Scilab Textbook Companion for Data Communications And Networking by B. A. Forouzan[1]

Created by
Akshatha.m
B.Tech
Computer Engineering
NITK,Surathkal
College Teacher
K Vinay Kumar
Cross-Checked by
Ganesh R

May 23, 2016

[1]Funded by a grant from the National Mission on Education through ICT, http://spoken-tutorial.org/NMEICT-Intro. This Textbook Companion and Scilab codes written in it can be downloaded from the "Textbook Companion Project" section at the website http://scilab.in

Book Description

Title: Data Communications And Networking

Author: B. A. Forouzan

Publisher: Tata McGraw - Hill Education, New York

Edition: 4

Year: 2007

ISBN: 9780072967753

Scilab numbering policy used in this document and the relation to the above book.

Exa Example (Solved example)

Eqn Equation (Particular equation of the above book)

AP Appendix to Example(Scilab Code that is an Appednix to a particular Example of the above book)

For example, Exa 3.51 means solved example 3.51 of this book. Sec 2.3 means a scilab code whose theory is explained in Section 2.3 of the book.

Contents

List of Scilab Codes

List of Figures

Chapter 2

Network Models

Scilab code Exa 2.1 Bus topology LAN

```
clear;
clc;
disp("-------------Example 2.1-------------")
// example explanation
printf("A node with physical address 10 sends a
    frame to a node with physical address 87. The two
     nodes are connected by a link (bus topology LAN)
    .\nAt the data link layer, this frame contains
    physical (link) addresses in the header.The
    trailer contains extra bits needed for error
    detection.\nThe computer with physical address 10
     is the sender, and the computer with physical
    address 87 is the receiver.\nThe header, among
    other pieces of information, carries the receiver
     and the sender physical (link) The destination
    address, 87 comes \nbefore the source address 10.
     In a bus topology, the frame is propagated in
    both directions (left and right). The frame
    propagated\nto the left dies when it reaches the
    end of the cable if the cable end is terminated
    appropriately. The frame propagated to the right
```

```
    is sent\nto every station on the network. Each
    station with a physical addresses other than 87
    drops the frame because the destination address
    in the frame\ndoes not match its own physical
    address. The intended destination computer,
    however, finds a match between the destination
    address\nin the frame and its own physical
    address. The frame is checked, the header and
    trailer are dropped, and the data part is
    decapsulated and\ndelivered to the upper layer.")
    ;
sender_address="10";
reciever_address="87";
clf();
xname("—————————————Example 2.1——————————————");
// display the figure
xset("thickness",2.5);
xpoly([.049 .86],[.48 .48]);
xpoly ([.35 .35],[.73 .48]);
xpoly ([.55 .55],[.73 .48]);
xset("thickness",1);
xset("font size",2);
xstring(.36,.6,"Destination address does not");
xstring(.36,.56,"match;the packet is dropped");
xstring(.25,.63,"Trailer");
xstring(.001,.63,"Destination address");
xstring(.001,.5,"Source address");
xpoly([.1 .11],[.52 .55]);
xpoly([.22 .25],[.6 .63]);
xpoly([.05 .08],[.63 .6]);
xset("font size",4);
xstring(0.1,.75,["10 -  Sender"]);
xstring(0.7,.75,["87 -  Reciever"]);
xstring(.32,.75,"Station 28");
xstring(.52, .75,"Station 53");
xstring(.44,.42,"LAN");
xrect(.13,.7,.07,.05);
xstringb(.13,.65,["Data"],.07,.05);
```

```
xrect(.73,.7,.07,.05);
xstringb(.73,.65,["Data"],.07,.05);
xrects([ 0.07 .13 .2;.6 .6 .6;.06 .07 .03;.06 .06
   .06]);
xstring(0.07,.55,["87"]);
xstring(0.1,.55,["10"]);
xstring(0.2,.55,"T2");
xrect(.135,.595,.06,.05);
xstring(.137,.555,["Data"]);
xstring(0.67,.55,["87"]);
xstring(0.7,.55,["10"]);
xstring(.8,.55,"T2");
xrect(.735,.595,.06,.05);
xstring(.737,.555,["Data"]);
xset("font size",6);
xstring(.43,.51,". . .");
xfrect(.009,.5,.04,.05);
xfrect(.86,.5,.04,.05);
xpoly([.17 .17],[.75 .7]);
xpoly([.77 .77],[.75 .7]);
xpoly([.17 .17],[.65 .6]);
xpoly([.77 .77],[.65 .6]);
xpoly([.17 .17],[.54 .48]);
xpoly([.77 .77],[.54 .48]);
xarrows([.74 .8],[.45 .45],.5);
xarrows([.76 .76],[.45 .54],.5);
xarrows([.34 .4],[.45 .45],.5);
xarrows([.36 .36],[.45 .57],.5);
xarrows([.54 .6],[.45 .45],.5);
xarrows([.56 .56],[.45 .57],.5);
xarrows([.14 .2],[.45 .45],.5);
xarrows([.14 .08],[.45 .45],.5);
xpoly([.16 .16],[.45 .54]);
xset("line style",2);
xrects([.67 .73 .8;.6 .6 .6;.06 .07 .03;.06 .06
   .06]);
```

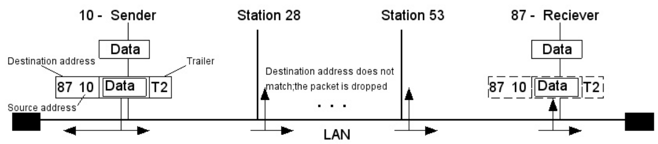

Figure 2.1: Bus topology LAN

Scilab code Exa 2.2 physical address example

```
clear;
clc;
disp("————————————————Example 2.2————————————————")
disp("07:01:02:01:2C:4B ")          //display the example
printf("This is an example of a 6−byte (12 hexadecimal digits) physical address.")
```

Scilab code Exa 2.3 LAN and router communication

```
clear;
clc;
```

```
disp("---------------Example 2.3---------------")
// display the example
printf("Each device(computer or router) has a pair of addresses (logical and physical) for each connection. In this\ncase, each computer is connected to only one link and therefore has only one pair of addresses. Each router,\nhowever, is connected to three networks . So each router has three pairs of addresses,one for each\nconnection. Although it may obvious that each router must have a separate physical address for each connection,\n;it may not be obvious why it needs a logical address for each connection.");
printf("\n\nIn this example,the computer with logical address A and physical address 10 needs to send a packet to\nthe computer with logical address P and physical address 95. The sender encapsulates its data\nin a packet at the network layer and adds two logical addresses (A and P). The logical source address comes before the logical destination address .\nThe network layer, however, needs to find the physical address of the next hop before the packet can be\ndelivered. The network layer consults its routing table and finds the logical address of the next hop (router 1) to be F.\nThe ARP finds the physical address of router 1 that corresponds to the logical address of 20. Now the network layer passes\nthis address to the data link layer, which in turn, encapsulates the packet with physical destination address 20 and physical source address 10.\n\n");
printf("The frame is received by every device on LAN 1, but is discarded by all except router 1, which finds that\nthe destination physical address in the frame matches with its own physical address. The router decapsulates the\npacket from the frame to read the logical
```

```
destination address P. Since the logical destination address does not match the\nrouters logical address, the router knows that the packet needs to be forwarded. The router consults its routing table\nand ARP to find the physical destination address of the next hop (router 2), creates a new frame, encapsulates the packet, and sends it to router 2.\n\n");
printf("The source physical address changes from 10 to 99. The destination physical address changes from 20 (router 1 physical address) to 33\n(router 2 physical address). The logical source and destination addresses must remain the same; otherwise the packet will be lost.\n\n");
printf("At router 2 we have a similar scenario. The physical addresses are changed, and a new frame is sent to the destination computer.\nWhen the frame reaches the destination, the packet is decapsulated. The destination logical address P matches the logical address\nof the computer. The data are decapsulated from the packet and delivered to the upper layer.")
```

Scilab code Exa 2.4 Two computers communicating

```
clear;
clc;
clf();
xname("---------------Example 2.4------------------");
// display the figure
xset("color",0);
xset("font size",3);
xrect(-.05,1,.09,.09);
xrect(0.06,1,.09,.09);
xrect(0.17,1,.09,.09);
```

```
xrect(0.72,1,.09,.09);
xrect(0.83,1,.09,.09);
xstring(-.02,1,["a"]);
xstring(0.09,1,["b"]);
xstring(.20,1,["c"]);
xstring(.75,1,["j"]);
xstring(.86,1,["k"]);
xstring(0.06,.75,["A —  Sender"]);
xstring(0.8,.75,["P —  Reciever"]);
xrect(.06,.7,.07,.05);
xstringb(.06,.65,["Data"],.07,.05);
xrect(.8,.7,.07,.05);
xstringb(.8,.65,["Data"],.07,.05);
xstring(.13,.65,["——————————————————————————————— Application Layer ——————————————————————————————"])
xrects([ 0 .03 .06;.6 .6 .6;.03 .03 .07;.06 .06 .06]);
xrects([.74 .77 .80;.6 .6 .6;.03 .03 .07;.06 .06 .06]);
xstring(0.005,.55,["a"]);
xstring(0.035,.55,["j"]);
xrect(.065,.595,.06,.05);
xstring(.074,.555,["Data"]);
xstring(0.745,.55,["a"]);
xstring(0.775,.55,["j"]);
xrect(.805,.595,.06,.05);
xstring(.82,.555,["Data"]);
xstring(.13,.55,["——————————————————————————————— Transport Layer ————————————————————————"])
xrect(-.06,.465,.19,.065);
xrect(.68,.465,.19,.065);
xrects([ -.06 -.03 0;.47 .47 .47;.03 .03 .13;.06 .06 .06]);
```

```
xrects([.68 .71 .74;.47 .47 .47;.03 .03 .13;.06 .06 .06]);
xrects([ 0 .03 .06;.46 .46 .46;.03 .03 .07;.057 .057 .057]);
xrects([.74 .77 .80;.46 .46 .46;.03 .03 .07;.057 .057 .057]);
xstring(0.005,.41,["a"]);
xstring(0.035,.41,["j"]);
xrect(.065,.455,.06,.05);
xstring(.074,.415,["Data"]);
xstring(0.745,.41,["a"]);
xstring(0.775,.41,["j"]);
xrect(.805,.455,.06,.05);
xstring(.82,.415,["Data"]);
xstring(-.055,.42,["A"]);
xstring(-.02,.42,["P"]);
xstring(.69,.42,["A"]);
xstring(.72, .42,["P"]);
xstring(.13,.42,["------------------------------------------------ Network Layer ----------------------------------------"]);
xrect(-.06,.305,.19,.075);
xrect(.68,.305,.19,.075);
xrects([ -.06 -.03 0;.3 .3 .3;.03 .03 .13;.06 .06 .06]);
xrects([.68 .71 .74;.3 .3 .3;.03 .03 .13;.06 .06 .06]);
xrects([ 0 .03 .06;.297 .297 .297;.03 .03 .07;.057 .057 .057]);
xrects([.74 .77 .80;.297 .297 .297;.03 .03 .07;.057 .057 .057]);
xstring(0.005,.24,["a"]);
xstring(0.035,.24,["j"]);
xrect(.065,.29,.06,.05);
xstring(.074,.245,["Data"]);
xstring(0.745,.24,["a"]);
xstring(0.775,.24,["j"]);
```

```
xrect(.805,.29,.06,.05);
xstring(.82,.245,["Data"]);
xstring(-.055,.25,["A"]);
xstring(-.02,.25,["P"]);
xstring(.69,.25,["A"]);
xstring(.72, .25,["P"]);
xset("color",2);
xfrect(-.09,.305,.03,.075);
xfrect(.65,.305,.03,.075);
xfrect(.13,.305,.03,.075);
xfrect(.87,.305,.03,.075);
xstring(-.087,.24,["H2"]);
xstring(.13,.24,["T2"]);
xstring(.655,.24,["H2"]);
xstring(.87,.24,["T2"]);
xstring(.155,.26,["————————————————————Data link Layer—————————————————"]);
xset("color",0);
xstring(.38,.05,["Internet"]);
xarc(0.3,.15,.2,.2,0,360*64);
xpoly([0.09,0.09],[.75,.7]);
xpoly([0.09,0.09],[.65,.6]);
xpoly([0.09,0.09],[.54,.47]);
xpoly([0.09,0.09],[.4,.3]);
xarrows([0.1,.3],[.23 .08]);
xpoly([0.84,0.84],[.75,.7]);
xpoly([0.84,0.84],[.65,.6]);
xpoly([0.84,0.84],[.54,.47]);
xpoly([0.84,0.84],[.4,.3]);
xpoly([.5 .8],[.08 .23]);
xpoly([-.02 .06],[.91 0.77]);
xarrows([.82 .75],[.77 0.91]);
disp("———————Example 2.4————————")
// display the text
printf("Figure shows two computers communicating via the Internet. The sending computer is running three processes at\nthis time with port addresses
```

```
a, b, and c. The receiving computer is running
two processes at this time with port addresses j
and k.\nProcess a in the sending computer needs
to communicate with process j in the receiving
computer.\nNote that although both computers are
using the same application, FTP, the port
addresses are different because one is a client\
nprogram and the other is a server program. To
show that data from process a need to be
delivered to process j, and not k,\nthe transport
 layer encapsulates data from the application
layer in a packet and adds two port addresses (a
and j), source and destination.\nThe packet from
the transport layer is then encapsulated in
another packet at the network layer with logical
source and\ndestination addresses (A and P).
Finally, this packet is encapsulated in a frame
with the physical source and destination
addresses of the next\nhop. We have not shown the
 physical addresses because they change from hop
to hop inside the cloud designated as the
Internet. Note\nthat although physical addresses
change from hop to hop, logical and port\
naddresses remain the same from the source to
destination.");
```

Scilab code Exa 2.5 single number portaddress

```
clear;
clc;
disp("----------------Example 2.5----------------")
disp("753 -  A 16-bit port address represented as
   one single decimal number.")
```

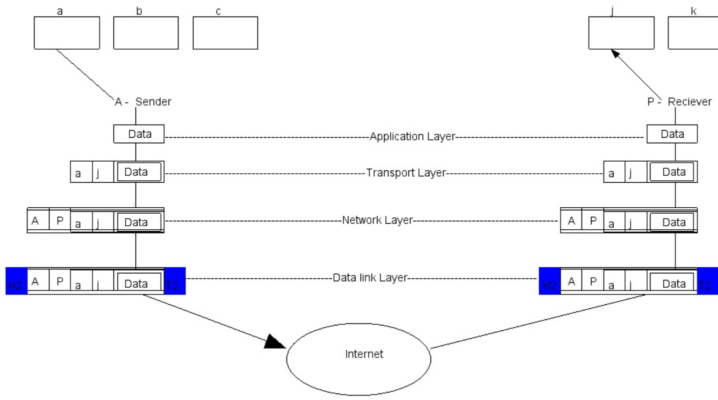

Figure 2.2: Two computers communicating

Chapter 3

Data and Signals

Scilab code Exa 3.1 Power in houses

```
clear;
clc;
disp("—————————————Example 3.1———————————————")
printf("The power in our house can be represented by
    a sine wave with a peak amplitude of 155 to 170
   V.\nHowever, it is common knowledge that the
   voltage of the power in U.S. homes is 110 to 120
   V.");
```

Scilab code Exa 3.2 Battery voltage

```
clear;
clc;
disp("—————————————Example 3.2———————————————")
printf("The voltage of battery is a constant; this
   constant value can be considered a sine wave .\
   nFor example, the peak value of an AA battery is
   normally 1.5 V.")
```

Scilab code Exa 3.3 House power period

```
clear;
clc;
disp("———————————————Example 3.3———————————————")
// frequency of power used at home = 60Hz
f=60
t=1/f // formula to calculate time period
T=t*10^3 // express time period in milliseconds
printf("\n The period of the power for our lights at home is %fs or %fms .",t,T); //display the result
disp("Our eyes are not sensitive enough to distinguish these rapid changes in amplitude.")
```

Scilab code Exa 3.4 Period in microseconds

```
clear;
clc;
disp("———————————————Example 3.4———————————————")
// period = 100ms
T = 100*10^-3*10^6; // multiply with the conversion factor
printf("The period in microseconds = %2.0E microseconds.",T)// display result in microseconds
```

Scilab code Exa 3.5 Frequency from period

```
clear;
clc;
disp("————————————————Example 3.5————————————————")
// period = 100 ms
T= 100*10^-3;
f1=1/T;  // formula to find frequency
f= f1*10^-3; // multiply with conversion factor
printf("The frequency in kHz = %2.1E kHz",f);  // display the final result in kHz
```

Scilab code Exa 3.6 Calculation of phase

```
clear;
clc;
disp("————————————————Example 3.6————————————————")
//offset = 1/6
phase= (1/6)*360;  //formula to calculate phase in degrees
printf("The phase in degrees = %d \n\n",phase);
pr=phase*((2*%pi)/360); // degrees to radians conversion
printf("The phase in radians = %4.3f rad",pr); // display the result in radians
```

Scilab code Exa 3.7 Time and frequency domains

```
clear;
clc;
clf();
xname("————————————————Example 3.7————————————————");
//display the figure
subplot(121) // time domain plot
a1=gca();
```

```
a1.y_label.text="Amplitude"; // y-axis
a1.y_label.font_style = 3;
a1.y_label.font_size = 4;
a1.y_label.foreground = 3;
a1.y_location = "left";
a1.x_label.text="Time"; // x-axis
a1.x_label.font_style = 3;
a1.x_label.font_size = 4;
a1.x_label.foreground = 3;
a1.x_location = "middle";
a1.data_bounds=[0,-15;1,15];
x=[0:.000125:1];
// sine waves to be plotted
plot2d(x,10*sin(8*x));
plot2d(x,5*sin(16*x));
plot2d(x,15*sin(%pi/2 - 0*x));
xarrows([.5 .01],[-12 -12],5);
xarrows([.5 .99],[-12 -12],5);
xpoly([1 1],[-3 -12.5]);
xset("font size",3)
xstring(.5,-11,"1s")
xset("font size",2)
xstring(.02,-13.5,"a. Time-domain representation of
    three sine waves with frequencies 0, 8, and 16")
subplot(122) // frequency domain plot
a1=gca();
xarrows([0 1],[.5 .5],.5)
xarrows([0 0],[.5 1],.5)
xset("font size",3)
xstring(-.01,1,"Amplitude") // y-axis
xstring(1,.45,"Frequency") // x-axis
xstring(.02,.4,"b. Frequency-domain representation
    of the same three signals")
xstring(0,.45,"0
                                                   8
                                                     16")
xstring(-.05,.7,"5")
xstring(-.05,.8,"10")
```

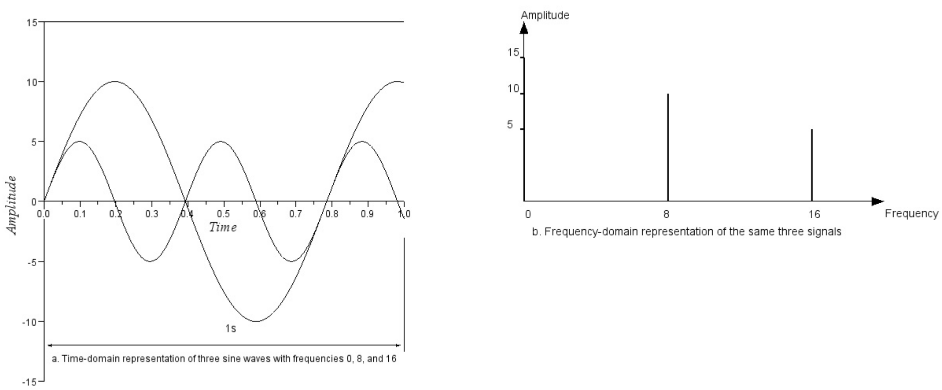

Figure 3.1: Time and frequency domains

```
xstring(-.05,.9,"15")
xset("thickness",2.5)
xpoly([0 0],[.5 .9])
xpoly([.4 .4],[.5 .8])
xpoly([.8 .8],[.5 .7])
xpoly([0 -.01],[.7 .7])
xpoly([0 -.01],[.8 .8])
xpoly([0 -.01],[.9 .9])
```

Scilab code Exa 3.8 Periodic composite signal

```
clear;
clc;
clf();
xname("—————————————Example 3.8——————————————");
a1=gca();
// define the properties of the axes
a1.x_location = "middle";
```

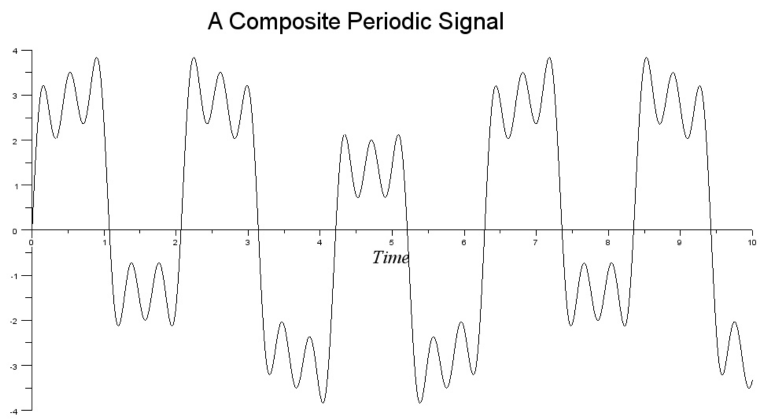

Figure 3.2: Periodic composite signal

```
a1.x_label.text="Time"; // display the quantity along x-axis
a1.x_label.font_style = 3;
a1.x_label.font_size = 5;
a1.x_label.foreground = 3;
a1.title.text="A Composite Periodic Signal"  // display title
a1.title.foreground = 12;
a1.title.font_size = 6;
x=[0:.001:10]; // x-range
plot2d(x,sin(x)+3*sin(3*x)+sin(9*x)+sin(15*x)); // equation to be plotted
```

Scilab code Exa 3.9 Nonperiodic composite signal

```
clear;
clc;
disp("--------------Example 3.9--------------")
```

```
fl=0; // 0kHz
fh=4; // 4kHz
// example explanation
printf("Figure shows a nonperiodic composite signal. It can be the signal created by a microphone or a telephone set when a word or two\nis pronounced. In this case, the composite signal cannot be periodic, because that implies that we are repeating the same word or\nwords with exactly the same tone.\n\n");
printf("In a time-domain representation of this composite signal, there are an infinite number of simple sine frequencies.\nAlthough the number of frequencies in a human voice is infinite, the range is limited. A normal human being\ncan create a continuous range of frequencies between %d and %d kHz.",fl,fh);
printf("\n\nThe frequency decomposition of the signal yields a continuous curve. There are an infinite number of frequencies between %2.1f\nand %5.1f (real values). To find the amplitude related to frequency f, draw a vertical line at f to intersect the envelope curve.\nThe height of the vertical line is the amplitude of the corresponding frequency.",fl,fh*10^3);
clf();
xname("---------------Example 3.9----------------");
//display the figure
subplot(121) // time domain plot
a1=gca();
x=[0:.001:10]; // x-range
a1.title.text="a. Time domain";
a1.x_location = "middle";
a1.x_label.text="Time"; // display the quantity along x-axis
a1.y_label.text="Amplitude"; // display the quantity along y-axis
plot(x,sin(.5*cos(x))+.8*sin(3*x)+sin(9*x)+sin(57*x)
```

```
+cos(57*x)+sin(%pi/7*x)); // equation to be plotted

subplot(122) // frequency domain plot
a1=gca();
p=[0:.01:%pi/5]; // x-range
a1.title.text="b. Frequency domain";
a1.x_location = "bottom";
a1.x_label.text="Frequency"; // display the quantity along x-axis
a1.y_label.text="Amplitude";// display the quantity along y-axis
a1.data_bounds=[0,0;2,5];
// equations for the plot
plot(p,%e^p+2^p-2);
xset("color",2);
xpoly([.62 1.5],[1.4 1.4]);
xarc(1.02,1.54,.5,.5,0,64*25);
xpoly([1.52 1.8],[1.31 0])
xpoly([.62 .62],[0 1.4])
xfarc(.6,1.41,.05,.05,0,64*360)
xset("font size",2)
xstring(.6,1.5,"Amplitude for sine wave of frequency f");
xstring(.59,0,"f

4 kHz")
```

Scilab code Exa 3.10 Bandwidth spectrum

```
clear;
clc;
disp("————————————————Example 3.10————————————————")
```

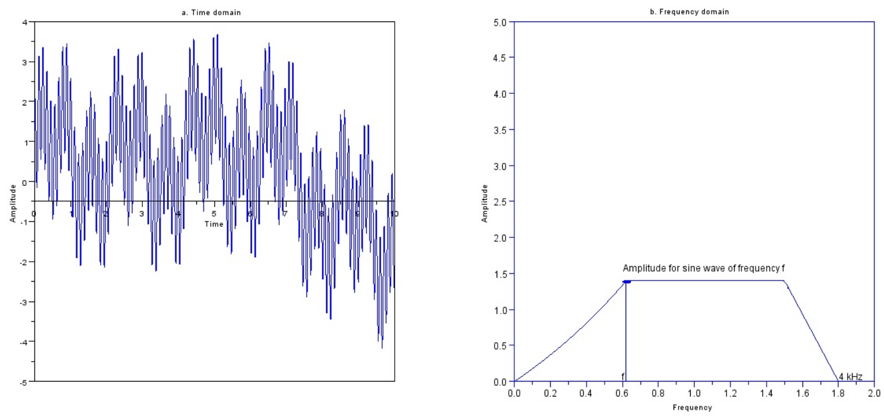

Figure 3.3: Nonperiodic composite signal

```
fl=100;   //lowest frequency in Hz
fh=900;   //highest frequency in Hz
B=fh-fl; // formula to calculate bandwidth
n=5; // nymber of sine waves
dif_f=(fh+fl)/n; // difference between each spike
f=[]; // spikes
for i=0:4
    f(i+1)=i*dif_f+fl; // formula
end
printf("The bandawidth = %d Hz\n",B); // display the result
printf("The spectrum has only %d spikes, at %d, %d, %d, %d and %d Hz.",n,f(1),f(2),f(3),f(4),f(5));
clf();
xname("-----------------Example 3.10-------------------");
// display the figure
xarrows([.1 1.1],[.2 .2],.5);
xarrows([.1 .1],[.2 .9],.5);
xset("font size",5);
xstring(.1,.9,"Amplitude"); // y axis
xstring(1.03,.1,"Frequency"); // x axis
xset("font size",4);
```

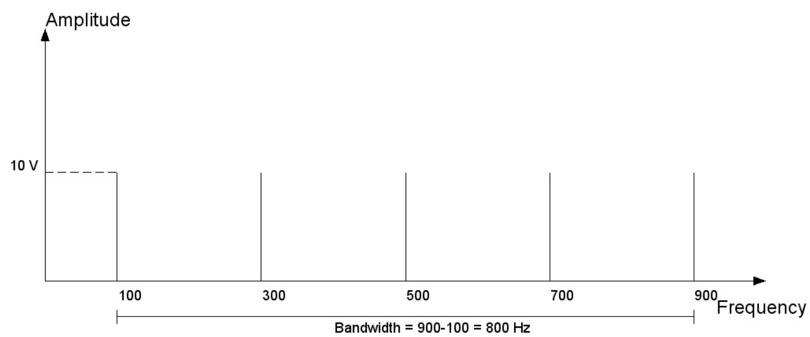

Figure 3.4: Bandwidth spectrum

```
xstring(.2,.14,"100");
xstring(.4,.14,"300");
xstring(.6,.14,"500");
xstring(.8,.14,"700");
xstring(1,.14,"900");
xpoly([.2 .2],[.2 .5]);
xpoly([.4 .4],[.2 .5]);
xpoly([.6 .6],[.2 .5]);
xpoly([.8 .8],[.2 .5]);
xpoly([1 1],[.2 .5]);
xpoly([.2 1],[.1 .1]);
xpoly([.2 .2],[.08 .12]);
xpoly([1 1],[.08 .12]);
xstring(0.05,.5,"10 V");
xset("line style",2);
xpoly([.1 .2],[.5 .5],"lines");
xstring(.5,.05,"Bandwidth = 900-100 = 800 Hz");
```

Scilab code Exa 3.11 Bandwidth spectrum 2

```
clear;
clc;
disp("−−−−−−−−−−−−−−−−Example 3.11−−−−−−−−−−−−−−−−−")
B=20;   //bandwidth in Hz
fh=60;   //highest frequency in Hz
fl=fh-B; // formula to calculate lowest frequency
printf("The lowest frequency = %d Hz\nThe spectrum
    contains all integer frequencies which is shown
    as a series of spikes in the figure.",fl); //
    display the result
clf();
xname("−−−−−−−−−−−−−−−−Example 3.11−−−−−−−−−−−−−−−−−−");
xarrows([.1 .9],[.2 .2],.5);
xset("font size",5);
xstring(.92,.1,"Frequency(Hz)");
xset("thickness",2);
x=linspace(.2,.8,21);
for i=1:21
    xpoly([x(i) x(i)],[.2 .5]);
end
xset("font size",3);
for i=1:21
    s=40+i-1;
    xstring(x(i),.17,string(s));
end
xpoly([.2 .8],[.1 .1]);
xpoly([.2 .2],[.08 .12]);
xpoly([.8 .8],[.08 .12]);
xstring(.4,.1,"Bandwidth = 60−40 = 20 Hz");
```

Scilab code Exa 3.12 Frequency domain of signal

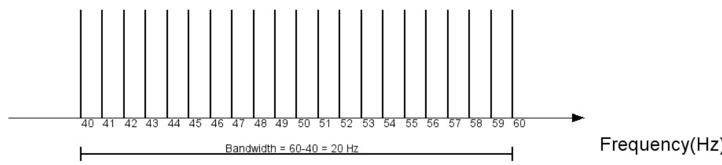

Figure 3.5: Bandwidth spectrum 2

```
clear;
clc;
disp("--------------Example 3.12-----------------");
B=200; // kHz
mf=140; // kHz - middle frequency
fh=(2*mf+B)/2; // formula for higher frequency
fl=fh-B; // formula for lower frequency
printf("The lowest frequency is %d kHz and highest frequency is %d kHz.",fl,fh);// display the result
//display the figure
clf();
xname("--------------Example 3.12-----------------");
xarrows([.2 1],[.2 .2],.5);
xarrows([.2 .2],[.2 1],.5);
xpoly([.3 .55],[.2 .6]);
xpoly([.55 .8],[.6 .2]);
xset("line style",2);
xpoly([.55 .2],[.6 .6],"lines");
xset("font size",5);
xstring(.1,1,"Amplitude"); // y axis
xstring(1,.1,"Frequency"); // x axis
```

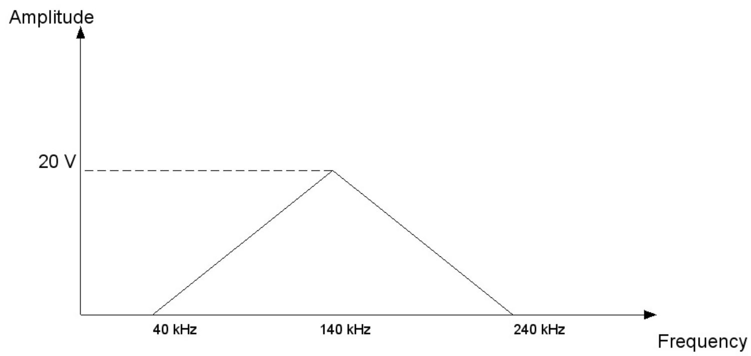

Figure 3.6: Frequency domain of signal

```
xstring(0.14,.6,"20 V");
xset("font size",4);
xstring(.3,.14,"40 kHz");
xstring(.8,.14,"240 kHz");
xstring(.53,.14,"140 kHz");
```

Scilab code Exa 3.13 AM Radio Bandwidth

```
clear;
clc;
disp("————————————Example 3.13————————————")
printf("An example of a nonperiodic composite signal
    is the signal propagated by an AM radio station
   .\nIn the United States, each AM radio station is
    assigned a 10-kHz band width.\nThe total
   bandwidth dedicated to AM radio ranges from 530
   to 1700 kHz.")    // display the example
```

Scilab code Exa 3.14 FM Radio Bandwidth

```
clear;
clc;
disp("—————————————Example 3.14—————————————")
printf("Another example of a nonperiodic composite
    signal is the signal propagated by an FM radio
    station.\nIn the United States, each FM radio
    station is assigned a 200-kHz bandwidth.\nThe
    total bandwidth dedicated to FM radio ranges from
     88 to 108 MHz.")    // display the example
```

