

Scilab Textbook Companion for  
Data Communications And Networking  
by B. A. Forouzan<sup>1</sup>

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

August 17, 2013

<sup>1</sup>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	5
2 Network Models	12
3 Data and Signals	23
4 Digital Transmission	55
5 Analog Transmission	67
6 Bandwidth Utilization Multiplexing and Spreading	77
8 Switching	94
10 Error Detection and Correction	100
11 Data link control	132
12 Multiple Access	145
13 Wired LANs Ethernet	153
16 Wireless WANs Cellular Telephone and Satellite Networks	157
17 SONET SDH	159
19 Network layer Logical addressing	162
20 Network Layer Internet Protocol	175

<b>21 Network Layer Address Mapping Error Reporting and Multicasting</b>	<b>184</b>
<b>22 Network layer Delivery Forwarding and Routing</b>	<b>197</b>
<b>23 Process to Process Delivery UDP TCp and SCTP</b>	<b>211</b>
<b>26 Remote Logging Electronic Mail and File Transfer</b>	<b>215</b>
<b>27 WWW and HTTP</b>	<b>221</b>
<b>28 Network Management SNMP</b>	<b>226</b>
<b>30 Cryptography</b>	<b>233</b>
<b>31 Network Security</b>	<b>246</b>

# List of Scilab Codes

Exa 2.1	Bus topology LAN . . . . .	12
Exa 2.2	physical address example . . . . .	15
Exa 2.3	LAN and router communication . . . . .	15
Exa 2.4	Two computers communicating . . . . .	17
Exa 2.5	single number portaddress . . . . .	21
Exa 3.1	Power in houses . . . . .	23
Exa 3.2	Battery voltage . . . . .	23
Exa 3.3	House power period . . . . .	24
Exa 3.4	Period in microseconds . . . . .	24
Exa 3.5	Frequency from period . . . . .	24
Exa 3.6	Calculation of phase . . . . .	25
Exa 3.7	Time and frequency domains . . . . .	25
Exa 3.8	Periodic composite signal . . . . .	27
Exa 3.9	Nonperiodic composite signal . . . . .	28
Exa 3.10	Bandwidth spectrum . . . . .	30
Exa 3.11	Bandwidth spectrum 2 . . . . .	33
Exa 3.12	Frequency domain of signal . . . . .	33
Exa 3.13	AM Radio Bandwidth . . . . .	35
Exa 3.14	FM Radio Bandwidth . . . . .	36
Exa 3.15	Black and white TV . . . . .	36
Exa 3.16	Bits per level . . . . .	37
Exa 3.17	Bits per level 2 . . . . .	37
Exa 3.18	Download text document . . . . .	38
Exa 3.19	Bitrate calculation . . . . .	39
Exa 3.20	HDTV bitrate . . . . .	39
Exa 3.21	LAN . . . . .	39
Exa 3.22	Base band transmission . . . . .	40
Exa 3.23	Maximum bitrate calculation . . . . .	41

Exa 3.24	Broadband transmission example . . . . .	41
Exa 3.25	Digital cellular telephone . . . . .	42
Exa 3.26	Attenuation calculation . . . . .	43
Exa 3.27	Amplification calculation . . . . .	43
Exa 3.28	Resultant decibel calculation . . . . .	44
Exa 3.29	Pm from dBm . . . . .	44
Exa 3.30	dB per km . . . . .	44
Exa 3.31	SNR and SNRdB . . . . .	45
Exa 3.32	Noiseless channel SNR and SNRdB . . . . .	45
Exa 3.33	Nyquist and baseband transmission . . . . .	46
Exa 3.34	Noiseless channel bitrate . . . . .	46
Exa 3.35	Noiseless channel bitrate 2 . . . . .	47
Exa 3.36	Noiseless channel signal levels . . . . .	47
Exa 3.37	C for extremely noisy channel . . . . .	48
Exa 3.38	C for noisy channel . . . . .	48
Exa 3.39	C using SNRdB . . . . .	49
Exa 3.40	C formula simplification . . . . .	49
Exa 3.41	Shannon and Nyquist formula . . . . .	50
Exa 3.42	Modulation using modem . . . . .	50
Exa 3.43	Increasing bandwidth of line . . . . .	50
Exa 3.44	Throughput of network . . . . .	51
Exa 3.45	Propagation time calculation . . . . .	51
Exa 3.46	Propagation and transmission time . . . . .	52
Exa 3.47	Propagation and transmission time 2 . . . . .	53
Exa 3.48	Bandwidth delay product . . . . .	53
Exa 4.1	average baud rate . . . . .	55
Exa 4.2	Nyquist formula equivalence . . . . .	55
Exa 4.3	Extra bits calculation . . . . .	56
Exa 4.4	avg baud and min bandwidth . . . . .	56
Exa 4.5	Block coding minimum bandwidth . . . . .	57
Exa 4.6	sampling and recovery . . . . .	58
Exa 4.7	Sampling of clock . . . . .	61
Exa 4.8	wheel rotation undersampling . . . . .	63
Exa 4.9	telephone sampling rate . . . . .	63
Exa 4.10	minimum sampling rate . . . . .	64
Exa 4.11	bandpass sampling rate . . . . .	64
Exa 4.12	SNR db formula . . . . .	65
Exa 4.13	telephone bits per sample . . . . .	65

Exa 4.14	human voice digitization . . . . .	65
Exa 4.15	digital minimum bandwidth . . . . .	66
Exa 5.1	analog signal bit rate . . . . .	67
Exa 5.2	data and signal elements . . . . .	67
Exa 5.3	ASK $f_c$ and bitrate . . . . .	68
Exa 5.4	Full duplex ASK . . . . .	68
Exa 5.5	FSK $f_c$ and bitrate . . . . .	70
Exa 5.6	bandwidth of MFSK . . . . .	71
Exa 5.7	bandwidth of QPSK . . . . .	73
Exa 5.8	Three constellations diagrams . . . . .	73
Exa 6.1	FDM configuration . . . . .	77
Exa 6.2	bandwidth with guards . . . . .	80
Exa 6.3	satellite channel FDM . . . . .	82
Exa 6.4	Advanced MobilePhone System . . . . .	83
Exa 6.5	three durations calculation . . . . .	84
Exa 6.6	Synchronous TDM . . . . .	85
Exa 6.7	Four connections multiplexing . . . . .	86
Exa 6.8	Four channels TDM . . . . .	86
Exa 6.9	2 bits timeslot . . . . .	88
Exa 6.10	4 sources interleaving . . . . .	91
Exa 6.11	2 channels multiplexing . . . . .	92
Exa 8.1	circuit switched network for telephones . . . . .	94
Exa 8.2	circuit switched network of computers . . . . .	95
Exa 8.3	Design a three stage switch . . . . .	97
Exa 8.4	Clos criteria switch . . . . .	98
Exa 10.1	block coding error . . . . .	100
Exa 10.2	2 bit dataword . . . . .	100
Exa 10.3	3 redundant bits codeword . . . . .	102
Exa 10.4	find Hamming distance . . . . .	103
Exa 10.5	minimum Hamming distance . . . . .	104
Exa 10.6	minimum Hamming distance 2 . . . . .	106
Exa 10.7	error detection and correction . . . . .	107
Exa 10.8	block code $d_{min}$ . . . . .	107
Exa 10.9	Hamming distance $d_{min} 4$ . . . . .	108
Exa 10.10	linear block codes . . . . .	108
Exa 10.11	$d_{min}$ calculation . . . . .	111
Exa 10.12	some transmission scenarios . . . . .	112
Exa 10.13	path of three datawords . . . . .	114



Exa 10.14	calculate k and n . . . . .	120
Exa 10.15	single bit error . . . . .	121
Exa 10.16	two isolated single bit errors . . . . .	122
Exa 10.17	burst error generators . . . . .	123
Exa 10.18	sum error detection . . . . .	125
Exa 10.19	checksum error detection . . . . .	125
Exa 10.20	21 1s complement . . . . .	126
Exa 10.21	negative 1s complement . . . . .	127
Exa 10.22	complement checksum error . . . . .	127
Exa 10.23	Forouzan text checksum . . . . .	129
Exa 11.1	Simplest protocol . . . . .	132
Exa 11.2	Stop and wait . . . . .	133
Exa 11.3	Stop and wait ARQ . . . . .	135
Exa 11.4	Bandwidth delay product . . . . .	136
Exa 11.5	link utilization percentage . . . . .	137
Exa 11.6	Go Back N . . . . .	137
Exa 11.7	loss of frame . . . . .	138
Exa 11.8	Selective Repeat ARQ . . . . .	139
Exa 11.9	Connection and disconnection . . . . .	141
Exa 11.10	Piggybacking without Error . . . . .	141
Exa 11.11	Piggybacking with Error . . . . .	142
Exa 11.12	Network layer packet . . . . .	143
Exa 12.1	ALOHA TB calculation . . . . .	145
Exa 12.2	Collision free ALOHA . . . . .	146
Exa 12.3	Pure ALOHA throughput . . . . .	147
Exa 12.4	Slotted ALOHA throughput . . . . .	148
Exa 12.5	Minimum frame size . . . . .	149
Exa 12.6	Chips for network . . . . .	150
Exa 12.7	Number of sequences . . . . .	151
Exa 12.8	Proof . . . . .	151
Exa 13.1	Define the type . . . . .	153
Exa 13.2	Sending the address . . . . .	154
Exa 16.1	Period of moon . . . . .	157
Exa 16.2	Period of geostationary satellite . . . . .	158
Exa 17.1	Datarate of STS1 . . . . .	159
Exa 17.2	Datarate of STS3 . . . . .	159
Exa 17.3	Duration of STSs . . . . .	160
Exa 17.4	User datarate STS1 . . . . .	160

Exa 17.5	H1 and H2 . . . . .	161
Exa 19.1	Dotted decimal notation . . . . .	162
Exa 19.2	Binary notation . . . . .	163
Exa 19.3	Find the error . . . . .	163
Exa 19.4	Find the class . . . . .	164
Exa 19.5	block of addresses . . . . .	166
Exa 19.6	Find first address . . . . .	167
Exa 19.7	Find last address . . . . .	168
Exa 19.8	number of addresses . . . . .	169
Exa 19.9	addresses using mask . . . . .	169
Exa 19.10	design sub blocks . . . . .	172
Exa 19.11	find original address . . . . .	173
Exa 20.1	Acceptance of packet . . . . .	175
Exa 20.2	Bytes of options . . . . .	176
Exa 20.3	Bytes of data . . . . .	176
Exa 20.4	Time and protocol . . . . .	177
Exa 20.5	M equals 0 . . . . .	178
Exa 20.6	M equals 1 . . . . .	178
Exa 20.7	M and offset . . . . .	179
Exa 20.8	First byte number . . . . .	180
Exa 20.9	First and last byte number . . . . .	180
Exa 20.10	IPv4 header checksum . . . . .	181
Exa 21.1	ARP request and reply . . . . .	184
Exa 21.2	Echo request message checksum . . . . .	186
Exa 21.3	ping program . . . . .	188
Exa 21.4	trace route program . . . . .	189
Exa 21.5	trace longer route . . . . .	190
Exa 21.6	report messages sequence . . . . .	191
Exa 21.7	Ethernet multicast physical address . . . . .	192
Exa 21.8	Ethernet multicast address . . . . .	193
Exa 21.9	netstat nra . . . . .	195
Exa 22.1	Routing Table for router . . . . .	197
Exa 22.2	Forwarding process 1 . . . . .	198
Exa 22.3	Forwarding process 2 . . . . .	200
Exa 22.4	Forwarding process 3 . . . . .	202
Exa 22.5	Hierarchical routing . . . . .	205
Exa 22.6	netstat and ifconfig . . . . .	209
Exa 23.1	grep etc services . . . . .	211

Exa 23.2	Checksum with padding . . . . .	212
Exa 23.3	Segment sequence number . . . . .	212
Exa 23.4	Value of rwnd . . . . .	213
Exa 23.5	Window for host . . . . .	214
Exa 23.6	Unrealistic sliding window . . . . .	214
Exa 26.1	Option negotiation . . . . .	215
Exa 26.2	Suboption negotiation . . . . .	216
Exa 26.3	Email using SMTP . . . . .	217
Exa 26.4	FTP session . . . . .	218
Exa 26.5	Anonymous FTP . . . . .	220
Exa 27.1	Retrieving document using GET . . . . .	221
Exa 27.2	Send data using POST . . . . .	222
Exa 27.3	Connect to server using TELNET . . . . .	224
Exa 28.1	Define INTEGER 14 . . . . .	226
Exa 28.2	Define OCTETSTRING HI . . . . .	227
Exa 28.3	Define ObjectIdentifier . . . . .	227
Exa 28.4	Define IPAddress . . . . .	228
Exa 28.5	SNMP message to retrieve UDP datagrams . . . . .	230
Exa 30.1	Monoalphabetic cipher . . . . .	233
Exa 30.2	Polyalphabetic cipher . . . . .	233
Exa 30.3	Shiftkey 15 encryption . . . . .	234
Exa 30.4	Shiftkey 15 decryption . . . . .	235
Exa 30.5	Encryption of message . . . . .	236
Exa 30.6	Decryption of message . . . . .	238
Exa 30.7	RSA plaintext 5 . . . . .	239
Exa 30.8	RSA message NO . . . . .	240
Exa 30.9	RSA realistic example . . . . .	242
Exa 30.10	Diffie Hellman method . . . . .	244
Exa 31.1	Lossless compression method . . . . .	246
Exa 31.2	Checksum method . . . . .	246

# List of Figures

2.1	Bus topology LAN . . . . .	15
2.2	Two computers communicating . . . . .	22
3.1	Time and frequency domains . . . . .	27
3.2	Periodic composite signal . . . . .	28
3.3	Nonperiodic composite signal . . . . .	31
3.4	Bandwidth spectrum . . . . .	32
3.5	Bandwidth spectrum 2 . . . . .	34
3.6	Frequency domain of signal . . . . .	35
4.1	sampling and recovery . . . . .	61
5.1	Full duplex ASK . . . . .	70
5.2	bandwidth of MFSK . . . . .	73
5.3	Three constellations diagrams . . . . .	76
6.1	FDM configuration . . . . .	80
6.2	bandwidth with guards . . . . .	82
6.3	satellite channel FDM . . . . .	84
6.4	Four channels TDM . . . . .	89
6.5	2 bits timeslot . . . . .	91
8.1	circuit switched network of computers . . . . .	96
11.1	Simplest protocol . . . . .	134
11.2	Stop and wait . . . . .	136
20.1	IPv4 header checksum . . . . .	183
28.1	SNMP message to retrieve UDP datagrams . . . . .	232

# Chapter 2

## Network Models

Scilab code Exa 2.1 Bus topology LAN

```
1 clear all;
2 clc;
3 disp("-----Example 2.1-----")
4 // example explanation
5 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.")

```

;
6 sender_address="10";
7 reciever_address="87";
8 clf();
9 xname("-----Example 2.1-----");
10 // display the figure
11 xset("thickness",2.5);
12 xpoly([.049 .86],[.48 .48]);
13 xpoly ([.35 .35],[.73 .48]);
14 xpoly ([.55 .55],[.73 .48]);
15 xset("thickness",1);
16 xset("font size",2);
17 xstring(.36,.6,"Destination address does not");
18 xstring(.36,.56,"match;the packet is dropped");
19 xstring(.25,.63,"Trailer");
20 xstring(.001,.63,"Destination address");
21 xstring(.001,.5,"Source address");
22 xpoly([.1 .11],[.52 .55]);
23 xpoly([.22 .25],[.6 .63]);
24 xpoly([.05 .08],[.63 .6]);
25 xset("font size",4);
26 xstring(0.1,.75,["10 - Sender"]);
27 xstring(0.7,.75,["87 - Reciever"]);
28 xstring(.32,.75,"Station 28");
29 xstring(.52, .75,"Station 53");
30 xstring(.44,.42,"LAN");
31 xrect(.13,.7,.07,.05);
32 xstringb(.13,.65,["Data"],.07,.05);

```

```

33 xrect(.73,.7,.07,.05);
34 xstringb(.73,.65,["Data"],.07,.05);
35 xrects([ 0.07 .13 .2;.6 .6 .6;.06 .07 .03;.06 .06
        .06]);
36 xstring(0.07,.55,["87"]);
37 xstring(0.1,.55,["10"]);
38 xstring(0.2,.55,"T2");
39 xrect(.135,.595,.06,.05);
40 xstring(.137,.555,["Data"]);
41 xstring(0.67,.55,["87"]);
42 xstring(0.7,.55,["10"]);
43 xstring(.8,.55,"T2");
44 xrect(.735,.595,.06,.05);
45 xstring(.737,.555,["Data"]);
46 xset("font size",6);
47 xstring(.43,.51,". . .");
48 xfrect(.009,.5,.04,.05);
49 xfrect(.86,.5,.04,.05);
50 xpoly([.17 .17],[.75 .7]);
51 xpoly([.77 .77],[.75 .7]);
52 xpoly([.17 .17],[.65 .6]);
53 xpoly([.77 .77],[.65 .6]);
54 xpoly([.17 .17],[.54 .48]);
55 xpoly([.77 .77],[.54 .48]);
56 xarrows([.74 .8],[.45 .45],.5);
57 xarrows([.76 .76],[.45 .54],.5);
58 xarrows([.34 .4],[.45 .45],.5);
59 xarrows([.36 .36],[.45 .57],.5);
60 xarrows([.54 .6],[.45 .45],.5);
61 xarrows([.56 .56],[.45 .57],.5);
62 xarrows([.14 .2],[.45 .45],.5);
63 xarrows([.14 .08],[.45 .45],.5);
64 xpoly([.16 .16],[.45 .54]);
65 xset("line style",2);
66 xrects([.67 .73 .8;.6 .6 .6;.06 .07 .03;.06 .06
        .06]);

```

---

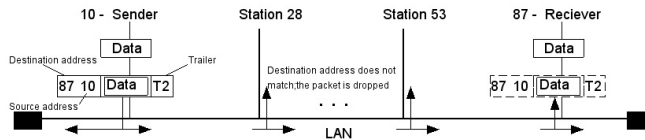


Figure 2.1: Bus topology LAN

### Scilab code Exa 2.2 physical address example

```

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

```

---

### Scilab code Exa 2.3 LAN and router communication

```

1 clear all;
2 clc;

```



```

3 disp("-----Example 2.3-----")
4 // display the example
5 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.");
6 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 .\n\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.\n\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");
7 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\routers
logical address , the router knows that the packet
needs to be forwarded. The router consults its
routing table\and 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");
8 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");
9 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

```

1 clear all;
2 clc;
3 clf();
4 xname("-----Example 2.4-----");
5 // display the figure
6 xset("color",0);
7 xset("font size",3);
8 xrect(-.05,1,.09,.09);
9 xrect(0.06,1,.09,.09);
10 xrect(0.17,1,.09,.09);

```

```

11 xrect(0.72,1,.09,.09);
12 xrect(0.83,1,.09,.09);
13 xstring(-.02,1,["a"]);
14 xstring(0.09,1,["b"]);
15 xstring(.20,1,["c"]);
16 xstring(.75,1,["j"]);
17 xstring(.86,1,["k"]);
18 xstring(0.06,.75,["A - Sender"]);
19 xstring(0.8,.75,["P - Reciever"]);
20 xrect(.06,.7,.07,.05);
21 xstringb(.06,.65,["Data"],.07,.05);
22 xrect(.8,.7,.07,.05);
23 xstringb(.8,.65,["Data"],.07,.05);
24 xstring(.13,.65,["

```

---

Application Layer

---

```

    ])
25 xrects([ 0 .03 .06;.6 .6 .6;.03 .03 .07;.06 .06
    .06]);
26 xrects([.74 .77 .80;.6 .6 .6;.03 .03 .07;.06 .06
    .06]);
27 xstring(0.005,.55,["a"]);
28 xstring(0.035,.55,["j"]);
29 xrect(.065,.595,.06,.05);
30 xstring(.074,.555,["Data"]);
31 xstring(0.745,.55,["a"]);
32 xstring(0.775,.55,["j"]);
33 xrect(.805,.595,.06,.05);
34 xstring(.82,.555,["Data"]);
35 xstring(.13,.55,["

```

---

Transport Layer

---

```

    ])
36 xrect(-.06,.465,.19,.065);
37 xrect(.68,.465,.19,.065);
38 xrects([ -.06 -.03 0;.47 .47 .47;.03 .03 .13;.06 .06
    .06]);

```

```

39 xrects([.68 .71 .74;.47 .47 .47;.03 .03 .13;.06 .06
    .06]);
40 xrects([ 0 .03 .06;.46 .46 .46;.03 .03 .07;.057 .057
    .057]);
41 xrects([.74 .77 .80;.46 .46 .46;.03 .03 .07;.057
    .057 .057]);
42 xstring(0.005,.41,["a"]);
43 xstring(0.035,.41,["j"]);
44 xrect(.065,.455,.06,.05);
45 xstring(.074,.415,["Data"]);
46 xstring(0.745,.41,["a"]);
47 xstring(0.775,.41,["j"]);
48 xrect(.805,.455,.06,.05);
49 xstring(.82,.415,["Data"]);
50 xstring(-.055,.42,["A"]);
51 xstring(-.02,.42,["P"]);
52 xstring(.69,.42,["A"]);
53 xstring(.72,.42,["P"]);
54 xstring(.13,.42,["

```

---

Network Layer

---

```

]");
55 xrect(-.06,.305,.19,.075);
56 xrect(.68,.305,.19,.075);
57 xrects([ -.06 -.03 0;.3 .3 .3;.03 .03 .13;.06 .06
    .06]);
58 xrects([.68 .71 .74;.3 .3 .3;.03 .03 .13;.06 .06
    .06]);
59 xrects([ 0 .03 .06;.297 .297 .297;.03 .03 .07;.057
    .057 .057]);
60 xrects([.74 .77 .80;.297 .297 .297;.03 .03 .07;.057
    .057 .057]);
61 xstring(0.005,.24,["a"]);
62 xstring(0.035,.24,["j"]);
63 xrect(.065,.29,.06,.05);
64 xstring(.074,.245,["Data"]);
65 xstring(0.745,.24,["a"]);
66 xstring(0.775,.24,["j"]);

```

```

67 xrect(.805,.29,.06,.05);
68 xstring(.82,.245,["Data"]);
69 xstring(-.055,.25,["A"]);
70 xstring(-.02,.25,["P"]);
71 xstring(.69,.25,["A"]);
72 xstring(.72,.25,["P"]);
73 xset("color",2);
74 xfrect(-.09,.305,.03,.075);
75 xfrect(.65,.305,.03,.075);
76 xfrect(.13,.305,.03,.075);
77 xfrect(.87,.305,.03,.075);
78 xstring(-.087,.24,["H2"]);
79 xstring(.13,.24,["T2"]);
80 xstring(.655,.24,["H2"]);
81 xstring(.87,.24,["T2"]);
82 xstring(.155,.26,["
    _____Data link
    Layer_____"]);
83 xset("color",0);
84 xstring(.38,.05,["Internet"]);
85 xarc(0.3,.15,.2,.2,0,360*64);
86 xpoly([0.09,0.09],[.75,.7]);
87 xpoly([0.09,0.09],[.65,.6]);
88 xpoly([0.09,0.09],[.54,.47]);
89 xpoly([0.09,0.09],[.4,.3]);
90 xarrows([0.1,.3],[.23 .08]);
91 xpoly([0.84,0.84],[.75,.7]);
92 xpoly([0.84,0.84],[.65,.6]);
93 xpoly([0.84,0.84],[.54,.47]);
94 xpoly([0.84,0.84],[.4,.3]);
95 xpoly([.5 .8],[.08 .23]);
96 xpoly([-0.02 .06],[.91 0.77]);
97 xarrows([.82 .75],[.77 0.91]);
98 disp("_____Example 2.4_____")
99 // display the text
100 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. Process a in the sending computer needs to communicate with process j in the receiving computer. Note that although both computers are using the same application, FTP, the port addresses are different because one is a client program and the other is a server program. To show that data from process a need to be delivered to process j, and not k, the transport layer encapsulates data from the application layer in a packet and adds two port addresses (a and j), source and destination. The packet from the transport layer is then encapsulated in another packet at the network layer with logical source and destination addresses (A and P). Finally, this packet is encapsulated in a frame with the physical source and destination addresses of the next hop. We have not shown the physical addresses because they change from hop to hop inside the cloud designated as the Internet. Note that although physical addresses change from hop to hop, logical and port addresses remain the same from the source to destination.”);

---

#### Scilab code Exa 2.5 single number portaddress

```

1 clear all;
2 clc;
3 disp("-----Example 2.5-----")
4 disp("753 - A 16-bit port address represented as
    one single decimal number.")

```

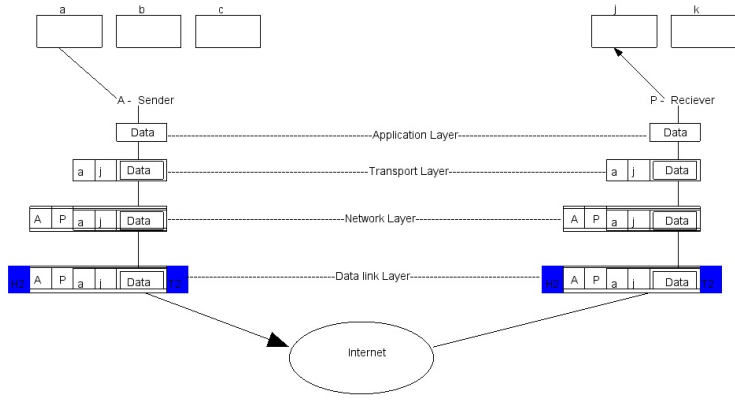


Figure 2.2: Two computers communicating

# Chapter 3

## Data and Signals

Scilab code Exa 3.1 Power in houses

