JaganNath Gupta Institute of Engineering and Technology

Department of Computer Science and Engineering

LAB MANUAL

Programme	**:**	**Bachelor of Engineering and Technology**
Semester	**:**	**IV**
Course Code	**:**	**4CS4-23**
Subject Name	**:**	**Network Programming Lab**

INDEX

S.No.	List of Experiments	Date Of		Remark
		Experiment	Submission	
1	Study of Different Type of LAN & Network Equipment's.			
2	Study and Verification of standard Network topologies i.e. Star, Bus, Ring etc.			
3	LAN installations and Configurations.			
4	Write a program to implement various types of error correcting techniques.			
5	Write a program to Implement various types of framing methods.			
6	Study of Tool Command Language (TCL).			
7	Study and Installation of Standard Network Simulator: NS-2.			
8	Configure 802.11 WLAN.			
9	Implement & Simulate various types of routing algorithm.			
10	Study & Simulation of MAC Protocols like Aloha, CSMA, CSMA/CD and CSMA/CA.			
11	Study of Application layer protocols- DNS, HTTP, HTTPS, FTP and TelNet.			

EXPERIMENT 1

Aim: Study of Different Type of LAN & Network Equipment's.

Theory:

Types of LAN Equipment

"Local-Area Networking Introduction," discussed the different types of hardware found in a LAN environment hubs, bridges, and switches and how each piece of hardware functions specifically in an Ethernet environment.

a) Repeaters (Layer 1 Devices)

A repeater is a network device used to regenerate or replicate a signal. Repeaters are used in transmission systems to regenerate analog or digital signals distorted by transmission loss. Repeaters are used in both local-and wide-area networking environments to extend the distance a signal can reach. In the LAN environment, you would use a repeater to extend the distance a data signal can travel on a cable.

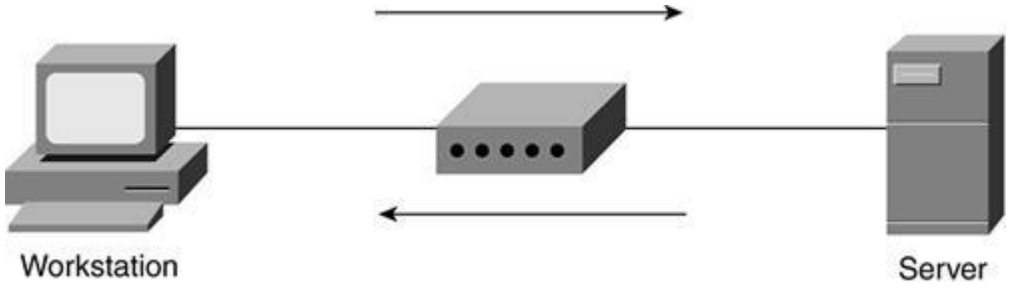


Fig. 1.1 Repeaters

b) Hubs (Layer 1 Devices)

A hub is often used to connect small LAN segments in which the number of devices is generally 24 or fewer, and hubs are multiport repeaters. Hubs provide the signal amplification required to allow a segment to be extended a greater distance.

A hub takes an incoming signal on any one port and repeats it out all ports to enable users to share the Ethernet network resources.

Ethernet hubs create star topologies in 10-Mbps or 100-Mbps half-duplex Ethernet LANs.

It is the hub that enables several point-to-point segments to be joined together into one single network. A shared Ethernet LAN means that all members of the network are contending for transmission of data onto a single network.

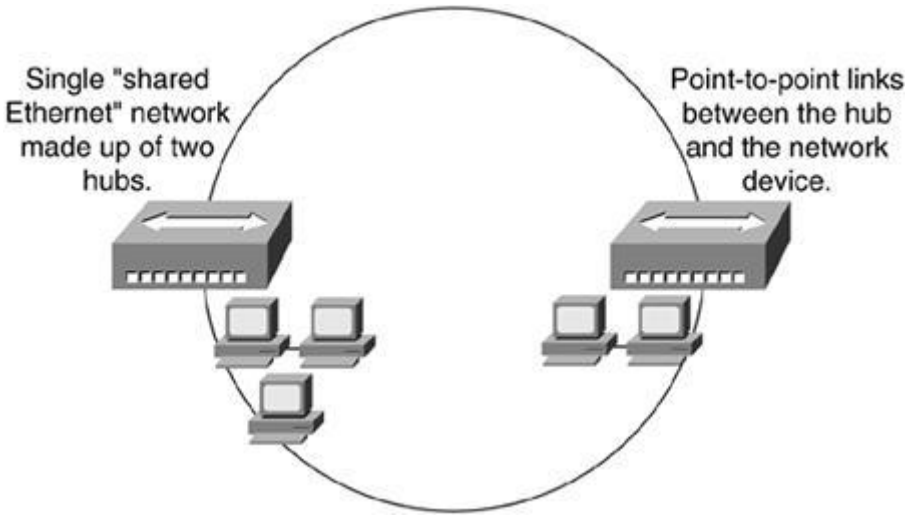


Fig. 1.2 HUB

Hub falls in two categories:

Active Hub: They are smarter than the passive hubs. They not only provide the path for the data signals in fact they regenerate, concentrate and strengthen the signals before sending them to their destinations. Active hubs are also termed as 'repeaters'.

Passive Hub: They are more like point contact for the wires to build in the physical network. They have

nothing to do with modifying the signals.

c) Bridges (Layer 2 Devices)

Repeaters and hubs have no intelligence; they just repeat whatever signal is received from one port out all ports without looking at what is being sent or received. Bridges add a level of intelligence to the network by using the MAC address to build a table of hosts, mapping these hosts to a network segment and containing traffic within these network segments. For example, Figure 5-6 illustrates a bridged network with two network segments.

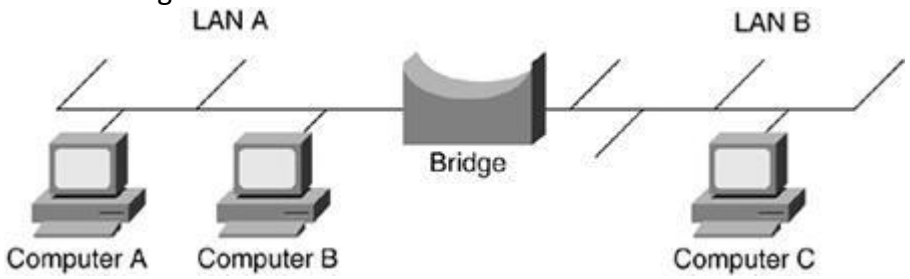


Fig. 1.3 Bridges

Types of Bridges:

There are mainly three types in which bridges can be characterized:

Transparent Bridge: It appears to be transparent for the other devices on the network. The other devices are ignorant of its existence. It only blocks or forwards the data as per the MAC **address.**

Source Route Bridge: It is mainly used in Token ring networks. The path which packet takes through the network is implanted within the packet in this type of bridges.

Translational Bridge: It converts the data format of one networking to another. For instance, Token ring to Ethernet and vice versa.

d) Switches (Layer 2 Devices)

Switches sit in the same place in the network as hubs. Unlike hubs, however, switches examine each frame and process the frame accordingly instead of just repeating the signal to all ports. Switches map the MAC addresses of the nodes residing on each network segment and then allow only the necessary traffic to pass through the switch. A switch performs the same functions as a bridge; so, when the switch receives a frame, it examines the destination and source MAC addresses and compares them to a table of network segments and addresses. If the segments are the same, the frame is dropped, or filtered; if the segments differ, the frame is forwarded to the proper segment.

The filtering of frames and regeneration of forwarded frames enables switches to split a network into separate collision domains. Frame regeneration enables greater distances and more network devices, or nodes, to be used in the total network design, and lowers the overall collision rates. In switched networks, each segment is an independent collision domain, whereas in shared networks all nodes reside in one, big, shared collision domain.

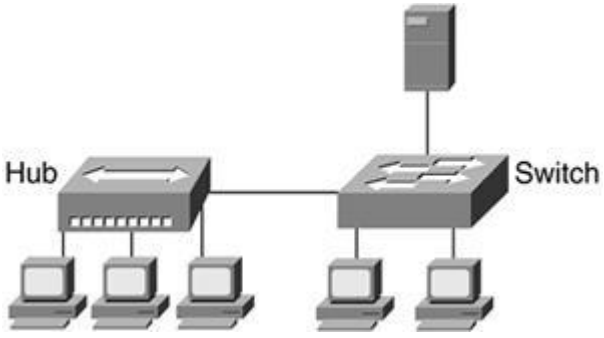


Fig.1.4 Switch

e) Routers (Layer 3 Devices)

Routers are devices that forward data packets from one LAN or WAN to another. Based on routing tables

and routing protocols, routers read the network address in the packet contained within each transmitted frame. Routers then select a sending method for the packet based on the most expedient route. This most expedient route is determined by factors such as traffic load, line quality, and available bandwidth. Routers work at Layer 3 (network) in the protocol stack, whereas bridges and switches work at Layer 2 (data link).

Routers segment LANs to balance traffic within workgroups and to filter traffic for security purposes and policy management.

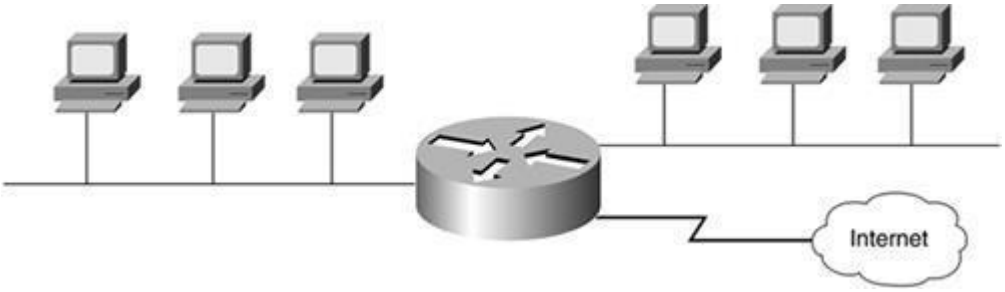


Fig. 1.5 Router

f) Gateways (Layer 7 Devices)

Gateway is a device which is used to connect multiple networks and passes packets from one packet to the other network. Acting as the 'gateway' between different networking systems or computer programs, a gateway is a device which forms a link between them. It allows the computer programs, either on the same computer or on different computers to share information across the network through protocols. A router is also a gateway, since it interprets data from one network protocol to another.

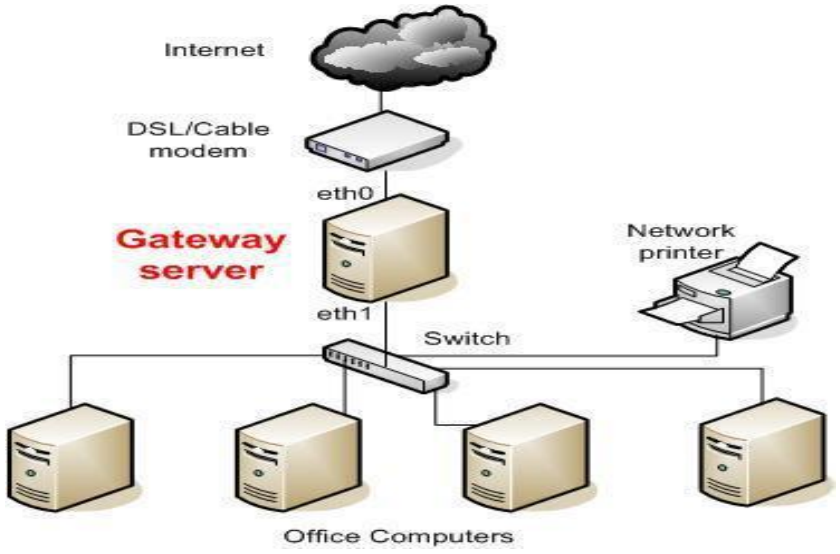


Fig 1.6 Gateways

Networking Equipment's:

a) Network card

Network cards also known as Network Interface Cards (NICs) are hardware devices that connect a computer with the network. They are installed on the mother board. They are responsible for developing a physical connection between the network and the computer. Computer data is translated into electrical signals send to the network via Network Interface Cards.