Scilab code Exa 3.15 Black and white TV

```
clear;
clc;
disp("—————————————Example 3.15—————————————")
printf("Another example of a nonperiodic composite
    signal is the signal received by an old-fashioned
     analog black-and-white TV.\n");
s=30; // screen is scanned 30 times per second
//screen resolution = 525 x 700
vl=525;
hl=700;
pixels=vl*hl; // total number of pixels
pixels_per_second=pixels*s; // pixels scanned per
    second
cycles_per_second=pixels_per_second/2; // 2 pixels
    per cycle in the worst-case scenario i.e
    alternating black and white pixels
bandwidth = cycles_per_second*10^-6;
b7=bandwidth*0.7; // 70% of the bandwidth
```

```
final =ceil(b7); // final rounded bandwidth
// display the result
printf("The bandwidth needed in the worst-case
   scenario i.e alternating black and white pixels
   where we need to represent \none color by the
   minimum amplitude and the other color by the
   maximum amplitude is %5.4f MHz.\n\n",bandwidth);
printf("This worst-case scenario has such a low
   probability of occurrence that the assumption is
   that we need only 70 percent\nof this bandwidth,
   which is %3.2f MHz. Since audio and
   synchronization signals are also needed, a %d MHz
    bandwidth\nhas been set aside for each black and
    white TV channel.\nAn analog color TV channel
   has a 6-MHz bandwidth.",b7,final);
```

Scilab code Exa 3.16 Bits per level

```
clear;
clc;
disp("----------------Example 3.16----------------")
L=8; // number of levels
bits_per_level=log2(L); // formula to calculate
   number of bits per level
printf("The number of bits per level is %d bits.",
   bits_per_level); // display the result
```

Scilab code Exa 3.17 Bits per level 2

```
clear;
clc;
disp("----------------Example 3.17----------------")
L=9; // number of levels
```

```
bits_per_level=log2(L); // formula to calculate number of bits per level
printf("The number of bits per level is %3.2f bits.\n",bits_per_level);
if(~(bits_per_level/10 == 0))  // if the number of bits is not an integer or power of 2
   printf("This answer is not realistic. The number of bits sent per level needs to be an integer as well as a power of 2.\n");
   r=nextpow2(bits_per_level);   // find nearesr power of 2
   bits=2^r;
   printf("Therefore %d bits can represent one level.",bits); // display result
end
```

Scilab code Exa 3.18 Download text document

```
clear;
clc;
disp("———————————————Example 3.18————————————————")
num_line=24; // number of lines
num_char=80; // number of characters in each line
bits=8; // bits per character
rate = 100; // pages per minute
br=rate*bits*num_char*num_line;  // formula to calculate bit rate
bit_rate=br*10^-6; // multiply with conversion factor
printf("The bit rate of the channel is %4.3f Mbps.",bit_rate); // display the result (answer in the text is 1.636 Mbps,which is wrong)
```

Scilab code Exa 3.19 Bitrate calculation

```
clear;
clc;
disp("————————————Example 3.19————————————")
bandwidth=4000; // 4 kHz
bits =8; // bits per sample
br=2*bandwidth*bits; // formula to calculate bit rate
bit_rate=br*10^-3;  //multiply with conversion factor
printf("The bit rate of the channel is %d kbps.", bit_rate); //display result
```

Scilab code Exa 3.20 HDTV bitrate

```
clear;
clc;
disp("————————————Example 3.20————————————")
rate=30 ; // screen is refreshed 30 times per second
 // screen resolution = 1920x1080
vl = 1920;
hl = 1080;
bits_per_pixel=24; // bits to represent one color pixel
br=vl*hl*rate*bits_per_pixel; // formula to calculate bit rate
bit_rate=br*10^-9;  // multiply with conversion factor
printf("The bit rate of HDTV is %2.1f Gbps.", bit_rate); // display the result
```

Scilab code Exa 3.21 LAN

```
clear;
clc;
disp("---------------Example 3.21---------------")
printf("An example of a dedicated channel where the
    entire bandwidth of the medium is used as one
    single channel is a LAN.\nAlmost every wired LAN
    today uses a dedicated channel for two stations
    communicating with each other.\nIn a bus topology
     LAN with multipoint connections, only two
    stations can communicate with each other at each
    moment\nin time (timesharing); the other stations
     need to refrain from sending data. In a star
    topology LAN,\nthe entire channel between each
    station and the hub is used for communication
    between these two entities.");  // display the
    example
```

Scilab code Exa 3.22 Base band transmission

```
clear;
clc;
disp("---------------Example 3.22---------------")
bit_rate=10^6; // 1 Mbps
//a) rough approximation
mb=bit_rate/2; // formula to caluculate bandwidth
min_bandwidth=mb*10^-3; //multiply with conversion
    factor
printf("\n a) The minimum bandwidth is %d kHz.",
    min_bandwidth); // display result
printf("\n     A low-pass channel with frequencies
    between 0 and %d kHz is required.\n\n",
    min_bandwidth);
//b) using the first and the third harmonics
bw = 3*mb;  // formula to caluculate bandwidth
bandwidth=bw * 10^-6; //multiply with conversion
```

```
    factor
printf(" b) The required bandwidth is %2.1f MHz.",
    bandwidth); // display result
printf("\n  Hence a better result can be achieved by
     using the first and the third harmonics.\n\n");
//c) using the first , third , and fifth harmonics
bw = 5*mb;  // formula to caluculate bandwidth
bandwidth = bw*10^-6; //multiply with conversion
    factor
printf(" c) The required bandwidth is %2.1f MHz.",
    bandwidth);  // display result
printf("\n  Hence a still better result can be
    achieved by using the first , third and the fifth
    harmonics.\n\n");
```

Scilab code Exa 3.23 Maximum bitrate calculation

```
clear;
clc;
disp("—————————————Example 3.23—————————————")
bandwidth = 100; // 100 kHz
max_bitrate= 2* bandwidth ; // max bitrate is
    achieved by using the 1st harmonic
printf("The maximum bit rate is %d kbps.",
    max_bitrate); // display the result
```

Scilab code Exa 3.24 Broadband transmission example

```
clear;
clc;
disp("—————————————Example 3.24—————————————")
printf("An example of broadband transmission using
    modulation is the sending of computer data
```

```
    through a telephone subscriber line,\nthe line
    connecting a resident to the central telephone
    office. These lines, installed many years ago,are
     designed to\ncarry voice (analog signal) with a
    limited bandwidth(frequencies between 0 and 4 kHz
    ).\nAlthough this channel can be used as a low-
    pass channel,it is normally considered a bandpass
     channel. One reason is that\nthe bandwidth is so
     narrow(4 kHz) that if we treat the channel as
    low-pass and use it for baseband transmission,
    the maximum\nbit rate can be only 8 kbps. The
    solution is to consider the channel a bandpass
    channel, convert the digital signal from the\
    ncomputer to an analog signal, and send the
    analog signal.\nWe can install two converters to
    change the digital signal to analog and vice
    versa at the receiving end.\nThe converter,in
    this case, is called a modem (modulator/
    demodulator)."); // display the example
```

Scilab code Exa 3.25 Digital cellular telephone

```
clear;
clc;
disp("-------------------Example 3.25-------------------")
printf("Another example of broadband transmission is
     the digital cellular telephone. For better
    reception, digital cellular phones convert the
    analog\nvoice signal to a digital signal.Although
     the bandwidth allocated to a company providing
    digital cellular phone service is very wide,\nwe
    still cannot send the digital signal without
    conversion. The reason is that we only have a
    bandpass channel available between caller and
    callee.\nFor example, if the available bandwidth
```

```
    is W and we allow 1000 couples to talk
    simultaneously, this means the available channel
    is W/1000,\njust part of the entire bandwidth. We
     need to convert the digitized voice to a
    composite analog signal before sending.\nThe
    digital cellular phones convert the analog audio
    signal to digital and then convert it again to
    analog for transmission over\na bandpass channel.
    "); //display the example
```

Scilab code Exa 3.26 Attenuation calculation

```
clear;
clc;
disp("--------------Example 3.26--------------")
ratio=0.5; // power of signal after attenuation/
    initial power of signal = p2/p1
at=10*log10(ratio); // formula to calculate
    attenuation or loss of power
printf("The attenuation is %d dB.\nA loss of %d dB (
    %d dB) is equivalent to losing one-half the power
    .",at,-at,at); // display result
```

Scilab code Exa 3.27 Amplification calculation

```
clear;
clc;
disp("--------------Example 3.27--------------")
ratio=10; // power of signal after amplification/
    initial power of signal = p2/p1
amp=10*log10(ratio); // formula to calculate
    amplification or gain of power
```

```
printf("The amplification is %d dB.",amp); //
   display result
```

Scilab code Exa 3.28 Resultant decibel calculation

```
clear;
clc;
disp("—————————————Example 3.28—————————————")
dB1=-3; // signal is attenuated
dB2=7;  // signal is amplified
dB3=-3; // signal is attenuated
dB=dB1+dB2+dB3; // add to get final dB
printf("The final decibel value is +%d dB . Hence
   the signal has gained in power.",dB); //display
   result
```

Scilab code Exa 3.29 Pm from dBm

```
clear;
clc;
disp("—————————————Example 3.29—————————————")
dBm=-30; // dBm = 10*log10 (Pm)
Pm = 10^(dBm/10); // power in milliwatts
printf("The power of the signal in milliwatts is %2
   .1E mW.",Pm); // display result
```

Scilab code Exa 3.30 dB per km

```
clear;
clc;
```

```
disp("---------------Example 3.30---------------")
dBpkm= -0.3 ; // dB/km
p1=2; // 2 mW - initial power
distance = 5; // 5km
dB= dBpkm*distance; // loss in the cable in decibel
ratio=10^(dB/10); // dB=10*log10(ratio)
p2 = p1*ratio; // ratio = p2/p1
printf("The power of the signal at 5 km is %2.1f mW.",p2);// display result
```

Scilab code Exa 3.31 SNR and SNRdB

```
clear;
clc;
disp("---------------Example 3.31---------------")
p_signal=10*10^-3; // 10 mW
p_noise=10^-6; // 1 microW
SNR = p_signal/p_noise; // SNR = signal power/noise power
SNRdB=10*log10(SNR); // formula to calculate SNR in dB
printf("SNR = %d \n\nSNRdB = %d ",SNR,SNRdB); // display result
```

Scilab code Exa 3.32 Noiseless channel SNR and SNRdB

```
clear;
clc;
disp("---------------Example 3.32---------------")
//for noiseless channels , noise power =0
//SNR = signal power/ 0 = infinity
//SNRdB = 10*log10(SNR)=10*log10(infinity)=infinity
```

```
printf("The values of SNR and SNRdB for a noiseless
    channel are both infinity . We can never achieve
    this ratio in real life; it is an ideal situation
    .");
```

Scilab code Exa 3.33 Nyquist and baseband transmission

```
clear;
clc;
disp("---------------Example 3.33----------------")
printf("The Nyquist theorem bit rate and the
    intuitive bit rate match when there are only two
    levels.\nIn baseband transmission , the bit rate
    is 2 times the bandwidth if only the first
    harmonic is used in the worst case.\nHowever, the
     Nyquist formula is more general than what we
    derived intuitively; it can be applied to
    baseband transmission and modulation.\nAlso, it
    can be applied when we have two or more levels of
     signals.") //display the answer
```

Scilab code Exa 3.34 Noiseless channel bitrate

```
clear;
clc;
disp("---------------Example 3.34----------------")
L=2; //number of levels
bandwidth=3000; // Hz
max_bitrate=2*bandwidth*log2(L); // formula to
    calculate maximum bit rate
printf("The maximum bit rate of the noiseless
    channel is %d bps.",max_bitrate); //display
    result
```

Scilab code Exa 3.35 Noiseless channel bitrate 2

```
clear;
clc;
disp("————————————Example 3.35————————————")
L=4; //number of levels
bandwidth=3000; // Hz
max_bitrate=2*bandwidth*log2(L); // formula to
    calculate maximum bit rate
printf("The maximum bit rate of the noiseless
    channel is %d bps.",max_bitrate); // display
    result
```

Scilab code Exa 3.36 Noiseless channel signal levels

```
clear;
clc;
disp("————————————Example 3.36————————————")
bitrate=265*10^3; // 256 kbps
bandwidth= 20 * 10^3; // 20 kHz
L= 2^(bitrate/(2*bandwidth)); // bit rate = 2*
    bandwidth*log2(L)
printf("\nThe number of levels is %3.1f .",L);
c=log2(L);
if(~(modulo(c,10)==0)) // check correctness of the
    answer . If not practical , change it .
    printf("\nSince this result is not a power of 2,
         we need to either increase the number of
        levels or reduce the bit rate.");
    r=floor(bitrate/(2*bandwidth));
    L=2^r;
```

```
    n_bitrate=2*bandwidth*log2(L);
    printf("\n\nHence the number of signal levels is %d and bit rate is %d kbps.",L,n_bitrate*10^-3); // display final result
end
```

Scilab code Exa 3.37 C for extremely noisy channel

```
clear;
clc;
disp("----------------Example 3.37----------------")
SNR = 0; //an extremely noisy channel in which the value of the signal-to-noise ratio is almost zero.
m=log2(1+SNR);
//display result
printf("The value of log2(1+SNR) = %d \nHence C = B*log2(1+SNR)= B*0 = %d",m,m);
printf("\nThis means that the capacity of this channel is zero regardless of the bandwidth. In other words, any data cant be recieved through this channel.")
```

Scilab code Exa 3.38 C for noisy channel

```
clear;
clc;
disp("----------------Example 3.38----------------")
SNR = 3162;
B= 3000; // bandwidth in Hz
t=log2(1+SNR);
le=floor(t*100)/100; // rounding the log value
C= B*le;// formula to calculate capacity
```

```
c=C*10^-3; // multiply with conversion factor
printf("\nThe capacity of the channel is %d bps.",C); //display result
printf("\nHence the highest bit rate for a telephone line is %5.3f kbps.",c);
```

Scilab code Exa 3.39 C using SNRdB

```
clear;
clc;
disp("——————————————Example 3.39———————————————")
SNRdB=36;
B=2*10^6; // bandwidth = 2 MHz
SNR= 10^(SNRdB/10); // SNRdB= 10*log10(SNR)
C= B*log2(1+SNR); //formula to calculate capacity
c=C*10^-6; //multiply the conversion factor
printf("\nThe theoretical channel capacity is %2.0f Mbps.",c); //display result
```

Scilab code Exa 3.40 C formula simplification

```
clear;
clc;
disp("——————————————Example 3.40———————————————")
SNRdB=36;
B=2; // bandwidth = 2 MHz
C=B*(SNRdB/3); //when the SNR is very high, we can assume that SNR + 1 is almost the same as SNR
printf("The theoretical channel capacity is %d Mbps.",C); // display result
```

Scilab code Exa 3.41 Shannon and Nyquist formula

```
clear;
clc;
disp("———————————————Example 3.41————————————————")
SNR=63;
B=10^6; // bandwidth = 1 MHz
b=4*10^6; // chosen bit rate =4 Mbps
C= B*log2(1+SNR);  // Shannon's capacity formula
c=C*10^-6; //multiply with conversion factor
L=2^(b/(2*B)); // bit rate = 2*bandwidth*log2(L) ; L
   = number of signal levels
//display result
printf("\nThe Shannon formula gives us %d Mbps, the
   upper limit. For better performance choose
   something lower, 4 Mbps, for example.",c);
printf("\n\nThe Nyquist formula gives the number of
   signal levels as %d .",L);
```

Scilab code Exa 3.42 Modulation using modem

```
clear;
clc;
disp("———————————————Example 3.42————————————————")
printf("The bandwidth of a subscriber line is 4 kHz
   for voice or data. The bandwidth of this line for
    data transmission\ncan be up to 56,000 bps using
    a sophisticated modem to change the digital
   signal to analog.") // display the examples
```

Scilab code Exa 3.43 Increasing bandwidth of line

```
clear;
```

```
clc;
disp("———————————Example 3.43———————————")
printf("If the telephone company improves the quality of the line and increases the bandwidth to 8 kHz,\nwe can send 112,000 bps by using the same technology as mentioned in Example 3.42.")// display the example
```

Scilab code Exa 3.44 Throughput of network

```
clear;
clc;
disp("———————————Example 3.44———————————")
bandwidth = 10 ; // 10 MHz
fpm=12000; //frames per second
bits=10000; //bits carried by each frame
throughput=((fpm*bits)/60)*10^-6; //formula for throughput
frac=bandwidth/throughput;  // ratio of bandwidth to throughput
printf("The throughput is %d Mbps.\n",throughput);// display result
printf("The throughput is almost 1/%dth of the bandwidth in this case.",frac);
```

Scilab code Exa 3.45 Propagation time calculation

```
clear;
clc;
disp("———————————Example 3.45———————————")
distance=12000*10^3; // 12000km
propagation_speed=2.4*10^8; // 2.4*10^8 m/s
```

```
propagation_time=distance/propagation_speed; // formula for propagation time
propagation_time=propagation_time*10^3; //multiply with conversion factor
// display result
printf("\nThe propagation time is %d ms.\n",propagation_time);
printf("\nThe example shows that a bit can go over the Atlantic Ocean in only %d ms if there is a direct cable between the source and the destination.",propagation_time);
```

Scilab code Exa 3.46 Propagation and transmission time

```
clear;
clc;
disp("---------------Example 3.46---------------")
message_size=2.5*10^3; //2.5 kbyte
bandwidth=10^9; // 1Gbps
propagation_speed=2.4*10^8; //2.4*10^8 m/s
distance=12000*10^3; // 12,000 km
propagation_time=distance/propagation_speed; // propagation time formula
transmission_time=(message_size*8)/bandwidth; // transmission time formula
// display result
printf("\nThe propagation time is %d ms.\n",propagation_time*10^3);
printf("The transmission time is %4.3f ms.\n",transmission_time*10^3);
printf("\nNote that in this case, because the message is short and the bandwidth is high, the dominant factor is the\npropagation time, not the transmission time. The transmission time can be ignored.")
```

Scilab code Exa 3.47 Propagation and transmission time 2

```
clear;
clc;
disp("--------------Example 3.47--------------")
message_size=5*10^6; //5 M byte
bandwidth=10^6; // 1Mbps
propagation_speed=2.4*10^8; //2.4*10^8 m/s
distance=12000*10^3; // 12,000 km
propagation_time=distance/propagation_speed; //
    propagation time formula
transmission_time=(message_size*8)/bandwidth; //
    transmission time formula
// display result
printf("\nThe propagation time is %d ms.\n",
    propagation_time*10^3);
printf("The transmission time is %d s.\n",
    transmission_time);
printf("\nNote that in this case, because the
    message is very long and the bandwidth is not
    very high, the dominant factor is\nthe
    transmission time, not the propagation time. The
    propagation time can be ignored.")
```

Scilab code Exa 3.48 Bandwidth delay product

```
clear;
clc;
disp("--------------Example 3.48--------------")
printf("Consider the link between two points to be a
    pipe. The cross section of the pipe represents
```

```
the bandwidth,\nand the length of the pipe
represents the delay. The volume of the pipe
defines the bandwidth-delay product.")//display
the example
```

Chapter 4

Digital Transmission

Scilab code Exa 4.1 average baud rate

```
clear;
clc;
disp("————————————————Example 4.1————————————————")
r=1; // 1 data element/ 1 signal element
N=100*10^3; // bitrate=100 kbps
c=(0+1)/2; //case factor
S=c*N*(1/r); // formula for baud rate
printf("The average baud rate is %d kbaud.",S*10^-3);//display result
```

Scilab code Exa 4.2 Nyquist formula equivalence

```
clear;
clc;
disp("————————————————Example 4.2————————————————")
printf("\nA signal with L levels actually can carry log2 L bits per level. If each level corresponds to one signal element\nand we assume the average
```

```
case (c = 1/2), then the two formulas agree with each other.") //display the example
disp("Nmax = (1/c) x B x r = 2 x B x log2L");
```

Scilab code Exa 4.3 Extra bits calculation

```
clear;
clc;
disp("————————————Example 4.3————————————")
data_rate1=10^3; //1 kbps
data_rate2=10^6; //1 Mbps
frac= 0.1*10^-2; // receiver clock is 0.1 percent faster than the sender clock
function []=extrabits(data_rate,frac) // function to calculate extra bps and recieved bps
    extra = data_rate*frac; // formula to calculate extra bps
    recieved = data_rate+extra; //formula to calculate reieved bps
    printf("Data rate = %d bps\nBits sent = %d\nBits recieved = %d\nExtra bps = %d ",recieved,data_rate,recieved,extra); //display the result
endfunction
//data rate = 1 kbps
printf("\nAt 1 kbps, \n");
extrabits(data_rate1,frac); //calling the function
//data rate = 1 Mbps
printf("\n\nAt 1 Mbps, \n")
extrabits(data_rate2,frac); //calling the function
```

Scilab code Exa 4.4 avg baud and min bandwidth

```
clear;
clc;
disp("---------------Example 4.4---------------")
N=10*10^6; // bit rate = 10 Mbps
S=N/2; // formula for average signal rate
Bmin=S; // minimum bandwidth is equal to average baud
// display result
printf("\nThe average signal rate is %d kbaud.",S*10^-3);
printf("\n\nThe minimum bandwidth is %d kHz.",Bmin*10^-3);
```

Scilab code Exa 4.5 Block coding minimum bandwidth

```
clear;
clc;
disp("---------------Example 4.5---------------")
data_rate=1; // 1 Mbps
frac= 0.25 // 4B/5B coding adds 25% to the baud rate
add=data_rate*frac;
N = (data_rate+add)*10^6; // Hz
NRZI_B= N/2; // minimum bandwidth using NRZ-I
Manchester_B = data_rate; // minimum bandwidth using Manchester scheme
// display result
printf("\n 4B/5B block coding increases the bit rate to %3.2f Mbps\n The minimum bandwidth using NRZ-I scheme is %d kHz.\n\n The minimum bandwidth using Manchester scheme is %d MHz.",N*10^-6,NRZI_B*10^-3,Manchester_B);
printf("\n\nThe NRZ-I scheme needs a lower bandwidth, but has a DC component problem; the Manchester scheme needs a higher bandwidth,\nbut does not have a DC component problem.")
```

Scilab code Exa 4.6 sampling and recovery

```
clear;
clc;
disp("---------------Example 4.6---------------")
// example explanation
printf("A simple sine wave is sampled at three sampling rates:\nfs = 4f (2 times the Nyquist rate )\nfs = 2f (Nyquist rate\nfs = f (one-half the Nyquist rate)\n\n");
printf("It can be seen that sampling at the Nyquist rate can create a good approximation of the original sine wave (part a).\nOversampling in part b can also create the same approximation, but it is redundant and unnecessary.\nSampling below the Nyquist rate (part c) does not produce a signal that looks like the original sine wave.")
// display the figure
clf();
xname("---------------Example 4.6----------------");
subplot(325)
a1=gca();
a1.x_label.text="c. Undersampling fs=f";
a1.x_location="middle";
x=[0:.1:5*%pi]; // x-range
plot(x,sin(x),nax=[0,0,0,0]);
xfarc(1.5,1,.1,.1,0,360*64);
xfarc(6.2,.1,.1,.1,0,360*64);
xfarc(11,-.9,.1,.1,0,360*64);
xfarc(15.7,0,.1,.1,0,360*64);
subplot(321)
a1=gca();
a1.x_label.text="a.Nyquist rate sampling fs=2f";
```

```
a1.x_location="middle";
x=[0:.1:5*%pi]; // x-range
plot(x,sin(x),nax=[0,0,0,0]);
xfarc(1.5,1,.1,.1,0,360*64);
xfarc(4.7,-.9,.1,.1,0,360*64);
xfarc(7.9,1,.1,.1,0,360*64);
xfarc(11,-.9,.1,.1,0,360*64);
xfarc(14,1,.1,.1,0,360*64);
subplot(323)
a1=gca();
a1.x_label.text="b.Oversampling  fs=4f";
a1.x_location="middle";
x=[0:.1:5*%pi]; // x-range
plot(x,sin(x),nax=[0,0,0,0]);
xfarc(1.5,1,.1,.1,0,360*64);
xfarc(4.7,-.9,.1,.1,0,360*64);
xfarc(7.9,1,.1,.1,0,360*64);
xfarc(11,-.9,.1,.1,0,360*64);
xfarc(14,1,.1,.1,0,360*64);
xfarc(0,0.1,.1,.1,0,360*64);
xfarc(3.1,.1,.1,.1,0,360*64);
xfarc(6.2,.1,.1,.1,0,360*64);
xfarc(9.4,.1,.1,.1,0,360*64);
xfarc(12.5,.1,.1,.1,0,360*64);
xfarc(15.7,0,.1,.1,0,360*64);
subplot(322)
a1=gca();
xarrows([0 1],[.5 .5],.7);
xarrows([0 0],[0 1],.7);
xset("line style",2);
for i=0:2
    xpoly([0+(i/2.5) .1+(i/2.5)],[.5 1]);
    xpoly([.1+(i/2.5) .2+(i/2.5)],[1 .5]);
end
for i=0:1
    xpoly([.2+(i/2.5) .3+(i/2.5)],[.5 0]);
    xpoly([.3+(i/2.5) .4+(i/2.5)],[0 .5]);
end
```

```
xfarc(.09,1,.02,.02,0,360*64);
xfarc(.29,0,.02,.02,0,360*64);
xfarc(.49,1,.02,.02,0,360*64);
xfarc(.69,0,.02,.02,0,360*64);
xfarc(.89,1,.02,.02,0,360*64);
subplot(324)
a1=gca();
xarrows([0 1],[.5 .5],.7);
xarrows([0 0],[0 1],.7);
xset("line style",2);
for i=0:2
    xpoly([0+(i/2.5) .1+(i/2.5)],[.5 1]);
    xpoly([.1+(i/2.5) .2+(i/2.5)],[1 .5]);
end
for i=0:1
    xpoly([.2+(i/2.5) .3+(i/2.5)],[.5 0]);
    xpoly([.3+(i/2.5) .4+(i/2.5)],[0 .5]);
end
xfarc(.09,1,.02,.02,0,360*64);
xfarc(.29,0,.02,.02,0,360*64);
xfarc(.49,1,.02,.02,0,360*64);
xfarc(.69,0,.02,.02,0,360*64);
xfarc(.89,1,.02,.02,0,360*64);
xfarc(0,.52,.02,.02,0,64*360);
xfarc(.2,.52,.02,.02,0,64*360);
xfarc(.4,.52,.02,.02,0,64*360);
xfarc(.6,.52,.02,.02,0,64*360);
xfarc(.8,.52,.02,.02,0,64*360);
xfarc(1,.52,.02,.02,0,64*360);
subplot(326)
a1=gca();
//a1.x_location="middle";
//x=[0:.1:3*%pi]; // x-range
//plot(x,cos(.5*x),nax=[0,0,0,0]);
xfarc(.1,.75,.02,.02,0,360*64);
xfarc(.318,.5,.02,.02,0,360*64);
xfarc(.67,0,.02,.02,0,360*64);
xfarc(.97,.5,.02,.02,0,360*64);
```

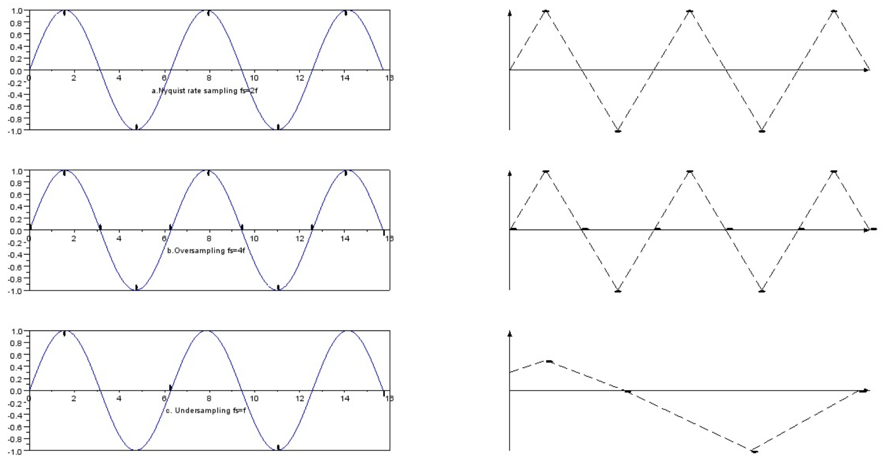

Figure 4.1: sampling and recovery

```
xarrows([0 1],[.5 .5],.7);
xarrows([0 0],[0 1],.7);
xset("line style",2);
xpoly([0 .1],[.65 .75]);
xpoly([.1 .318],[.75 .5]);
xpoly([.318 .67],[.5 0]);
xpoly([.67 .97],[0 .5]);
```

Scilab code Exa 4.7 Sampling of clock

```
clear;
clc;
disp("--------------Example 4.7--------------")
T=60; // The second hand of a clock has a period of
    60 s.
printf("The second hand of a clock has a period of
    %d s.\n",T);
// a) Sampling at Nyquist rate
```

```
Ts=0.5*T ; // or fs = 2f
sp=[]; // sampling points
time=[12 6];
for i=0:5  // assign sampling points
    if(modulo(i,2)==0)
        sp(i+1)=time(1);
    else
        sp(i+1)=time(2);
    end
end
//display the result
printf("\na)According to the Nyquist theorem, the
   second hand is sampled every %d s .\nThe sample
   points, in order, are %d, %d, %d, %d, %d, and %d
   .\nThe receiver of the samples cannot tell if the
    clock is moving forward or backward.",Ts,sp(1),
   sp(2),sp(3),sp(4),sp(5),sp(6));
// b) Sampling at double the Nyquist rate
Ts=0.25*T; // or fs = 4f
sp=[]; //sampling points
time=[12 3 6 9];
for i=0:4  // assign sampling points
    if (i==4) then
        sp(i+1)=time(1);
    else
        sp(i+1)=time(i+1);
    end
end
//display the result
printf("\n\nb)The second hand is sampled at double
   the Nyquist rate or every %d s.\nThe sample
   points, in order, are %d, %d, %d, %d, and %d. The
    clock is moving forward.",Ts,sp(1),sp(2),sp(3),
   sp(4),sp(5));
// b) Sampling at lesser than Nyquist rate
Ts=0.75*T; // or fs = 4/3f
sp=[]; //sampling points
time=[12 9 6 3];
```

```
for i=0:4  // assign sampling points
    if (i==4) then
        sp(i+1)=time(1);
    else
        sp(i+1)=time(i+1);
    end
end
//display the result
printf("\n\nc)The second hand is sampled below the
   Nyquist rate or every %d s.\nThe sample points,
   in order, are %d, %d, %d, %d, and %d. Although
   the clock is moving forward, the receiver thinks
   that the clock is moving backward.",Ts,sp(1),sp
   (2),sp(3),sp(4),sp(5));
```

Scilab code Exa 4.8 wheel rotation undersampling

```
clear;
clc;
disp("—————————————Example 4.8—————————————")
printf("\nThe seemingly backward rotation of the
   wheels of a forward moving car in a movie:-  This
    can be explained by undersampling.\nA movie is
   filmed at 24 frames per second. If a wheel is
   rotating more than 12 times per second,the
   undersampling\ncreates the impression of a
   backward rotation.")//display the example
```

Scilab code Exa 4.9 telephone sampling rate

```
clear;
clc;
disp("—————————————Example 4.9—————————————")
```

```
f=4000; // max frequency
sr=2*f; // sampling rate = 2*max frequency
printf("\nTelephone companies digitize voice by assuming a maximum frequency of %d Hz.\nThe sampling rate therefore is %d samples per second.",f,sr); // display result
```

Scilab code Exa 4.10 minimum sampling rate

```
clear;
clc;
disp("————————————————Example 4.10————————————————")
bandwidth = 200*10^3; // 200 kHz
//The bandwidth of a low-pass signal is between 0 and f ,where f is the maximum frequency in the signal.
// Therefore, highest frequency =200 kHz
f = bandwidth; // max frequency
sr = 2*f ; // sampling rate = 2*max frequency
printf("The sampling rate is %d samples per second.",sr);// display result
```

Scilab code Exa 4.11 bandpass sampling rate

```
clear;
clc;
disp("————————————————Example 4.11————————————————")
bandwidth = 200; // 200 kHz
printf("\nThe minimum sampling rate cannot be determined in this case because we do not know where the bandwidth starts\nor ends or in order words we do not know the maximum frequency in the signal."); // display example
```

Scilab code Exa 4.12 SNR db formula

```
clear;
clc;
disp("——————————————Example 4.12——————————————")
L=8; // number of levels
nb=log2(L); // number of bits per sample
SNRdB=6.02*(nb)+1.76; // formula
printf("The SNRdB is %4.2f dB.\nIncreasing number of
    levels increases the SNR.",SNRdB); // display
   result
```

Scilab code Exa 4.13 telephone bits per sample

```
clear;
clc;
disp("——————————————Example 4.13——————————————")
SNRdB=40;
nb=(SNRdB-1.76)/6.02; // SNRdB = 6.02(nb)+1.76
printf("\nnb = %4.2f",nb); // display result
printf("\nTherefore telephone companies usually
   assign %d or %d bits per sample.",ceil(nb),ceil(
   nb)+1); // round off to nearest integer as number
    of bits should be a whole number
```

