SOFTWARE FOR TESTING THE APPLE LOCALTALK EXTENSION

Technical Report

BMW/89

submitted to the

Department of Electrical Engineering

University of Saskatchewan

by

BRIAN MICHAEL WIRTH

Saskatoon, Saskatchewan

October, 1989

Table of Contents

1. INTRODUCTION

This report describes a set of programs used to measure the throughput of an AppleTalk network under various levels of offered traffic. These programs were used to obtain the results reported in *Fiber Optic Extension of Apple LocalTalk Networks* [1].

In order to measure the throughput of AppleTalk, a number of computers generated messages at random intervals. The number of computers generating the messages together with the average delay between the messages gave the level of traffic offered to the network. The number of messages successfully transmitted over a given time interval gave the throughput. The messages were sent using the AppleTalk Link Access Protocol [2]. This is the lowest of several protocol layers on which all other protocols are based. The functions used to transmit the test messages are described in *Inside Macintosh* [3].

One computer on the test network was designated the supervisor and provided the user interface and overall control over the tests. The *Supervisor* program is presented in Chapter 2. The remaining computers were designated remotes and performed actions in response to commands received from the supervisor. The *Remote* program is presented in Chapter 3.

The results of each test were stored in a log file by the supervisor. This log file was subsequently stripped of unnecessary information by the *Strip* program described in

Chapter 4. The *Parse* program described in Chapter 5 formatted the stripped log file so that it can be easily read by a spreadsheet program.

Each chapter shows the structure of the program, descriptions of each function within the program, sample input or output files related to the program and the program listing. The structure of the program is represented by the number and the level of indent in the structure list. It can be used to determine which functions call which other functions within the program. For example, in the *Supervisor* program, function 1.2 (OpenProtocols) calls functions 1.2.1 (OpenMPP) and 1.2.2 (GetNodeID). Utility functions which are called from several places within the program hierarchy are numbered separately starting at 2.1.

2. SUPERVISOR PROGRAM

2.1. Structure

```
1.  main();
        1.1.  Initialization();
              2.1.  Reset();
        1.2.  OpenProtocols();
              1.2.1.  OpenMPP();
              1.2.2.  GetNodeID();
        1.3.  GetABRecords();
        1.4.  MainEvent();
              1.4.1.  ProcessSVPacket();
              2.2.  SendAbort();
              1.4.2.  SendQuit();
                    3.1.  SendSVPacketA();
              1.4.3.  GetDest();
                    1.4.3.1.  SendDest();
                          3.1.  SendSVPacketA();
              2.3.  FloodNet();
              1.4.5.  GoAutomatic();
                    3.1.  SendSVPacketA();
                    2.1.  Reset();
                    2.4.  SendReset();
                    2.5.  UpdateStats();
                    2.6.  SendFloodLambda();
                    1.4.5.1.  Pause4();
                    2.7.  SendPacketSize();
                    2.3.  FloodNet();
                    1.4.5.2.  WriteResults();
                    1.4.5.3.  WriteLocalResults();
                    2.2.  SendAbort();
              1.4.6.  PrintInfo();
                    2.5.  UpdateStats();
              1.4.7.  GetFloodLambda();
                    2.6.  SendFloodLambda();
              1.4.8.  GetRemoteNodes();
              1.4.9.  ShowRemoteNodes();
              1.4.10.  QueryNode();
                    3.1.  SendSVPacketA();
                    1.4.10.1.  PrintResults();
              2.1.  Reset();
              2.4.  SendReset();
```

```
        2.5.  UpdateStats();
        1.4.11.  GetPacketSize();
              2.7.  SendPacketSize();
        1.4.12.  GetFloodTime();
        1.4.13.  ShowHelp();
    1.5  CloseProtocols();
```

UTILITY FUNCTIONS:

```
    2.1.  Reset();
    2.2.  SendAbort();
          3.1.  SendSVPacketA();
    2.3.  FloodNet();
          3.1.1.  SendSVPacket();
    2.4.  SendReset();
          3.1.  SendSVPacketA();
    2.5.  UpdateStats();
    2.6.  SendFloodLambda();
          3.1.  SendSVPacketA();
    2.7.  SendPacketSize();
          3.1.  SendSVPacketA();
    2.8.  MyProtoHandler();
    2.9.  MySupervisorHandler();

    3.1.  SendSVPacketA();
          3.1.1.  SendSVPacket();
          3.1.2.  WaitForResults();
```

2.2. Function descriptions

1. main()	Supervisor mainline.
1.1. initialization()	Initialize global variables and print sign-on message.
1.2. OpenProtocols()	Open AppleTalk drivers and get network node ID. Install protocol handlers for test and supervisor protocols.
1.2.1. OpenMPP()	Open AppleTalk port. Make sure AppleTalk is enabled in the CHOOSER desk accessory.
1.2.2. GetNodeID()	Get and print the network node ID and network number. The network number should be zero for a standalone network.
1.3. GetABRecords()	Allocate dynamic 'AppleBus' record for the supervisor protocol and initialize the appropriate fields. AppleBus records hold

	information used when sending messages on AppleTalk.
1.5. CloseProtocols()	Close test and supervisor protocol handlers.
1.4. MainEvent()	Check for supervisor message received and keyboard commands and act on them appropriately.
1.4.1. ProcessSVPacket()	Print an error message since this function is only called when an unexpected supervisor message is received.
1.4.2. SendQuit()	Send a QUIT command to all remotes to halt the remote software.
1.4.3. GetDest()	Acquire destination addresses for all remote computers. The can be entered manually for each remote, or the remotes will be paired up based on their node IDs.
1.4.3.1. SendDest()	Inform a specific remote of its destination for test messages.
1.4.5. GoAutomatic()	Control a series of tests using parameters stored in a file.
1.4.5.1. Pause4()	Pause for 4 seconds. Used to give the remotes a chance to catch up if they fall behind.
1.4.5.2. WriteResults()	Write test results obtained from a remote to a file.
1.4.5.3. WriteLocalResults()	Write supervisor's 'local' results to a file.
1.4.6. PrintInfo()	Print the supervisor's node ID, network number, number of remotes identified and local statistics to the screen.
1.4.7. GetFloodLambda()	The user is prompted for the flood delay parameter (lambda). It is used by the exponential random number generator to generate the desired distribution.
1.4.8. GetRemoteNodes()	The supervisor compiles a list of all remotes present on the network.
1.4.9. ShowRemoteNodes()	The list of remote nodes present on the network is printed to the screen.
1.4.10. QueryNode()	Queries a remote computer for its current test statistics.