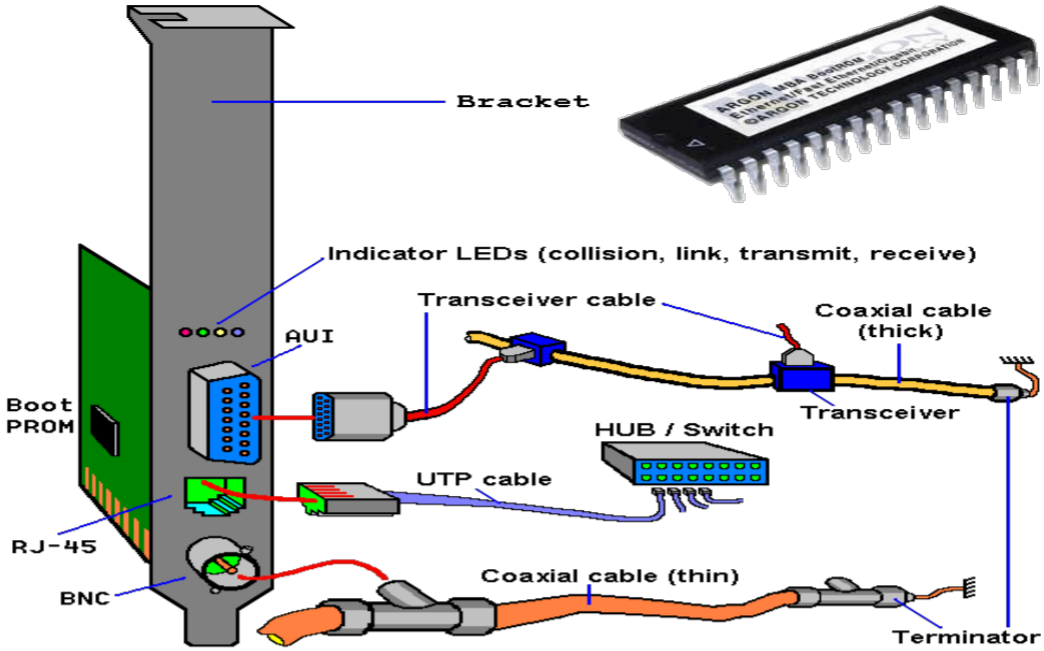


Fig 1.7 Network Interface Card

b) Modems

Modem is a device which converts the computer-generated digital signals of a computer into analog signals to enable their travelling via phone lines. The 'modulator-demodulator' or modem can be used as a dial up for LAN or to connect to an ISP. Modems can be both external, as in the device which connects to the USB or the serial port of a computer, or proprietary devices for handheld gadgets and other devices, as well as internal; in the form of add-in expansion cards for computers and PCMCIA cards for laptops.

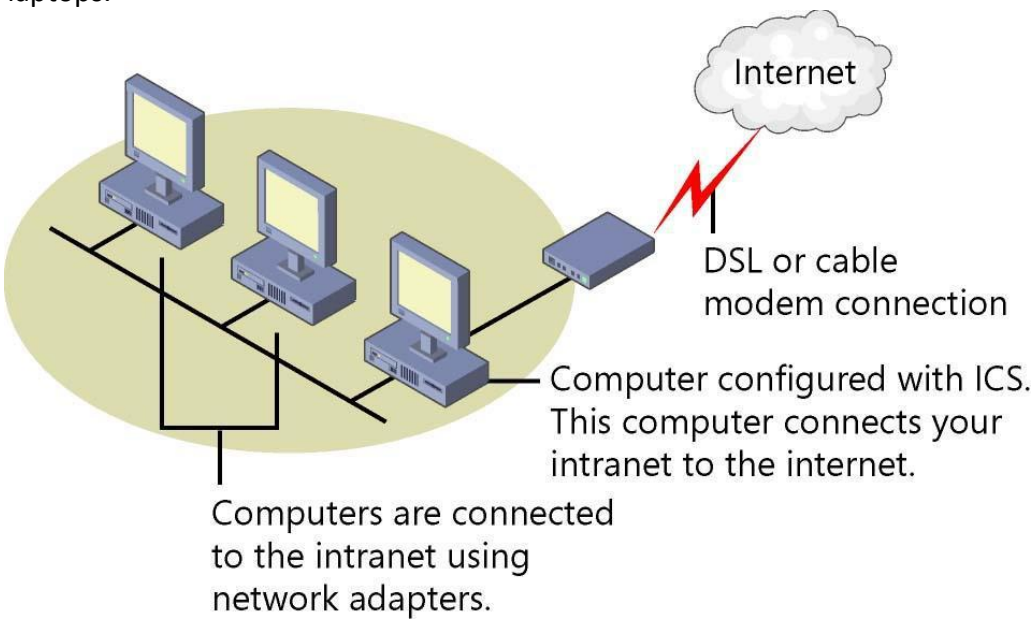


Fig 1.8 Modems

EXPERIMENT 2

Aim: Study and Verification of standard Network topologies i.e. Star, Bus, Ring etc.

Theory:

Standard Network topology

Network topology is the physical interconnections of the elements (links, nodes, etc.) of a computer network. A local area network (LAN) is one example of a network that exhibits both a physical topology and a logical topology. Any given node in the LAN has one or more links to one or more other nodes in the network and the mapping of these links and nodes in a graph results in a geometrical shape that may be used to describe the physical topology of the network. Likewise, the mapping of the data flows between the nodes in the network determines the logical topology of the network. The physical and logical topologies may or may not be identical in any particular.

The various types of network topologies are as follows:

1. Hierarchical topology
2. Bus topology
3. Star topology
4. Ring topology
5. Mesh topology
6. Hybrid topology

1) Hierarchical Topology

The hierarchical topology is also known as tree topology, which is divided into different levels connected with the help of twisted pair, coaxial cable or fiber optics. This type of topology is arranged in the form of a tree structure in which top level contains parent node (root node), which is connected with the child nodes in the second level of hierarchy with point-to-point link. The second level nodes are connected to the third level nodes, which in turn are connected to the fourth level nodes and so on. Except the top-level nodes, each level node has a parent node.

The number of point-to-point links in the hierarchical type of topology is generally one less than the total number of nodes in the structure. The hierarchical topology is symmetrical, having a fixed branching factor, f, associated with each node. The branching factor is the number of point-to-point links between the levels of hierarchy.

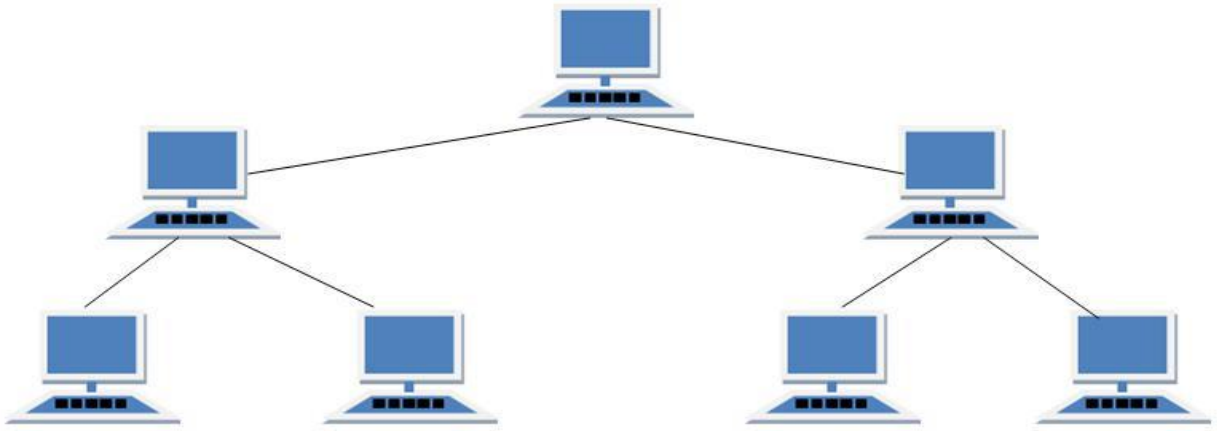


Figure 2.1 Hierarchical topology.

Advantages of hierarchical topology are:

- The hierarchical topology is generally supported by most hardware and software.
- In the hierarchical topology, data is received by all the nodes efficiently because of point-to- point link.

- In the hierarchical topology, when the root node fails, the whole network crashes.
- The hierarchical topology is difficult to configure.

2) Linear Bus Topology

In the linear bus topology, all the nodes are connected to the single backbone or bus with some medium such as twisted pair, coaxial cable etc. When a node wants to communicate with the other nodes in the network, it simply sends a message to the common bus. All the nodes in the network then receive the message but the node for which it was actually sent only processes it. The other nodes discard the message.

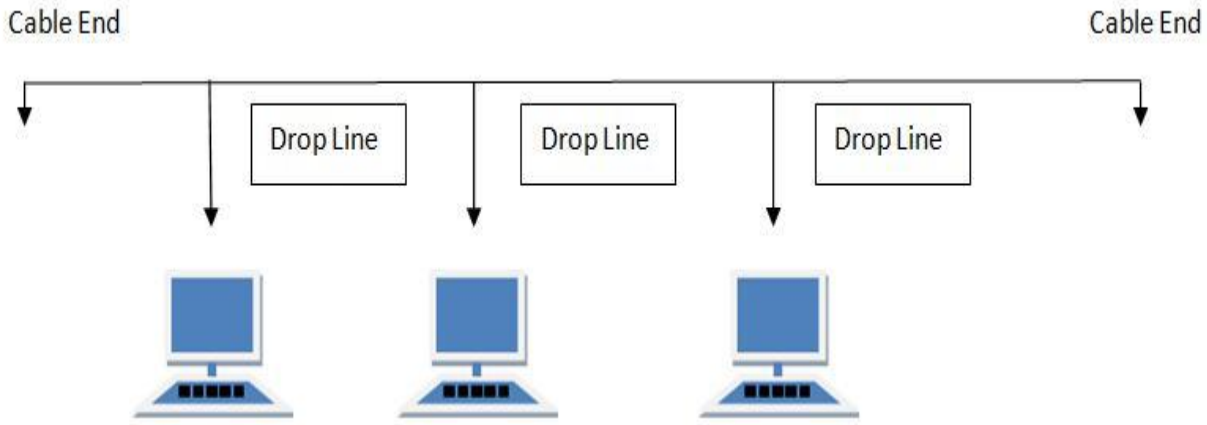


Figure 2.2 Linear bus topology.

Advantages of linear bus topology are:

- The linear bus topology usually requires less cabling.
- The linear bus topology is relatively simple to configure and install.
- In the linear bus topology, the failure of one computer does not affect the other computers in the network.
- In the linear bus topology, the failure of the backbone cable results in the breakdown of entire network.
- Addition of computers in the linear bus topology results in the performance degradation of the network.
- The bus topology is difficult to reconstruct in case of faults.

3) Star Topology

In the star topology, all the nodes are connected to a common device known as hub. Nodes are connected with the help of twisted pair, coaxial cable or optical fiber. When a node wants to send a message to the other nodes, it first sends the message to the hub, which in turn forwards the message to the intended node. Each node in the network is connected with a point-to-point link to the centralized hub. The task of hub is to detect the faulty node present in the network. On the other hand, it also manages the overall data transmission in the network.

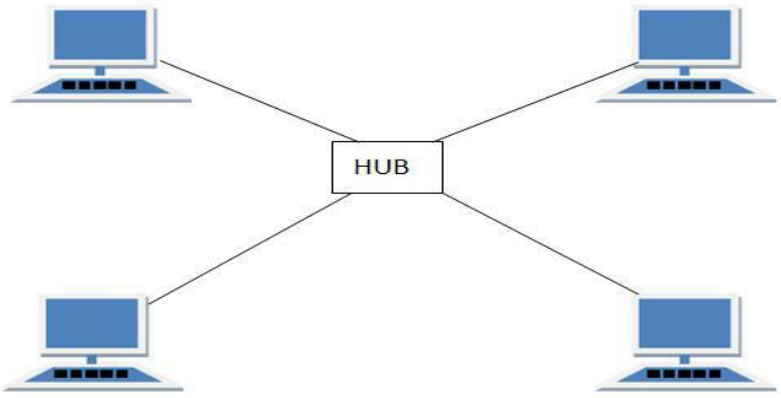


Figure 2.3 Star topology.

Advantages of star topology are:

- This topology allows easy error detection and correction.
- In the star topology, the failure of one computer does not affect the other computers in the network.
- Star topology is easy to install.

Disadvantages of star topology:

- In the star topology, the hub failure leads to the overall network crash.
- The star topology requires more amount of cable for connecting the nodes.
- It is expensive due to the cost of the hub.

4) Ring Topology

In the ring topology, the nodes are connected in the form of a ring with the help of twisted-pair cable. Each node is connected directly to the other two nodes in the network. The node, which wants to send a message, first passes the message to its consecutive node in the network. Data is transmitted in the clockwise direction from one node to another. Each node incorporates a repeater, which passes the message to next node when the message is intended for another node.