Scilab code Exa 4.14 human voice digitization

```
clear;
clc;
```

```
disp("———————————————Example 4.14————————————————")
bits= 8; // bits per sample
fl=0; // The human voice normally contains
    frequencies from 0 to 4000 Hz.
fh=4000; // Hz
sampling_rate = 2*fh; // twice the highest frequency
bit_rate=sampling_rate*bits;  // formula
printf("The sampling rate is %d samples/s and the
    bit rate is %d kbps.",sampling_rate,bit_rate
    *10^-3); // display the result with appropriate
    units
```

Scilab code Exa 4.15 digital minimum bandwidth

```
clear;
clc;
disp("———————————————Example 4.15————————————————")
analog_min_bandwidth = 4; //4 kHz
bits = 8; // bits per sample
digital_min_bandwidth = analog_min_bandwidth*bits;
    // formula
printf("The minimum bandwidth for the digital signal
     is %d kHz.",digital_min_bandwidth); // display
    result
```

Chapter 5

Analog Transmission

Scilab code Exa 5.1 analog signal bit rate

```
clear;
clc;
disp("———————————————Example 5.1———————————————")
S=1000; // baud rate
r=4; // bits/signal element
N=S*r; // bit rate formula
printf("The bit rate is %d bps.",N); // display
   result
```

Scilab code Exa 5.2 data and signal elements

```
clear;
clc;
disp("———————————————Example 5.2———————————————")
S=1000; // baud rate
N= 8000; // bit rate in bps
r= N/S; // data elements/signal element
L= 2^r ; // number of signal elements
```

```
printf("The number of data elements per signal
    element is %d bits/baud and the number of signal
    elements is %d .",r,L); // display result
```

Scilab code Exa 5.3 ASK fc and bitrate

```
clear;
clc;
disp("---------------Example 5.3---------------")
d=1;
r=1;
B=100; // 100 kHz
fl=200; // lower frequency=200 kHz
fh=300; // highest frequency =300 kHz
middle_bandwidth = (fl+fh)/2; // kHz
Fc=middle_bandwidth; // carrier frequency
N=(B*r)/2; // B= (1+d)*S = 2*N*(1/r) , N - bit rate
printf("\nThe carrier frequency is %d kHz.\nThe bit
    rate is %d kbps.",Fc,N); //display result
```

Scilab code Exa 5.4 Full duplex ASK

```
clear;
clc;
disp("---------------Example 5.4---------------")
B=100;// total Bandwidth in kHz
B1=B/2; // bandwith for one direction
B2=B/2; // bandwidth for other direction
fl=200; // lower frequency=200 kHz
fh=300; // highest frequency =300 kHz
middle_bandwidth1= (fl+(fl+B1))/2; // kHz
Fc1=middle_bandwidth1; // carrier frequency for one
    direction
```

```
middle_bandwidth2= ((fl+B1)+fh)/2; // kHz
Fc2=middle_bandwidth2; // carrier frequency for
    other direction
N1=B1/2; // data rate in one direction
N2=B2/2; //data rate in other direction
// display result
printf("\nThe carrier frequency for one direction is
     %d kHz , bandwidth is %d kHz and data rate is %d
     kbps.\n",Fc1,B1,N1);
printf("\nThe carrier frequency for other direction
    is %d kHz , bandwidth is %d kHz and data rate is
    %d kbps.\n",Fc2,B2,N2);
// display the figure
clf();
xname("—————————Example 5.4—————————");
x=linspace(.45,.55,2);
for i=1:2
    xpoly([x(i) x(i)],[.18 .22]);
end
x=linspace(.5,.6,2);
for i=1:2
    xpoly([x(i) x(i)],[.2 .3]);
end
x=linspace(.4,.5,2);
for i=1:2
    xpoly([x(i) x(i)],[.2 .3]);
end
for i=0:1
    xarc(.47+(i/10),.31,.03,.03,0,90*64);
    xarc(.4+(i/10),.31,.03,.03,90*64,91*64);
end
xpoly([.35 .65],[.2 .2])
for i=0:1
    xpoly([.41+(i/10) .49+(i/10)],[.31 .31]);

end
xpoly([.4 .49],[.32 .32]);
xpoly([.5 .6],[.32 .32]);
```

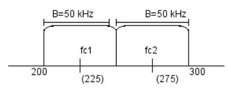

Figure 5.1: Full duplex ASK

```
xpoly([.4 .4],[.31 .33]);
xpoly([.49 .49],[.31 .33]);
xpoly([.5 .5],[.31 .33]);
xpoly([.6 .6],[.31 .33]);
xset("font size",2)
xstring(.38,.17,"200

   300");
xstring(.45,.15,"(225)                         (275)");
xstring(.449,.23,"fc1                         fc2");
xstring(.41,.33,"B=50 kHz                         B=50
   kHz");
```

Scilab code Exa 5.5 FSK fc and bitrate

```
clear;
clc;
disp("———————————Example 5.5———————————")
```

```
B=100 ; //bandwidth = 100 kHz
two_df=50; // 2df = 50 kHz
fl=200; // lower frequency in kHz
fh=300; // higher frequency in kHz
mid_bandwidth = (fl+fh)/2; // mid frequency of
    bandwidth in kHz
Fc=mid_bandwidth;
d=1;
S=(B-two_df)/(1+d); // B= (1+d)*s + 2df
N=S; // bit rate
printf("\nThe carrier frequency is %d kHz , the
    signal rate is %d kbaud and the bit rate %d kbps.
    ",Fc,S,N); // display result
```

Scilab code Exa 5.6 bandwidth of MFSK

```
clear;
clc;
disp("--------------Example 5.6---------------")
n=3; // number of bits per sample
N=3; // bit rate = 3 MHz
Fc=10; // carrier f requency = 10 MHz
L=2^n; // number of levels
S=N/n; // baud rate
two_df=S; // 2df = 1 MHz
B=L*S; // bandwidth
printf("\nThe number of levels is %d , the signal
    rate is %d Mbaud and the bandwidth is %d MHz.",L,
    S,B);// display result
// display the figure
clf();
xname("---------------Example 5.6----------------");
xarrows([0 1],[.2 .2],.5);
xset("font size",5);
xstring(1,.1,"Frequency");
```

```
xpoly([.1 .9],[.55 .55]);
xpoly([.1 .1],[.57 .53]);
xpoly([.9 .9],[.57 .53]);
xstring(.4,.6,"Bandwidth = 8 MHz");
x=linspace(.15,.85,8);
for i=1:8
    xpoly([x(i) x(i)],[.18 .22]);
end
x=linspace(.1,.9,9);
for i=1:9
    xpoly([x(i) x(i)],[.2 .3]);
end
for i=0:7
    xarc(.17+(i/10),.31,.03,.03,0,90*64);
    xarc(.1+(i/10),.31,.03,.03,90*64,91*64);
end

for i=0:7
    xpoly([.11+(i/10) .19+(i/10)],[.31 .31]);
end
xset("thickness",2);
xpoly([.5 .5],[.2 .35])
xset("font size",3);
x=linspace(.15,.85,8);
for i=1:8
    s=6.5+i-1;
    xstring(x(i),.14,"f"+string(i));
    xstring(x(i),.1,string(s));
    xstring(x(i),.06,"MHz");
end
xstring(.5,.14,"fc");
xstring(.5,.1,"10");
xstring(.5,.06,"MHz");
```

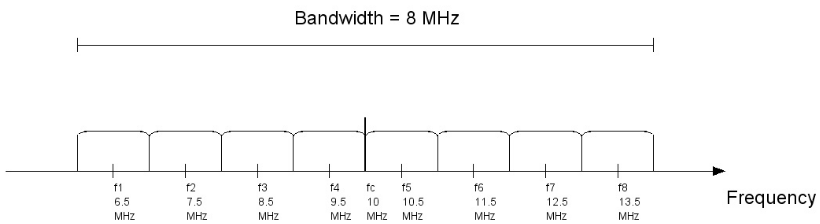

Figure 5.2: bandwidth of MFSK

Scilab code Exa 5.7 bandwidth of QPSK

```
clear;
clc;
disp("---------------Example 5.7---------------")
d=0;
r=2; // For QPSK, 2 bits is carried by one signal
    element
N=12; // bit rate = 12 Mbps
S=N*(1/r); // formula for signal rate
B=S; // bandwidth , as d=0
printf("The bandwidth is %d MHz.",B); // display
    result
```

Scilab code Exa 5.8 Three constellations diagrams

```
clear;
clc;
// drawing the constellation diagrams
clf();
```

```
xname("-----------------Example 5.8-----------------");
title('Constellation diagrams','fontsize',5);
xrect(-.1,.8,.35,.45);
xrect(.3,.8,.35,.45);
xrect(.7,.8,.35,.45);
xpoly([-.1 .25],[.575 .575]);
xpoly([.3 .65],[.575 .575]);
xpoly([.7 1.05],[.575 .575]);
xpoly([.075 .075],[.8 .35]);
xpoly([.475 .475],[.8 .35]);
xpoly([.875 .875],[.8 .35]);
xset("font size",4);
xstring(-.1,.3,"a. ASK(OOK)");
xstring(.3,.3,"b. BPSK");
xstring(.7,.3,"c. QPSK");
xfarc( .059,.585 , 0.03, 0.03, 0, 64 * 360 ) ;
xfarc( .15,.585 , 0.03, 0.03, 0, 64 * 360 ) ;
xfarc( .37,.585 , 0.03, 0.03, 0, 64 * 360 ) ;
xfarc( .55,.585 , 0.03, 0.03, 0, 64 * 360 ) ;
xfarc( .77,.7 , 0.03, 0.03, 0, 64 * 360 ) ;
xfarc( .95,.7 , 0.03, 0.03, 0, 64 * 360 ) ;
xfarc( .77,.46 , 0.03, 0.03, 0, 64 * 360 ) ;
xfarc( .95,.46 , 0.03, 0.03, 0, 64 * 360 ) ;
xset("font size",3);
xstring(.055,.5,"0");
xstring(.15,.5,"1");
xstring(.37,.5,"0");
xstring(.55,.5,"1");
xstring(.745,.7,"01");
xstring(.975,.7,"11");
xstring(.74,.44,"00");
xstring(.98,.44,"10");
xset("line style",2);
xarc(.75,.7,.275,.275,0,64*360);
// display the explanation of the diagrams
disp("-----------------Example 5.8-----------------")
printf("\na. For ASK,only an in-phase carrier is
    used. Therefore, the two points should be on the
```

```
X axis.\nBinary 0 has an amplitude of 0 V; binary
 1 has an amplitude of 1V (for example). The
points are located at the origin and at 1 unit.\n
\n");
printf("b. BPSK also uses only an in-phase carrier.
However, polar NRZ signal is used for modulation
.\nIt creates two types of signal elements, one
with amplitude 1 and the other with amplitude -1.
 This can be stated in other words:\nBPSK creates
 two different signal elements, one with
amplitude 1 V and in phase and the other with
amplitude 1 V and 180   out of phase.\n\n");
printf("c. QPSK uses two carriers, one in-phase and
the other quadrature. The point representing 11
is made of two combined signal elements,\nboth
with an amplitude of 1 V. One element is
represented by an in-phase carrier, the other
element by a quadrature carrier.\nThe amplitude
of the final signal element sent for this 2-bit
data element is 2^(1/2), and the phase is 45  .\
nThe argument is similar for the other three
points. All signal elements have an amplitude of
2^(1/2),\nbut their phases are different (45  ,
135  , -135  , and -45  ).Of course, we could
have chosenthe amplitude of the carrier to be
1/(2^(1/2))\nto make the final amplitudes 1 V.");
```

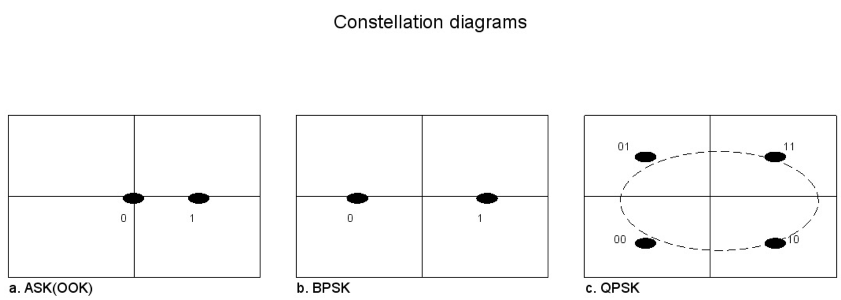

Figure 5.3: Three constellations diagrams

Chapter 6

Bandwidth Utilization Multiplexing and Spreading

Scilab code Exa 6.1 FDM configuration

```
clear;
clc;
disp("————————————Example 6.1————————————")
channel_bandwidth=4; // a voice channel occupies a
   bandwidth of 4 kHz.
n=3; // number of channels
link_bandwidth=12; // kHz
fl=20; // link bandwidth 20-32 kHz
fl1=20; // lower frequency of bandwidth for channel
   1
fh1=fl1+channel_bandwidth; // higher frequency of
   bandwidth for channel 1
fl2=fh1; // lower frequency of bandwidth for channel
    2
fh2=fl2+channel_bandwidth; // higher frequency of
   bandwidth for channel 2
fl3=fh2; // lower frequency of bandwidth for channel
    3
fh3=fl3+channel_bandwidth; // higher frequency of
```

```
    bandwidth for channel 3
printf("The %d- to %d-kHz bandwidth is used for the
    first channel, the %d- to %d-kHz bandwidth for
    the second channel, and the %d- to %d-kHz\
    nbandwidth for the third one. Then they are
    combined as shown in the figure.\n\nAt the
    receiver, each channel receives the entire signal
    , using a filter to separate out its own signal.
    The first channel uses a filter that passes\
    nfrequencies between %d and %d kHz and filters
    out (discards) any other frequencies. The second
    channel uses a filter that passes frequencies
    between\n%d and %d kHz,and the third channel uses
     a filter that passes frequencies between %d and
    %d kHz. Each channel then shifts the frequency to
     start\nfrom zero.",fl1,fh1,fl2,fh2,fl3,fh3,fl1,
    fh1,fl2,fh2,fl3,fh3); // display result
// display the figure
clf();
xname("-----------------Example 6.1-------------------");
xset("font size",3)
for i=0:2
    xset("color",i+1);
    xfpoly([-.05 -.05 -.02 .01 .01],[.6-(i/6) .64-(i
        /6) .68-(i/6) .64-(i/6) .6-(i/6)]);
    xfpoly([1.05 1.05 1.08 1.11 1.11],[.6-(i/6)
        .64-(i/6) .68-(i/6) .64-(i/6) .6-(i/6)]);
    xfpoly([.13+(i/15) .13+(i/15) .16+(i/15) .19+(i
        /15) .19+(i/15)],[.6-(i/6) .64-(i/6) .68-(i
        /6) .64-(i/6) .6-(i/6)]);
    xfpoly([.805+(i/15) .805+(i/15) .835+(i/15)
        .865+(i/15) .865+(i/15)],[.6-(i/6) .64-(i/6)
        .68-(i/6) .64-(i/6) .6-(i/6)]);
    xfpoly([.42+(i/16.5) .42+(i/16.5) .45+(i/16.5)
        .48+(i/16.5) .48+(i/16.5)],[.6-(1/6)
        .64-(1/6) .68-(1/6) .64-(1/6) .6-(1/6)]);
    xset("color",0);
    xpoly([-.1 .05],[.6-(i/6) .6-(i/6)]);
```

```
    xrect(.05,.64-(i/6),.07,.06);
    xstring(.05,.6-(i/6),"Modulator");
    xpoly([.12 .32],[.6-(i/6) .6-(i/6)]);
    xarrows([.33 .36],[.6-(i/6) .435],.5);
    xarrows([.69 .72],[.435 .6-(i/6)],.5);
    xrect(.73,.64-(i/6),.07,.06);
    xstring(.76,.6-(i/6),"Filter");
    xpoly([.8 .995],[.6-(i/6) .6-(i/6)]);
    xpoly([1.01 1.2],[.6-(i/6) .6-(i/6)]);
    xstring(-.05,.56-(i/6),"0");
    xstring(.01,.56-(i/6),"4");
    xstring(1.05,.56-(i/6),"0");
    xstring(1.1,.56-(i/6),"4");
end
xarc(.36,.445,.045,.045,0,64*360);
xarc(.64,.445,.045,.045,0,64*360);
xarrows([.41 .63],[.6-(1/6) .6-(1/6)],.4);
xrect(.04,.69,.37,.46);
xrect(.72,.69,.28,.46);
xset("font size",3)
xstring(.375,.41,"+");
xstring(.04,.7,"Shift and combine");
xstring(.75,.2,"Filter and shift");
xstring(.43,.35,"Higher-bandwidth link");
xstring(.42,.39,"20");
xstring(.59,.39,"32");
xstring(.12,.57,"20");
xstring(.19,.57,"24");
xstring(.8,.57,"20");
xstring(.87,.57,"24");
xstring(.18,.4,"24");
xstring(.25,.4,"28");
xstring(.86,.4,"24");
xstring(.93,.4,"28");
xstring(.25,.23,"28");
xstring(.32,.23,"32");
xstring(.93,.23,"28");
xstring(.99,.23,"32");
```

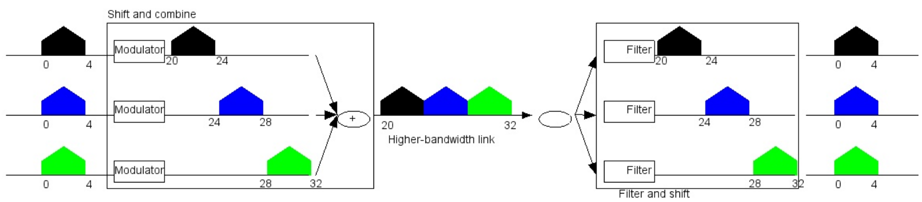

Figure 6.1: FDM configuration

Scilab code Exa 6.2 bandwidth with guards

```
clear;
clc;
disp("—————————————Example 6.2—————————————")
n=5; // five channels
b=100; // bandwidth of each channel in kHz
gb=10; // guard band in kHz
n_gb= n-1; // number of guard bands= number of channels - 1
min_B = (n*b)+(n_gb*gb); // formula for total bandwidth or minimum bandwidth
printf("The required bandwidth is atleast %d kHz.", min_B); // display result
// display the figure
clf();
```

```
xname("---------------Example 6.2-----------------");
xarrows([0 1],[.2 .2],.5);
xset("font size",5);
xstring(1,.1,"Frequency");
xpoly([.1 .8],[.07 .07]);
xpoly([.1 .1],[.09 .05]);
xpoly([.8 .8],[.09 .05]);
xset("font size",4);
xstring(.4,.01,"540 kHz");
x=linspace(.1,.7,5);
y=linspace(.2,.8,5);
for i=1:5
    xpoly([x(i) x(i)],[.2 .3]);
end
for i=1:5
    xpoly([y(i) y(i)],[.2 .3]);
end
for i=0:4
    xarc(.17+(i/6.65),.31,.03,.03,0,90*64);
    xarc(.1+(i/6.65),.31,.03,.03,90*64,91*64);
end

for i=0:4
    xpoly([.11+(i/6.65) .19+(i/6.65)],[.31 .31]);
    xpoly([.1+(i/6.65) .2+(i/6.65)],[.35 .35]);
    xpoly([.1+(i/6.65) .1+(i/6.65)],[.33 .37]);
    xpoly([.2+(i/6.65) .2+(i/6.65)],[.33 .37]);
end

xset("font size",3);
x=linspace(.15,.75,5);
for i=1:5
    xstring(x(i)-.03,.4,"100 kHz");
end
xarrows([.23 .23],[.5 .36],.5);
xstring(.2,.53,"Guard band of 10 kHz");
```

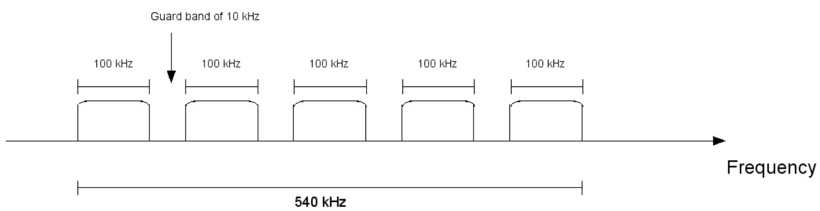

Figure 6.2: bandwidth with guards

Scilab code Exa 6.3 satellite channel FDM

```
clear;
clc;
disp("————————————————Example 6.3————————————————")
n=4; // number of channels
bitrate=1; // 1 Mbps
total_bandwidth=1*10^6; // 1 MHz
channel_bandwidth=total_bandwidth/n;
bits=(bitrate*10^6/total_bandwidth)*n; // number of
    bits per Hz
printf("The satellite channel is analog. Each
    channel has a %d kHz bandwidth.\nEach digital
    channel of %d Mbps is modulated such that each %d
     bits is modulated to 1 Hz. One solution is 16−
    QAM modulation.",channel_bandwidth*10^-3,bitrate,
    bits); // display the result
```

```
// display the figure
clf();
xname("——————————————Example 6.3———————————————");
xpoly([.5 .5],[.3 .8]);
xpoly([.5 .65],[.8 .55]);
xpoly([.5 .65],[.3 .55]);
xset("font size",2.8);
for i=0:3
    xstring(.22,.71-(i/10),"1 Mbps

       250 kHz");
    xstring(.22,.665-(i/10),"Digital

       Analog");
    xpoly([.2 .3],[.7-(i/10) .7-(i/10)]);
    xpoly([.4 .5],[.7-(i/10) .7-(i/10)]);
    xrect(.3,.72-(i/10),.1,.05);
    xstring(.33,.68-(i/10),"16—QAM");
end
xset("font size",4);
xstring(.53,.53,"FDM");
xstring(.67,.57,"1 MHz");
xpoly([.65 .75],[.55 .55]);
```

Scilab code Exa 6.4 Advanced MobilePhone System

```
clear;
clc;
disp("——————————————Example 6.4———————————————")
band= 25*10^6; // each band is 25 MHz
bandwidth= 30 *10^3; // Each user has a bandwidth of
    30 kHz in each direction.
control_channels=42; // 42 channels are used for
```

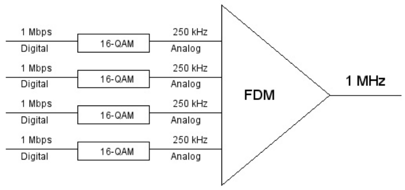

Figure 6.3: satellite channel FDM

```
   control
channels=floor((band/bandwidth)-1); // number of
   channels
user_channels=channels-control_channels; // number
   of channels available for users
printf("%d channels are available for cellular phone
    users.",user_channels); // display result
```

Scilab code Exa 6.5 three durations calculation

```
clear;
clc;
disp("————————————Example 6.5————————————")
data_rate=1*10^3; // data rate for each input
   connection is 1 kbps
unit =1; // 1 bit
n=3; // number of channels
// a) duration of each input slot
bit_duration = 1/data_rate;
ip_timeslot=bit_duration;
```

```
printf("a) The duration of each input time slot is
    %d ms.\n",ip_timeslot*10^3); //display result
//b) duration of each output slot
op_timeslot=ip_timeslot/n;
printf("\nb) The duration of each output time slot
    is %3.2f ms.\n",op_timeslot*10^3); //display
    result
// c)what is the duration of each frame
frame_time = n*op_timeslot;
printf("\nc) The duration of each frame is %d ms.\n"
    ,frame_time*10^3);//display result
```

Scilab code Exa 6.6 Synchronous TDM

```
clear;
clc;
disp("--------------Example 6.6--------------")
data_rate=1*10^6; // data rate for each input
    connection is 1 Mbps
unit =1; // 1 bit
n=4; //number of channels
//a) input bit duration
ip_bd=1/data_rate;
printf("a) The input bit duration is %d microseconds
    .\n",ip_bd*10^6); //display result
//b)output bit duration
op_bd=ip_bd/n;
printf("\nb) The output bit duration is %3.2f
    microseconds.\n",op_bd*10^6);//display result
// c) output bit rate
op_bitrate=1/op_bd;
printf("\nc) The output bit rate is %d Mbps.\n",
    op_bitrate*10^-6);//display result
//d) output frame rate
frame_rate=data_rate;
```

```
printf("\nd) The frame rate is %d frames per second.
   ",frame_rate);//display result
```

Scilab code Exa 6.7 Four connections multiplexing

```
clear;
clc;
disp("————————————Example 6.7—————————————")
data_rate=1*10^3; // data rate for each input
   connection is 1 kbps
unit =1; // 1 bit
n=4; //number of channels
// (a) the duration of 1 bit before multiplexing
bit_duration=1/data_rate;
printf("a)The duration of 1 bit before multiplexing
   is %d ms.\n",bit_duration*10^3); //display result
// (b) the transmission rate of the link
trans_rate=n*data_rate;
printf("\nb)The transmission rate of the link is %d
   kbps.\n",trans_rate*10^-3); //display result
// (c) the duration of a time slot
time_slot = bit_duration/n;
printf("\nc)The duration of each time slot is %d
   microseconds.\n",time_slot*10^6); //display
   result
// (d) the duration of a frame
frame_time = bit_duration;
printf("\nd)The duration of each frame is %d ms.\n",
   frame_time*10^3); //display result
```

Scilab code Exa 6.8 Four channels TDM

```
clear;
```

```
clc;
disp("---------------Example 6.8---------------")
n=4; // number of channels
channel_byte=1; // each frame carries 1 byte from each channel
frame_size=n*channel_byte; //bytes
frame_size_bits=frame_size*8; // 1 byte = 8 bits
byte_rate=100; // each channel sends 100 bytes/s
frame_rate=channel_byte*byte_rate; // frames per second
frame_duration=1/frame_rate; // seconds
bit_rate=frame_rate*frame_size_bits; // bps
// display the result
printf("Each frame carries %d byte from each channel; the size of each frame, therefore, is %d bytes, or %d bits.\nThe frame rate is %d frames per second. The duration of a frame is %3.2f s.\nThe bit rate is %d bps.",channel_byte,frame_size,frame_size_bits,frame_rate,frame_duration,bit_rate);
// display the figure
clf();
xname("---------------Example 6.8-----------------");
xpoly([.3 .3],[.3 .7]);
xset("color",4.2);
xfrect(0,.7,.28,.05);
xfrect(.58,.58,.035,.045);
xfrect(.88,.58,.035,.045);
xset("color",2.9);
xfrect(0,.6,.28,.05);
xfrect(.545,.58,.035,.045);
xfrect(.845,.58,.035,.045);
xset("color",3.8);
xfrect(0,.5,.28,.05);
xfrect(.51,.58,.035,.045);
xfrect(.81,.58,.035,.045);
xset("color",0);
xfrect(0,.4,.28,.05);
```

```
xfrect(.475,.58,.035,.045);
xfrect(.775,.58,.035,.045);
xpoly([-.05 .3],[.625 .625]);
xpoly([-.05 .3],[.525 .525]);
xpoly([-.05 .3],[.425 .425]);
xpoly([-.05 .3],[.325 .325]);
xset("font size",4)
xstring(.33,.5,"MUX");
xpoly([.3 .45],[.7 .5]);
xpoly([.3 .45],[.3 .5]);
xstring(.1,.27,"100 bytes/s");
xset("font size",3)
xstring(.65,.45,"100 frames/s");
xstring(.67,.41,"3200 bps");
xstring(.6,.33,"Frame duration = 1/100 s");
xstring(.5,.7,"Frame 4 bytes");
xstring(.8,.7,"Frame 4 bytes");
xstring(.54,.65,"32 bits");
xstring(.84,.65,"32 bits");
xrect(.47,.59,.15,.07);
xrect(.77,.59,.15,.07);
xset("thickness",2);
xarrows([.45 1],[.5 .5],.2);
xset("font size",7);
xstring(.66,.53,". . .");
```

Scilab code Exa 6.9 2 bits timeslot

```
clear;
clc;
disp("————————————————Example 6.9————————————————")
n=4; // number of channels
channel_bits=2; // each frame carries 2 bits from
```

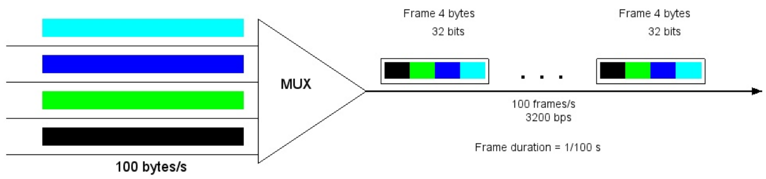

Figure 6.4: Four channels TDM

```
   each channel
bitrate_channel=100*10^3 // kbps of each channel
frame_rate=bitrate_channel/channel_bits; // frames
   per second
frame_duration=1/frame_rate; // seconds
frame_bits=n*channel_bits; // bits carried by each
   frame
bit_rate=frame_rate*frame_bits;// of the link in bps
bit_duration=1/bit_rate; // seconds
// display result with appropriate units
printf("The frame rate is %d frames per second. The
   frame duration is therefore %d microseconds .\
   nEach frame carries %d bits; the bit rate is %d
   kbps.\nThe bit duration is %2.1f microseconds.",
   frame_rate,frame_duration*10^6,frame_bits,
   bit_rate*10^-3,bit_duration*10^6);//display
   result
// display the figure
clf();
xname("—————————Example 6.9——————————");
xpoly([0 .3],[.7 .7]);
xpoly([0 .3],[.6 .6]);
```

```
xpoly([0 .3],[.5 .5]);
xpoly([0 .3],[.4 .4]);
xpoly([.3 .3],[.3 .8]);
xset("font size",4)
xstring(.33,.53,"MUX");
xpoly([.3 .45],[.8 .55]);
xpoly([.3 .45],[.3 .55]);
xset("font size",2.5)
xstring(.05,.71,"100 kbps                    ...
      110010");
xstring(.05,.61,"100 kbps                    ...
      001010");
xstring(.05,.51,"100 kbps                    ...
      101101");
xstring(.05,.41,"100 kbps                    ...
      000111");
xset("font size",3)
xstring(.65,.5,"50,000 frames/s");
xstring(.67,.46,"400 kbps");
xstring(.6,.8,"Frame duration = 1/50000 = 20 microseconds");
xstring(.5,.65,"Frame : 8 bits");
xstring(.69,.65,"Frame : 8 bits");
xstring(.88,.65,"Frame : 8 bits");
xstring(.51,.58,"00");
xstring(.55,.58,"10");
xstring(.59,.58,"00");
xstring(.63,.58,"11");
xstring(.7,.58,"01");
xstring(.74,.58,"11");
xstring(.78,.58,"10");
xstring(.82,.58,"00");
xstring(.89,.58,"11");
xstring(.93,.58,"01");
xstring(.97,.58,"10");
xstring(1.01,.58,"10");
xstring(.48,.58,"...");
xrects([.5 .54 .58 .62;.63 .63 .63 .63;.04 .04 .04
```

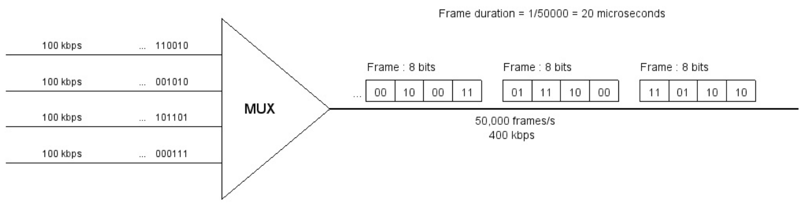

Figure 6.5: 2 bits timeslot

```
    0.04;.06 .06 .06 .06]);
xrects([.69 .73 .77 .81;.63 .63 .63 .63;.04 .04 .04
    0.04;.06 .06 .06 .06]);
xrects([.88 .92 .96 1.0;.63 .63 .63 .63;.04 .04 .04
    0.04;.06 .06 .06 .06]);
xset("thickness",2);
xpoly([.45 1.1],[.55 .55]);
```

Scilab code Exa 6.10 4 sources interleaving

```
clear;
clc;
disp("--------------Example 6.10--------------")
cps=250; // characters per second
unit=8; // 1 unit = 1 character = 8 bits
n=4; //number of sources
sb=1; // 1 synchronization bit
//a)the data rate of each source
```

```
data_rate_source=cps*unit;
printf("a)The data rate of each source is %d kps.\n"
    ,data_rate_source*10^-3); // display result
// b) the duration of each character in each source
character_duration=1/cps;
printf("\nb)The duration of each character in each
    source is %d ms.\n",character_duration*10^3); //
    display result
// c) the frame rate
frame_rate=cps;
printf("\nc)The frame rate is %d frames per second.\
    n",frame_rate);// display result
// d) the duration of each frame
frame_duration=1/frame_rate;
printf("\nd)The duration of each frame is %d ms.\n",
    frame_duration*10^3);// display result
//e) the number of bits in each frame
bits=n*unit+sb;
printf("\ne)The number of bits in each frame is %d.\
    n",bits); // display result
//f) the data rate of the link
data_rate_link=frame_rate*bits;
printf("\nf)The data rate of the link is %d bps.",
    data_rate_link); // display result
```