1.4.10.1. PrintResults()	Prints the results obtained from a remote query.
1.4.11. GetPacketSize()	Prompts the user for the length of the test messages used.
1.4.12. GetFloodTime()	Prompts the user for the duration of the next test. This is used only in manual command mode.
1.4.13. ShowHelp()	A list of all available manual commands is printed to the screen.
2.1. Reset()	Resets local statistics and error counters.
2.2. SendAbort()	Sends an ABORT command to all remotes on the network. This halts the remote software and may optionally re-boot the remote computer.
2.3. FloodNet()	A message is broadcast to all remotes to initiate a test. All remotes will then 'flood' the network with messages at a given rate.
2.4. SendReset()	A RESET command is issued to all remotes. This causes the remotes to reset their internal statistics and error counters.
2.5. UpdateStats()	Prints local statistics and number of errors to the screen.
2.6. SendFloodLambda()	Sends the flood delay parameter (lambda) to all remotes.
2.7. SendPacketSize()	Sends the length of the test messages to all remotes.
2.8. MyProtoHandler()	Protocol handler to receive the DATA portions of test messages. A 'messages received' counter is incremented and the message is discarded.
2.9. MySupervisorHandler()	Protocol handler to receive the DATA portions of supervisor messages. The message is saved in a buffer for further processing.
3.1. SendSVPacketA()	Sends a supervisor message to a specific remote and waits for an acknowledgement.
3.1.1. SendSVPacket()	Sends a message to a specific remote.
3.1.2. WaitForResults()	Wait up to 1/2 second for an acknowledgement message to be received.

2.3. Sample Parameter File

Each line of the parameter file contains three fields. The first field indicates the duration of the test (in seconds). The second field indicates the average delay (in seconds) between messages sent by each remote. The final field indicates the number of data bytes to be used in the test messages.

```
1800            0.020          6
1800            0.0          600
```

2.4. Sample Log File

```
Automatic Test Log - Jun 26, 1989

********** 15:24:13 *******
Reseting Remote Nodes
Sending Lambda = 0.8333333
(avg time = 0.02)
Sending Packet Size = 6
Test Time = 1800
Flooding Network
Querying Nodes...
Results from Node 51
     Destination Node ID = 67
     Packets Sent = 3633
     Tx Errors = 361
     Packets Received = 2
     SU Packet Overflows = 4
     Flood Lambda = 0.8333333
Results from Node 52
     Destination Node ID = 84
     Packets Sent = 4861
     Tx Errors = 45
     Packets Received = 36005
     SU Packet Overflows = 4
     Flood Lambda = 0.8333333
Results from Node 53
     Destination Node ID = 9
     Packets Sent = 0
     Tx Errors = 2248
     Packets Received = 36380
     SU Packet Overflows = 4
     Flood Lambda = 0.8333333
Results from Node 54
     Destination Node ID = 82
     Packets Sent = 4921
     Tx Errors = 46
     Packets Received = 1488
     SU Packet Overflows = 4
     Flood Lambda = 0.8333333
Results from Node 55
     Destination Node ID = 81
     Packets Sent = 4750
     Tx Errors = 78
     Packets Received = 36008
     SU Packet Overflows = 4
     Flood Lambda = 0.8333333
Results from Node 56
     Destination Node ID = 77
     Packets Sent = 4885
     Tx Errors = 31
     Packets Received = 35754
     SU Packet Overflows = 4
     Flood Lambda = 0.8333333
Results from Node 57
     Destination Node ID = 76
     Packets Sent = 36257
     Tx Errors = 41
     Packets Received = 8637
     SU Packet Overflows = 8
     Flood Lambda = 0.8333333
Results from Node 58
     Destination Node ID = 75
     Packets Sent = 4884
     Tx Errors = 35
     Packets Received = 36125
     SU Packet Overflows = 4
     Flood Lambda = 0.8333333
Results from Node 61
     Destination Node ID = 74
     Packets Sent = 4843
     Tx Errors = 56
     Packets Received = 34636
     SU Packet Overflows = 4
     Flood Lambda = 0.8333333
Results from Node 62
     Destination Node ID = 73
     Packets Sent = 4882
     Tx Errors = 41
     Packets Received = 35988
     SU Packet Overflows = 4
     Flood Lambda = 0.8333333
Results from Node 63
     Destination Node ID = 72
     Packets Sent = 35289
     Tx Errors = 34
     Packets Received = 35664
     SU Packet Overflows = 4
     Flood Lambda = 0.8333333
Results from Node 64
     Destination Node ID = 71
     Packets Sent = 4805
     Tx Errors = 44
     Packets Received = 36434
     SU Packet Overflows = 4
     Flood Lambda = 0.8333333
Results from Node 65
     Destination Node ID = 66
     Packets Sent = 4978
     Tx Errors = 25
     Packets Received = 36038
     SU Packet Overflows = 4
     Flood Lambda = 0.8333333
Results from Node 66
     Destination Node ID = 65
     Packets Sent = 36033
     Tx Errors = 58
     Packets Received = 4983
```

```
     SU Packet Overflows = 8
     Flood Lambda = 0.8333333
Results from Node 71
     Destination Node ID = 64
     Packets Sent = 36420
     Tx Errors = 28
     Packets Received = 4805
     SU Packet Overflows = 5
     Flood Lambda = 0.8333333
Results from Node 72
     Destination Node ID = 63
     Packets Sent = 35661
     Tx Errors = 40
     Packets Received = 35290
     SU Packet Overflows = 8
     Flood Lambda = 0.8333333
Results from Node 73
     Destination Node ID = 62
     Packets Sent = 35984
     Tx Errors = 41
     Packets Received = 4882
     SU Packet Overflows = 6
     Flood Lambda = 0.8333333
Results from Node 74
     Destination Node ID = 61
     Packets Sent = 34632
     Tx Errors = 99
     Packets Received = 4846
     SU Packet Overflows = 8
     Flood Lambda = 0.8333333
Results from Node 75
     Destination Node ID = 58
     Packets Sent = 36122
     Tx Errors = 52
     Packets Received = 4887
     SU Packet Overflows = 8
     Flood Lambda = 0.8333333
Results from Node 76
     Destination Node ID = 57
     Packets Sent = 8633
     Tx Errors = 51
     Packets Received = 36258
     SU Packet Overflows = 4
     Flood Lambda = 0.8333333
Results from Node 77
     Destination Node ID = 56
     Packets Sent = 35750
     Tx Errors = 53
     Packets Received = 4887
     SU Packet Overflows = 8
     Flood Lambda = 0.8333333
Results from Node 81
     Destination Node ID = 55
     Packets Sent = 36006
     Tx Errors = 81
     Packets Received = 4751
     SU Packet Overflows = 8
     Flood Lambda = 0.8333333
Results from Node 82
     Destination Node ID = 54
     Packets Sent = 1486
     Tx Errors = 19
     Packets Received = 4925
     SU Packet Overflows = 11
     Flood Lambda = 0.8333333
Results from Node 83
     Destination Node ID = 53
     Packets Sent = 36362
     Tx Errors = 79
     Packets Received = 0
     SU Packet Overflows = 8
     Flood Lambda = 0.8333333
Results from Node 84
     Destination Node ID = 52
     Packets Sent = 36001
     Tx Errors = 57
     Packets Received = 4862
     SU Packet Overflows = 8
     Flood Lambda = 0.8333333
Packets Sent: 0
Tx Errors: 0
Packets Received: 3634

********** 15:54:34 *******
Reseting Remote Nodes
Sending Lambda = 0
(avg time = 0)
Sending Packet Size = 600
Test Time = 1800
Flooding Network
Querying Nodes...
Results from Node 51
     Destination Node ID = 67
     Packets Sent = 1953
     Tx Errors = 2194
     Packets Received = 0
     SU Packet Overflows = 4
     Flood Lambda = 0
Results from Node 52
     Destination Node ID = 84
     Packets Sent = 3955
     Tx Errors = 1519
     Packets Received = 3642
     SU Packet Overflows = 8
     Flood Lambda = 0
Results from Node 53
     Destination Node ID = 9
     Packets Sent = 0
     Tx Errors = 3051
     Packets Received = 3405
     SU Packet Overflows = 8
     Flood Lambda = 0
Results from Node 54
     Destination Node ID = 82
     Packets Sent = 3953
     Tx Errors = 1466
```

```
     Packets Received = 3300
     SV Packet Overflows = 6
     Flood Lambda = 0
Results from Node 55
     Destination Node ID = 81
     Packets Sent = 1500
     Tx Errors = 805
     Packets Received = 78
     SV Packet Overflows = 4
     Flood Lambda = 0
Results from Node 56
     Destination Node ID = 77
     Packets Sent = 1507
     Tx Errors = 633
     Packets Received = 3806
     SV Packet Overflows = 4
     Flood Lambda = 0
Results from Node 57
     Destination Node ID = 76
     Packets Sent = 4082
     Tx Errors = 1478
     Packets Received = 3760
     SV Packet Overflows = 7
     Flood Lambda = 0
Results from Node 58
     Destination Node ID = 75
     Packets Sent = 4123
     Tx Errors = 1424
     Packets Received = 3829
     SV Packet Overflows = 8
     Flood Lambda = 0
Results from Node 61
     Destination Node ID = 74
     Packets Sent = 3850
     Tx Errors = 1542
     Packets Received = 3119
     SV Packet Overflows = 8
     Flood Lambda = 0
Results from Node 62
     Destination Node ID = 73
     Packets Sent = 705
     Tx Errors = 336
     Packets Received = 4041
     SV Packet Overflows = 4
     Flood Lambda = 0
Results from Node 63
     Destination Node ID = 72
     Packets Sent = 4096
     Tx Errors = 1474
     Packets Received = 3898
     SV Packet Overflows = 8
     Flood Lambda = 0
Results from Node 64
     Destination Node ID = 71
     Packets Sent = 3876
     Tx Errors = 1532
     Packets Received = 4289
     SV Packet Overflows = 8
     Flood Lambda = 0
Results from Node 65
     Destination Node ID = 66
     Packets Sent = 4390
     Tx Errors = 1392
     Packets Received = 774
     SV Packet Overflows = 8
     Flood Lambda = 0
Results from Node 66
     Destination Node ID = 65
     Packets Sent = 770
     Tx Errors = 361
     Packets Received = 4394
     SV Packet Overflows = 4
     Flood Lambda = 0
Results from Node 71
     Destination Node ID = 64
     Packets Sent = 4277
     Tx Errors = 1331
     Packets Received = 3878
     SV Packet Overflows = 8
     Flood Lambda = 0
Results from Node 72
     Destination Node ID = 63
     Packets Sent = 3898
     Tx Errors = 1531
     Packets Received = 4105
     SV Packet Overflows = 5
     Flood Lambda = 0
Results from Node 73
     Destination Node ID = 62
     Packets Sent = 4034
     Tx Errors = 1448
     Packets Received = 708
     SV Packet Overflows = 8
     Flood Lambda = 0
Results from Node 74
     Destination Node ID = 61
     Packets Sent = 3118
     Tx Errors = 1824
     Packets Received = 3858
     SV Packet Overflows = 7
     Flood Lambda = 0
Results from Node 75
     Destination Node ID = 58
     Packets Sent = 3826
     Tx Errors = 1474
     Packets Received = 4127
     SV Packet Overflows = 8
     Flood Lambda = 0
Results from Node 76
     Destination Node ID = 57
     Packets Sent = 3756
     Tx Errors = 1548
     Packets Received = 4083
     SV Packet Overflows = 8
     Flood Lambda = 0
Results from Node 77
```

```
     Destination Node ID = 56
     Packets Sent = 3805
     Tx Errors = 1547
     Packets Received = 1508
     SV Packet Overflows = 8
     Flood Lambda = 0
Results from Node 81
     Destination Node ID = 55
     Packets Sent = 77
     Tx Errors = 56
     Packets Received = 1503
     SV Packet Overflows = 4
     Flood Lambda = 0
Results from Node 82
     Destination Node ID = 54
     Packets Sent = 3299
     Tx Errors = 1378
     Packets Received = 3954
     SV Packet Overflows = 4
     Flood Lambda = 0
Results from Node 83
     Destination Node ID = 53
     Packets Sent = 3402
     Tx Errors = 1794
     Packets Received = 0
     SV Packet Overflows = 7
     Flood Lambda = 0
Results from Node 84
     Destination Node ID = 52
     Packets Sent = 3640
     Tx Errors = 1543
     Packets Received = 3962
     SV Packet Overflows = 5
     Flood Lambda = 0
Packets Sent: 0
Tx Errors: 0
Packets Received: 1953

********** 16:24:55 *******
Done
```