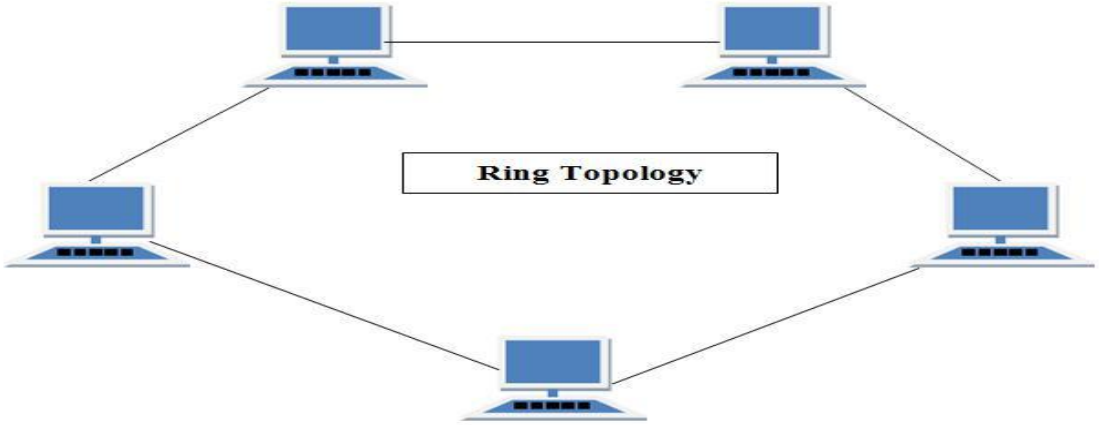


Figure 2.4 Ring topology

Advantages of ring topology are:

- Each node has an equal access to other nodes in the network.
- Addition of new nodes does not degrade the performance of the network.
- Ring topology is easy to configure and install.

Disadvantages of ring topology:

- It is relatively expensive to construct the ring topology.
- The failure of one node in the ring topology affects the other nodes in the ring.

5) Mesh Topology

In mesh topology, each computer is connected to every other computer in point-to-point mode as shown in figure 5. For example, if we have four computers, we must have six links. If we have n computers, we must have n(n-1)/2 links. A message can take several possible paths to reach a destination.

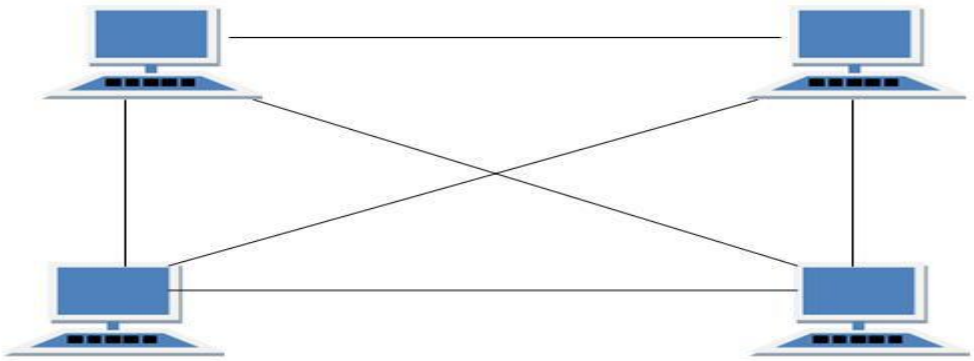


Figure 2.5 Mesh topology.

Advantages of mesh topology are:

- Message delivery is more reliable.
- Network congestion is minimal due to large number of links.

Disadvantages of mesh topology:

- It is very expensive to implement.
- It is very difficult to configure and install.

6) Hybrid Topology

The hybrid topology is the combination of multiple topologies, used for constructing a single large topology. The hybrid topology is created when two different network topologies are interconnected.

If two ring topologies are connected then the resultant topology is not the hybrid topology. On the other hand, if the ring topology is connected to the bus topology then the resulting topology is called the hybrid topology. This topology generally combines the features of the two topologies and is therefore more effective and efficient than the individual topologies.

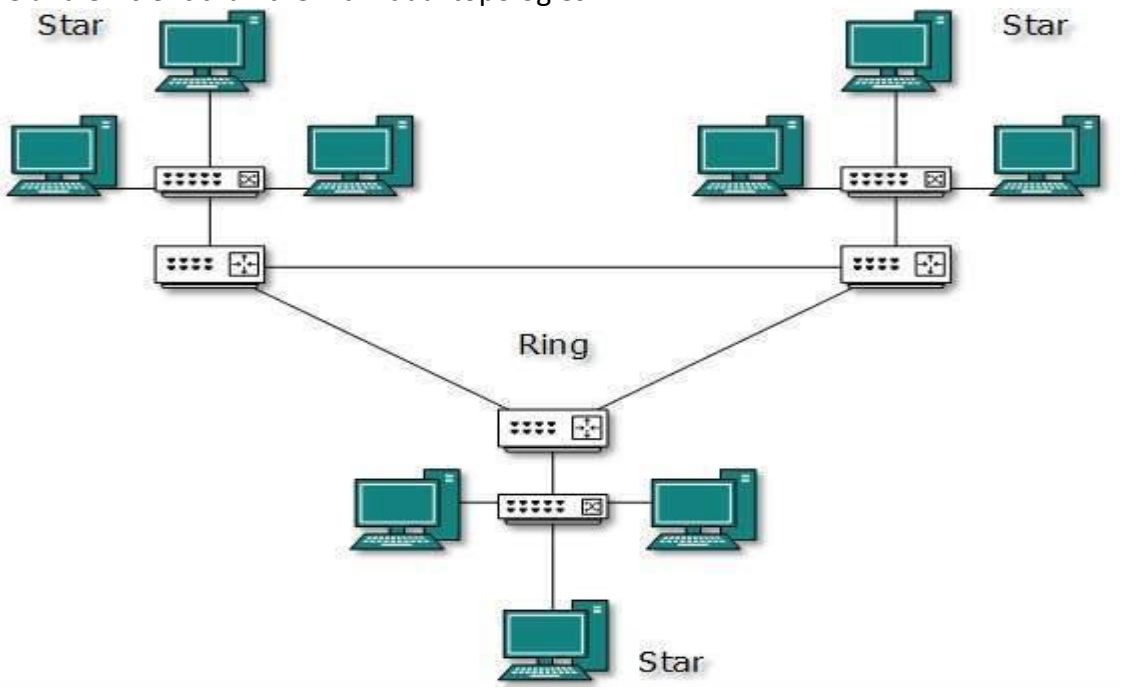


Figure 2.6 Hybrid topology.

Advantages of hybrid topology are:

- The hybrid topology is more effective as it uses multiple topologies.
- The hybrid topology contains the best and efficient features of the combined topologies from which it is constructed.

- The hybrid topology is relatively more complex than the other topologies.
- The hybrid topology is difficult to install and configure.

EXPERIMENT 3

Aim: LAN installations and Configurations

Theory:

Simply put, a LAN is a computer network that connects a relatively small area (a single building or group of buildings). Most LANs connect workstations and computers to each other. Each computer (also known as a "node"), has its own processing unit and executes its own programs; however, it can also access data and devices anywhere on the LAN. This means that many users can access and share the same information and devices. A good example of a LAN device is a network printer. Most companies cannot afford the budgetary or hardware expense of providing printers for each of its users. Therefore, one printer (i.e., device) is placed on the LAN where every user can access the same printer.

The LAN uses IP addresses to route data to different destinations on the network. An IP Address is a 32-bit numeric address written as four numbers separated by periods (For example, 1.160.10.240).

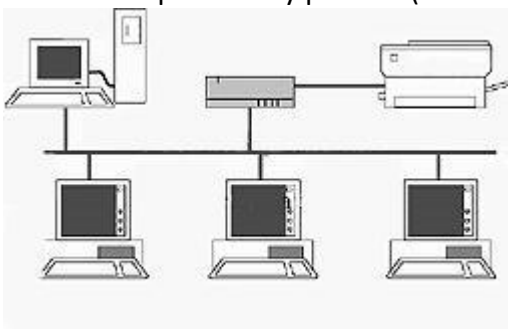


Fig. 3.1 Local Area Network Diagram

STEPS: Proceed with following steps to configure your LAN.

1. From your Windows Desktop, click **"Start"**
2. Click on **"Control Panel"**
3. Click on "**Network and Internet Connections**"
4. Click on "**Local Area Connection**"
5. Click "**Properties**" on the "Local Area Connection Status" panel.
6. You should find a check mark already in the box next to "Internet Protocol (TCP/IP)". Double Click "Internet Protocol (TCP/IP)".
7. Select **"Obtain an IP Address automatically"**
8. Select "**Obtain DNS server addresses automatically**"
9. Click the "**OK**" button
10. Click the next "**OK**" button
11. Close the "**Network Connections**" panel

EXPERIMENT 4

Aim: Write a program to implement various types of error correcting techniques.

Theory:

Error detection is the detection of errors caused by noise or other impairments during transmission from the transmitter to the receiver.

Error correction is the detection of errors and reconstruction of the original, error-free data. Introduction

The general idea for achieving error detection and correction is to add some redundancy (i.e., some extra data) to a message, which receivers can use to check consistency of the delivered message, and to recover data determined to be erroneous. Error-detection and correction schemes can be either systematic or non-systematic. Good error control performance requires the scheme to be selected based on the characteristics of the communication channel. Common channel models include memory-less models where errors occur randomly and with a certain probability, and dynamic models where errors occur primarily in bursts.

Implementation

Error correction may generally be realized in two different ways:

- Automatic repeat request (ARQ) (sometimes also referred to as backward error correction): This is an error control technique whereby an error detection scheme is combined with requests for retransmission of erroneous data.
- Forward error correction (FEC): The sender encodes the data using an error-correcting code (ECC) prior to transmission. The additional information (redundancy) added by the code is used by the receiver to recover the original data.

Error detection schemes

Error detection is most commonly realized using a suitable hash function (or checksum algorithm). A hash function adds a fixed-length tag to a message, which enables receivers to verify the delivered message by re-computing the tag and comparing it with the one provided.

There exists a vast variety of different hash function designs. However, some are of particularly widespread use because of either their simplicity or their suitability for detecting certain kinds of errors (e.g., the cyclic redundancy check's performance in detecting burst errors).

Random-error-correcting codes based on minimum distance coding can provide a suitable alternative to hash functions when a strict guarantee on the minimum number of errors to be detected is desired

- **Parity bits**

  A parity bit is a bit that is added to a group of source bits to ensure that the number of set bits (i.e., bits with value 1) in the outcome is even or odd. It is a very simple scheme that can be used to detect single or any other odd number (i.e., three, five, etc.) of errors in the output. An even number of flipped bits will make the parity bit appear correct even though the data is erroneous.

- **Checksums**

  A checksum of a message is a modular arithmetic sum of message code words of a fixed word length (e.g., byte values). The sum may be negated by means of aones'-complement operation prior to transmission to detect errors resulting in all-zero messages.

  Checksum schemes include parity bits, check digits, and longitudinal redundancy checks.

- **Cyclic redundancy checks (CRCs)**

  A cyclic redundancy check (CRC) is a single-burst-error-detecting cyclic code and non-secure hash function designed to detect accidental changes to digital data in computer networks. It is characterized by specification of a so-called generator polynomial, which is used as the divisor in a polynomial long division over a finite field, taking the input data as the dividend, and where the remainder becomes the result.

  Even parity is a special case of a cyclic redundancy check, where the single-bit CRC is generated by the

divisor x + 1.

- **Cryptographic hash functions**

  A cryptographic hash function can provide strong assurances about data integrity, provided that changes of the data are only accidental (i.e., due to transmission errors). Any modification to the data will likely be detected through a mismatching hash value. Furthermore, given some hash value, it is infeasible to find some input data (other than the one given) that will yield the same hash value. Message authentication codes.

Error-correcting codes

Any error-correcting code can be used for error detection. A code with minimum Hamming distance, d, can detect up to d − 1 errors in a code word. Using minimum-distance-based error-correcting codes for error detection can be suitable if a strict limit on the minimum number of errors to be detected is desired.

Codes with minimum Hamming distance d = 2 are degenerate cases of error-correcting codes, and can be used to detect single errors. The parity bit is an example of a single-error-detecting code.

Error correction

- Automatic repeat request

  Automatic Repeat request (ARQ) is an error control method for data transmission that makes use of error-detection codes, acknowledgment and/or negative acknowledgment messages, and timeouts to achieve reliable data transmission. An acknowledgment is a message sent by the receiver to indicate that it has correctly received a data frame.

  Three types of ARQ protocols are Stop-and-wait ARQ, Go-Back-N ARQ, and Selective Repeat ARQ.

Applications

- Applications that require low latency (such as telephone conversations) cannot use Automatic Repeat request (ARQ); they must use Forward Error Correction (FEC). By the time an ARQ system discovers an error and re-transmits it, the re-sent data will arrive too late to be any good.
- Applications where the transmitter immediately forgets the information as soon as it is sent (such as most television cameras) cannot use ARQ; they must use FEC because when an error occurs, the original data is no longer available. (This is also why FEC is used in data storage systems such as RAID and distributed data store).
- Applications that use ARQ must have a return channel. Applications that have no return channel cannot use ARQ.
- Applications that require extremely low error rates (such as digital money transfers) must use ARQ.