```
1 clear all;
2 clc;
3 disp("-----Example 3.1-----")
4 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

```
1 clear all;
2 clc;
3 disp("-----Example 3.2-----")
4 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

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

---

### Scilab code Exa 3.4 Period in microseconds

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

---

### Scilab code Exa 3.5 Frequency from period

```

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

```

---

#### Scilab code Exa 3.6 Calculation of phase

```

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

```

---

#### Scilab code Exa 3.7 Time and frequency domains

```

1 clear all;
2 clc;
3 clf();
4 xname("-----Example 3.7-----");
5 //display the figure
6 subplot(121) // time domain plot
7 a1=gca();

```

```

8 a1.y_label.text="Amplitude"; // y-axis
9 a1.y_label.font_style = 3;
10 a1.y_label.font_size = 4;
11 a1.y_label.foreground = 3;
12 a1.y_location = "left";
13 a1.x_label.text="Time"; // x-axis
14 a1.x_label.font_style = 3;
15 a1.x_label.font_size = 4;
16 a1.x_label.foreground = 3;
17 a1.x_location = "middle";
18 a1.data_bounds=[0,-15;1,15];
19 x=[0:.000125:1];
20 // sine waves to be plotted
21 plot2d(x,10*sin(8*x));
22 plot2d(x,5*sin(16*x));
23 plot2d(x,15*sin(%pi/2 - 0*x));
24 xarrows([.5 .01],[ -12 -12],5);
25 xarrows([.5 .99],[ -12 -12],5);
26 xpoly([1 1],[ -3 -12.5]);
27 xset("font size",3)
28 xstring(.5,-11,"1s")
29 xset("font size",2)
30 xstring(.02,-13.5,"a. Time-domain representation of
    three sine waves with frequencies 0, 8, and 16")
31 subplot(122) // frequency domain plot
32 a1=gca();
33 xarrows([0 1],[.5 .5],.5)
34 xarrows([0 0],[.5 1],.5)
35 xset("font size",3)
36 xstring(-.01,1,"Amplitude") // y-axis
37 xstring(1,.45,"Frequency") // x-axis
38 xstring(.02,.4,"b. Frequency-domain representation
    of the same three signals")
39 xstring(0,.45,"0
    8
    16")

40 xstring(-.05,.7,"5")
41 xstring(-.05,.8,"10")

```

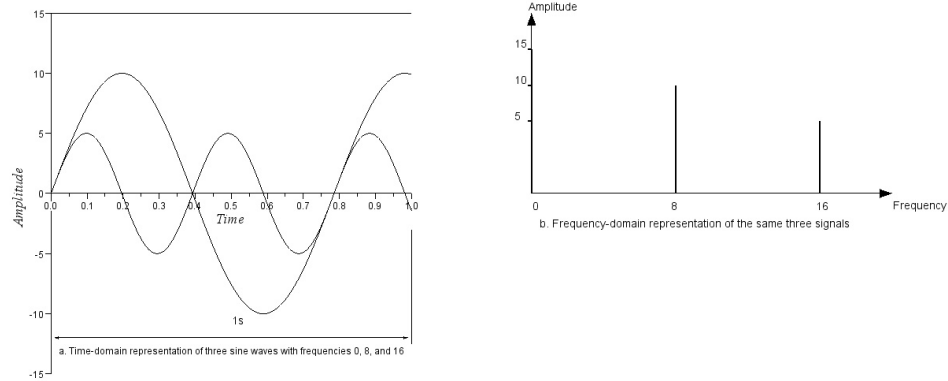


Figure 3.1: Time and frequency domains

```

42 xstring(-.05,.9,"15")
43 xset("thickness",2.5)
44 xpoly([0 0],[.5 .9])
45 xpoly([.4 .4],[.5 .8])
46 xpoly([.8 .8],[.5 .7])
47 xpoly([0 -.01],[.7 .7])
48 xpoly([0 -.01],[.8 .8])
49 xpoly([0 -.01],[.9 .9])

```

---

### Scilab code Exa 3.8 Periodic composite signal

```

1 clear all;
2 clc;
3 clf();
4 xname("-----Example 3.8-----");
5 a1=gca();
6 // define the properties of the axes
7 a1.x_location = "middle";

```

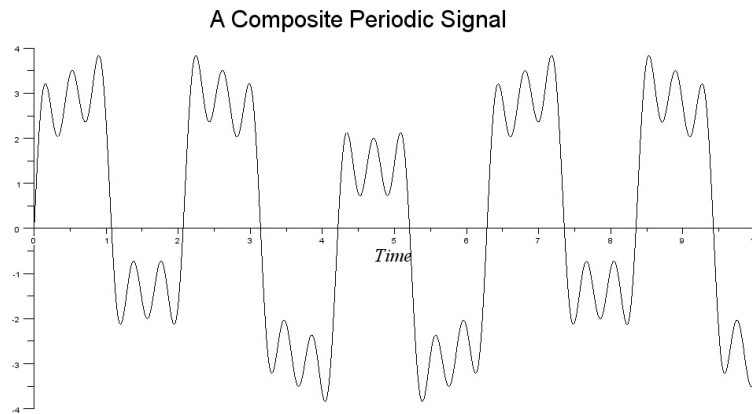


Figure 3.2: Periodic composite signal

```

8 a1.x_label.text="Time"; // display the quantity
  along x-axis
9 a1.x_label.font_style = 3;
10 a1.x_label.font_size = 5;
11 a1.x_label.foreground = 3;
12 a1.title.text="A Composite Periodic Signal" //
  display title
13 a1.title.foreground = 12;
14 a1.title.font_size = 6;
15 x=[0:.001:10]; // x-range
16 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

```

1 clear all;
2 clc;
3 disp("-----Example 3.9-----")

```

```

4 f1=0; // 0kHz
5 fh=4; // 4kHz
6 // example explanation
7 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");
8 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.",f1,fh);
9 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.",f1,fh*10^3);
10 clf();
11 xname("-----Example 3.9-----");
12 //display the figure
13 subplot(121) // time domain plot
14 a1=gca();
15 x=[0:.001:10]; // x-range
16 a1.title.text="a. Time domain";
17 a1.x_location = "middle";
18 a1.x_label.text="Time"; // display the quantity
      along x-axis
19 a1.y_label.text="Amplitude"; // display the quantity
      along y-axis
20 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
21
22 subplot(122) // frequency domain plot
23 a1=gca();
24 p=[0:.01:%pi/5]; // x-range
25 a1.title.text="b. Frequency domain";
26 a1.x_location = "bottom";
27 a1.x_label.text="Frequency"; // display the quantity
    along x-axis
28 a1.y_label.text="Amplitude"; // display the quantity
    along y-axis
29 a1.data_bounds=[0,0;2,5];
30 // equations for the plot
31 plot(p,%e^p+2^p-2);
32 xset("color",2);
33 xpoly([.62 1.5],[1.4 1.4]);
34 xarc(1.02,1.54,.5,.5,0,64*25);
35 xpoly([1.52 1.8],[1.31 0]);
36 xpoly([.62 .62],[0 1.4]);
37 xfarc(.6,1.41,.05,.05,0,64*360)
38 xset("font size",2)
39 xstring(.6,1.5,"Amplitude for sine wave of frequency
    f");
40 xstring(.59,0,"f
    4 kHz")

```

---

### Scilab code Exa 3.10 Bandwidth spectrum

```

1 clear all;
2 clc;
3 disp("-----Example 3.10-----")

```

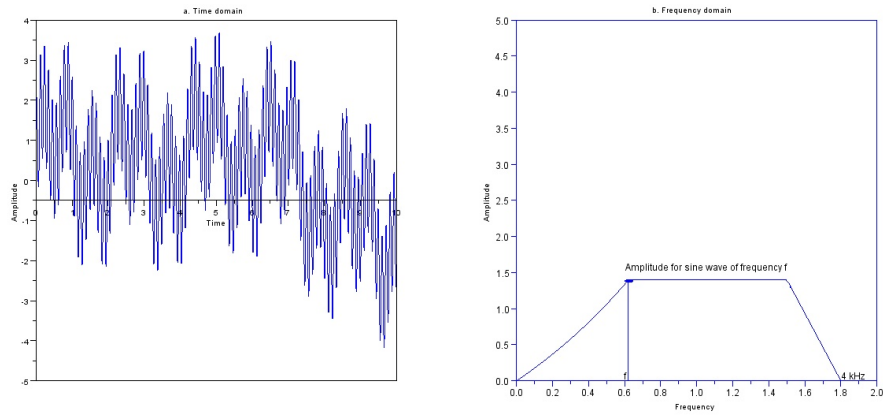


Figure 3.3: Nonperiodic composite signal

```

4 fl=100; //lowest frequency in Hz
5 fh=900; //highest frequency in Hz
6 B=fh-fl; // formula to calculate bandwidth
7 n=5; // number of sine waves
8 dif_f=(fh+fl)/n; // difference between each spike
9 f=[]; // spikes
10 for i=0:4
11     f(i+1)=i*dif_f+fl; // formula
12 end
13 printf("The bandwidth = %d Hz\n",B); // display the
    result
14 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));
15 clf();
16 xname("-----Example 3.10-----");
17 // display the figure
18 xarrows([.1 1.1],[.2 .2],.5);
19 xarrows([.1 .1],[.2 .9],.5);
20 xset("font size",5);
21 xstring(.1,.9,"Amplitude"); // y axis
22 xstring(1.03,.1,"Frequency"); // x axis
23 xset("font size",4);

```



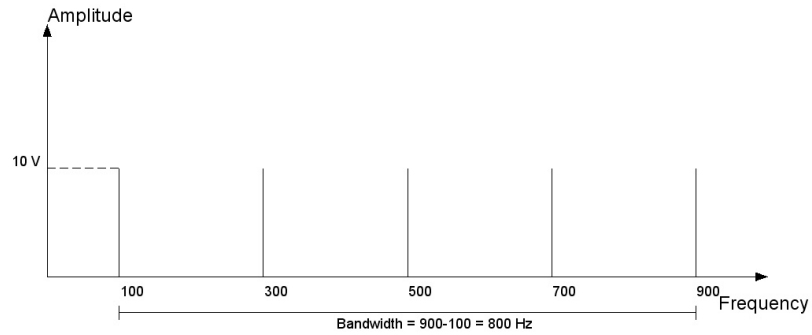


Figure 3.4: Bandwidth spectrum

```

24 xstring(.2,.14,"100");
25 xstring(.4,.14,"300");
26 xstring(.6,.14,"500");
27 xstring(.8,.14,"700");
28 xstring(1,.14,"900");
29 xpoly([.2 .2],[.2 .5]);
30 xpoly([.4 .4],[.2 .5]);
31 xpoly([.6 .6],[.2 .5]);
32 xpoly([.8 .8],[.2 .5]);
33 xpoly([1 1],[.2 .5]);
34 xpoly([.2 1],[.1 .1]);
35 xpoly([.2 .2],[.08 .12]);
36 xpoly([1 1],[.08 .12]);
37 xstring(0.05,.5,"10 V");
38 xset("line style",2);
39 xpoly([.1 .2],[.5 .5],"lines");
40 xstring(.5,.05,"Bandwidth = 900-100 = 800 Hz");

```

---

### Scilab code Exa 3.11 Bandwidth spectrum 2

```
1 clear all;
2 clc;
3 disp("-----Example 3.11-----")
4 B=20; //bandwidth in Hz
5 fh=60; //highest frequency in Hz
6 fl=fh-B; // formula to calculate lowest frequency
7 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
8 clf();
9 xname("-----Example 3.11-----");
10 xarrows([.1 .9],[.2 .2],.5);
11 xset("font size",5);
12 xstring(.92,.1,"Frequency (Hz)");
13 xset("thickness",2);
14 x=linspace(.2,.8,21);
15 for i=1:21
16     xpoly([x(i) x(i)],[.2 .5]);
17 end
18 xset("font size",3);
19 for i=1:21
20     s=40+i-1;
21     xstring(x(i),.17,string(s));
22 end
23 xpoly([.2 .8],[.1 .1]);
24 xpoly([.2 .2],[.08 .12]);
25 xpoly([.8 .8],[.08 .12]);
26 xstring(.4,.1,"Bandwidth = 60-40 = 20 Hz");
```

---

### Scilab code Exa 3.12 Frequency domain of signal

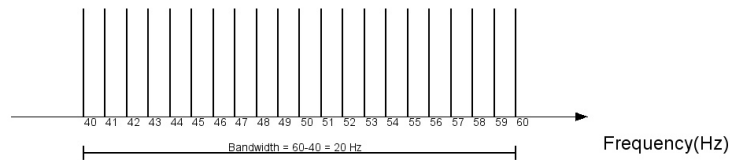


Figure 3.5: Bandwidth spectrum 2

```

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

```

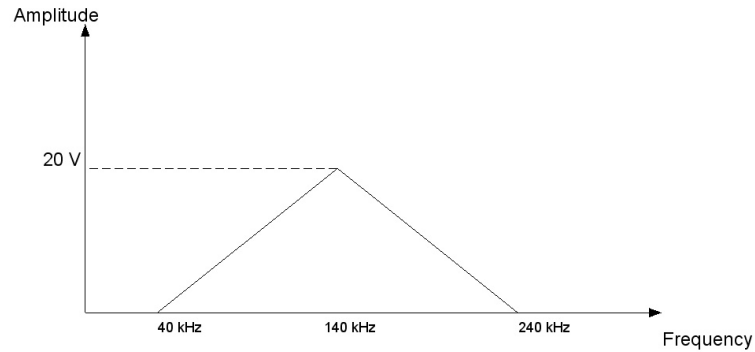


Figure 3.6: Frequency domain of signal

```

21 xstring(0.14,.6," 20 V");
22 xset("font size",4);
23 xstring(.3,.14," 40 kHz");
24 xstring(.8,.14," 240 kHz");
25 xstring(.53,.14," 140 kHz");

```

---

### Scilab code Exa 3.13 AM Radio Bandwidth

```

1 clear all;
2 clc;
3 disp("-----Example 3.13-----")
4 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

```
1 clear all;
2 clc;
3 disp("-----Example 3.14-----")
4 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

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

```

14 final =ceil(b7); // final rounded bandwidth
15 // display the result
16 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);
17 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

```

1 clear all;
2 clc;
3 disp("-----Example 3.16-----")
4 L=8; // number of levels
5 bits_per_level=log2(L); // formula to calculate
    number of bits per level
6 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

```

1 clear all;
2 clc;
3 disp("-----Example 3.17-----")
4 L=9; // number of levels

```

```

5 bits_per_level=log2(L); // formula to calculate
  number of bits per level
6 printf("The number of bits per level is %3.2f bits.\n
  n",bits_per_level);
7 if (~(bits_per_level/10 == 0)) // if the number of
  bits is not an integer or power of 2
8   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");
9   r=nextpow2(bits_per_level); // find nearest
  power of 2
10  bits=2^r;
11  printf("Therefore %d bits can represent one
  level.",bits); // display result
12 end

```

---

Scilab code Exa 3.18 Download text document

```

1 clear all;
2 clc;
3 disp("-----Example 3.18-----")
4 num_line=24; // number of lines
5 num_char=80; // number of characters in each line
6 bits=8; // bits per character
7 rate = 100; // pages per minute
8 br=rate*bits*num_char*num_line; // formula to
  calculate bit rate
9 bit_rate=br*10^-6; // multiply with conversion
  factor
10 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

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

---

### Scilab code Exa 3.20 HDTV bitrate

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

---

### Scilab code Exa 3.21 LAN



```

1 clear all;
2 clc;
3 disp("-----Example 3.21-----")
4 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

```

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

```

```

    factor
13 printf(" b) The required bandwidth is %2.1f MHz.",
    bandwidth); // display result
14 printf("\n Hence a better result can be achieved by
    using the first and the third harmonics.\n\n");
15 //c) using the first , third , and fifth harmonics
16 bw = 5*mb; // formula to calculate bandwidth
17 bandwidth = bw*10^-6; //multiply with conversion
    factor
18 printf(" c) The required bandwidth is %2.1f MHz.",
    bandwidth); // display result
19 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

```

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

```

---

### Scilab code Exa 3.24 Broadband transmission example

```

1 clear all;
2 clc;
3 disp("-----Example 3.24-----")
4 printf("An example of broadband transmission using
    modulation is the sending of computer data

```

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

---

### Scilab code Exa 3.25 Digital cellular telephone

```

1 clear all;
2 clc;
3 disp("-----Example 3.25-----")
4 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  $W/1000$ , just part of the entire bandwidth. We need to convert the digitized voice to a composite analog signal before sending. The digital cellular phones convert the analog audio signal to digital and then convert it again to analog for transmission over a bandpass channel.

```
"); //display the example
```

---

#### Scilab code Exa 3.26 Attenuation calculation

```
1 clear all;
2 clc;
3 disp("-----Example 3.26-----")
4 ratio=0.5; // power of signal after attenuation/
   initial power of signal = p2/p1
5 at=10*log10(ratio); // formula to calculate
   attenuation or loss of power
6 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

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

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

---

#### Scilab code Exa 3.28 Resultant decibel calculation

```
1 clear all;  
2 clc;  
3 disp("-----Example 3.28-----")  
4 dB1=-3; // signal is attenuated  
5 dB2=7; // signal is amplified  
6 dB3=-3; // signal is attenuated  
7 dB=dB1+dB2+dB3; // add to get final dB  
8 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

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

---

#### Scilab code Exa 3.30 dB per km

```
1 clear all;  
2 clc;
```

```

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

```

---

#### Scilab code Exa 3.31 SNR and SNRdB

```

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

```

---

#### Scilab code Exa 3.32 Noiseless channel SNR and SNRdB

```

1 clear all;
2 clc;
3 disp("-----Example 3.32-----")
4 //for noiseless channels , noise power =0
5 //SNR = signal power/ 0 = infinity
6 //SNRdB = 10*log10(SNR)=10*log10(infinity)=infinity

```

```
7 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

```
1 clear all;
2 clc;
3 disp("-----Example 3.33-----")
4 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

```
1 clear all;
2 clc;
3 disp("-----Example 3.34-----")
4 L=2; //number of levels
5 bandwidth=3000; // Hz
6 max_bitrate=2*bandwidth*log2(L); // formula to
    calculate maximum bit rate
7 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

```
1 clear all;
2 clc;
3 disp("-----Example 3.35-----")
4 L=4; //number of levels
5 bandwidth=3000; // Hz
6 max_bitrate=2*bandwidth*log2(L); // formula to
   calculate maximum bit rate
7 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

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



```

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

```

---

**Scilab code Exa 3.37** C for extremely noisy channel

```

1 clear all;
2 clc;
3 disp("-----Example 3.37-----")
4 SNR = 0; //an extremely noisy channel in which the
           value of the signal-to-noise ratio is almost zero
           .
5 m=log2(1+SNR);
6 //display result
7 printf("The value of log2(1+SNR) = %d \nHence C = B*
           log2(1+SNR)= B*0 = %d",m,m);
8 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

```

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

```

```

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

```

---

**Scilab code Exa 3.39** C using SNRdB

```

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

```

---

**Scilab code Exa 3.40** C formula simplification

```

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

```

---

### Scilab code Exa 3.41 Shannon and Nyquist formula

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

---

### Scilab code Exa 3.42 Modulation using modem

```
1 clear all;
2 clc;
3 disp("-----Example 3.42-----")
4 printf("The bandwidth of a subscriber line is 4 kHz
    for voice or data. The bandwidth of this line for
    data transmission\n can 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

```
1 clear all;
```

```

2 clc;
3 disp("-----Example 3.43-----")
4 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

```

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

```

---

#### Scilab code Exa 3.45 Propagation time calculation

```

1 clear all;
2 clc;
3 disp("-----Example 3.45-----")
4 distance=12000*10^3; // 12000km
5 propagation_speed=2.4*10^8; // 2.4*10^8 m/s

```

```

6 propagation_time=distance/propagation_speed; //
  formula for propagation time
7 propagation_time=propagation_time*10^3; //multiply
  with conversion factor
8 // display result
9 printf("\nThe propagation time is %d ms.\n",
  propagation_time);
10 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

```

1 clear all;
2 clc;
3 disp("-----Example 3.46-----")
4 message_size=2.5*10^3; //2.5 kbyte
5 bandwidth=10^9; // 1Gbps
6 propagation_speed=2.4*10^8; //2.4*10^8 m/s
7 distance=12000*10^3; // 12,000 km
8 propagation_time=distance/propagation_speed; //
  propagation time formula
9 transmission_time=(message_size*8)/bandwidth; //
  transmission time formula
10 // display result
11 printf("\nThe propagation time is %d ms.\n",
  propagation_time*10^3);
12 printf("The transmission time is %4.3f ms.\n",
  transmission_time*10^3);
13 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

```
1 clear all;
2 clc;
3 disp("-----Example 3.47-----")
4 message_size=5*10^6; //5 M byte
5 bandwidth=10^6; // 1Mbps
6 propagation_speed=2.4*10^8; //2.4*10^8 m/s
7 distance=12000*10^3; // 12,000 km
8 propagation_time=distance/propagation_speed; //
   propagation time formula
9 transmission_time=(message_size*8)/bandwidth; //
   transmission time formula
10 // display result
11 printf("\nThe propagation time is %d ms.\n",
   propagation_time*10^3);
12 printf("The transmission time is %d s.\n",
   transmission_time);
13 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

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

the bandwidth, \(\) and 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

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

---

Scilab code Exa 4.2 Nyquist formula equivalence

```
1 clear all;
2 clc;
3 disp("-----Example 4.2-----")
4 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
5  disp("Nmax = (1/c) x B x r = 2 x B x log2L");

```

---

#### Scilab code Exa 4.3 Extra bits calculation

```

1  clear all;
2  clc;
3  disp("-----Example 4.3-----")
4  data_rate1=10^3; //1 kbps
5  data_rate2=10^6; //1 Mbps
6  frac= 0.1*10^-2; // receiver clock is 0.1 percent
   faster than the sender clock
7  function []=extrabits(data_rate,frac) // function to
   calculate extra bps and recieved bps
8     extra = data_rate*frac; // formula to calculate
   extra bps
9     recieved = data_rate+extra; //formula to
   calculate recieved bps
10    printf("Data rate = %d bps\nBits sent = %d\nBits
   recieved = %d\nExtra bps = %d ",recieved,
   data_rate,recieved,extra); //display the
   result
11  endfunction
12  //data rate = 1 kbps
13  printf("\nAt 1 kbps, \n");
14  extrabits(data_rate1,frac); //calling the function
15  //data rate = 1 Mbps
16  printf("\n\nAt 1 Mbps, \n")
17  extrabits(data_rate2,frac); //calling the function

```

---

#### Scilab code Exa 4.4 avg baud and min bandwidth

```

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

```

---

#### Scilab code Exa 4.5 Block coding minimum bandwidth

```

1 clear all;
2 clc;
3 disp("-----Example 4.5-----")
4 data_rate=1; // 1 Mbps
5 frac= 0.25 // 4B/5B coding adds 25% to the baud rate
6 add=data_rate*frac;
7 N = (data_rate+add)*10^6; // Hz
8 NRZI_B= N/2; // minimum bandwidth using NRZ-I
9 Manchester_B = data_rate; // minimum bandwidth using
   Manchester scheme
10 // display result
11 printf("\n\n 4B/5B block coding increases the bit rate
   to %3.2f Mbps\n\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);
12 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

```
1 clear all;
2 clc;
3 disp("-----Example 4.6-----")
4 // example explanation
5 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");
6 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."
        )
7 // display the figure
8 clf();
9 xname("-----Example 4.6-----");
10 subplot(325)
11 a1=gca();
12 a1.x_label.text="c. Undersampling fs=f";
13 a1.x_location="middle";
14 x=[0:.1:5*pi]; // x-range
15 plot(x, sin(x), nax=[0,0,0,0]);
16 xfarc(1.5, 1, .1, .1, 0, 360*64);
17 xfarc(6.2, .1, .1, .1, 0, 360*64);
18 xfarc(11, -.9, .1, .1, 0, 360*64);
19 xfarc(15.7, 0, .1, .1, 0, 360*64);
20 subplot(321)
21 a1=gca();
22 a1.x_label.text="a. Nyquist rate sampling fs=2f";
```