2.5. Program listing

2.5.1. File SV.h

```
/*          SUPERVISOR PROGRAM  V1.31                    */
/*                                                       */
/*                  written by:                          */
/*                                                       */
/*              BRIAN MICHAEL WIRTH                      */
/*                                                       */
/*              Copyright 9 1989                         */
/*                                                       */
/*                                                       */
/*      MODULE:     HEADER                               */
/*                                                       */
/*                                                       */
/*  This program is the command console for              */
/*  performing tests on AppleTalk.  It                   */
/*  communicates with other nodes running the            */
/*  'REMOTE' program version 1.3 or greater.             */
/*                                                       */
/*  NOTE:  This program will only compile                */
/*  under LightSpeed C version 3.0                       */
/*                                                       */
/*  The following Libraries are included in              */
/*  the LightSpeed project file:                         */
/*          MacTraps,                                    */
/*          AppleTalk,                                   */
/*          stdio,                                       */
/*          strings,                                     */
/*          unix.                                        */
/*                                                       */
/*  C release notes give the following table on using AppleTalk:      */
/*                                                                    */
/*                      "preferred"     "alternate"     pre-2.15      */
/*                                                                    */
/*  interface code:     MacTraps        Appletalk       Appletalk.Lib */
/*  #include file:      nAppletalk.h    Appletalk.h     Appletalk.h   */
/*  resource file:      (n/a)           (n/a)           ATalk/ABPackage */
/*                                                                    */
/* Source for <MacHeaders> is in :NEW LSC:Mac #includes.c             */
#include <MacHeaders>
#include <proto.h>
#include <time.h>
#include <stdio.h>

/*  0.  FUNCTION PROTOTYPES                              */
/*                                                       */

void main(void);
void Initialization(void);
void OpenMPP(void);
void GetNodeID(void);
void GetABRecords(void);
```

```
void SendTestPacket(void);
void OpenProtocols(void);
void MyProtoHandler(void);
void MySupervisorHandler(void);
int CloseProtocols(void);
void FloodNet(void);
void GetRemoteNodes(void);
void ShowRemoteNodes(void);
void SendSVPacket(char, int, Ptr);
int SendSVPacketA(char, int, Ptr);
int WaitForResults(void);

int MainEvent(void);
void SendQuit(void);
void SendAbort(void);
void ShowHelp(void);
void ProcessSVPacket(void);
void PrintInfo(void);
void DoQuitCommand(void);
void GetDest(void);
void SendDest(char, char);
void UpdateStats(void);
void WriteLocalResults(FILE *);
void GoAutomatic(void);
void GetFloodTime(void);
void GetPacketSize(void);
void SendPacketSize(int);
void GetFloodLambda(void);
void SendFloodLambda(double);
void Pause4(void);
void QueryNode(void);
void PrintResults(unsigned char *, unsigned char);
void WriteResults(FILE *, unsigned char *, unsigned char);
void Reset(void);
void SendReset(void);
```