Internet

In a typical TCP/IP stack, error control is performed at multiple levels:

- Each Ethernet frame carries a CRC-32checksum. Frames received with incorrect checksums are discarded by the receiver hardware.
- The IPv4 header contains a checksum protecting the contents of the header. Packets with mismatching checksums are dropped within the network or at the receiver.
- The checksum was omitted from the IPv6 header in order to minimize processing costs in network routing and because current link layer technology is assumed to provide sufficient error detection.
- UDP has an optional checksum covering the payload and addressing information from the UDP and IP headers. Packets with incorrect checksums are discarded by the system network. The checksum is optional under IPv4, only, because the Data-Link layer checksum may already provide the desired level of error protection.
- TCP provides a checksum for protecting the payload and addressing information from the TCP and IP headers. Packets with incorrect checksums are discarded within the network stack, and eventually get

retransmitted using ARQ, either explicitly (such as through triple-ack) or implicitly due to a timeout.

Program: 1. Using Hamming Code

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<math.h>
void main()
{
	char sender[20],buffer[50],reciver[20];
	int d,p=0,i=0,j,k,length,power,count;
	clrscr();
	printf("ENTER THE MESSAGE IN BIT FORM:- ");
	gets(sender);
	d=strlen(sender); //power
	for(p=0; ;p++)
	{
		if(pow(2,p)>=d+p)
		{
			break;
		}
	}
	p--;
	 /* COPY THE BITS INTO BUFFER */
	 i=1; power=0; j=0;
	do {
		if(i==pow(2,power))
		{
			buffer[i]='0';
			power++;
			i++;
		}
		else
		{
			buffer[i]=sender[j];
			j++;
			i++; }
		}
		while(j<=d);
		buffer[i]='\0'; //APPLY BIT IN CODE
		count=0;
		length=d+p;
		j=0;
		for(j=0;j<=p;j++)
		{
			k=pow(2,j);
			for(i=k+1;i<=length; )
			{
				if(k==1)
				{
					if(buffer[i]=='1')
```

```
                                        { count++;
                                        } i=i+2;
                                }
                                else {
                                        if(buffer[i]=='1')
                                        { count++;
                                        } i++;
                                        if(i%k==0) {
                                        i=i+k; }
                                        }
                        }
                        if(count%2==1)
                        {
                                buffer[k]='1';
                        }
                        count=0;
                }
                printf("ENCODED MESSAGE IS \n ");
                i=1;
                do { printf("%c",buffer[i]);
                i++;
                }while(buffer[i]!='\0');
                 getch();
 }
```

Program 2. ARQ for Error Correcting

```
#include<stdio.h>
int main()
{
        int windowsize,sent=0,ack,i;
        printf("enter window size\n");
        scanf("%d",&windowsize);
        while(1)
        {
                for( i = 0; i < windowsize; i++)
                        {
                                printf("Frame %d has been transmitted.\n",sent);
                                sent++;
                                if(sent == windowsize)
                                        break;
                        }
                        printf("\n Please enter the last Acknowledgement received.\n");
                        scanf("%d",&ack);

                        if(ack == windowsize)
                                break;
                        else
                                sent = ack;
        }
return 0;
}
```

EXPERIMENT 5

Aim: Write a program to implement various types of framing methods.

Theory:

Framing

Since the physical layer merely accepts and transmits a stream of bits without any regard to meaning or structure, it is upto the data link layer to create and recognize frame boundaries. This can be accomplished by attaching special bit patterns to the beginning and end of the frame. If these bit patterns can accidentally occur in data, special care must be taken to make sure these patterns are not incorrectly interpreted as frame delimiters.

The three framing methods that are widely used are

- Character count
- Starting and ending characters, with character stuffing
- Starting and ending flags, with bit stuffing

Character Count

This method uses a field in the header to specify the number of characters in the frame. When the data link layer at the destination sees the character count, it knows how many characters follow, and hence where the end of the frame is. The disadvantage is that if the count is garbled by a transmission error, the destination will lose synchronization and will be unable to locate the start of the next frame. So, this method is rarely used.

Character stuffing

In the second method, each frame starts with the ASCII character sequence DLE STX and ends with the sequence DLE ETX.(where DLE is Data Link Escape, STX is Start of TeXt and ETX is End of TeXt.) This method overcomes the drawbacks of the character count method. If the destination ever loses synchronization, it only has to look for DLE STX and DLE ETX characters. If however, binary data is being transmitted then there exists a possibility of the characters DLE STX and DLE ETX occurring in the data. Since this can interfere with the framing, a technique called character stuffing is used. The sender's data link layer inserts an ASCII DLE character just before the DLE character in the data. The receiver's data link layer removes this DLE before this data is given to the network layer. However character stuffing is closely associated with 8-bit characters and this is a major hurdle in transmitting arbitrary sized characters.

Bit stuffing

The third method allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary number of bits per character. At the start and end of each frame is a flag byte consisting of the special bit pattern 01111110. Whenever the sender's data link layer encounters five consecutive 1s in the data, it automatically stuffs a zero bit into the outgoing bit stream. This technique is called bit stuffing. When the receiver sees five consecutive 1s in the incoming data stream, followed by a zero bit, it automatically destuffs the 0 bit. The boundary between two frames can be determined by locating the flag pattern.

Program: 1. Character Count

1. **SENDER SIDE CODE**

```
#include<stdio.h>
#include<fcntl.h>
#include<string.h>
void main()
{
 int no,j=0,len,i,k,pid; char
 data[50],frame[100];
 system("clear");

 system(">pipe");
 pid=open("pipe",O_WRONLY);
```

```
 printf("Enter how many data want to enter: ");
 scanf("%d",&no);
 for(i=0;i<no;i++)
 {
  printf("\nEnter data : ");
  scanf("%s",data);
  len=strlen(data);
  frame[j]=len+1+48;
  for(k=0;k<len;k++)
  {
   j++;
   frame[j]=data[k];
  }
  j++;
 }
 frame[j]='\0';
 printf("\nFRAME TO SEND : %s",frame);

 write(pid,&frame,sizeof(frame));
}
```

2. RECEIVER SIDE CODE

```
#include<stdio.h>
#include<fcntl.h>
#include<string.h>
void main()
{
 int no,j=0,len,i=0,k,pid;
 char data[50],frame[100];
 system("clear");

 pid=open("pipe",O_RDONLY);
 read(pid,&frame,sizeof(frame));

 while(frame[i]!='\0')
 {
  len=frame[i]-48;
  i++;
  for(k=0;k<len-1;k++)
  {
   data[k]=frame[i];
   i++;
  }
  data[k]='\0';
  printf("\ndata %s",data);
 }
}
```

Program: 2.Character Stuffing

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<process.h>
void main()
{
int i=0,j=0,n,pos;
char a[20],b[50],ch;
clrscr();
printf("enter string\n");
scanf("%s",&a);
n=strlen(a);
printf("enter position\n");
scanf("%d",&pos);
if(pos>n)
{
printf("invalid position, Enter again :");
scanf("%d",&pos);
}
printf("enter the character\n");
ch=getche();

b[0]='d';
b[1]='l';
b[2]='e';
b[3]='s';
b[4]='t';
b[5]='x';
j=6;
while(i<n)
{
if(i==pos-1)
{
b[j]='d';
b[j+1]='l';
b[j+2]='e';
b[j+3]=ch;
b[j+4]='d';
b[j+5]='l';
b[j+6]='e';
j=j+7;
}
if(a[i]=='d' && a[i+1]=='l' && a[i+2]=='e')
{
b[j]='d';
b[j+1]='l';
b[j+2]='e';
j=j+3;
}
```

```
b[j]=a[i];
i++;
j++;
}
b[j]='d';
b[j+1]='l';
b[j+2]='e';
b[j+3]='e';
b[j+4]='t';
b[j+5]='x';
b[j+6]='\0';
printf("\nframe after stuffing:\n");
printf("%s",b);
getch();
}
```

EXPERIMENT 6

Aim: Study of Tool Command Language (TCL).

Theory:

Tcl (originally from "Tool Command Language", but conventionally spelled "Tcl" rather than "TCL"; pronounced as "tickle" or "tee-see-ell") is a scripting language created by John Ouster out. Originally "born out of frustration" , according to the author, with programmers devising their own languages intended to be embedded into applications, Tcl gained acceptance on its own. It is commonly used for rapid prototyping, scripted applications, GUIs and testing. Tcl is used on embedded systems platforms, both in its full form and in several other small-footprint versions.

The combination of Tcl and the TkGUI toolkit is referred to as **Tcl/Tk.**

Tcl's features include

- All operations are commands, including language structures. They are written in prefix notation.
- Commands are commonly variadic.
- Everything can be dynamically redefined and overridden.
- All data types can be manipulated as strings, including source code.
- Event-driven interface to sockets and files. Time-based and user-defined events are also possible.
- Variable visibility restricted to lexical (static) scope by default, but uplevel and upvar allowing process to interact with the enclosing functions' scopes.
- All commands defined by Tcl itself generate error messages on incorrect usage.
- Extensibility, via C, C++, Java, and Tcl.
- Interpreted language using byte code
- Full Unicode (3.1) support, first released 1999.
- Cross-platform: Windows API; UNIX, Linux, Macintosh, etc.
- Close integration with windowing (GUI) interface Tk.
- Multiple distribution mechanisms exist:
    - Full development version (e.g., Active State Tcl).
    - tclkit (kind of single-file runtime, only about 1 megabyte in size)
    - starpack (single-file executable of a script/program, derived from the tclkit technology). ○ FreewrapTCLSH turns TCL scripts into single-file binary executable programs.
    - BSD licenses, freely distributable source.

Installation:

Step 1: The Website

Open website: http://www.activestate.com/activetcl

Step 2: Downloading Active Tcl

Click on the "Download Now" button then download either the x86 or x64 version of Tcl (Whichever works best for you). Your download should start automatically, if not then just start the download manually.

Step 3: Installing Tcl

- First find where you downloaded the ActiveTcl installer and open it up
- Read (if you want) and then click next and accept the License Agreement
- Now choose where you want to install everything
- Last but not least, click next to begin installing

Now you have successfully installed Tcl and can now start programming.

Syntax and fundamental semantics

A Tcl script consists of several command invocations. A command invocation is a list of words separated by whitespace and terminated by a newline or semicolon.

word0 word1 word2 ... wordN

The first word is the name of a command, which is not built into the language, but which is in the library. The following words are arguments. So, we have:

```
commandName argument1 argument2 ... argumentN
```

Practical example, using the puts command which outputs a string, adding a trailing newline, by default to the stdout channel:

```
puts"Hello, world!"
```

Variables and the results of other commands can be substituted inside strings too, such as in this example where we use set and expr to store a calculation result in a variable, and puts to print the result together with some explanatory text:

```
# Good style would put the expression (1+2+3+4+5, in this case) inside {curly
braces} set sum [expr1+2+3+4+5]
puts"The sum of the numbers 1..5 is $sum."

#expr function will be evaluated faster if curly braces are added on the equation.

set sum [expr{1+2+3+4+5}]
puts"The sum of the numbers 1..5 is $sum."
```

There is one basic construct (the command) and a set of simple substitution rules.

Formally, words are either written as-is, with double-quotes around them (allowing whitespace characters to be embedded), or with curly-brace characters around them, which suppresses all substitutions inside (except for backslash-newline elimination). In bare and double-quoted words, three types of substitution occur (once, in a single left-to-right scan through the word):

- Command substitution replaces the contents of balanced square brackets with the result of evaluating the script contained inside. For example, "[expr 1+2+3]" is replaced with the result of evaluating the contained expression (i.e. 6) since that's what the expr command does.
- Variable substitution replaces a dollar-sign followed by the name of a variable with the contents of the variable. For example, "$foo" is replaced with the contents of the variable called "foo". The variable name may be surrounded in curly braces so as to delimit what is and isn't the variable name in otherwise ambiguous cases.
- Backslash substitution replaces a backslash followed by a letter with another character. For example, "\n" is replaced with a newline.

From Tcl 8.5 onwards, any word may be prefixed by "{*}" to cause that word to be split apart into its constituent sub-words for the purposes of building the command invocation (similar to the ",@" sequence of Lisp's quasiquote feature).

As a consequence of these rules, the result of any command may be used as an argument to any other command. Also, there is no operator or command for string concatenation, as the language concatenates directly. Note that, unlike in Unix command shells, Tcl does not reparse any string unless explicitly directed to do so, which makes interactive use more cumbersome but scripted use more predictable (e.g. the presence of spaces in filenames does not cause difficulties).