```

23 a1.x_location="middle";
24 x=[0:.1:5*pi]; // x-range
25 plot(x, sin(x), max=[0,0,0,0]);
26 xfarc(1.5,1,.1,.1,0,360*64);
27 xfarc(4.7,-.9,.1,.1,0,360*64);
28 xfarc(7.9,1,.1,.1,0,360*64);
29 xfarc(11,-.9,.1,.1,0,360*64);
30 xfarc(14,1,.1,.1,0,360*64);
31 subplot(323)
32 a1=gca();
33 a1.x_label.text="b. Oversampling fs=4f";
34 a1.x_location="middle";
35 x=[0:.1:5*pi]; // x-range
36 plot(x, sin(x), max=[0,0,0,0]);
37 xfarc(1.5,1,.1,.1,0,360*64);
38 xfarc(4.7,-.9,.1,.1,0,360*64);
39 xfarc(7.9,1,.1,.1,0,360*64);
40 xfarc(11,-.9,.1,.1,0,360*64);
41 xfarc(14,1,.1,.1,0,360*64);
42 xfarc(0,0.1,.1,.1,0,360*64);
43 xfarc(3.1,.1,.1,.1,0,360*64);
44 xfarc(6.2,.1,.1,.1,0,360*64);
45 xfarc(9.4,.1,.1,.1,0,360*64);
46 xfarc(12.5,.1,.1,.1,0,360*64);
47 xfarc(15.7,0,.1,.1,0,360*64);
48 subplot(322)
49 a1=gca();
50 xarrows([0 1],[.5 .5],.7);
51 xarrows([0 0],[0 1],.7);
52 xset("line style",2);
53 for i=0:2
54     xpoly([0+(i/2.5) .1+(i/2.5)], [.5 1]);
55     xpoly([.1+(i/2.5) .2+(i/2.5)], [1 .5]);
56 end
57 for i=0:1
58     xpoly([.2+(i/2.5) .3+(i/2.5)], [.5 0]);
59     xpoly([.3+(i/2.5) .4+(i/2.5)], [0 .5]);
60 end

```

```

61 xfarcc (.09,1,.02,.02,0,360*64);
62 xfarcc (.29,0,.02,.02,0,360*64);
63 xfarcc (.49,1,.02,.02,0,360*64);
64 xfarcc (.69,0,.02,.02,0,360*64);
65 xfarcc (.89,1,.02,.02,0,360*64);
66 subplot(324)
67 a1=gca();
68 xarrows([0 1],[.5 .5],.7);
69 xarrows([0 0],[0 1],.7);
70 xset("line style",2);
71 for i=0:2
72     xpoly([0+(i/2.5) .1+(i/2.5)], [.5 1]);
73     xpoly([.1+(i/2.5) .2+(i/2.5)], [1 .5]);
74 end
75 for i=0:1
76     xpoly([.2+(i/2.5) .3+(i/2.5)], [.5 0]);
77     xpoly([.3+(i/2.5) .4+(i/2.5)], [0 .5]);
78 end
79 xfarcc (.09,1,.02,.02,0,360*64);
80 xfarcc (.29,0,.02,.02,0,360*64);
81 xfarcc (.49,1,.02,.02,0,360*64);
82 xfarcc (.69,0,.02,.02,0,360*64);
83 xfarcc (.89,1,.02,.02,0,360*64);
84 xfarcc (0,.52,.02,.02,0,64*360);
85 xfarcc (.2,.52,.02,.02,0,64*360);
86 xfarcc (.4,.52,.02,.02,0,64*360);
87 xfarcc (.6,.52,.02,.02,0,64*360);
88 xfarcc (.8,.52,.02,.02,0,64*360);
89 xfarcc (1,.52,.02,.02,0,64*360);
90 subplot(326)
91 a1=gca();
92 //a1.x_location="middle";
93 //x=[0:.1:3*%pi]; // x-range
94 //plot(x,cos(.5*x),nax=[0,0,0,0]);
95 xfarcc (.1,.75,.02,.02,0,360*64);
96 xfarcc (.318,.5,.02,.02,0,360*64);
97 xfarcc (.67,0,.02,.02,0,360*64);
98 xfarcc (.97,.5,.02,.02,0,360*64);

```

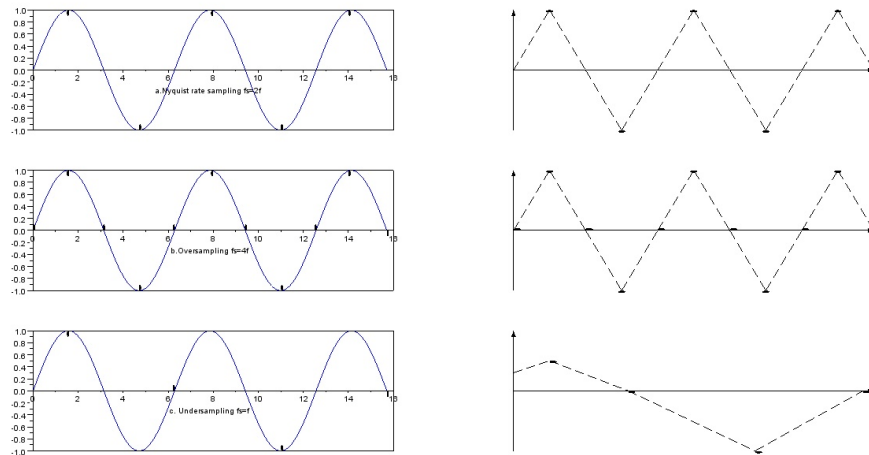


Figure 4.1: sampling and recovery

```

99 xarrows ([0 1], [.5 .5], .7);
100 xarrows ([0 0], [0 1], .7);
101 xset ("line style", 2);
102 xpoly ([0 .1], [.65 .75]);
103 xpoly ([.1 .318], [.75 .5]);
104 xpoly ([.318 .67], [.5 0]);
105 xpoly ([.67 .97], [0 .5]);

```

---

#### Scilab code Exa 4.7 Sampling of clock

```

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

```

```

7 Ts=0.5*T ; // or fs = 2f
8 sp=[]; // sampling points
9 time=[12 6];
10 for i=0:5 // assign sampling points
11     if(modulo(i,2)==0)
12         sp(i+1)=time(1);
13     else
14         sp(i+1)=time(2);
15     end
16 end
17 //display the result
18 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));
19 // b) Sampling at double the Nyquist rate
20 Ts=0.25*T; // or fs = 4f
21 sp=[]; //sampling points
22 time=[12 3 6 9];
23 for i=0:4 // assign sampling points
24     if (i==4) then
25         sp(i+1)=time(1);
26     else
27         sp(i+1)=time(i+1);
28     end
29 end
30 //display the result
31 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));
32 // b) Sampling at lesser than Nyquist rate
33 Ts=0.75*T; // or fs = 4/3f
34 sp=[]; //sampling points
35 time=[12 9 6 3];

```

```

36 for i=0:4 // assign sampling points
37     if (i==4) then
38         sp(i+1)=time(1);
39     else
40         sp(i+1)=time(i+1);
41     end
42 end
43 //display the result
44 printf("\n\n(c)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

```

1 clear all;
2 clc;
3 disp("-----Example 4.8-----")
4 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

```

1 clear all;
2 clc;
3 disp("-----Example 4.9-----")

```



```

4 f=4000; // max frequency
5 sr=2*f; // sampling rate = 2*max frequency
6 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

```

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

```

---

**Scilab code Exa 4.11** bandpass sampling rate

```

1 clear all;
2 clc;
3 disp("-----Example 4.11-----")
4 bandwidth = 200; // 200 kHz
5 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

```
1 clear all;
2 clc;
3 disp("-----Example 4.12-----")
4 L=8; // number of levels
5 nb=log2(L); // number of bits per sample
6 SNRdB=6.02*(nb)+1.76; // formula
7 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

```
1 clear all;
2 clc;
3 disp("-----Example 4.13-----")
4 SNRdB=40;
5 nb=(SNRdB-1.76)/6.02; // SNRdB = 6.02(nb)+1.76
6 printf("\nnb = %4.2f",nb); // display result
7 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

```
1 clear all;
2 clc;
```

```

3 disp("-----Example 4.14-----")
4 bits= 8; // bits per sample
5 fl=0; // The human voice normally contains
    frequencies from 0 to 4000 Hz.
6 fh=4000; // Hz
7 sampling_rate = 2*fh; // twice the highest frequency
8 bit_rate=sampling_rate*bits; // formula
9 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

```

1 clear all;
2 clc;
3 disp("-----Example 4.15-----")
4 analog_min_bandwidth = 4; //4 kHz
5 bits = 8; // bits per sample
6 digital_min_bandwidth = analog_min_bandwidth*bits;
    // formula
7 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

```
1 clear all;
2 clc;
3 disp("-----Example 5.1-----")
4 S=1000; // baud rate
5 r=4; // bits/signal element
6 N=S*r; // bit rate formula
7 printf("The bit rate is %d bps.",N); // display
   result
```

---

Scilab code Exa 5.2 data and signal elements

```
1 clear all;
2 clc;
3 disp("-----Example 5.2-----")
4 S=1000; // baud rate
5 N= 8000; // bit rate in bps
6 r= N/S; // data elements/signal element
7 L= 2^r ; // number of signal elements
```

```

8 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

```

1 clear all;
2 clc;
3 disp("-----Example 5.3-----")
4 d=1;
5 r=1;
6 B=100; // 100 kHz
7 fl=200; // lower frequency=200 kHz
8 fh=300; // highest frequency =300 kHz
9 middle_bandwidth = (fl+fh)/2; // kHz
10 Fc=middle_bandwidth; // carrier frequency
11 N=(B*r)/2; // B= (1+d)*S = 2*N*(1/r) , N - bit rate
12 printf("\nThe carrier frequency is %d kHz.\n\nThe bit
   rate is %d kbps.",Fc,N); //display result

```

---

### Scilab code Exa 5.4 Full duplex ASK

```

1 clear all;
2 clc;
3 disp("-----Example 5.4-----")
4 B=100; // total Bandwidth in kHz
5 B1=B/2; // bandwith for one direction
6 B2=B/2; // bandwidth for other direction
7 fl=200; // lower frequency=200 kHz
8 fh=300; // highest frequency =300 kHz
9 middle_bandwidth1= (fl+(fl+B1))/2; // kHz
10 Fc1=middle_bandwidth1; // carrier frequency for one
   direction

```

```

11 middle_bandwidth2= ((f1+B1)+fh)/2; // kHz
12 Fc2=middle_bandwidth2; // carrier frequency for
    other direction
13 N1=B1/2; // data rate in one direction
14 N2=B2/2; //data rate in other direction
15 // display result
16 printf("\nThe carrier frequency for one direction is
    %d kHz , bandwidth is %d kHz and data rate is %d
    kbps.\n",Fc1,B1,N1);
17 printf("\nThe carrier frequency for other direction
    is %d kHz , bandwidth is %d kHz and data rate is
    %d kbps.\n",Fc2,B2,N2);
18 // display the figure
19 clf();
20 xname("-----Example 5.4-----");
21 x=linspace(.45,.55,2);
22 for i=1:2
23     xpoly([x(i) x(i)],[.18 .22]);
24 end
25 x=linspace(.5,.6,2);
26 for i=1:2
27     xpoly([x(i) x(i)],[.2 .3]);
28 end
29 x=linspace(.4,.5,2);
30 for i=1:2
31     xpoly([x(i) x(i)],[.2 .3]);
32 end
33 for i=0:1
34     xarc(.47+(i/10),.31,.03,.03,0,90*64);
35     xarc(.4+(i/10),.31,.03,.03,90*64,91*64);
36 end
37 xpoly([.35 .65],[.2 .2])
38 for i=0:1
39     xpoly([.41+(i/10) .49+(i/10)],[.31 .31]);
40
41 end
42 xpoly([.4 .49],[.32 .32]);
43 xpoly([.5 .6],[.32 .32]);

```

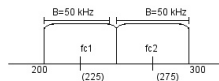


Figure 5.1: Full duplex ASK

```

44 xpoly([.4 .4],[.31 .33]);
45 xpoly([.49 .49],[.31 .33]);
46 xpoly([.5 .5],[.31 .33]);
47 xpoly([.6 .6],[.31 .33]);
48 xset("font size",2)
49 xstring(.38,.17,"200
        300");
50 xstring(.45,.15,"(225)          (275)");
51 xstring(.449,.23,"fc1          fc2");
52 xstring(.41,.33,"B=50 kHz      B=50
        kHz");

```

---

#### Scilab code Exa 5.5 FSK fc and bitrate

```

1 clear all;
2 clc;
3 disp("-----Example 5.5-----")

```

```

4 B=100 ; //bandwidth = 100 kHz
5 two_df=50; // 2df = 50 kHz
6 fl=200; // lower frequency in kHz
7 fh=300; // higher frequency in kHz
8 mid_bandwidth = (fl+fh)/2; // mid frequency of
   bandwidth in kHz
9 Fc=mid_bandwidth;
10 d=1;
11 S=(B-two_df)/(1+d); // B= (1+d)*s + 2df
12 N=S; // bit rate
13 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

```

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

```



```

18 xpoly([.1 .9],[.55 .55]);
19 xpoly([.1 .1],[.57 .53]);
20 xpoly([.9 .9],[.57 .53]);
21 xstring(.4,.6,"Bandwidth = 8 MHz");
22 x=linspace(.15,.85,8);
23 for i=1:8
24     xpoly([x(i) x(i)],[.18 .22]);
25 end
26 x=linspace(.1,.9,9);
27 for i=1:9
28     xpoly([x(i) x(i)],[.2 .3]);
29 end
30 for i=0:7
31     xarc(.17+(i/10),.31,.03,.03,0,90*64);
32     xarc(.1+(i/10),.31,.03,.03,90*64,91*64);
33 end
34
35 for i=0:7
36     xpoly([.11+(i/10) .19+(i/10)],[.31 .31]);
37 end
38 xset("thickness",2);
39 xpoly([.5 .5],[.2 .35])
40 xset("font size",3);
41 x=linspace(.15,.85,8);
42 for i=1:8
43     s=6.5+i-1;
44     xstring(x(i),.14,"f"+string(i));
45     xstring(x(i),.1,string(s));
46     xstring(x(i),.06,"MHz");
47 end
48 xstring(.5,.14,"fc");
49 xstring(.5,.1,"10");
50 xstring(.5,.06,"MHz");

```

---

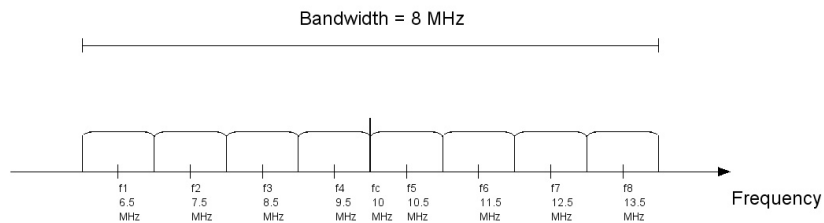


Figure 5.2: bandwidth of MFSK

**Scilab code Exa 5.7** bandwidth of QPSK

```

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

```

---

**Scilab code Exa 5.8** Three constellations diagrams

```

1 clear all;
2 clc;
3 // drawing the constellation diagrams
4 clf();

```

```

5  xname("-----Example 5.8-----");
6  title('Constellation diagrams','fontsize',5);
7  xrect(-.1,.8,.35,.45);
8  xrect(.3,.8,.35,.45);
9  xrect(.7,.8,.35,.45);
10 xpoly([- .1 .25],[.575 .575]);
11 xpoly([.3 .65],[.575 .575]);
12 xpoly([.7 1.05],[.575 .575]);
13 xpoly([.075 .075],[.8 .35]);
14 xpoly([.475 .475],[.8 .35]);
15 xpoly([.875 .875],[.8 .35]);
16 xset("font size",4);
17 xstring(-.1,.3,"a. ASK(OOK)");
18 xstring(.3,.3,"b. BPSK");
19 xstring(.7,.3,"c. QPSK");
20 xfarc( .059,.585 , 0.03, 0.03, 0, 64 * 360 ) ;
21 xfarc( .15,.585 , 0.03, 0.03, 0, 64 * 360 ) ;
22 xfarc( .37,.585 , 0.03, 0.03, 0, 64 * 360 ) ;
23 xfarc( .55,.585 , 0.03, 0.03, 0, 64 * 360 ) ;
24 xfarc( .77,.7 , 0.03, 0.03, 0, 64 * 360 ) ;
25 xfarc( .95,.7 , 0.03, 0.03, 0, 64 * 360 ) ;
26 xfarc( .77,.46 , 0.03, 0.03, 0, 64 * 360 ) ;
27 xfarc( .95,.46 , 0.03, 0.03, 0, 64 * 360 ) ;
28 xset("font size",3);
29 xstring(.055,.5,"0");
30 xstring(.15,.5,"1");
31 xstring(.37,.5,"0");
32 xstring(.55,.5,"1");
33 xstring(.745,.7,"01");
34 xstring(.975,.7,"11");
35 xstring(.74,.44,"00");
36 xstring(.98,.44,"10");
37 xset("line style",2);
38 xarc(.75,.7,.275,.275,0,64*360);
39 // display the explanation of the diagrams
40 disp("-----Example 5.8-----")
41 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");
42 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");
43 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 .\n
  \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 chosen the amplitude of the carrier to be
   $1/(2^{(1/2)})$ \nto make the final amplitudes 1 V.");

```

---

Constellation diagrams

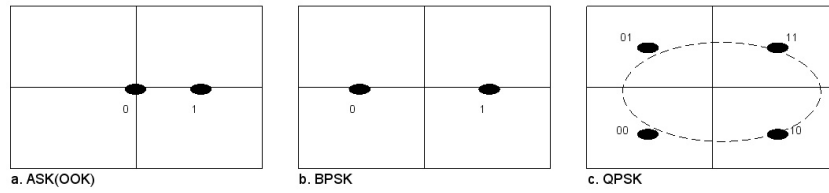


Figure 5.3: Three constellations diagrams

# Chapter 6

## Bandwidth Utilization Multiplexing and Spreading

Scilab code Exa 6.1 FDM configuration

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

```

bandwidth for channel 3
14 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.",f11,fh1,f12,fh2,f13,fh3,f11,
    fh1,f12,fh2,f13,fh3); // display result
15 // display the figure
16 clf();
17 xname("-----Example 6.1-----");
18 xset("font size",3)
19 for i=0:2
20     xset("color",i+1);
21     xfpoly([- .05 - .05 - .02 .01 .01],[.6-(i/6) .64-(i
        /6) .68-(i/6) .64-(i/6) .6-(i/6)]);
22     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)]);
23     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)]);
24     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)]);
25     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)]);
26     xset("color",0);
27     xpoly([- .1 .05],[.6-(i/6) .6-(i/6)]);

```

```

28     xrect(.05,.64-(i/6),.07,.06);
29     xstring(.05,.6-(i/6),"Modulator");
30     xpoly([.12 .32],[.6-(i/6) .6-(i/6)]);
31     xarrows([.33 .36],[.6-(i/6) .435],.5);
32     xarrows([.69 .72],[.435 .6-(i/6)],.5);
33     xrect(.73,.64-(i/6),.07,.06);
34     xstring(.76,.6-(i/6),"Filter");
35     xpoly([.8 .995],[.6-(i/6) .6-(i/6)]);
36     xpoly([1.01 1.2],[.6-(i/6) .6-(i/6)]);
37     xstring(-.05,.56-(i/6),"0");
38     xstring(.01,.56-(i/6),"4");
39     xstring(1.05,.56-(i/6),"0");
40     xstring(1.1,.56-(i/6),"4");
41 end
42 xarc(.36,.445,.045,.045,0,64*360);
43 xarc(.64,.445,.045,.045,0,64*360);
44 xarrows([.41 .63],[.6-(1/6) .6-(1/6)],.4);
45 xrect(.04,.69,.37,.46);
46 xrect(.72,.69,.28,.46);
47 xset("font size",3)
48 xstring(.375,.41,"+");
49 xstring(.04,.7,"Shift and combine");
50 xstring(.75,.2,"Filter and shift");
51 xstring(.43,.35,"Higher-bandwidth link");
52 xstring(.42,.39,"20");
53 xstring(.59,.39,"32");
54 xstring(.12,.57,"20");
55 xstring(.19,.57,"24");
56 xstring(.8,.57,"20");
57 xstring(.87,.57,"24");
58 xstring(.18,.4,"24");
59 xstring(.25,.4,"28");
60 xstring(.86,.4,"24");
61 xstring(.93,.4,"28");
62 xstring(.25,.23,"28");
63 xstring(.32,.23,"32");
64 xstring(.93,.23,"28");
65 xstring(.99,.23,"32");

```



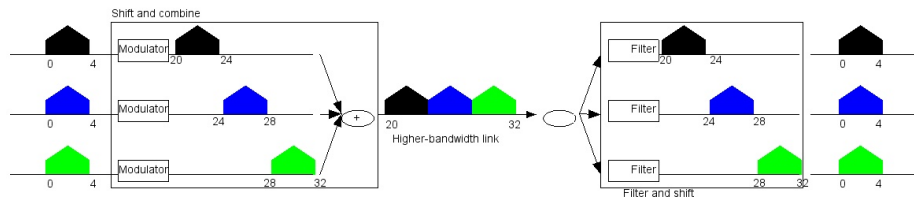


Figure 6.1: FDM configuration

---

Scilab code Exa 6.2 bandwidth with guards

```

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

```

```

12 xname("-----Example 6.2-----");
13 xarrows([0 1],[.2 .2],.5);
14 xset("font size",5);
15 xstring(1,.1,"Frequency");
16 xpoly([.1 .8],[.07 .07]);
17 xpoly([.1 .1],[.09 .05]);
18 xpoly([.8 .8],[.09 .05]);
19 xset("font size",4);
20 xstring(.4,.01,"540 kHz");
21 x=linspace(.1,.7,5);
22 y=linspace(.2,.8,5);
23 for i=1:5
24     xpoly([x(i) x(i)],[.2 .3]);
25 end
26 for i=1:5
27     xpoly([y(i) y(i)],[.2 .3]);
28 end
29 for i=0:4
30     xarc(.17+(i/6.65),.31,.03,.03,0,90*64);
31     xarc(.1+(i/6.65),.31,.03,.03,90*64,91*64);
32 end
33
34 for i=0:4
35     xpoly([.11+(i/6.65) .19+(i/6.65)],[.31 .31]);
36     xpoly([.1+(i/6.65) .2+(i/6.65)],[.35 .35]);
37     xpoly([.1+(i/6.65) .1+(i/6.65)],[.33 .37]);
38     xpoly([.2+(i/6.65) .2+(i/6.65)],[.33 .37]);
39 end
40
41 xset("font size",3);
42 x=linspace(.15,.75,5);
43 for i=1:5
44     xstring(x(i)-.03,.4,"100 kHz");
45 end
46 xarrows([.23 .23],[.5 .36],.5);
47 xstring(.2,.53,"Guard band of 10 kHz");

```

---

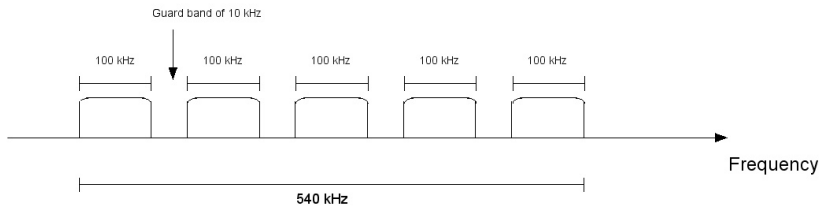


Figure 6.2: bandwidth with guards

### Scilab code Exa 6.3 satellite channel FDM

```

1 clear all;
2 clc;
3 disp("-----Example 6.3-----")
4 n=4; // number of channels
5 bitrate=1; // 1 Mbps
6 total_bandwidth=1*10^6; // 1 MHz
7 channel_bandwidth=total_bandwidth/n;
8 bits=(bitrate*10^6/total_bandwidth)*n; // number of
   bits per Hz
9 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

```

```

10 // display the figure
11 clf();
12 xname("-----Example 6.3-----");
13 xpoly([.5 .5],[.3 .8]);
14 xpoly([.5 .65],[.8 .55]);
15 xpoly([.5 .65],[.3 .55]);
16 xset("font size",2.8);
17 for i=0:3
18     xstring(.22,.71-(i/10),"1 Mbps
           250 kHz");
19     xstring(.22,.665-(i/10),"Digital
           Analog");
20     xpoly([.2 .3],[.7-(i/10) .7-(i/10)]);
21     xpoly([.4 .5],[.7-(i/10) .7-(i/10)]);
22     xrect(.3,.72-(i/10),.1,.05);
23     xstring(.33,.68-(i/10),"16-QAM");
24 end
25 xset("font size",4);
26 xstring(.53,.53,"FDM");
27 xstring(.67,.57,"1 MHz");
28 xpoly([.65 .75],[.55 .55]);

```

---

#### Scilab code Exa 6.4 Advanced MobilePhone System

```

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

```

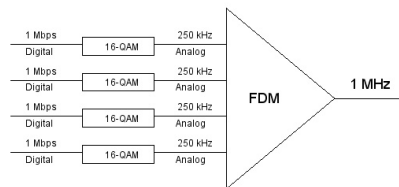


Figure 6.3: satellite channel FDM

```

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

```

---

#### Scilab code Exa 6.5 three durations calculation

```

1 clear all;
2 clc;
3 disp("-----Example 6.5-----")
4 data_rate=1*10^3; // data rate for each input
  connection is 1 kbps
5 unit =1; // 1 bit
6 n=3; // number of channels
7 // a) duration of each input slot
8 bit_duration = 1/data_rate;
9 ip_timeslot=bit_duration;

```

```

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

```

---

#### Scilab code Exa 6.6 Synchronous TDM