Scilab code Exa 6.11 2 channels multiplexing

```
clear;
clc;
disp("--------------Example 6.11---------------")
N1=100; // bit rate of one channel = 100 kbps
N2=200; // bit rate of other channel = 200 kbps
printf("\n Multiplexing can be achieved by
    allocating one slot to the first channel and two
    slots to the second channel.\n");
```

```
bits=3; // let each frame carry 3 bits
frame_rate=N1*10^3;
frame_duration=1/frame_rate;
bit_rate=frame_rate*bits;
printf("\nThe frame rate is %d frames per second ,
    the frame duration is %d ms and the bit rate of
    the link is %d kbps.\n",frame_rate,frame_duration
    *10^6,bit_rate*10^-3); // display result
```

Chapter 8

Switching

Scilab code Exa 8.1 circuit switched network for telephones

```
clear;
clc;
disp("——————————————Example 8.1——————————————")
channel_bandwidth=4; // 4 kHz
n=2; // each link uses FDM to connect a maximum of
   two voice channels.
link_bandwidth=n*channel_bandwidth; // formula
// display the result
printf("A circuit-switched network is used to
   connect eight telephones in a small area.
   Communication is through %d-kHz voice channels.\
   nIt is assumed that each link uses FDM to connect
    a maximum of %d voice channels. The bandwidth of
    each link is then %d kHz.\nTelephone 1 is
   connected to telephone 7; 2 to 5; 3 to 8; and 4
   to 6. Of course the situation may change when new
    connections are made.\nThe switch controls the
   connections.",channel_bandwidth,n,link_bandwidth)
   ;
```

Scilab code Exa 8.2 circuit switched network of computers

```
clear;
clc;
disp("-----------------------------Example 8.2-----------------------------")
printf("Consider a circuit-switched network that connects computers in two remote offices of a private company. The offices are\nconnected using a T-1 line leased from a communication service provider. There are two 4 X 8 (4 inputs and 8 outputs)\nswitches in this network. For each switch, four output ports are folded into the input ports to allow communication between\ncomputers in the same office. Four other output ports allow communication between the two offices."); // display example explanation
// display the figure
clf();
xname("-----------------------------Example 8.2-------------------------------");
xset("font size",3);
xstring(0,.9,"Circuit-switched network");
xstring(.12,.67,"4x8 switch");
xstring(.62,.67,"4x8 switch");
xstring(.3,.63,"T-1 line with 1.544 Mbps");
xrects([0 .1 .6;.89 .8 .8;.8 .1 .1;.6 .3 .3]);
xpoly([.23 .28 .23],[.73 .68 .63],"lines",1);
xpoly([.57 .52 .57],[.73 .68 .63],"lines",1);
for i=0:2
    xpoly([.2 .23],[.72-(i/25) .72-(i/25)]);
    xpoly([.57 .6],[.72-(i/25) .72-(i/25)]);
end

for i=0:3
    xpoly([.11+(i/45) .11+(i/45)],[.5 .45-(i/25)]);
```

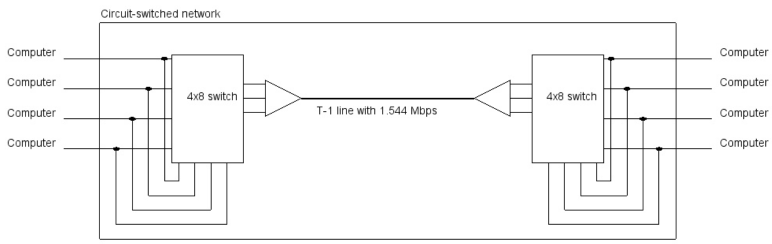

Figure 8.1: circuit switched network of computers

```
    xpoly([.11+(i/45) .09-(i/45)],[.45-(i/25) .45-(i/25)]);
    xpoly([.09-(i/45) .09-(i/45)],[.45-(i/25) .79-(i/12)]);
    xpoly([.69-(i/45) .69-(i/45)],[.5 .45-(i/25)]);
    xpoly([.69-(i/45) .71+(i/45)],[.45-(i/25) .45-(i/25)]);
    xpoly([.71+(i/45) .71+(i/45)],[.45-(i/25) .79-(i/12)]);
    xpoly([-.05 .1],[.79-(i/12) .79-(i/12)]);
    xpoly([.7 .85],[.79-(i/12) .79-(i/12)]);
    xstring(-.13,.79-(i/12),"Computer");
    xstring(.86,.79-(i/12),"Computer");
    xfarc(.085-(i/45),.795-(i/12),.01,.01,0,64*360);
    xfarc(.705+(i/45),.795-(i/12),.01,.01,0,64*360);
end
xset("thickness",2.5);
xpoly([.28 .52],[.68 .68]);
```

Scilab code Exa 8.3 Design a three stage switch

```
clear;
clc;
disp("————————————————Example 8.3————————————————")
N=200; //input lines
n=20; // lines in a group
k=4; //number of crossbars in the middle stage
crossbars1=N/n; // 1st stage
crossbars2=k; // 2nd stage
crossbars3=N/n; // 3rd stage
size1a=n; // size of a crossbar in 1st stage
size1b=k;
size2=N/n; // size of a crossbar in 2nd stage
size3a=k; // size of a crossbar in 3rd stage
size3b=n;
total_crosspoints= 2*k*N + k*((N/n)^2); //2kN + k(N/n)^2
singlestage=40000;
p=(total_crosspoints/singlestage)*100;
// display result
printf("\nIn the first stage there are %d crossbars, each of size %d x %d.\n",crossbars1,size1a,size1b);
printf("\nIn the second stage there are %d crossbars, each of size %d x %d.\n",crossbars2,size2,size2);
printf("\nIn the third stage there are %d crossbars, each of size %d x %d.\n",crossbars3,size3a,size3b);
printf("\nThe total number of crosspoints is %d. This is %d percent of the number of crosspoints in a single stage switch.",total_crosspoints,p);
```

Scilab code Exa 8.4 Clos criteria switch

```
clear;
clc;
disp("---------------Example 8.4---------------")
N=200;
n=(N/2)^(0.5); // formula
k=2*n-1; // formula
crossbars_1stage=N/n; // formula
crosspoints_1stage=n*k; // formula
crossbars_2stage=k; // formula
crosspoints_2stage=n*n; // formula
crossbars_3stage=N/n; // formula
crosspoints_3stage=k*n; // formula
total_crosspoints=(crossbars_1stage*crosspoints_1stage)+(crossbars_2stage*crosspoints_2stage)+(crossbars_3stage*crosspoints_3stage); // formula
crosspoints_singlestage=N*N; // formula
p=(total_crosspoints/crosspoints_singlestage)*100; // percentage formula
// display the resut
printf("The value of n is %d and the value of k is %d .\nIn the first stage, there are %d crossbars, each with %d X %d crosspoints.\nIn the second stage, there are %d crossbars, each with %d X %d crosspoints.\nIn the third stage, there are %d crossbars each with %d X %d crosspoints.\nThe total number of crosspoints is %d .If a single-stage switch is used, %d crosspoints are needed.\nThe number of crosspoints in this three-stage switch is %d percent that of a single-stage switch.\nMore points are needed than in single stage . The extra crosspoints are needed to
```

```
prevent blocking.",n,k,crossbars_1stage,n,k,
crossbars_2stage,n,n,crossbars_3stage,k,n,
total_crosspoints,crosspoints_singlestage,ceil(p)
);
```

Chapter 10

Error Detection and Correction

Scilab code Exa 10.1 block coding error

```
clear;
clc;
disp("---------------Example 10.1---------------")
k=4;
n=5;
datawords=2^k; // number of datawords
codewords=2^n; // number of codewords
printf("The 4B/5B block coding scheme has %d
    datawords and %d codewords.\n 16 out of 32
    codewords are used for message transfer and the
    rest are either used for other purposes or unused
    .",datawords,codewords); // display result
```

Scilab code Exa 10.2 2 bit dataword

```
clear;
clc;
disp("---------------Example 10.2---------------")
```

```
k=2;
n=3;
table=["Datawords","Codewords";"00","000";"01","011";"10","101";"11","110"];
disp(table) // display the table
dataword="01";
codeword="011";
printf("\nAssume the sender encodes the dataword 01 as 011 and sends it to the receiver. Consider the following cases:\n");
function []=case_func(codeword,dataword) // function to display appropriate result
    select codeword
    case "011"
        printf("\n1. The receiver receives %s. It is a valid codeword. The receiver extracts the dataword %s from it.",codeword,dataword);
    case "111"
        printf("\n\n2. The codeword is corrupted during transmission, and %s is received (the leftmost bit is corrupted).\nThis is not a valid codeword and is discarded.",codeword);
    case "000"
        printf("\n\n3. The codeword is corrupted during transmission, and %s is received (the right two bits are corrupted).\nThis is a valid codeword. The receiver incorrectly extracts the dataword 00. Two corrupted bits have made the error undetectable.",codeword);
    end
endfunction
funcprot(0);
// case 1
case_func(codeword,dataword); //calling the function
// case 2
```

```
codeword="111";
case_func(codeword,dataword); // calling the function
// case 3
codeword="000";
case_func(codeword,dataword); // calling the function
```

Scilab code Exa 10.3 3 redundant bits codeword

```
clear;
clc;
disp("————————————————Example 10.3————————————————")
table=["Datawords","Codewords";"00","00000";"01","01011";"10","10101";"11","11110"];
codewords=["00000","01011","10101","11110"];
disp(table); // display the table
dataword="01";
codeword="01011";
corrupt_codeword="01001";
printf("\nThe dataword is %s. The sender consults the table to create the codeword %s.\nThe codeword is corrupted during transmission, and %s is received (error in the second bit from the right).\nFirst, the receiver finds that the received codeword is not in the table. This means an error has occurred. (Detection must come before correction.)\nThe receiver, assuming that there is only 1 bit corrupted, uses the following strategy to guess the correct dataword.\n",dataword,codeword,corrupt_codeword); // display result
for i=1:4 // check for each codeword
    bit=strsplit(codewords(i));
    r=strsplit(corrupt_codeword);
```

```
    count=0;
    for k=1:5
        if(bit(k)==r(k)) // check each bit
            continue;
        else
            count=count+1; // udate the count of
               errorenous bits
        end
    end
    if(count>1) // if more than 1 bit is errorenous
        continue;
    else
        correct_codeword=codewords(i); // the
           correct codeword determined
        break;
    end
end
printf("\n1. Comparing the received codeword with
   the first codeword in the table (%s versus 00000)
   ,the receiver decides\nthat the first codeword is
    not the one that was sent because there are two
   different bits.\n\n2. By the same reasoning, the
   original codeword cannot be the third or fourth
   one in the table.\n\n3. The original codeword
   must be the second one in the table because this
   is the only one that differs from the received
   codeword by 1 bit.\nThe receiver replaces %s with
    %s and consults the table to find the dataword
   %s.",corrupt_codeword,corrupt_codeword,
   correct_codeword,dataword); // display result
```

Scilab code Exa 10.4 find Hamming distance

```
clear;
clc;
```

```
disp("——————————————Example 10.4——————————————")
//words
x1=[0 0 0];
y1=[0 1 1];
x2=[1 0 1 0 1];
y2=[1 1 1 1 0];
// formula to find Hamming distance 'd'
d1=bitxor(x1,y1);
d2=bitxor(x2,y2);
function [count]= num_of_ones (d)// function to find the number of ones in a binary number
    count=0;
    for i=1:length(d)
        if(d(i)== 1)
            count = count+1; // number of one's
        end
    end
endfunction
d=num_of_ones(d1); // calling the function
printf("\nThe Hamming distance d(OOO, 011) is %d.\n",d); // display result
d=num_of_ones(d2); // calling the function
printf("\nThe Hamming distance d(10101, 11110) is %d.\n",d); // display result
```

Scilab code Exa 10.5 minimum Hamming distance

```
clear;
clc;
disp("——————————————Example 10.5——————————————")
//words
x1=[0 0 0];
x2=[0 1 1];
x3=[1 0 1];
x4=[1 1 0];
```

```
//function to find Hamming distance
function [d]=hamming_distance(x,y)
    xd=bitxor(x,y);
    d=num_of_ones(xd);
endfunction
function [count]= num_of_ones (d)// function to find the number of ones in a binary number
    count=0;
    for i=1:length(d)
        if(d(i)== 1)
            count = count+1; // number of ones
        end
    end
endfunction
d1=hamming_distance(x1,x2);
printf("\nThe Hamming distance d(OOO, 011) is %d.\n",d1); // display result
d2=hamming_distance(x1,x3);
printf("\nThe Hamming distance d(OOO, 101) is %d.\n",d2); // display result
d3=hamming_distance(x1,x4);
printf("\nThe Hamming distance d(OOO, 110) is %d.\n",d3); // display result
d4=hamming_distance(x2,x3);
printf("\nThe Hamming distance d(O11, 101) is %d.\n",d4); // display result
d5=hamming_distance(x2,x4);
printf("\nThe Hamming distance d(O11, 110) is %d.\n",d5); // display result
d6=hamming_distance(x3,x4);
printf("\nThe Hamming distance d(101, 110) is %d.\n",d6); // display result
dmin=min(d1,d2,d3,d4,d5,d6);
printf("\nThe minimum Hamming distance dmin is %d.",dmin); // display result
```

Scilab code Exa 10.6 minimum Hamming distance 2

```
clear;
clc;
disp("−−−−−−−−−−−−−−−Example 10.6−−−−−−−−−−−−−−−−")
//words
x1=[0 0 0 0 0];
x2=[0 1 0 1 1];
x3=[1 0 1 0 1];
x4=[1 1 1 1 0];
//function to find Hamming distance
function [d]=hamming_distance(x,y)
    xd=bitxor(x,y);
    d=num_of_ones(xd);
endfunction
function [count]= num_of_ones (d)// function to find
    the number of ones in a binary number
    count=0;
    for i=1:length(d)
        if(d(i)== 1)
            count = count+1; //number of ones
        end
    end
endfunction
d1=hamming_distance(x1,x2);
printf("\nThe Hamming distance d(OOO00, 01011) is %d
   .\n",d1); // display result
d2=hamming_distance(x1,x3);
printf("\nThe Hamming distance d(OOO00, 10101) is %d
   .\n",d2); // display result
d3=hamming_distance(x1,x4);
printf("\nThe Hamming distance d(OOO00, 11110) is %d
   .\n",d3); // display result
d4=hamming_distance(x2,x3);
```

```
printf("\nThe Hamming distance d(O1011, 10101) is %d
    .\n",d4); // display result
d5=hamming_distance(x2,x4);
printf("\nThe Hamming distance d(O1011, 11110) is %d
    .\n",d5); // display result
d6=hamming_distance(x3,x4);
printf("\nThe Hamming distance d(10101, 11110) is %d
    .\n",d6); // display result
dmin=min(d1,d2,d3,d4,d5,d6);
printf("\nThe minimum Hamming distance dmin is %d.",
    dmin); // display result
```

Scilab code Exa 10.7 error detection and correction

```
clear;
clc;
disp("————————————————Example 10.7————————————————")
printf("The minimum Hamming distance for the first
    code scheme (Table 10.1) is 2.\nThis code
    guarantees detection of only a single error. For
    example, if the third codeword (l01)\nis sent and
     one error occurs, the received codeword does not
     match any valid codeword. If two errors occur,
    however,\nthe received codeword may match a valid
     codeword and the errors are not detected."); //
    display the example
```

Scilab code Exa 10.8 block code dmin

```
clear;
clc;
disp("————————————————Example 10.8————————————————")
```

```
printf("The second block code scheme (Table 10.2)
   has dmin = 3. This code can detect up to two
   errors.\nWhen any of the valid codewords is sent,
    two errors create a codeword which is not in the
    table of valid codewords.\nHowever, some
   combinations of three errors change a valid
   codeword to another valid codeword.\nThe receiver
    accepts the received codeword and the errors are
    undetected."); //display the example
```

Scilab code Exa 10.9 Hamming distance dmin 4

```
clear;
clc;
disp("——————————————Example 10.9——————————————")
dmin=4; // minimum Hamming distance
s=dmin-1; //error detection
t=(dmin-1)/2; //error correction
// display result
printf("This code guarantees the detection of up to
   %d errors.\n\n",s);
printf("It can correct upto %d error.\nIn other
   words, if this code is used for error correction,
    part of its capability is wasted. Error
   correction codes need to have an odd\nminimum
   distance (3, 5, 7, ... ).",t);
```

Scilab code Exa 10.10 linear block codes

```
clear;
clc;
disp("——————————————Example 10.10——————————————")
```

```
// function to check if given scheme is a linear
   block code
function[]=check_linearcode (c1,c2,c3,c4)
    if(bitxor(c1,c2)==c1|bitxor(c1,c2)==c2|bitxor(c1
       ,c2)==c3|bitxor(c1,c2)==c4)
        if(bitxor(c1,c3)==c1|bitxor(c1,c3)==c2|
           bitxor(c1,c3)==c3|bitxor(c1,c3)==c4)
            if(bitxor(c1,c4)==c1|bitxor(c1,c4)==c2|
               bitxor(c1,c4)==c3|bitxor(c1,c4)==c4)
                if(bitxor(c2,c3)==c1|bitxor(c2,c3)==
                   c2|bitxor(c2,c3)==c3|bitxor(c2,c3
                   )==c4)
                    if(bitxor(c2,c4)==c1|bitxor(c2,
                       c4)==c2|bitxor(c2,c4)==c3|
                       bitxor(c2,c4)==c4)
                        if(bitxor(c3,c4)==c1|bitxor(
                           c3,c4)==c2|bitxor(c3,c4)
                           ==c3|bitxor(c3,c4)==c4)
                            printf("\nThis scheme is
                                a linear block code
                               because the result of
                                XORing any codeword
                               with any other
                               codeword is a valid
                               codeword.\n");
                        else
                            printf("\nThis scheme is
                                not a linear block
                               code because the
                               result of XORing any
                               codewordwith any
                               other codeword is not
                                a valid codeword.\n"
                               );
                        end
                    else
                        printf("\nThis scheme is not
                            a linear block code
```

```
because the result of XORing any codewordwith any other codeword is not a valid codeword.\n");
                    end
                else
                    printf("\nThis scheme is not a linear block code because the result of XORing any codewordwith any other codeword is not a valid codeword.\n");
                end
            else
                printf("\nThis scheme is not a linear block code because the result of XORing any codewordwith any other codeword is not a valid codeword.\n");
            end
        else
            printf("\nThis scheme is not a linear block code because the result of XORing any codewordwith any other codeword is not a valid codeword.\n");
        end
    else
        printf("\nThis scheme is not a linear block code because the result of XORing any codewordwith any other codeword is not a valid codeword.\n");
    end
endfunction

// 1) Table 10.1
c1=[0 0 0];
c2=[0 1 1];
```

```
c3=[1 0 1];
c4=[1 1 0];
printf("\n1)");
check_linearcode(c1,c2,c3,c4); //calling the
   function

// 2) Table 10.2
c1=[0 0 0 0 0];
c2=[0 1 0 1 1];
c3=[1 0 1 0 1];
c4=[1 1 1 1 0];
printf("\n2)");
check_linearcode(c1,c2,c3,c4);  //calling the
   function
```

Scilab code Exa 10.11 d min calculation

```
clear;
clc;
disp("--------------Example 10.11--------------")
function [count]= num_of_ones (d)// function to find
    the number of ones in a binary number
    count=0;
    for i=1:length(d)
        if(d(i)== 1)
            count = count+1; // number of ones
        end
    end
endfunction
//1)Table 10.1
c1=[0 0 0];
c2=[0 1 1];
c3=[1 0 1];
c4=[1 1 0];
o2=num_of_ones(c2);
```

```
o3=num_of_ones(c3);
o4=num_of_ones(c4);
dmin=min(o2,o3,o4); //The minimum Hamming distance is the number of 1s in the nonzero valid codeword with the smallest number of 1s.
printf("\nIn the first code (Table 10.1), the numbers of 1s in the nonzero codewords are %d,%d, and %d. So the minimum Hamming distance is dmin = %d.",o2,o3,o4,dmin); //display result

// 2) Table 10.2
c1=[0 0 0 0 0];
c2=[0 1 0 1 1];
c3=[1 0 1 0 1];
c4=[1 1 1 1 0];
o2=num_of_ones(c2);
o3=num_of_ones(c3);
o4=num_of_ones(c4);
dmin=min(o2,o3,o4); //The minimum Hamming distance is the number of 1s in the nonzero valid codeword with the smallest number of 1s.
printf("\nIn the second code (Table 10.2), the numbers of 1s in the nonzero codewords are %d,%d, and %d. So the minimum Hamming distance is dmin = %d.",o2,o3,o4,dmin); // display result
```

Scilab code Exa 10.12 some transmission scenarios

```
clear;
clc;
disp("--------------Example 10.12---------------")
dataword=1011;
function []= display (codeword) // function to display the result according to the codeword recieved at the reciever
```

```
select codeword
case 10111
    printf("\n1)No error occurs;the received codeword is %d.The syndrome is O. The dataword 1011 is created.\n",codeword);
case 10011
    printf("\n2)One single-bit error changes a1. The received codeword is %d.The syndrome is 1.No dataword is created.\n",codeword);
case 10110
    printf("\n3)One single-bit error changes r0. The received codeword is %d.The syndrome is 1.No dataword is created.\nNote that although none of the dataword bits are corrupted, no dataword is created because the code is not\nsophisticated enough to show the position of the corrupted bit.\n",codeword);
case 00110
    printf("\n4)An error changes r0 and a second error changes a3.The received codeword is 00110.The syndrome is 0.\nThe dataword 0011 is created at the receiver. Note that here the dataword is wrongly created due to the syndrome value.\nThe simple parity-check decoder cannot detect an even number of errors. The errors cancel each other out and give the syndrome a value of 0.\n");
case 01011
    printf("\n5)Three bits-a3, a2, and a1 are changed by errors.The received codeword is 01011.The syndrome is 1.\nThe dataword is not created. This shows that the simple parity check, guaranteed to detect one single error,\ncan also find any odd number of errors.\n");
```

```
    end
endfunction

//codeword = 10111
codeword = 10111;
display(10111); //calling the function
//codeword = 10011
codeword = 10011;
display(10011); //calling the function
//codeword = 10110
codeword = 10110;
display(10110); //calling the function
//codeword = 00110
codeword = 00110;
display(00110); //calling the function
//codeword = 01011
codeword = 01011;
display(01011); //calling the function
funcprot(0);
```

Scilab code Exa 10.13 path of three datawords

```
clear;
clc;
disp("—————————————Example 10.13—————————————")
function[codeword]=generate_codeword (dataword)  //
   functionto generate the codeword at the sender
    r0=bitxor(bitxor(matrix(dataword(4),1,1),matrix(
       dataword(3),1,1)),matrix(dataword(2),1,1));
       // r0=a0+a1+a2
    s1=bitxor(bitxor(matrix(dataword(3),1,1),matrix(
       dataword(2),1,1)),matrix(dataword(1),1,1));
       // s1=a1+a2+a3
    r2=bitxor(bitxor(matrix(dataword(3),1,1),matrix(
       dataword(4),1,1)),matrix(dataword(1),1,1));
```

```
    // r2=a0+a1+a3
    codeword=string(dataword(1))+string(dataword(2))+string(dataword(3))+string(dataword(4))+string(r2)+string(s1)+string(r0); // form the codeword
endfunction
function[syndrome] = generate_syndrome(codeword_recieved)  // function to generate syndrome at the reciever
    s0=bitxor(bitxor(matrix(codeword_recieved(7),1,1),matrix(codeword_recieved(2),1,1)),bitxor(matrix(codeword_recieved(3),1,1),matrix(codeword_recieved(4),1,1)));  // s0=b2+b1+b0+q0
    s1=bitxor(bitxor(matrix(codeword_recieved(6),1,1),matrix(codeword_recieved(1),1,1)),bitxor(matrix(codeword_recieved(2),1,1),matrix(codeword_recieved(3),1,1)));  // s0=b3+b2+b1+q1
    s2=bitxor(bitxor(matrix(codeword_recieved(5),1,1),matrix(codeword_recieved(4),1,1)),bitxor(matrix(codeword_recieved(3),1,1),matrix(codeword_recieved(1),1,1)));  // s0=b3+b1+b0+q2
    syndrome=string(s2)+string(s1)+string(s0); // the syndrome formed
endfunction

function[]=find_error (syndrome,dataword,codeword,codeword_recieved) // functin to find the error bit and display the final corrected data word
    select syndrome
    case "000"
        dw=string(dataword(1))+string(dataword(2))+string(dataword(3))+string(dataword(4));
        cw=string(codeword_recieved(1))+string(codeword_recieved(2))+string(codeword_recieved(3))+string(
```

```
    codeword_recieved(4))+string(
    codeword_recieved(5))+string(
    codeword_recieved(6))+string(
    codeword_recieved(7));
  printf("The dataword %s becomes the codeword
     %s. The codeword %s is received.The
    syndrome is %s (no error), the final
    dataword is %s.",dw,codeword,cw,syndrome,
    dw);
case "001"
  dw=string(dataword(1))+string(dataword(2))+
    string(dataword(3))+string(dataword(4));
  cw=string(codeword_recieved(1))+string(
    codeword_recieved(2))+string(
    codeword_recieved(3))+string(
    codeword_recieved(4))+string(
    codeword_recieved(5))+string(
    codeword_recieved(6))+string(
    codeword_recieved(7));
  error_bit="q0";
  printf("The dataword %s becomes the codeword
     %s. The codeword %s is received. The
    syndrome is %s.\n%s is the error. After
    flipping %s, the final dataword is %s.",
    dw,codeword,cw,syndrome,error_bit,
    error_bit,dw);
case "010"
  dw=string(dataword(1))+string(dataword(2))+
    string(dataword(3))+string(dataword(4));
  cw=string(codeword_recieved(1))+string(
    codeword_recieved(2))+string(
    codeword_recieved(3))+string(
    codeword_recieved(4))+string(
    codeword_recieved(5))+string(
    codeword_recieved(6))+string(
    codeword_recieved(7));
  error_bit="q1";
  printf("The dataword %s becomes the codeword
```

```
 %s. The codeword %s is received. The syndrome is %s.\n%s is the error. After flipping %s, the final dataword is %s.",dw,codeword,cw,syndrome,error_bit,error_bit,dw);
    case "011"
        dw=string(dataword(1))+string(dataword(2))+string(dataword(3))+string(dataword(4));
        cw=string(codeword_recieved(1))+string(codeword_recieved(2))+string(codeword_recieved(3))+string(codeword_recieved(4))+string(codeword_recieved(5))+string(codeword_recieved(6))+string(codeword_recieved(7));
        error_bit="b2";
        fdw=string(codeword_recieved(1))+string(bitcmp(codeword_recieved(2),1))+string(codeword_recieved(3))+string(codeword_recieved(4)); // corrected dataword
        printf("The dataword %s becomes the codeword %s. The codeword %s is received. The syndrome is %s.\n%s is the error. After flipping %s, the final dataword is %s.",dw,codeword,cw,syndrome,error_bit,error_bit,fdw);
    case "100"
        dw=string(dataword(1))+string(dataword(2))+string(dataword(3))+string(dataword(4));
        cw=string(codeword_recieved(1))+string(codeword_recieved(2))+string(codeword_recieved(3))+string(codeword_recieved(4))+string(codeword_recieved(5))+string(codeword_recieved(6))+string(codeword_recieved(7));
        error_bit="q2";
```

```
        printf("The dataword %s becomes the codeword %s. The codeword %s is received. The syndrome is %s.\n%s is the error. After flipping %s, the final dataword is %s.",dw,codeword,cw,syndrome,error_bit,error_bit,dw);
    case "101"
        dw=string(dataword(1))+string(dataword(2))+string(dataword(3))+string(dataword(4));
        cw=string(codeword_recieved(1))+string(codeword_recieved(2))+string(codeword_recieved(3))+string(codeword_recieved(4))+string(codeword_recieved(5))+string(codeword_recieved(6))+string(codeword_recieved(7));
        error_bit="b0";
        fdw=string(codeword_recieved(1))+string(codeword_recieved(2))+string(codeword_recieved(3))+string(bitcmp(codeword_recieved(4),1)); // corrected dataword
        printf("The dataword %s becomes the codeword %s. The codeword %s is received. The syndrome is %s.\n%s is the error. After flipping %s, the final dataword is %s.",dw,codeword,cw,syndrome,error_bit,error_bit,fdw);
    case "110"
        dw=string(dataword(1))+string(dataword(2))+string(dataword(3))+string(dataword(4));
        cw=string(codeword_recieved(1))+string(codeword_recieved(2))+string(codeword_recieved(3))+string(codeword_recieved(4))+string(codeword_recieved(5))+string(codeword_recieved(6))+string(codeword_recieved(7));
```

```
        error_bit="b3";
        fdw=string(bitcmp(codeword_recieved(1),1))+string(codeword_recieved(2))+string(codeword_recieved(3))+string(codeword_recieved(4)); // corrected dataword
        printf("The dataword %s becomes the codeword %s. The codeword %s is received. The syndrome is %s.\n%s is the error. After flipping %s, the final dataword is %s.",dw,codeword,cw,syndrome,error_bit,error_bit,fdw);
    case "111"
        dw=string(dataword(1))+string(dataword(2))+string(dataword(3))+string(dataword(4));
        cw=string(codeword_recieved(1))+string(codeword_recieved(2))+string(codeword_recieved(3))+string(codeword_recieved(4))+string(codeword_recieved(5))+string(codeword_recieved(6))+string(codeword_recieved(7));
        error_bit="b1";
        fdw=string(codeword_recieved(1))+string(codeword_recieved(2))+string(bitcmp(codeword_recieved(3),1))+string(codeword_recieved(4)); // corrected dataword
        printf("The dataword %s becomes the codeword %s. The codeword %s is received. The syndrome is %s.\n%s is the error. After flipping %s, the final dataword is %s.",dw,codeword,cw,syndrome,error_bit,error_bit,fdw);
    end
endfunction

// 1)
```

```
dataword=[0 1 0 0];
codeword=generate_codeword(dataword); // calling the function
codeword_recieved=[0 1 0 0 0 1 1];
syndrome=generate_syndrome(codeword_recieved) // calling the function
printf("\n1)");
find_error(syndrome,dataword,codeword,codeword_recieved); // calling the function

// 2)
dataword=[0 1 1 1];
codeword=generate_codeword(dataword); // calling the function
codeword_recieved=[0 0 1 1 0 0 1];
syndrome=generate_syndrome(codeword_recieved) // calling the function
printf("\n\n2)");
find_error(syndrome,dataword,codeword,codeword_recieved); // calling the function

// 3)
dataword=[1 1 0 1];
codeword=generate_codeword(dataword); // calling the function
codeword_recieved=[0 0 0 1 0 0 0];
syndrome=generate_syndrome(codeword_recieved) // calling the function
printf("\n\n3)");
find_error(syndrome,dataword,codeword,codeword_recieved); // calling the function
printf("\nThis is the wrong dataword. This shows that Hamming code cannot correct two errors.");
```

Scilab code Exa 10.14 calculate k and n

```
clear;
clc;
disp("−−−−−−−−−−−−−−−Example 10.14−−−−−−−−−−−−−−−−")
// function to chexk if k >=7 and display appropriate result
function[] = check (m)
    n=2^m - 1;
    k=n-m;
    if( k > = 7)
        printf(" m = %d :− The code is C(%d, %d) or k = %d and n = %d.\n",m,n,k,k,n);
    else
        printf(" m = %d :− n = %d .k = %d , which is less than 7. Hence doesnt satisfy the condition.\n",m,n,k);
    end
endfunction
// case 1
m=3;
printf("\n1)");
check(m); // calling the function
//case 2
m=4;
printf("\n2)");
check(m);// calling the function
```

Scilab code Exa 10.15 single bit error

```
clear;
clc;
disp("−−−−−−−−−−−−−−−Example 10.15−−−−−−−−−−−−−−−−")
//a) g(x)= x+1
gx="x+1";
printf("\na)No x^i can be divisible by x + 1. In other words, x^i/()x + 1) always has a remainder.
```

```
    So the syndrome is nonzero. Any single-bit error
    can be caught.\n"); // display result
//b) g(x)= x3
gx="x3";
printf("\nb)If i is equal to or greater than 3, x^i
   is divisible by g(x). The remainder of x^i/x3 is
   zero, and the receiver is fooled into believing\
   nthat there is no error,although there might be
   one.Note that in this case, the corrupted bit
   must be in position 4 or above.\nAll single-bit
   errors in positions 1 to 3 are caught.\n"); //
   display result
//c) 1
gx="1";
printf("\nc)All values of i make x^i divisible by g(
   x). No single-bit error can be caught. In
   addition, this g(x) is useless because it means
   the\ncodeword is just the dataword augmented with
    (n - k) zeros."); // display result
```