The single equality sign (=) for example is not used at all, and the double equality sign (==) is the test for equality, and even then, only in expression contexts such as the expr command or the first argument to if. (Both of those commands are just part of the standard library; they have no particularly special place in the library and can be replaced if so desired.)

The majority of Tcl commands, especially in the standard library, are variadic, and the proc (the constructor for scripted command procedures) allows one to define default values for unspecified arguments and a catch-all argument to allow the code to process arbitrary numbers of arguments.

Tcl is not statically typed: each variable may contain integers, floats, strings, lists, command names, dictionaries, or any other value; values are reinterpreted (subject to syntactic constraints) as other types on demand. However, values are immutable and operations that appear to change them actually just return a new value instead.

Basic Principles of Tcl:

- Commands have a sequence of arguments, separated by white space
- Commands can be separated by line ends or semicolons
- Variables:
  set a hello
  set b $a
- Substitution of results of Tcl
  commands: set b 42
  set a [expr $b+2]
- Quoting (almost) like in a shell: ", { }, \
- Simple substitution rules (but sometimes confusing)

Data Types and
Commands: Tcl data types:

- Everything can be represented as a character string
- Character strings can be interpreted as lists
- Associative arrays support fast keyed access to data
- Program text and data is interchangeable

Tcl commands:

- All language constructs are commands (no keywords)
- Some command arguments can be arithmetic expressions
- Conditional commands and loop commands
- Procedures are sequences of commands that can be called
- Built-in commands to access file systems and basic operating system services

Tcl Syntax Rules:

- A Tcl script is a sequence of Tcl commands
- Commands are separated by line ends or semicolons
- The words of a command are separated by white space
- The first word of a command is the command name
- The remaining words are the arguments passed to the command

Examples

set a 23; set b
33 set t Hello
string match *ll*
Hello expr 23 * 2 – 4

EXPERIMENT 7

Aim: Study and Installation of Standard Network Simulator: N.S-2.

Theory:

Study and Installation of Standard Network Simulator

In communication and computer network research, network simulation is a technique where a program models the behaviour of a network either by calculating the interaction between the different network entities (hosts/routers, data links, packets, etc.) using mathematical formulas, or actually capturing and playing back observations from a production network. The behaviour of the network and the various applications and services it supports can then be observed in a test lab; various attributes of the environment can also be modified in a controlled manner to assess how the network would behave under different conditions. When a simulation program is used in conjunction with live applications and services in order to observe end-to-end performance to the user desktop, this technique is also referred to as network emulation.

Network simulator

A network simulator is a piece of software or hardware that predicts the behaviour of a network, without an actual network being present. A network simulator is a software program that imitates the working of a computer network. In simulators, the computer network is typically modeled with devices, traffic etc. and the performance is analysed. Typically, users can then customize the simulator to fulfill their specific analysis needs. Simulators typically come with support for the most popular protocols in use today, such as WLAN, Wi-Max, UDP, and TCP.

Simulations

- Most of the commercial simulators are GUI driven, while some network simulators require input scripts or commands (network parameters). The network parameters describe the state of the network (node placement, existing links) and the events (data transmissions, link failures, etc.). An important output of simulations is the trace files. Trace files can document every event that occurred in the simulation and are used for analysis. Certain simulators have added functionality of capturing this type of data directly from a functioning production environment, at various times of the day, week, or month, in order to reflect average, worst-case, and best-case conditions. Network simulators can also provide other tools to facilitate visual analysis of trends and potential trouble spots.
- Most network simulators use discrete event simulation, in which a list of pending "events" is stored, and those events are processed in order, with some events triggering future events–such as the event of the arrival of a packet at one node triggering the event of the arrival of that packet at a downstream node.
- Some network simulation problems, notably those relying on queueing theory, are well suited to Markov chain simulations, in which no list of future events is maintained and the simulation consists of transiting between different system "states" in a memory less fashion. Markov chain simulation is typically faster but less accurate and flexible than detailed discrete event simulation. Some simulations are cyclic based simulations and these are faster as compared to event-based simulations.
- Simulation of networks can be a difficult task. For example, if congestion is high, then estimation of the average occupancy is challenging because of high variance. To estimate the likelihood of a buffer overflow in a network, the time required for an accurate answer can be extremely large. Specialized techniques such as "control variants" and "importance sampling" have been developed to speed simulation.

Examples of network simulators

Examples of notable network simulation software are, ordered after how often they are mentioned in research papers:

1. ns2/ns3
2. OPNET
3. NetSim

1. NS (simulator)

Ns (from network simulator) is a name for series of discrete event network simulators, specifically ns-1, ns-2 and ns-3. These simulators are used in the simulation of routing protocols, among others, and are heavily used in ad-hoc networking research, and support popular network protocols, offering simulation results for wired and wireless networks alike.

Design

Ns-3 is built using C++ and Python and scripting is available with either language. Split over 30 modules, features of ns-3 include:

- Callback-driven events
- Attribute system that manages default and per-object simulation values
- Helpers that allow using simpler API when configuring simulations

Workflow for ns

It includes four steps:

- Implement protocol models
- Setup simulation scenario, i.e. create Tcl file describing type of scenario, e.g. number of nodes, kind of agent working on nodes etc.
- Run simulation, i.e. Run the Tcl file
- Analyse simulation results, i.e. by GNU Awk and gnuplot

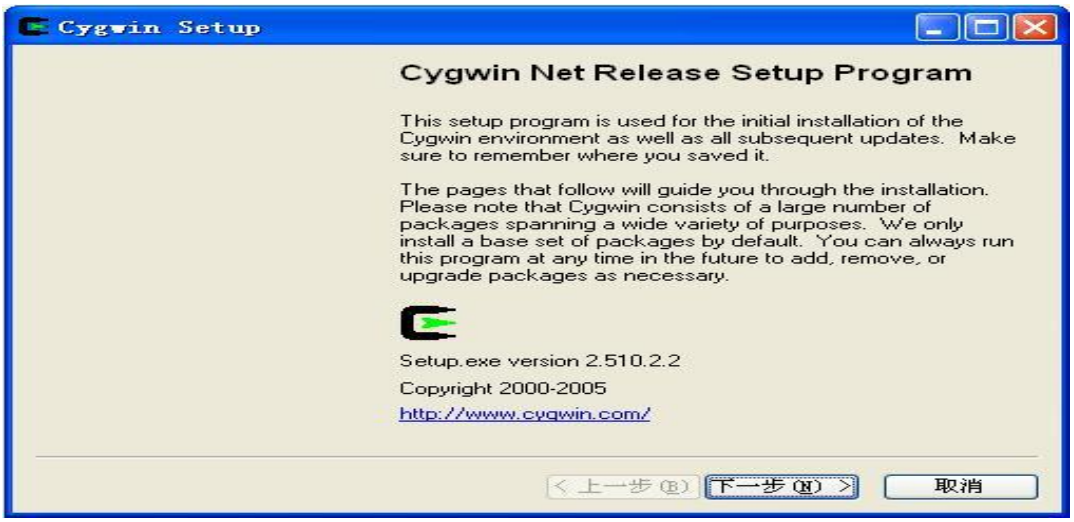
Components

- Ns, the simulator itself has Nam, the network animator to visualize ns (or other) output.
- Pre-processing component for Traffic and topology generators.
- Post-processing for Simple trace analysis, often in Awk, Perl, or Tcl.

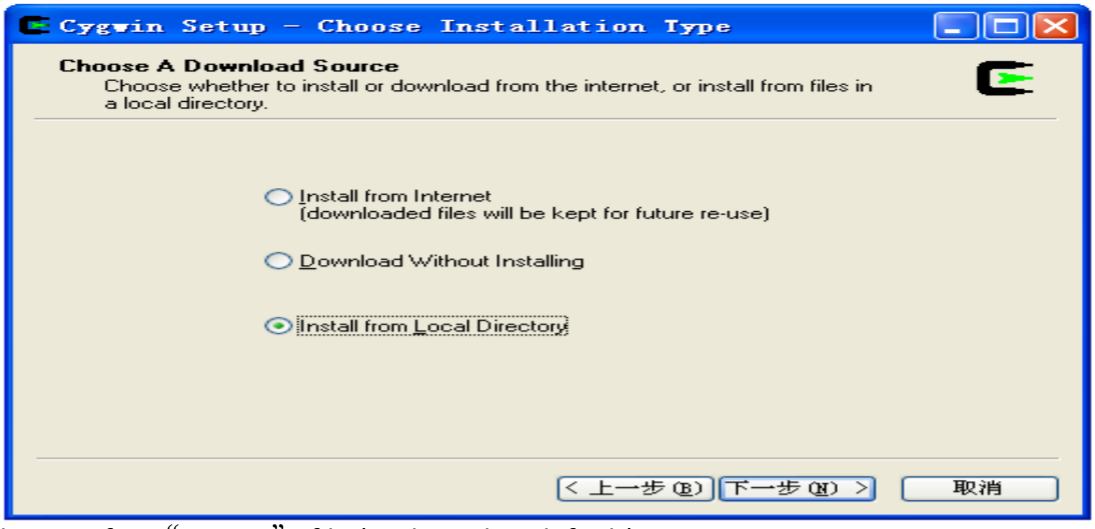
Installation of NS 2

1. Install CYGWIN

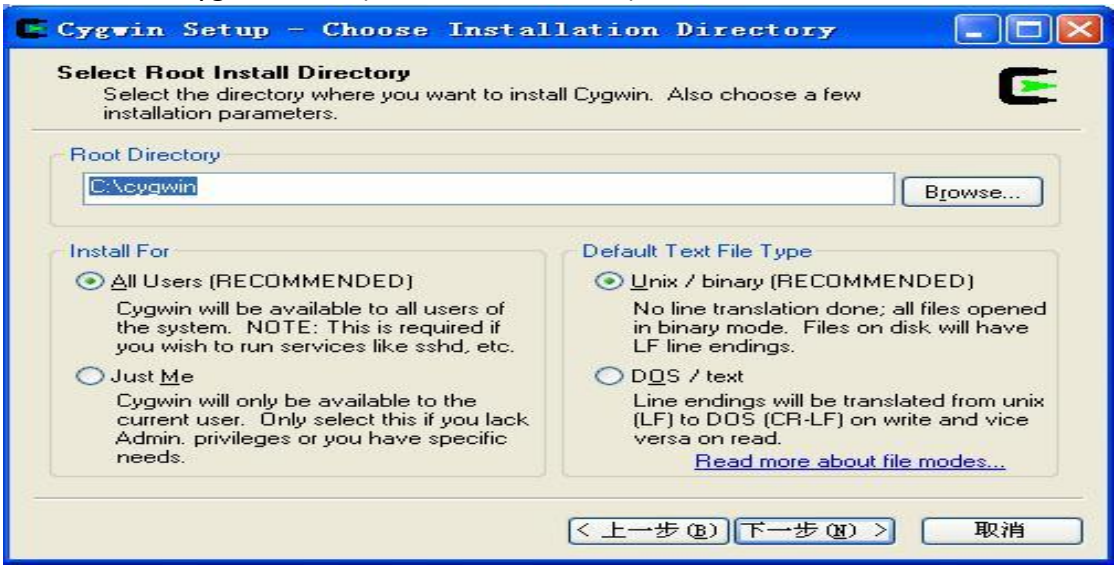
- Download zip file of ns2.29 (allinone) from: http://www.isi.edu/nsnam/dist/ns-allinone-2.29.2.tar.gz
- download cygwin setup.exe from www.cygwin.com
- Click on "cygwin.exe".