```

1 clear all;
2 clc;
3 disp("-----Example 6.6-----")
4 data_rate=1*10^6; // data rate for each input
    connection is 1 Mbps
5 unit =1; // 1 bit
6 n=4; //number of channels
7 //a) input bit duration
8 ip_bd=1/data_rate;
9 printf("a) The input bit duration is %d microseconds
    .\n",ip_bd*10^6); //display result
10 //b)output bit duration
11 op_bd=ip_bd/n;
12 printf("\nb) The output bit duration is %3.2f
    microseconds.\n",op_bd*10^6);//display result
13 // c) output bit rate
14 op_bitrate=1/op_bd;
15 printf("\nc) The output bit rate is %d Mbps.\n",
    op_bitrate*10^-6);//display result
16 //d) output frame rate
17 frame_rate=data_rate;

```

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

---

#### Scilab code Exa 6.7 Four connections multiplexing

```
1 clear all;
2 clc;
3 disp("-----Example 6.7-----")
4 data_rate=1*10^3; // data rate for each input
   connection is 1 kbps
5 unit =1; // 1 bit
6 n=4; //number of channels
7 // (a) the duration of 1 bit before multiplexing
8 bit_duration=1/data_rate;
9 printf("a)The duration of 1 bit before multiplexing
   is %d ms.\n",bit_duration*10^3); //display result
10 // (b) the transmission rate of the link
11 trans_rate=n*data_rate;
12 printf("\nb)The transmission rate of the link is %d
   kbps.\n",trans_rate*10^-3); //display result
13 // (c) the duration of a time slot
14 time_slot = bit_duration/n;
15 printf("\nc)The duration of each time slot is %d
   microseconds.\n",time_slot*10^6); //display
   result
16 // (d) the duration of a frame
17 frame_time = bit_duration;
18 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

```
1 clear all;
```

```

2  clc;
3  disp("-----Example 6.8-----")
4  n=4; // number of channels
5  channel_byte=1; // each frame carries 1 byte from
   each channel
6  frame_size=n*channel_byte; //bytes
7  frame_size_bits=frame_size*8; // 1 byte = 8 bits
8  byte_rate=100; // each channel sends 100 bytes/s
9  frame_rate=channel_byte*byte_rate; // frames per
   second
10 frame_duration=1/frame_rate; // seconds
11 bit_rate=frame_rate*frame_size_bits; // bps
12 // display the result
13 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);
14 // display the figure
15 clf();
16 xname("-----Example 6.8-----");
17 xpoly([.3 .3],[.3 .7]);
18 xset("color",4.2);
19 xfrect(0,.7,.28,.05);
20 xfrect(.58,.58,.035,.045);
21 xfrect(.88,.58,.035,.045);
22 xset("color",2.9);
23 xfrect(0,.6,.28,.05);
24 xfrect(.545,.58,.035,.045);
25 xfrect(.845,.58,.035,.045);
26 xset("color",3.8);
27 xfrect(0,.5,.28,.05);
28 xfrect(.51,.58,.035,.045);
29 xfrect(.81,.58,.035,.045);
30 xset("color",0);
31 xfrect(0,.4,.28,.05);

```



```

32 xfrect(.475,.58,.035,.045);
33 xfrect(.775,.58,.035,.045);
34 xpoly([-0.05 .3],[.625 .625]);
35 xpoly([-0.05 .3],[.525 .525]);
36 xpoly([-0.05 .3],[.425 .425]);
37 xpoly([-0.05 .3],[.325 .325]);
38 xset("font size",4)
39 xstring(.33,.5,"MUX");
40 xpoly([.3 .45],[.7 .5]);
41 xpoly([.3 .45],[.3 .5]);
42 xstring(.1,.27,"100 bytes/s");
43 xset("font size",3)
44 xstring(.65,.45,"100 frames/s");
45 xstring(.67,.41,"3200 bps");
46 xstring(.6,.33,"Frame duration = 1/100 s");
47 xstring(.5,.7,"Frame 4 bytes");
48 xstring(.8,.7,"Frame 4 bytes");
49 xstring(.54,.65,"32 bits");
50 xstring(.84,.65,"32 bits");
51 xrect(.47,.59,.15,.07);
52 xrect(.77,.59,.15,.07);
53 xset("thickness",2);
54 xarrows([.45 1],[.5 .5],.2);
55 xset("font size",7);
56 xstring(.66,.53,". . .");

```

---

### Scilab code Exa 6.9 2 bits timeslot

```

1 clear all;
2 clc;
3 disp("-----Example 6.9-----")
4 n=4; // number of channels
5 channel_bits=2; // each frame carries 2 bits from

```

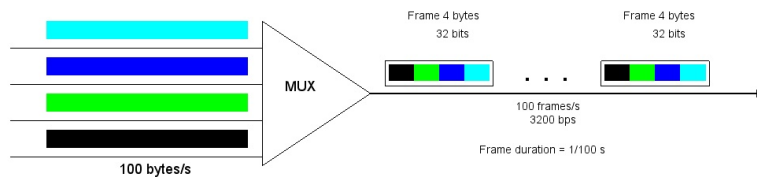


Figure 6.4: Four channels TDM

```

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

```

```

19 xpoly([0 .3],[.5 .5]);
20 xpoly([0 .3],[.4 .4]);
21 xpoly([.3 .3],[.3 .8]);
22 xset("font size",4)
23 xstring(.33,.53,"MUX");
24 xpoly([.3 .45],[.8 .55]);
25 xpoly([.3 .45],[.3 .55]);
26 xset("font size",2.5)
27 xstring(.05,.71,"100 kbps      ...
    110010");
28 xstring(.05,.61,"100 kbps      ...
    001010");
29 xstring(.05,.51,"100 kbps      ...
    101101");
30 xstring(.05,.41,"100 kbps      ...
    000111");
31 xset("font size",3)
32 xstring(.65,.5,"50,000 frames/s");
33 xstring(.67,.46,"400 kbps");
34 xstring(.6,.8,"Frame duration = 1/50000 = 20
    microseconds");
35 xstring(.5,.65,"Frame : 8 bits");
36 xstring(.69,.65,"Frame : 8 bits");
37 xstring(.88,.65,"Frame : 8 bits");
38 xstring(.51,.58,"00");
39 xstring(.55,.58,"10");
40 xstring(.59,.58,"00");
41 xstring(.63,.58,"11");
42 xstring(.7,.58,"01");
43 xstring(.74,.58,"11");
44 xstring(.78,.58,"10");
45 xstring(.82,.58,"00");
46 xstring(.89,.58,"11");
47 xstring(.93,.58,"01");
48 xstring(.97,.58,"10");
49 xstring(1.01,.58,"10");
50 xstring(.48,.58,"...");
51 xrects([.5 .54 .58 .62;.63 .63 .63 .63;.04 .04 .04

```

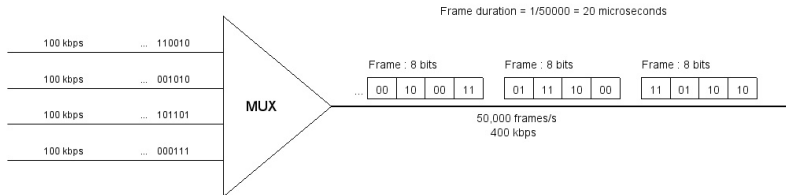


Figure 6.5: 2 bits timeslot

```

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

```

---

#### Scilab code Exa 6.10 4 sources interleaving

```

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

```

```

9 data_rate_source=cps*unit;
10 printf("a)The data rate of each source is %d kps.\n"
    ,data_rate_source*10^-3); // display result
11 // b) the duration of each character in each source
12 character_duration=1/cps;
13 printf("\nb)The duration of each character in each
    source is %d ms.\n",character_duration*10^3); //
    display result
14 // c) the frame rate
15 frame_rate=cps;
16 printf("\nc)The frame rate is %d frames per second.\n"
    ,frame_rate); // display result
17 // d) the duration of each frame
18 frame_duration=1/frame_rate;
19 printf("\nd)The duration of each frame is %d ms.\n",
    frame_duration*10^3); // display result
20 //e) the number of bits in each frame
21 bits=n*unit+sb;
22 printf("\ne)The number of bits in each frame is %d.\n"
    ,bits); // display result
23 //f) the data rate of the link
24 data_rate_link=frame_rate*bits;
25 printf("\nf)The data rate of the link is %d bps.",
    data_rate_link); // display result

```

---

### Scilab code Exa 6.11 2 channels multiplexing

```

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

```

```
7 bits=3; // let each frame carry 3 bits
8 frame_rate=N1*10^3;
9 frame_duration=1/frame_rate;
10 bit_rate=frame_rate*bits;
11 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

```
1 clear all;
2 clc;
3 disp("-----Example 8.1-----")
4 channel_bandwidth=4; // 4 kHz
5 n=2; // each link uses FDM to connect a maximum of
      two voice channels.
6 link_bandwidth=n*channel_bandwidth; // formula
7 // display the result
8 printf("A circuit-switched network is used to
      connect eight telephones in a small area.
      Communication is through %d-kHz voice channels.\n
      It is assumed that each link uses FDM to connect
      a maximum of %d voice channels. The bandwidth of
      each link is then %d kHz.\n
      Telephone 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.\n
      The switch controls the connections.",channel_bandwidth,n,link_bandwidth)
;
```

---

## Scilab code Exa 8.2 circuit switched network of computers

```
1 clear all;
2 clc;
3 disp("-----Example 8.2-----")
4 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\n
   ncomputers in the same office. Four other output
   ports allow communication between the two offices
   ."); // display example explanation
5 // display the figure
6 clf();
7 xname("-----Example 8.2-----");
8 xset("font size",3);
9 xstring(0,.9,"Circuit-switched network");
10 xstring(.12,.67,"4x8 switch");
11 xstring(.62,.67,"4x8 switch");
12 xstring(.3,.63,"T-1 line with 1.544 Mbps");
13 xrects([0 .1 .6;.89 .8 .8;.8 .1 .1;.6 .3 .3]);
14 xpoly([.23 .28 .23],[.73 .68 .63],"lines",1);
15 xpoly([.57 .52 .57],[.73 .68 .63],"lines",1);
16 for i=0:2
17     xpoly([.2 .23],[.72-(i/25) .72-(i/25)]);
18     xpoly([.57 .6],[.72-(i/25) .72-(i/25)]);
19 end
20
21 for i=0:3
22     xpoly([.11+(i/45) .11+(i/45)],[.5 .45-(i/25)]);
```



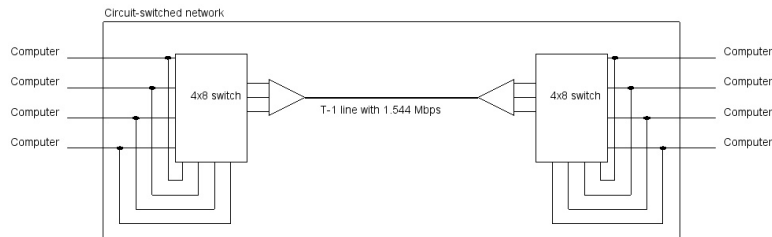


Figure 8.1: circuit switched network of computers

```

23     xpoly([.11+(i/45) .09-(i/45)], [.45-(i/25) .45-(i
        /25)]);
24     xpoly([.09-(i/45) .09-(i/45)], [.45-(i/25) .79-(i
        /12)]);
25     xpoly([.69-(i/45) .69-(i/45)], [.5 .45-(i/25)]);
26     xpoly([.69-(i/45) .71+(i/45)], [.45-(i/25) .45-(i
        /25)]);
27     xpoly([.71+(i/45) .71+(i/45)], [.45-(i/25) .79-(i
        /12)]);
28     xpoly([-0.05 .1], [.79-(i/12) .79-(i/12)]);
29     xpoly([.7 .85], [.79-(i/12) .79-(i/12)]);
30     xstring(-.13, .79-(i/12), "Computer");
31     xstring(.86, .79-(i/12), "Computer");
32     xfarc(.085-(i/45), .795-(i/12), .01, .01, 0, 64*360);
33     xfarc(.705+(i/45), .795-(i/12), .01, .01, 0, 64*360);
34 end
35 xset("thickness", 2.5);
36 xpoly([.28 .52], [.68 .68]);

```

---

### Scilab code Exa 8.3 Design a three stage switch

```
1 clear all;
2 clc;
3 disp("-----Example 8.3-----")
4 N=200; //input lines
5 n=20; // lines in a group
6 k=4; //number of crossbars in the middle stage
7 crossbars1=N/n; // 1st stage
8 crossbars2=k; // 2nd stage
9 crossbars3=N/n; // 3rd stage
10 size1a=n; // size of a crossbar in 1st stage
11 size1b=k;
12 size2=N/n; // size of a crossbar in 2nd stage
13 size3a=k; // size of a crossbar in 3rd stage
14 size3b=n;
15 total_crosspoints= 2*k*N + k*((N/n)^2); //2kN + k(N/
    n)^2
16 singlestage=40000;
17 p=(total_crosspoints/singlestage)*100;
18 // display result
19 printf("\nIn the first stage there are %d crossbars ,
    each of size %d x %d.\n",crossbars1,size1a,
    size1b);
20 printf("\nIn the second stage there are %d crossbars
    , each of size %d x %d.\n",crossbars2,size2,size2
    );
21 printf("\nIn the third stage there are %d crossbars ,
    each of size %d x %d.\n",crossbars3,size3a,
    size3b);
22 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

```
1 clear all;
2 clc;
3 disp("-----Example 8.4-----")
4 N=200;
5 n=(N/2)^(0.5); // formula
6 k=2*n-1; // formula
7 crossbars_1stage=N/n; // formula
8 crosspoints_1stage=n*k; // formula
9 crossbars_2stage=k; // formula
10 crosspoints_2stage=n*n; // formula
11 crossbars_3stage=N/n; // formula
12 crosspoints_3stage=k*n; // formula
13 total_crosspoints=(crossbars_1stage*
    crosspoints_1stage)+(crossbars_2stage*
    crosspoints_2stage)+(crossbars_3stage*
    crosspoints_3stage); // formula
14 crosspoints_singlestage=N*N; // formula
15 p=(total_crosspoints/crosspoints_singlestage)*100;
    // percentage formula
16 // display the result
17 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.\n
    The 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

```
1 clear all;
2 clc;
3 disp("-----Example 10.1-----")
4 k=4;
5 n=5;
6 datawords=2^k; // number of datawords
7 codewords=2^n; // number of codewords
8 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

```
1 clear all;
2 clc;
3 disp("-----Example 10.2-----")
```

```

4 k=2;
5 n=3;
6 table=["Datawords", "Codewords"; "00", "000"; "01", "011"
        ; "10", "101"; "11", "110"];
7 disp(table) // display the table
8 dataword="01";
9 codeword="011";
10 printf("\nAssume the sender encodes the dataword 01
        as 011 and sends it to the receiver. Consider the
        following cases:\n");
11 function []=case_func(codeword,dataword) // function
        to display appropriate result
12     select codeword
13     case "011"
14         printf("\n1. The receiver receives %s. It is
                a valid codeword. The receiver extracts
                the dataword %s from it.",codeword,
                dataword);
15     case "111"
16         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);
17     case "000"
18         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);
19     end
20 endfunction
21 funcprot(0);
22 // case 1
23 case_func(codeword,dataword); //calling the function
24 // case 2

```

```

25 codeword="111";
26 case_func(codeword,dataword); // calling the
    function
27 // case 3
28 codeword="000";
29 case_func(codeword,dataword); // calling the
    function

```

---

### Scilab code Exa 10.3 3 redundant bits codeword

```

1 clear all;
2 clc;
3 disp("-----Example 10.3-----")
4 table=["Datawords", "Codewords"; "00", "00000"; "01", "
    01011"; "10", "10101"; "11", "11110"];
5 codewords=["00000", "01011", "10101", "11110"];
6 disp(table); // display the table
7 dataword="01";
8 codeword="01011";
9 corrupt_codeword="01001";
10 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
11 for i=1:4 // check for each codeword
12     bit=strsplit(codewords(i));
13     r=strsplit(corrupt_codeword);

```

```

14     count=0;
15     for k=1:5
16         if(bit(k)==r(k)) // check each bit
17             continue;
18         else
19             count=count+1; // update the count of
                errorenous bits
20         end
21     end
22     if(count>1) // if more than 1 bit is errorenous
23         continue;
24     else
25         correct_codeword=codewords(i); // the
                correct codeword determined
26         break;
27     end
28 end
29 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

```

1 clear all;
2 clc;

```



```

3 disp("-----Example 10.4-----")
4 //words
5 x1=[0 0 0];
6 y1=[0 1 1];
7 x2=[1 0 1 0 1];
8 y2=[1 1 1 1 0];
9 // formula to find Hamming distance 'd'
10 d1=bitxor(x1,y1);
11 d2=bitxor(x2,y2);
12 function [count]= num_of_ones (d)// function to find
    the number of ones in a binary number
13     count=0;
14     for i=1:length(d)
15         if(d(i)== 1)
16             count = count+1; // number of one's
17         end
18     end
19 endfunction
20 d=num_of_ones(d1); // calling the function
21 printf("\nThe Hamming distance d(000, 011) is %d.\n"
    ,d); // display result
22 d=num_of_ones(d2); // calling the function
23 printf("\nThe Hamming distance d(10101, 11110) is %d
    .\n",d); // display result

```

---

#### Scilab code Exa 10.5 minimum Hamming distance

```

1 clear all;
2 clc;
3 disp("-----Example 10.5-----")
4 //words
5 x1=[0 0 0];
6 x2=[0 1 1];
7 x3=[1 0 1];
8 x4=[1 1 0];

```

```

 9 //function to find Hamming distance
10 function [d]=hamming_distance(x,y)
11     xd=bitxor(x,y);
12     d=num_of_ones(xd);
13 endfunction
14 function [count]= num_of_ones (d)// function to find
    the number of ones in a binary number
15     count=0;
16     for i=1:length(d)
17         if(d(i)== 1)
18             count = count+1; // number of ones
19         end
20     end
21 endfunction
22 d1=hamming_distance(x1,x2);
23 printf("\nThe Hamming distance d(000, 011) is %d.\n"
    ,d1); // display result
24 d2=hamming_distance(x1,x3);
25 printf("\nThe Hamming distance d(000, 101) is %d.\n"
    ,d2); // display result
26 d3=hamming_distance(x1,x4);
27 printf("\nThe Hamming distance d(000, 110) is %d.\n"
    ,d3); // display result
28 d4=hamming_distance(x2,x3);
29 printf("\nThe Hamming distance d(011, 101) is %d.\n"
    ,d4); // display result
30 d5=hamming_distance(x2,x4);
31 printf("\nThe Hamming distance d(011, 110) is %d.\n"
    ,d5); // display result
32 d6=hamming_distance(x3,x4);
33 printf("\nThe Hamming distance d(101, 110) is %d.\n"
    ,d6); // display result
34 dmin=min(d1,d2,d3,d4,d5,d6);
35 printf("\nThe minimum Hamming distance dmin is %d.",
    dmin); // display result

```

---

## Scilab code Exa 10.6 minimum Hamming distance 2

```
1 clear all;
2 clc;
3 disp("-----Example 10.6-----")
4 //words
5 x1=[0 0 0 0 0];
6 x2=[0 1 0 1 1];
7 x3=[1 0 1 0 1];
8 x4=[1 1 1 1 0];
9 //function to find Hamming distance
10 function [d]=hamming_distance(x,y)
11     xd=bitxor(x,y);
12     d=num_of_ones(xd);
13 endfunction
14 function [count]= num_of_ones (d)// function to find
    the number of ones in a binary number
15     count=0;
16     for i=1:length(d)
17         if(d(i)== 1)
18             count = count+1; //number of ones
19         end
20     end
21 endfunction
22 d1=hamming_distance(x1,x2);
23 printf("\nThe Hamming distance d(OOOO0, 01011) is %d
    .\n",d1); // display result
24 d2=hamming_distance(x1,x3);
25 printf("\nThe Hamming distance d(OOOO0, 10101) is %d
    .\n",d2); // display result
26 d3=hamming_distance(x1,x4);
27 printf("\nThe Hamming distance d(OOOO0, 11110) is %d
    .\n",d3); // display result
28 d4=hamming_distance(x2,x3);
```

```

29 printf("\nThe Hamming distance d(01011, 10101) is %d
   .\n",d4); // display result
30 d5=hamming_distance(x2,x4);
31 printf("\nThe Hamming distance d(01011, 11110) is %d
   .\n",d5); // display result
32 d6=hamming_distance(x3,x4);
33 printf("\nThe Hamming distance d(10101, 11110) is %d
   .\n",d6); // display result
34 dmin=min(d1,d2,d3,d4,d5,d6);
35 printf("\nThe minimum Hamming distance dmin is %d.",
   dmin); // display result

```

---

**Scilab code Exa 10.7** error detection and correction

```

1 clear all;
2 clc;
3 disp("-----Example 10.7-----")
4 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 (101)\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

```

1 clear all;
2 clc;
3 disp("-----Example 10.8-----")

```

```

4 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

```

1 clear all;
2 clc;
3 disp("-----Example 10.9-----")
4 dmin=4; // minimum Hamming distance
5 s=dmin-1; //error detection
6 t=(dmin-1)/2; //error correction
7 // display result
8 printf("This code guarantees the detection of up to
   %d errors.\n\n",s);
9 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

```

1 clear all;
2 clc;
3 disp("-----Example 10.10-----")

```

```

4 // function to check if given scheme is a linear
  block code
5 function []=check_linearcode (c1,c2,c3,c4)
6     if(bitxor(c1,c2)==c1|bitxor(c1,c2)==c2|bitxor(c1
      ,c2)==c3|bitxor(c1,c2)==c4)
7         if(bitxor(c1,c3)==c1|bitxor(c1,c3)==c2|
          bitxor(c1,c3)==c3|bitxor(c1,c3)==c4)
8             if(bitxor(c1,c4)==c1|bitxor(c1,c4)==c2|
                bitxor(c1,c4)==c3|bitxor(c1,c4)==c4)
9                 if(bitxor(c2,c3)==c1|bitxor(c2,c3)==
                    c2|bitxor(c2,c3)==c3|bitxor(c2,c3)
                      )==c4)
10                    if(bitxor(c2,c4)==c1|bitxor(c2,
                        c4)==c2|bitxor(c2,c4)==c3|
                          bitxor(c2,c4)==c4)
11                        if(bitxor(c3,c4)==c1|bitxor(
                            c3,c4)==c2|bitxor(c3,c4)
                              )==c3|bitxor(c3,c4)==c4)
12                            printf("\nThis scheme is
                                a linear block code
                                because the result of
                                XORing any codeword
                                with any other
                                codeword is a valid
                                codeword.\n");
13                        else
14                            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"
                                    );
15                            end
16                        else
17                            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");
18         end
19     else
20         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");
21     end
22 else
23     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");
24     end
25 else
26     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")
                ;
27     end
28 else
29     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");
30     end
31 endfunction
32
33 // 1) Table 10.1
34 c1=[0 0 0];
35 c2=[0 1 1];

```

```

36 c3=[1 0 1];
37 c4=[1 1 0];
38 printf("\n1");
39 check_linearcode(c1,c2,c3,c4); //calling the
    function
40
41 // 2) Table 10.2
42 c1=[0 0 0 0 0];
43 c2=[0 1 0 1 1];
44 c3=[1 0 1 0 1];
45 c4=[1 1 1 1 0];
46 printf("\n2");
47 check_linearcode(c1,c2,c3,c4); //calling the
    function

```

---

#### Scilab code Exa 10.11 d min calculation

```

1 clear all;
2 clc;
3 disp("-----Example 10.11-----")
4 function [count]= num_of_ones (d)// function to find
    the number of ones in a binary number
5     count=0;
6     for i=1:length(d)
7         if(d(i)== 1)
8             count = count+1; // number of ones
9         end
10    end
11 endfunction
12 //1) Table 10.1
13 c1=[0 0 0];
14 c2=[0 1 1];
15 c3=[1 0 1];
16 c4=[1 1 0];
17 o2=num_of_ones(c2);

```



```

18 o3=num_of_ones(c3);
19 o4=num_of_ones(c4);
20 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.
21 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
22
23 // 2) Table 10.2
24 c1=[0 0 0 0 0];
25 c2=[0 1 0 1 1];
26 c3=[1 0 1 0 1];
27 c4=[1 1 1 1 0];
28 o2=num_of_ones(c2);
29 o3=num_of_ones(c3);
30 o4=num_of_ones(c4);
31 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.
32 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

```

1 clear all;
2 clc;
3 disp("-----Example 10.12-----")
4 dataword=1011;
5 function []= display (codeword) // function to
    display the result according to the codeword
    recieved at the reciever

```

```

6      select codeword
7      case 10111
8          printf("\n1)No error occurs;the received
              codeword is %d.The syndrome is 0. The
              dataword 1011 is created.\n",codeword);
9      case 10011
10         printf("\n2)One single-bit error changes a1.
              The received codeword is %d.The syndrome
              is 1.No dataword is created.\n",codeword)
              ;
11      case 10110
12         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);
13      case 00110
14         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");
15      case 01011
16         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");

```