2.5.2. File SV.c

```
/*          SUPERVISOR PROGRAM  V1.31                    */
/*                                                       */
/*                  written by:                          */
/*                                                       */
/*              BRIAN MICHAEL WIRTH                      */
/*                                                       */
/*              Copyright ⊜ 1989                         */
/*                                                       */
/*                                                       */
/*      MODULE:     MAIN                                 */
/*                                                       */
/*                                                       */
/*  This program is the command console for              */
/*  performing tests on AppleTalk.  It                   */
/*  communicates with other nodes running the            */
/*  'REMOTE' program version 1.3 or greater.             */
/*                                                       */
/*  NOTE:  This program will only compile                */
/*  under LightSpeed C version 3.0                       */
/*                                                       */

#include    "SV.h"
#include    "Remote Commands.h"

/*  GLOBAL VARIABLES */

long            PacketsSent;        /* total number of packets sent */
long            SendErrors;         /* total number of errors while sending */
long            PacketsRecd;        /* total number of packets received */
long            FloodTime;          /* time to flood network */
double          FloodLambda;        /* parameter for random delay during flood */

int             myNode, myNet;
int             svPacketRecd;       /* true when supervisor packet received */
int             svLAPType = 67;
ABRecHandle     svABRec;
unsigned char   svRBuf[620];        /* leave 20 extra bytes in receive buffer */
unsigned char   svTBuf[620];
int             svOverflow;         /* incremented when SV packet is overwritten
*/
unsigned char   RemoteTable[100];   /* table of remote node IDs */
int             DestSet;            /* flag to ensure dest's are set before
floods */

int             testLAPType = 66;
unsigned char   testRBuf[620];      /* leave 10 extra bytes in receive buffer */
FILE            *svlogfile;         /* supervisor log file */

char            *months[] = {   "Jan", "Feb", "Mar",
                                "Apr", "May", "Jun",
                                "Jul", "Aug", "Sep",
                                "Oct", "Nov", "Dec" };
```

```
/*   1.  SUPERVISOR MAINLINE                          */

void main()
{
    Initialization();
    OpenProtocols();
    GetABRecords();

    while (MainEvent());

    CloseProtocols();
}


/*   1.1.  INITIALIZATION                             */
/*                                                    */
/*   Initialize global variables and print            */
/*   sign on message.                                 */
/*                                                    */

void Initialization()
{
    printf("\t\tAppleTalk SuperVisor V1.31\n\n");
    Click_On(false);          /* Don't do exit window on program completion */
    svPacketRecd = false;     /* haven't received supervisor packet */
    DestSet = false;          /* haven't specified destinations yet */
    Reset();                  /* reset stats to zero */
    FloodTime = 0;
    FloodLambda = 0.0;
}


/*   1.2.  OPEN PROTOCOLS                             */
/*                                                    */
/*   Open AppleTalk drivers and get network           */
/*   node ID.  Open protocol handlers for test        */
/*   messages and supervisor messages.                */

void OpenProtocols()
{
    OSErr       Err;

    OpenMPP();
    GetNodeID();
    if ((testLAPType > 0) && (testLAPType < 128)) {
        Err = LAPOpenProtocol((char)testLAPType, &MyProtoHandler);
        if (Err != noErr) {
            printf("Error Openning Test Protocol\n");
        }
    }
    else {
        printf("OpenProtocols: Invalid Test LAP type.\n");
        SysBeep(15);
    }
    if ((svLAPType > 0) && (svLAPType < 128)) {
        Err = LAPOpenProtocol((char)svLAPType, &MySupervisorHandler);
        if (Err != noErr) {
```

```
            printf("Error Openning Supervisor Protocol\n");
        }
    }
    else {
        printf("OpenProtocols: Invalid Supervisor LAP type.\n");
        SysBeep(15);
    }
}


/*  1.2.1.  OPEN MPP                                */
/*                                                  */
/*  Open AppleTalk port.  Make sure AppleTalk       */
/*  is enabled in the CHOOSER desk accessory.       */

void OpenMPP()
{
    OSErr   Err;

    Err = MPPOpen();
    switch (Err) {
    case noErr:
        break;
    case portInUse:
        printf("MPP: Port B already in Use\n");
        break;
    case portNotCf:
        printf("MPP: Port B not configured for AppleTalk\n\tUse CHOOSER to
enable AppleTalk\n");
        break;
    }
}


/*  1.2.2.  GET NODE ID                             */
/*                                                  */
/*  Get and print current network node ID and       */
/*  network number. (The network number should      */
/*  be ZERO if no bridges are connected).           */

void GetNodeID()
{
    OSErr           Err;

    Err = GetNodeAddress(&myNode, &myNet);
    if (Err != noErr) {
        printf("MPP not installed\n\tUse CHOOSER to enable AppleTalk\n");
    }
    else {
        printf("Node ID: %d", myNode);
        printf("\t\t\tNetwork number: %d\n", myNet);
    }
}
```

```
/*   1.3.  GET AB RECORDS                          */
/*                                                 */
/*   Allocate dynamic 'AppleBus' Record for the    */
/*   supervisor protocol and initialize the        */
/*   appropriate fields.  Since no TEST messages   */
/*   are sent by the supervisor, there is no       */
/*   need to allocate a record for the test        */
/*   protocol.                                     */

void GetABRecords()
{
    svABRec = (ABRecHandle) NewHandle(lapSize);
    (**svABRec).lapProto.lapAddress.dstNodeID = 0;
    (**svABRec).lapProto.lapAddress.srcNodeID = (char) myNode;
    (**svABRec).lapProto.lapAddress.lapProtType = (char) svLAPType;
    (**svABRec).lapProto.lapDataPtr = (Ptr) svTBuf;
}


/*   1.5.  CLOSE PROTOCOLS                         */
/*                                                 */
/*   Close Test and Supervisor protocol handlers   */
/*   If they aren't properly closed, then          */
/*   OpenProtocols will return an error next       */
/*   time it is called.                            */

int CloseProtocols()
{
    OSErr       Err;

    Err = LAPCloseProtocol(testLAPType);
    if (Err == noErr) {
       printf("Test LAP Protocol Closed\n");
    }
    else {
       printf("Error Closing Test Protocol\n");
    }
    Err = LAPCloseProtocol(svLAPType);
    if (Err == noErr) {
       printf("Supervisor LAP Protocol Closed\n");
    }
    else {
       printf("Error Closing Supervisor Protocol\n");
    }
}
```