Scilab code Exa 10.16 two isolated single bit errors

```
clear;
clc;
disp("----------------Example 10.16----------------");
x=poly(0,"x");
// a) x+1
g=x^1+1;
t=0;
// compute t
while(%T)
    q=(x^t+1)/g;
    if(q == 1)
        break;
    end
```

```
    t=t+1;
end
printf("a. t = %d . This is a very poor choice for a generator. Any two errors next to each other cannot be detected.\n\n",t); // display result
// b) x^4+1
g=x^4+1;
t=0;
// compute t
while(%T)
    q=(x^t+1)/g;
    if(q == 1)
        break;
    end
    t=t+1;
end
printf("b. t = %d .This generator cannot detect two errors that are four positions apart. The two errors can be anywhere, but if their\ndistance is %d, they remain undetected.\n\n",t,t); // display result
// c) x^7+x^6+1
g=x^7+x^6+1;
printf("c. This is a good choice for this purpose.\n\n"); // display result
// d) x^15+x^14+1
t=32768; // very large to compute
printf("d. This polynomial cannot divide any error of type x^t + 1 if t is less than %d. This means that a codeword with two isolated\nerrors that are next to each other or up to %d bits apart can be detected by this generator.",t,t); // display result
```

Scilab code Exa 10.17 burst error generators

```
clear;
clc;
disp("—————————————Example 10.17——————————————")
// a) x^6+1
r=6;
p1=(1/2)^(r-1); // formula
p2=(1/2)^r; // formula
slip1=round(p1*100);
slip2=round(p2*1000);
// display the result
printf("\na. This generator can detect all burst
   errors with a length less than or equal to %d
   bits; %d out of 100 burst errors with\nlength %d
   will slip by; %d out of 1000 burst errors of
   length %d or more will slip by.\n\n",r,slip1,r+1,
   slip2,r+2);
// b) x^18+x^7x+1
r=18;
p1=(1/2)^(r-1); // formula
p2=(1/2)^r; // formula
slip1=round(p1*10^6);
slip2=round(p2*10^6);
// display the result
printf("b. This generator can detect all burst
   errors with a length less than or equal to %d
   bits; %d out of 1 million burst errors with\
   nlength %d will slip by; %d out of 1 million
   burst errors of length %d or more will slip by.\n
   \n",r,slip1,r+1,slip2,r+2);
// c) x^32+x^23+x^7+1
r=32;
p1=(1/2)^(r-1); // formula
p2=(1/2)^r; // formula
slip1=round(p1*10^10);
slip2=ceil(p2*10^10);
// display the result
printf("c. This generator can detect all burst
   errors with a length less than or equal to %d
```

```
bits; %d out of 10 billion burst errors with\
nlength %d will slip by; %d out of 10 billion
burst errors of length %d or more will slip by.\n
\n",r,slip1,r+1,slip2,r+2);
```

Scilab code Exa 10.18 sum error detection

```
clear;
clc;
disp("-----------------Example 10.18-----------------")
// set of numbers sent
n1=7;
n2=11;
n3=12;
n4=0;
n5=6;
n_sum=n1+n2+n3+n4+n5; // find the sum
printf("\nThe set of numbers is (%d, %d, %d, %d, %d)
    . The sender sends (%d, %d, %d, %d, %d, %d),
    where %d is the sum of the original numbers.\nThe
     receiver adds the five numbers and compares the
    result with the sum.\nIf the two are the same,
    the receiver assumes no error, accepts the five
    numbers, and discards the sum.\nOtherwise, there
    is an error somewhere and the data are not
    accepted.",n1,n2,n3,n4,n5,n1,n2,n3,n4,n5,n_sum,
    n_sum); // display the result
```

Scilab code Exa 10.19 checksum error detection

```
clear;
clc;
disp("-----------------Example 10.19-----------------")
```

```
// set of numbers sent
n1=7;
n2=11;
n3=12;
n4=0;
n5=6;
n_sum=n1+n2+n3+n4+n5; // find the sum
checksum= - n_sum; // formula
printf("\nThe job of the receiver becomes easier if the negative (complement) of the sum, called the checksum is sent along with the numbers.\nIn this case, we send (%d, %d, %d, %d, %d, %d). The receiver can add all the numbers received (including the checksum).\nIf the result is 0, it assumes no error; otherwise, there is an error.",n1,n2,n3,n4,n5,checksum); // display result
```

Scilab code Exa 10.20 21 1s complement

```
clear;
clc;
disp("--------------Example 10.20----------------")
n=21;
// compute one's complement
bin=dec2bin(n);
s=strsplit(bin,1);
a=bin2dec(s(1));
b=bin2dec(s(2));
f=a+b;
complement=dec2bin(f,4); //1's complement
dec_complement=bin2dec(complement); // convert 1's complement to decimal
printf("The number %d in ones complement arithmetic using only four bits is %s or %d.",n,complement,dec_complement); // display result
```

Scilab code Exa 10.21 negative 1s complement

```
clear;
clc;
disp("——————————————Example 10.21——————————————")
n=-6;
// compute 1's complement
d_complement=bitcmp(-n,4);
b_complement=dec2bin(d_complement,4);
printf("The number %d in ones complement arithmetic
   using only four bits is %s or %d.",n,b_complement
   ,d_complement); // display result
```

Scilab code Exa 10.22 complement checksum error

```
clear;
clc;
disp("——————————————Example 10.22——————————————")
// at the sender
n1=7;
n2=11;
n3=12;
n4=0;
n5=6;
s_sum=n1+n2+n3+n4+n5; // find the sum
s_bin=dec2bin(s_sum);
s=strsplit(s_bin,length(s_bin)-4);
a=bin2dec(s(1));
b=bin2dec(s(2));
f=a+b;  // wrapping the sum
s_checksum=bitcmp(f,4); // complementing
```

```
// display the result
printf("The sender initializes the checksum to 0 and adds all data items and the checksum . The result is %d. However, %d cannot\nbe expressed in 4 bits. The extra two bits are wrapped and added with the sum to create the wrapped sum value %d. The sum is then\ncomplemented, resulting in the checksum value %d . The sender now sends six data items to the receiver including the checksum %d.\n\n",s_sum,s_sum,f,s_checksum,s_checksum);

// at the reciever
r_sum=n1+n2+n3+n4+n5+s_checksum; // find the sum including checksum sent by sender
r_bin=dec2bin(r_sum);
r=strsplit(r_bin,length(r_bin)-4);
c=bin2dec(r(1));
d=bin2dec(r(2));
e=c+d; // wrapping the sum
r_checksum=bitcmp(e,4); // complementing
// display the result
printf("The receiver follows the same procedure as the sender. It adds all data items (including the checksum); the result is %d.\nThe sum is wrapped and becomes %d. The wrapped sum is complemented and becomes %d the checksum.\n",r_sum,e,r_checksum);
// check if data is corrupt or not
if(r_checksum==0)
    printf("Since the value of the checksum is 0, this means that the data is not corrupted. The receiver drops the checksum and keeps the other data items.");
else
    printf("The checksum is not zero,hence the entire packet is dropped.");
end
```

Scilab code Exa 10.23 Forouzan text checksum

```
clear;
clc;
disp("---------------Example 10.23---------------")
// sender
text="Forouzan";
// computing the checksum
a=ascii(text);
h1=dec2hex(a(1));
h2=dec2hex(a(2));
h3=dec2hex(a(3));
h4=dec2hex(a(4));
h5=dec2hex(a(5));
h6=dec2hex(a(6));
h7=dec2hex(a(7));
h8=dec2hex(a(8));
// form the hexadecimal words
Fo=h1+h2;
ro=h3+h4;
uz=h5+h6;
an=h7+h8;
d1=hex2dec(Fo);
d2=hex2dec(ro);
d3=hex2dec(uz);
d4=hex2dec(an);
ps=d1+d2+d3+d4;
partial_sum=dec2hex(ps); // partial sum of the words
s=strsplit(partial_sum,[1 4]);
Sum=s(2)+dec2hex(hex2dec(s(1))+hex2dec(s(3))); // wrapping the sum
a=hex2dec(Sum)
c=bitcmp(a,16);
Checksum=dec2hex(c); // Checksum in hex
```

```
carries="1013";
printf("Checksum for a text of 8 characters (Forouzan). The text needs to be divided into 2-byte (16-bit) words.\nWe use ASCII to change each byte to a 2-digit hexadecimal number.\n\n");
// display the process
printf("        a) Checksum at the sender site");
printf("\n          %s           Carries\n",carries);
printf("\n           %s                         Fo",Fo);
printf("\n           %s                         ro",ro);
printf("\n           %s                         uz",uz);
printf("\n           %s                         an",an);
printf("\n           0000          Checksum(initial)");
printf("\n           -----\n");
printf("\n           %s                       Sum(partial)",s(2)+s(3));
printf("\n           ----\n");
printf("\n              %s",s(1));
printf("\n           -----\n");
printf("\n           %s                       Sum",Sum);
printf("\n           %s                  Checksum(to send)",Checksum);

// reciever
Checksum_r=Checksum;
d_Sum=ps+c; // sum of data and checksum
partial_sum=dec2hex(d_Sum);
s=strsplit(partial_sum,[1 4]);
Sum=s(2)+dec2hex(hex2dec(s(1))+hex2dec(s(3))); // wrapping the sum
a=hex2dec(Sum)
```

```
c=bitcmp(a,16);
Checksum=dec2hex(c); // checksum in hex
// display the process
printf("\n\n        b) Checksum at the reciever site");
printf("\n\n          %s                    Carries\n",carries);
printf("\n          %s                    Fo",Fo);
printf("\n          %s                    ro",ro);
printf("\n          %s                    uz",uz);
printf("\n          %s                    an",an);
printf("\n          %s                    Checksum(recieved)",Checksum_r);
printf("\n          -----\n");
printf("\n          %s                    Sum(partial)",s(2)+s(3));
printf("\n          ----\n");
printf("\n             %s",s(1));
printf("\n          -----\n");
printf("\n          %s                    Sum",Sum);
printf("\n          %s000                 Checksum(new)",Checksum);
```

Chapter 11

Data link control

Scilab code Exa 11.1 Simplest protocol

```
clear;
clc;
disp("--------------Example 11.1----------------")
//explain the example
printf("This an example of communication using the simplest protocol. It is very simple. The sender sends a sequence of frames\nwithout even thinking about the receiver. To send three frames, three events occur at the sender site and three events at the receiver site.\nThe data frames are shown by tilted boxes in the figure; the height of the box defines the transmission time difference between the first bit\nand the last bit in the frame.");
// display the figure
clf();
xname("--------------Example 11.1----------------");
xrects([.3 .6;.7 .7;.05 .05;.06 .06]);
xset("font size",3);
xstring(.3,.75,"Sender");
xstring(.6,.75,"Reciever");
```

```
xstring(.32,.65,"A");
xstring(.62,.65,"B");
xstring(.22,.327,"Request");
xstring(.22,.427,"Request");
xstring(.22,.527,"Request");
xstring(.67,.29,"Arrival");
xstring(.67,.39,"Arrival");
xstring(.67,.49,"Arrival");
xstring(.35,.52,"Frame",8);
xstring(.35,.42,"Frame",8);
xstring(.35,.32,"Frame",8);
xarrows([.29 .325],[.55 .55],.3);
xarrows([.29 .325],[.45 .45],.3);
xarrows([.29 .325],[.35 .35],.3);
xarrows([.625 .66],[.5 .5],.3);
xarrows([.625 .66],[.4 .4],.3);
xarrows([.625 .66],[.3 .3],.3);
xset("color",4.9);
xfpoly([.325 .625 .625 .325],[.56 .51 .46 .51]);
xfpoly([.325 .625 .625 .325],[.46 .41 .36 .41]);
xfpoly([.325 .625 .625 .325],[.36 .31 .26 .31]);
xset("color",0);
xset("line style",2);
xarrows([.325 .325],[.64 .14],.3);
xarrows([.625 .625],[.64 .14],.3);
xstring(.3,.1,"Time");
xstring(.6,.1,"Time");
```

Scilab code Exa 11.2 Stop and wait

```
clear;
clc;
disp("−−−−−−−−−−−−−−−Example 11.2−−−−−−−−−−−−−−−−")
```

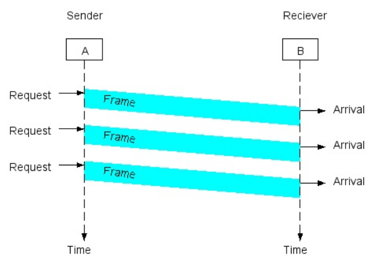

Figure 11.1: Simplest protocol

```
//explain the example
printf("This is an example of communication using
   the Stop and Wait protocol. It is still very
   simple. The sender sends one frame and\nwaits for
    feedback from the receiver. When the ACK arrives
   , the sender sends the next frame. Sending two
   frames in this\nprotocol involves the sender in
   four events and the receiver in two events.");
// display the figure
clf();
xname("---------------Example 11.2-----------------");
xrects([.3 .6;.7 .7;.05 .05;.06 .06]);
xset("font size",3);
xstring(.3,.75,"Sender");
xstring(.6,.75,"Reciever");
xstring(.32,.65,"A");
xstring(.62,.65,"B");
xstring(.35,.52,"Frame",8);
xstring(.35,.3,"Frame",8);
xstring(.58,.41,"ACK",-8);
xstring(.58,.19,"ACK",-8);
xstring(.22,.527,"Request");
```

```
xstring(.22,.3,"Request");
xstring(.24,.38,"Arrival");
xstring(.24,.16,"Arrival");
xstring(.67,.49,"Arrival");
xstring(.67,.27,"Arrival");
xarrows([.29 .325],[.55 .55],.3);
xarrows([.29 .325],[.32 .32],.3);
xarrows([.325 .29],[.39 .39],.3);
xarrows([.325 .29],[.17 .17],.3);
xarrows([.625 .66],[.5 .5],.3);
xarrows([.625 .66],[.28 .28],.3);
xset("color",4.9);
xfpoly([.325 .625 .625 .325],[.56 .51 .46 .51]);
xfpoly([.325 .625 .625 .325],[.34 .29 .24 .29]);
xfpoly([.325 .625 .625 .325],[.41 .46 .41 .36]);
xfpoly([.325 .625 .625 .325],[.19 .24 .19 .14]);
xset("color",0)
xset("line style",2);
xarrows([.325 .325],[.64 .1],.3);
xarrows([.625 .625],[.64 .1],.3);
xstring(.3,.06,"Time");
xstring(.6,.06,"Time");
xset("font size",8);
xstring(.46,.08,".");
xstring(.46,.06,".");
xstring(.46,.1,".");
```

Scilab code Exa 11.3 Stop and wait ARQ

```
clear;
clc;
disp("---------------Example 11.3---------------")
// example explaination
```

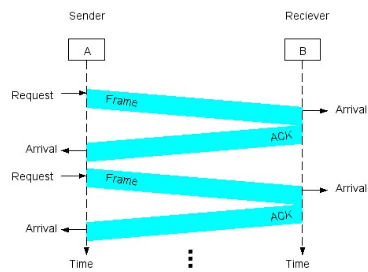

Figure 11.2: Stop and wait

```
printf("This an example of Stop-and-Wait ARQ. The
    series of events taking place are as follows : \n
    \n* Frame 0 is sent and acknowledged.\n* Frame 1
    is lost and resent after the time-out.\n* The
    resent frame 1 is acknowledged and the timer
    stops.\n* Frame 0 is sent and acknowledged, but
    the acknowledgment is lost.\n* The sender has no
    idea if the frame or the acknowledgment is lost,
    so after the time-out, it resends frame 0, which
    is acknowledged.")
```

Scilab code Exa 11.4 Bandwidth delay product

```
clear;
clc;
disp("-------------------Example 11.4-------------------")
bandwidth= 1*10^6; // 1 Mbps
delay = 20*10^-3; // 20 ms
bandwidth_delay_product=bandwidth*delay;
frame_length=1000; // each frame has 1000 bits
```

```
utilization_percentage = (frame_length/
    bandwidth_delay_product)*100; // formula
printf("\nThe bandwidth-delay product is %d. Hence
    the system can send %d bits during the time it
    takes for the data to go from\nthe sender to the
    receiver and then back again.\nThe utilization
    percentage of the link is %d percent.",
    bandwidth_delay_product,bandwidth_delay_product,
    utilization_percentage); // display result
```

Scilab code Exa 11.5 link utilization percentage

```
clear;
clc;
disp("---------------Example 11.5----------------")
bandwidth= 1*10^6; // 1 Mbps
delay = 20*10^-3; // 20 ms
bandwidth_delay_product=bandwidth*delay;
frame_length=1000; // each frame has 1000 bits
num_frames= 15; // The system can send up to 15
    frames during a round trip.
utilization_percentage = (frame_length*num_frames/
    bandwidth_delay_product)*100; // formula
printf("The bandwidth delay product is %d and the
    utilization percentage of the link is %d percent.
    ",bandwidth_delay_product,utilization_percentage)
    ; // display result
```

Scilab code Exa 11.6 Go Back N

```
clear;
clc;
disp("---------------Example 11.6----------------")
```

```
// example explaination
printf("This an example of Go-Back-N. This is an example of a case where the forward channel is reliable, but the reverse is not.\nNo data frames are lost, but some ACKs are delayed and one is lost. The example also shows how cumulative\nacknowledgments can help if acknowledgments are delayed or lost.\n");
printf("\nAfter initialization, there are seven sender events. Request events are triggered by data from the network layer;\narrival events are triggered by acknowledgments from the physical layer. There is no time-out event here because all\noutstanding frames are acknowledged before the timer expires.Although ACK 2 is lost, ACK 3 serves as both ACK 2 and ACK3.\n\nThere are four receiver events, all triggered by the arrival of frames from the physical layer.")
```

Scilab code Exa 11.7 loss of frame

```
clear;
clc;
disp("---------------Example 11.7---------------")
// example explaination
printf("This example shows what happens when a frame is lost. The sequence of events that occur is as follows:\n\n* Frames 0, 1, 2, and 3 are sent. However, frame 1 is lost. The receiver receives frames 2 and 3, but they are discarded because they are received out\nof order (frame 1 is expected).\n\n* The sender receives no acknowledgment about frames 1, 2, or 3. Its timer finally expires. The sender sends all outstanding frames (1, 2, and 3)\nbecause it does
```

```
not know what is wrong. The resending of frames
1, 2, and 3 is the response to one single event.\
n\n* When the sender is responding to this event,
 it cannot accept the triggering of other events.
 This means that when ACK 2 arrives,\nthe sender
is still busy with sending frame 3. The physica1
layer must wait until this event is completed and
 the data link layer goes back\nto its sleeping
state. A vertical line indicates this delay in
the figure.\n\n* It is the same story with ACK 3;
 but when ACK 3 arrives, the sender is busy
responding to ACK 2. It happens\nagain when ACK 4
 arrives. Before the second timer expires, all
outstanding frames have been sent and the timer
is stopped.")
```

Scilab code Exa 11.8 Selective Repeat ARQ

```
clear;
clc;
disp("----------------Example 11.8----------------")
// example explaination
printf("This example shows the behaviour of
   Selective Repeat when a frame is lost.\n\n");
printf("Here, each frame sent or resent needs a
   timer, which means that the timers need to be
   numbered (0, 1,2, and 3).\nThe timer for frame 0
   starts at the first request, but stops when the
   ACK for this frame arrives.\nThe timer for frame
   1 starts at the second request, restarts when a
   NAK arrives, and finally stops when the last ACK
   arrives.\nThe other two timers start when the
   corresponding frames are sent and stop at the
   last arrival event.");
printf("\n\nAt the second arrival, frame 2 arrives
```

```
and is stored and marked (colored slot), but it
cannot be delivered because frame 1 is missing.\
nAt the next arrival, frame 3 arrives and is
marked and stored, but still none of the frames
can be delivered.\nOnly at the last arrival, when
 finally a copy of frame 1 arrives, can frames 1,
 2, and 3 be delivered to the network layer.\
nThere are two conditions for the delivery of
frames to the network layer: First, a set of
consecutive frames must have arrived.\nSecond,
the set starts from the beginning of the window.\
nAfter the first arrival, there was only one
frame and it started from the beginning of the
window. After the last arrival,\nthere are three
frames and the first one starts from the
beginning of the window.");
printf("\n\nA NAK is sent after the second arrival,
   but not after the third,although both situations
   look the same.\nThe reason is that the protocol
   does not want to crowd the network with
   unnecessary NAKs and unnecessary resent frames.\
   nThe second NAK would still be NAK1 to inform the
    sender to resend frame 1 again; this has already
    been done. The first NAK sent is remembered\n(
   using the nakSent variable) and is not sent again
    until the frame slides. A NAK is sent once for
   each window position and defines\nthe first slot
   in the window.");
printf("\n\nOnly two ACKs are sent here. The first
   one acknowledges only the first frame; the second
    one acknowledges three frames.\nIn Selective
   Repeat, ACKs are sent when data are delivered to
   the network layer. If the data belonging to n
   frames are delivered in one shot,\nonly one ACK
   is sent for all of them.")
```

Scilab code Exa 11.9 Connection and disconnection

```
clear;
clc;
disp("---------------Example 11.9---------------")
// example explaination
printf("This example shows how U-frames can be used for connection establishment and connection release.\n\n* Node A asks for a connection with a set asynchronous balanced mode (SABM) frame; node B gives a positive response with\n an unnumbered acknowledgment (UA) frame.\n* After these two exchanges, data can be transferred between the two nodes (not shown in the figure).\n* After data transfer, node A sends a DISC (disconnect) frame to release the connection; it is confirmed by node B\n responding with a UA (unnumbered acknowledgment).");
```

Scilab code Exa 11.10 Piggybacking without Error

```
clear;
clc;
disp("---------------Example 11.10---------------")
// example explaination
printf("This example shows an exchange using piggybacking. The sequence of events that occur are as follows:\n\n* Node A begins the exchange of information with an I-frame numbered 0 followed by another I-frame numbered 1.\n\n* Node B piggybacks its acknowledgment of both frames onto an I-frame of its own.\n Node Bs first I-
```

```
frame is also numbered 0 [N(S) field] and
contains a 2 in its N(R) field, acknowledging the
 receipt of As\n frames 1 and 0 and indicating
that it expects frame 2 to arrive next.\n\n* Node
 B transmits its second and third I-frames (
numbered 1 and 2) before accepting further frames
 from node A.\n Its N(R) information, therefore,
 has not changed: B frames 1 and 2 indicate that
node B is still expecting As frame 2 to arrive
next.\n\n* Node A has sent all its data.
Therefore, it cannot piggyback an acknowledgment
onto an I-frame and sends an S-frame instead.\n
The RR code indicates that A is still ready to
receive. The number 3 in the N(R) field tells B
that frames 0, 1, and 2 have all been\n accepted
 and that A is now expecting frame number 3.")
```

Scilab code Exa 11.11 Piggybacking with Error

```
clear;
clc;
disp("---------------Example 11.11----------------")
// example explaination
printf("This example shows an exchange in which a
   frame is lost. The sequence of events that occur
   is as follows :\n\n* Node B sends three data
   frames (0, 1,and 2), but frame 1 is lost.\n\n*
   When node A receives frame 2, it discards it and
   sends a REJ frame for frame 1 since the protocol
   being used is Go-Back-N\nwith the special use of
   an REJ frame as a NAK frame.\n\n* The NAK frame
   does two things here: It confirms the receipt of
   frame 0 and declares that frame 1 and any
   following frames must be resent.\n\n* Node B,
   after receiving the REJ frame, resends frames 1
```

```
and 2.\n\n* Node A acknowledges the receipt by
    sending an RR frame (ACK) with acknowledgment
    number 3.");
```

Scilab code Exa 11.12 Network layer packet

```
clear;
clc;
disp("——————————————Example 11.12——————————————")
// example explaination
printf("This example shows the steps and the phases
    followed by a network layer packet as it is
    transmitted through a PPP connection.\nFor
    simplicity, unidirectional movement of data from
    the user site to the system site is assumed (such
     as sending an e-mail through an ISP).\n\n");
printf("The first two frames show link establishment
    . Two options are chosen(not shown in the figure)
    : using PAP for authentication and\nsuppressing
    the address control fields. Frames 3 and 4 are
    for authentication. Frames 5 and 6 establish the
    network layer connection using IPCP.\n\n");
printf("The next several frames show that some IP
    packets are encapsulated in the PPP frame. The
    system (receiver) may have been running\nseveral
    network layer protocols, but it knows that the
    incoming data must be delivered to the IP
    protocol because the NCP protocol\nused before
    the data transfer was IPCP.\n\n");
printf("After data transfer, the user then
    terminates the data link connection, which is
    acknowledged by the system.\nOf course the user
    or the system could have chosen to terminate the
    network layer IPCP and keep the data link layer
    running if it\nwanted to run another NCP protocol
```

.”)

Chapter 12

Multiple Access

Scilab code Exa 12.1 ALOHA TB calculation

```
clear;
clc;
disp("--------------Example 12.1--------------")
d=600*10^3; // 600 km
speed = 3*10^8; // 3*10^8 m/s
Tp=(d/speed)*10^3; // propagation time

// a) K=1
K=[0 1]; // range
TB1=Tp*K(1);
TB2=Tp*K(2);
printf("\na)K=1 :-  TB is either  %d ms (0 x 2) or
   %d ms (1 x 2), based on the outcome of the random
    variable.\n",TB1,TB2); // display result
// b) K=2
K=[0 1 2 3]; // range
TB1=Tp*K(1);
TB2=Tp*K(2);
TB3=Tp*K(3);
TB4=Tp*K(4);
printf("\nb)K=2 :-  TB can be %d, %d, %d, or %d ms,
```

```
    based on the outcome of the random variable.\n",
    TB1,TB2,TB3,TB4); // display result
// c) K=3
K=[0 1 2 3 4 5 6 7]; //range
TB1=Tp*K(1);
TB2=Tp*K(2);
TB3=Tp*K(3);
TB4=Tp*K(4);
TB5=Tp*K(5);
TB6=Tp*K(6);
TB7=Tp*K(7);
TB8=Tp*K(8);
printf("\nc)K=3 :-  TB can be %d, %d, %d, %d, %d, %d
    , %d or %d ms, based on the outcome of the random
     variable.\n",TB1,TB2,TB3,TB4,TB5,TB6,TB7,TB8);
    // display result
// d) K >10
printf("\nd)K>10 :-  If K > 10, it is normally set
    to 10.") // display result
```

Scilab code Exa 12.2 Collision free ALOHA

```
clear;
clc;
disp("---------------Example 12.2---------------")
frame_bits=200;
datarate=200*10^3; // 200 kbps
Tfr=frame_bits/datarate;
Tv=2*Tfr; // vulnerable time
printf("The Tfr is %d ms and the vulnerable time is
    %d ms.\nThis means no station should send later
    than %d ms before this station starts
    transmission\nand no station should start sending
     during the one %d ms period that this station is
     sending.",Tfr*10^3,Tv*10^3,Tfr*10^3,Tfr*10^3);
```

```
// display results with appropriate units
```

Scilab code Exa 12.3 Pure ALOHA throughput

```
clear;
clc;
disp("——————————————Example 12.3———————————————")
frame_bits=200;
datarate=200*10^3; // 200 kbps
Tfr=frame_bits/datarate;
printf("\nThe frame transmission time is %d ms.\n",
   Tfr*10^3);

function[S]=s_func (frame_rate) // function to
   determine S
    G=frame_rate*10^-3; // load
    S=G*(%e^(-2*G)); // formula
    percent=S*100;
    printf("S = %4.3f or %3.1f percent.",S,percent);
endfunction

//a. 1000 frames per second
frame_rate=1000;
printf("\na)");
S=s_func(frame_rate); //calling the function
throughput=S*frame_rate;
printf("\nThe throughput is %d frames.Only %d frames
    out of %d will probably survive.\n",throughput,
   throughput,frame_rate); // display result
//b. 500 frames per second
frame_rate=500;
printf("\nb)");
S=s_func(frame_rate); //calling the function
throughput=rat(S,10^-2)*frame_rate; // approximation
// display result
```

```
printf("\nThe throughput is %d frames.Only %d frames out of %d will probably survive.\n",throughput,throughput,frame_rate);
printf("Note that this is the maximum throughput case, percentage wise.\n");

//c. 250 frames per second
frame_rate=250;
printf("\nc)");
S=s_func(frame_rate); //calling the function
throughput=rat(S,10^-1.5)*frame_rate; //approximation
printf("\nThe throughput is %d frames.Only %d frames out of %d will probably survive.\n",throughput,throughput,frame_rate); // display result
```

Scilab code Exa 12.4 Slotted ALOHA throughput

```
clear;
clc;
disp("---------------Example 12.4---------------")
frame_bits=200;
datarate=200*10^3; // 200 kbps
Tfr=frame_bits/datarate;
printf("\nThe frame transmission time is %d ms.\n",Tfr*10^3);

function[S]=s_func (frame_rate) // function to determine S
    G=frame_rate*10^-3; // load
    S=G*(%e^(-G)); //formula
    percent=S*100;
    printf("S = %4.3f or %3.1f percent.",S,percent);
endfunction

```

```
//a. 1000 frames per second
frame_rate=1000;
printf("\na)");
S=s_func(frame_rate); //calling the function
throughput=rat(S,10^-2.5)*frame_rate;//approximation
// display result
printf("\nThe throughput is %d frames.Only %d frames out of %d will probably survive.\n",throughput,throughput,frame_rate);
printf("Note that this is the maximum throughput case, percentage wise.\n");

//b. 500 frames per second
frame_rate=500;
printf("\nb)");
S=s_func(frame_rate); //calling the function
throughput=S*frame_rate;
printf("\nThe throughput is %d frames.Only %d frames out of %d will probably survive.\n",throughput,throughput,frame_rate); // display result

//c. 250 frames per second
frame_rate=250;
printf("\nc)");
S=s_func(frame_rate); //calling the function
throughput=S*frame_rate;
printf("\nThe throughput is %d frames.Only %d frames out of %d will probably survive.\n",round(throughput),round(throughput),frame_rate);//display result
```

Scilab code Exa 12.5 Minimum frame size

```
clear;
```

```
clc;
disp("----------------Example 12.5----------------")
bandwidth = 10*10^6; // 10 Mbps
Tp=25.6*10^-6; // 25.6 microseconds
Tfr=2*Tp; // formula
min_frame_size = bandwidth*Tfr; // formula
bytes=min_frame_size/8; // 1 byte = 8 bits
printf("The minimum frame size is %d bits or %d bytes.\nThis is the minimum size of the frame for Standard Ethernet.",min_frame_size,bytes); // display result
```

Scilab code Exa 12.6 Chips for network

```
clear;
clc;
disp("----------------Example 12.6----------------")
// use the rows of W2 and W4 in the solution
W2=[1 1;1 -1];
W4=[1 1 1 1;1 -1 1 -1;1 1 -1 -1;1 -1 -1 1];
//a. Two stations
C1= W2(1,:); //select 1st row of W2
C2= W2(2,:); // select 2nd row of W2
// display result
disp("a)The chips for a two-station network are ");
disp(C1)
disp("and")
disp(C2)

//b. Four stations
C1= W4(1,:); // select 1st row of W4
C2= W4(2,:); // select 2nd row of W4
C3= W4(3,:); // select 3rd row of W4
C4= W4(4,:); // select 4th row of W4
// display result
```

```
disp("b)The chips for a four-station network are ");
disp(C1)
printf(",")
disp(C2)
printf(",")
disp(C3)
printf("and")
disp(C4)
```

Scilab code Exa 12.7 Number of sequences

```
clear;
clc;
disp("---------------Example 12.7---------------")
stations=90;
m=7;
N=2^m; // formula
printf("The number of sequences is %d. Hence %d of
   the sequences can be used as the chips.",N,
   stations); // display result
```

Scilab code Exa 12.8 Proof

```
clear;
clc;
disp("---------------Example 12.8---------------")
//Proof
printf("Proof:-\n Let us prove this for the first
   station, using the previous four-station example
   .\n The data on the channel is D = (d1*c1 + d2*c2
    + d3*c3 + d4*c4) .\n The receiver which wants to
    get the data sent by station 1 multiplies these
   data by c1.\n  D*c1 = (d1*c1+d2*c2+d3*c3+d4*c4)*
```

```
c1\n          = d1*c1*c1 + d2*c2*c1 + d3*c3*c1 + d4*
c4*c1\n          = d1*N + d2*0 + d3*0 + d4*0\n
      = d1*N\n When the result is divided by N,
we get d1. Hence Proved.");
```

Chapter 13

Wired LANs Ethernet

Scilab code Exa 13.1 Define the type

```
clear;
clc;
disp("—————————————Example 13.1——————————————")

// function to check if thhe 2nd hex digit from the
   left is even or odd
function []=check (a)

    s=strsplit(a,[1,2]); // extract the 2nd hex
       digit from left
    d= hex2dec(s(2));
    bin=dec2bin(d,4); // convert to binary
    bits=strsplit(bin,3); // least significant bit
    lb=bits(2);

    if(lb=='0') // check if even or odd
        printf("This is a unicast address because
           the second hexadecimal digit from the
           left i.e %s in binary is %s and is even.\
           n",s(2),bin);
    else
```

```
        printf("This is a multicast address because
            the second hexadecimal digit from the
            left i.e %s in binary is %s and is odd.\n
            ",s(2),bin);
    end
endfunction

// a) 4A:30:10:21:1O:1A
a="4A:30:10:21:1O:1A";
printf("\na)");
check(a); // calling the function

// b) 47:20:1B:2E:08:EE
b="47:20:1B:2E:08:EE";
printf("\nb)");
check(b); // calling the function

// c) FF:FF:FF:FF:FF:FF
c="FF:FF:FF:FF:FF:FF";
s = strsplit(c,":",6); // split into 2 hex digits
for i=1:6
    if(s(i)=="FF") // check if equal to FF
        continue;
    else
        break;
    end

end
if(i==6)
    printf("\nc)This is a broadcast address because
        all digits are Fs.") //print the result
end
```