```

17     end
18 endfunction
19
20 //codeword = 10111
21 codeword = 10111;
22 display(10111); //calling the function
23 //codeword = 10011
24 codeword = 10011;
25 display(10011); //calling the function
26 //codeword = 10110
27 codeword = 10110;
28 display(10110); //calling the function
29 //codeword = 00110
30 codeword = 00110;
31 display(00110); //calling the function
32 //codeword = 01011
33 codeword = 01011;
34 display(01011); //calling the function
35 funcprot(0);

```

---

### Scilab code Exa 10.13 path of three datawords

```

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

```

```

        // r2=a0+a1+a3
8      codeword=string(dataword(1))+string(dataword(2))
        +string(dataword(3))+string(dataword(4))+
        string(r2)+string(s1)+string(r0); // form the
        codeword
9  endfunction
10 function[syndrome] = generate_syndrome(
    codeword_recieved) // function to generate
    syndrome at the reciever
11  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
12  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
13  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
14  syndrome=string(s2)+string(s1)+string(s0); //
        the syndrome formed
15 endfunction
16
17 function[]=find_error (syndrome,dataword,codeword,
    codeword_recieved) // functin to find the error
    bit and display the final corrected data word
18  select syndrome
19  case "000"
20      dw=string(dataword(1))+string(dataword(2))+
        string(dataword(3))+string(dataword(4));
21      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));
22 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);
23 case "001"
24 dw=string(dataword(1))+string(dataword(2))+
    string(dataword(3))+string(dataword(4));
25 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));
26 error_bit="q0";
27 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);
28 case "010"
29 dw=string(dataword(1))+string(dataword(2))+
    string(dataword(3))+string(dataword(4));
30 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));
31 error_bit="q1";
32 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);
33     case "011"
34         dw=string(dataword(1))+string(dataword(2))+
           string(dataword(3))+string(dataword(4));
35         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));
36         error_bit="b2";
37         fdw=string(codeword_recieved(1))+string(
           bitcmp(codeword_recieved(2),1))+string(
           codeword_recieved(3))+string(
           codeword_recieved(4)); // corrected
           dataword
38         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);
39     case "100"
40         dw=string(dataword(1))+string(dataword(2))+
           string(dataword(3))+string(dataword(4));
41         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));
42         error_bit="q2";

```

```

43     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);
44     case "101"
45         dw=string(dataword(1))+string(dataword(2))+
            string(dataword(3))+string(dataword(4));
46         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));
47         error_bit="b0";
48         fdw=string(codeword_recieved(1))+string(
            codeword_recieved(2))+string(
            codeword_recieved(3))+string(bitcmp(
            codeword_recieved(4),1)); // corrected
            dataword
49         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);
50         case "110"
51             dw=string(dataword(1))+string(dataword(2))+
                string(dataword(3))+string(dataword(4));
52             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));

```

```

53     error_bit="b3";
54     fdw=string(bitcmp(codeword_recieved(1),1))+
        string(codeword_recieved(2))+string(
            codeword_recieved(3))+string(
            codeword_recieved(4)); // corrected
        dataword
55     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);
56     case "111"
57         dw=string(dataword(1))+string(dataword(2))+
            string(dataword(3))+string(dataword(4));
58         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));
59         error_bit="b1";
60         fdw=string(codeword_recieved(1))+string(
            codeword_recieved(2))+string(bitcmp(
            codeword_recieved(3),1))+string(
            codeword_recieved(4)); // corrected
            dataword
61         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);
62     end
63 endfunction
64
65 // 1)

```



```

66 dataword=[0 1 0 0];
67 codeword=generate_codeword(dataword); // calling the
    function
68 codeword_recieved=[0 1 0 0 0 1 1];
69 syndrome=generate_syndrome(codeword_recieved) //
    calling the function
70 printf("\n1");
71 find_error(syndrome,dataword,codeword,
    codeword_recieved); // calling the function
72
73 // 2)
74 dataword=[0 1 1 1];
75 codeword=generate_codeword(dataword); // calling the
    function
76 codeword_recieved=[0 0 1 1 0 0 1];
77 syndrome=generate_syndrome(codeword_recieved) //
    calling the function
78 printf("\n\n2");
79 find_error(syndrome,dataword,codeword,
    codeword_recieved); // calling the function
80
81 // 3)
82 dataword=[1 1 0 1];
83 codeword=generate_codeword(dataword); // calling the
    function
84 codeword_recieved=[0 0 0 1 0 0 0];
85 syndrome=generate_syndrome(codeword_recieved) //
    calling the function
86 printf("\n\n3");
87 find_error(syndrome,dataword,codeword,
    codeword_recieved); // calling the function
88 printf("\nThis is the wrong dataword. This shows
    that Hamming code cannot correct two errors.");

```

---

Scilab code Exa 10.14 calculate k and n

```

1 clear all;
2 clc;
3 disp("-----Example 10.14-----")
4 // function to chekx if k >=7 and display
   appropriate result
5 function [] = check (m)
6     n=2^m - 1;
7     k=n-m;
8     if( k > = 7)
9         printf(" m = %d :- The code is C(%d, %d) or
              k = %d and n = %d.\n",m,n,k,k,n);
10    else
11        printf(" m = %d :- n = %d .k = %d , which is
              less than 7. Hence doesnt satisfy the
              condition.\n",m,n,k);
12    end
13 endfunction
14 // case 1
15 m=3;
16 printf("\n1");
17 check(m); // calling the function
18 //case 2
19 m=4;
20 printf("\n2");
21 check(m); // calling the function

```

---

**Scilab code Exa 10.15** single bit error

```

1 clear all;
2 clc;
3 disp("-----Example 10.15-----")
4 //a) g(x)= x+1
5 gx="x+1";
6 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
7 //b) g(x)= x3
8 gx="x3";
9 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
10 //c) 1
11 gx="1";
12 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

```

1 clear all;
2 clc;
3 disp("-----Example 10.16-----");
4 x=poly(0,"x");
5 // a) x+1
6 g=x^1+1;
7 t=0;
8 // compute t
9 while(%T)
10     q=(x^t+1)/g;
11     if(q == 1)
12         break;
13     end

```

```

14     t=t+1;
15 end
16 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
17 // b) x^4+1
18 g=x^4+1;
19 t=0;
20 // compute t
21 while(%T)
22     q=(x^t+1)/g;
23     if(q == 1)
24         break;
25     end
26     t=t+1;
27 end
28 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
29 // c) x^7+x^6+1
30 g=x^7+x^6+1;
31 printf("c. This is a good choice for this purpose.\n
        \n"); // display result
32 // d) x^15+x^14+1
33 t=32768; // very large to compute
34 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

```

1 clear all;
2 clc;
3 disp("-----Example 10.17-----")
4 // a)  $x^6+1$ 
5 r=6;
6 p1=(1/2)^(r-1); // formula
7 p2=(1/2)^r; // formula
8 slip1=round(p1*100);
9 slip2=round(p2*1000);
10 // display the result
11 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);
12 // b)  $x^{18}+x^7x+1$ 
13 r=18;
14 p1=(1/2)^(r-1); // formula
15 p2=(1/2)^r; // formula
16 slip1=round(p1*10^6);
17 slip2=round(p2*10^6);
18 // display the result
19 printf("\nb. 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 \n
        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);
20 // c)  $x^{32}+x^{23}+x^7+1$ 
21 r=32;
22 p1=(1/2)^(r-1); // formula
23 p2=(1/2)^r; // formula
24 slip1=round(p1*10^10);
25 slip2=ceil(p2*10^10);
26 // display the result
27 printf("\nc. 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

```
1 clear all;  
2 clc;  
3 disp("-----Example 10.18-----")  
4 // set of numbers sent  
5 n1=7;  
6 n2=11;  
7 n3=12;  
8 n4=0;  
9 n5=6;  
10 n_sum=n1+n2+n3+n4+n5; // find the sum  
11 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

```
1 clear all;  
2 clc;  
3 disp("-----Example 10.19-----")
```

```

4 // set of numbers sent
5 n1=7;
6 n2=11;
7 n3=12;
8 n4=0;
9 n5=6;
10 n_sum=n1+n2+n3+n4+n5; // find the sum
11 checksum= - n_sum; // formula
12 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

```

1 clear all;
2 clc;
3 disp("-----Example 10.20-----")
4 n=21;
5 // compute one's complement
6 bin=dec2bin(n);
7 s=strsplit(bin,1);
8 a=bin2dec(s(1));
9 b=bin2dec(s(2));
10 f=a+b;
11 complement=dec2bin(f,4); //1's complement
12 dec_complement=bin2dec(complement); // convert 1's
    complement to decimal
13 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

```
1 clear all;
2 clc;
3 disp("-----Example 10.21-----")
4 n=-6;
5 // compute 1's complement
6 d_complement=bitcmp(-n,4);
7 b_complement=dec2bin(d_complement,4);
8 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

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



```

17 // display the result
18 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);
19
20 // at the reciever
21 r_sum=n1+n2+n3+n4+n5+s_checksum; // find the sum
        including checksum sent by sender
22 r_bin=dec2bin(r_sum);
23 r=strsplit(r_bin,length(r_bin)-4);
24 c=bin2dec(r(1));
25 d=bin2dec(r(2));
26 e=c+d; // wrapping the sum
27 r_checksum=bitcmp(e,4); // complementing
28 // display the result
29 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);
30 // check if data is corrupt or not
31 if(r_checksum==0)
32     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.");
33 else
34     printf("The checksum is not zero ,hence the
            entire packet is dropped.");
35 end

```

### Scilab code Exa 10.23 Forouzan text checksum

```
1 clear all;
2 clc;
3 disp("-----Example 10.23-----")
4 // sender
5 text="Forouzan";
6 // computing the checksum
7 a=ascii(text);
8 h1=dec2hex(a(1));
9 h2=dec2hex(a(2));
10 h3=dec2hex(a(3));
11 h4=dec2hex(a(4));
12 h5=dec2hex(a(5));
13 h6=dec2hex(a(6));
14 h7=dec2hex(a(7));
15 h8=dec2hex(a(8));
16 // form the hexadecimal words
17 Fo=h1+h2;
18 ro=h3+h4;
19 uz=h5+h6;
20 an=h7+h8;
21 d1=hex2dec(Fo);
22 d2=hex2dec(ro);
23 d3=hex2dec(uz);
24 d4=hex2dec(an);
25 ps=d1+d2+d3+d4;
26 partial_sum=dec2hex(ps); // partial sum of the words
27 s=strsplit(partial_sum,[1 4]);
28 Sum=s(2)+dec2hex(hex2dec(s(1))+hex2dec(s(3))); //
    wrapping the sum
29 a=hex2dec(Sum)
30 c=bitcmp(a,16);
31 Checksum=dec2hex(c); // Checksum in hex
```

```

32 carries="1013";
33 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");
34 // display the process
35 printf("    a) Checksum at the sender site");
36 printf("\n        %s
    Carries\n",carries);
37 printf("\n        %s                Fo",
    Fo);
38 printf("\n        %s                ro",
    ro);
39 printf("\n        %s                uz",
    uz);
40 printf("\n        %s                an",
    an);
41 printf("\n        0000
    Checksum(initial)");
42 printf("\n        ----- \n");
43 printf("\n        %s                Sum(
    partial)",s(2)+s(3));
44 printf("\n        ---- \n");
45 printf("\n        %s",s(1));
46 printf("\n        ----- \n");
47 printf("\n        %s                Sum",
    Sum);
48 printf("\n        %s                Checksum(to
    send)",Checksum);
49
50 // reciever
51 Checksum_r=Checksum;
52 d_Sum=ps+c; // sum of data and checksum
53 partial_sum=dec2hex(d_Sum);
54 s=strsplit(partial_sum,[1 4]);
55 Sum=s(2)+dec2hex(hex2dec(s(1))+hex2dec(s(3))); //
    wrapping the sum
56 a=hex2dec(Sum)

```

```

57 c=bitcmp(a,16);
58 Checksum=dec2hex(c); // checksum in hex
59 // display the process
60 printf("\n\n          b) Checksum at the reciever site"
);
61 printf("\n\n          %s
Carries\n",carries);
62 printf("\n          %s          Fo",
Fo);
63 printf("\n          %s          ro",
ro);
64 printf("\n          %s          uz",
uz);
65 printf("\n          %s          an",
an);
66 printf("\n          %s
Checksum( recieved)",Checksum_r);
67 printf("\n          ----- \n");
68 printf("\n          %s          Sum(
partial)",s(2)+s(3));
69 printf("\n          ---- \n");
70 printf("\n          %s",s(1));
71 printf("\n          ----- \n");
72 printf("\n          %s          Sum",
Sum);
73 printf("\n          %s000          Checksum
(new)",Checksum);

```

---

# Chapter 11

## Data link control

Scilab code Exa 11.1 Simplest protocol

```
1 clear all;
2 clc;
3 disp("-----Example 11.1-----")
4 //explain the example
5 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.");
6 // display the figure
7 clf();
8 xname("-----Example 11.1-----");
9 xrects([.3 .6;.7 .7;.05 .05;.06 .06]);
10 xset("font size",3);
11 xstring(.3,.75,"Sender");
12 xstring(.6,.75,"Reciever");
```

```

13 xstring(.32,.65,"A");
14 xstring(.62,.65,"B");
15 xstring(.22,.327,"Request");
16 xstring(.22,.427,"Request");
17 xstring(.22,.527,"Request");
18 xstring(.67,.29,"Arrival");
19 xstring(.67,.39,"Arrival");
20 xstring(.67,.49,"Arrival");
21 xstring(.35,.52,"Frame",8);
22 xstring(.35,.42,"Frame",8);
23 xstring(.35,.32,"Frame",8);
24 xarrows([.29 .325],[.55 .55],.3);
25 xarrows([.29 .325],[.45 .45],.3);
26 xarrows([.29 .325],[.35 .35],.3);
27 xarrows([.625 .66],[.5 .5],.3);
28 xarrows([.625 .66],[.4 .4],.3);
29 xarrows([.625 .66],[.3 .3],.3);
30 xset("color",4.9);
31 xfpoly([.325 .625 .625 .325],[.56 .51 .46 .51]);
32 xfpoly([.325 .625 .625 .325],[.46 .41 .36 .41]);
33 xfpoly([.325 .625 .625 .325],[.36 .31 .26 .31]);
34 xset("color",0);
35 xset("line style",2);
36 xarrows([.325 .325],[.64 .14],.3);
37 xarrows([.625 .625],[.64 .14],.3);
38 xstring(.3,.1,"Time");
39 xstring(.6,.1,"Time");

```

---

### Scilab code Exa 11.2 Stop and wait

```

1 clear all;
2 clc;
3 disp("-----Example 11.2-----")

```

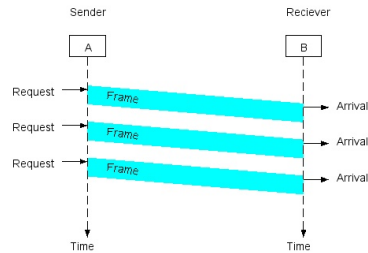


Figure 11.1: Simplest protocol

```

4 //explain the example
5 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.");
6 // display the figure
7 clf();
8 xname("-----Example 11.2-----");
9 xrects([.3 .6;.7 .7;.05 .05;.06 .06]);
10 xset("font size",3);
11 xstring(.3,.75,"Sender");
12 xstring(.6,.75,"Reciever");
13 xstring(.32,.65,"A");
14 xstring(.62,.65,"B");
15 xstring(.35,.52,"Frame",8);
16 xstring(.35,.3,"Frame",8);
17 xstring(.58,.41,"ACK",-8);
18 xstring(.58,.19,"ACK",-8);
19 xstring(.22,.527,"Request");

```

```

20 xstring(.22,.3,"Request");
21 xstring(.24,.38,"Arrival");
22 xstring(.24,.16,"Arrival");
23 xstring(.67,.49,"Arrival");
24 xstring(.67,.27,"Arrival");
25 xarrows([.29 .325],[.55 .55],.3);
26 xarrows([.29 .325],[.32 .32],.3);
27 xarrows([.325 .29],[.39 .39],.3);
28 xarrows([.325 .29],[.17 .17],.3);
29 xarrows([.625 .66],[.5 .5],.3);
30 xarrows([.625 .66],[.28 .28],.3);
31 xset("color",4.9);
32 xfpoly([.325 .625 .625 .325],[.56 .51 .46 .51]);
33 xfpoly([.325 .625 .625 .325],[.34 .29 .24 .29]);
34 xfpoly([.325 .625 .625 .325],[.41 .46 .41 .36]);
35 xfpoly([.325 .625 .625 .325],[.19 .24 .19 .14]);
36 xset("color",0)
37 xset("line style",2);
38 xarrows([.325 .325],[.64 .1],.3);
39 xarrows([.625 .625],[.64 .1],.3);
40 xstring(.3,.06,"Time");
41 xstring(.6,.06,"Time");
42 xset("font size",8);
43 xstring(.46,.08,".");
44 xstring(.46,.06,".");
45 xstring(.46,.1,".");

```

---

### Scilab code Exa 11.3 Stop and wait ARQ

```

1 clear all;
2 clc;
3 disp("-----Example 11.3-----")
4 // example explanation

```



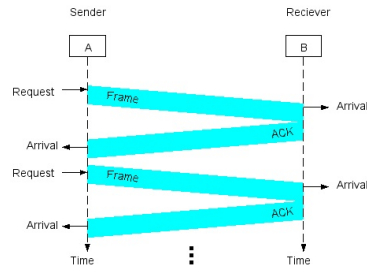


Figure 11.2: Stop and wait

```
5 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

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

```

8 utilization_percentage = (frame_length/
    bandwidth_delay_product)*100; // formula
9 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

```

1 clear all;
2 clc;
3 disp("-----Example 11.5-----")
4 bandwidth= 1*10^6; // 1 Mbps
5 delay = 20*10^-3; // 20 ms
6 bandwidth_delay_product=bandwidth*delay;
7 frame_length=1000; // each frame has 1000 bits
8 num_frames= 15; // The system can send up to 15
    frames during a round trip.
9 utilization_percentage = (frame_length*num_frames/
    bandwidth_delay_product)*100; // formula
10 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

```

1 clear all;
2 clc;
3 disp("-----Example 11.6-----")

```

```

4 // example explanation
5 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");
6 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

```

1 clear all;
2 clc;
3 disp("-----Example 11.7-----")
4 // example explanation
5 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. When the sender is responding to this event, it cannot accept the triggering of other events. This means that when ACK 2 arrives, the sender is still busy with sending frame 3. The physical layer must wait until this event is completed and the data link layer goes back to its sleeping state. A vertical line indicates this delay in the figure. It is the same story with ACK 3; but when ACK 3 arrives, the sender is busy responding to ACK 2. It happens again 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

```

1 clear all;
2 clc;
3 disp("-----Example 11.8-----")
4 // example explanation
5 printf("This example shows the behaviour of
   Selective Repeat when a frame is lost.\n\n");
6 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.");
7 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.  
At the next arrival, frame 3 arrives and is marked and stored, but still none of the frames can be delivered.  
Only at the last arrival, when finally a copy of frame 1 arrives, can frames 1, 2, and 3 be delivered to the network layer.  
There are two conditions for the delivery of frames to the network layer: First, a set of consecutive frames must have arrived.  
Second, the set starts from the beginning of the window.  
After the first arrival, there was only one frame and it started from the beginning of the window. After the last arrival, there are three frames and the first one starts from the beginning of the window.”);

8 `printf("\n\nA NAK is sent after the second arrival, but not after the third, although both situations look the same.\n\nThe reason is that the protocol does not want to crowd the network with unnecessary NAKs and unnecessary resent frames.\n\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.”);`

9 `printf("\n\nOnly two ACKs are sent here. The first one acknowledges only the first frame; the second one acknowledges three frames.\n\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

```
1 clear all;
2 clc;
3 disp("-----Example 11.9-----")
4 // example explanation
5 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
  \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

```
1 clear all;
2 clc;
3 disp("-----Example 11.10-----")
4 // example explanation
5 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 frames 1 and 0 and indicating that it expects frame 2 to arrive next. Node B transmits its second and third I-frames (numbered 1 and 2) before accepting further frames from node A. 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. Node A has sent all its data. Therefore, it cannot piggyback an acknowledgment onto an I-frame and sends an S-frame instead. 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 accepted and that A is now expecting frame number 3.”)

---

#### Scilab code Exa 11.11 Piggybacking with Error

```

1 clear all;
2 clc;
3 disp("-----Example 11.11-----")
4 // example explanation
5 printf("This example shows an exchange in which a
   frame is lost. The sequence of events that occur
   is as follows : Node B sends three data
   frames (0, 1, and 2), but frame 1 is lost.
   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 with the special use of
   an REJ frame as a NAK frame. 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. 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

```
1 clear all;
2 clc;
3 disp("-----Example 11.12-----")
4 // example explanation
5 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");
6 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");
7 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");
8 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

```
1 clear all;
2 clc;
3 disp("-----Example 12.1-----")
4 d=600*10^3; // 600 km
5 speed = 3*10^8; // 3*10^8 m/s
6 Tp=(d/speed)*10^3; // propagation time
7
8 // a) K=1
9 K=[0 1]; // range
10 TB1=Tp*K(1);
11 TB2=Tp*K(2);
12 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
13 // b) K=2
14 K=[0 1 2 3]; // range
15 TB1=Tp*K(1);
16 TB2=Tp*K(2);
17 TB3=Tp*K(3);
18 TB4=Tp*K(4);
19 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
20 // c) K=3
21 K=[0 1 2 3 4 5 6 7]; //range
22 TB1=Tp*K(1);
23 TB2=Tp*K(2);
24 TB3=Tp*K(3);
25 TB4=Tp*K(4);
26 TB5=Tp*K(5);
27 TB6=Tp*K(6);
28 TB7=Tp*K(7);
29 TB8=Tp*K(8);
30 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
31 // d) K >10
32 printf("\nd)K>10 :- If K > 10, it is normally set
    to 10.") // display result

```

---

### Scilab code Exa 12.2 Collision free ALOHA

```

1 clear all;
2 clc;
3 disp("-----Example 12.2-----")
4 frame_bits=200;
5 datarate=200*10^3; // 200 kbps
6 Tfr=frame_bits/datarate;
7 Tv=2*Tfr; // vulnerable time
8 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

```
1 clear all;
2 clc;
3 disp("-----Example 12.3-----")
4 frame_bits=200;
5 datarate=200*10^3; // 200 kbps
6 Tfr=frame_bits/datarate;
7 printf("\nThe frame transmission time is %d ms.\n",
      Tfr*10^3);
8
9 function[S]=s_func (frame_rate) // function to
      determine S
10     G=frame_rate*10^-3; // load
11     S=G*(%e^(-2*G)); // formula
12     percent=S*100;
13     printf("S = %4.3f or %3.1f percent.",S,percent);
14 endfunction
15
16 //a. 1000 frames per second
17 frame_rate=1000;
18 printf("\na)");
19 S=s_func(frame_rate); //calling the function
20 throughput=S*frame_rate;
21 printf("\nThe throughput is %d frames.Only %d frames
      out of %d will probably survive.\n",throughput,
      throughput,frame_rate); // display result
22 //b. 500 frames per second
23 frame_rate=500;
24 printf("\nb)");
25 S=s_func(frame_rate); //calling the function
26 throughput=rat(S,10^-2)*frame_rate; // approximation
27 // display result
```

```

28 printf("\nThe throughput is %d frames.Only %d frames
      out of %d will probably survive.\n",throughput ,
      throughput,frame_rate);
29 printf("Note that this is the maximum throughput
      case , percentage wise.\n");
30
31 //c. 250 frames per second
32 frame_rate=250;
33 printf("\nc");
34 S=s_func(frame_rate); //calling the function
35 throughput=rat(S,10^-1.5)*frame_rate; //
      approximation
36 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

```

1 clear all;
2 clc;
3 disp("-----Example 12.4-----")
4 frame_bits=200;
5 datarate=200*10^3; // 200 kbps
6 Tfr=frame_bits/datarate;
7 printf("\nThe frame transmission time is %d ms.\n",
      Tfr*10^3);
8
9 function [S]=s_func (frame_rate) // function to
      determine S
10     G=frame_rate*10^-3; // load
11     S=G*(%e^(-G)); //formula
12     percent=S*100;
13     printf("S = %4.3f or %3.1f percent.",S,percent);
14 endfunction
15

```

```

16 //a. 1000 frames per second
17 frame_rate=1000;
18 printf("\na");
19 S=s_func(frame_rate); //calling the function
20 throughput=rat(S,10^-2.5)*frame_rate;//
    approximation
21 // display result
22 printf("\nThe throughput is %d frames.Only %d frames
    out of %d will probably survive.\n",throughput,
    throughput,frame_rate);
23 printf("Note that this is the maximum throughput
    case, percentage wise.\n");
24
25 //b. 500 frames per second
26 frame_rate=500;
27 printf("\nb");
28 S=s_func(frame_rate); //calling the function
29 throughput=S*frame_rate;
30 printf("\nThe throughput is %d frames.Only %d frames
    out of %d will probably survive.\n",throughput,
    throughput,frame_rate); // display result
31
32 //c. 250 frames per second
33 frame_rate=250;
34 printf("\nc");
35 S=s_func(frame_rate); //calling the function
36 throughput=S*frame_rate;
37 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