2.5.3. File SVEvent.c

```
/*          SUPERVISOR PROGRAM  V1.31                    */
/*                                                       */
/*                    written by:                        */
/*                                                       */
/*                BRIAN MICHAEL WIRTH                    */
/*                                                       */
/*                Copyright ® 1989                       */
/*                                                       */
/*                                                       */
/*      MODULE:      EVENT LOOP                          */
/*                                                       */
/*                                                       */
/*  This program is the command console for              */
/*  performing tests on AppleTalk.  It                   */
/*  communicates with other nodes running the            */
/*  'REMOTE' program version 1.3 or greater.             */
/*                                                       */
/*  NOTE:  This program will only compile                */
/*  under LightSpeed C version 3.0                       */
/*                                                       */

#include    "SV.h"
#include    "Remote Commands.h"

#define onesixty        (1.0 / 60.0)

/*                                                       */
/*  EXTERNAL GLOBAL VARIABLES                            */
/*                                                       */
/*  Defined in SV.c                                      */

extern  int             myNode, myNet;
extern  int             svPacketRecd;
extern  unsigned char   svTBuf[];
extern  unsigned char   svRBuf[];
extern  int             svOverflow;
extern  long            PacketsSent;
extern  long            SendErrors;
extern  long            PacketsRecd;
extern  long            FloodTime;
extern  double          FloodLambda;
extern  unsigned char   RemoteTable[50];    /* table of remote node IDs */
extern  int             DestSet;
extern  ABRecHandle     svABRec;
extern  int             svLAPType;

extern  FILE            *svlogfile;
extern  char            *months[];
```

```
/*   1.4.  Supervisor MAIN EVENT loop.          */
/*                                              */
/*   Check for 'Supervisor Packet Received', and*/
/*   keyboard commands and process them         */
/*   accordingly.                               */

int MainEvent()
{
    EventRecord myEvent;
    int done;

    done = false;
    SystemTask();
    if (svPacketRecd == true) {
       ProcessSVPacket();
       svPacketRecd = false;
    }
    GetNextEvent(everyEvent, &myEvent);
    switch (myEvent.what) {
    case keyDown:
    case autoKey:
       {
          char theChar;

          theChar = myEvent.message & charCodeMask;
          if ((myEvent.modifiers & cmdKey) != 0) {
             switch (theChar) {
             case 'a':
                if ((myEvent.modifiers & shiftKey) != 0) {
                   SendAbort();
                }
                else {
                   SendQuit();
                }
                done = true;
                break;
             case 'q':
                done = true;
                break;
             }
          }
          else {
             switch (theChar) {
             case 'd':
                GetDest();
                break;
             case 'f':
                FloodNet();
                break;
             case 'g':
                GoAutomatic();
                break;
             case 'i':
                PrintInfo();
                break;
             case 'l':
                GetFloodLambda();
                break;
```

```
                case 'n':
                    GetRemoteNodes();
                    break;
                case 'p':
                    ShowRemoteNodes();
                    break;
                case 'q':
                    QueryNode();
                    break;
                case 'r':
                    Reset();
                    SendReset();
                    UpdateStats();
                    break;
                case 's':
                    GetPacketSize();
                    break;
                case 't':
                    GetFloodTime();
                    break;
                case '?':
                    ShowHelp();
                    break;
                }
            }
        }
        break;
    }
    return(!done);
}

/*  1.4.1.  PROCESS SV PACKET                          */
/*                                                     */
/*  This will be called whenever an unexpected         */
/*  Supervisor message is received.  It is             */
/*  basically an error condition.                      */

void ProcessSVPacket()
{
    printf("Supervisor Packet Received (?)\n");
}
```

```
/*   1.4.2.  SEND QUIT                                  */
/*                                                      */
/*   Send a 'QUIT' command to all remotes listed*/
/*   in 'RemoteTable'.  It must be verified to  */
/*   ensure that computer in other (possibly    */
/*   unaccessable) rooms are not accidentally   */
/*   shut down.                                 */

void SendQuit()
{
    int         i;
    Str255  line;

    SysBeep(15);
    printf("Are you sure you want all remotes to QUIT? (Y to continue)");
    cgets((char *) line);
    if (line[0] == 'Y') {
        for (i = 1; i <= RemoteTable[0]; i++) {
            svTBuf[2] = QUITID;
            svTBuf[3] = QUITID;
            if (SendSVPacketA(RemoteTable[i], 4, (Ptr) svTBuf)) {
                svPacketRecd = false;
            }
        }
    }
}


/*   1.4.3.  GET DEST                                   */
/*                                                      */
/*   Acquire destination addresses for all      */
/*   remote computers.  They can be entered     */
/*   manually or paired automatically.  The     */
/*   remotes are paired by their node ID.       */
/*   The lowest number is paired with the highest*/
/*   and so on...                               */

void GetDest()
{
    int         dest, i, j;
    Str255  line;

    if (RemoteTable[0] == 0) {
        SysBeep(15);
        printf("No Remotes Identified!\n");
    }
    else {
        printf("Pair nodes automatically? ");
        cgets((char *) line);
        if ((line[0] == 'y') || (line[0] == 'Y')) {
            j = RemoteTable[0];
            if (j & 1) {        /* if j is odd then */
                SendDest(RemoteTable[1], myNode);       /* first node goes to SV
if odd */
                printf("Destination for Node %d = %d\n", RemoteTable[1],
myNode);
                for (i = 2; i <= RemoteTable[0]; i++) {
                    SendDest(RemoteTable[i], RemoteTable[j]);
```

```
                    printf("Destination for Node %d = %d\n", RemoteTable[i],
RemoteTable[j]);
                    j--;
                }
            }
            else {
                for (i = 1; i <= RemoteTable[0]; i++) {
                    SendDest(RemoteTable[i], RemoteTable[j]);
                    printf("Destination for Node %d = %d\n", RemoteTable[i],
RemoteTable[j]);
                    j--;
                }
            }
        }
        else {
            for (i = 1; i <= RemoteTable[0]; i++) {
                printf("Enter Destination address for Node %d:",
RemoteTable[i]);
                cgets((char *) line);
                sscanf((char *) line, "%d", &dest);
                SendDest(RemoteTable[i], (char) dest);
            }
        }
        DestSet = true;
    }
}


/*   1.4.3.1.  SEND DEST                              */
/*                                                    */
/*  Send a message to a specific node to inform*/
/*  it of it's destination for test messages.   */

void SendDest(node, dest)
char node;
char dest;
{
    svTBuf[2] = DESTINATIONID;
    svTBuf[3] = dest;
    if (SendSVPacketA(node, 4, (Ptr) svTBuf)) {
        svPacketRecd = false;
    }
}
```

