i=1:1:r+n-1
    if(res(i) == -1)
        x = log2(i);
        ones = 0;

        for j=i+2:1:r+n-1
            if(j && bitsll(1, x))
                if(res(j-1)==1)
                    ones = ones+1;
                end
            end
        end

        if(mod(ones,2)==0)
            res(i) = 0;
        else
            res(i) = 1;
        end
    end
end

disp(res);
```

Output

```
     1     1     0     1     1     0     0
```

Conclusion:

Experiment 4

Date:

Aim: To perform bit stuffing and destuffing.

Tools Required: MATLAB

Theory:

Data link layer is responsible for something called Framing, which is the division of stream of bits from network layer into manageable units (called frames). Frames could be of fixed size or variable size. In variable-size framing, we need a way to define the end of the frame and the beginning of the next frame. However, if the pattern occurs in the message, then mechanisms needs to be incorporated so that this situation is avoided.

The two common approaches are −

- **Byte - Stuffing** − A byte is stuffed in the message to differentiate from the delimiter. This is also called character-oriented framing.
- **Bit - Stuffing** − A pattern of bits of arbitrary length is stuffed in the message to differentiate from the delimiter. This is also called bit - oriented framing.

Bit stuffing is the mechanism of inserting one or more non-information bits into a message to be transmitted, to break up the message sequence, for synchronization purpose.

Note that stuffed bits should not be confused with overhead bits. **Overhead bits** are non-data bits that are necessary for transmission (usually as part of headers, checksums etc.).

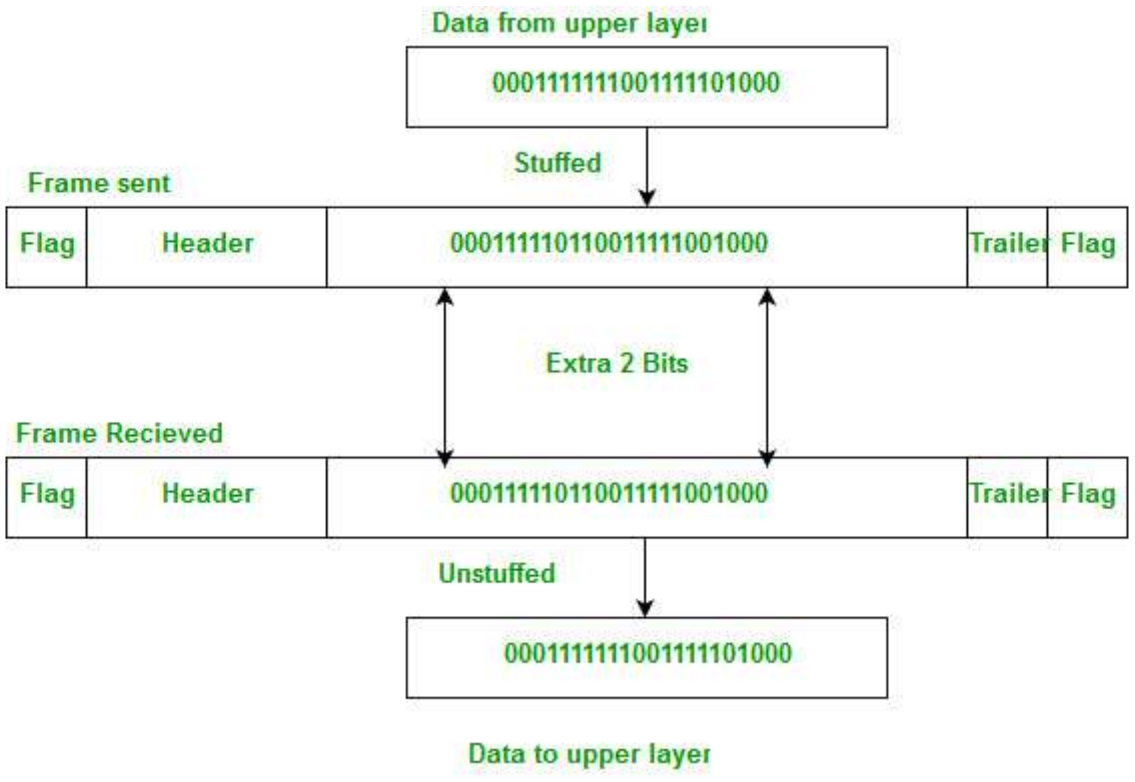
Bit Stuffing Mechanism

In a data link frame, the delimiting flag sequence generally contains six or more consecutive 1s. In order to differentiate the message from the flag in case of the same sequence, a single bit is stuffed in the message. Whenever a 0 bit is followed by five consecutive 1bits in the message, an extra 0 bit is stuffed at the end of the five 1s.

When the receiver receives the message, it removes the stuffed 0s after each sequence of five 1s. The un-stuffed message is then sent to the upper layers.

Applications of Bit Stuffing –

1. synchronize several channels before multiplexing
2. rate-match two single channels to each other
3. run length limited coding



Run length limited coding – To limit the number of consecutive bits of the same value(i.e., binary value) in the data to be transmitted. A bit of the opposite value is inserted after the maximum allowed number of consecutive bits.

Bit stuffing technique does not ensure that the sent data is intact at the receiver side (i.e., not corrupted by transmission errors). It is merely a way to ensure that the transmission starts and ends at the correct places.

Disadvantages of Bit Stuffing –
The code rate is unpredictable; it depends on the data being transmitted.

Code:

1. Bit Stuffing