```

1 clear all;

```

```

2  clc;
3  disp("-----Example 12.5-----")
4  bandwidth = 10*10^6; // 10 Mbps
5  Tp=25.6*10^-6; // 25.6 microseconds
6  Tfr=2*Tp; // formula
7  min_frame_size = bandwidth*Tfr; // formula
8  bytes=min_frame_size/8; // 1 byte = 8 bits
9  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

```

1  clear all;
2  clc;
3  disp("-----Example 12.6-----")
4  // use the rows of W2 and W4 in the solution
5  W2=[1 1;1 -1];
6  W4=[1 1 1 1;1 -1 1 -1;1 1 -1 -1;1 -1 -1 1];
7  //a. Two stations
8  C1= W2(1,:); //select 1st row of W2
9  C2= W2(2,:); // select 2nd row of W2
10 // display result
11 disp("a)The chips for a two-station network are ");
12 disp(C1)
13 disp("and")
14 disp(C2)
15
16 //b. Four stations
17 C1= W4(1,:); // select 1st row of W4
18 C2= W4(2,:); // select 2nd row of W4
19 C3= W4(3,:); // select 3rd row of W4
20 C4= W4(4,:); // select 4th row of W4
21 // display result

```

```

22 disp("b)The chips for a four-station network are ");
23 disp(C1)
24 printf(" ,")
25 disp(C2)
26 printf(" ,")
27 disp(C3)
28 printf(" and")
29 disp(C4)

```

---

#### Scilab code Exa 12.7 Number of sequences

```

1 clear all;
2 clc;
3 disp("-----Example 12.7-----")
4 stations=90;
5 m=7;
6 N=2^m; // formula
7 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

```

1 clear all;
2 clc;
3 disp("-----Example 12.8-----")
4 //Proof
5 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)*

```



$$\begin{aligned}
c1 \backslash n &= d1 * c1 * c1 + d2 * c2 * c1 + d3 * c3 * c1 + d4 * \\
c4 * c1 \backslash n &= d1 * N + d2 * 0 + d3 * 0 + d4 * 0 \backslash n \\
&= d1 * N \backslash n
\end{aligned}$$

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

```
1 clear all;
2 clc;
3 disp("-----Example 13.1-----")
4
5 // function to check if thhe 2nd hex digit from the
  left is even or odd
6 function []=check (a)
7
8     s=strsplit(a,[1,2]); // extract the 2nd hex
  digit from left
9     d= hex2dec(s(2));
10    bin=dec2bin(d,4); // convert to binary
11    bits=strsplit(bin,3); // least significant bit
12    lb=bits(2);
13
14    if(lb=='0') // check if even or odd
15        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);
16    else
```

```

17         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);
18     end
19 endfunction
20
21 // a) 4A:30:10:21:10:1A
22 a="4A:30:10:21:10:1A";
23 printf("\na");
24 check(a); // calling the function
25
26 // b) 47:20:1B:2E:08:EE
27 b="47:20:1B:2E:08:EE";
28 printf("\nb");
29 check(b); // calling the function
30
31 // c) FF:FF:FF:FF:FF:FF
32 c="FF:FF:FF:FF:FF:FF";
33 s = strsplit(c,":",6); // split into 2 hex digits
34 for i=1:6
35     if(s(i)=="FF") // check if equal to FF
36         continue;
37     else
38         break;
39     end
40
41 end
42 if(i==6)
43     printf("\nc)This is a broadcast address because
           all digits are Fs.") //print the result
44 end

```

---

Scilab code Exa 13.2 Sending the address

```

1 clear all;
2 clc;
3 disp("-----Example 13.2-----")
4 // address = 47:20:1B:2E:08:EE
5 address = "47:20:1B:2E:08:EE";
6
7 function [bin_str]=bin_address (address) // function
   to convert address in hexadecimal to binary
8     b=strsplit(address);
9     bin_str="";
10    for i=1:length(address)
11        if(modulo(i,3)==0) // to exclude ":"
12            continue;
13        else
14            d=hex2dec(b(i));
15            bin=dec2bin(d,4);
16            bin_str=bin_str+bin; // address in
                binary
17        end
18
19    end
20 endfunction
21 bin_str=bin_address(address);
22
23 function [addr] = revstr(bin_str) // function to
   reverse the nibbles in the address
24     str_nibble=strsplit(bin_str,[4 8 12 16 20 24 28
        32 36 40 44 ]);
25     addr="";
26     for i=1:12
27         rev=strrev(str_nibble(i));
28         addr=addr+rev; // resultant string
29     end
30 endfunction
31 addr=revstr(bin_str);
32 bytes=strsplit(addr,[8 16 24 32 40]); // spilt into
   bytes
33

```

```

34 function [bytes]=exchgnib(bytes) // function to
    exchange the nibbles in each byte
35     for i=1:6
36         nib=strsplit(bytes(i),4);
37
38         temp=nib(1); // exachanging nibbles
39         nib(1)=nib(2);
40         nib(2)=temp;
41         bytes(i)=nib(1)+nib(2);
42     end
43 endfunction
44 bytes=exchgnib(bytes); // final address
45 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

```
1 clear all;
2 clc;
3 disp("-----Example 16.1-----")
4 C=1/100;
5 dist_moon=384000; // 384,000 km
6 radius_earth = 6378; // 6378 km
7 distance=dist_moon+radius_earth ;// total distance
   in km
8 Period=C*((distance)^1.5); //formula
9 month=round(Period/2592000); // 1 month =
   60*60*24*30=2592000 seconds
10 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

```
1 clear all;
2 clc;
3 disp("-----Example 16.2-----")
4 C=1/100;
5 orbit=35786; // 35,786 km
6 radius_earth = 6378; // 6378 km
7 distance=orbit+radius_earth ;// total distance in
  km
8 Period=C*((distance)^1.5); //formula
9 hour=round(Period/3600); // 1 hour = 60*60=3600
  seconds
10 printf("According to Keplers law, the period of the
  satellite is %d s or %d hours.",floor(Period),
  hour);
11 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

```
1 clear all;
2 clc;
3 disp("-----Example 17.1-----")
4 frame_rate=8000; // frames per sec
5 frame=9*(1*90); // each frame is made of 9 by (1x90)
   bytes
6 bits=8; // 1byte = 8 bits
7 STS1_data_rate=frame_rate*frame*bits; // formula
8 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

```
1 clear all;
2 clc;
3 disp("-----Example 17.2-----")
4 frame_rate=8000; // frames per sec
```



```

5 frame=9*(3*90); // each frame is made of 9 by (3x90)
   bytes
6 bits=8; // 1byte = 8 bits
7 STS3_data_rate=frame_rate*frame*bits; // formula
8 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

```

1 clear all;
2 clc;
3 disp("-----Example 17.3-----")
4 // for all STS-1 , STS-3 and STS-n frame rate is
   same hence frame duration is same
5 frame_rate=8000; // frames per sec
6 frame_duration=1/frame_rate;
7 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

```

1 clear all;
2 clc;
3 disp("-----Example 17.4-----")
4 frame_rate=8000; // frames per sec
5 //each frame is made of 9 rows and 86 columns
6 rows=9;
7 columns=(1*86);
8 bits=8; // 1byte = 8 bits

```

```

9 STS1_user_data_rate=frame_rate*rows*columns*bits; //
  formula
10 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

```

1 clear all;
2 clc;
3 disp("-----Example 17.5-----")
4 byte_number=650;
5 temp=dec2hex(byte_number); // convert to hexadecimal
6 hex='0'+temp;
7 h=strsplit(hex,2); // split the hexadecimal number
  into 2
8 H1=h(1);
9 H2=h(2);
10 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

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

```

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

```

---

### Scilab code Exa 19.2 Binary notation

```

1 clear all;
2 clc;
3 disp("-----Example 19.2-----")
4 // a) 111.56.45.78
5 n=8; //number of bits i.e 1 byte
6 a3=dec2bin(111,n); // convert decimal
    numbers to binary numbers
7 a2=dec2bin(56,n);
8 a1=dec2bin(45,n);
9 a0=dec2bin(78,n);
10 disp("a) Binary notation :- "+a3+" "+a2+" "+a1+" "+
    a0) //result in binary notation
11 // b) 221.34.7.82
12 b3=dec2bin(221,n); //convert decimal
    numbers into binary numbers
13 b2=dec2bin(34,n);
14 b1=dec2bin(7,n);
15 b0=dec2bin(82,n);
16 disp("b) Binary notation :- "+b3+" "+b2+" "+b1+" "+
    b0) //result in binary nootation

```

---

### Scilab code Exa 19.3 Find the error

```

1 clear all;
2 clc;

```

```

3 disp("-----Example 19.3-----")
4 // a) 111.56.045.78
5 disp("a) There must be no leading zero (045).") //
   display the result
6 //b) 221.34.7.8.20
7 disp("b) There can be no more than four numbers in
   an IPv4 address.") //display the result
8 //c) 75.45.301.14
9 disp("c) Each number needs to be less than or equal
   to 255 (301 is outside this range).") //display
   the result
10 //d) 11100010.23.14.67
11 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

```

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

```

```

14         if(c(1)=="110")
15             disp(" The first 3 bits are 110. This is
                a class C address.")
16         else
17             c=strsplit(q,4);
18             if(c(1)=="1110")
19                 disp(" The first 4 bits are 1110.
                    This is a class D address.")
20             elseif(c(1)=="1111")
21                 disp(" The first 4 bits are 1111.
                    This is a class E address.")
22     end
23 end
24 end
25 end
26 endfunction //end of function
27
28 function [] = byteclass (q) //function to
    determine the class of a address in decimal
    notation
29 if(q>=0 & q<= 127)
30     disp(" The first byte is between 0 and 127; the
        class is A.")
31 elseif(q>=128 & q<=191)
32     disp(" The first byte is between 128 and 191;
        the class is B.")
33 elseif( q>=192 & q<=223)
34     disp(" The first byte is between 192 and 223;
        the class is C.")
35 elseif( q>=224 & q<=239)
36     disp(" The first byte is between 224 and 239;
        the class is D.")
37 elseif(q>=240 & q<=255)
38     disp(" The first byte is between 240 and 255;
        the class is E.")
39 end
40 endfunction //end of function
41

```

```

42
43 //a) 00000001 00001011 00001011 11101111
44 q="00000001";
45 printf("\na");
46 binclass(q); //calling the function
47 //b) 11000001 10000011 00011011 11111111
48 q="11000001";
49 printf("\nb");
50 binclass(q); //calling the function
51 //c) 14.23.120.8
52 q=14;
53 printf("\nc");
54 byteclass(q); //calling the function
55 //d) 252.5.15.111
56 q=252;
57 printf("\nd");
58 byteclass(q); //calling the function

```

---

### Scilab code Exa 19.5 block of addresses

```

1 clear all;
2 clc;
3 disp("-----Example 19.5-----")
4 n=16; //cidr
5 fa1=205; //bytes of 1st address in decimal
6 fa2=16;
7 fa3=37;
8 fa4=32;
9 fab1=dec2bin(fa1,8); //convert the bytes to binary
10 fab2=dec2bin(fa2,8);
11 fab3=dec2bin(fa3,8);
12 fab4=dec2bin(fa4,8);
13 la4=fa4+n-1; //determine the last byte of the last
    address
14 lab4=dec2bin(la4,8); //last address in binary

```

```

15 disp("The block of addresses is"); //display the
    results
16 disp("i) In binary notation :- 1st address = "+fab1+
    " "+fab2+" "+fab3+" "+fab4)
17 disp(" last address = "+fab1+" "+fab2+" "+fab3+" "+
    lab4)
18 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

```

1 clear all;
2 clc;
3 disp("-----Example 19.6-----")
4 //address :- 205.16.37.39/28
5 n=8; //number of bits i.e 1 byte
6 a3=dec2bin(205,n); // convert decimal
    numbers to binary numbers
7 a2=dec2bin(16,n);
8 a1=dec2bin(37,n);
9 a0=dec2bin(39,n);
10 disp(" Binary notation of the address is "+a3+" "+a2
    +" "+a1+" "+a0)
11 mask= 28;
12 num_of_zeros=32-mask; // calculate the number of
    bits to be set to zero
13 a=a3+a2+a1+a0;
14 p1=strsplit(a,mask); //truncate the binary address
15 p=p1(1);
16 for i= 1:num_of_zeros
17     p=p+'0'; //appending zeros
18 end
19 b=strsplit(p,[8 16 24 ]) //separate the bytes
20 b3=bin2dec(b(1)); // convert binary numbers

```



```

    to decimal numbers
21 b2=bin2dec(b(2));
22 b1=bin2dec(b(3));
23 b0=bin2dec(b(4));
24 disp(" Binary notation of first address:- "+b(1)+"
    "+b(2)+" "+b(3)+" "+b(4))
25 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

```

1 clear all;
2 clc;
3 disp("-----Example 19.7-----")
4 //address :- 205.16.37.39/28
5 n=8; //number of bits i.e 1 byte
6 a3=dec2bin(205,n); // convert decimal
    numbers to binary numbers
7 a2=dec2bin(16,n);
8 a1=dec2bin(37,n);
9 a0=dec2bin(39,n);
10 disp(" Binary notation of the address is "+a3+" "+a2
    +" "+a1+" "+a0)
11 mask= 28;
12 num_of_ones=32-mask; // calculate the number of bits
    to be set to one
13 a=a3+a2+a1+a0;
14 p1=strsplit(a,mask); //truncate the binary address
15 p=p1(1);
16 for i= 1:num_of_ones
17     p=p+'1'; //appending ones
18 end
19 b=strsplit(p,[8 16 24 ]) //separate the bytes
20 b3=bin2dec(b(1)); // convert binary numbers

```

```

    to decimal numbers
21 b2=bin2dec(b(2));
22 b1=bin2dec(b(3));
23 b0=bin2dec(b(4));
24 disp(" Binary notation of last address:- "+b(1)+" "
    +b(2)+" "+b(3)+" "+b(4))
25 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

```

1 clear all;
2 clc;
3 disp("-----Example 19.8-----")
4 //address :- 205.16.37.39/28
5 n=28; //cidr
6 exp_of_2 = 32-n;
7 num_of_addresses= 2^(exp_of_2); //formula to
    calculate number of addresses
8 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

```

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

```

```

8 a2=dec2bin(16,by);
9 a1=dec2bin(37,by);
10 a0=dec2bin(39,by);
11 disp("Binary notation of the address is "+a3+" "+a2+
      " "+a1+" "+a0)//display address in binary
      notation
12 a=a3+a2+a1+a0;
13 mask="";
14 for i = 1:n
15     mask=mask+'1'; //adding 1s to the mask
16 end
17 m=32-n;
18 for i = 1:m
19     mask=mask+'0'; //adding 0s to the mask
20 end
21 ma=strsplit(mask,[8 16 24]);
22 disp("The mask is "+ma(1)+" "+ma(2)+" "+ma(3)+" "+ma
      (4)) //display the mask
23 //a) first address
24 x=strsplit(a);
25 y=strsplit(mask);
26 for i= 1:32
27     fa0(i)=bitand(strtod(x(i)),strtod(y(i))); //
      perform 'and' of address and mask to get 1st
      address
28 end
29 fa1=string(fa0(1:8));
30 fa2=string(fa0(9:16));
31 fa3=string(fa0(17:24));
32 fa4=string(fa0(25:32));
33 printf("\na) The first address is "); //display
      result
34 printf("%s",fa1);
35 printf(" ");
36 printf("%s",fa2);
37 printf(" ");
38 printf("%s",fa3);
39 printf(" ");

```

```

40 printf("%s",fa4);
41
42 //b) last address
43 for i=1:32
44     cp0(i)=bitcmp(strtod(y(i)),1); //find
        complement of the mask
45 end
46 for i=1:32
47     la0(i)=bitor(strtod(x(i)),cp0(i)); //perform 'or
        ' of address and complement of the mask
48 end
49 cp1=string(cp0(1:8));
50 cp2=string(cp0(9:16));
51 cp3=string(cp0(17:24));
52 cp4=string(cp0(25:32));
53 printf("\n\nb) The complement of the mask is ");
54 printf("%s",cp1);
55 printf(" ");
56 printf("%s",cp2);
57 printf(" ");
58 printf("%s",cp3);
59 printf(" ");
60 printf("%s",cp4);
61
62 la1=string(la0(1:8));
63 la2=string(la0(9:16));
64 la3=string(la0(17:24));
65 la4=string(la0(25:32));
66 printf("\n\n The last address is "); //display the
    result
67 printf("%s",la1);
68 printf(" ");
69 printf("%s",la2);
70 printf(" ");
71 printf("%s",la3);
72 printf(" ");
73 printf("%s",la4);
74

```

```

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

```

---

Scilab code Exa 19.10 design sub blocks

```

1 clear all;
2 clc;
3 disp("-----Example 19.10-----")
4 // address :- 190.100.0.0/16 i.e 65,536 addresses
5 num_of_ISP_addresses=65536; //total number of
    addresses
6 printf('\n');
7 function [total]= addresses (num_of_customers ,
    num_of_addresses) //function to find total number
    of addresses allocated to a group
8     total=num_of_customers*num_of_addresses; //
    formula to calculate total number of
    addresses
9     bits=log2(num_of_customers);
10    n=32-bits;
11    printf("Number of bits needed to define each
    host = %d",bits); //display results
12    printf("\nThe prefix length = %d",n);
13    endfunction
14 //group 1
15 g1=addresses(64,256); //calling fuction
16 printf("\nThe total number of addresses allotted to

```

```

    Group 1 = %d\n\n",g1);
17 //group 2
18 g2=addresses(128,128); //calling function
19 printf("\nThe total number of addresses allotted to
    Group 2 = %d\n\n",g2);
20 //group 3
21 g3=addresses(128,64); //calling function
22 printf("\nThe total number of addresses allotted to
    Group 3 = %d\n\n",g3);
23 num_allocated=g1+g2+g3; //total number of addresses
    allocated
24 num_remaining_addresses=num_of_ISP_addresses -
    num_allocated; // formula to calculate number of
    remaining addresses
25 printf("\nThe total number of addresses granted to
    the ISP = %d",num_of_ISP_addresses);
26 printf("\n\nThe total number of addresses allocated
    by the ISP = %d",num_allocated);
27 printf("\n\nThe number of addresses remaining = %d",
    num_remaining_addresses); //display result

```

---

Scilab code Exa 19.11 find original address

```

1 clear all;
2 clc;
3 disp("-----Example 19.11-----")
4 //address :- 0:15::1:12:1213
5 a1=0;
6 a2=15;
7 a8=1213;
8 a7=12;
9 a6=1;
10 n1=strsplit(string(a1));
11 n2=strsplit(string(a2));
12 n8=strsplit(string(a8));

```

```

13 n7=strsplit(string(a7));
14 n6=strsplit(string(a6));
15 function [n]=org_address (num,nmatrix) //function to
    form the 2 bytes in hexadecimal
16     n="";
17     for i=1:4-length(string(num))
18         n=n+'0';
19     end
20     for i=1:length(string(num))
21         n=n+nmatrix(i);
22     end
23 endfunction
24 f1=org_address(a1,n1); //2 byte addresses
25 f2=org_address(a2,n2);
26 f8=org_address(a8,n8);
27 f7=org_address(a7,n7);
28 f6=org_address(a6,n6);
29 f3="0000"; // zeros
30 f4=f3;
31 f5=f3;
32 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

```
1 clear all;
2 clc;
3 disp("-----Example 20.1-----")
4 // 01000010 - first 8 bits of IP4 packet
5 p="01000010";
6 s=strsplit(p,4); // split into two
7 v=bin2dec(s(1)); // version
8 d=bin2dec(s(2)); // header length
9 bytes=d*4; // formula
10 if(((bytes >= 20 )&((v == 4) |(v == 6)))) //minimum
    number of bytes is 20 and version should be IP4
    or IP6
11     printf("The packet is accepted .");
12 else
13     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

```
1 clear all;
2 clc;
3 disp("-----Example 20.2-----")
4 HLEN="1000";
5 d=bin2dec(HLEN); //convert to decimal
6 bytes=d*4; // formula
7 base_header=20;
8 options=bytes-base_header; // formula
9 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

```
1 clear all;
2 clc;
3 disp("-----Example 20.3-----")
4 HLEN=5;
5 total_length="0028"; // 0x0028 in hexadecimal
6 total_bytes=hex2dec(total_length); // get length in
    decimal or total bytes
7 header_bytes = HLEN*4; // number of header bytes
8 data_bytes=total_bytes-header_bytes; // formula
9 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

```
1 clear all;
2 clc;
3 disp("-----Example 20.4-----")
4 // packet = Ox45000028000100000102
5 packet="45000028000100000102";
6 bytes=strsplit(packet,[2 4 6 8 10 12 14 16 18]); //
    split the packet into bytes
7 time_to_live=hex2dec(bytes(9));
8 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);
9 protocol_field=hex2dec(bytes(10));
10 select protocol_field // display the result
    according to the protocol
11 case 1
12     printf("The protocol field is the next byte i.e
    %s, which means that the upper-layer protocol
    is ICMP.",bytes(10));
13 case 2
14     printf("The protocol field is the next byte i.e
    %s, which means that the upper-layer protocol
    is IGMP.",bytes(10));
15 case 6
16     printf("The protocol field is the next byte i.e
    %s, which means that the upper-layer protocol
    is TCP.",bytes(10));
17 case 17
18     printf("The protocol field is the next byte i.e
    %s, which means that the upper-layer protocol
    is UDP.",bytes(10));
```

```

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

```

---

#### Scilab code Exa 20.5 M equals 0

```

1 clear all;
2 clc;
3 disp("-----Example 20.5-----")
4 M_bit = 0;
5 if(M_bit==0) // display the result according to the
               value of the M bit
6     printf("There are no more fragments; this
           fragment is the last one.");
7 else
8     printf("There are more fragments; this fragment
           is not the last one.");
9 end
10 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

```

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

```

```

6     printf("There are no more fragments; this
          fragment is the last one.");
7 else
8     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.");
9 end

```

---

#### Scilab code Exa 20.7 M and offset

```

1 clear all;
2 clc;
3 disp("-----Example 20.7-----")
4 M_bit = 1;
5 fragmentation_offset = 0;
6 if(M_bit==0) // display the result according to the
              value of the M bit
7     printf("There are no more fragments; this
          fragment is the last one.");
8 else
9     if(fragmentation_offset == 0) // display the
              result according to the value of the
              fragmentation offset
10        printf("The M bit is %d and the offset value
              is %d. Hence it is the first fragment.",
              M_bit, fragmentation_offset);
11    else
12        printf("It is not first or last fragment. It
              can be any fragment in between.");
13    end
14
15 end

```

---

**Scilab code Exa 20.8** First byte number

```
1 clear all;
2 clc;
3 disp("-----Example 20.8-----")
4 offset = 100;
5 first_byte=offset*8; // formula
6 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

```
1 clear all;
2 clc;
3 disp("-----Example 20.9-----")
4 offset = 100;
5 HLEN=5;
6 total_length_field=100; // 100 bytes
7 first_byte=offset*8; // formula
8 header_bytes=HLEN*4; // formula
9 data_bytes=total_length_field-header_bytes; //
   formula
10 last_byte=first_byte+data_bytes-1; // formula
11 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

```
1 clear all;
2 clc;
3 disp("-----Example 20.10-----")
4 source_address="10.12.14.5";
5 destination_address="12.6.7.9";
6 // convert all the fields to hexadecimal
7 a1=hex2dec("4500");
8 a2=28;
9 a3=1;
10 a4=0;
11 a5=hex2dec("0411");
12 a6=0;
13 a7=hex2dec("0A0C");
14 a8=hex2dec("0E05");
15 a9=hex2dec("0C06");
16 a10=hex2dec("0709");
17 d_sum=a1+a2+a3+a4+a5+a6+a7+a8+a9+a10; // find the
    sum of all fields
18 Sum=dec2hex(d_sum);
19 c=bitcmp(d_sum,16); // complement sum
20 Checksum=dec2hex(c);
21 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
22 // display the figure
23 clf();
24 xname("-----Example 20.10-----")
    ;
25 xrects([.28 .33 .38 .44;.8 .8 .8 .8;.05 .05 .06
    .16;.06 .06 .06 .06]);
26 xrects([.28 .44 .52;.74 .74 .74 ;.16 .08 .08 ;.06
    .06 .06 ]);
```