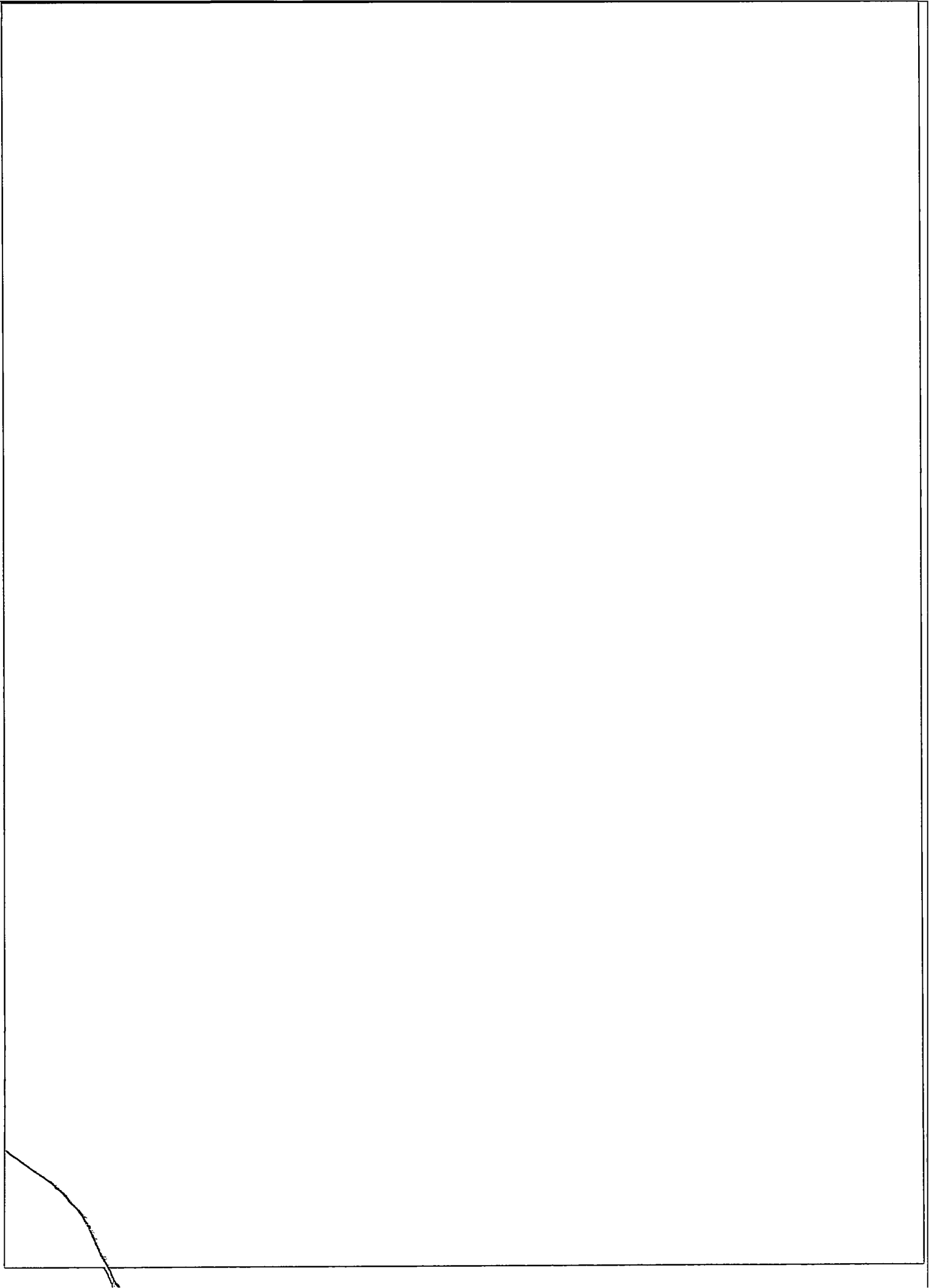
```
/*   1.4.5.  GO AUTOMATIC                                  */
/*                                                         */
/*   Control a sequence of tests based on                  */
/*   parameters from a file.  The user is asked            */
/*   for the name of the parameter file and the            */
/*   name of a log file which will be created.             */
/*   There is provisions for resetting all of              */
/*   the remote nodes following the test.  The             */
/*   following sequence of events is performed             */
/*   for each set of parameters in the file :              */
/*                                                         */
/*          Remote nodes are Reset,                        */
/*          Parameters are downloaded,                     */
/*          Test is started,                               */
/*          Upon completion, results recorded.             */
/*                                                         */
/*   The log file is closed during the course              */
/*   of each test.  This is to ensure that the             */
/*   results will be flushed to the file in the            */
/*   event of a system crash during a test.                */

void GoAutomatic()
{
    double  lambda, avgtime;
    int          i, resetremotes, packetsize;
    FILE    *testfile;
    Str255  testfilename, line, logfilename;
    tm      *trec;
    char    c;

    if (DestSet == false) {
       SysBeep(15);
       SysBeep(15);
       printf("Enter Destination Addresses First!!!\n");
       return;
    }
    printf("\tTest Parameter File Name? ");
    cgets((char *) testfilename);
    testfile = fopen((char *) testfilename, "r");
    if (testfile == NULL) {
       printf("Can't open file %s\n", testfilename);
       return;
    }
    printf("\tLog File Name? ");
    cgets((char *) logfilename);
    svlogfile = fopen((char *) logfilename, "a");
    if (testfile == NULL) {
       printf("Can't open file (%s)\n", logfilename);
       return;
    }
    printf("\tReset Remote Nodes at End of Test? ");
    cgets((char *) line);
    if ((line[0] == 'y') || (line[0] == 'Y')) {
       resetremotes = true;
    }
    else {
       resetremotes = false;
    }
```

```
    printf("Automatic Test Proceeding\n");
    for (i = 1; i <= RemoteTable[0]; i++) {
        svTBuf[2] = AUTOID;          /* instruct remotes to redirect to a file */
        if (SendSVPacketA(RemoteTable[i], 3, (Ptr) svTBuf)) {
            svPacketRecd = false;
        }
    }
    trec = localtime(NULL);
    fprintf(svlogfile, "Automatic Test Log - %s %d, %d\n", months[trec->tm_mon],
trec->tm_mday, 1900 + trec->tm_year);
    fprintf(svlogfile, "\n********** %02d:%02d:%02d *******\n",trec->tm_hour,
trec->tm_min, trec->tm_sec);
    while (fgets((char *) line, 80, testfile) != NULL) {
        if (kbhit()) {
            c = getchar();
            if (c == 'q') {
                printf("Abort!\n");
                break;
            }
        }
        sscanf((char *)line, "%ld%lf%d", &FloodTime, &avgtime, &packetsize);
        fprintf(svlogfile, "Reseting Remote Nodes\n");
        Reset();
        SendReset();
        UpdateStats();
        if (avgtime == 0.0)
            lambda = 0.0;
        else
            lambda = onesixty / avgtime;
        fprintf(svlogfile, "Sending Lambda = %lg\n(avg time = %lg)\n", lambda,
avgtime);
        SendFloodLambda(lambda);
        Pause4();
        fprintf(svlogfile, "Sending Packet Size = %d\n", packetsize);
        SendPacketSize(packetsize);
        Pause4();
        fprintf(svlogfile, "Test Time = %ld\n", FloodTime);
        fprintf(svlogfile, "Flooding Network\n");
        fclose(svlogfile);
        trec = localtime(NULL);
        printf("** %02d:%02d:%02d **\n",trec->tm_hour, trec->tm_min, trec-
>tm_sec);
        FloodNet();
        Pause4();
        svlogfile = fopen((char *) logfilename, "a");
        fprintf(svlogfile, "Querying Nodes...\n");
        for (i = 1; i <= RemoteTable[0]; i++) {
            svTBuf[2] = QUERYID;
            if (SendSVPacketA(RemoteTable[i], 3, (Ptr) svTBuf)) {
                svPacketRecd = false;
                WriteResults(svlogfile, svRBuf, RemoteTable[i]);
            }
            else {
                fprintf(svlogfile, "Node %d not responding\n", RemoteTable[i]);
            }
        }
        WriteLocalResults(svlogfile);
        trec = localtime(NULL);
```

```
        fprintf(svlogfile, "\n********** %02d:%02d:%02d *******\n",trec-
>tm_hour, trec->tm_min, trec->tm_sec);
    }
    fprintf(svlogfile, "Done\n");
    for (i = 1; i <= RemoteTable[0]; i++) {
        svTBuf[2] = DONEAUTOID;
        if (SendSVPacketA(RemoteTable[i], 3, (Ptr) svTBuf)) {
            svPacketRecd = false;
        }
    }
    printf("Done Automatic Test\n");
    if (resetremotes) {
        printf("Reseting Remote Nodes\n");
        SendAbort();
    }
}


/*  1.4.5.1.  PAUSE 4                                   */
/*                                                      */
/*  Pause for 4 seconds.  This is simply to             */
/*  account for any timing errors.  A remote            */
/*  may be one or two seconds slow since it's           */
/*  60 Hz interrupt is disabled during AppleTalk*/
/*  transmission and reception.                         */

void Pause4()
{
    long    endtime;

    endtime = Ticks + 240;          /* 4 * 60 ticks = 4 sec. */
    while (Ticks < endtime) ;
}


/*  1.4.5.2.  WRITE RESULTS                             */
/*                                                      */
/*  Write test results obtained from a remote.          */
/*  The file should already be openned and              */
/*  ready to have information appended to it.           */
/*  The contents of the received message are            */
/*  printed in a readable TEXT format.                  */

void WriteResults(log, buf, node)
FILE *log;
unsigned char *buf;
unsigned char node;
{                       /* print results of Query starting at buf[1] */
    char    i;
    long    ltemp;
    double  dtemp;

    fprintf(log, "Results from Node %hd\n", node);
    fprintf(log, "\tDestination Node ID = %hd\n", buf[2]);
    ((unsigned char *) &ltemp)[0] = buf[3];
    ((unsigned char *) &ltemp)[1] = buf[4];
    ((unsigned char *) &ltemp)[2] = buf[5];
    ((unsigned char *) &ltemp)[3] = buf[6];
```

```
    fprintf(log, "\tPackets Sent = %ld\n", ltemp);
    ((unsigned char *) &ltemp)[0] = buf[7];
    ((unsigned char *) &ltemp)[1] = buf[8];
    ((unsigned char *) &ltemp)[2] = buf[9];
    ((unsigned char *) &ltemp)[3] = buf[10];
    fprintf(log, "\tTx Errors = %ld\n", ltemp);
    ((unsigned char *) &ltemp)[0] = buf[11];
    ((unsigned char *) &ltemp)[1] = buf[12];
    ((unsigned char *) &ltemp)[2] = buf[13];
    ((unsigned char *) &ltemp)[3] = buf[14];
    fprintf(log, "\tPackets Received = %ld\n", ltemp);
    fprintf(log, "\tSV Packet Overflows = %d\n", buf[15] * 256 + buf[16]);
    for (i = 0; i < 10; i++) {
        ((unsigned char *) &dtemp)[i] = buf[i+17];
    }
    fprintf(log, "\tFlood Lambda = %lg\n", dtemp);
}


/*  1.4.5.3.  WRITE LOCAL RESULTS                */
/*                                               */
/*  The supervisor's 'local' results are also    */
/*  written to the log file.                     */

void WriteLocalResults(f)
FILE *f;
{
    fprintf(f, "Packets Sent: %ld\n", PacketsSent);
    fprintf(f, "Tx Errors: %ld\n", SendErrors);
    fprintf(f, "Packets Received: %ld\n", PacketsRecd);
}


/*  1.4.6.  PRINT INFO                           */
/*                                               */
/*  The supervisor's node ID, network number,    */
/*  number of remotes identified and local       */
/*  statistics are printed to the screen.        */

void PrintInfo()
{
    printf("Node ID = %d\n", myNode);
    printf("Network number = %d\n", myNet);
    printf("Number of Remote Nodes = %d\n",  RemoteTable[0]);
    if (svOverflow != 0) {