Scilab code Exa 13.2 Sending the address

```
clear;
clc;
disp("—————————————Example 13.2——————————————")
// address = 47:20:1B:2E:08:EE
address = "47:20:1B:2E:08:EE";

function[bin_str]=bin_address (address) // function to convert address in hexadecimal to binary
    b=strsplit(address);
    bin_str="";
    for i=1:length(address)
        if(modulo(i,3)==0)  // to exclude ":"
            continue;
        else
            d=hex2dec(b(i));
            bin=dec2bin(d,4);
            bin_str=bin_str+bin;  // address in binary
        end

    end
endfunction
bin_str=bin_address(address);

function [addr] = revstr(bin_str) // function to reverse the nibbles in the address
    str_nibble=strsplit(bin_str,[4 8 12 16 20 24 28 32 36 40 44 ]);
    addr="";
    for i=1:12
        rev=strrev(str_nibble(i));
        addr=addr+rev; // resultant string
    end
endfunction
addr=revstr(bin_str);
bytes=strsplit(addr,[8 16 24 32 40]);  // spilt into bytes

```

```
function [bytes]=exchgnib(bytes) // function to
   exchange the nibbles in each byte
    for i=1:6
        nib=strsplit(bytes(i),4);

        temp=nib(1); // exachanging nibbles
        nib(1)=nib(2);
        nib(2)=temp;
        bytes(i)=nib(1)+nib(2);
    end
endfunction
bytes=exchgnib(bytes); // final address
printf("\nThe address is sent left-to-right, byte by
    byte; for each byte, it is sent right-to-left,
   bit by bit, as:\n\n       %s %s %s %s %s %s",
   bytes(1),bytes(2),bytes(3),bytes(4),bytes(5),
   bytes(6));
```

Chapter 16

Wireless WANs Cellular Telephone and Satellite Networks

Scilab code Exa 16.1 Period of moon

```
clear;
clc;
disp("--------------Example 16.1---------------")
C=1/100;
dist_moon=384000; // 384,000 km
radius_earth = 6378; // 6378 km
distance=dist_moon+radius_earth ;//  total distance
   in km
Period=C*((distance)^1.5); //formula
month=round(Period/2592000); // 1 month =
   60*60*24*30=2592000 seconds
printf("The period of the Moon, according to Keplers
    law is %d s or approximately %d month.",floor(
   Period),month);
```

Scilab code Exa 16.2 Period of geostationary satellite

```
clear;
clc;
disp("———————————————Example 16.2———————————————")
C=1/100;
orbit=35786; // 35,786 km
radius_earth = 6378; // 6378 km
distance=orbit+radius_earth ;//  total distance in
   km
Period=C*((distance)^1.5); //formula
hour=round(Period/3600); // 1 hour = 60*60=3600
   seconds
printf("According to Keplers law, the period of the
   satellite is %d s or %d hours.",floor(Period),
   hour);
printf("\nThis means that a satellite located at %d
   km has a period of %d h, which is the same as the
    rotation period of the Earth.\nA satellite like
   this is said to be stationary to the Earth. The
   orbit is called a geosynchronous orbit.",orbit,
   hour);
```

Chapter 17

SONET SDH

Scilab code Exa 17.1 Datarate of STS1

```
clear;
clc;
disp("---------------Example 17.1---------------")
frame_rate=8000; // frames per sec
frame=9*(1*90); // each frame is made of 9 by (1x90)
    bytes
bits=8; // 1byte = 8 bits
STS1_data_rate=frame_rate*frame*bits; // formula
printf("The STS-1 data rate is %5.3f Mbps.",
   STS1_data_rate*10^-6); // display result with
   appropriate unit
```

Scilab code Exa 17.2 Datarate of STS3

```
clear;
clc;
disp("---------------Example 17.2---------------")
frame_rate=8000; // frames per sec
```

```
frame=9*(3*90); // each frame is made of 9 by (3x90)
    bytes
bits=8; // 1byte = 8 bits
STS3_data_rate=frame_rate*frame*bits; // formula
printf("The STS−3 data rate is %5.2f Mbps.",
   STS3_data_rate*10^-6); // display result with
   appropriate unit
```

Scilab code Exa 17.3 Duration of STSs

```
clear;
clc;
disp("−−−−−−−−−−−−−−−Example 17.3−−−−−−−−−−−−−−−−")
// for all STS−1 , STS−3 and STS−n frame rate is
   same hence frame duration is same
frame_rate=8000; // frames per sec
frame_duration=1/frame_rate;
printf("In SONET, %d frames are sent per second.
   This means that the duration of an STS−1, STS−3,
   or STS−n frame is the same and equal %d
   microseconds.",frame_rate,frame_duration*10^6);
   // display result
```

Scilab code Exa 17.4 User datarate STS1

```
clear;
clc;
disp("−−−−−−−−−−−−−−−Example 17.4−−−−−−−−−−−−−−−−")
frame_rate=8000; // frames per sec
//each frame is made of 9 rows and 86 columns
rows=9;
columns=(1*86);
bits=8; // 1byte = 8 bits
```

```
STS1_user_data_rate=frame_rate*rows*columns*bits; //
    formula
printf("The STS−1 user data rate is %5.3f Mbps.",
   STS1_user_data_rate*10^-6); // display result
   with appropriate unit
```

Scilab code Exa 17.5 H1 and H2

```
clear;
clc;
disp("−−−−−−−−−−−−−−−Example 17.5−−−−−−−−−−−−−−−−")
byte_number=650;
temp=dec2hex(byte_number);// convert to hexadecimal
hex='0'+temp;
h=strsplit(hex,2); // split the hexadecimal number
   into 2
H1=h(1);
H2=h(2);
printf("The number %d can be expressed in four
   hexadecimal digits as Ox%s.\nHence the value of
   H1 is Ox%s and the value of H2 is Ox%s.",
   byte_number,hex,H1,H2); // display result
```

Chapter 19

Network layer Logical addressing

Scilab code Exa 19.1 Dotted decimal notation

```
clear;
clc;
disp("——————————Example 19.1——————————")
// a) 10000001 00001011 00001011 11101111
a="10000001000010110000101111101111"
ab=strsplit(a,[8 16 24 ])  //separate the bytes
a3=bin2dec(ab(1));          // convert binary numbers
    to decimal numbers
a2=bin2dec(ab(2));
a1=bin2dec(ab(3));
a0=bin2dec(ab(4));
printf("\na) Decimal notation :-  %d.%d.%d.%d",a3,a2
   ,a1,a0);        //result in decimal notation
// b) 11000001 10000011 00011011 11111111
b="11000001100000110001101111111111"
bb=strsplit(b,[8 16 24 ])        //separate the
   bytes
b3=bin2dec(bb(1));             //convert binary
   numbers into decimal numbers
```

```
b2=bin2dec(bb(2));
b1=bin2dec(bb(3));
b0=bin2dec(bb(4));
printf("\n\nb) Decimal notation :-  %d.%d.%d.%d",b3,
   b2,b1,b0);   //result in decimal nootation
```

Scilab code Exa 19.2 Binary notation

```
clear;
clc;
disp("—————————————Example 19.2—————————————")
// a) 111.56.45.78
n=8;    //number of bits i.e 1 byte
a3=dec2bin(111,n);          // convert decimal
   numbers to binary numbers
a2=dec2bin(56,n);
a1=dec2bin(45,n);
a0=dec2bin(78,n);
disp("a) Binary notation :- "+a3+" "+a2+" "+a1+" "+
   a0)   //result in binary notation
// b) 221.34.7.82
b3=dec2bin(221,n);          //convert decimal
   numbers into binary numbers
b2=dec2bin(34,n);
b1=dec2bin(7,n);
b0=dec2bin(82,n);
disp("b) Binary notation :- "+b3+" "+b2+" "+b1+" "+
   b0)  //result in binary nootation
```

Scilab code Exa 19.3 Find the error

```
clear;
clc;
```

```
disp("---------------Example 19.3----------------")
// a) 111.56.045.78
disp("a) There must be no leading zero (045).") //display the result
//b) 221.34.7.8.20
disp("b) There can be no more than four numbers in an IPv4 address.")    //display the result
//c) 75.45.301.14
disp("c) Each number needs to be less than or equal to 255 (301 is outside this range).") //display the result
//d) 11100010.23.14.67
disp("d) A mixture of binary notation and dotted-decimal notation is not allowed.")        //display the result
```

Scilab code Exa 19.4 Find the class

```
clear;
clc;
disp("---------------Example 19.4----------------")
function []= binclass (q)        //function to determine the class of a address in binary notation
    c=strsplit(q,1);
if(c(1)=="0")
     disp(" The first bit is 0. This is a class A address.")
else
    c=strsplit(q,2);
    if(c(1)=="10")
        disp(" The first 2 bits are 10. This is a class B address.")
    else
        c=strsplit(q,3);
```

```
        if(c(1)=="110")
            disp(" The first 3 bits are 110. This is a class C address.")
        else
            c=strsplit(q,4);
            if(c(1)=="1110")
                disp(" The first 4 bits are 1110. This is a class D address.")
            elseif(c(1)=="1111")
                disp(" The first 4 bits are 1111. This is a class E address.")
end
end
end
end
endfunction  //end of function

function [] = byteclass (q)           //function to determine the class of a address in decimal notation
if(q>=0 & q<= 127)
    disp(" The first byte is between 0 and 127; the class is A.")
elseif(q>=128 & q<=191)
    disp(" The first byte is between 128 and 191; the class is B.")
elseif( q>=192 & q<=223)
    disp(" The first byte is between 192 and 223; the class is C.")
elseif( q>=224 & q<=239)
    disp(" The first byte is between 224 and 239; the class is D.")
elseif(q>=240 & q<=255)
    disp(" The first byte is between 240 and 255; the class is E.")
end
endfunction     //end of function

```

```

//a) 00000001 00001011 00001011 11101111
q="00000001";
printf("\na)");
binclass(q);  //calling the function
//b) 11000001 10000011 00011011 11111111
q="11000001";
printf("\nb)");
binclass(q);  //calling the function
//c) 14.23.120.8
q=14;
printf("\nc)");
byteclass(q);  //calling the function
//d) 252.5.15.111
q=252;
printf("\nd)");
byteclass(q);  //calling the function
```

Scilab code Exa 19.5 block of addresses

```
clear;
clc;
disp("————————————Example 19.5————————————")
n=16;   //cidr
fa1=205;  //bytes of 1st address in decimal
fa2=16;
fa3=37;
fa4=32;
fab1=dec2bin(fa1,8); //convert the bytes to binary
fab2=dec2bin(fa2,8);
fab3=dec2bin(fa3,8);
fab4=dec2bin(fa4,8);
la4=fa4+n-1;  //determine the last byte of the last
    address
lab4=dec2bin(la4,8); //last address in binary
```

```
disp("The block of addresses is");  //display the
    results
disp("i) In binary notation :- 1st address = "+fab1+
    " "+fab2+" "+fab3+" "+fab4)
disp("  last address = "+fab1+" "+fab2+" "+fab3+" "+
    lab4)
printf("\nii) In dotted decimal notation :- 1st
    address = %d.%d.%d.%d\n\n    last address = %d.%d
    .%d.%d",fa1,fa2,fa3,fa4,fa1,fa2,fa3,la4);
```

Scilab code Exa 19.6 Find first address

```
clear;
clc;
disp("--------------Example 19.6--------------")
//address :- 205.16.37.39/28
n=8;     //number of bits i.e 1 byte
a3=dec2bin(205,n);          // convert decimal
    numbers to binary numbers
a2=dec2bin(16,n);
a1=dec2bin(37,n);
a0=dec2bin(39,n);
disp(" Binary notation of the address is "+a3+" "+a2
    +" "+a1+" "+a0)
mask= 28;
num_of_zeros=32-mask; // calculate the number of
    bits to be set to zero
a=a3+a2+a1+a0;
p1=strsplit(a,mask); //truncate the binary address
p=p1(1);
for i= 1:num_of_zeros
    p=p+'0';        //appending zeros
end
b=strsplit(p,[8 16 24 ])  //separate the bytes
b3=bin2dec(b(1));          // convert binary numbers
```

```
    to decimal numbers
b2=bin2dec(b(2));
b1=bin2dec(b(3));
b0=bin2dec(b(4));
disp(" Binary notation of first address:-  "+b(1)+"
    "+b(2)+" "+b(3)+" "+b(4))
printf(" Decimal notation of first address:-  %d.%d.
    %d.%d",b3,b2,b1,b0);        //result in decimal
    notation
```

Scilab code Exa 19.7 Find last address

```
clear;
clc;
disp("----------------Example 19.7----------------")
//address :- 205.16.37.39/28
n=8;    //number of bits i.e 1 byte
a3=dec2bin(205,n);          // convert decimal
    numbers to binary numbers
a2=dec2bin(16,n);
a1=dec2bin(37,n);
a0=dec2bin(39,n);
disp(" Binary notation of the address is "+a3+" "+a2
    +" "+a1+" "+a0)
mask= 28;
num_of_ones=32-mask; // calculate the number of bits
     to be set to one
a=a3+a2+a1+a0;
p1=strsplit(a,mask); //truncate the binary address
p=p1(1);
for i= 1:num_of_ones
    p=p+'1';        //appending ones
end
b=strsplit(p,[8 16 24 ])  //separate the bytes
b3=bin2dec(b(1));         // convert binary numbers
```

```
to decimal numbers
b2=bin2dec(b(2));
b1=bin2dec(b(3));
b0=bin2dec(b(4));
disp(" Binary notation of last address:-  "+b(1)+" "+b(2)+" "+b(3)+" "+b(4))
printf(" Decimal notation of last address:-  %d.%d.%d.%d",b3,b2,b1,b0);        //result in decimal notation
```

Scilab code Exa 19.8 number of addresses

```
clear;
clc;
disp("———————————————Example 19.8———————————————")
//address :- 205.16.37.39/28
n=28;    //cidr
exp_of_2 = 32-n;
num_of_addresses= 2^(exp_of_2); //formula to calculate number of addresses
printf("\n The number of addresses in the block is %d.",num_of_addresses); //display the results
```

Scilab code Exa 19.9 addresses using mask

```
clear;
clc;
disp("———————————————Example 19.9———————————————")
//address :- 205.16.37.39/28
n=28;
by=8;     //number of bits i.e 1 byte
a3=dec2bin(205,by);          // convert decimal numbers to binary numbers
```

```
a2=dec2bin(16,by);
a1=dec2bin(37,by);
a0=dec2bin(39,by);
disp("Binary notation of the address is "+a3+" "+a2+" "+a1+" "+a0)//display address in binary notation
a=a3+a2+a1+a0;
mask="";
for i = 1:n
    mask=mask+'1'; //adding 1s to the mask
end
m=32-n;
for i = 1:m
    mask=mask+'0'; //adding 0s to the mask
end
ma=strsplit(mask,[8 16 24]);
disp("The mask is "+ma(1)+" "+ma(2)+" "+ma(3)+" "+ma(4)) //display the mask
//a) first address
x=strsplit(a);
y=strsplit(mask);
for i= 1:32
    fa0(i)=bitand(strtod(x(i)),strtod(y(i)));  //perform 'and' of address and mask to get 1st address
end
fa1=string(fa0(1:8));
fa2=string(fa0(9:16));
fa3=string(fa0(17:24));
fa4=string(fa0(25:32));
printf("\na) The first address is "); //display result
printf("%s",fa1);
printf(" ");
printf("%s",fa2);
printf(" ");
printf("%s",fa3);
printf(" ");
```

```
printf("%s",fa4);

//b) last address
for i=1:32
    cp0(i)=bitcmp(strtod(y(i)),1);  //find complement of the mask
end
for i=1:32
    la0(i)=bitor(strtod(x(i)),cp0(i)); //perform 'or' of address and complement of the mask
end
cp1=string(cp0(1:8));
cp2=string(cp0(9:16));
cp3=string(cp0(17:24));
cp4=string(cp0(25:32));
printf("\n\nb) The complement of the mask is ");
printf("%s",cp1);
printf(" ");
printf("%s",cp2);
printf(" ");
printf("%s",cp3);
printf(" ");
printf("%s",cp4);

la1=string(la0(1:8));
la2=string(la0(9:16));
la3=string(la0(17:24));
la4=string(la0(25:32));
printf("\n\n  The last address is ");  //display the result
printf("%s",la1);
printf(" ");
printf("%s",la2);
printf(" ");
printf("%s",la3);
printf(" ");
printf("%s",la4);

```

```
// c) number of addresses
cp="";
for i=1:32
    cp=cp+string(cp0(i));
end
dec=bin2dec(cp);
num_of_addresses=dec+1;  //  formula to find number
   of addresses
printf("\n\nc) The number of addresses is %d.",
   num_of_addresses); // display the result
```

Scilab code Exa 19.10 design sub blocks

```
clear;
clc;
disp("————————————Example 19.10————————————")
// address :- 190.100.0.0/16 i.e 65,536 addresses
num_of_ISP_addresses=65536; //total number of
   addresses
printf('\n');
function [total]= addresses (num_of_customers,
   num_of_addresses) //function to find total number
    of addresses allocated to a group
    total=num_of_customers*num_of_addresses; //
       formula to calculate total number of
       addresses
    bits=log2(num_of_customers);
    n=32-bits;
     printf("Number of bits needed to define each
        host = %d",bits); //display results
    printf("\nThe prefix length = %d",n);
    endfunction
//group 1
g1=addresses(64,256); //calling fuction
printf("\nThe total number of addresses alloted to
```

```
    Group 1 = %d\n\n",g1);
//group 2
g2=addresses(128,128); //calling function
printf("\nThe total number of addresses alloted to
    Group 2 = %d\n\n",g2);
//group 3
g3=addresses(128,64); //calling function
printf("\nThe total number of addresses alloted to
    Group 3 = %d\n\n",g3);
num_allocated=g1+g2+g3; //total number of addresses
    allocated
num_remaining_addresses=num_of_ISP_addresses-
    num_allocated; // formula to calculate number of
    remaining addresses
printf("\nThe total number of addresses granted to
    the ISP = %d",num_of_ISP_addresses);
printf("\n\nThe total number of addresses allocated
    by the ISP = %d",num_allocated);
printf("\n\nThe number of addresses remaining = %d",
    num_remaining_addresses); //display result
```

Scilab code Exa 19.11 find original address

```
clear;
clc;
disp("————————————Example 19.11————————————")
//address :- 0:15::1:12:1213
a1=0;
a2=15;
a8=1213;
a7=12;
a6=1;
n1=strsplit(string(a1));
n2=strsplit(string(a2));
n8=strsplit(string(a8));
```

```
n7=strsplit(string(a7));
n6=strsplit(string(a6));
function [n]=org_address (num,nmatrix) //function to
    form the 2 bytes in hexadecimal
    n="";
    for i=1:4-length(string(num))
        n=n+'0';
    end
    for i=1:length(string(num))
        n=n+nmatrix(i);
    end
endfunction
f1=org_address(a1,n1); //2 byte addresses
f2=org_address(a2,n2);
f8=org_address(a8,n8);
f7=org_address(a7,n7);
f6=org_address(a6,n6);
f3="0000";  // zeros
f4=f3;
f5=f3;
printf("\nThe original address is %s:%s:%s:%s:%s:%s:
   %s:%s",f1,f2,f3,f4,f5,f6,f7,f8);  //display the
   original addresses
```

Chapter 20

Network Layer Internet Protocol

Scilab code Exa 20.1 Acceptance of packet

```
clear;
clc;
disp("--------------Example 20.1--------------")
// 01000010 - first 8 bits of IP4 packet
p="01000010";
s=strsplit(p,4); // split into two
v=bin2dec(s(1)); // version
d=bin2dec(s(2)); // header length
bytes=d*4; // formula
if(((bytes > = 20 )&((v == 4) |(v == 6)))) //minimum number of bytes is 20 and version should be IP4 or IP6
    printf("The packet is accepted .");
else
    printf("There is an error in this packet. The 4 leftmost bits %s show the version , which is correct.\nThe next 4 bits %s show an invalid header length %d. The minimum number of bytes in the header must be 20.\nThe packet has
```

```
been corrupted in transmission.",s(1),s(2),bytes); // display result
```

Scilab code Exa 20.2 Bytes of options

```
clear;
clc;
disp("–––––––––––––––Example 20.2–––––––––––––––––")
HLEN="1000";
d=bin2dec(HLEN); //convert to decimal
bytes=d*4; // formula
base_header=20;
options=bytes-base_header; // formula
printf("The total number of bytes in the header is %d bytes.The first %d bytes are the base header, the next %d bytes are the options.",bytes,base_header,options); // display result
```

Scilab code Exa 20.3 Bytes of data

```
clear;
clc;
disp("–––––––––––––––Example 20.3–––––––––––––––––")
HLEN=5;
total_length="0028"; // 0x0028 in hexadecimal
total_bytes=hex2dec(total_length); // get length in decimal or total bytes
header_bytes = HLEN*4; // number of header bytes
data_bytes=total_bytes-header_bytes; // formula
printf(" The total length is %d bytes. The total number of bytes in the header is %d bytes, which means the packet is carrying %d bytes of data.",
```

```
total_bytes,header_bytes,data_bytes);  // display result
```

Scilab code Exa 20.4 Time and protocol

```
clear;
clc;
disp("--------------Example 20.4---------------")
// packet = 0x4500002800010000102
packet="4500002800010000102";
bytes=strsplit(packet,[2 4 6 8 10 12 14 16 18 ]); // split the packet into bytes
time_to_live=hex2dec(bytes(9));
printf("The time-to-live field is the ninth byte, which is %s. Hence the packet can travel %d hop.\n",bytes(9),time_to_live);
protocol_field=hex2dec(bytes(10));
select protocol_field  // display the result according to the protocol
case 1
    printf("The protocol field is the next byte i.e %s, which means that the upper-layer protocol is ICMP.",bytes(10));
case 2
    printf("The protocol field is the next byte i.e %s, which means that the upper-layer protocol is IGMP.",bytes(10));
case 6
    printf("The protocol field is the next byte i.e %s, which means that the upper-layer protocol is TCP.",bytes(10));
case 17
    printf("The protocol field is the next byte i.e %s, which means that the upper-layer protocol is UDP.",bytes(10));
```

```
case 89
    printf("The protocol field is the next byte i.e
       %s, which means that the upper-layer protocol
        is OSPF".,bytes(10));
end
```

Scilab code Exa 20.5 M equals 0

```
clear;
clc;
disp("---------------Example 20.5---------------")
M_bit = 0;
if(M_bit==0) // display the result according to the
   value of the M bit
    printf("There are no more fragments; this
       fragment is the last one.");
else
    printf("There are more fragments; this fragment
       is not the last one.");
end
printf("\nHowever, it cannot be determined if the
   original packet was fragmented or not. A non-
   fragmented packet is considered the last fragment
   .");
```

Scilab code Exa 20.6 M equals 1

```
clear;
clc;
disp("---------------Example 20.6---------------")
M_bit = 1;
if(M_bit==0) // display the result according to the
   value of the M bit
```

```
    printf("There are no more fragments; this
       fragment is the last one.");
else
    printf("There is at least one more fragment.
       This fragment can be the first one or a
       middle one, but not the last one.\nIt cannot
       be determined if it is the first one or a
       middle one. The value of the fragmentation
       offset is needed to determine this.");
end
```

Scilab code Exa 20.7 M and offset

```
clear;
clc;
disp("----------------Example 20.7----------------")
M_bit = 1;
fragmentation_offset = 0;
if(M_bit==0) // display the result according to the
   value of the M bit
    printf("There are no more fragments; this
       fragment is the last one.");
else
    if(fragmentation_offset == 0) // display the
       result according to the value of the
       fragmentation offset
        printf("The M bit is %d and the offset value
            is %d. Hence it is the first fragment.",
           M_bit,fragmentation_offset);
    else
        printf("It is not first or last fragment. It
            can be any fragment in between.");
    end

end
```

Scilab code Exa 20.8 First byte number

```
clear;
clc;
disp("—————————————Example 20.8——————————————")
offset = 100;
first_byte=offset*8; // formula
printf("The first byte number is %d. The number of
    the last byte cannot be determined as the length
    of the data is unknown.",first_byte); // length
    of data is unknown
```

Scilab code Exa 20.9 First and last byte number

```
clear;
clc;
disp("—————————————Example 20.9——————————————")
offset = 100;
HLEN=5;
total_length_field=100; // 100 bytes
first_byte=offset*8; // formula
header_bytes=HLEN*4; // formula
data_bytes=total_length_field-header_bytes; //
    formula
last_byte=first_byte+data_bytes-1; // formula
printf("The first byte number is %d and the last
    byte number is %d.",first_byte,last_byte);  //
    display result
```

Scilab code Exa 20.10 IPv4 header checksum

```
clear;
clc;
disp("—————————————Example 20.10——————————————")
source_address="10.12.14.5";
destination_address="12.6.7.9";
// convert all the fields to hexadecimal
a1=hex2dec("4500");
a2=28;
a3=1;
a4=0;
a5=hex2dec("0411");
a6=0;
a7=hex2dec("0A0C");
a8=hex2dec("0E05");
a9=hex2dec("0C06");
a10=hex2dec("0709");
d_sum=a1+a2+a3+a4+a5+a6+a7+a8+a9+a10; // find the
   sum of all fielda
Sum=dec2hex(d_sum);
c=bitcmp(d_sum,16); // complement sum
Checksum=dec2hex(c);
printf("Figure shows an example of a checksum
   calculation for an IPv4 header without options.
   The header is divided into 16-bit sections.\nAll
   the sections are added and the sum is
   complemented.The sum is %s and the checksum is %s
   .\nThe result is inserted in the checksum field."
   ,Sum,Checksum); // display result
// display the figure
clf();
xname("—————————————Example 20.10——————————————")
   ;
xrects([.28 .33 .38 .44;.8 .8 .8 .8;.05 .05 .06
   .16;.06 .06 .06 .06]);
xrects([.28 .44 .52;.74 .74 .74 ;.16 .08 .08 ;.06
   .06 .06 ]);
```

```
xrects([.28 .36 .44 ;.68 .68 .68 ;.08 .08 .16 ;.06
   .06 .06 ]);
xrect(.28,.62,.32,.06);
xrect(.28,.56,.32,.06);
for i=0:9
    xarrows([.38 .42],[.47-(i/25) .47-(i/25)],.3);
end
xarrows([.38 .42],[.04 .04],.3);
xarrows([.38 .42],[.002 .002],.3);
xpoly([.44 .52],[.06 .06]);
xset("font size",3);
xstring(.34,.033,"Sum");
xstring(.3,.00001,"Checksum");
xstring(.3,.46,"4,5 and 0");
xstring(.35,.42,"28");
xstring(.36,.38,"1");
xstring(.31,.34,"0 and 0");
xstring(.31,.3,"4 and 17");
xstring(.44,.46,"4  5  0  0");
xstring(.44,.42,"0  0  1  C");
xstring(.44,.38,"0  0  0  1");
xstring(.44,.34,"0  0  0  0");
xstring(.44,.3,"0  4  1  1");
xstring(.44,.26,"0  0  0  0");
xstring(.44,.22,"0  A  0  C");
xstring(.44,.18,"0  E  0  5");
xstring(.44,.14,"0  C  0  6");
xstring(.44,.1,"0  7  0  9");
xstring(.44,.031,"7  4  4  E");
xstring(.44,.000001,"8  B  B  1");
xstring(.36,.26,"0");
xstring(.33,.22,"10.12");
xstring(.34,.18,"14.5");
xstring(.34,.14,"12.6");
xstring(.35,.1,"7.9");
xstring(.4,.565,source_address);
xstring(.41,.51,destination_address);
xstring(.3,.75," 4          5          0
```

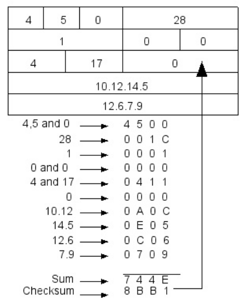

Figure 20.1: IPv4 header checksum

```
                    28");
xstring(.35 ,.69,"1                    0
          0");
xstring(.31,.63,"4              17
                0");
xpoly([.51 .55],[.02 0.02]);
xarrows([.55 .55],[0.02 .65],.6);
```

Chapter 21

Network Layer Address Mapping Error Reporting and Multicasting

Scilab code Exa 21.1 ARP request and reply

```
clear;
clc;
disp("---------------Example 21.1---------------")
// ARP Request
Hardware_type="0001"; // Ethetnet = 1 , in
   hexadecimal
Protocol_type="0800";// IPv4 = 0800 in hexadecimal
Hardware_length="06"; // for Ethernet in hexadecimal
Protocol_length="04"; // for IPv4 in hexadecimal
Operation="0001"; // request=1 , in hexadecimal
Sender_hw_addr="B23455102210" // sender hardware
   address = B2:34:55:10:22:10 in hexadecimal
Sender_pr_addr="130.23.43.20"; // sender protocol
   address=IP address
Target_hw_addr="000000000000"; // unknown to sender
   , hence target hardware address = broadcast
   address
```

```
Target_pr_addr="130.23.43.25"; // target protocol address=IP address
// display ARP Request packet
printf("ARP Request Packet\n");
printf("_____________________________________________________\n");
printf("          |____0x%s_______________|____0x%s_______________|\n",Hardware_type,Protocol_type);
printf("          |___0x%s______|___0x%s____|____0x%s_______________|\n",Hardware_length,Protocol_length,Operation);
printf("          |_______________0x%s______________________|\n",Sender_hw_addr);
printf("          |_________________%s______________________|\n",Sender_pr_addr);
printf("          |_______________0x%s______________________|\n",Target_hw_addr);
printf("          |_________________%s______________________|\n",Target_pr_addr);

// ARP Reply
Hardware_type="0001"; // Ethetnet = 1 , in hexadecimal
Protocol_type="0800";// IPv4 = 0800 in hexadecimal
Hardware_length="06"; // for Ethernet in hexadecimal
Protocol_length="04"; // for IPv4 in hexadecimal
Operation="0002"; // reply=1 , in hexadecimal
Sender_hw_addr="A46EF45983AB" // sender hardware address = A4:6E:F4:59:83:AB in hexadecimal
Sender_pr_addr="130.23.43.25"; // sender protocol address=IP address
Target_hw_addr="B23455102210"; //  target hardware
```

```
address = B2:34:55:10:22:10 in hexadecimal
Target_pr_addr="130.23.43.20"; // target protocol address=IP address
// display ARP Reply Packet
printf("\nARP Reply Packet\n");
printf("         ----------------------------------------------------\n");
printf("         |____0x%s_______________|____0x%s_______________|\n",Hardware_type,Protocol_type);
printf("         |___0x%s______|___0x%s____|____0x%s_______________|\n",Hardware_length,Protocol_length,Operation);
printf("         |_______________0x%s_____________________|\n",Sender_hw_addr);
printf("         |_________________%s_____________________|\n",Sender_pr_addr);
printf("         |_______________0x%s_____________________|\n",Target_hw_addr);
printf("         |_________________%s_____________________|\n",Target_pr_addr);
```

Scilab code Exa 21.2 Echo request message checksum

```
clear;
clc;
disp("----------------Example 21.2-----------------")
identifier=1;
sequence_number=9;
// 8 & 0
```

```
word1a=dec2bin(8,8);
word1b=dec2bin(0,8);
// 0
word2a=dec2bin(0,8);
word2b=dec2bin(0,8);
// 1
word3a=dec2bin(0,8);
word3b=dec2bin(identifier,8);
// 9
word4a=dec2bin(0,8);
word4b=dec2bin(sequence_number,8);
 // TEST
 // T & E
word5a=dec2bin(ascii('T'),8);
word5b=dec2bin(ascii('E'),8);
 // S & T
word6a=dec2bin(ascii('S'),8);
word6b=dec2bin(ascii('T'),8);

sum_dec=bin2dec(word1a+word1b)+0+identifier+
   sequence_number+bin2dec(word5a+word5b)+bin2dec(
   word6a+word6b);
Sum=dec2bin(sum_dec,16); // sum
sum_bytes=strsplit(Sum,8);
cmp=bitcmp(sum_dec,16);
Checksum=dec2bin(cmp,16); // checksum
Checksum_bytes=strsplit(Checksum,8);

// display the result
printf("      8 & 0 :- %s %s\n           0 :- %s %s\n
           %d :- %s %s\n           %d :- %s %s\n
      T & E :- %s %s\n      S & T :- %s %s\n
   Sum    :- %s %s\n   Checksum :- %s %s",word1a,
   word1b,word2a,word2b,identifier,word3a,word3b,
   sequence_number,word4a,word4b,word5a,word5b,
   word6a,word6b,sum_bytes(1),sum_bytes(2),
   Checksum_bytes(1),Checksum_bytes(2));
printf("\n\nThe message is divided into 16-bit (2-
```

```
    byte) words. The words are added and the sum is
    complemented.\nNow the sender can put this value
    in the checksum field.");
```