```

27 xrects([.28 .36 .44 ;.68 .68 .68 ;.08 .08 .16 ;.06
        .06 .06 ]);
28 xrect(.28,.62,.32,.06);
29 xrect(.28,.56,.32,.06);
30 for i=0:9
31     xarrows([.38 .42],[.47-(i/25) .47-(i/25)],.3);
32 end
33 xarrows([.38 .42],[.04 .04],.3);
34 xarrows([.38 .42],[.002 .002],.3);
35 xpoly([.44 .52],[.06 .06]);
36 xset("font size",3);
37 xstring(.34,.033,"Sum");
38 xstring(.3,.00001,"Checksum");
39 xstring(.3,.46,"4,5 and 0");
40 xstring(.35,.42,"28");
41 xstring(.36,.38,"1");
42 xstring(.31,.34,"0 and 0");
43 xstring(.31,.3,"4 and 17");
44 xstring(.44,.46,"4 5 0 0");
45 xstring(.44,.42,"0 0 1 C");
46 xstring(.44,.38,"0 0 0 1");
47 xstring(.44,.34,"0 0 0 0");
48 xstring(.44,.3,"0 4 1 1");
49 xstring(.44,.26,"0 0 0 0");
50 xstring(.44,.22,"0 A 0 C");
51 xstring(.44,.18,"0 E 0 5");
52 xstring(.44,.14,"0 C 0 6");
53 xstring(.44,.1,"0 7 0 9");
54 xstring(.44,.031,"7 4 4 E");
55 xstring(.44,.000001,"8 B B 1");
56 xstring(.36,.26,"0");
57 xstring(.33,.22,"10.12");
58 xstring(.34,.18,"14.5");
59 xstring(.34,.14,"12.6");
60 xstring(.35,.1,"7.9");
61 xstring(.4,.565,source_address);
62 xstring(.41,.51,destination_address);
63 xstring(.3,.75," 4           5           0

```

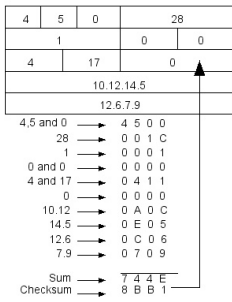


Figure 20.1: IPv4 header checksum

```

64 xstring(.35 ,.69,"1
    28");
65 xstring(.31,.63,"4
    0");
66 xpoly([.51 .55],[.02 0.02]);
67 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

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

```

13 Target_pr_addr=" 130.23.43.25"; // target protocol
    address=IP address
14 // display ARP Request packet
15 printf("ARP Request Packet\n");
16 printf("
    -----
\n");
17 printf("          | -----0x%s----- |
    -----0x%s----- |\n",Hardware_type ,
    Protocol_type);
18 printf("          | ---0x%s----- | ---0x%s---- |
    -----0x%s----- |\n",Hardware_length ,
    Protocol_length,Operation);
19 printf("          |
    -----0x%s----- |\n",
    Sender_hw_addr);
20 printf("          |
    ----- %s ----- |\n",
    Sender_pr_addr);
21 printf("          |
    -----0x%s----- |\n",
    Target_hw_addr);
22 printf("          |
    ----- %s ----- |\n",
    Target_pr_addr);
23
24 // ARP Reply
25 Hardware_type=" 0001"; // Ethetnet = 1 , in
    hexadecimal
26 Protocol_type=" 0800"; // IPv4 = 0800 in hexadecimal
27 Hardware_length=" 06"; // for Ethernet in hexadecimal
28 Protocol_length=" 04"; // for IPv4 in hexadecimal
29 Operation=" 0002"; // reply=1 , in hexadecimal
30 Sender_hw_addr=" A46EF45983AB" // sender hardware
    address = A4:6E:F4:59:83:AB in hexadecimal
31 Sender_pr_addr=" 130.23.43.25"; // sender protocol
    address=IP address
32 Target_hw_addr=" B23455102210"; // target hardware

```

```

        address = B2:34:55:10:22:10 in hexadecimal
33 Target_pr_addr="130.23.43.20"; // target protocol
    address=IP address
34 // display ARP Reply Packet
35 printf("\nARP Reply Packet\n");
36 printf("
-----
\n");
37 printf("          |-----0x%s-----|
-----0x%s-----|\n",Hardware_type,
    Protocol_type);
38 printf("          |---0x%s-----|---0x%s----|
-----0x%s-----|\n",Hardware_length,
    Protocol_length,Operation);
39 printf("          |
-----0x%s-----|\n",
    Sender_hw_addr);
40 printf("          |
-----%s-----|\n",
    Sender_pr_addr);
41 printf("          |
-----0x%s-----|\n",
    Target_hw_addr);
42 printf("          |
-----%s-----|\n",
    Target_pr_addr);
-----

```

### Scilab code Exa 21.2 Echo request message checksum

```

1 clear all;
2 clc;
3 disp("-----Example 21.2-----")
4 identifier=1;
5 sequence_number=9;
6 // 8 & 0

```

```

7 word1a=dec2bin(8,8);
8 word1b=dec2bin(0,8);
9 // 0
10 word2a=dec2bin(0,8);
11 word2b=dec2bin(0,8);
12 // 1
13 word3a=dec2bin(0,8);
14 word3b=dec2bin(identifier,8);
15 // 9
16 word4a=dec2bin(0,8);
17 word4b=dec2bin(sequence_number,8);
18 // TEST
19 // T & E
20 word5a=dec2bin(ascii('T'),8);
21 word5b=dec2bin(ascii('E'),8);
22 // S & T
23 word6a=dec2bin(ascii('S'),8);
24 word6b=dec2bin(ascii('T'),8);
25
26 sum_dec=bin2dec(word1a+word1b)+0+identifier+
    sequence_number+bin2dec(word5a+word5b)+bin2dec(
    word6a+word6b);
27 Sum=dec2bin(sum_dec,16); // sum
28 sum_bytes=strsplit(Sum,8);
29 cmp=bitcmp(sum_dec,16);
30 Checksum=dec2bin(cmp,16); // checksum
31 Checksum_bytes=strsplit(Checksum,8);
32
33 // display the result
34 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));
35 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

```
1 clear all;
2 clc;
3 disp("-----Example 21.3-----")
4 // display the example
5 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% packet loss , time 10103ms\n
rttminJavg/max = 1.899/1.955/2.041 ms", "%");
```

```

6 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 more 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

```

1 clear all;
2 clc;
3 disp("-----Example 21.4-----")
4 // display the example
5 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");
6 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");
7 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");
8 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");
9 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");
10 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

```

1 clear all;
2 clc;
3 disp("-----Example 21.5-----")
4 // display the example
5 printf("In this example, we trace a longer route,
the route to xerox.com.\n\n");
6 printf("$ traceroute xerox.com\n\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");
7 printf(" ... ..
... ..\n");
8 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\n
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\n
n18 alpha.Xerox.COM (13.1.64.93) 11.172 ms
11.048 ms 10.922ms\n\n");
9 printf("There are 17 hops between source and
destination. Some round-trip times look unusual.\n
nIt could be that a router was too busy to
process the packet immediately.")

```

---

### Scilab code Exa 21.6 report messages sequence

```

1 clear all;
2 clc;
3 disp("-----Example 21.6-----")
4 // display the sequence of report messages.
5 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

```

1 clear all;
2 clc;
3 disp("-----Example 21.7-----")
4 // multicast IP address 230.43.14.7
5 multicast_IP_address=dec2bin(230,5)+dec2bin(43,7)+
    dec2bin(14,7)+dec2bin(7,7);
6 s=strsplit(multicast_IP_address,length(
    multicast_IP_address)-23);
7 b=strsplit(s(2),[9 16]);
8 starting_Ethernet_addr = "01:00:5E"; // 01:00:5E
    :00:00:00
9 Ethernet_multicast_addr=starting_Ethernet_addr;
10
11 function[Ethernet_multicast_addr] = ethernet_address
    (b) // function to form Ethernet multicast
    physical address
12     for i=1:3
13         d=bin2dec(b(i));
14         h(i)=dec2hex(d); // rightmost 23 bits of
            the IP address in hexadecimal
15

```

```

16     end
17
18     hs=strsplit(h(1));
19     if(hex2dec(hs(1)) >= 8) //subtract 8 from the
        leftmost digit if it is greater than or
        equal to 8
20         hs(1)=dec2hex(hex2dec(hs(1))-8);
21     end
22     h(1)=hs(2)+hs(3);
23     for i=1:6 // add these hexadecimal digits to
        the starting Ethernet multicast address,
        which is 01:00:5E:00:00:00
24         if(modulo(i,2) == 0)
25             if(length(h(i/2))==2)
26                 Ethernet_multicast_addr=
                    Ethernet_multicast_addr+h(i/2);
27             else
28                 Ethernet_multicast_addr=
                    Ethernet_multicast_addr+'0'+h(i
                    /2);
29             end
30
31         else
32             Ethernet_multicast_addr=
                Ethernet_multicast_addr+":";
33         end
34     end
35 endfunction
36
37 Ethernet_multicast_addr=ethernet_address(b);
38 printf("The Ethernet multicast physical address is
    %s.",Ethernet_multicast_addr); // display result

```

---

Scilab code Exa 21.8 Ethernet multicast address

```

1 clear all;
2 clc;
3 disp("-----Example 21.8-----")
4 // multicast IP address 238.212.24.9
5 multicast_IP_address=dec2bin(238,5)+dec2bin(212,7)+
    dec2bin(24,7)+dec2bin(9,7);
6 s=strsplit(multicast_IP_address,length(
    multicast_IP_address)-23);
7 b=strsplit(s(2),[9 16]);
8 starting_Ethernet_addr = "01:00:5E"; // 01:00:5E
    :00:00:00
9 Ethernet_multicast_addr=starting_Ethernet_addr;
10
11 function[Ethernet_multicast_addr] = ethernet_address
    (b) // function to form Ethernet multicast
    physical address
12     for i=1:3
13         d=bin2dec(b(i));
14         h(i)=dec2hex(d); // rightmost 23 bits of
            the IP address in hexadecimal
15
16     end
17
18     hs=strsplit(h(1));
19     if(hex2dec(hs(1)) >= 8) //subtract 8 from the
        leftmost digit if it is greater than or
        equal to 8
20         hs(1)=dec2hex(hex2dec(hs(1))-8);
21     end
22     h(1)=hs(1)+hs(2);
23     for i=1:6 // add these hexadecimal digits to
        the starting Ethernet multicast address,
        which is 01:00:5E:00:00:00
24         if(modulo(i,2) == 0)
25             if(length(h(i/2))==2)
26                 Ethernet_multicast_addr=
                    Ethernet_multicast_addr+h(i/2);
27             else

```

```

28             Ethernet_multicast_addr=
                Ethernet_multicast_addr+'0'+h(i
                /2);
29         end
30
31     else
32         Ethernet_multicast_addr=
            Ethernet_multicast_addr+": ";
33     end
34 end
35 endfunction
36
37 Ethernet_multicast_addr=ethernet_address(b);
38 printf("The Ethernet multicast physical address is
    %s.",Ethernet_multicast_addr); // display result

```

---

#### Scilab code Exa 21.9 netstat nra

```

1 clear all;
2 clc;
3 disp("-----Example 21.9-----")
4 // display the example
5 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");
6 printf("$ netstat -nra\nKernel IP routing table\n
    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\n
    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");
7 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

```
1 clear all;
2 clc;
3 disp("-----Example 22.1-----")
4 // network addresses
5 network_address1=" 180.70.65.192";
6 network_address2=" 180.70.65.128";
7 network_address3=" 201.4.22.0";
8 network_address4=" 201.4.16.0";
9 network_address5=" Any" // Rest of the internet
10 // masks
11 mask1="/26";
12 mask2="/25";
13 mask3="/24";
14 mask4="/22";
15 mask5="Any"; // Rest of the internet
16 // interfaces
17 interface1=" m2";
18 interface2=" m0";
19 interface3=" m3";
```

```

20 interface4="      m1";
21 interface5="      m2"; // Rest of the internet
22
23 router_address="180.70.65.200"; // Router R1
24 // next hop addresses
25 next_hop1="      -";
26 next_hop2="      -";
27 next_hop3="      -";
28 next_hop4="      -";
29 next_hop5=router_address; // For rest of the
    universe
30
31 // define matrices for the 4 columns of the routing
    table
32 mask = [mask1; mask2; mask3; mask4; mask5];
33 network_address=[network_address1; network_address2;
    network_address3; network_address4;
    network_address5];
34 interface=[interface1; interface2; interface3 ;
    interface4; interface5];
35 next_hop=[next_hop1;next_hop2;next_hop3;next_hop4;
    next_hop5];
36
37 // define a matrix for the whole routing table
38 routing_table=[mask network_address next_hop
    interface];
39
40 // displaying the routing table
41 printf("\n      ROUTING TABLE FOR ROUTER R1\n");
42 printf("\n!Mask|Network address|  Next hop  |
    Interface!\n"); // display the headings
43 disp(routing_table); // display the routing table
    matrix

```

---

Scilab code Exa 22.2 Forwarding process 1

```

1 clear all;
2 clc;
3 disp("-----Example 22.2-----")
4 // network addresses
5 network_addresses=["180.70.65.192", "180.70.65.128", "
    201.4.22.0", "201.4.16.0", "Any"];
6 // masks
7 mask=[26,25,24,22];
8
9 // interfaces
10 interface=["m2" "m0" "m3" "m1" "m2"];
11
12 //destination address = 180.70.65.140
13 byte1=180;
14 byte2=70;
15 byte3=65;
16 byte4=140;
17 // convert it to binary
18 b1=dec2bin(byte1,8);
19 b2=dec2bin(byte2,8);
20 b3=dec2bin(byte3,8);
21 b4=dec2bin(byte4,8);
22 destination_address=b1+b2+b3+b4; // destination
    address in binary
23 network_address="";
24
25 for i=1:4 // applying the each of the masks to the
    destination address
26     na="";
27     printf("\n\n%d) The mask %d is applied to the
        destination address.",i,mask(i));
28     nz=32-mask(i); // number of zeros after
        applying the mask
29     s=strsplit(destination_address);
30     for k=33-nz:32
31         s(k)='0'; // replacing last 'nz' bits
            with zeros
32     end

```



```

33     for k=1:32
34         na=na+s(k); // new address in binary
35     end
36     bytes=strsplit(na,[8 16 24]); // split it
           into bytes
37 // convert them to binary
38     d1=bin2dec(bytes(1));
39     d2=bin2dec(bytes(2));
40     d3=bin2dec(bytes(3));
41     d4=bin2dec(bytes(4));
42     network_address=string(d1)+"."+string(d2)+".
           "+string(d3)+"."+string(d4); // network
           address formed in decimal notation
43
44     if(network_address==network_addresses(i)) //
           check if it matches with any given network
           addresses and display appropriate results
45         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));
46         break;
47     else
48         printf("\nThe result is %s, which does not
           match the corresponding network address
           %s.",network_address,network_addresses(i)
           );
49     end
50 end

```

---

Scilab code Exa 22.3 Forwarding process 2

```

1 clear all;
2 clc;
3 disp("-----Example 22.3-----")
4 // network addresses
5 network_addresses=["180.70.65.192","180.70.65.128","
    201.4.22.0","201.4.16.0","Any"];
6 // masks
7 mask=[26,25,24,22];
8
9 // interfaces
10 interface=["m2" "m0" "m3" "m1" "m2"];
11
12 //destination address = 201.4.22.35
13 byte1=201;
14 byte2=4;
15 byte3=22;
16 byte4=35;
17 // convert it to binary
18 b1=dec2bin(byte1,8);
19 b2=dec2bin(byte2,8);
20 b3=dec2bin(byte3,8);
21 b4=dec2bin(byte4,8);
22 destination_address=b1+b2+b3+b4; // binary form
23 network_address="";
24
25 for i=1:4 // applying the each of the masks to the
    destination address
26     na="";
27     printf("\n\n%d) The mask /%d is applied to the
        destination address.",i,mask(i));
28     nz=32-mask(i); // number of zeros after
        applying the mask
29     s=strsplit(destination_address);
30     for k=33-nz:32
31         s(k)='0'; // replacing last 'nz' bits
            with zeros
32     end
33     for k=1:32

```

```

34         na=na+s(k); // new address in binary
35     end
36     bytes=strsplit(na,[8 16 24]); // split it
        into bytes
37 // convert it into decimal
38     d1=bin2dec(bytes(1));
39     d2=bin2dec(bytes(2));
40     d3=bin2dec(bytes(3));
41     d4=bin2dec(bytes(4));
42     network_address=string(d1)+"."+string(d2)+".
        "+string(d3)+"."+string(d4); // network
        address formed in decimal notation
43
44     if(network_address==network_addresses(i)) //
        check if it matches with any given network
        addresses and display appropriate results
45         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));
46         break;
47     else
48         printf("\nThe result is %s, which does not
            match the corresponding network address
            %s.",network_address,network_addresses(i)
            );
49     end
50 end

```

---

#### Scilab code Exa 22.4 Forwarding process 3

```

1 clear all;
2 clc;

```

```

3 disp("-----Example 22.4-----")
4 // network addresses
5 network_addresses=["180.70.65.192","180.70.65.128","
    201.4.22.0","201.4.16.0","Any"];
6 // masks
7 mask=[26,25,24,22];
8
9 // interfaces
10 interface=["m2" "m0" "m3" "m1" "m2"];
11 //destination address = 18.24.32.78
12 byte1=18;
13 byte2=24;
14 byte3=32;
15 byte4=78;
16 // convert it to binary
17 b1=dec2bin(byte1,8);
18 b2=dec2bin(byte2,8);
19 b3=dec2bin(byte3,8);
20 b4=dec2bin(byte4,8);
21 destination_address=b1+b2+b3+b4;
22 network_address="";
23
24 nexthop_address="180.70.65.200"; // Router R1
    address
25
26 for i=1:4 // applying the each of the masks to the
    destination address
27     na="";
28     printf("\n\n%d) The mask /%d is applied to the
        destination address.",i,mask(i));
29     nz=32-mask(i); // number of zeros after
        applying the mask
30     s=strsplit(destination_address);
31     for k=33-nz:32
32         s(k)='0'; // replacing last 'nz' bits
            with zeros
33     end
34     for k=1:32

```

```

35         na=na+s(k); // new address in binary
36     end
37     bytes=strsplit(na,[8 16 24]); // split the
        new address into bytes
38 // convert them to decimal
39     d1=bin2dec(bytes(1));
40     d2=bin2dec(bytes(2));
41     d3=bin2dec(bytes(3));
42     d4=bin2dec(bytes(4));
43     network_address=string(d1)+"."+string(d2)+".
        "+string(d3)+"."+string(d4); // final
        network address in decimal notation
44
45     if(network_address==network_addresses(i)) //
        check if it matches with any given network
        addresses and display appropriate results
46         printf("\nThe result is %s, which matches
            the corresponding network address %s .\n
            The destination address of the packet
            and the interface number %s are passed to
            ARP.",network_address,network_addresses(
            i),interface(i));
47         break;
48     else
49         printf("\nThe result is %s, which does not
            match the corresponding network address
            %s.",network_address,network_addresses(i)
            );
50     end
51 end
52
53 if(i==4) // if it doesnt match any of the 4 given
        network addresses
54     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))
55 end

```

---

### Scilab code Exa 22.5 Hierarchical routing

```

1 clear all;
2 clc;
3 disp("-----Example 22.5-----")
4 total_addresses=16384; // A regional ISP is granted
   with 16,384 addresses
5 // starting address = 120.14.64.0
6 sa1=120;
7 sa2=14;
8 sa3=64;
9 sa4=0;
10 bin_sa=dec2bin(sa1,8)+dec2bin(sa2,8)+dec2bin(sa3,8)+
   dec2bin(sa4,8); // starting address in binary
11 la=dec2bin(bin2dec(bin_sa)+total_addresses-1,32); //
   last address in binary
12 a=strsplit(la,[8 16 24 ]) //separate the bytes
13 a3=bin2dec(a(1)); // convert binary numbers
   to decimal numbers
14 a2=bin2dec(a(2));
15 a1=bin2dec(a(3));
16 a0=bin2dec(a(4));
17 last_address=string(a3)+"."+string(a2)+"."+string(a1
   )+". "+string(a0); // last address in decimal
   notation
18 main_mask=18;
19 main_subblocks=4;
20 msb_addresses=total_addresses/main_subblocks; // The
   regional ISP divides this block into four
   subblocks , each with 4096 addresses.
21 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
22
23 msb1_subblocks=8; // The first local ISP has divided
    its assigned subblock into 8 smaller blocks and
    assigned each to a small ISP
24 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.
25 household_addresses=4; // a household has four
    addresses
26 household_mask=32-log2(household_addresses); //
    formula
27 num_households=msb_addresses/(household_addresses*
    msb1_subblocks); // Each small ISP provides
    services to 128 households (H001 to H128)
28 msb1_sa=string(sa1)+"."+string(sa2)+"."+string(sa3)+
    "."+string(sa4); // starting address in decimal
    notation
29 la1=dec2bin(bin2dec(bin_sa)+msb_addresses-1,32); //
    last address in binary
30 a=strsplit(la1,[8 16 24 ]) //separate the bytes
31 a3=bin2dec(a(1)); // convert binary numbers
    to decimal numbers
32 a2=bin2dec(a(2));
33 a1=bin2dec(a(3));
34 a0=bin2dec(a(4));
35 last_address1=string(a3)+"."+string(a2)+"."+string(
    a1)+"."+string(a0); // last address in decimal
    notation
36 H001_la=dec2bin(bin2dec(bin_sa)+household_addresses
    -1,32); // last address in binary
37 a=strsplit(H001_la,[8 16 24 ]) //separate the bytes
38 a3=bin2dec(a(1)); // convert binary numbers
    to decimal numbers
39 a2=bin2dec(a(2));
40 a1=bin2dec(a(3));
41 a0=bin2dec(a(4));

```

```

42 last_address_H001=string(a3)+"."+string(a2)+"."+
    string(a1)+"."+string(a0); // last address in
    decimal notation
43
44
45 msb2_subblocks=4; //The second local ISP divides its
    block into 4 blocks and assigns the addresses to
    four large organizations (LOrg01 to LOrg04).
46 Lorg_addresses=msb_addresses/msb2_subblocks; //
    number of addresses possessed by each large
    organization
47 mssb2_mask=msb_mask+log2(msb2_subblocks); // mask of
    the large organization addresses
48 sas2=dec2bin(bin2dec(bin_sa)+2*(msb_addresses),32);
    // starting address in binary
49 a=strsplit(sas2,[8 16 24 ]) //separate the bytes
50 a3=bin2dec(a(1)); // convert binary numbers
    to decimal numbers
51 a2=bin2dec(a(2));
52 a1=bin2dec(a(3));
53 a0=bin2dec(a(4));
54 start_address2=string(a3)+"."+string(a2)+"."+string(
    a1)+"."+string(a0); // starting address in
    decimal notation
55
56 msb3_subblocks=16; //The third local ISP divides its
    block into 16 blocks and assigns the addresses
    to 15 small organizations (SOrg01 to SOrg16).
57 Sorg_addresses=msb_addresses/msb3_subblocks; //
    number of addresses possessed by each small
    organization
58 mssb3_mask=msb_mask+log2(msb3_subblocks); // mask of
    the small organization addresses
59 sas3=dec2bin(bin2dec(bin_sa)+3*(msb_addresses),32);
    // starting address in binary
60 a=strsplit(sas3,[8 16 24 ]) //separate the bytes
61 a3=bin2dec(a(1)); // convert binary numbers
    to decimal numbers

```



```

62 a2=bin2dec(a(2));
63 a1=bin2dec(a(3));
64 a0=bin2dec(a(4));
65 start_address3=string(a3)+"."+string(a2)+"."+string(
    a1)+"."+string(a0); // starting address in
    decimal notation
66
67 // display the result
68
69 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

```

1 clear all;
2 clc;
3 disp("-----Example 22.6-----")
4 // display the example
5 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");
6 // output of $netstat -rn command
7 printf("\n$ 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
");
8 // explain the different columns
9 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 actually 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.");
10 //output of $ifconfig eth0 command
11 printf("\n\nMore information about the IP address
and physical address of the server can be found
by using the ifconfig command on the given
interface (eth0).\n$ 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

```
1 clear all;
2 clc;
3 disp("-----Example 23.1-----")
4 // display the example
5 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$grep ftp /etc/services\n
   nftp      21/tcp\nnftp      21/udp");
6 printf("\n\nSNMP uses two port numbers (161 and 162)
   , each for a different purpose.\n\n$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

```
1 clear all;
2 clc;
3 disp("-----Example 23.2-----")
4 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

```
1 clear all;
2 clc;
3 disp("-----Example 23.3-----")
4 first_byte= 10001;
5 num_segments=5;
6 total_bytes = 5000;
7 segment=1000; // each segment carries 1000 bytes
8 // compute the sequence numbers of each segment
9 segment1_sequence_num = first_byte;
10 segment2_sequence_num = segment1_sequence_num+
    segment;
11 segment3_sequence_num = segment2_sequence_num+
    segment;
12 segment4_sequence_num = segment3_sequence_num+
    segment;
13 segment5_sequence_num = segment4_sequence_num+
    segment;
14 // find the range of each segment
```

```

15 range1=segment1_sequence_num+segment-1;
16 range2=range1+segment;
17 range3=range2+segment;
18 range4=range3+segment;
19 range5=range4+segment;
20 // display the result
21 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

```

1 clear all;
2 clc;
3 disp("-----Example 23.4-----")
4 buffer_size=5000; //bytes
5 recieved_unprocessed = 1000; // bytes
6 rwnd=buffer_size-recieved_unprocessed ; // formula
7 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

```
1 clear all;
2 clc;
3 disp("-----Example 23.5-----")
4 rwnd=3000; // bytes
5 cwnd=3500; // bytes
6 window_size=min(rwnd,cwnd); // formula
7 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

```
1 clear all;
2 clc;
3 disp("-----Example 23.6-----")
4 rwnd=9; // bytes
5 cwnd=20; // bytes
6 window_size=min(rwnd,cwnd); // formula
7 // display result
8 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