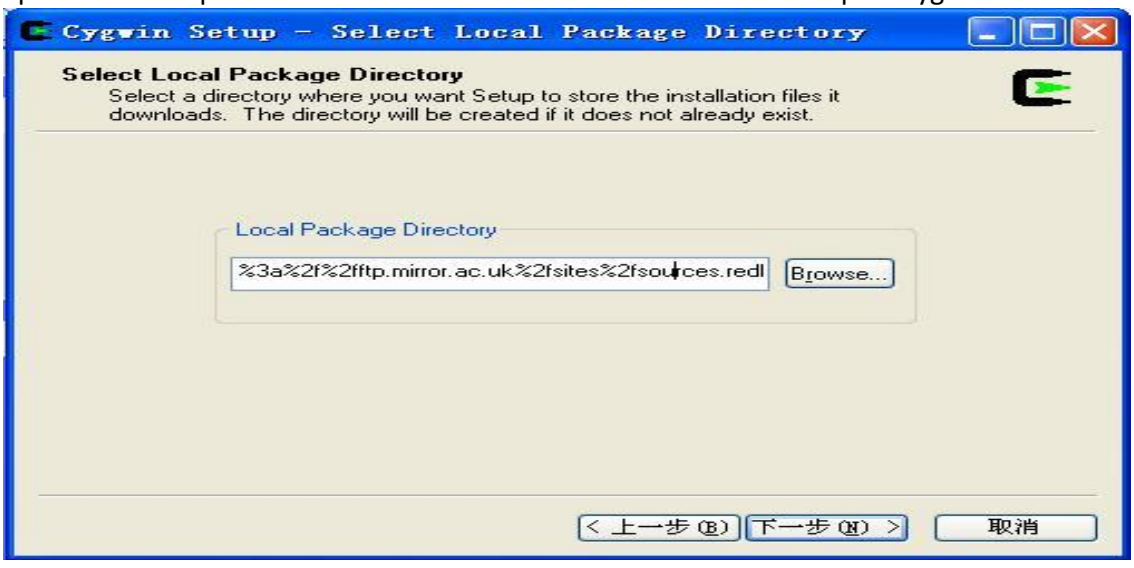
- Select "install local directory"



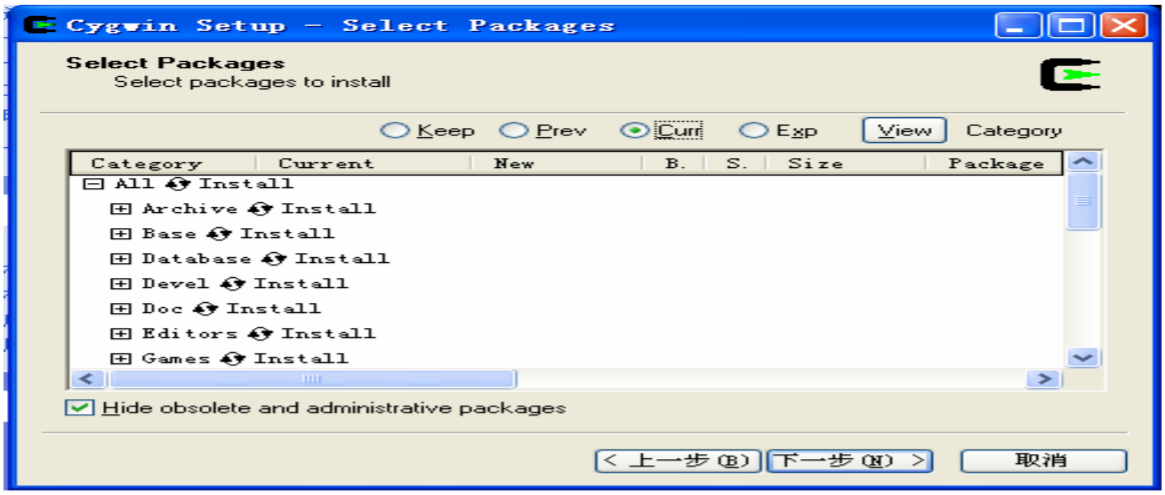
- Select browse for “cygwin” file (I selected as default).



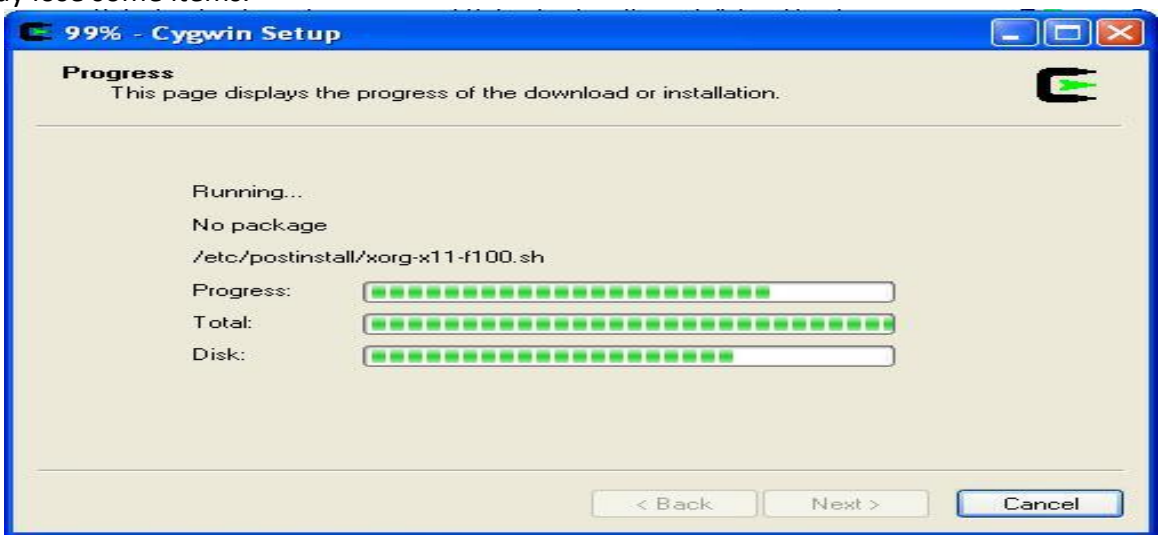
- Install
  “C:\NS2\NS-2.29 Install files\Cygwin files\ftp%3a%2f%2fftp.mirror.ac.uk%2fsites%2fsources.redhat.com%2fftp%2fcygwin”



- Select to install all

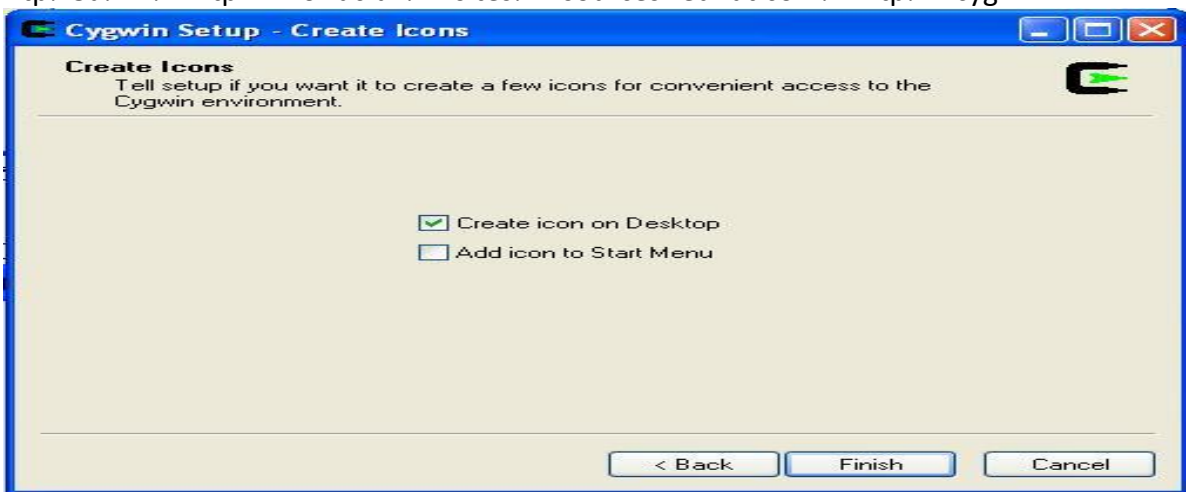


You can select “install”, “default”, “uninstall”, “install”, and “reinstall” behind the first line “all”. If we select “install”, then all sub items will be automatically selected. Otherwise you may lose some items.



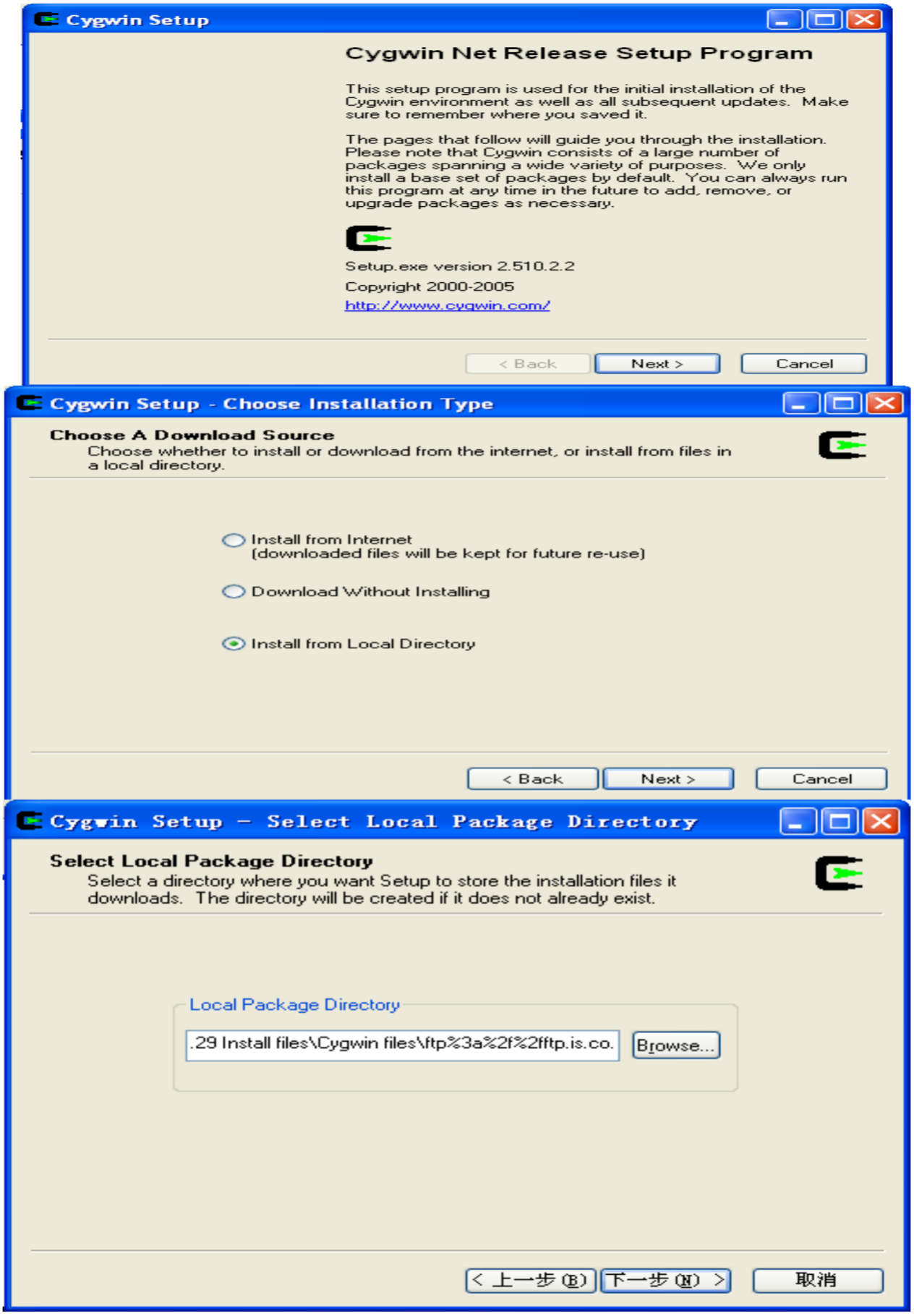
- Finish in installing

“ftp%3a%2f%2fftp.mirror.ac.uk%2fsites%2fsources.redhat.com%2fftp%2fcygwin”