```
clc;
clear all;
close all;

%msg=[ 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 ]
%Unstuffed
%msg=[ 1 0 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 0]
%Stuffed

msg=input('Input Message binary bit stream :');
count=0;
stuffcount=0;
[M, N]=size(msg);
for j=1:N-5+stuffcount
```

```
        j;
        for i=j:j+5
            i;
            if msg(i)==1
                count=count+1;
            else
                count=0;
                break;
            end
        end
            if(count ==6)
                msg=[msg(1:j+4) 0 msg(j+5 : end)];
                count=0;
                stuffcount=stuffcount+1;
            end
    end
    disp(msg);
```

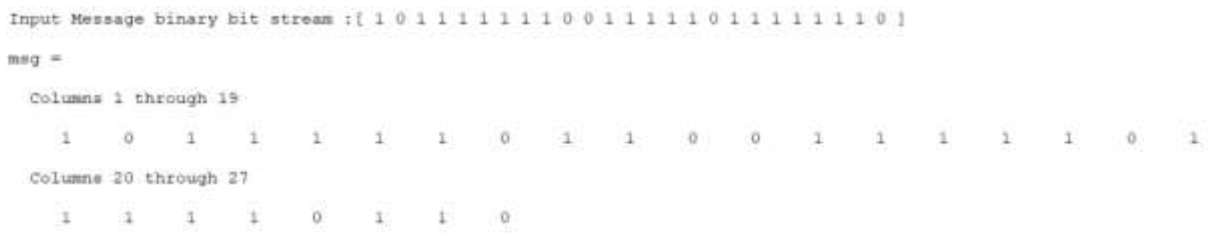
2. Bit Destuffing

```
    clc;
    clear all;
    close all;
    msg=input('Input Message binary bit stream :');
    % msg=[ 1 0 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1
    0 ];
    %umsg=[ 1 0 1 1 1 1 1 % 1 1 0 0 1 1 1 1 1 % 0 1 1 1 1 1 % 1 1
    0 ];

    umsg =msg;
    ct=0;
    cts=0;
    N=length(msg);
    for j=1:N-5-cts
        ct=0;
        for i=j:j+4
            disp(i)
            disp(msg(i))
            if msg(i)==0
                break;
            else
                ct=ct+1;
            end
        end
        disp('ct')
        disp(ct);
        if ct==5
            if msg(i+1)==0
                disp('test')
                umsg(i+1-cts) = [];
                cts=cts+1;
                j=i+1;
            end
        end
```

```
end
msg
umsg
```

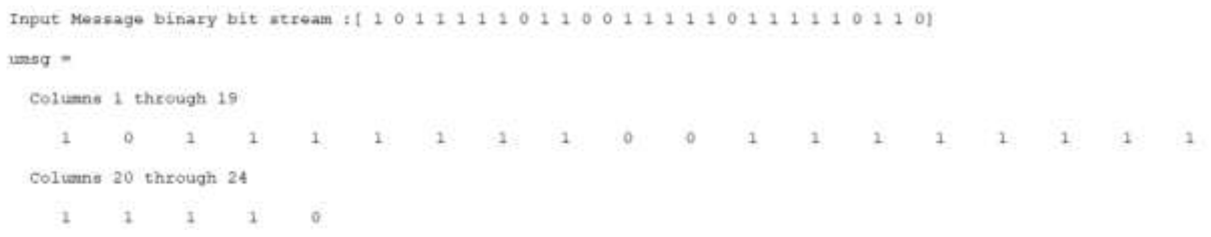
Output:

1. **Bit Stuffing**