Scilab code Exa 21.3 ping program

```
clear;
clc;
disp("————————————Example 21.3————————————")
// display the example
printf("We use the ping program to test the server
    fhda.edu. The result is shown below:\n$ ping thda
    .edu\nPING thda.edu (153.18.8.1) 56 (84) bytes of
     data.\n64 bytes from tiptoe.fhda.edu
    (153.18.8.1): icmp_seq=0  ttl=62 time=1.91 ms\n64
     bytes from tiptoe.fhda.edu (153.18.8.1):
    icmp_seq=1  ttl=62 time=2.04 ms\n64 bytes from
    tiptoe.fhda.edu (153.18.8.1): icmp_seq=2  ttl=62
    time=1.90 ms\n64 bytes from tiptoe.fhda.edu
    (153.18.8.1): icmp_seq=3  ttl=62 time=1.97 ms\n64
     bytes from tiptoe.fhda.edu (153.18.8.1):
    icmp_seq=4  ttl=62 time=1.93 ms\n64 bytes from
    tiptoe.fhda.edu (153.18.8.1): icmp_seq=5  ttl=62
    time=2.00 ms\n64 bytes from tiptoe.fhda.edu
    (153.18.8.1): icmp_seq=6  ttl=62 time=1.94 ms\n64
     bytes from tiptoe.fhda.edu (153.18.8.1):
    icmp_seq=7  ttl=62 time=1.94 ms\n64 bytes from
    tiptoe.fhda.edu (153.18.8.1): icmp_seq=8  ttl=62
    time=1.97 ms\n64 bytes from tiptoe.fhda.edu
    (153.18.8.1): icmp_seq=9  ttl=62 time=1.89 ms\n64
     bytes from tiptoe.fhda.edu (153.18.8.1):
    icmp_seq=10 ttl=62 time=1.98 ms\n\n——— thda.edu
    ping statistics ———\n11 packets transmitted, 11
    received, 0%s packet loss, time 10103ms\n
    rttminJavg/max = 1.899/1.955/2.041 ms","%");
```

```
printf("\n\nThe ping program sends messages with sequence numbers starting from 0. For each probe it gives us the RTT time.\nThe TTL (time to live) field in the IP datagram that encapsulates an ICMP message has been set to 62,\nwhich means the packet cannot travel mare than 62 hops. At the beginning, ping defines the number of data bytes as 56\nand the total number of bytes as 84. It is obvious that if we add 8 bytes of ICMP header and 20 bytes of IP header to 56, the result is 84.\nHowever, in each probe ping defines the number of bytes as 64. This is the total number of bytes in the ICMP packet (56 + 8).\nThe ping program continues to send messages, if we do not stop it by using the interrupt key . After it is interrupted,\nit prints the statistics of the probes. It tells us the number of packets sent, the number of packets received, the total time,\nand the RTT minimum, maximum, and average. Some systems may print more information.")
```

Scilab code Exa 21.4 trace route program

```
clear;
clc;
disp("--------------Example 21.4--------------")
// display the example
printf("The traceroute program is used to find the route from the computer voyager.deanza.edu to the server fhda.edu.\nThe following shows the result:\n\n");
printf("$ traceroute fbda.edu\ntraceroute to fbda.edu  (153.18.8.1),30 hops max, 38 byte packets\n1 Dcore.fhda.edu    (153.18.31.254)  0.995 ms  0.899 ms  0.878 ms\n2 Dbackup.fhda.edu
```

```
(153.18.251.4)    1.039 ms   1.064 ms   1.083 ms\n3 tiptoe.fhda.edu    (153.18.8.1)      1.797 ms   1.642 ms   1.757 ms\n\n");
printf("The unnumbered line after the command shows that the destination is 153.18.8.1. The TTL value is 30 hops.\nThe packet contains 38 bytes: 20 bytes of IP header, 8 bytes of UDP header, and 10 bytes of application data.\nThe application data are used by traceroute to keep track of the packets.\n\n");
printf("The first line shows the first router visited. The router is named Dcore.fhda.edu with IP address 153.18.31.254.\nThe first round-trip time was 0.995 ms, the second was 0.899 ms, and the third was 0.878 ms.\n\n");
printf("The second line shows the second router visited. The router is named Dbackup.fhda.edu with IP address 153.18.251.4.\nThe three round-trip times are also shown.\n\n");
printf("The third line shows the destination host. This is the destination host because there are no more lines.\nThe destination host is the server thda.edu, but it is named tiptoe.fhda.edu with the IP address 153.18.8.1.\nThe three round-trip times are also shown.")
```

Scilab code Exa 21.5 trace longer route

```
clear;
clc;
disp("————————————Example 21.5————————————")
// display the example
printf("In this example, we trace a longer route, the route to xerox.com.\n\n");
printf("$ traceroute xerox.com\ntraceroute to xerox.
```

```
    com (13.1.64.93), 30 hops max, 38 byte packets\n1
     Dcore.fbda.edu (153.18.31.254)  0.622 ms  0.891
    ms  0.875 ms\n2 Ddmz.fbda.edu  (153.18.251.40)
    2.132 ms  2.266 ms  2.094ms\n3 Cinic.fhda.edu
    (153.18.253.126) 2.110 ms  2.145 ms  1.763 ms\n4
    cenic.net      (137.164.32.140) 3.069 ms  2.875
    ms  2.930ms\n5 cenic.net      (137.164.22.31)
    4.205 ms  4.870 ms  4.197 ms\n");
printf("      ...              ...               ...
          ...         ...\n");
printf("14 snfc21.pbi.net  (151.164.191.49)  7.656
    ms  7.129 ms  6.866ms\n15 sbcglobaLnet
    (151.164.243.58)  7.844 ms  7.545 ms  7.353 ms\
    n16 pacbell.net     (209.232.138.114) 9.857 ms
    9.535 ms  9.603 ms\n17 209.233.48.223
    (209.233.48.223)  10.634 ms10.771 ms  10.592 ms\
    n18 alpha.Xerox.COM (13.1.64.93)      11.172 ms
    11.048 ms 10.922ms\n\n");
printf("There are 17 hops between source and
    destination. Some round-trip times look unusual.\
    nIt could be that a router was too busy to
    process the packet immediately.")
```

Scilab code Exa 21.6 report messages sequence

```
clear;
clc;
disp("--------------Example 21.6---------------")
// display the sequence of report messages.
printf("The events occur in this sequence:\n\na.
    Time 12: The timer for 228.42.0.0 in host A
    expires, and a membership report is sent,\nwhich
    is received by the router and every host
    including host B which cancels its timer\nfor
    228.42.0.0.\n\nb. Time 30: The timer for
```

```
225.14.0.0 in host A expires, and a membership
report is sent,\nwhich is received by the router
and every host including host C which cancels its
timer\nfor 225.14.0.0.\n\nc. Time 50: The timer
for 238.71.0.0 in host B expires, and a
membership report is sent,\nwhich is received by
the router and every host.\n\nd. Time 70: The
timer for 230.43.0.0 in host C expires, and a
membership report is sent,\nwhich is received by
the router and every host including host A which
cancels its timer\nfor 230.43.0.0.")
```

Scilab code Exa 21.7 Ethernet multicast physical address

```
clear;
clc;
disp("--------------Example 21.7----------------")
// multicast IP address 230.43.14.7
multicast_IP_address=dec2bin(230,5)+dec2bin(43,7)+
    dec2bin(14,7)+dec2bin(7,7);
s=strsplit(multicast_IP_address,length(
    multicast_IP_address)-23);
b=strsplit(s(2),[9 16]);
starting_Ethernet_addr = "01:00:5E"; // 01:00:5E
    :00:00:00
Ethernet_multicast_addr=starting_Ethernet_addr;

function[Ethernet_multicast_addr] = ethernet_address
    (b) // function to form Ethernet multicast
    physical address
  for i=1:3
      d=bin2dec(b(i));
      h(i)=dec2hex(d);  // rightmost 23 bits of
          the IP address in hexadecimal

```

```
    end

    hs=strsplit(h(1));
    if(hex2dec(hs(1)) > = 8)   //subtract 8 from the leftmost digit if it is greater than or equal to 8
        hs(1)=dec2hex(hex2dec(hs(1))-8);
    end
    h(1)=hs(2)+hs(3);
    for i=1:6   // add these hexadecimal digits to the starting Ethernet multicast address, which is 01:00:5E:00:00:00
        if(modulo(i,2) == 0)
            if(length(h(i/2))==2)
                Ethernet_multicast_addr=Ethernet_multicast_addr+h(i/2);
            else
                Ethernet_multicast_addr=Ethernet_multicast_addr+'0'+h(i/2);
            end

        else
            Ethernet_multicast_addr=Ethernet_multicast_addr+":";
        end
    end
endfunction

Ethernet_multicast_addr=ethernet_address(b);
printf("The Ethernet multicast physical address is %s.",Ethernet_multicast_addr); // display result
```

Scilab code Exa 21.8 Ethernet multicast address

```
clear;
clc;
disp("----------------Example 21.8-----------------")
// multicast IP address 238.212.24.9
multicast_IP_address=dec2bin(238,5)+dec2bin(212,7)+dec2bin(24,7)+dec2bin(9,7);
s=strsplit(multicast_IP_address,length(multicast_IP_address)-23);
b=strsplit(s(2),[9 16]);
starting_Ethernet_addr = "01:00:5E"; // 01:00:5E:00:00:00
Ethernet_multicast_addr=starting_Ethernet_addr;

function[Ethernet_multicast_addr] = ethernet_address(b) // function to form Ethernet multicast physical address
    for i=1:3
        d=bin2dec(b(i));
        h(i)=dec2hex(d);  // rightmost 23 bits of the IP address in hexadecimal

    end

    hs=strsplit(h(1));
    if(hex2dec(hs(1)) > = 8)   //subtract 8 from the leftmost digit if it is greater than or equal to 8
        hs(1)=dec2hex(hex2dec(hs(1))-8);
    end
    h(1)=hs(1)+hs(2);
    for i=1:6   // add these hexadecimal digits to the starting Ethernet multicast address, which is 01:00:5E:00:00:00
        if(modulo(i,2) == 0)
            if(length(h(i/2))==2)
                Ethernet_multicast_addr=Ethernet_multicast_addr+h(i/2);
            else
```

```
                Ethernet_multicast_addr=Ethernet_multicast_addr+'0'+h(i/2);
            end

        else
            Ethernet_multicast_addr=Ethernet_multicast_addr+":";
        end
    end
endfunction

Ethernet_multicast_addr=ethernet_address(b);
printf("The Ethernet multicast physical address is %s.",Ethernet_multicast_addr);  // display result
```

Scilab code Exa 21.9 netstat nra

```
clear;
clc;
disp("-----------------Example 21.9-----------------")
// display the example
printf("We use netstat with three options: -n, -r, and -a. The -n option gives the numeric versions of IP\naddresses, the -r option gives the routing table, and the -a option gives all addresses (unicast and\nmulticast). Gateway defines the router, Iface defines the interface.\n\n");
printf("$ netstat -nra\nKernel IP routing table\nDestination     Gateway          Mask     Flags    Iface\n153.18.16.0     0.0.0.0     255.255.240.0   U       eth0\n169.254.0.0     0.0.0.0          255.255.0.0     U       eth0\n127.0.0.0     0.0.0.0           255.0.0.0       U        lo\n224.0.0.0       0.0.0.0            224.0.0.0        U
```

```
        eth0\n0.0.0.0         153.18.31.254      0.0.0.0
           UG      eth0\n\n");
printf("Any packet with a multicast address from
   224.0.0.0 to 239.255.255.255 is masked and
   delivered to the Ethernet interface.")
```

Chapter 22

Network layer Delivery Forwarding and Routing

Scilab code Exa 22.1 Routing Table for router

```
clear;
clc;
disp("-----------------Example 22.1-------------------")
// network addresses
network_address1="180.70.65.192";
network_address2="180.70.65.128";
network_address3="201.4.22.0";
network_address4="201.4.16.0";
network_address5="Any"   // Rest of the internet
// masks
mask1="/26";
mask2="/25";
mask3="/24";
mask4="/22";
mask5="Any";   // Rest of the internet
// interfaces
interface1="      m2";
interface2="      m0";
interface3="      m3";
```

```
interface4="      m1";
interface5="      m2";  // Rest of the internet

router_address="180.70.65.200";   // Router R1
// next hop addresses
next_hop1="       -";
next_hop2="       -";
next_hop3="       -";
next_hop4="       -";
next_hop5=router_address; // For rest of the
   universe

// define matrices for the 4 columns of the routing
   table
mask = [mask1; mask2; mask3; mask4; mask5];
network_address=[network_address1; network_address2;
    network_address3; network_address4;
   network_address5];
interface=[interface1; interface2; interface3 ;
   interface4; interface5];
next_hop=[next_hop1;next_hop2;next_hop3;next_hop4;
   next_hop5];

// define a matrix for the whole routing table
routing_table=[mask network_address next_hop
   interface];

// displaying the routing table
printf("\n         ROUTING TABLE FOR ROUTER R1\n");
printf("\n!Mask|Network address|   Next hop   |
   Interface!\n"); // display the headings
disp(routing_table); // display the routing table
   matrix
```

Scilab code Exa 22.2 Forwarding process 1

```
clear;
clc;
disp("----------------Example 22.2-----------------")
// network addresses
network_addresses=["180.70.65.192","180.70.65.128","201.4.22.0","201.4.16.0","Any"];
// masks
mask=[26,25,24,22];

// interfaces
interface=["m2" "m0" "m3" "m1" "m2"];

//destination address = 180.70.65.140
byte1=180;
byte2=70;
byte3=65;
byte4=140;
// convert it to binary
b1=dec2bin(byte1,8);
b2=dec2bin(byte2,8);
b3=dec2bin(byte3,8);
b4=dec2bin(byte4,8);
destination_address=b1+b2+b3+b4; // destination address in binary
network_address="";

for i=1:4  // applying the each of the masks to the destination address
    na="";
    printf("\n\n%d) The mask /%d is applied to the destination address.",i,mask(i));
    nz=32-mask(i);  // number of zeros after applying the mask
    s=strsplit(destination_address);
        for k=33-nz:32
            s(k)='0'; // replacing last 'nz' bits with zeros
        end
```

```
        for k=1:32
            na=na+s(k);  // new address in binary
        end
        bytes=strsplit(na,[8 16 24]); // split it into bytes
       // convert them to binary
        d1=bin2dec(bytes(1));
        d2=bin2dec(bytes(2));
        d3=bin2dec(bytes(3));
        d4=bin2dec(bytes(4));
        network_address=string(d1)+"."+string(d2)+"."+string(d3)+"."+string(d4); // network address formed in decimal notation

    if(network_address==network_addresses(i))  // check if it matches with any given network addresses and display appropriate results
        printf("\nThe result is %s,which matches the corresponding network address %s.\nThe next-hop address (the destination address of the packet in this case) and the interface number %s are passed to ARP for further processing.",network_address,network_addresses(i),interface(i));
        break;
    else
        printf("\nThe result is %s, which does not match the corresponding network address %s.",network_address,network_addresses(i));
    end
end
```

Scilab code Exa 22.3 Forwarding process 2

```
clear;
clc;
disp("---------------Example 22.3-----------------")
// network addresses
network_addresses=["180.70.65.192","180.70.65.128","
   201.4.22.0","201.4.16.0","Any"];
// masks
mask=[26,25,24,22];

// interfaces
interface=["m2" "m0" "m3" "m1" "m2"];

//destination address = 201.4.22.35
byte1=201;
byte2=4;
byte3=22;
byte4=35;
// convert it to binary
b1=dec2bin(byte1,8);
b2=dec2bin(byte2,8);
b3=dec2bin(byte3,8);
b4=dec2bin(byte4,8);
destination_address=b1+b2+b3+b4;  // binary form
network_address="";

for i=1:4  // applying the each of the masks to the
   destination address
    na="";
    printf("\n\n%d) The mask /%d is applied to the
       destination address.",i,mask(i));
    nz=32-mask(i);  // number of zeros after
       applying the mask
    s=strsplit(destination_address);
        for k=33-nz:32
            s(k)='0';  // replacing last 'nz' bits
               with zeros
        end
        for k=1:32
```

```
            na=na+s(k);   // new address in binary
        end
        bytes=strsplit(na,[8 16 24]);   // split it into bytes
       // convert it into decimal
        d1=bin2dec(bytes(1));
        d2=bin2dec(bytes(2));
        d3=bin2dec(bytes(3));
        d4=bin2dec(bytes(4));
        network_address=string(d1)+"."+string(d2)+"."+string(d3)+"."+string(d4);   // network address formed in decimal notation

    if(network_address==network_addresses(i))   // check if it matches with any given network addresses and display appropriate results
        printf("\nThe result is %s, which matches the corresponding network address %s.\nThe destination address of the packet and the interface number %s are passed to ARP.",network_address,network_addresses(i),interface(i));
        break;
    else
        printf("\nThe result is %s, which does not match the corresponding network address %s.",network_address,network_addresses(i));
    end
end
```

Scilab code Exa 22.4 Forwarding process 3

```
clear;
clc;
```

```
disp("---------------Example 22.4-----------------")
// network addresses
network_addresses=["180.70.65.192","180.70.65.128","201.4.22.0","201.4.16.0","Any"];
// masks
mask=[26,25,24,22];

// interfaces
interface=["m2" "m0" "m3" "m1" "m2"];
//destination address = 18.24.32.78
byte1=18;
byte2=24;
byte3=32;
byte4=78;
// convert it to binary
b1=dec2bin(byte1,8);
b2=dec2bin(byte2,8);
b3=dec2bin(byte3,8);
b4=dec2bin(byte4,8);
destination_address=b1+b2+b3+b4;
network_address="";

nexthop_address="180.70.65.200";  // Router R1 address

for i=1:4  // applying the each of the masks to the destination address
    na="";
    printf("\n\n%d) The mask /%d is applied to the destination address.",i,mask(i));
    nz=32-mask(i);  // number of zeros after applying the mask
    s=strsplit(destination_address);
        for k=33-nz:32
            s(k)='0';  // replacing last 'nz' bits with zeros
        end
        for k=1:32
```

```
            na=na+s(k); // new address in binary
        end
        bytes=strsplit(na,[8 16 24]); // split the new address into bytes
       // convert them to decimal
        d1=bin2dec(bytes(1));
        d2=bin2dec(bytes(2));
        d3=bin2dec(bytes(3));
        d4=bin2dec(bytes(4));
        network_address=string(d1)+"."+string(d2)+"."+string(d3)+"."+string(d4); // final network address in decimal notation

    if(network_address==network_addresses(i))  // check if it matches with any given network addresses and display appropriate results
        printf("\nThe result is %s, which matches the corresponding network address %s .\nThe destination address of the packet and the interface number %s are passed to ARP.",network_address,network_addresses(i),interface(i));
        break;
    else
        printf("\nThe result is %s, which does not match the corresponding network address %s.",network_address,network_addresses(i));
    end
end

if(i==4) // if it doesnt match any of the 4 given network addresses
    printf("\n\nNo matching network address is found.\nWhen it reaches the end of the table, the module gives the next-hop address %s and interface number %s to ARP.\nThis is probably an outgoing package that needs to be sent,
```

```
via the default router, to someplace else in
    the Internet.",nexthop_address,interface(5))
end
```

Scilab code Exa 22.5 Hierarchical routing

```
clear;
clc;
disp("—————————————Example 22.5———————————————")
total_addresses=16384; // A regional ISP is granted
    with 16,384 addresses
// starting address = 120.14.64.0
sa1=120;
sa2=14;
sa3=64;
sa4=0;
bin_sa=dec2bin(sa1,8)+dec2bin(sa2,8)+dec2bin(sa3,8)+
    dec2bin(sa4,8); // starting address in binary
la=dec2bin(bin2dec(bin_sa)+total_addresses-1,32); //
     last address in binary
a=strsplit(la,[8 16 24 ])  //separate the bytes
a3=bin2dec(a(1));          // convert binary numbers
    to decimal numbers
a2=bin2dec(a(2));
a1=bin2dec(a(3));
a0=bin2dec(a(4));
last_address=string(a3)+"."+string(a2)+"."+string(a1
    )+"."+string(a0); // last address in decimal
    notation
main_mask=18;
main_subblocks=4;
msb_addresses=total_addresses/main_subblocks; // The
     regional ISP divides this block into four
    subblocks, each with 4096 addresses.
msb_mask=main_mask+log2(main_subblocks); // the mask
```

```
for each block is /20 because the original block with mask /18 is divided into 4 blocks

msb1_subblocks=8; // The first local ISP has divided its assigned subblock into 8 smaller blocks and assigned each to a small ISP
mssb1_mask=msb_mask+log2(msb1_subblocks); // he mask for each small ISP is now /23 because the block is further divided into 8 blocks.
household_addresses=4; // a household has four addresses
household_mask=32-log2(household_addresses); // formula
num_households=msb_addresses/(household_addresses*msb1_subblocks); // Each small ISP provides services to 128 households (H001 to H128)
msb1_sa=string(sa1)+"."+string(sa2)+"."+string(sa3)+"."+string(sa4); // starting address in decimal notation
la1=dec2bin(bin2dec(bin_sa)+msb_addresses-1,32); // last address in binary
a=strsplit(la1,[8 16 24 ])  //separate the bytes
a3=bin2dec(a(1));           // convert binary numbers to decimal numbers
a2=bin2dec(a(2));
a1=bin2dec(a(3));
a0=bin2dec(a(4));
last_address1=string(a3)+"."+string(a2)+"."+string(a1)+"."+string(a0); // last address in decimal notation
H001_la=dec2bin(bin2dec(bin_sa)+household_addresses-1,32); // last address in binary
a=strsplit(H001_la,[8 16 24 ])  //separate the bytes
a3=bin2dec(a(1));           // convert binary numbers to decimal numbers
a2=bin2dec(a(2));
a1=bin2dec(a(3));
a0=bin2dec(a(4));
```

```
last_address_H001=string(a3)+"."+string(a2)+"."+string(a1)+"."+string(a0); // last address in decimal notation


msb2_subblocks=4; //The second local ISP divides its block into 4 blocks and assigns the addresses to four large organizations (LOrg01 to LOrg04).
Lorg_addresses=msb_addresses/msb2_subblocks; // number of addresses possessed by each large organization
mssb2_mask=msb_mask+log2(msb2_subblocks); // mask of the large organization addresses
sas2=dec2bin(bin2dec(bin_sa)+2*(msb_addresses),32); // starting address in binary
a=strsplit(sas2,[8 16 24 ]) //separate the bytes
a3=bin2dec(a(1));          // convert binary numbers to decimal numbers
a2=bin2dec(a(2));
a1=bin2dec(a(3));
a0=bin2dec(a(4));
start_address2=string(a3)+"."+string(a2)+"."+string(a1)+"."+string(a0); // starting address in decimal notation

msb3_subblocks=16; //The third local ISP divides its block into 16 blocks and assigns the addresses to 15 small organizations (SOrg01 to SOrg16).
Sorg_addresses=msb_addresses/msb3_subblocks; // number of addresses possessed by each small organization
mssb3_mask=msb_mask+log2(msb3_subblocks); // mask of the small organization addresses
sas3=dec2bin(bin2dec(bin_sa)+3*(msb_addresses),32); // starting address in binary
a=strsplit(sas3,[8 16 24 ]) //separate the bytes
a3=bin2dec(a(1));          // convert binary numbers to decimal numbers
```

```
a2=bin2dec(a(2));
a1=bin2dec(a(3));
a0=bin2dec(a(4));
start_address3=string(a3)+"."+string(a2)+"."+string(
    a1)+"."+string(a0); // starting address in
    decimal notation

// display the result

printf("A regional ISP is granted %d addresses
    starting from %s .The regional ISP has decided to
     divide this block into %d subblocks ,\neach with
    %d addresses.Three of these subblocks are
    assigned to three local ISPs; the second subblock
     is reserved for future use.\nThe mask for each
    block is /%d because the original block with mask
     /%d is divided into %d blocks.\n\nThe first
    local ISP has divided its assigned subblock into
    %d smaller blocks and assigned each to a small
    ISP.\nEach small ISP provides services to %d
    households (H001 to H128),each using %d addresses
    .\nThe mask for each small ISP is now /%d because
     the block is further divided into %d blocks.
    Each household has a mask of /%d\nbecause a
    household has only %d addresses.\n\nThe second
    local ISP has divided its block into %d blocks
    and has assigned the addresses to %d large
    organizations (LOrg01 to LOrg04).\nEach large
    organization has %d addresses, and the mask is /
    %d and the starting address is %s .\n\nThe third
    local ISP has divided its block into %d blocks
    and has assigned the addresses to %d large
    organizations (SOrg01 to SOrg16).\nEach large
    organization has %d addresses, and the mask is /
    %d and the starting address is %s .\n\nThere is a
     sense of hierarchy in this configuration. All
    routers in the Internet send a packet with
    destination address\n%s to %s to the regional ISP
```

```
.\n\nThe regional ISP sends every packet with
destination address %s to %s to local ISP1.\
nLocal ISP1 sends every packet with destination
address %s to %s to H001.",total_addresses,
msb1_sa,main_subblocks,msb_addresses,msb_mask,
main_mask,main_subblocks,msb1_subblocks,
num_households,household_addresses,mssb1_mask,
msb1_subblocks,household_mask,household_addresses
,msb2_subblocks,msb2_subblocks,Lorg_addresses,
mssb2_mask,start_address2,msb3_subblocks,
msb3_subblocks,Sorg_addresses,mssb3_mask,
start_address3,msb1_sa,last_address,msb1_sa,
last_address1,msb1_sa,last_address_H001);
```

Scilab code Exa 22.6 netstat and ifconfig

```
clear;
clc;
disp("—————————————Example 22.6——————————————")
// display the example
printf("One utility that can be used to find the
   contents of a routing table for a host or router
   is netstat in UNIX or LINUX.\nThe following shows
    the list of the contents of a default server.
   Two options, r and n are used.\nThe option r
   indicates that we are interested in the routing
   table, and the option n indicates that we are
   looking for numeric addresses.\nThis is a routing
    table for a host, not a router. Although we
   discussed the routing table for a router
   throughout the chapter,\na host also needs a
   routing table.\n");
// output of $netstat -rn command
printf("\n$ netstat -rn\nKernel IP routing table\
   nDestination    Gateway        Mask            Flags
```

```
    Iface\n153.18.16.0    0.0.0.0
    255.255.240.0      U      eth0\n127.0.0.0
    0.0.0.0          255.0.0.0            U      la\n0.0.0.0
         153.18.31.254    0.0.0.0              G      eth0
    ");
// explain the diffrent columns
printf("\n\nNote also that the order of columns is
    different from what we showed. The destination
    column here defines the network address.\nThe
    term gateway used by UNIX is synonymous with
    router. This column acmally defines the address
    of the next hop.\nThe value 0.0.0.0 shows that
    the delivery is direct. The last entry has a flag
     of G,\nwhich means that the destination can be
    reached through a router (default router). The
    Iface defines the interface.\nThe host has only
    one real interface,eth0,which means interface 0
    connected to an Ethernet network.\nThe second
    interface, la,is actually a virtual loopback
    interface indicating that the host accepts
    packets with loopback address 127.0.0.0.");
//output of $ifconfig eth0 command
printf("\n\nMore information about the IP address
    and physical address of the server can be found
    by using the ifconfig command on the given
    interface (eth0).\n$ ifconfig eth0\neth0 Link
    encap:Ethernet HWaddr 00:BO:DO:DF:09:5D\ninet
    addr:153.18.17.11 Bcast: 153.18.31.255 Mask
    :255.255.240.0");
```

Chapter 23

Process to Process Delivery UDp TCp and SCTP

Scilab code Exa 23.1 grep etc services

```
clear;
clc;
disp("--------------Example 23.1----------------")
// display the example
printf("In UNIX, the well-known ports are stored in
   a file called fetcfservices. Each line in this
   file gives\nthe name of the server and the well-
   known port number.The following shows the port
   for FTP. Note that FTP can use port 21 with
   either UDP or TCP.\n\n$grep   ftp   /etc/services\
   nftp     21/tcp\nftp     21/udp");
printf("\n\nSNMP uses two port numbers (161 and 162)
   , each for a different purpose.\n\n$grep snmp /
   etc/services\nsnmp              161/tcp
   #Simple Net Mgmt Proto\nsnmp              161/udp
              #Simple Net Mgmt Proto\nsnmptrap
          162/udp            #Traps for SNMP");
```

Scilab code Exa 23.2 Checksum with padding

```
clear;
clc;
disp("------------------Example 23.2--------------------")
printf("The datagram has only 7 bytes of data. Because the number of bytes of data is odd, padding is added for checksum calculation.\nThe pseudoheader as well as the padding will be dropped when the user datagram is delivered to IP."); // display the example
```

Scilab code Exa 23.3 Segment sequence number

```
clear;
clc;
disp("------------------Example 23.3--------------------")
first_byte= 10001;
num_segments=5;
total_bytes = 5000;
segment=1000; // each segment carries 1000 bytes
// compute the sequence numbers of each segment
segment1_sequence_num = first_byte;
segment2_sequence_num = segment1_sequence_num+segment;
segment3_sequence_num = segment2_sequence_num+segment;
segment4_sequence_num = segment3_sequence_num+segment;
segment5_sequence_num = segment4_sequence_num+segment;
// find the range of each segment
```

```
range1=segment1_sequence_num+segment-1;
range2=range1+segment;
range3=range2+segment;
range4=range3+segment;
range5=range4+segment;
// display the result
printf("Segment 1:- Sequence Number: %d and range:
    %d to %d\nSegment 2:- Sequence Number: %d and
    range: %d to %d\nSegment 3:- Sequence Number: %d
    and range: %d to %d\nSegment 4:- Sequence Number:
     %d and range: %d to %d\nSegment 5:- Sequence
    Number: %d and range: %d to %d",
    segment1_sequence_num,segment1_sequence_num,
    range1,segment2_sequence_num,
    segment2_sequence_num,range2,
    segment3_sequence_num,segment3_sequence_num,
    range3,segment4_sequence_num,
    segment4_sequence_num,range4,
    segment5_sequence_num,segment5_sequence_num,
    range5);
```

Scilab code Exa 23.4 Value of rwnd

```
clear;
clc;
disp("---------------Example 23.4-----------------")
buffer_size=5000; //bytes
recieved_unprocessed = 1000; // bytes
rwnd=buffer_size-recieved_unprocessed ; // formula
printf("The value of rwnd = %d . Hence Host B can
    receive only %d bytes of data before overflowing
    its buffer.",rwnd,rwnd); // display result
```

Scilab code Exa 23.5 Window for host

```
clear;
clc;
disp("————————————Example 23.5—————————————")
rwnd=3000; // bytes
cwnd=3500; // bytes
window_size=min(rwnd,cwnd); // formula
printf("The size of the window is the smaller of
   rwnd and cwnd, which is %d bytes.",window_size);
   // display the result
```

Scilab code Exa 23.6 Unrealistic sliding window

```
clear;
clc;
disp("————————————Example 23.6—————————————")
rwnd=9; // bytes
cwnd=20; // bytes
window_size=min(rwnd,cwnd); // formula
// display result
printf("This an unrealistic example of a sliding
   window. The sender has sent bytes up to 202.\nThe
    receiver has sent an acknowledgment number of
   200 with an rwnd of %d bytes.\nThe size of the
   sender window is the minimum of rwnd and cwnd, or
    %d bytes. Bytes 200 to 202 are sent, but not
   acknowledged.\nBytes 203 to 208 can be sent
   without worrying about acknowledgment. Bytes 209
   and above cannot be sent.",rwnd,window_size);
```

Chapter 26

Remote Logging Electronic Mail and File Transfer

Scilab code Exa 26.1 Option negotiation

```
clear;
clc;
disp("--------------Example 26.1---------------")
// client request :- Do enable the echo option
r_character1="IAC";
r_character2="DO";
r_character3="ECHO";
//server approval :- I will enable the echo option
a_character1="IAC";
a_character2="WILL";
a_character3="ECHO";
printf("In this example, the client wants the server
    to echo each character sent to the server. In
   other words,\nwhen a character is typed at the
   user keyboard terminal, it goes to the server and
    is sent back to the screen of the user before
   being processed.\nThe echo option is enabled by
   the server because it is the server that sends
   the characters back to the user terminal.\
```

```
nTherefore, the client should request from the
server the enabling of the option using DO.\nThe
request consists of three characters: %s, %s and
%s.\nThe server accepts the request and enables
the option WILL.\nIt informs the client by
sending the three-character approval: %s, %s and
%s.",r_character1,r_character2,r_character3,
a_character1,a_character2,a_character3); //
display result
```