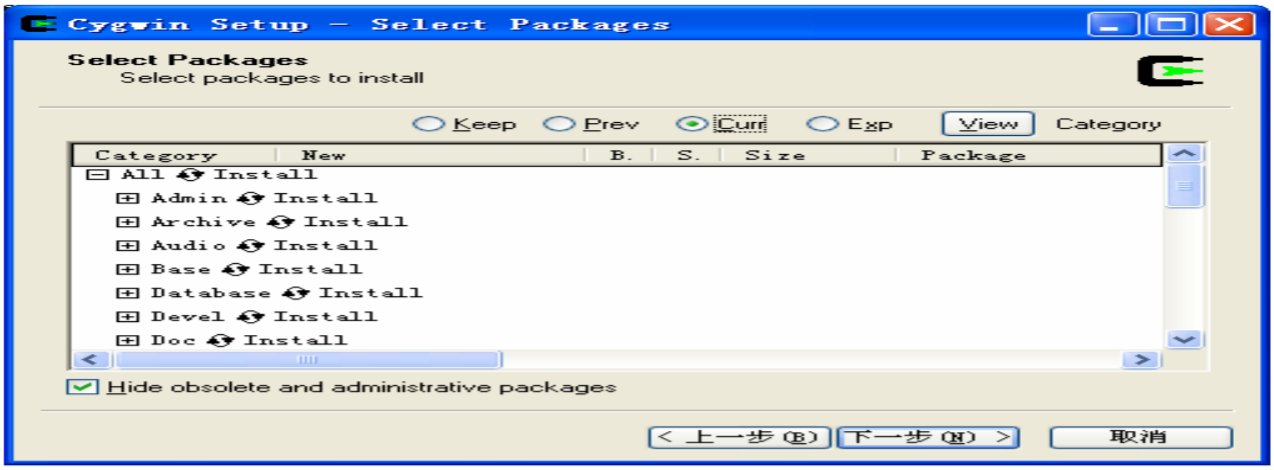


- Install “Install

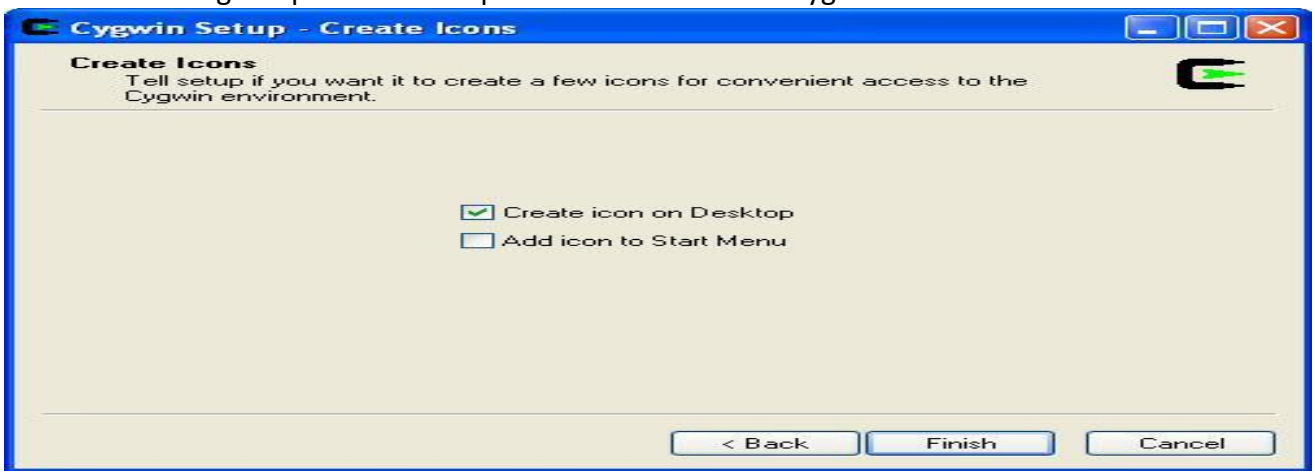
“C:\NS2\NS-2.29 Install files\Cygwin files \ ftp%3a%2f%2fftp.is.co.za%2fmirrors%2fcygwin”.



- Select to install all.



- Finish in stalling “ftp%3a%2f%2fftp.is.co.za%2fmirrors%2fcygwin”

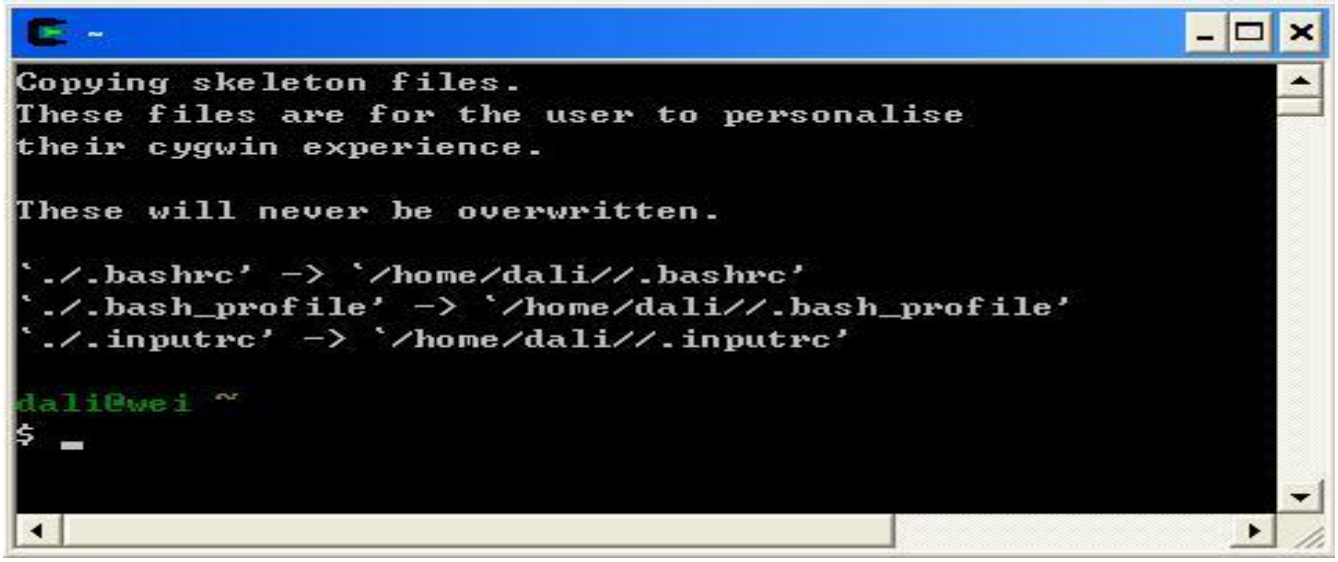


2. Install NS2

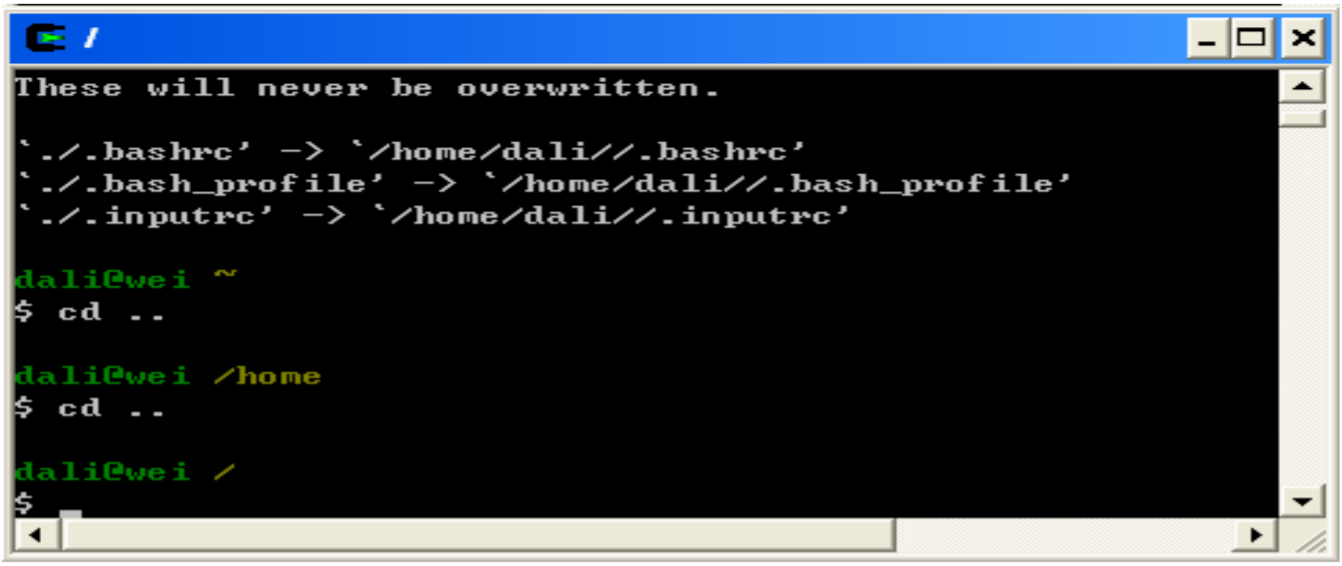
a. Copy ns-allinone-2.29.2.tar to directory c:/cygwin/usr/local

b. Unzip “ns-allinone-2.29.2.tar”

c. Click on desktop icon “cygwin”



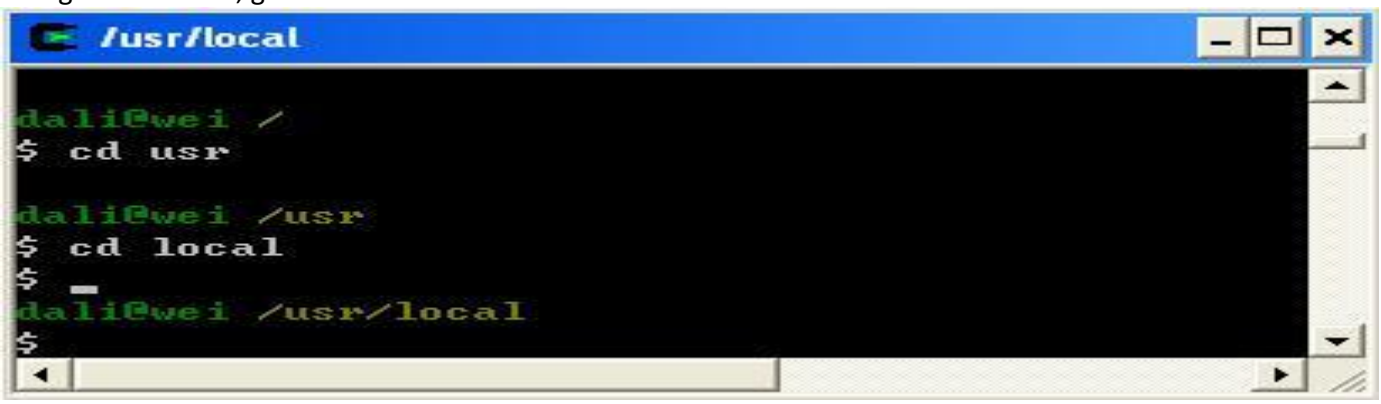
d. Type “cd ..” to go to the upper folder(“cd” must be low case. And there is one space between “d” and “.”)



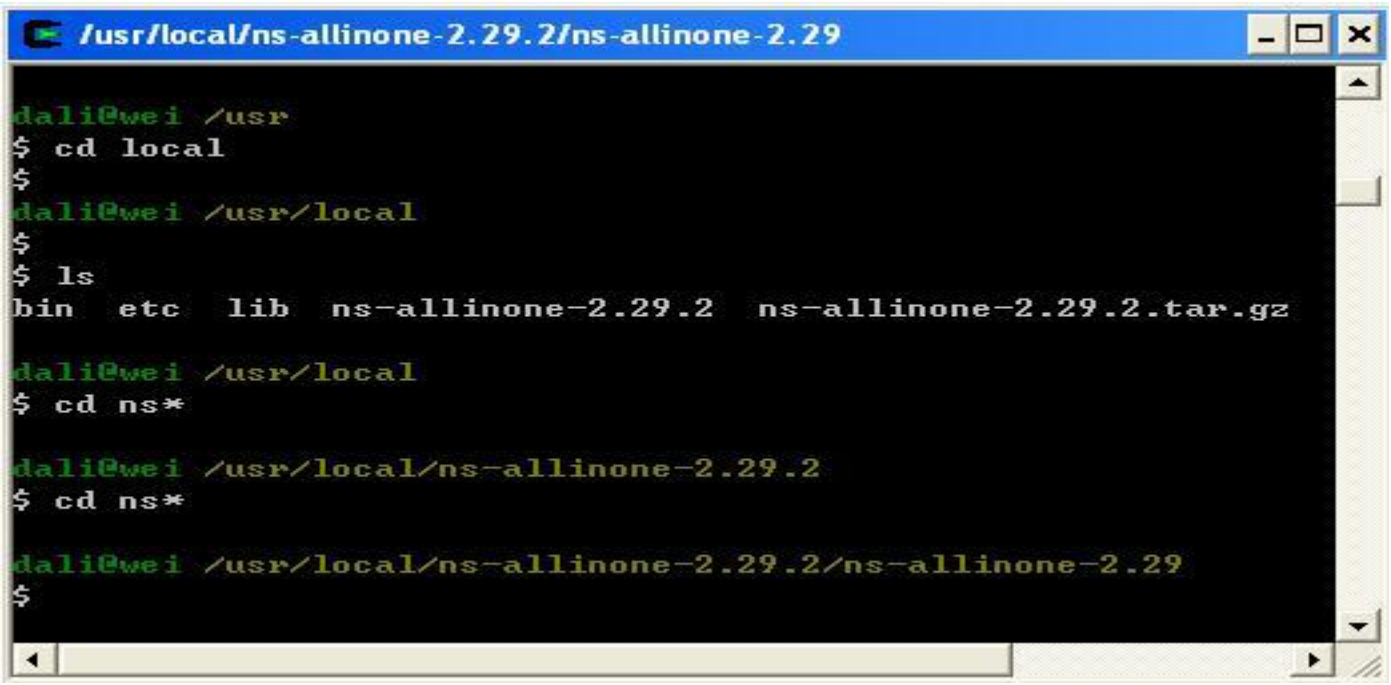
e. Type “cd ..” again

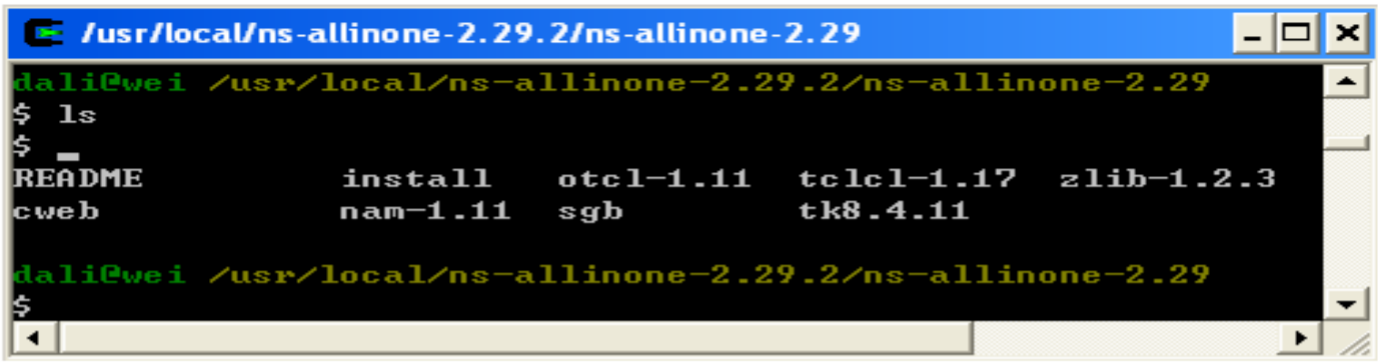
f. cd usr”, go to folder “usr”

g. “cd local”, go to folder “local”