```
Input Message binary bit stream :[ 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 ]

msg =

  Columns 1 through 19

     1     0     1     1     1     1     1     0     1     1     0     0     1     1     1     1     1     0     1

  Columns 20 through 27

     1     1     1     1     0     1     1     0
```

2. **Bit Destuffing**



```
Input Message binary bit stream :[ 1 0 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 0]

umsg =

  Columns 1 through 19

     1     0     1     1     1     1     1     1     1     0     0     1     1     1     1     1     1     1     1

  Columns 20 through 24

     1     1     1     1     0
```

Conclusion:

Experiment 5

Date:

Aim: To perform shortest path routing algorithm.

Tools Required: MATLAB

Theory:

In computer networks, the shortest path algorithms aim to find the optimal paths between the network nodes so that routing cost is minimized. They are direct applications of the shortest path algorithms proposed in graph theory.

Explanation

Consider that a network comprises of N vertices (nodes or network devices) that are connected by M edges (transmission lines). Each edge is associated with a weight, representing the physical distance or the transmission delay of the transmission line. The target of shortest path algorithms is to find a route between any pair of vertices along the edges, so the sum of weights of edges is minimum. If the edges are of equal weights, the shortest path algorithm aims to find a route having minimum number of hops.

Common Shortest Path Algorithms

Some common shortest path algorithms are −

- Bellman Ford’s Algorithm
- Dijkstra’s Algorithm
- Floyd Warshall’s Algorithm

Dijkstra’s Algorithm

Input − A graph representing the network; and a source node, s

Output − A shortest path tree, spt[], with s as the root node.

Basics of Dijkstra's Algorithm

- Dijkstra's Algorithm basically starts at the node that you choose (the source node) and it analyzes the graph to find the shortest path between that node and all the other nodes in the graph.
- The algorithm keeps track of the currently known shortest distance from each node to the source node and it updates these values if it finds a shorter path.
- Once the algorithm has found the shortest path between the source node and another node, that node is marked as "visited" and added to the path.

- The process continues until all the nodes in the graph have been added to the path. This way, we have a path that connects the source node to all other nodes following the shortest path possible to reach each node.

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.

The algorithm exists in many variants. Dijkstra's original algorithm found the shortest path between two given nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree. For a given source node in the graph, the algorithm finds the shortest path between that node and every other.

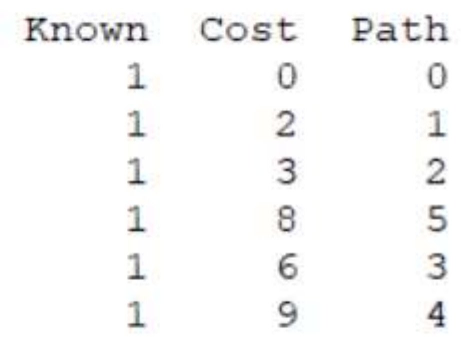
It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined. For example, if the nodes of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. A widely used application of shortest path algorithms is network routing protocols, most notably IS-IS (Intermediate System to Intermediate System) and Open Shortest Path First (OSPF).

Code:

```
clc;
clear all;
close all;

graph = [
     [0 2 4 0 0 0];
     [2 0 1 7 0 0];
     [4 1 0 0 3 0];
     [0 7 0 0 2 1];
     [0 0 3 2 0 5];
     [0 0 0 1 5 0];
];

%% Print Graph
i=1;
x=1;
temp=triu(graph); %for coloring of the lines after computing
path