Scilab code Exa 26.2 Suboption negotiation

```
clear;
clc;
disp("----------------Example 26.2----------------")
// request :- Do enable terminal option
re_character1="IAC";
re_character2="DO";
re_character3="Terminal type";
//approval :- will enable the terminal option
a_character1="IAC";
a_character2="WILL";
a_character3="Terminal type";
//request :- Set the terminal type to "VT"
r_character1="IAC";
r_character2="SB";
r_character3="Terminal type";
r_character4="V";
r_character5="T";
r_character6="IAC";
r_character7="SE";
// display the example
printf("   Client

   Server\n");
```

```
printf("     |                    I will enable the terminal option          |\n");
printf("     |-------------|%s|-|%s|-|%s|------------->|\n",a_character3,a_character2,a_character1);
printf("     |                                                               |\n");
printf("     |                    Do enable terminal option                  |\n");
printf("     |<------------|%s|-|%s|-|%s|----------------|\n",re_character1,re_character2,re_character3);
printf("     |                                                               |\n");
printf("     |                    Set the terminal type to VT                |\n");
printf("     |----|%s|-|%s|-|%s|-|%s|-|%s|-|%s|-|%s|--->|\n",r_character7,r_character6,r_character5,r_character4,r_character3,r_character2,r_character1);
```

Scilab code Exa 26.3 Email using SMTP

```
clear;
clc;
disp("---------------Example 26.3---------------")
// display the example
printf("We use TELNET to log into port 25 (the wellknown port for SMTP). We then use the commands directly to send an e-mail.\nIn this example,forouzanb@adelphia.net is sending an e-mail to himself. The first few lines show TELNET trying to connect to the Adelphia mail server.\
```

```
nAfter connection, we can type the SMTP commands and then receive the responses, as shown below.\nComment lines are designated by the = signs. These lines are not part of the e-mail procedure.");
// display the SMTP commands and responses
printf("\n\n$ telnet mail.adelphia.net25\nTrying 68.168.78.100...\nConnected to mail.adelphia.net (68.168.78.100).");
printf("\n\n================ Connection Establishment===================\n 220 mta13.adelphia.net SMTP server ready Fri, 6 Aug 2004 ...\nHELO mail.adelphia.net\n 250 mtal3.adelphia.net");
printf("\n\n============= Mail Transfer ================\nMAIL FROM: forouzanb@adelphia.net\n 250 Sender:<forouzanb@adelphia.net>:Ok\nRCPT TO:forounzanb.@adelphia.net\n 250 Recipient:<forouzanb@adelphia.net>Ok\nDATA\n 354 Ok Send data ending with <CRLF>.<CRLF>\nFrom:Forouzan\nTO:Forouzan");
printf("\n\nThis is a test message\nto show SMTP in action...");
printf("\n\n=============Connection Termination================\n 250 Message received:adelphia.net@mail.adelphia.net\nQUIT\n 221 mta13.adelphia.net SMTP server closing connection\nConnection closed by foreign host.");
```

Scilab code Exa 26.4 FTP session

```
clear;
clc;
disp("-------------Example 26.4-------------")
// display the example
```

```
printf("The following shows an actual FTP session for retrieving a list of items in a directory. Lines in the middle show \ncommands sent by the client and the top and bottom lines show data transfer.");
// display the commands and responses
printf("\n\n$ ftp voyager.deanza.tbda.edu\nConnected to voyager.deanza.tbda.edu.\n220(vsFTPd 1.2.1)\n530 Please login with USER and PASS.\nName(voyager.deanza.tbda.edu:forouzan): forouzan\n331 Please specify the password.\nPassword:\n230 Login successful.\nRemote system type is UNIX.\nUsing binary mode to transfer files.\nftp> Is reports\n227 Entering Passive Mode (153,18,17,11,238,169)\n150 Here comes the directory listing.");
printf("\n\ndrwxr-xr-x  2 3027    411         4096 Sep24 2002 business\ndrwxr-xr-x  2 3027    411         4096Sep24 2002 personal\ndrwxr-xr-x  2 3027    411         4096Sep24 2002 school");
printf("\n\n226 Directory send OK.\nftp>quit\n221 Goodbye.");
printf("\n\n\nn1. After the control connection is created, the FIP server sends the 220 (service ready) response on the control connection.\n2. The client sends its name.\n3. The server responds with 331 (user name is OK, password is required).\n4. The client sends the password (not shown).\n5. The server responds with 230 (user log-in is OK).\n6. The client sends the list command Os reports) to find the list of files on the directory named report.\n7. Now the server responds with 150 and opens the data connection.\n8. The server then sends the list of the files or directories (as a file) on the data connection.\nWhen the whole list (file) is sent, the server responds with 226 (closing data connection)over the control connection.\n9. The client now has
```

```
two choices. It can use the QUIT command to request the closing of the control connection, or it can send\nanother command to start another activity (and eventually open another data connection). In our example, the client sends a QUIT command.\n10. After receiving the QUIT command, the server responds with 221 (service closing) and then closes the control connection.")
```

Scilab code Exa 26.5 Anonymous FTP

```
clear;
clc;
disp("----------------Example 26.5----------------")
printf("This an example of anonymous FTP. We assume that some public data are available at internic.net.Lines in the middle show \ncommands sent by the client and the top and bottom lines show data transfer.\n\n");
printf("$ ftp internic.net\nConnected to internic.net\n220 Server ready\nName: anonymous\n331 Guest login OK, send guest as password\nPassword:guest\nftp>pwd\n257 / is current directory\nftp>ls\n200 OK\n150 Opening ASCII mode");
printf("\n\nbin\n...\n...\n...");
printf("\n\nftp>close\n221 Goodbye\nftp>quit");
```

Chapter 27

WWW and HTTP

Scilab code Exa 27.1 Retrieving document using GET

```
clear;
clc;
disp("---------------Example 27.1---------------")
// display the example
printf("This example retrieves a document. We use the GET method to retrieve an image with the path /usr/bin/image1.\nThe request line shows the method (GET), the URL, and the HTTP version(1.1).\nThe header has two lines that show that the client can accept images in the GIF or JPEG format.\nThe request does not have a body. The response message contains the status line\nand four lines of header. The header lines define the date, server,\nMIME version, and length of the document. The body of the document follows the header.");
// figure
printf("\n\n  Client                                                        Server\n");
printf("    |             Request (GET method)
```

```
                             |\n");
printf("    |                                                    |\n");
printf("    |------------|GET /usr/bin/image1 HTTP/1.1  |--------->|\n");
printf("    |            |Accept: image/gif              |          |\n");
printf("    |            |Accept: image/jpeg             |          |\n");
printf("    |            |-------------------------------|          |\n");
printf("    |                                                       |\n");
printf("    |                                                    |\n");
printf("    |            |HTTP/1.1 200 OK                |          |\n");
printf("    |            |Date:Mon07-Jan-05 13:15:14 GMT|          |\n");
printf("    |<-----------|Server: Challenger             |----------|\n");
printf("    |            |MIME-version: 1.0              |          |\n");
printf("    |            |Content-length: 2048           |          |\n");
printf("    |            |                               |          |\n");
printf("    |            |(Body of the document)         |          |\n");
printf("    |            |-------------------------------|          |\n");
printf("    |            Response                                   |\n");
```

Scilab code Exa 27.2 Send data using POST

```
clear;
clc;
disp("--------------------Example 27.2--------------------")
// display the example
printf("In this example, the client wants to send data to the server. The POST method is used. The request line shows the method (POST),\nURL, and HTTP version (1.1). There are four lines of headers. The request body contains the input information. The response message contains\nthe status line and four lines of headers. The created document, which is a CGI document, is included as the body.");
// figure
printf("\n\n   Client                                                                                 Server\n");
printf("    |                Request (POST method)                |\n");
printf("    |         -------------------------------             |\n");
printf("    |-----------|POST /cgi-bin/doc.pl HTTP/1.1 |--------->|\n");
printf("    |           |Accept: */*                   |          |\n");
printf("    |           |Accept: image/gif             |          |\n");
printf("    |           |Accept: image/jpeg            |          |\n");
printf("    |           |Content-length: 50            |          |\n");
printf("    |           |                              |          |\n");
printf("    |           |(Input information)           |          |\n");
printf("    |           |
```

```
-------------------------------|          |\n");
printf("    |                                        |\n");
printf("    |                 ------------------------------          |\n");
printf("    |           |HTTP/1.1 200 OK               |          |\n");
printf("    |           |Date:Mon,07-Jan-02 13:15:14 GMT|          |\n");
printf("    |<----------|Server: Challenger            |----------|\n");
printf("    |           |MIME-version: 1.0             |          |\n");
printf("    |           |Content-length: 2000          |          |\n");
printf("    |           |                              |          |\n");
printf("    |           |(Body of the document)        |          |\n");
printf("    |           |------------------------------|          |\n");
printf("    |              Response                              |\n");
```

Scilab code Exa 27.3 Connect to server using TELNET

```
clear;
clc;
disp("----------------Example 27.3----------------")
// display the example
printf("HTTP uses ASCII characters. A client can directly connect to a server using TELNET, which logs into port 80.\nThe next three lines show that the connection is successful.\nThe first
```

```
shows the request line (GET method), the second
   is the header (defining the host), the third is a
    blank, terminating the request.\nThe server
   response is seven lines starting with the status
   line.\nThe blank line at the end terminates the
   server response. The file of 14,230 lines is
   received after the blank line.\nThe last line is
   the output by the client.\n\n");
printf("$ telnet www.mhhe.com 80\nTrying
   198.45.24.104...\nConnected to www.mhhe.com
   (198.45.24.104).\nEscape character is ^]\nGET /
   engcs/compsci/forouzan HTTP/1.1\nFrom:
   forouzanbehrouz@fbda.edu\n\n");
printf("HTTP/1.1 200 OK\nDate: Thu, 28 Oct 2004
   16:27:46 GMT\nServer: Apache/1.3.9 (Unix)
   ApacheJServ/1.1.2 PHP/4.1.2 PHP/3.0.18\nMIME-
   version:1.0\nContent-Type: text/html\nLast-
   modified: Friday, 15-Oct-04 02:11:31 GMT\nContent
   -length: 14230\n\nConnection closed by foreign
   host.");
```

Chapter 28

Network Management SNMP

Scilab code Exa 28.1 Define INTEGER 14

```
clear;
clc;
disp("--------------Example 28.1---------------")
tag="00000010"; // INTEGER tag
Length="00000100"; // 4 bytes = 4*8=32 bits
value=14; // INTEGER 14
value_bin=dec2bin(value,32); // value in binary
value_hex=dec2hex(value); // value in hexadecimal
tag_dec=bin2dec(tag); // tag's decimal value
Length_dec=bin2dec(Length); // length's decimal
   value or number of bytes
bytes=strsplit(value_bin,[8 16 24]); // split value
   into 4 bytes
// convert the bytes to decimal
byte1=bin2dec(bytes(1));
byte2=bin2dec(bytes(2));
byte3=bin2dec(bytes(3));
// display the format
printf("\n        0%d            0%d            0%d
            0%d          0%d          0%s\n",tag_dec,
   Length_dec,byte1,byte2,byte3,value_hex);
```

```
printf("      |  %s  |  %s  |%s  |    %s  |    %s  |    %s|\n",tag,Length,bytes(1),bytes(2),bytes(3),bytes(4));
printf("         Tag           Length          Value(%d)\n",value);
printf("       (integer)   (4  bytes)");
```

Scilab code Exa 28.2 Define OCTETSTRING HI

```
clear;
clc;
disp("---------------Example 28.2---------------")
tag="00000100"; // OCTET STRING tag
Length="00000010"; // 2 bytes
octet_string="HI";
H_value="01001000"; // 48
I_value="01001001"; // 49
H_value_dec=48; // value in decimal
I_value_dec=49; // value in decimal
tag_dec=bin2dec(tag); // tag's decimal value
Length_dec=bin2dec(Length); // length's decimal value or number of bytes
// display the format
printf("\n         0%d              0%d              %d           %d\n",tag_dec,Length_dec,H_value_dec,I_value_dec);
printf("      |  %s  |  %s  |  %s  |    %s  |\n",tag,Length,H_value,I_value);
printf("         Tag           Length          Value         Value\n");
printf("       (string)    (2  bytes)        (H)            (I)");
```

Scilab code Exa 28.3 Define ObjectIdentifier

```
clear;
clc;
disp("——————————————Example 28.3——————————————")
tag="00000110"; // OBJECT IDENTIFIER tag
Length="00000100"; // 4 bytes
ObjectIdentifier="1.3.6.1"; // (iso.org.dod.internet
    )
ot1=1;
ot2=3;
ot3=6;
ot4=1;
// convert the bytes to binary
byte1=dec2bin(ot1,8);
byte2=dec2bin(ot2,8);
byte3=dec2bin(ot3,8);
byte4=dec2bin(ot4,8);
tag_dec=bin2dec(tag); // tag's decimal value
Length_dec=bin2dec(Length); // length's decimal
    value or number of bytes
// display the format
printf("\n          0%d                0%d                0%d
                0%d           0%d             0%d\n",tag_dec,
    Length_dec,ot1,ot2,ot3,ot4);
printf("      | %s | %s |%s |   %s |   %s |   %s|\n",tag,
    Length,byte1,byte2,byte3,byte4);
printf("         Tag          Length         Value        Value
          Value           Value\n");
printf("       (ObjectId)    (4 bytes)      (%d)          (%d)
            (%d)                (%d)\n",ot1,ot2,ot3,ot4);
printf("                                   |——————%s (iso.org
    .dod.internet)————————|",ObjectIdentifier);
```

Scilab code Exa 28.4 Define IPAddress

```
clear;
```

```
clc;
disp("—————————Example 28.4—————————")
tag="01000000"; // IPAddress tag
Length="00000100"; // 4 bytes
IPAddress="131.21.14.8"; // value
tag_dec=bin2dec(tag); // tag's decimal value
tag_hex=dec2hex(tag_dec); // tag's hex value
Length_dec=bin2dec(Length); // length's decimal value or number of bytes
ip1=131;
ip2=21;
ip3=14;
ip4=8;
// convert the bytes to binary
byte1=dec2bin(ip1,8);
byte2=dec2bin(ip2,8);
byte3=dec2bin(ip3,8);
byte4=dec2bin(ip4,8);
// connvert bytes to hexadecimal
h1=dec2hex(ip1);
h2=dec2hex(ip2);
h3=dec2hex(ip3);
h4=dec2hex(ip4);
// display the format
printf("\n          %s            0%d            %s            %s            0%s            0%s\n",tag_hex,Length_dec,h1,h2,h3,h4);
printf("     | %s | %s |%s |  %s |  %s |  %s|\n",tag,Length,byte1,byte2,byte3,byte4);
printf("        Tag          Length          Value          Value          Value          Value\n");
printf("     (IPAddress)   (4 bytes)    (%d)         (%d)         (%d)           (%d)\n",ip1,ip2,ip3,ip4);
printf("                                   |————————————%s————————————|",IPAddress);
```

Scilab code Exa 28.5 SNMP message to retrieve UDP datagrams

```
clear;
clc;
disp("−−−−−−−−−−−−−−−−Example 28.5−−−−−−−−−−−−−−−−−")
//example explanation
printf("In this example, a manager station (SNMP
    client) uses the GetRequest message to retrieve
    the number of UDP datagrams that a\nrouter has
    received.\n\nThere is only one VarBind entity.
    The corresponding MIB variable related to this
    information is udpInDatagrams with the object
    identifier\n1.3.6.1.2.1.7.1.0. The manager wants
    to retrieve a value (not to store a value), so
    the value defines a null entity.\n\nThe VarBind
    list has only one VarBind. The variable is of
    type 06 and length 09.\nThe value is of type 05
    and length 00. The whole VarBind is a sequence of
     length 0D (13).\nThe VarBind list is also a
    sequence of length 0F (15). The GetRequest PDU is
     of length 1D (29).\n\nThere are three OCTET
    STRINGs related to the security parameter,
    security model, and flags. Then we have two
    integers defining\nmaximum size (1024) and
    message ID (64). The header is a sequence of
    length 12, which we left blank for simplicity.
    There is one integer,\nversion (version 3). The
    whole message is a sequence of 52 bytes.\n\nThe
    figure shows the actual message sent by the
    manager station (client) to the agent (server).It
     shows the conceptual view of\nthe packet and the
     hierarchical nature of sequences.White boxes and
     colored boxes are used for the sequences and a
    gray one for the PDU.");
```

```
// display the figure
clf();
xname("————————————————Example 28.5——————————————————");
xfrect(.04,.5,.5,.3);
a=gce();
a.background=color('gray')
xfrect(.06,.33,.4,.1);
a=gce();
a.background=color('white');
xfrect(.08,.3,.3,.06);
a=gce();
a.background=color('gray')
xrects([0 .02 .02 .04 .06 .08;1 .9 .55 .5 .33 .3;.7 .1 .6 .5 .4 .3;.9 .04 .4 .3 .1 .06]);
xset("font size",2.5)
xstring(.72,.55,"Whole message a sequence of 52 bytes");
xstring(0.01,.96,"02 01 03 INTEGER, version");
xstring(.023,.88,"__   __   ...   __          Header, sequence of length 12, not shown");
xstring(0.01,.81,"02 01 40                    Two");
xstring(0.01,.78,"02 02 04 00            INTEGERs");
xstring(0.01,.7,"04 01 00                    Three");
xstring(0.01,.67,"04 00                             OCTET STRINGs");
xstring(0.01,.64,"04 00");
xstring(0.04,.51,"A0 1D");
xstring(0.042,.47,"02 04 00 01 06 11 Three");
xstring(0.042,.44,"02 01 00                                   INTEGERs");
xstring(0.042,.41,"02 01 00");
xstring(0.082,.27,"06 09 01 03 06 01 02 07 01 00                                   VarBind");
xstring(0.082,.24,"05 00");
xstring(0.46,.29,"VarBind list");
xstring(0.54,.3,"length 29");
```

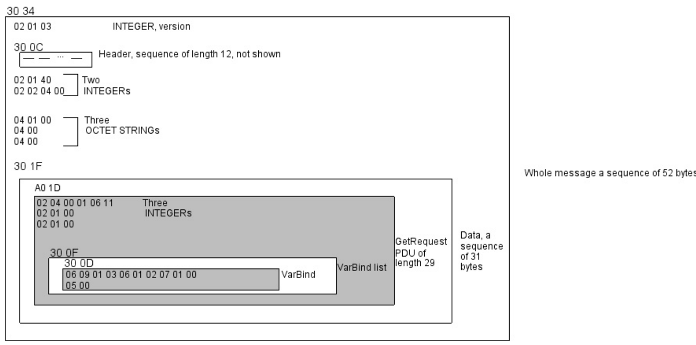

Figure 28.1: SNMP message to retrieve UDP datagrams

```
xstring(0.54,.33,"PDU of");
xstring(0.54,.36,"GetRequest");
xstring(0.63,.38,"Data, a");
xstring(0.63,.35,"sequence");
xstring(0.63,.32,"of 31");
xstring(0.63,.29,"bytes");
xpoly([.08 .1],[.84 .84]);
xpoly([.08 .1],[.78 .78]);
xpoly([.1 .1],[.78 .84]);
xpoly([.08 .1],[.72 .72]);
xpoly([.08 .1],[.64 .64]);
xpoly([.1 .1],[.64 .72]);
xset("font size",3.5);
xlfont("Monospaced",10,%t,%t);
xstring(0,1,"30 34");
xstring(0.01,.9,"30 0C");
xstring(0.01,.57,"30 1F");
xstring(0.06,.33,"30 0F");
xstring(0.08,.3,"30 0D");
```

Chapter 30

Cryptography

Scilab code Exa 30.1 Monoalphabetic cipher

```
clear;
clc;
disp("---------------Example 30.1-----------------")
plaintext=['H' 'E' 'L' 'L' 'O'];
ciphertext=['K' 'H' 'O' 'O' 'R'];
L1=ciphertext(3); // 1st L's encryption
L2=ciphertext(4); // 2nd L's encryption
// display appropriate result
if(L1==L2)
    printf("The cipher is probably monoalphabetic
       because both occurrences of Ls are encrypted
       as %ss.",L1);
else
    printf("The cipher is polyalphabetic because 1st
        occurrence of L is encrypted as %s and 2nd
       occurrence of L is encrypted as %s.",L1,L2);
end
```

Scilab code Exa 30.2 Polyalphabetic cipher

```
clear;
clc;
disp("——————————————Example 30.2————————————————")
plaintext=['H' 'E' 'L' 'L' 'O'];
ciphertext=['A' 'B' 'N' 'Z' 'F'];
L1=ciphertext(3); // 1st L's encryption
L2=ciphertext(4); // 2nd L's encryption
// display appropriate result
if(L1==L2)
    printf("The cipher is probably monoalphabetic because both occurrences of Ls are encrypted as %ss.",L1);
else
    printf("The cipher is not monoalphabetic because each occurrence of L is encrypted by a different character. The first L is encrypted as %s; the second as %s.",L1,L2);
end
```

Scilab code Exa 30.3 Shiftkey 15 encryption

```
clear;
clc;
disp("——————————————Example 30.3————————————————")
message=['H' 'E' 'L' 'L' 'O'];
key=15; // shift down key
alphabet=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P' 'Q' 'R' 'S' 'T' 'U' 'V' 'W' 'X' 'Y' 'Z'];
ciphertext="";

for k=1:5 // encrypt each character in the message
    for i=1:26
        if(message(k)==alphabet(i)) // find the index of the character in the alphabet
```

```
            array
                break;
            end
        end
        temp=i+15;  // shift down by 15 towards end of the alphabet
        if(temp > = 26)
            a=modulo(temp,26); // wrap around thhe alphabet if its greater than 26
        else
            a=temp;
        end
        ciphertext=ciphertext+alphabet(a); // form the ciphertext
end
printf("The cipher text is %s.",ciphertext); // display the result
```

Scilab code Exa 30.4 Shiftkey 15 decryption

```
clear;
clc;
disp("—————————————Example 30.4———————————————")
ciphertext=['W' 'T' 'A' 'A' 'D'];
key=15; // shift up key
plaintext="";
alphabet=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P' 'Q' 'R' 'S' 'T' 'U' 'V' 'W' 'X' 'Y' 'Z'];

for k=1:5  // decrypt each character in the ciphertext
    for i=1:26
        if(ciphertext(k)==alphabet(i))  // find the index of the character in the alphabet
```

```
            array
            break;
        end
    end
    temp=i-15; // shift up by 15 towards the
       beginning of the alphabet
    if(temp < = 0)
        a=26+temp;
        if(a>26)
            a=modulo(a,26); // wrapping around the
               alphabet
        end

    else
        a=temp;
    end
    plaintext=plaintext+alphabet(a); // form the
       plain text
end
printf("The plain text is %s.",plaintext);  //
   display the result
```

Scilab code Exa 30.5 Encryption of message

```
clear;
clc;
disp("—————————————Example 30.5———————————————")
message=['H' 'E' 'L' 'L' 'O' ' ' 'M' 'Y' ' ' 'D' 'E'
    'A' 'R']; // HELLO MY DEAR
l=size(message,'c'); // length of message
ns="";
ciphertext="";
for i=1:l
    if(message(i)==' ')  // remove the spaces
        continue;
```

```
    else
        ns=ns+message(i); // form message with no spaces
    end

end

block=strsplit(ns,[4 8]);  // split the message into blocks of 4
nz=4-length(block(3));  // number of 'Z's to be added to the last block

for i=1:nz
    block(3)=block(3)+'Z';  // adding 'Z's to the last block
end
f=[];
for i=1:size(block,'r') // for each block
    c=strsplit(block(i));
    f(1)=c(2); // move the character at position 2 to position 1
    f(2)=c(4); // move the character at position 4 to position 2
    f(3)=c(1); // move the character at position 1 to position 3
    f(4)=c(3); // move the character at position 3 to position 4
    str=f(1)+f(2)+f(3)+f(4); // new block
    ciphertext=ciphertext+str; // form the ciphertext
end
// display the result
printf("The 3 blocks are %s , %s and %s.",block(1),block(2),block(3));
printf("\n\nThe ciphertext is %s .",ciphertext)
```

Scilab code Exa 30.6 Decryption of message

```
clear;
clc;
disp("————————————Example 30.6————————————")
ciphertext="ELHLMDOYAZER";
block=strsplit(ciphertext,[4 8]);  // split into
   blocks

f=[];
for i=1:size(block,'r') // for each block
    c=strsplit(block(i));
    f(2)=c(1); // move the character at position 1
       to position 2
    f(4)=c(2); // move the character at position 2
       to position 4
    f(1)=c(3); // move the character at position 3
       to position 1
    f(3)=c(4); // move the character at position 4
       to position 3
    str=f(1)+f(2)+f(3)+f(4);
    block(i)=str;// new block
end
printf("The 3 blocks are %s , %s and %s.",block(1),
   block(2),block(3));
nz=0;
b3=strsplit(block(3));
for i=1:4
    if(b3(i)=='Z');
        nz=nz+1; // number of 'Z's in the last block
    end

end

```

```
  f=strsplit(block(3),size(b3,'r')-nz);  // remove
     the 'Z's in the last block
block(3)=f(1);  // new last block

text=block(1)+block(2)+block(3);
sp=strsplit(text,[5 7]);
plaintext=sp(1)+" "+sp(2)+" "+sp(3); // add the
   spaces
printf("\n\nThe message is %s.",plaintext) //
   display the result
```

Scilab code Exa 30.7 RSA plaintext 5

```
clear;
clc;
disp("−−−−−−−−−−−−−−−−Example 30.7−−−−−−−−−−−−−−−−−−")
p=7;
q=11;
n=p*q; // formula
phi=(p-1)*(q-1); // formula
e=13;
d=1; // not actual 'd' value; it has to be computed
t=1;
P=5;
plaintext=P;
while t==1 do  // compute d such that d*e = 1 mod n
    if(modulo(e*d,phi)== 1)
        t=0;
    else
        d=d+1;
    end
end
 // encryption by Alice
C=modulo(P^e,n); // formula
printf("Alice sends the plaintext %d to Bob. She
```

```
   uses the public key %d to encrypt %d.\nBob
   receives the ciphertext %d and uses the private
   key %d to decipher the ciphertext.",P,e,P,C,d);
   // display the result

// decryption by Bob
P=modulo(C^d,n); // formula
printf("\n\nThe plaintext %d sent by Alice is
   received as plaintext %d by Bob.",plaintext,
   plaintext); // display the result
```

Scilab code Exa 30.8 RSA message NO

```
clear;
clc;
disp("—————————————Example 30.8—————————————")
p=397;
q=401;
n=p*q; // formula
phi=(p-1)*(q-1); // formula
e=343;
d=1;  // not actual 'd' value; it has to be computed
message=['N' 'O']; // NO
t=1;
alphabet=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K
   ' 'L' 'M' 'N' 'O' 'P' 'Q' 'R' 'S' 'T' 'U' 'V' 'W'
    'X' 'Y' 'Z'];
// Encryption process by Ted
while t==1 do  // compute d such that d*e = 1 mod n
    if(modulo(e*d,phi)== 1)
        t=0;
    else
        d=d+1;
    end
end
```

```
l=size(message,'c'); // length of the message
c=[];
for k=1:l // determine the code for each character
    in the message
    for i=1:26
        if(message(k)==alphabet(i))
            c(k)=i-1;  // compute the code
            break;
        end
    end
end
plaintext=c(1)*100+c(2); // form the plaintext

//C=modulo((plaintext)^e,n) -- formula to find the
    ciphertext
ciphertext=33677; // from calculation
printf("\nThe plaintext is %d and the ciphertext
    sent by Ted is %d.\n",plaintext,ciphertext); //
    display the result

//  Decryption by Jennifer

//P=modulo((ciphertext)^d,n);  -- formula to find
    the plaintext
P=1314; // the plaintext that is computed
// separate the codes for each character
c(2)=modulo(P,100);
c(1)=floor(P/100);
d_message="";  // deciphered message
for k=1:l // find the corresponding letter for each
    code
    for i=1:26
        if(i==c(k)+1)
            d_message=d_message+alphabet(i);  //
               form the deciphered message
        end

    end
```

```
end
printf("\nJennifer deciphers the ciphertext %d as
    the plaintext %d and decodes it as the message %s
    .",ciphertext,P,d_message) // display the result
```

Scilab code Exa 30.9 RSA realistic example

```
clear;
clc;
disp("————————————Example 30.9——————————————")
p
    =961303453135835045741915812806154279093098455949962158225831508
     // 159 digit number
q
    =1206019195723144691827679420445089600155592505463703393606179832
     // 160 digit number
n=p*q; // formula
phi=(p-1)*(q-1);// formula
e=35535;
d="
    58008302860037763936093661289677917594669062089650962180422866111
    "; // compute d such that d*e = 1 mod n ( very
    huge value to compute)
alphabet=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K
    ' 'L' 'M' 'N' 'O' 'P' 'Q' 'R' 'S' 'T' 'U' 'V' 'W'
     'X' 'Y' 'Z' ' '];
p_str="
    9613034531358350457419158128061542790930984559499621582258315087
    ";
q_str="
    1206019195723144691827679420445089600155592505463703393606179832
    ";
n_str="11593504
    I7396761496889250986461588752377145737545414477548552613761478854
    "; // 309 digits
```

```
phi_str="
    115935041739676149688925098646158875237714573754541447754855261376
    "; // 309 digits
// encoding by Alice
message=["T" "H" "I" "S" " " "I" "S" " " "A" " " "T"
     "E" "S" "T"]; //THIS IS A TEST
l=size(message,'c'); // length of the message
c=[];
plaintext="";
for k=1:l // determine the code for each character
    in the message
      for i=1:27
          if(message(k)==alphabet(i))
              c(k)=string(i-1);  // compute the code
              if(length(c(k))==1)
                  c(k)='0'+c(k);
              end
              break;
          end
      end
end
for i=1:l
      plaintext=plaintext+c(i); // form the plaintext
          , code 26 is for space
end
P=plaintext;
C="
    475309123646226827206365550610545180942371796070491716523239243305
    "; // C= P^e -- ciphertext ( very huge value to
    compute)
printf("p = %s\n\nq = %s\n\nn = %s\n\nphi = %s\n\ne
   = %d\n\nd = %s\n\n",p_str,q_str,n_str,phi_str,e,d
   );
printf("The plaintext is %s and the ciphertext sent
    by Alice is\nC = %s\n\n",P,C);
// Decoding by Bob
P="19070818260818260026190418 19"; // P=C^d --
    plaintext (very huge to compute)
```

```
d_message="";
c=strsplit(P,[2 4 6 8 10 12 14 16 18 20 22 24 26]);
    // separate the codes for each character
for k=1:l // find the corresponding letter for each
    code
    for i=1:27
        a=string(i-1);
        b=strsplit(c(k));
        if(b(1)=='0')
            c(k)=b(2);
        end
        if(a==c(k))
            d_message=d_message+alphabet(i);  //
                form the deciphered message
            break;
        end
    end
end
printf("\nBob recovers the plaintext %s and decodes
    it as the message %s.",P,d_message);
```

Scilab code Exa 30.10 Diffie Hellman method

```
clear;
clc;
disp("————————————Example 30.10—————————————")
g=7;
p=23;
printf("\nThe steps are as follows:\n\n");
x=3;
y=6;
R1=modulo(g^x,p); // formuula
R2=modulo(g^y,p); // formula
printf("1) Alice chooses x = %d and calculates R1 =
    %d.\n\n2) Bob chooses y = %d and calculates R2 =
```

```
%d.\n\n3) Alice sends the number %d to Bob.\n\n4) Bob sends the number %d to Alice.\n\n",x,R1,y,R2,R1,R2);

K_Alice=modulo((R2)^x,p); // K calculated by Alice
K_Bob=modulo((R1)^y,p); // K calculated by Bob
K=modulo(g^(x*y),p);  // The symmetric (shared) key in the Diffie-Hellman protocol
printf("5) Alice calculates the symmetric key K =%d.\n\n6) Bob calculates the symmetric key K = %d.\n\n",K_Alice,K_Bob);

// check if the key values are equal and display appropriate result
if( K_Alice == K_Bob )
    printf("The value of K is the same for both Alice and Bob. The symmetric key K = %d.",K);
else
    printf("The value of K is not the same for both Alice and Bob. It is %d for Alice and %d for Bob.",K_Alice,K_Bob);
end
```

Chapter 31

Network Security

Scilab code Exa 31.1 Lossless compression method

```
clear;
clc;
disp("—————————————Example 31.1——————————————")
// display the example
printf("We cannot. A lossless compression method
   creates a compressed message that is reversible.\
   nThe compressed message can be uncompressed to
   get the original one.");
```

Scilab code Exa 31.2 Checksum method

```
clear;
clc;
disp("—————————————Example 31.2——————————————")
// display the example
printf("Yes. A checksum function is not reversible;
   it meets the first criterion. However, it does
   not meet the other criteria.");
```