        printf("Supervisor Packet Overflows: %d\n", svOverflow);
    }
    printf("\n");
    UpdateStats();
}
```

```
/*   1.4.7.  GET FLOOD LAMBDA                          */
/*                                                     */
/*   The user is prompted for the flood delay          */
/*   parameter (lambda) for use in manual              */
/*   command mode.  The global variable                */
/*   'FloodLamba' is set to this value.                */

void GetFloodLambda()
{
    Str255  line;

    printf("Enter Flood Delay Parameter (lambda):");
    cgets((char *)line);
    sscanf((char *)line, "%lf", &FloodLambda);
    SendFloodLambda(FloodLambda);
}


/*   1.4.8.  GET REMOTE NODES                          */
/*                                                     */
/*   The supervisor compiles a list of all             */
/*   remote nodes on the network.  This is             */
/*   done by sending a message to all other            */
/*   possible node IDs and waiting for a               */
/*   response.  If an acknowledgement is               */
/*   received, then that node is present and           */
/*   running the remote software.                      */
/*   If the remote software is not running, the        */
/*   node may still be detected since the              */
/*   message will get through (i.e. CTS received)*/

void GetRemoteNodes()
{
    long    endtime;
    int         node, i;
    OSErr   Err;
    int         tries;

    RemoteTable[0] = 0;                         /* no remote nodes yet */
    (**svABRec).lapProto.lapAddress.srcNodeID = (char) myNode;
    (**svABRec).lapProto.lapAddress.lapProtType = (char) svLAPType;
    (**svABRec).lapProto.lapReqCount = 4;
    (**svABRec).lapProto.lapDataPtr = (Ptr) svTBuf;
    svTBuf[0] = 0;
    svTBuf[1] = 4;
    svTBuf[2] = SUPERVISORID;
    svTBuf[3] = (char) myNode;
    printf("Getting Remote Node IDs ..\n");

    i = 0;
    for (node = 1; node < 128; node++) {    /* do for all nodes */
        tries = 0;
        do {
            (**svABRec).lapProto.lapAddress.dstNodeID = node;
            Err = LAPWrite(svABRec, false);
            switch (Err) {
            case noErr:
                endtime = Ticks + 20;    /* wait maximum 20 ticks for reply */
```



```
                while ((Ticks <= endtime) && (!svPacketRecd)) ;
                if (svPacketRecd) {
                    i++;
                    RemoteTable[i] = node;
                    svPacketRecd = false;
                    printf("\tRemote %hd Identified\n", RemoteTable[i]);
                    tries = 99;
                }
                else {
                    printf("\tRemote %hd Identified but not Acknowledged\n",
node);
                    tries ++;
                }
                break;
            case excessCollsns:
                tries = 99;              /* only try once if excess collisions */
                break;
            }
        } while (tries < 4);
    }
    RemoteTable[0] = i;
    printf("%hd Remotes Identified\n", RemoteTable[0]);
}


/*   1.4.9.  SHOW REMOTE NODES                    */
/*                                                */
/*   Print the supervisors list of remotes to     */
/*   the screen.                                  */

void ShowRemoteNodes()
{
    int i;

    if (RemoteTable[0] == 0) {
        printf("No Remote Nodes Identified.\n");
    }
    else {
        printf("%hd Remote Nodes Identified:\n", RemoteTable[0]);
        for (i = 1; i <= RemoteTable[0]; i++) {
            printf("Remote Node %hd\n", RemoteTable[i]);
        }
        printf("Supervisor Node %hd\n", myNode);
    }
}
```

```
/*   1.4.10.  QUERY NODE                                */
/*                                                      */
/*   Used in manual command mode to get a               */
/*   remote computers current statistics.  The          */
/*   user is prompted for the node ID of the            */
/*   desired remote.                                    */
void QueryNode()
{
    int         node;
    Str255  line;

    printf("Enter Node ID:");
    cgets((char *)line);
    sscanf((char *)line, "%d", &node);
    printf("Querying Node %d\n", node);
    svTBuf[2] = QUERYID;
    if (SendSVPacketA((char)node, 3, (Ptr) svTBuf)) {
        PrintResults(svRBuf, (char) node);
        svPacketRecd = false;
    }
}

/*   1.4.10.1.  PRINT RESULTS                           */
/*                                                      */
/*   Prints the results obtained from a remote          */
/*   query.  This is similar to WRITE RESULTS           */
/*   but prints the results to the screen rather*/
/*   than to a file.                                    */
void PrintResults(buf, node)
unsigned char *buf, node;
{
    char    i;
    long    ltemp;
    double  dtemp;

    printf("Results from Node %d\n", node);
    printf("\tDestination Node ID = %d\n", buf[2]);
    ((unsigned char *) &ltemp)[0] = buf[3];
    ((unsigned char *) &ltemp)[1] = buf[4];
    ((unsigned char *) &ltemp)[2] = buf[5];
    ((unsigned char *) &ltemp)[3] = buf[6];
    printf("\tPackets Sent = %ld\n", ltemp);
    ((unsigned char *) &ltemp)[0] = buf[7];
    ((unsigned char *) &ltemp)[1] = buf[8];
    ((unsigned char *) &ltemp)[2] = buf[9];
    ((unsigned char *) &ltemp)[3] = buf[10];
    printf("\tTx Errors = %ld\n", ltemp);
    ((unsigned char *) &ltemp)[0] = buf[11];
    ((unsigned char *) &ltemp)[1] = buf[12];
    ((unsigned char *) &ltemp)[2] = buf[13];
    ((unsigned char *) &ltemp)[3] = buf[14];
    printf("\tPackets Received = %ld\n", ltemp);
    printf("\tSV Packet Overflows = %d\n", buf[15] * 256 + buf[16]);
    for (i = 0; i < 10; i++) {
        ((unsigned char *) &dtemp)[i] = buf[i+17];
    }
    printf("\tFlood Lambda = %lg\n", dtemp);
}
```

```
/*   1.4.11.  GET PACKET SIZE                          */
/*                                                     */
/*   The user is prompted for the length of the        */
/*   test messages used in the tests.  A check         */
/*   is made to ensure that the length is within       */
/*   the allowable limits of AppleTalk (0-600)         */

void GetPacketSize()
{
    Str255  line;
    int         Psize;

    printf("Enter Test Packet Size: ");
    cgets((char *)line);
    sscanf((char *)line, "%d", &Psize);
    if ((Psize > 600) || (Psize < 1)) {
        SysBeep(15);
        printf("Invalid Test Packet Size (%d)\n", Psize);
    }
    else {
        SendPacketSize(Psize);
        printf("Sent packet size = %d bytes\n", Psize);
    }
}


/*   1.4.12.  GET FLOOD TIME                           */
/*                                                     */
/*   The user is prompted for the duration of          */
/*   the next test. (Used only in manual command       */
/*   mode).  The global variable 'FloodTime' is        */
/*   set to the value entered (in seconds).            */

void GetFloodTime()
{
    Str255  line;

    printf("Enter Flood Time:");
    cgets((char *)line);
    sscanf((char *)line, "%ld", &FloodTime);
}
```

```
/*   1.4.13.  SHOW HELP                                  */
/*                                                       */
/*   A list of all available commands is printed*/
/*   to the display.                                     */

void ShowHelp()
{
     printf("SuperVisor Commands:\n");
     printf("d\tGet Destination Addresses\n");
     printf("f\tFlood Network\n");
     printf("g\tGo automatic test\n");
     printf("i\tShow Info and Statistics\n");
     printf("l\tSet Flood Delay (lambda)\n");
     printf("n\tBuild Network Node Table\n");
     printf("p\tPrint Network Node Table (to screen)\n");
     printf("q\tQuery Node for Results\n");
     printf("r\tReset Statistics\n");
     printf("s\tSet Packet Size\n");
     printf("t\tSet Flood Time\n");
     printf("?\tPrint Help\n");
     printf("command-a\tQuit (all nodes)\n");
     printf("command-A\tRestart all remote nodes\n");
     printf("command-q\tQuit\n");
}
```