for i=1:6
     for j=1:6
          if(temp(i,j)~=0)
               lc(i,j)=x;      %just numbering in sequence for
line count
               lc(j,i)=lc(i,j);
```

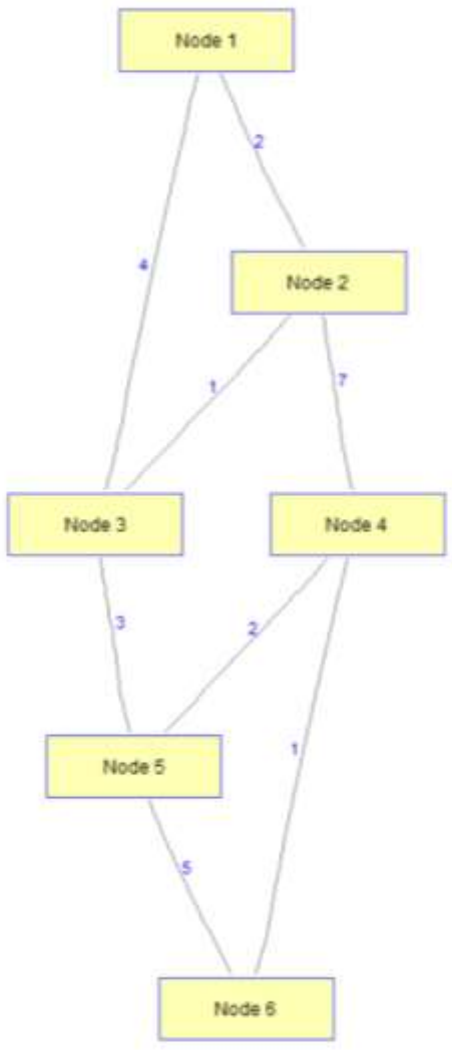
```
             x=x+1;
         end
     end
end

bg=biograph(triu(graph),[],'showarrows','off','ShowWeights','o
n','EdgeTextColor',[0 0 1]);
view(bg);

%% Shortest path
cost = Inf(1, 6);
known = false(1, 6);
path = zeros(1, 6);
cost(1) = 0;

for i=1:6
    disp([' Known ', ' Cost ', ' Path']);
    disp([known; cost; path]');
    min_val = 1/0;
    vertex=0;
    for j=1:6
        if((~known(j)) && (cost(j) < min_val))
            min_val = cost(j);
            vertex = j;
        end
    end

    known(vertex) = 1;

    for j=1:6
        if((~known(j)) && (graph(vertex,j) ~= 0))
            if(graph(vertex, j) + cost(vertex) < cost(j))
                cost(j) = graph(vertex, j) + cost(vertex);
                path(j) = vertex;
            end
        end
    end
end
disp([' Known ', ' Cost ', ' Path']);
disp([known; cost; path]');
```

Output



```
 Known   Cost   Path
     1      0      0
     1      2      1
     1      3      2
     1      8      5
     1      6      3
     1      9      4
```

Graph



Conclusion:

Thus, we implemented Dijkstra Algorithm to find the shortest path with source vertex 2, given 7 node undirected graph. The path and minimum cost to reach respective nodes can be found using this algorithm.

Experiment 6

Date:

Aim: To perform Symmetric Key Ciphering and Deciphering using Classical Ciphers.

Tools Required: MATLAB

Theory:

Symmetric key cryptography is any cryptographic algorithm that is based on a shared key that is used to encrypt or decrypt text/cyphertext, in contract to asymmetric key cryptography, where the encryption and decryption keys are different.

Symmetric encryption is generally more efficient than asymmetric encryption and therefore preferred when large amounts of data need to be exchanged.

Establishing the shared key is difficult using only symmetric encryption algorithms, so in many cases, an asymmetric encryption is used to establish the shared key between two parties.

Examples for symmetric key cryptography include AES, DES, and 3DES.

XOR Encryption

It is an encryption method used to encrypt data and is hard to crack by brute-force method, i.e generating random encryption keys to match with the correct one. The XOR Encryption algorithm is a very effective yet easy to implement method of symmetric encryption. Due to its effectiveness and simplicity, the XOR Encryption is an extremely common component used in more complex encryption algorithms used nowadays. In cryptography, the simple XOR cipher is a type of *additive cipher*, an encryption algorithm that operates according to the principles:

A XOR 0 = A,
A XOR A = 0,
A XOR B = B XOR A,
(A XOR B) XOR C = A XOR (B XOR C),
(B XOR A) XOR A = B XOR 0 = B

where `XOR` denotes the exclusive disjunction (XOR) operation. This operation is sometimes called modulus 2 addition (or subtraction, which is identical). With this logic, a string of text can be encrypted by applying the bitwise XOR operator to every character using a given key. To decrypt the output, merely reapplying the XOR function with the key will remove the cipher.

The XOR operator is extremely common as a component in more complex ciphers. By itself, using a constant repeating key, a simple XOR cipher can trivially be broken using frequency analysis. If the content of any message can be guessed or otherwise known then the key can be revealed. Its primary merit is that it is simple to implement, and that the XOR operation is computationally inexpensive. A simple repeating XOR (i.e. using the same key for xor operation on the whole data) cipher is therefore sometimes used for hiding information in cases where no particular security is required. The XOR cipher is often used in computer malware to make reverse engineering more difficult.

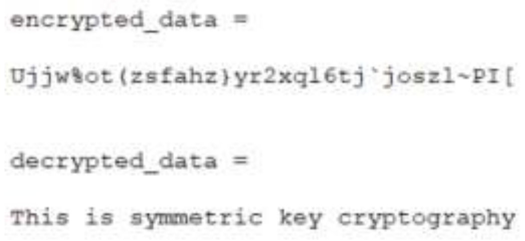
If the key is random and is at least as long as the message, the XOR cipher is much more secure than when there is key repetition within a message. When the keystream is generated by a pseudo-random number generator, the result is a stream cipher. With a key that is truly random, the result is a one-time pad, which is unbreakable in theory.

The XOR operator in any of these ciphers is vulnerable to a known-plaintext attack, since *plaintext* ^ *ciphertext* = *key*. It is also trivial to flip arbitrary bits in the decrypted plaintext by manipulating the ciphertext. This is called malleability.

Code:

```
clc;
close all;
clear all;
file=fileread('plaintext.txt');
n=length(file);
for i=1:1:n
    encript(i)=bitxor(int32(file(i)),i);    %Xoring to encript
data
end
encripted_data=char(encript);
display(encripted_data);
n=length(encripted_data);
for i=1:1:n
decript(i)=bitxor(int32(encripted_data(i)),i);  %Xoring again
to decript data
end
decripted_data=char(decript);
display(decripted_data);
```

Output



```
encrypted_data =

Ujjw%ot(zsfahz)yr2xql6tj`joszl~PI[


decrypted_data =

This is symmetric key cryptography
```

Conclusion:

Experiment 7

Date:

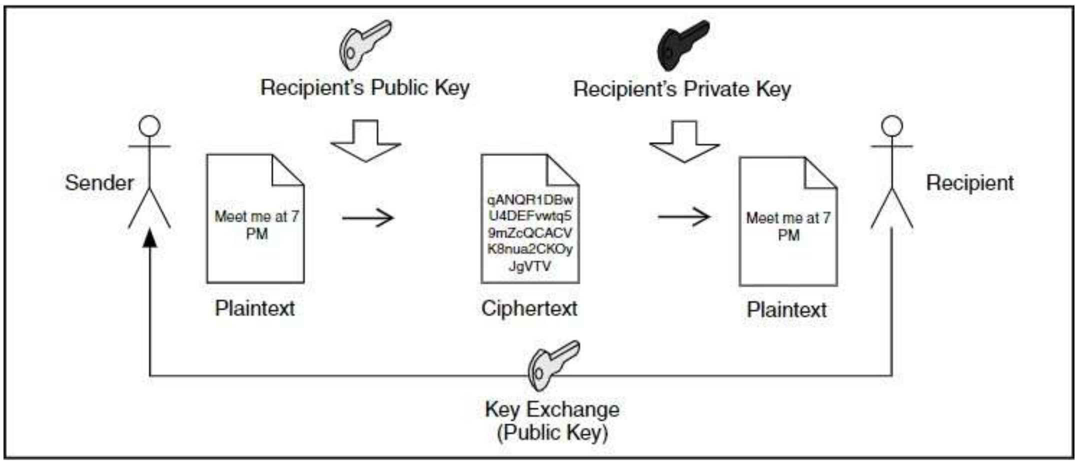
Aim: To perform Asymmetric Key Ciphering and Deciphering using Modern Ciphers.

Tools Required: MATLAB

Theory:

Asymmetric key cryptosystems, also known as public key cryptosystems, is more modern practice than the symmetric key cryptosystems. In asymmetric key cryptography, the decryption key is different than the key that was used to encrypt the message. Each party taking part in the communication stream in an asymmetric key cryptosystem has a pair of keys, known as the public key and the private key. The private key is highly protected and is only known to the owner of that key. The public key is distributed to other entities that take part in the communication.When the goal is to achieve confidentiality, a recipient’s public key is used to encrypt messages. On receiving the message, the recipient is able to decrypt the message using his or her private key.

In an asymmetric key cryptosystem, a public key cannot be used to derive the private key.In addition to confidentiality, asymmetric key cryptosystems can also be used to achieve user authentication and non-repudiation aspects of information security. A message encrypted using a private key can be decrypted using the corresponding public key. As the public key may be common-ly known, this would not provide confidentiality. However, since the private key is only known to the sender, it authenticates the sender, and the sender cannot deny that he or she sent the message. Fig. 2.5 shows an example of how an asymmetric key cryptosystem works.



The recipient distributes his or her public key to entities wishing to secure their communications with the recipient. Unlike in the symmetric key cryptosystem, the secrecy of the key is not a requirement as it can only be used to encrypt messages. In fact, many asymmetric key cryptosystems have public key servers where users can publish their public keys. Using the recipient’s pubic key, the sender now encrypts the message. When the recipient receives the message, he or she can decrypt and access the message using his or her private key, which has never been released to any other party.

The most important properties of public key encryption scheme are −

- Different keys are used for encryption and decryption. This is a property which set this scheme different than symmetric encryption scheme.