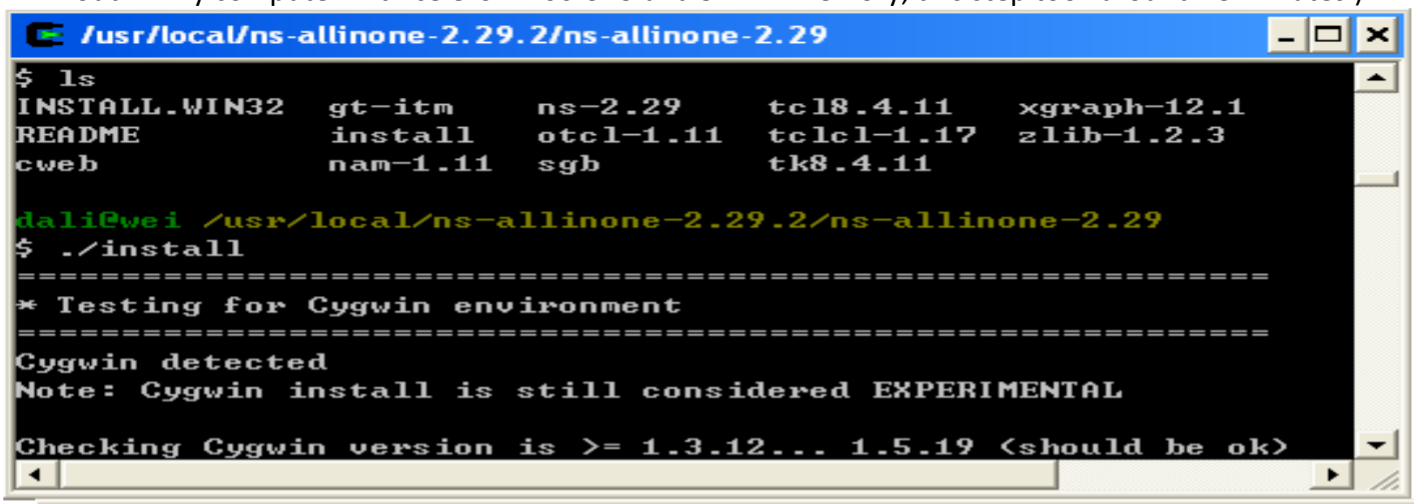


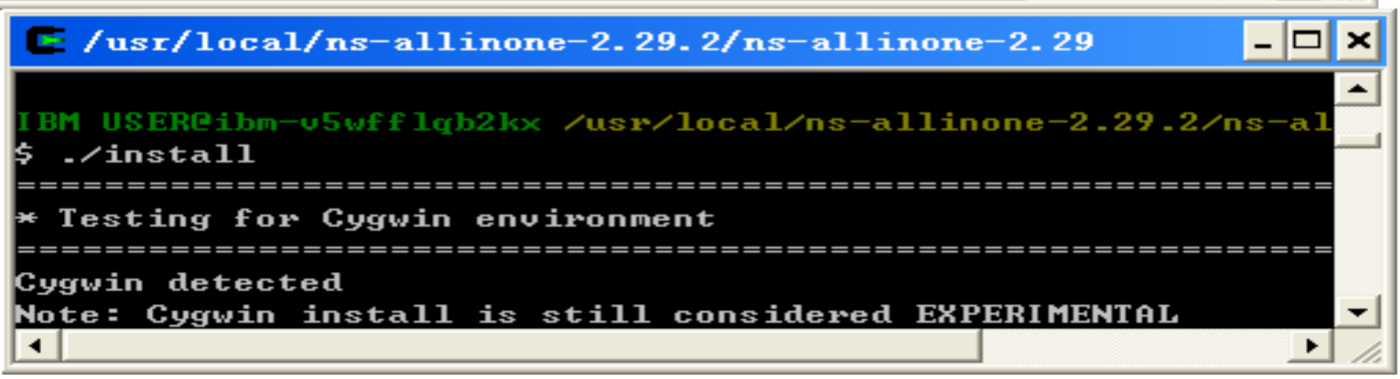
h. Find the “install.exe” file



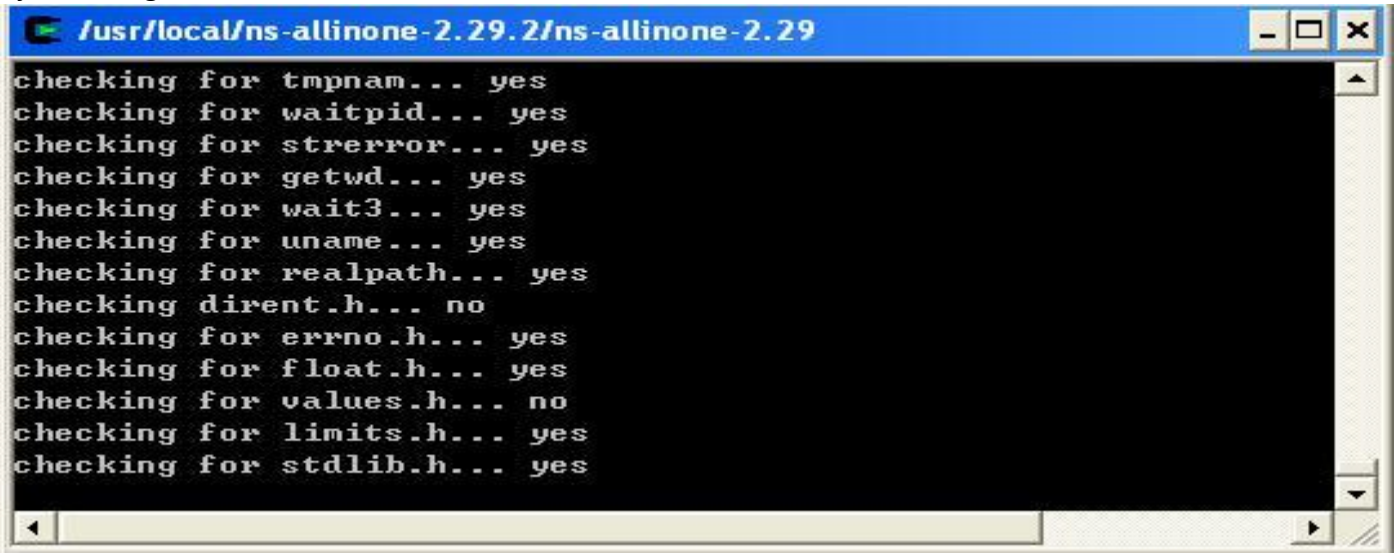


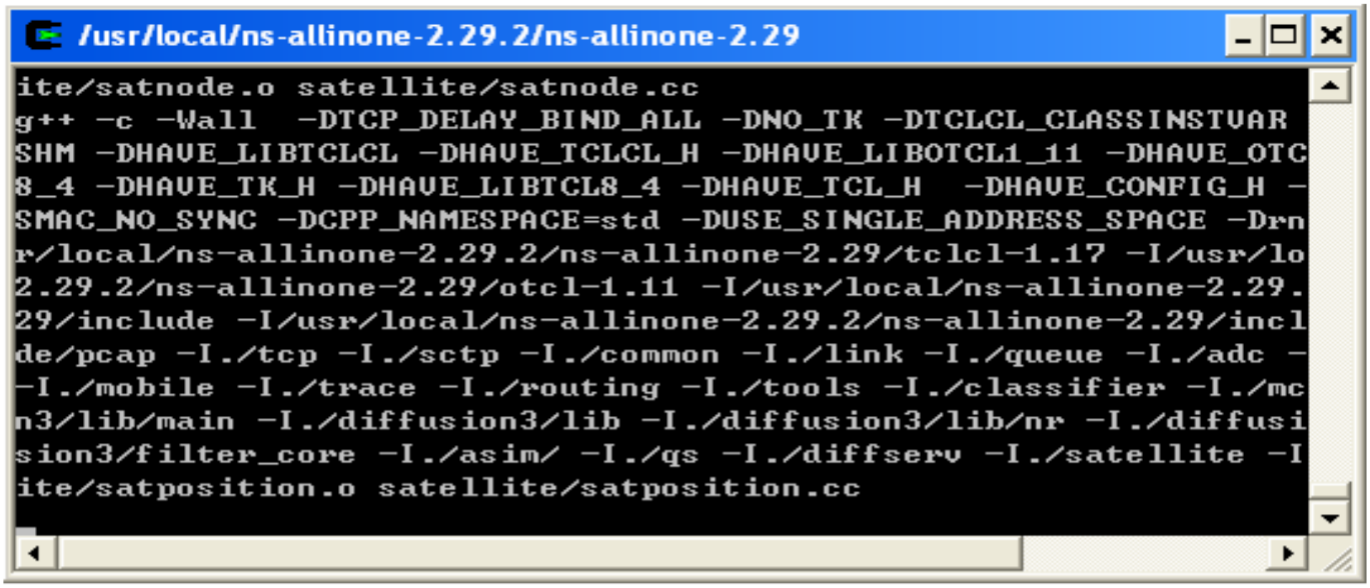
i. Start to run the installation "./install" (Attention: After finish installing, some comments will come out. In my computer with celeron 2.66 CPU and 512 M Memory, this step took around 75 minutes.)





j. Installing





3. Configure system variables and library paths

a. Installing is finished and the following window appears: Make sure to copy all these contents.

```
Please put /usr/local/ns-allinone-2.29.2/ns-allinone-2.29/bin:/usr/local/ns-alli
none-2.29.2/ns-allinone-2.29/tcl8.4.11/unix:/usr/local/ns-allinone-2.29.2/ns-all
inone-2.29/tk8.4.11/unix
into your PATH environment; so that you'll be able to run itm/tclsh/wish/xgraph.


IMPORTANT NOTICES:

(1) You MUST put /usr/local/ns-allinone-2.29.2/ns-allinone-2.29/otcl-1.11, /usr/
local/ns-allinone-2.29.2/ns-allinone-2.29/lib,
    into your LD_LIBRARY_PATH environment variable.
    If it complains about X libraries, add path to your X libraries
    into LD_LIBRARY_PATH.
    If you are using csh, you can set it like:
                setenv LD_LIBRARY_PATH <paths>
    If you are using sh, you can set it like:
                export LD_LIBRARY_PATH=<paths>

(2) You MUST put /usr/local/ns-allinone-2.29.2/ns-allinone-2.29/tcl8.4.11/librar
y into your TCL_LIBRARY environmental
    variable. Otherwise ns/nam will complain during startup.

(3) [OPTIONAL] To save disk space, you can now delete directories tcl8.4.11
    and tk8.4.11. They are now installed under /usr/local/ns-allinone-2.29.2/ns-
allinone-2.29/{bin,include,lib}

After these steps, you can now run the ns validation suite with
cd ns-2.29; ./validate

For trouble shooting, please first read ns problems page
http://www.isi.edu/nsnam/ns/ns-problems.html. Also search the ns mailing list ar
chive
for related posts.


dali@wei /usr/local/ns-allinone-2.29.2/ns-allinone-2.29
$
```