2.5.4. File SVUtil.c

```
/*          SUPERVISOR PROGRAM  V1.31          */
/*                                             */
/*                written by:                  */
/*                                             */
/*             BRIAN MICHAEL WIRTH             */
/*                                             */
/*             Copyright © 1989                */
/*                                             */
/*                                             */
/*      MODULE:     UTILITIES                  */
/*                                             */
/*                                             */
/*  This program is the command console for    */
/*  performing tests on AppleTalk.  It         */
/*  communicates with other nodes running the  */
/*  'REMOTE' program version 1.3 or greater.   */
/*                                             */
/*  NOTE:  This program will only compile      */
/*  under LightSpeed C version 3.0             */
/*                                             */

#include    "SV.h"
#include    "Remote Commands.h"

/*  EXTERNAL GLOBAL VARIABLES                  */
/*                                             */
/*  Defined in SV.c                            */
/*                                             */

extern  int             myNode;
extern  int             svPacketRecd;
extern  unsigned char   svTBuf[];
extern  unsigned char   svRBuf[];
extern  int             svOverflow;
extern  long            PacketsSent;
extern  long            SendErrors;
extern  long            PacketsRecd;
extern  long            FloodTime;
extern  double          FloodLambda;
extern  unsigned char   RemoteTable[];   /* table of remote node IDs */
extern  int             DestSet;
extern  ABRecHandle     svABRec;
extern  int             svLAPType;
```

```
/*  2.1.  RESET                                     */
/*                                                  */
/*  Reset local statistics, counters and errors.*/

void Reset()
{
    PacketsSent = 0;
    SendErrors = 0;
    PacketsRecd = 0;
    svOverflow = 0;
}


/*  2.2. SEND ABORT                                 */
/*                                                  */
/*  Send an 'ABORT' command to all remotes          */
/*  listed in 'RemoteTable'.  This command          */
/*  must be verified to prevent accidental          */
/*  restart of the remote computers. (They          */
/*  may not be accessable).                         */

void SendAbort()
{
    int     i;
    Str255  line;

    SysBeep(15);
    printf("Are you sure you want all remotes to Restart? (Y to continue)");
    cgets((char *) line);
    if (line[0] == 'Y') {
        for (i = 1; i <= RemoteTable[0]; i++) {
            svTBuf[2] = ABORTID;
            svTBuf[3] = ABORTID;
            if (SendSVPacketA(RemoteTable[i], 4, (Ptr) svTBuf)) {
                svPacketRecd = false;
            }
        }
    }
}
```