- Each receiver possesses a unique decryption key, generally referred to as his private key.
- Receiver needs to publish an encryption key, referred to as his public key.
- Some assurance of the authenticity of a public key is needed in this scheme to avoid spoofing by adversary as the receiver. Generally, this type of cryptosystem involves trusted third party which certifies that a particular public key belongs to a specific person or entity only.
- Encryption algorithm is complex enough to prohibit attacker from deducing the plaintext from the ciphertext and the encryption (public) key.
- Though private and public keys are related mathematically, it is not be feasible to calculate the private key from the public key. In fact, intelligent part of any public-key cryptosystem is in designing a relationship between two keys.

RSA Cryptosystem

This cryptosystem is one the initial system. It remains most employed cryptosystem even today. The system was invented by three scholars **Ron Rivest, Adi Shamir,** and **Len Adleman** and hence, it is termed as RSA cryptosystem.

There are two important aspects of the RSA cryptosystem, firstly generation of key pair and secondly encryption-decryption algorithms.

Generation of RSA Key Pair

Each person or a party who desires to participate in communication using encryption needs to generate a pair of keys, namely public key and private key. The process followed in the generation of keys is described below −

- RSA modulus (n)
    - Two large primes, p and q are selected and n=p*q is calculated where n is a large number, typically a minimum of 512 bits.
- Derived Number (e)
    - Number **e** must be greater than 1 and less than (p − 1)(q − 1).
    - There must be no common factor for e and (p − 1)(q − 1) except for 1
- Form the public key
    - The pair of numbers (n, e) form the RSA public key and is made public.
- Generate the private key
    - Private Key d is calculated from p, q, and e. For given n and e, there is unique number d such that d is the inverse of e modulo (p - 1)(q – 1).

- This relationship is written mathematically as follows −

$$ed = 1 \bmod (p − 1)(q − 1)$$

The Extended Euclidean Algorithm takes p, q, and e as input and gives d as output.

Encryption and Decryption

RSA Encryption

- The sender has public key as (n, e).
- To encrypt the first plaintext P, which is a number modulo n. −

$$C = P^e \bmod n$$

- The ciphertext C is equal to the plaintext P multiplied by itself e times and then reduced modulo n. This means that C is also a number less than n.

RSA Decryption

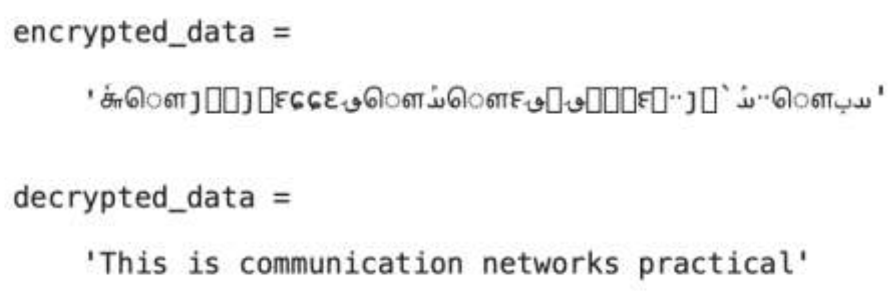
- The receiver has public-key pair (n, e) and has received a ciphertext C.
- Receiver raises C to the power of his private key d. The result modulo n will be the plaintext P.

$$P = C^d \bmod n$$

Code:

```
close all;
text = 'This is communication networks practical';
p = 61;
q = 53;
n = p*q;
phi = lcm((p-1),(q-1));
e=7;
d=1;
while 1
    if mod(e*d, phi) ==1
        break
    end
    d=d+1;
end
text = double(text);
encrypt=powermod(text,e,n);
encrypted_data = char(encrypt);
display(encrypted_data);
t=double(encrypted_data);
decrypt=powermod(t,d,n);
decrypted_data=char(decrypt);
display(decrypted_data);
```

Output



```
encrypted_data =
    'கூ๑ொளյ□□յ□ϵϛϛɛفொளئொளϵف□ف□□□ϵ□¨յ□`ئ¨ொளبد'

decrypted_data =
    'This is communication networks practical'
```

Conclusion:

Experiment 8

Date:

Aim: Introduction to Network Simulator 3 (NS3)

Tools Required: Oracle Virtual Machine, Ubuntu 20.04 LTS

Theory:

Network simulator is a tool used for simulating the real world network on one computer by writing scripts in C++ or Python. Normally if we want to perform experiments, to see how our network works using various parameters. We don't have required number of computers and routers for making different topologies. Even if we have these resources it is very expensive to build such a network for experiment purposes.

So to overcome these drawbacks we used NS3, which is a discrete event network simulator for Internet. NS3 helps to create various virtual nodes (i.e., computers in real life) and with the help of various Helper classes it allows us to install devices, internet stacks, application, etc to our nodes.

Using NS3 we can create PointToPoint, Wireless, CSMA, etc connections between nodes. PointToPoint connection is same as a LAN connected between two computers. Wireless connection is same as WiFi connection between various computers and routers. CSMA connection is same as bus topology between computers.

Code for first.cc:

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program is free software; you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-
1307  USA
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
```

```
#include "ns3/applications-module.h"

// Default Network Topology
//
//       10.1.1.0
// n0 -------------- n1
//    point-to-point
//

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  CommandLine cmd (__FILE__);
  cmd.Parse (argc, argv);

  Time::SetResolution (Time::NS);
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute      ("DataRate",      StringValue
("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

  NetDeviceContainer devices;
  devices = pointToPoint.Install (nodes);

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer  serverApps  =  echoServer.Install  (nodes.Get
(1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer  clientApps  =  echoClient.Install  (nodes.Get
(0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

Command Description:

NodeContainer: Initialise a list of nodes and creates the nodes using ".Create (size)" command.

Create a **NodeContainer** with the requested number of Nodes.

This is syntatic sugar for

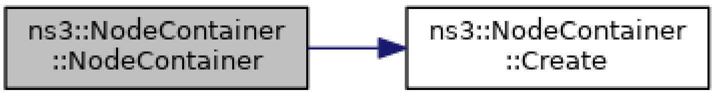
```
NodeContainer nodes;
nodes.Create (size);
// or nodes.Create (size, systemId);
```

Parameters

[in] **n**	The number of nodes to create.
[in] **systemId**	The system id or rank associated with this node

Definition at line **39** of file **node-container.cc**.

References **Create()**, and **m_nodes**.


ns3::NodeContainer ::NodeContainer
ns3::NodeContainer ::Create

PointToPointHelper: Used to initialise network with point to point link.

Build a set of **PointToPointNetDevice** objects.

Normally we eschew multiple inheritance, however, the classes PcapUserHelperForDevice and AsciiTraceUserHelperForDevice are "mixins".

Definition at line **45** of file **point-to-point-helper.h**.

ns3::PointToPointHelper::PointToPointHelper

Create a **PointToPointHelper** to make life easier when creating point to point networks.

Definition at line **41** of file **point-to-point-helper.cc**.