```
1 clear all;
2 clc;
3 disp("-----Example 26.1-----")
4 // client request :- Do enable the echo option
5 r_character1="IAC";
6 r_character2="DO";
7 r_character3="ECHO";
8 //server approval :- I will enable the echo option
9 a_character1="IAC";
10 a_character2="WILL";
11 a_character3="ECHO";
12 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.\n")
```



nTherefore, the client should request from the server the enabling of the option using DO.\n\nThe request consists of three characters: %s, %s and %s.\n\nThe server accepts the request and enables the option WILL.\n\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

```

1 clear all;
2 clc;
3 disp("-----Example 26.2-----")
4 // request :- Do enable terminal option
5 re_character1="IAC";
6 re_character2="DO";
7 re_character3="Terminal type";
8 //approval :- will enable the terminal option
9 a_character1="IAC";
10 a_character2="WILL";
11 a_character3="Terminal type";
12 //request :- Set the terminal type to "VT"
13 r_character1="IAC";
14 r_character2="SB";
15 r_character3="Terminal type";
16 r_character4="V";
17 r_character5="T";
18 r_character6="IAC";
19 r_character7="SE";
20 // display the example
21 printf("  Client

      Server\n");

```

```

22 printf("      |                               I will enable the terminal
      option                               |\n");
23 printf("      |-----|%s| - |%s| - |%s
      |----->|\n", a_character3, a_character2,
      a_character1);
24 printf("      |

      |\n");
25 printf("      |                               Do enable terminal option
      |\n");
26 printf("      |<-----|%s| - |%s| - |%s
      |-----|\n", re_character1, re_character2,
      re_character3);
27 printf("      |

      |\n");
28 printf("      |                               Set the terminal type to VT
      |\n");
29 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

```

1 clear all;
2 clc;
3 disp("-----Example 26.3-----")
4 // display the example
5 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.\n

```

```

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 .
");
6 // display the SMTP commands and responses
7 printf("\n\n$ telnet mail.adelphia.net25\nTrying
68.168.78.100...\nConnected to mail.adelphia.net
(68.168.78.100).");
8 printf("\n\n===== Connection
Establishment===== \n 220 mta13.
adelphia.net SMTP server ready Fri , 6 Aug 2004
...\nHELO mail.adelphia.net\n 250 mta13.adelphia.
net");
9 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");
10 printf("\n\nThis is a test message\nto show SMTP in
action ... ");
11 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

```

1 clear all;
2 clc;
3 disp("-----Example 26.4-----")
4 // display the example

```

```

5 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.");
6 // display the commands and responses
7 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.");
8 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");
9 printf("\n\n226 Directory send OK.\nftp>quit\n221
   Goodbye.");
10 printf("\n\n\n1. 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 another command to start another activity (and eventually open another data connection). In our example, the client sends a QUIT command. 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

```
1 clear all;
2 clc;
3 disp("-----Example 26.5-----")
4 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");
5 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");
6 printf("\n\nbin\n...\n...\n...");
7 printf("\n\nftp>close\n221 Goodbye\nftp>quit");
```

---

# Chapter 27

## WWW and HTTP

Scilab code Exa 27.1 Retrieving document using GET

```
1 clear all;
2 clc;
3 disp("-----Example 27.1-----")
4 // display the example
5 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.");
6 // figure
7 printf("\n\n Client
   Request (GET method)
   Server\n\n");
8 printf(" |
```

```

9     printf("      |\n");
10    printf("      |-----|\n");
10    printf("      |-----|GET /usr/bin/image1 HTTP/1.1
      |----->|\n");
11    printf("      |           | Accept: image/gif
      |           |\n");
12    printf("      |           | Accept: image/jpeg
      |           |\n");
13    printf("      |           |
      |-----| |\n");
14    printf("      |
      |\n");
15    printf("      |
      |-----|\n");
16    printf("      |           |HTTP/1.1 200 OK
      |           |\n");
17    printf("      |           | Date: Mon07-Jan-05 13:15:14
      GMT|           |\n");
18    printf("      |<-----|Server: Challenger
      |-----|\n");
19    printf("      |           |MIME-version: 1.0
      |           |\n");
20    printf("      |           |Content-length: 2048
      |           |\n");
21    printf("      |           |
      |           |\n");
22    printf("      |           |(Body of the document)
      |           |\n");
23    printf("      |           |
      |-----| |\n");
24    printf("      |           |Response
      |\n");

```

---

## Scilab code Exa 27.2 Send data using POST

```

1 clear all;
2 clc;
3 disp("-----Example 27.2-----")
4 // display the example
5 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.");
6 // figure
7 printf("\n\n Client                                Server\n
        n");
8 printf(" | Request (POST method)
9 printf(" |
        ----- |\n");
10 printf(" |-----|POST /cgi-bin/doc.pl HTTP/1.1
        |----->|\n");
11 printf(" | Accept: */*
        | |\n");
12 printf(" | Accept: image/gif
        | |\n");
13 printf(" | Accept: image/jpeg
        | |\n");
14 printf(" | Content-length: 50
        | |\n");
15 printf(" |
        | |\n");
16 printf(" | (Input information)
        | |\n");
17 printf(" |
        | |

```



```

18 printf("-----|                |\n");
    |\n");
19 printf("-----|                |\n");
20 printf("-----|                |HTTP/1.1 200 OK
    |                |\n");
21 printf("-----|                |Date: Mon,07-Jan-02 13:15:14
    GMT|                |\n");
22 printf("-----|Server: Challenger
    |-----|\n");
23 printf("-----|MIME-version: 1.0
    |                |\n");
24 printf("-----|Content-length: 2000
    |                |\n");
25 printf("-----|
    |                |\n");
26 printf("-----|(Body of the document)
    |                |\n");
27 printf("-----|
    |                |\n");
28 printf("-----|                |\n");
    Response                |\n");

```

---

### Scilab code Exa 27.3 Connect to server using TELNET

```

1 clear all;
2 clc;
3 disp("-----Example 27.3-----")
4 // display the example
5 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.\n\nThe server response is seven lines starting with the status line.\n\nThe blank line at the end terminates the server response. The file of 14,230 lines is received after the blank line.\n\nThe last line is the output by the client.\n\n");

```
6 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");
7 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

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

```

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

```

---

### Scilab code Exa 28.2 Define OCTETSTRING HI

```

1 clear all;
2 clc;
3 disp("-----Example 28.2-----")
4 tag="00000100"; // OCTET STRING tag
5 Length="00000010"; // 2 bytes
6 octet_string="HI";
7 H_value="01001000"; // 48
8 I_value="01001001"; // 49
9 H_value_dec=48; // value in decimal
10 I_value_dec=49; // value in decimal
11 tag_dec=bin2dec(tag); // tag's decimal value
12 Length_dec=bin2dec(Length); // length's decimal
      value or number of bytes
13 // display the format
14 printf("\n      0%d      0%d      0%d
      %d\n",tag_dec ,Length_dec ,H_value_dec ,
      I_value_dec);
15 printf("      | %s | %s | %s | %s |\n",tag,Length ,
      H_value ,I_value);
16 printf("      Tag      Length      Value
      Value\n");
17 printf("      (string)  (2 bytes)      (H)      (
      I)");

```

---

### Scilab code Exa 28.3 Define ObjectIdentifier

```

1 clear all;
2 clc;
3 disp("-----Example 28.3-----")
4 tag="00000110"; // OBJECT IDENTIFIER tag
5 Length="00000100"; // 4 bytes
6 ObjectIdentifier="1.3.6.1"; // (iso.org.dod.internet
   )
7 ot1=1;
8 ot2=3;
9 ot3=6;
10 ot4=1;
11 // convert the bytes to binary
12 byte1=dec2bin(ot1,8);
13 byte2=dec2bin(ot2,8);
14 byte3=dec2bin(ot3,8);
15 byte4=dec2bin(ot4,8);
16 tag_dec=bin2dec(tag); // tag's decimal value
17 Length_dec=bin2dec(Length); // length's decimal
   value or number of bytes
18 // display the format
19 printf("\n          0%d          0%d          0%d          0%d
          0%d          0%d          0%d\n",tag_dec ,
          Length_dec ,ot1,ot2,ot3,ot4);
20 printf("      | %s | %s | %s | %s | %s | %s|\n",tag ,
          Length,byte1,byte2,byte3,byte4);
21 printf("          Tag          Length          Value          Value
          Value          Value\n");
22 printf("          (ObjectId) (4 bytes)          (%d)          (%d)
          (%d)          (%d)\n",ot1,ot2,ot3,ot4);
23 printf("          |-----%s (iso.org
          .dod.internet)-----|",ObjectIdentifier);

```

---

#### Scilab code Exa 28.4 Define IPAddress

```

1 clear all;

```

```

2  clc;
3  disp("-----Example 28.4-----")
4  tag="01000000"; // IPAddress tag
5  Length="00000100"; // 4 bytes
6  IPAddress="131.21.14.8"; // value
7  tag_dec=bin2dec(tag); // tag's decimal value
8  tag_hex=dec2hex(tag_dec); // tag's hex value
9  Length_dec=bin2dec(Length); // length's decimal
   value or number of bytes
10 ip1=131;
11 ip2=21;
12 ip3=14;
13 ip4=8;
14 // convert the bytes to binary
15 byte1=dec2bin(ip1,8);
16 byte2=dec2bin(ip2,8);
17 byte3=dec2bin(ip3,8);
18 byte4=dec2bin(ip4,8);
19 // connvert bytes to hexadecimal
20 h1=dec2hex(ip1);
21 h2=dec2hex(ip2);
22 h3=dec2hex(ip3);
23 h4=dec2hex(ip4);
24 // display the format
25 printf("\n          %s          0%d          %s
          %s          0%s          0%s\n",tag_hex ,
          Length_dec ,h1 ,h2 ,h3 ,h4);
26 printf("      | %s | %s |%s | %s | %s | %s |\n",tag ,
          Length ,byte1 ,byte2 ,byte3 ,byte4);
27 printf("          Tag          Length          Value          Value
          Value          Value\n");
28 printf("          (IPAddress) (4 bytes) (0d) (0d)
          (0d)          (0d)\n",ip1 ,ip2 ,ip3 ,ip4);
29 printf("          |-----%s
          -----|\n", IPAddress);

```

---

### Scilab code Exa 28.5 SNMP message to retrieve UDP datagrams

```
1 clear all;
2 clc;
3 disp("-----Example 28.5-----")
4 //example explanation
5 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.\n\nThe value is of type 05
and length 00. The whole VarBind is a sequence of
length 0D (13).\n\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.");
```

```

6 // display the figure
7 clf();
8 xname("-----Example 28.5-----");
9 xfrect(.04,.5,.5,.3);
10 a=gce();
11 a.background=color('gray');
12 xfrect(.06,.33,.4,.1);
13 a=gce();
14 a.background=color('white');
15 xfrect(.08,.3,.3,.06);
16 a=gce();
17 a.background=color('gray');
18 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]);
19 xset("font size",2.5)
20 xstring(.72,.55,"Whole message a sequence of 52
        bytes");
21 xstring(0.01,.96,"02 01 03
        INTEGER, version");
22 xstring(.023,.88,"-- -- ... -- Header,
        sequence of length 12, not shown");
23 xstring(0.01,.81,"02 01 40 Two");
24 xstring(0.01,.78,"02 02 04 00 INTEGERS");
25 xstring(0.01,.7,"04 01 00 Three");
26 xstring(0.01,.67,"04 00 OCTET
        STRINGS");
27 xstring(0.01,.64,"04 00");
28 xstring(0.04,.51,"A0 1D");
29 xstring(0.042,.47,"02 04 00 01 06 11
        Three");
30 xstring(0.042,.44,"02 01 00
        INTEGERS");
31 xstring(0.042,.41,"02 01 00");
32 xstring(0.082,.27,"06 09 01 03 06 01 02 07 01 00
        VarBind");
33 xstring(0.082,.24,"05 00");
34 xstring(0.46,.29,"VarBind list");
35 xstring(0.54,.3,"length 29");

```



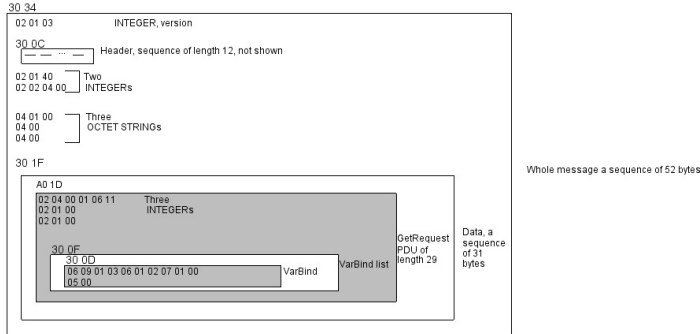


Figure 28.1: SNMP message to retrieve UDP datagrams

```

36 xstring(0.54, .33, "PDU of");
37 xstring(0.54, .36, "GetRequest");
38 xstring(0.63, .38, "Data, a");
39 xstring(0.63, .35, "sequence");
40 xstring(0.63, .32, "of 31");
41 xstring(0.63, .29, "bytes");
42 xpoly([.08 .1], [.84 .84]);
43 xpoly([.08 .1], [.78 .78]);
44 xpoly([.1 .1], [.78 .84]);
45 xpoly([.08 .1], [.72 .72]);
46 xpoly([.08 .1], [.64 .64]);
47 xpoly([.1 .1], [.64 .72]);
48 xset("font size", 3.5);
49 xlfont("Monospaced", 10, %t, %t);
50 xstring(0, 1, "30 34");
51 xstring(0.01, .9, "30 0C");
52 xstring(0.01, .57, "30 1F");
53 xstring(0.06, .33, "30 0F");
54 xstring(0.08, .3, "30 0D");

```

# Chapter 30

## Cryptography

Scilab code Exa 30.1 Monoalphabetic cipher

```
1 clear all;
2 clc;
3 disp("-----Example 30.1-----")
4 plaintext=['H' 'E' 'L' 'L' 'O'];
5 ciphertext=['K' 'H' 'O' 'O' 'R'];
6 L1=ciphertext(3); // 1st L's encryption
7 L2=ciphertext(4); // 2nd L's encryption
8 // display appropriate result
9 if(L1==L2)
10     printf("The cipher is probably monoalphabetic
            because both occurrences of Ls are encrypted
            as %ss.",L1);
11 else
12     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);
13 end
```

---

Scilab code Exa 30.2 Polyalphabetic cipher

```

1 clear all;
2 clc;
3 disp("-----Example 30.2-----")
4 plaintext=['H' 'E' 'L' 'L' 'O'];
5 ciphertext=['A' 'B' 'N' 'Z' 'F'];
6 L1=ciphertext(3); // 1st L's encryption
7 L2=ciphertext(4); // 2nd L's encryption
8 // display appropriate result
9 if(L1==L2)
10     printf("The cipher is probably monoalphabetic
            because both occurrences of Ls are encrypted
            as %ss.",L1);
11 else
12     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);
13 end

```

---

### Scilab code Exa 30.3 Shiftkey 15 encryption

```

1 clear all;
2 clc;
3 disp("-----Example 30.3-----")
4 message=['H' 'E' 'L' 'L' 'O'];
5 key=15; // shift down key
6 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'];
7 ciphertext="";
8
9 for k=1:5 // encrypt each character in the message
10     for i=1:26
11         if(message(k)==alphabet(i)) // find the
            index of the character in the alphabet

```

```

        array
12         break;
13     end
14 end
15 temp=i+15; // shift down by 15 towards end of
        the alphabet
16 if(temp > = 26)
17     a=modulo(temp,26); // wrap around thhe
        alphabet if its greater than 26
18 else
19     a=temp;
20 end
21 ciphertext=ciphertext+alphabet(a); // form the
        ciphertext
22 end
23 printf("The cipher text is %s.",ciphertext); //
        display the result

```

---

#### Scilab code Exa 30.4 Shiftkey 15 decryption

```

1 clear all;
2 clc;
3 disp("-----Example 30.4-----")
4 ciphertext=['W' 'T' 'A' 'A' 'D'];
5 key=15; // shift up key
6 plaintext="";
7 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'];
8
9 for k=1:5 // decrypt each character in the
        ciphertext
10     for i=1:26
11         if(ciphertext(k)==alphabet(i)) // find the
            index of the character in the alphabet

```

```

        array
12         break;
13     end
14 end
15 temp=i-15; // shift up by 15 towards the
        beginning of the alphabet
16 if(temp < = 0)
17     a=26+temp;
18     if(a>26)
19         a=modulo(a,26); // wrapping around the
                alphabet
20     end
21
22 else
23     a=temp;
24 end
25 plaintext=plaintext+alphabet(a); // form the
        plain text
26 end
27 printf("The plain text is %s.",plaintext); //
        display the result

```

---

### Scilab code Exa 30.5 Encryption of message

```

1 clear all;
2 clc;
3 disp("-----Example 30.5-----")
4 message=['H' 'E' 'L' 'L' 'O' ' ' 'M' 'Y' ' ' 'D' 'E'
        'A' 'R']; // HELLO MY DEAR
5 l=size(message,'c'); // length of message
6 ns="";
7 ciphertext="";
8 for i=1:l
9     if(message(i)==' ') // remove the spaces
10         continue;

```

```

11     else
12         ns=ns+message(i); // form message with no
            spaces
13     end
14
15 end
16
17 block=strsplit(ns,[4 8]); // split the message into
    blocks of 4
18 nz=4-length(block(3)); // number of 'Z's to be
    added to the last block
19
20 for i=1:nz
21     block(3)=block(3)+'Z'; // adding 'Z's to the
        last block
22 end
23 f=[];
24 for i=1:size(block,'r') // for each block
25     c=strsplit(block(i));
26     f(1)=c(2); // move the character at position 2
        to position 1
27     f(2)=c(4); // move the character at position 4
        to position 2
28     f(3)=c(1); // move the character at position 1
        to position 3
29     f(4)=c(3); // move the character at position 3
        to position 4
30     str=f(1)+f(2)+f(3)+f(4); // new block
31     ciphertext=ciphertext+str; // form the
        ciphertext
32 end
33 // display the result
34 printf("The 3 blocks are %s , %s and %s.",block(1),
        block(2),block(3));
35 printf("\n\nThe ciphertext is %s .",ciphertext)

```

---

### Scilab code Exa 30.6 Decryption of message

```
1 clear all;
2 clc;
3 disp("-----Example 30.6-----")
4 ciphertext="ELHLMDOYAZER";
5 block=strsplit(ciphertext,[4 8]); // split into
   blocks
6
7 f=[];
8 for i=1:size(block,'r') // for each block
9     c=strsplit(block(i));
10    f(2)=c(1); // move the character at position 1
        to position 2
11    f(4)=c(2); // move the character at position 2
        to position 4
12    f(1)=c(3); // move the character at position 3
        to position 1
13    f(3)=c(4); // move the character at position 4
        to position 3
14    str=f(1)+f(2)+f(3)+f(4);
15    block(i)=str; // new block
16 end
17 printf("The 3 blocks are %s , %s and %s.",block(1),
        block(2),block(3));
18 nz=0;
19 b3=strsplit(block(3));
20 for i=1:4
21     if(b3(i)=='Z');
22         nz=nz+1; // number of 'Z's in the last block
23     end
24
25 end
26
```

```

27     f=strsplit(block(3),size(b3,'r')-nz); // remove
        the 'Z's in the last block
28     block(3)=f(1); // new last block
29
30     text=block(1)+block(2)+block(3);
31     sp=strsplit(text,[5 7]);
32     plaintext=sp(1)+" "+sp(2)+" "+sp(3); // add the
        spaces
33     printf("\n\nThe message is %s.",plaintext) //
        display the result

```

---

#### Scilab code Exa 30.7 RSA plaintext 5

```

1  clear all;
2  clc;
3  disp("-----Example 30.7-----")
4  p=7;
5  q=11;
6  n=p*q; // formula
7  phi=(p-1)*(q-1); // formula
8  e=13;
9  d=1; // not actual 'd' value; it has to be computed
10 t=1;
11 P=5;
12 plaintext=P;
13 while t==1 do // compute d such that d*e = 1 mod n
14     if(modulo(e*d,phi)== 1)
15         t=0;
16     else
17         d=d+1;
18     end
19 end
20 // encryption by Alice
21 C=modulo(P^e,n); // formula
22 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
23
24 // decryption by Bob
25 P=modulo(C^d,n); // formula
26 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

```

1 clear all;
2 clc;
3 disp("-----Example 30.8-----")
4 p=397;
5 q=401;
6 n=p*q; // formula
7 phi=(p-1)*(q-1); // formula
8 e=343;
9 d=1; // not actual 'd' value; it has to be computed
10 message=['N' 'O']; // NO
11 t=1;
12 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'];
13 // Encryption process by Ted
14 while t==1 do // compute d such that d*e = 1 mod n
15     if(modulo(e*d,phi)== 1)
16         t=0;
17     else
18         d=d+1;
19     end
20 end

```

```

21 l=size(message,'c'); // length of the message
22 c=[];
23 for k=1:l // determine the code for each character
    in the message
24     for i=1:26
25         if(message(k)==alphabet(i))
26             c(k)=i-1; // compute the code
27             break;
28         end
29     end
30 end
31 plaintext=c(1)*100+c(2); // form the plaintext
32
33 //C=modulo((plaintext)^e,n) — formula to find the
    ciphertext
34 ciphertext=33677; // from calculation
35 printf("\nThe plaintext is %d and the ciphertext
    sent by Ted is %d.\n",plaintext,ciphertext); //
    display the result
36
37 // Decryption by Jennifer
38
39 //P=modulo((ciphertext)^d,n); — formula to find
    the plaintext
40 P=1314; // the plaintext that is computed
41 // separate the codes for each character
42 c(2)=modulo(P,100);
43 c(1)=floor(P/100);
44 d_message=""; // deciphered message
45 for k=1:l // find the corresponding letter for each
    code
46     for i=1:26
47         if(i==c(k)+1)
48             d_message=d_message+alphabet(i); //
                form the deciphered message
49         end
50
51     end

```

```

52 end
53 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

```

1 clear all;
2 clc;
3 disp("-----Example 30.9-----")
4 p
    =96130345313583504574191581280615427909309845594996215822583150879
    // 159 digit number
5 q
    =12060191957231446918276794204450896001555925054637033936061798321
    // 160 digit number
6 n=p*q; // formula
7 phi=(p-1)*(q-1); // formula
8 e=35535;
9 d="
    580083028600377639360936612896779175946690620896509621804228661111
    "; // compute d such that d*e = 1 mod n ( very
    huge value to compute)
10 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' ' '];
11 p_str="
    96130345313583504574191581280615427909309845594996215822583150879
    ";
12 q_str="
    12060191957231446918276794204450896001555925054637033936061798321
    ";
13 n_str=" 11593504
    I7396761496889250986461588752377145737545414477548552613761478854
    "; // 309 digits

```

```

14 phi_str="
    11593504173967614968892509864615887523771457375454144775485526137
    "; // 309 digits
15 // encoding by Alice
16 message=["T" "H" "I" "S" " " "I" "S" " " "A" " " "T"
    "E" "S" "T"]; //THIS IS A TEST
17 l=size(message,'c'); // length of the message
18 c=[];
19 plaintext="";
20 for k=1:l // determine the code for each character
    in the message
21     for i=1:27
22         if(message(k)==alphabet(i))
23             c(k)=string(i-1); // compute the code
24             if(length(c(k))==1)
25                 c(k)='0'+c(k);
26             end
27             break;
28         end
29     end
30 end
31 for i=1:l
32     plaintext=plaintext+c(i); // form the plaintext
        , code 26 is for space
33 end
34 P=plaintext;
35 C="
    47530912364622682720636555061054518094237179607049171652323924305
    "; // C= P^e -- ciphertext ( very huge value to
        compute)
36 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
    );
37 printf("The plaintext is %s and the ciphertext sent
    by Alice is\nC = %s\n\n",P,C);
38 // Decoding by Bob
39 P="1907081826081826002619041819"; // P=C^d --
    plaintext (very huge to compute)

```

```

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

```

---

### Scilab code Exa 30.10 Diffie Hellman method

```

1 clear all;
2 clc;
3 disp("-----Example 30.10-----")
4 g=7;
5 p=23;
6 printf("\nThe steps are as follows:\n\n");
7 x=3;
8 y=6;
9 R1=modulo(g^x,p); // formuula
10 R2=modulo(g^y,p); // formula
11 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);
12
13 K_Alice=modulo((R2)^x,p); // K calculated by Alice
14 K_Bob=modulo((R1)^y,p); // K calculated by Bob
15 K=modulo(g^(x*y),p); // The symmetric (shared) key
    in the Diffie-Hellman protocol
16 printf("5) Alice calculates the symmetric key K =%d
    .\n\n6) Bob calculates the symmetric key K = %d.\n
    n\n",K_Alice,K_Bob);
17
18 // check if the key values are equal and display
    appropriate result
19 if( K_Alice == K_Bob )
20     printf("The value of K is the same for both
        Alice and Bob. The symmetric key K = %d.",K);
21 else
22     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);
23 end

```

---

# Chapter 31

## Network Security

Scilab code Exa 31.1 Lossless compression method

```
1 clear all;
2 clc;
3 disp("-----Example 31.1-----")
4 // display the example
5 printf("We cannot. A lossless compression method
        creates a compressed message that is reversible.\n
        The compressed message can be uncompressed to
        get the original one.");
```

---

Scilab code Exa 31.2 Checksum method

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

---