References **m_channelFactory**, **m_deviceFactory**, **m_queueFactory**, **m_remoteChannelFactory**, and **ns3::ObjectFactory::SetTypeId()**.

NetDeviceContainer: Used to install devices that were instantiated previously.

holds a vector of **ns3::NetDevice** pointers

Typically ns-3 NetDevices are installed on nodes using a net device helper. The helper Install method takes a **NodeContainer** which holds some number of **Ptr<Node>**. For each of the Nodes in the **NodeContainer** the helper will instantiate a net device, add a MAC address and a queue to the device and install it to the node. For each of the devices, the helper also adds the device into a Container for later use by the caller. This is that container used to hold the **Ptr<NetDevice>** which are instantiated by the device helper.

Definition at line **41** of file **net-device-container.h**.

ns3::NetDeviceContainer::NetDeviceContainer

Create a **NetDeviceContainer** with exactly one device which has been previously instantiated and assigned a name using the **Object** name service.

This **NetDevice** is specified by its assigned name.

Parameters

devName The name of the device to add to the container

Create a **NetDeviceContainer** with exactly one device

Definition at line **33** of file **net-device-container.cc**.

References **m_devices**.

InternetStackHelper: Aggregate IP/TCP/UDP functionality to existing nodes.

aggregate IP/TCP/UDP functionality to existing Nodes.

This helper enables pcap and ascii tracing of events in the internet stack associated with a node. This is substantially similar to the tracing that happens in device helpers, but the important difference is that, well, there is no device. This means that the creation of output file names will change, and also the user-visible methods will not reference devices and therefore the number of trace enable methods is reduced.

Normally we avoid multiple inheritance in ns-3, however, the classes PcapUserHelperForIpv4 and AsciiTraceUserHelperForIpv4 are treated as "mixins". A mixin is a self-contained class that encapsulates a general attribute or a set of functionality that may be of interest to many other classes.

This class aggregates instances of these objects, by default, to each node:

- **ns3::ArpL3Protocol**
- **ns3::Ipv4L3Protocol**
- **ns3::Icmpv4L4Protocol**
- **ns3::Ipv6L3Protocol**
- **ns3::Icmpv6L4Protocol**
- **ns3::UdpL4Protocol**
- **ns3::TrafficControlLayer**
- a TCP based on the TCP factory provided
- a **PacketSocketFactory**
- **Ipv4** routing (a list routing object, a global routing object, and a static routing object)
- **Ipv6** routing (a static routing object)

Definition at line **86** of file **internet-stack-helper.h**

UdpEchoServerHelper: Create a server application which waits for input UDP packets and

Create **UdpEchoServerHelper** which will make life easier for people trying to set up simulations with echos.

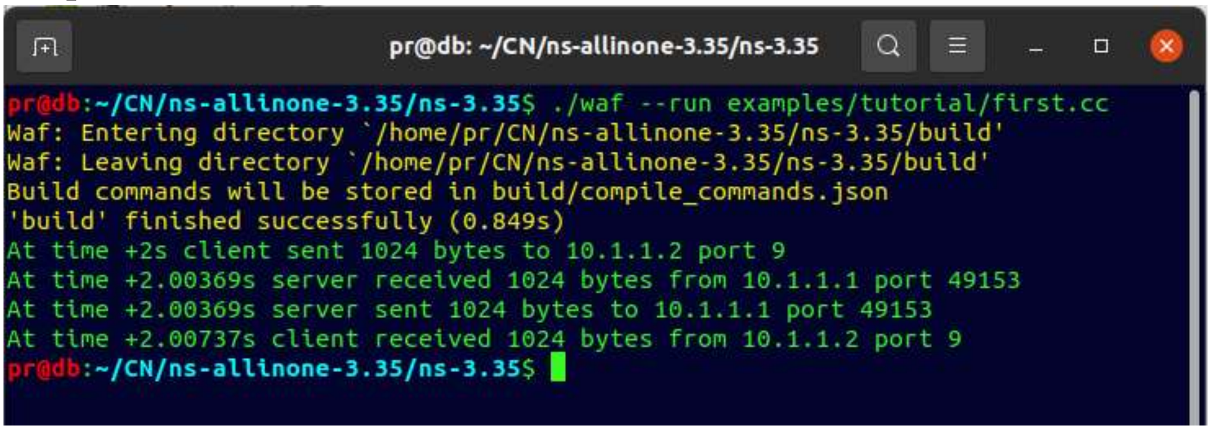
Parameters

port The port the server will wait on for incoming packets

Definition at line **28** of file **udp-echo-helper.cc**.

References **ns3::UdpEchoServer::GetTypeId()**, **m_factory**, **port**, **SetAttribute()**, and **ns3::ObjectFactory::SetTypeId()**.

Output



```
pr@db:~/CN/ns-allinone-3.35/ns-3.35$ ./waf --run examples/tutorial/first.cc
Waf: Entering directory `/home/pr/CN/ns-allinone-3.35/ns-3.35/build'
Waf: Leaving directory `/home/pr/CN/ns-allinone-3.35/ns-3.35/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (0.849s)
At time +2s client sent 1024 bytes to 10.1.1.2 port 9
At time +2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time +2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
At time +2.00737s client received 1024 bytes from 10.1.1.2 port 9
pr@db:~/CN/ns-allinone-3.35/ns-3.35$
```

Conclusion: