Appendix A

FreeBSD Raw Data Samples

Trpt Data Collected at Ravel (Fixed Host)

The lines shown below were taken from a file containing trpt data. Trpt reads the data contained in TCP's socket debugging buffer and prints it to a screen or to a file. Since the actual file for a 2 million byte transfer usually contains 800 or more lines, only several small samples are shown below. Each sample is formatted the same. The first column shows the time in milliseconds since midnight. The second indicates the type of entry (input, output, ackp, user, drop). For our studies we are only interested in the feedback from the receiver and the TCP sender's response. Therefore, no outgoing packets were logged. Next the sequence range of the packets outstanding or ready to be sent is shown. Then the congestion window (cwnd), the number of duplicate acknowledgments (dack), the round-trip-time (rtt) in 500 ms ticks, and the size of the receiver's advertised window (rwin) are shown. The items contained within the brackets are the flags set on the packet associated with the entry. Finally, the TCP state which the connection enters as a result of the packet being received is shown. Usually the state does not change.

The first trpt sample fragment shown below was taken from the very beginning of a transfer over a wireless link using standard TCP Reno. The first packet is part of the 3-way handshake used by TCP to establish a connection and this is why both the SYN and ACK bits are set. All of the other lines in this sample represent normal acknowledgments from the receiver (i.e. no packets have been lost). The sample shows that ACKs are coming back about once every 15 milliseconds. This is typical for our LAN testbed since the RTT for a packet is between 6 to 8 milliseconds, and TCP typically only sends a cumulative acknowledgment once for every two or more data packets during bulk transfers.

```
65703810 input 1840002:1840002 cwnd=65535 dack=0 rtt=0 ack=1840002 rwin=17520 <SYN,ACK> -> ESTABLISHED
65703840 input 1843510:1856650 cwnd=65535 dack=0 rtt=0 ack=1843510  rwin=17520 <ACK> -> ESTABLISHED
65703855 input 1846430:1861030 cwnd=65535 dack=0 rtt=1 ack=1846430  rwin=17520 <ACK> -> ESTABLISHED
65703869 input 1849350:1863950 cwnd=65535 dack=0 rtt=1 ack=1849350  rwin=17520 <ACK> -> ESTABLISHED
65703883 input 1852270:1866870 cwnd=65535 dack=0 rtt=1 ack=1852270  rwin=17520 <ACK> -> ESTABLISHED
65703897 input 1855190:1869790 cwnd=65535 dack=0 rtt=1 ack=1855190  rwin=17520 <ACK> -> ESTABLISHED
65703913 input 1858110:1872710 cwnd=65535 dack=0 rtt=0 ack=1858110  rwin=17520 <ACK> -> ESTABLISHED
65703927 input 1861030:1875630 cwnd=65535 dack=0 rtt=1 ack=1861030  rwin=17520 <ACK> -> ESTABLISHED
65703942 input 1863950:1878550 cwnd=65535 dack=0 rtt=1 ack=1863950  rwin=17520 <ACK> -> ESTABLISHED
65703956 input 1866870:1881470 cwnd=65535 dack=0 rtt=1 ack=1866870  rwin=17520 <ACK> -> ESTABLISHED
```

The second trpt output fragment shown below was taken from the middle of the same data transfer. The first few lines indicate duplicate acknowledgments that have been received due to a packet loss on the wireless link. The sender invokes Fast-Recovery after the third duplicate ACK (i.e. after dack=3) and as the sample shows, the sender cuts the congestion window in half (from roughly 8 KB to 4 KB). Several new ACKs come back indicating that the connection is making progress and then the duplicate ACKs appear again because of another loss very close to the first packet loss. The congestion window in only large enough to allow 3 packets to be outstanding. The first of the three has been lost, and the next two generate duplicate acknowledgments. Then the sender stalls. It's window is full and it doesn't have three duplicate ACKs so it can't perform Fast-Recovery. After about 1.15 seconds the sender's retransmission timer expires and the packet is retransmitted as shown in the last line of the sample.

```
65705938 drop  2246470:2253770 cwnd=7300 dack=3 rtt=0 ack=2246470 rwin=17520 <ACK> -> ESTABLISHED
65705941 drop  2246470:2255230 cwnd=8760 dack=4 rtt=1 ack=2246470 rwin=17520 <ACK> -> ESTABLISHED
65705955 input 2253770:2255230 cwnd=4380 dack=0 rtt=1 ack=2253770 rwin=10220 <ACK> -> ESTABLISHED
65705957 input 2253770:2258150 cwnd=4380 dack=0 rtt=1 ack=2253770 rwin=17520 <ACK> -> ESTABLISHED
65705973 input 2253770:2258150 cwnd=4380 dack=1 rtt=1 ack=2253770 rwin=17520 <ACK> -> ESTABLISHED
65705974 input 2253770:2258150 cwnd=4380 dack=2 rtt=1 ack=2253770 rwin=17520 <ACK> -> ESTABLISHED
65707121 user  2253770:2258150 SLOWTIMO<REXMT> -> ESTABLISHED
```

The next trpt sample is taken from the end of the same data transfer. It shows the sender finishing and passing through the various states required to close the connection. No packets are lost in this sample and the congestion window has managed to slowly grow in size after the loss above.

```
65727795 input 3819478:3825318 cwnd=9374 dack=0 rtt=1 ack=3819478 rwin=17520 <ACK> -> ESTABLISHED
65727802 input 3820938:3828238 cwnd=9601 dack=0 rtt=1 ack=3820938 rwin=17520 <ACK> -> ESTABLISHED
65727817 input 3823858:3829698 cwnd=9823 dack=0 rtt=0 ack=3823858 rwin=17520 <ACK> -> ESTABLISHED
65727831 input 3826778:3832618 cwnd=10040 dack=0 rtt=1 ack=3826778 rwin=17520 <ACK> -> FIN_WAIT_1
65727845 input 3829698:3835538 cwnd=10252 dack=0 rtt=1 ack=3829698 rwin=17520 <ACK> -> FIN_WAIT_1
65727860 input 3832618:3839918 cwnd=10459 dack=0 rtt=0 ack=3832618 rwin=17520 <ACK> -> FIN_WAIT_1
65727874 input 3835538:3840003 cwnd=10662 dack=0 rtt=1 ack=3835538 rwin=17520 <ACK> -> FIN_WAIT_1
65727888 input 3838458:3840003 cwnd=10861 dack=0 rtt=1 ack=3838458 rwin=17520 <ACK> -> FIN_WAIT_1
65727890 input 3840003:3840003 cwnd=11057 dack=0 rtt=0 ack=3840003 rwin=17436 <ACK> -> FIN_WAIT_2
65727891 input 3840003:3840003 cwnd=11057 dack=0 rtt=0 ack=3840003 rwin=17520 <ACK,FIN> -> TIME_WAIT
```

This last sample was taken from a separate data transfer. It shows how the "ackp" entries appear in the debugging buffer when the ACKP protocol is enabled at the base station. Since a partial acknowledgment does not come from the receiver, some of the fields such as “ack” are set to zero.

```
60868877 ackp  800653666:800671186 cwnd=65535 dack=0 rtt=1 ack=0  rwin=17280  -> ESTABLISHED
60868886 input  800656586:800671186 cwnd=65535 dack=0 rtt=1 ack=800656586  rwin=17520 <ACK> -> ESTABLISHED
60868889 ackp  800656586:800674106 cwnd=65535 dack=0 rtt=1 ack=0  rwin=17280  -> ESTABLISHED
60868891 ackp  800656586:800674106 cwnd=65535 dack=0 rtt=1 ack=0  rwin=17280  -> ESTABLISHED
60868900 input  800659506:800674106 cwnd=65535 dack=0 rtt=1 ack=800659506  rwin=17520 <ACK> -> ESTABLISHED
60868904 ackp  800659506:800677026 cwnd=65535 dack=0 rtt=1 ack=0  rwin=17280  -> ESTABLISHED
60868906 ackp  800659506:800677026 cwnd=65535 dack=0 rtt=1 ack=0  rwin=17280  -> ESTABLISHED
60868915 input  800662426:800677026 cwnd=65535 dack=0 rtt=1 ack=800662426  rwin=17520 <ACK> -> ESTABLISHED
60868918 ackp  800662426:800679946 cwnd=65535 dack=0 rtt=1 ack=0  rwin=17280  -> ESTABLISHED
60868920 ackp  800662426:800679946 cwnd=65535 dack=0 rtt=1 ack=0  rwin=17280  -> ESTABLISHED
```

Tcpdump Data Collected at Chopin (Mobile Host)

The lines shown below were taken from a file containing tcpdump data. Tcpdump filters the data captured at the link-layer by the BPF and prints it to a screen or to a file. Since the actual file for a 2 million byte transfer usually contains two thousand or more lines, only several small samples are shown below. Each sample is formatted the same. The first column shows the time in the format HH:MM:SS.USEC. The next set of fields shows the packet's source followed by its destination. Each is in the format machine-name.port. The data for each of these tcpdump samples is flowing from Ravel (the fixed host) to Chopin-wv (the mobile host). Therefore, Ravel is the source for data packets and Chopin is the source for ACKs. The next field is a single character which indicates the header flags that are set within the packet. Then for data packets the starting and ending sequence numbers are given along with the size in parenthesis. For ACKs the sequence number acknowledged is given. Data packets also contain an acknowledgment value, but the value is always the same in a unidirectional transfer. The advertised window is shown after the ACK field. Finally, the DF flag is set indicating not to fragment the packet.

The first tcpdump sample shown below was taken from the very beginning of a transfer over a wireless link using standard TCP Reno. The first three lines are the 3-way handshake that establishes the connection (SYN, SYN+ACK, ACK). During the connection establishment phase various TCP options may be negotiated. In this example the maximum segment size (mss) is negotiated as 1460 bytes. These initial packets also contain the entire sequence number. After that all sequence numbers are given as offsets from the initial values so that the numbers are smaller and easier to handle. Once the connection is established, normal data flow occurs with the receiver sending an ACK once every two or more packets. Note that Slow-Start is not invoked at the beginning of the connection because the transfer is local.

```
12:20:29.539127 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: S 6688001:6688001(0) win 16384 <mss 1460> (DF)
12:20:29.539410 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: S 80994817:80994817(0) ack 6688002 win 17520 <mss 1460> (DF)
12:20:29.541822 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: . ack 1 win 17520 (DF)
12:20:29.551705 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: . 1:1461(1460) ack 1 win 17520 (DF)
12:20:29.554497 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: P 1461:2049(588) ack 1 win 17520 (DF)
12:20:29.562794 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: . 2049:3509(1460) ack 1 win 17520 (DF)
12:20:29.562984 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 3509 win 17520 (DF)
12:20:29.569946 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: . 3509:4969(1460) ack 1 win 17520 (DF)
12:20:29.577116 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: . 4969:6429(1460) ack 1 win 17520 (DF)
12:20:29.577298 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 6429 win 17520 (DF)
```

This second tcpdump sample shows three duplicate acknowledgments being sent. Then the lost packet is retransmitted and the connection continues without any time-out occurring. This can be verified by looking at the time-stamps for the packets which differ by no more than about 10 milliseconds.

```
12:20:31.124461 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 32709 win 1460 (DF)
12:20:31.125127 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 32709 win 9652 (DF)
12:20:31.125779 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 32709 win 17520 (DF)
12:20:31.136154 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: . 32709:34169(1460) ack 1 win 17520 (DF)
12:20:31.140077 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 34169 win 17520 (DF)
12:20:31.143283 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: . 34169:35629(1460) ack 1 win 17520 (DF)
12:20:31.153674 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: . 35629:37089(1460) ack 1 win 17520 (DF)
12:20:31.153866 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 37089 win 17520 (DF)
```

This third tcpdump sample illustrates what happens when two losses occur in close proximity. The first packet lost is 194769. There are numerous duplicate ACKs sent for this packet, and the connection also continues to transfer new data during the Fast-Recovery period (i.e. after the third duplicate ACK). The lost packet is resent on line 23. The next ACK confirms that the retransmitted packet was successfully received so the Fast-Recovery phase ends and the congestion window is cut in half. Unfortunately, not all of the data outstanding was acknowledged because another packet very close to the first was also lost. With the congestion window now so small, not enough data can be sent to generate three duplicate ACKs and the transfer stalls for 1.1 seconds. Finally, a time-out at the sender occurs which breaks the temporary state of deadlock by resending the lost packet.

```
12:20:31.933703 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: . 193309:194769(1460) ack 1 win 17520 (DF)
12:20:31.940075 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 194769 win 17520 (DF)
12:20:31.940927 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: P 194769:196229(1460) ack 1 win 17520 (DF)
12:20:31.947998 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: . 196229:197689(1460) ack 1 win 17520 (DF)
12:20:31.948132 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 194769 win 17520 (DF)
12:20:31.955143 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: P 197689:199149(1460) ack 1 win 17520 (DF)
12:20:31.962270 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: . 199149:200609(1460) ack 1 win 17520 (DF)
12:20:31.962402 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 194769 win 17520 (DF)
12:20:31.969420 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: P 200609:202069(1460) ack 1 win 17520 (DF)
12:20:31.969611 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 194769 win 17520 (DF)
12:20:31.976593 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: . 202069:203529(1460) ack 1 win 17520 (DF)
12:20:31.976781 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 194769 win 17520 (DF)
12:20:31.983762 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: P 203529:204989(1460) ack 1 win 17520 (DF)
12:20:31.983946 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 194769 win 17520 (DF)
12:20:31.990935 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: . 204989:206449(1460) ack 1 win 17520 (DF)
12:20:31.991122 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 194769 win 17520 (DF)
12:20:31.998129 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: P 206449:207909(1460) ack 1 win 17520 (DF)
12:20:31.998314 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 194769 win 17520 (DF)
12:20:32.005324 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: . 207909:209369(1460) ack 1 win 17520 (DF)
12:20:32.005515 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 194769 win 17520 (DF)
12:20:32.012520 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: P 209369:210829(1460) ack 1 win 17520 (DF)
12:20:32.012709 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 194769 win 17520 (DF)
12:20:32.019707 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: . 194769:196229(1460) ack 1 win 17520 (DF)
12:20:32.019902 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 197689 win 14600 (DF)
12:20:32.020559 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 197689 win 17520 (DF)
12:20:33.124572 ravel.cs.tamu.edu.1054 > chopin-wv.cs.tamu.edu.discard: . 197689:199149(1460) ack 1 win 17520 (DF)
12:20:33.124766 chopin-wv.cs.tamu.edu.discard > ravel.cs.tamu.edu.1054: . ack 210829 win 4380 (DF)
```

Appendix B

Configuration and Test Programs

```
/******************************************************************************
 *                                                                            *
 *                              Stephen West                                  *
 *                            swest@cs.tamu.edu                               *
 *                           Test Sender Program                              *
 *                                                                            *
 *    Directions:  This file must be compiled as follows:                     *
 *                                                                            *
 *                 > cc -o test_sender test_sender.c                          *
 *                   or                                                       *
 *                 > cc -o test_sender test_sender.c -lsocket -lnsl           *
 *                                                                            *
 *    Reference:      Internetworking with TCP/IP, Douglas E.                 *
 *                    Comer David L. Stevens Vol. I,III                       *
 *                    UNIX Networking Programing, Stevens                     *
 *                                                                            *
 *    Affiliation:    Department of Computer Science                          *
 *                    Texas A&M University                                    *
 *                    College Station, TX 77843-3112                          *
 *                                                                            *
 ******************************************************************************/

#include <sys/types.h>
#include <sys/param.h>
#include <sys/time.h>
#ifdef  _AIX
#include <sys/select.h>
#endif
#include <sys/socket.h>
#include <sys/file.h>
#include <sys/ioctl.h>

#include <netinet/in_systm.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_var.h>
#include <netinet/ip_icmp.h>
#include <netinet/udp.h>
#include <netinet/tcp.h>
#include <netdb.h>
#include <ctype.h>

#include <stdio.h>
#include <errno.h>
#include <string.h>

#define TCP_TAMU_STATS

#define INPUT_LINE           80
#define TEST_ARRAY_SIZE 2000000


/* --------------------------- Function Prototypes -------------------------- */

int main_menu(void);
int tcp_stats_menu(void);
void send_test_data(void);
void sys_err(char *str);
void sys_msg(char *str);
void test_array_init(void);
long timediff(struct timeval lasttime);
int write_n(int fd);


/******************************************************************************
 *
 *  This program is used to send a stream of data from the local machine to a
 *  remote machine using a TCP connection.  The performance obtained during the
 *  transfer can then be measured both at the sender and the receiver.
 *
 ******************************************************************************/
```

```
char Test_Array[TEST_ARRAY_SIZE];
char dest_machine[INPUT_LINE] = "chopin-wv";
int dest_port=9;
int enable_stats=0;

main()
{

    int response=0, sub_menu_selection=0;

/*
 * Initialize the array structure used for sending transfers.
 */

    test_array_init();

    while(response != 1){
        response = main_menu();
        switch(response){
            case 1:
                break;
            case 2:
                sys_msg("FEATURE NOT IMPLEMENTED IN SENDER PROGRAM");
                break;
            case 3:
                send_test_data();
                break;
            case 4:
                sub_menu_selection = tcp_stats_menu();
                switch(sub_menu_selection){
                    case 1:
                        break;
                    case 2:
                        enable_stats = 1;
                        sys_msg("\n\n\t TCP stats collection enabled \n");
                        break;
                    case 3:
                        enable_stats = 0;
                        sys_msg("\n\n\t TCP stats collection disabled \n");
                        break;
                    default:
                        sys_err("Invalid Sub-Menu Response Returned");
                        break;
                }
                break;
            default:
                sys_err("Invalid Main Menu Response Returned");
                break;
        }
    }
    system("clear");

}


/******************************************************************************
 *
 *    The function main_menu displays the main menu for the wireless
 *    configuration program.
 *        Input:    User Selections
 *        Output:   Menu Number Related To The User's Selection
 *
 ******************************************************************************/
int main_menu(void)
{

    int response=0;

    while((response < 1) || (response > 4)){
        system("clear");
        printf("\n\n\n\t\t TCP NETWORK TEST PROGRAM \n");
        printf("\t\t\t Main Menu \n\n\n");
        printf("\t 1) Exit the Program\n");
        printf("\t 2) Set/Check TCP Enhancements - Not Needed at Sender\n");
        printf("\t 3) Send Test Data\n");
        printf("\t 4) Set/Check TCP Statistics\n");
        printf("\n\n\t Enter Selection (1 - 4) --> ");

        scanf("%d", &response);
    }

    return response;
}


/*****************************************************************************
 *
 *    The function tcp_stats_menu allows the user to enable and disable tcp
 *    statistical collection via tcp_trace().  Once stats collection is enabled,
 *    it will be used for each transfer until it is disabled.  The user must
 *    be careful not to overfill the stats buffer within the kernel during a
 *    transfer.  The buffer is circular and has a size of 2000 packets currently.
 *    After each run in which stats are collected, trpt should be run to store
 *    the stats to a file.
 *        Input:    User's Choice
 *        Output:   Menu Item Number Selected By The User
 *
 *****************************************************************************/
int tcp_stats_menu(void)
{

    int response=0;

    while((response < 1) || (response > 3)){
        system("clear");
        printf("\n\n\n\t\t TCP NETWORK TEST PROGRAM \n");
        printf("\t\t   TCP Statistics Menu \n\n");
        if(enable_stats == 1)
            printf("\t      TCP stats collection:  Enabled\n\n\n");
        if(enable_stats == 0)
            printf("\t    TCP stats collection:  Not Enabled\n\n\n");
        printf("\t 1) Exit to the main menu\n");
        printf("\t 2) Enable tcp stats collection \n");
        printf("\t 3) Disable tcp stats collection \n");
        printf("\n\n\t Enter Selection (1 - 3) --> ");

        scanf("%d", &response);
    }

    return response;
}


/*****************************************************************************
 *
 *  The function send_test_data is the heart of the program.  It sends a set of
 *  test data to the remote machine and records the transfer time and amount of
 *  data sent.  A menu is displayed which askes the user if they want to use
 *  the default settings, or enter new ones.  Any new settings entered by the
 *  user will become the new default.  If the enable_stats flag is set, socket
 *  debugging is also turned on within the kernel so that tcp_trace() will
 *  capture sender tcp statistics during the transfer.  This data can then be
 *  examined with the trpt program.  Socket debugging is on a per connection
 *  basis.  Therefore, it must be re-enabled for each transfer.  When the
 *  socket is closed, the debugging will automatically be turned off for that
 *  socket.
 *      Input:    User's Menu Choices and Destination Machine Name & Port Number
 *      Output:   Test Data to Remote Machine and Transfer Time & Size
 *
 *****************************************************************************/
void send_test_data(void)
```

```
{
    struct sockaddr_in dest_addr;
    char hostname[100], hostaddr[100];
    register struct hostent *hp;
    int debug_enable, debug_len;
    int option, option_len;
    int sockfd;

    int response1=0, response2=0;
    struct timeval start_time;
    long transfer_time_usec;
    double transfer_time_sec;
    int bytes_written;

    system("clear");
    printf("\t\t TCP NETWORK TEST PROGRAM\n");
    printf("\t    Destination Machine Selection Menu\n\n");

    printf("\t     Default dest. machine:  %s \n", dest_machine);
    printf("\t     Default dest. port number:  %d \n", dest_port);
    if(enable_stats == 1)
        printf("\t      TCP stats collection:  Enabled\n\n\n");
    if(enable_stats == 0)
        printf("\t    TCP stats collection:  Not Enabled\n\n\n");


/*
 * Get the destination machine from the user and try and convert it using
 * gethostbyname() function.
 */

    while((response1 < 1) || (response1 > 3)){
        printf("\t 1) Exit the Program \n");
        printf("\t 2) Use the default settings \n");
        printf("\t 3) Change the default settings.\n");
        printf("\n\t Enter Selection (1 - 3) --> ");
        scanf("%d", &response1);
    }

    if(response1 == 1)
        return;
    else if(response1 == 3){
        printf("\n\t Enter destination machine name  --> ", dest_machine);
        scanf("%s", dest_machine);
    }

    if((hp = gethostbyname(dest_machine)) == NULL){
        printf("\n\n\t Unable to find destination machine \"%s\"\n",
            dest_machine);
        printf("\t 1) Re-enter the destination machine name\n");
        printf("\t 2) Abort the transfer\n");
        printf("\t Enter Selection (1 - 2) --> ");
        scanf("%d", &response2);
    }

    while((hp == NULL) && (response2 !=2)){
        printf("\n\t Enter destination machine name  --> ", dest_machine);
        scanf("%s", dest_machine);
        if((hp = gethostbyname(dest_machine)) == NULL){
            printf("\n\n\t Unable to find destination machine \"%s\"\n",
                dest_machine);
            printf("\t 1) Re-enter the destination machine name\n");
            printf("\t 2) Abort the transfer\n");
            printf("\t Enter Selection (1 - 2) --> ");
            scanf("%d", &response2);
        }

    }

    if(response2 == 2)
        return;
    strcpy(hostname,hp->h_name);
    strcpy(hostaddr,(char *)inet_ntoa(*(struct in_addr *)hp->h_addr));


/*
 * Get the destination port from the user.
 */

    if(response1 == 3){
        printf("\t Enter the dest port number --> ");
        scanf("%d", &dest_port);
    }

    if((sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0){
        sys_msg("Socket Error - Aborting Transfer");
        return;
    }

    dest_addr.sin_addr.s_addr = inet_addr(hostaddr);
    dest_addr.sin_port = htons(dest_port);
    dest_addr.sin_family = AF_INET;
/*
 * Set socket option to monitor tcp variables and record them
 * in the buffer.
 */

    if(enable_stats == 1){
        debug_enable = 1;
        if(setsockopt(sockfd, SOL_SOCKET,SO_DEBUG,(char *) &debug_enable,
                sizeof(debug_enable)) < 0)
            printf("\t Unable to Set Socket to Debug Mode!\n");

        debug_enable = 0;
        debug_len = sizeof(debug_enable);
        if(getsockopt(sockfd,SOL_SOCKET,SO_DEBUG,(char *) &debug_enable,
                &debug_len) < 0)
            printf("\t Unable to Get Socket Debug Status!\n");

        if(debug_enable == 0)
            printf("\t SO_DEBUG Not Set to 1!\n");
        else
            printf("\n\t SO_DEBUG Successfully Turned On.\n");
    }


/*
 * Start the timer and open the connection.
 */

    printf("\t Starting transfer ... ");
    fflush(stdout);

    gettimeofday(&start_time, NULL);

    if(connect(sockfd,(struct sockaddr *)&dest_addr, sizeof(dest_addr)) < 0){
        sys_msg("\t Unable to Connect to Destination - Aborting Transfer");
        return;
    }

    bytes_written = write_n(sockfd);

    close(sockfd);


/*
 *    Finished with transfer.  Calculate amount of data sent and transfer time.
 */

    transfer_time_usec = timediff(start_time);
    transfer_time_sec = ((double) transfer_time_usec)/1000000;
    printf("%d Bytes Sent \n", bytes_written);
    printf("\t Total Transfer Time = %6.3f Seconds\n", transfer_time_sec);


    sys_msg("");

}


/*****************************************************************************
 *
 *  The function sys_err takes a string and displays it for the user.
 *  Then it terminates the program with the error.  It is intended to be used
 *  with fatal errors which must terminate the program, not for program
 *  warnings.
 *      Inputs:    Error Message
 *      Outputs:   Message to The Screen
 *
 *****************************************************************************/
void sys_err(char *str)
{
    printf("%s\n",str);
    exit(1);
}


/*****************************************************************************
 *
 *  The function sys_msg takes a string and displays it for the user.  It then
 *  pauses until the user presses the return key.  Unlike sys_err, it is not
 *  intended to be used with fatal errors.  It displays non-fatal errors and
 *  other general information.
 *      Inputs:    Error Message
 *      Outputs:   Message to The Screen
 *
 *****************************************************************************/
void sys_msg(char *str)
{
    int response;

    printf("%s\n",str);
    printf("\t Press the return key to continue\n");
    response = getchar();
    response = getchar();
}


/*****************************************************************************
 *
 *  The function test_array_init fills the array of characters called
 *  Test_Array with character values in the range of 48-122 (decimal).  This
 *  is 0 - z in terms of characters.  The idea was to have easily recognized
 *  printable characters in case they are ever printed to the screen.
 *      Input:    Empty Array
 *      Output:   Initialized Array
 *
 *****************************************************************************/
void test_array_init(void)
{
    int i;
    char j=48;

    for(i=0; i < TEST_ARRAY_SIZE; i++){
        Test_Array[i] = j;
        j++;
        if(j > 122)
            j = 48;
    }
}
```

```
/******************************************************************************
 *
 *  The timediff function checks the time that has elapsed during the transfer.
 *      Input:    None
 *      Output:   The time interval in microseconds
 *
 ******************************************************************************/
long timediff(struct timeval lasttime)
{
    struct timeval curtime;
    long cur=0, last=0;

    gettimeofday(&curtime, NULL);
    cur=(curtime.tv_sec%10000)*1000000+curtime.tv_usec;
    last=(lasttime.tv_sec%10000)*1000000+lasttime.tv_usec;

    return (cur-last);
}


/******************************************************************************
 *
 *  The function write_n is a more robust method of writing a message to a
 *  socket.  It takes into account the fact that the write system call will
 *  not necessarily write the entire length of the buffer.  Therefore, it
 *  loops until the entire message has actually been written.
 *      Inputs:    Socket Descriptor
 *      Outputs:   It Returns the Number of Bytes Actually Written
 *
 ******************************************************************************/
int write_n(int fd)
{
    int nleft, nwritten=0;
    char *output_ptr = &Test_Array[0];

    nleft = TEST_ARRAY_SIZE;
    while(nleft > 0){
        nwritten = write(fd, output_ptr, nleft);
        nleft -= nwritten;
        output_ptr += nwritten;
    }
    return(TEST_ARRAY_SIZE - nleft);            /* return >= 0 */
}
```

```
/*****************************************************************************
 *                                                                           *
 *                                  Stephen West                             *
 *                                swest@cs.tamu.edu                          *
 *                          Wireless Configuration Program                   *
 *                                                                           *
 *     Directions:  This file should be compiled as follows:                 *
 *                                                                           *
 *                  > cc -o config_wireless config_wireless.c                *
 *                                                                           *
 *     Reference:      Internetworking with TCP/IP, Douglas E.               *
 *                     Comer David L. Stevens Vol. I,III                     *
 *                     UNIX Networking Programing, Stevens                   *
 *                                                                           *
 *     Affiliation:    Department of Computer Science                        *
 *                     Texas A&M University                                  *
 *                     College Station, TX 77843-3112                        *
 *                                                                           *
 *****************************************************************************/

#include <stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include <netdb.h>
#include <errno.h>
#include <sys/time.h>
#include <netinet/tcp.h>

/*
 * Removing this definition causes only those operations which may be
 * performed at the mobile host to be displayed on the main menu.
 */
#define BASE_STATION


/* --------------------------- Function Prototypes -------------------------- */

int ackp_enable_disable(int choice);
void error_enable_disable(int choice);
void error_model_disp_params(void);
int error_model_menu(void);
void error_set_burst_size(void);
void error_set_mean_rate(void);
void error_set_model_type(void);
void error_set_timer_gran(void);
void error_set_TRANS0(void);
void error_set_TRANS1(void);
int main_menu(void);
int snoop_enable_disable(int choice);
void sys_err(char *str);
void sys_msg(char *str);
int tcp_model_menu(void);


/*****************************************************************************
 *
 *  This program is used to set parameters at the base station and the mobile
 *  host.  It creates a dummy socket and then uses that socket to make
 *  getsockopt() and setsockopt() system calls.  These calls read and write
 *  values from/to global variables defined in the kernel which control the
 *  error injection model and tcp performance improvement options.
 *
 *****************************************************************************/

int Dummy_Fd;        /* The dummy socket required to get/set sock options */

main()
{
    int response=0, sub_menu_selection=0;
```

```
/*
 * Assign a dummy TCP socket so that we are able to use setsockopt() and
 * getsockopt() to change error model and tcp model parameters and then
 * verify that the parameter was correctly set.
 */

    if((Dummy_Fd = socket(AF_INET,SOCK_STREAM,0)) < 0)
        sys_err("\t Dummy Socket Call Failed");


/*
 * This is the main program loop.
 */

    while(response != 1){
        response = main_menu();
        switch(response){
            case 1:
                break;
            case 2:
                sub_menu_selection = error_model_menu();
                switch(sub_menu_selection){
                    case 1:
                        break;
                    case 2:
                        error_model_disp_params();
                        break;
                    case 3:
                        error_enable_disable(1);
                        break;
                    case 4:
                        error_enable_disable(0);
                        break;
                    case 5:
                        error_set_mean_rate();
                        break;
                    case 6:
                        error_set_model_type();
                        break;
                    case 7:
                        error_set_TRANS0();
                        break;
                    case 8:
                        error_set_TRANS1();
                        break;
                    case 9:
                        error_set_timer_gran();
                        break;
                    case 10:
                        error_set_burst_size();
                        break;
                    default:
                        sys_err("Invalid Sub-Menu Response Returned");
                        break;
                }
                break;
            case 3:
                sub_menu_selection = tcp_model_menu();
                switch(sub_menu_selection){
                    case 1:
                        break;
                    case 2:
                        snoop_enable_disable(1);
                        break;
                    case 3:
                        snoop_enable_disable(0);
                        break;
                    case 4:
                        ackp_enable_disable(1);
                        break;
                    case 5:
                        ackp_enable_disable(0);
                        break;
                    default:
                        sys_err("Invalid Sub-Menu Response Returned");
                        break;
                }
                break;
            default:
                sys_err("Invalid Main Menu Response Returned");
                break;
        }
    }
    system("clear");
    close(Dummy_Fd);

}


/*****************************************************************************
 *
 *  The function ackp_enable_disable looks at the parameter choice and if it
 *  is 1, the ackp code is enabled for incoming packets.  If choice is 0, the
 *  ackp code is disabled for incoming packets.  The option is called
 *  TCP_SNOOP_ACKP_ENABLE within the kernel and it sets the global variable
 *  snoop_ackp_enable defined in snoop.c
 *      Input:    User's Choice (Enabled/Disabled)
 *      Output:   Updated Kernel Value for Enabling/Disabling Ackp
 *
 *****************************************************************************/
int ackp_enable_disable(int choice)
{
    int ackp_enable;
    int option_len;

    if(choice == 1){
        ackp_enable = 1;
        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_ACKP_ENABLE,
                (char *) &ackp_enable, sizeof(ackp_enable)) < 0){
            sys_msg("\n\t Unable to Enable Ackp Option at Basestation!");
            return;
        }
        ackp_enable = 0;
        option_len = sizeof(ackp_enable);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_ACKP_ENABLE,
                (char *) &ackp_enable, &option_len) < 0){
            sys_msg("\n\t Unable to Get Ackp Enabled/Disabled Status!");
            return;
        }

        if(ackp_enable == 1)
            sys_msg("\n\t Ackp Option Has Been Enabled at Basestation!");
        else
            sys_msg("\n\t Error:  Ackp Was Not Turned On Within The Kernel");
    }
    else{
        ackp_enable = 0;
        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_ACKP_ENABLE,
                (char *) &ackp_enable, sizeof(ackp_enable)) < 0){
            sys_msg("\n\t Unable to Disable Ackp Option at Basestation!");
            return;
        }
        ackp_enable = 1;
        option_len = sizeof(ackp_enable);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_ACKP_ENABLE,
                (char *) &ackp_enable, &option_len) < 0){
            sys_msg("\n\t Unable to Get Ackp Enabled/Disabled Status!");
            return;
        }

        if(ackp_enable == 0)
```

```
            sys_msg("\n\t Ackp Option Has Been Disabled at Basestation!");
        else
            sys_msg("\n\t Error:  Ackp Was Not Turned Off Within The Kernel");
    }
}


/*****************************************************************************
 *
 *  The function error_enable_disable looks at the parameter choice and if it
 *  is 1, the bit error model is enabled for incoming packets on the wireless
 *  link.  If choice is 0, the bit error model is disabled for incoming packets
 *  on the wireless link.  The option is called TCP_SNOOP_BER_DISABLE within
 *  the kernel and it sets the global variable ber.disable defined in ber.c and
 *  ber.h
 *      Input:    User's Choice (Enabled/Disabled)
 *      Output:   Updated Kernel Value for Enabling/Disabling Error Injection
 *
 *****************************************************************************/
void error_enable_disable(int choice)
{
    int error_disable;
    int option_len;

    if(choice == 1){
        error_disable = 0;
        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_BER_DISABLE,
                (char *) &error_disable, sizeof(error_disable)) < 0){
            sys_msg("\n\t Unable to Set Error Model to The On State!");
            return;
        }
        error_disable = 1;
        option_len = sizeof(error_disable);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_BER_DISABLE,
                (char *) &error_disable, &option_len) < 0){
            sys_msg("\n\t Unable to Get Error Model Enabled/Disabled Status!");
            return;
        }

        if(error_disable == 1)
            sys_msg("\n\t Set Option Error - Error Model Was Not Turned On");
        else
            sys_msg("\n\t Error Model Has Been Enabled!");
    }
    else{
        error_disable = 1;
        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_BER_DISABLE,
                (char *) &error_disable, sizeof(error_disable)) < 0){
            sys_msg("\n\t Unable to Set Error Model to The Off State!");
            return;
        }
        error_disable = 0;
        option_len = sizeof(error_disable);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_BER_DISABLE,
                (char *) &error_disable, &option_len) < 0){
            sys_msg("\n\t Unable to Get Error Model Enabled/Disabled Status!");
            return;
        }

        if(error_disable == 0)
            sys_msg("\n\t Set Option Error - Error Model Was Not Turned Off");
        else
            sys_msg("\n\t Error Model Has Been Disabled!");
    }
}


/*****************************************************************************
 *
 *  The function error_model_disp_params creates a display screen showing the
 *  current settings for each of the bit error model parameters.  It is a read
 *  only screen.
 *      Input:    None
 *      Output:   Prints Values of All Kernel Bit Error Models to Screen
 *
 ******************************************************************************/
void error_model_disp_params(void)
{
    int err_option, option_len;

    system("clear");
    printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
    printf("\t\t   Current Error Model Values \n\n\n");

    option_len = sizeof(err_option);
    if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_ERRPROB0,(char *)
            &err_option, &option_len) < 0)
        sys_msg("\n\t Unable to Get ERRPROB0!\n");
    printf("\t ERRPROB0 \t = \t %d\n", err_option);

    option_len = sizeof(err_option);
    if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_ERRPROB1,(char *)
            &err_option, &option_len) < 0)
        sys_msg("\n\t Unable to Get ERRPROB1!\n");
    printf("\t ERRPROB1 \t = \t %d\n", err_option);

    option_len = sizeof(err_option);
    if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_BER_MODEL,(char *)
            &err_option, &option_len) < 0)
        sys_msg("\n\t Unable to Get BER_MODEL!\n");
    if(err_option == 0)
        printf("\t BER_MODEL \t = \t Poisson\n");
    else
        printf("\t BER_MODEL \t = \t Markov\n");

    option_len = sizeof(err_option);
    if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_TRANS0,(char *)
            &err_option, &option_len) < 0)
        sys_msg("\n\t Unable to Get TRANS0!\n");
    printf("\t TRANS0 \t = \t %d\n", err_option);

    option_len = sizeof(err_option);
    if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_TRANS1,(char *)
            &err_option, &option_len) < 0)
        sys_msg("\n\t Unable to Get TRANS1!\n");
    printf("\t TRANS1 \t = \t %d\n", err_option);

    option_len = sizeof(err_option);
    if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_TIMERGRAN,(char *)
            &err_option, &option_len) < 0)
        sys_msg("\n\t Unable to Get TIMERGRAN!\n");
    printf("\t TIMERGRAN \t = \t %d\n", err_option);

    option_len = sizeof(err_option);
    if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_BURSTRATE,(char *)
            &err_option, &option_len) < 0)
        sys_msg("\n\t Unable to Get BURSTRATE!\n");
    printf("\t BURSTRATE \t = \t %d\n", err_option);

    option_len = sizeof(err_option);
    if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_BER_DISABLE,(char *)
            &err_option, &option_len) < 0)
        sys_msg("\n\t Unable to Get BER_DISABLE!\n");
    if(err_option == 0)
        printf("\t BER_DISABLE \t = \t False (Injecting Errors)\n");
    else
        printf("\t BER_DISABLE \t = \t True (Not Injecting Errors)\n");

    sys_msg("");
}
```

```
/******************************************************************************
 *
 *  The function error_model_menu creates a menu screen which allows the user
 *  to choose an error model parameter to change.  The user may also choose to
 *  view all of the current settings or to exit without doing anything.
 *      Input:    None
 *      Output:   The Menu Option Selected By The User
 *
 ******************************************************************************/
int error_model_menu(void)
{
    int response=0;

    while((response < 1) || (response > 10)){
        system("clear");
        printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
        printf("\t\t        Error Model Menu \n\n\n");
        printf("\t 1)  Exit to Main Menu\n");
        printf("\t 2)  Check Status of Bit Error Parameters \n");
        printf("\t 3)  Enable Error Injection \n");
        printf("\t 4)  Disable Error Injection \n");
        printf("\t 5)  Change Mean Bit Error Rate (ERRPROB0) \n");
        printf("\t 6)  Change Bit Error Model (Markov or Poisson) \n");
        printf("\t 7)  Change Markov TRANS0 \n");
        printf("\t 8)  Change Markov TRANS1 \n");
        printf("\t 9)  Change Error Model Timer Granularity \n");
        printf("\t 10) Change Error Burst Size \n");
        printf("\n\n\t Enter Selection (1 - 10) --> ");

        scanf("%d", &response);
    }

    return response;
}


/******************************************************************************
 *
 *  The function error_set_burst_size allows the user to choose the number of
 *  packets in a row which will be injected with errors.  The default is a
 *  single packet.  The option is called TCP_SNOOP_BURSTRATE within the kernel,
 *  and it sets the global variable ber.burst_rate defined in ber.c and ber.h.
 *      Input:    User Entered Burst Size
 *      Output:   Updated Kernel Value for Burst Size
 *
 ******************************************************************************/
void error_set_burst_size(void)
{
    int response=0;
    int err_option=0;
    int option_len;

    while((response < 1) || (response > 2)){
        system("clear");
        printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
        printf("\t      Set Error Model's Packet Burst Size \n");
        printf("\t\t Default Burst Size is 1 packet \n\n\n");

        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_BURSTRATE,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Get Burst Size! \n");
            return;
        }
        printf("\t Current Burst Size = %d packets \n\n", err_option);

        printf("\t 1) Exit to Main Menu \n");
        printf("\t 2) Change Burst Size \n");
        printf("\n\n\t Enter Selection (1 - 2) --> ");

        scanf("%d", &response);
    }

    if(response == 1)
        return;
    else{
        printf("\n\n\t Enter New Burst Size --> ");
        scanf("%d", &err_option);

        if(err_option < 1){
            sys_msg("\n\t Burst Size Must Be Greater Than or Equal to 1");
            return;
        }

        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_BURSTRATE,
                (char *) &err_option, sizeof(err_option)) < 0){
            sys_msg("\n\t Unable to Change Burst Size!");
            return;
        }

        err_option = 0;
        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_BURSTRATE,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Read New Burst Size!\n");
            return;
        }
        if(err_option == 0)
            sys_msg("\n\t Set Option Error - Burst Size Was Not Changed");
        else{
            printf("\t New Burst Size = %d packets\n\n", err_option);
            sys_msg("");
        }
    }
}


/******************************************************************************
 *
 *  The function error_set_mean_rate allows the user to choose the mean number
 *  of bytes between errors.  The default value of 65,536 means that one out
 *  every 65,536 bytes will have an error injected or roughtly one out of
 *  every 45 packets.  The option is called TCP_SNOOP_ERRPROB1 within the
 *  kernel, and it sets the global variable ber.error_prob[1] defined in ber.c
 *  and ber.h.
 *      Input:    User Entered Mean Error Rate
 *      Output:   Updated Kernel Value for Mean Error Rate
 *
 ******************************************************************************/
void error_set_mean_rate(void)
{
    int response=0;
    int err_option=0;
    int option_len;

    while((response < 1) || (response > 2)){
        system("clear");
        printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
        printf("\t       Set Error Model's Mean Error Rate \n");
        printf("\t       Default Rate is 1 Per 65536 Bytes \n\n\n");

        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_ERRPROB0,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Get Mean Error Rate (ERRPROB0)!\n");
            return;
        }
        printf("\t Current Mean Error Rate = 1 error per %d bytes\n\n",
                err_option);

        printf("\t 1) Exit to Main Menu\n");
        printf("\t 2) Change Mean Error Rate\n");
        printf("\n\n\t Enter Selection (1 - 2) --> ");

        scanf("%d", &response);
    }

    if(response == 1)
        return;
    else{
        printf("\n\n\t Enter New Mean Error Rate --> ");
        scanf("%d", &err_option);

        if(err_option < 1){
            sys_msg("\n\t Mean Error Rate Must Be Greater Than or Equal to 1");
            return;
        }

        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_ERRPROB0,
                (char *) &err_option, sizeof(err_option)) < 0){
            sys_msg("\n\t Unable to Change Mean Error Rate!");
            return;
        }

        err_option = 0;
        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_ERRPROB0,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Read New Mean Error Rate!\n");
            return;
        }

        if(err_option == 0)
            sys_msg("\n\t Set Option Error - Mean Error Rate Was Not Changed");
        else{
            printf("\t New Mean Error Rate = 1 error per %d bytes\n\n",
                    err_option);
            sys_msg("");
        }
    }
}


/*****************************************************************************
 *
 *  The function error_set_model allows the user to choose between a straight
 *  Poisson error model or a Markov error model with both good and bad states.
 *  The default is a Poisson error model.  The option is called
 *  TCP_SNOOP_BER_MODEL within the kernel, and it sets the global variable
 *  ber.model defined in ber.c and ber.h.
 *      Input:    User Entered Model Type
 *      Output:   Updated Kernel Value for Model Type
 *
 *****************************************************************************/
void error_set_model_type(void)
{
    int response=0;
    int err_option=0;
    int option_len;

    while((response < 1) || (response > 3)){
        system("clear");
        printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
        printf("\t\t Choose The Type of Error Model \n");
        printf("\t\t    Default Model is Poisson \n\n\n");

        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_BER_MODEL,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Get Model Type! \n");
            return;
        }
        if(err_option == 0)
            printf("\t Current Model Type = Poisson \n\n");
        else
            printf("\t Current Model Type = Markov \n\n");

        printf("\t 1) Exit to Main Menu \n");
        printf("\t 2) Set Model Type to Poisson \n");
        printf("\t 3) Set Model Type to Markov \n");
        printf("\n\n\t Enter Selection (1 - 3) --> ");

        scanf("%d", &response);
    }

    if(response == 1)
        return;
    else if(response == 2){
        err_option = 0;
        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_BER_MODEL,
                (char *) &err_option, sizeof(err_option)) < 0){
            sys_msg("\n\t Unable to Change Model Type!");
            return;
        }

        err_option = 1;
        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_ERRPROB0,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Read New Model Type!\n");
            return;
        }
        if(err_option == 1)
            sys_msg("\n\t Set Option Error - Model Type Not Set to Poisson");
        else
            sys_msg("\n\t New Model Type = Poisson");
    }
    else{
        err_option = 1;
        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_BER_MODEL,
                (char *) &err_option, sizeof(err_option)) < 0){
            sys_msg("\n\t Unable to Change Model Type!");
            return;
        }

        err_option = 0;
        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_ERRPROB0,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Read New Model Type!\n");
            return;
        }
        if(err_option == 0)
            sys_msg("\n\t Set Option Error - Model Type Not Set to Markov");
        else
            sys_msg("\n\t New Model Type = Markov");
    }
}


/******************************************************************************
 *
 *  The function error_set_timer_gran allows the user to choose the timer
 *  granularity used by the error model in microseconds.  The default is
 *  100,000 microseconds or 100 milliseconds.  This parameter is used only by
 *  the Markov model.  The option is called TCP_SNOOP_TIMERGRAN within the
 *  kernel, and it sets the global variable ber.time_granularity defined in
 *  ber.c and ber.h.
 *      Input:    User Entered Timer Granularity
 *      Output:   Updated Kernel Value for Timer Granularity
 *
 ******************************************************************************/
void error_set_timer_gran(void)
```

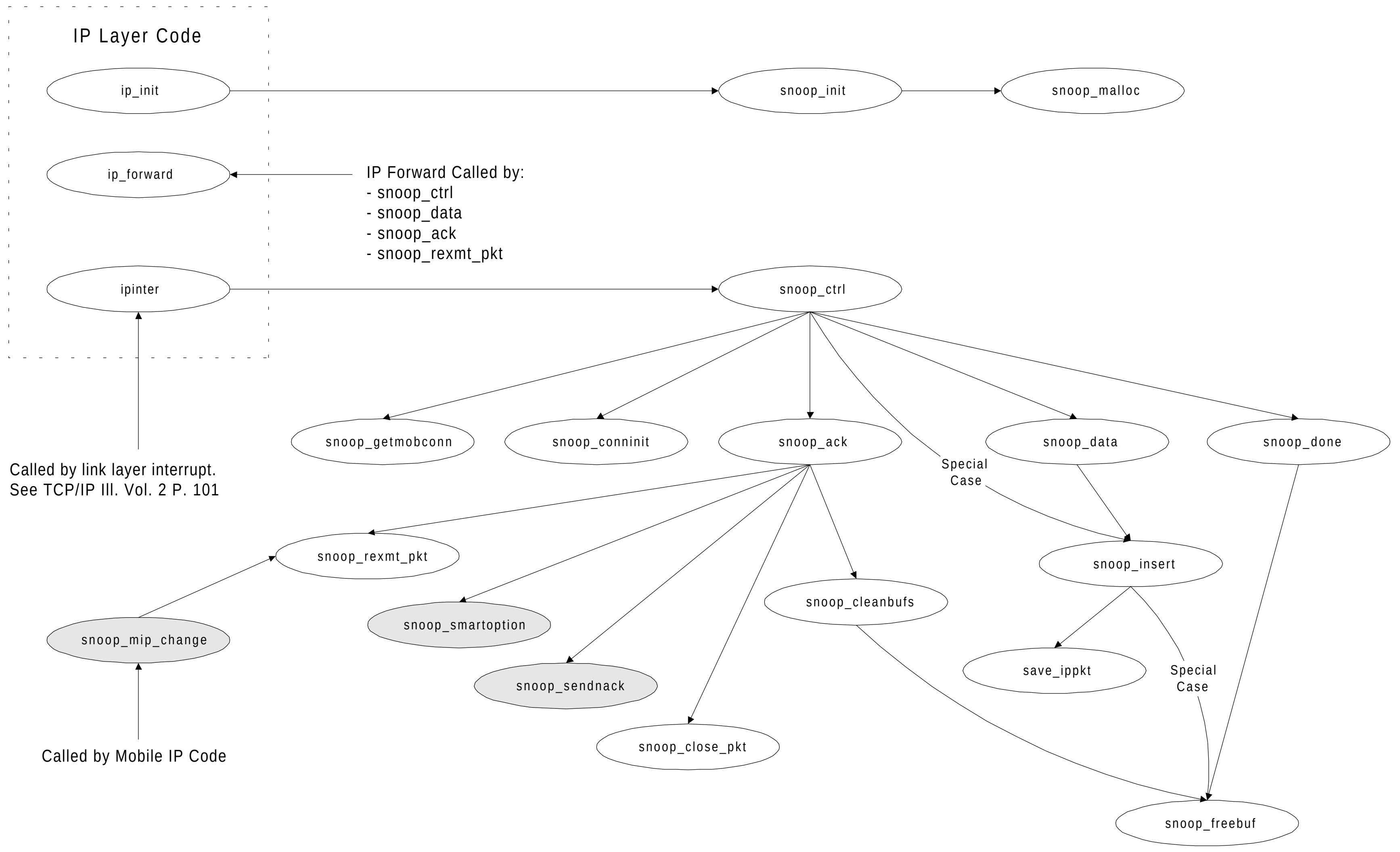
```
{
    int response=0;
    int err_option=0;
    int option_len;

    while((response < 1) || (response > 2)){
        system("clear");
        printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
        printf("\t      Set Markov Model's Timer Granularity \n");
        printf("\t      Default Gran. is 100000 microseconds \n\n\n");

        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_TIMERGRAN,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Get Mean Timer Granularity! \n");
            return;
        }
        printf("\t Current Timer Granularity = %d microseconds \n\n",
                err_option);

        printf("\t 1) Exit to Main Menu\n");
        printf("\t 2) Change Timer Granularity\n");
        printf("\n\n\t Enter Selection (1 - 2) --> ");

        scanf("%d", &response);
    }

    if(response == 1)
        return;
    else{
        printf("\n\n\t Enter New Timer Granularity --> ");
        scanf("%d", &err_option);

        if(err_option < 1){
            sys_msg("\n\t Timer Gran. Must Be Greater Than or Equal to 1");
            return;
        }

        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_TIMERGRAN,
                (char *) &err_option, sizeof(err_option)) < 0){
            sys_msg("\n\t Unable to Change Timer Granularity!");
            return;
        }

        err_option = 0;
        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_TIMERGRAN,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Read New Timer Granularity!\n");
            return;
        }

        if(err_option == 0)
            sys_msg("\n\t Set Option Error - Timer Gran. Was Not Changed");
        else{
            printf("\t New Timer Granularity = %d microseconds \n\n",
                    err_option);
            sys_msg("");
        }
    }
}


/*****************************************************************************
 *
 *  The function error_set_TRANS0 allows the user to set the Markov model's
 *  TRANS0 parameter.  The default is 30.  The option is called
 *  TCP_SNOOP_TRANS0 within the kernel, and it sets the global variable
 *  ber.trans_perc[0] defined in ber.c and ber.h.
 *      Input:    User Entered TRANS0
 *      Output:   Updated Kernel Value for TRANS0
 *
 ******************************************************************************/
void error_set_TRANS0(void)
{
    int response=0;
    int err_option=0;
    int option_len;

    while((response < 1) || (response > 2)){
        system("clear");
        printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
        printf("\t\t Set Markov Model's TRANS0 Value \n");
        printf("\t\t      Default Value is 30 \n\n\n");

        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_TRANS0,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to TRANS0! \n");
            return;
        }
        printf("\t Current TRANS0 Value = %d \n\n", err_option);

        printf("\t 1) Exit to Main Menu\n");
        printf("\t 2) Change TRANS0\n");
        printf("\n\n\t Enter Selection (1 - 2) --> ");

        scanf("%d", &response);
    }

    if(response == 1)
        return;
    else{
        printf("\n\n\t Enter New TRANS0 Value --> ");
        scanf("%d", &err_option);

        if(err_option < 1){
            sys_msg("\n\t TRANS0 Must Be Greater Than or Equal to 1");
            return;
        }

        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_TRANS0,
                (char *) &err_option, sizeof(err_option)) < 0){
            sys_msg("\n\t Unable to Change TRANS0 Value!");
            return;
        }

        err_option = 0;
        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_TRANS0,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Read New TRANS0 Value!\n");
            return;
        }

        if(err_option == 0)
            sys_msg("\n\t Set Option Error - TRANS0 Value Was Not Changed");
        else{
            printf("\t New TRANS0 Value = %d \n\n", err_option);
            sys_msg("");
        }
    }
}


/*******************************************************************************
 *
 *  The function error_set_TRANS1 allows the user to set the Markov model's
 *  TRANS1 parameter.  The default is 30.  The option is called
 *  TCP_SNOOP_TRANS1 within the kernel, and it sets the global variable
 *  ber.trans_perc[1] defined in ber.c and ber.h.
 *      Input:    User Entered TRANS1
 *      Output:   Updated Kernel Value for TRANS1
 *
 ******************************************************************************/
void error_set_TRANS1(void)
{
    int response=0;
    int err_option=0;
    int option_len;

    while((response < 1) || (response > 2)){
        system("clear");
        printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
        printf("\t\t Set Markov Model's TRANS1 Value \n");
        printf("\t\t      Default Value is 70 \n\n\n");

        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_TRANS1,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to TRANS1! \n");
            return;
        }
        printf("\t Current TRANS1 Value = %d \n\n", err_option);

        printf("\t 1) Exit to Main Menu\n");
        printf("\t 2) Change TRANS1\n");
        printf("\n\n\t Enter Selection (1 - 2) --> ");

        scanf("%d", &response);
    }

    if(response == 1)
        return;
    else{
        printf("\n\n\t Enter New TRANS1 Value --> ");
        scanf("%d", &err_option);

        if(err_option < 1){
            sys_msg("\n\t TRANS1 Must Be Greater Than or Equal to 1");
            return;
        }

        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_TRANS1,
                (char *) &err_option, sizeof(err_option)) < 0){
            sys_msg("\n\t Unable to Change TRANS1 Value!");
            return;
        }

        err_option = 0;
        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_TRANS1,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Read New TRANS1 Value!\n");
            return;
        }

        if(err_option == 0)
            sys_msg("\n\t Set Option Error - TRANS1 Value Was Not Changed");
        else{
            printf("\t New TRANS1 Value = %d \n\n", err_option);
            sys_msg("");
        }
    }
}


/******************************************************************************
 *
 *  The function main_menu displays the main menu for the wireless
 *  configuration program
 *      Input:    User Selections
 *      Output:   Menu Number Related To The User's Selection
 ****************************************************************************/
int main_menu(void)
{

    int response=0;

#ifdef BASE_STATION
    while((response < 1) || (response > 3)){
#else
    while((response < 1) || (response > 2)){
#endif
        system("clear");
        printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
        printf("\t\t\t   Main Menu \n\n\n");
        printf("\t 1) Exit the Program\n");
        printf("\t 2) Modify Error Model\n");
#ifdef BASE_STATION
        printf("\t 3) Set/Check TCP Enhancement Options\n");
        printf("\n\n\t Enter Selection (1 - 3) --> ");
#else
        printf("\n\n\t Enter Selection (1 - 2) --> ");
#endif

        scanf("%d", &response);
    }

    return response;
}


/****************************************************************************
 *
 *  The function snoop_enable_disable looks at the parameter choice and if it
 *  is 1, the snoop code is enabled for packets traversing the wireless link.
 *  If choice is 0, the snoop code is disabled for packets traversing the
 *  wireless link.  The option is called TCP_SNOOP_DISABLE within the
 *  kernel and it sets the global variable snoopstate->disable defined in
 *  snoop.c and snoop.h
 *      Input:    User's Choice (Enabled/Disabled)
 *      Output:   Updated Kernel Value for Enabling/Disabling Snoop
 *
 ****************************************************************************/
int snoop_enable_disable(int choice)
{
    int snoop_disable;
    int option_len;

    if(choice == 1){
        snoop_disable = 0;
        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_DISABLE,
                (char *) &snoop_disable, sizeof(snoop_disable)) < 0){
            sys_msg("\n\t Unable to Enable Snoop Option at Basestation!");
            return;
        }
        snoop_disable = 1;
        option_len = sizeof(snoop_disable);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_DISABLE,
                (char *) &snoop_disable, &option_len) < 0){
            sys_msg("\n\t Unable to Get Snoop Enabled/Disabled Status!");
            return;
        }

        if(snoop_disable == 1)
            sys_msg("\n\t Error:  Snoop Was Not Turned On Within The Kernel");
        else
            sys_msg("\n\t Snoop Option Has Been Enabled at Basestation!");
    }
    else{
        snoop_disable = 1;
        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_DISABLE,
                (char *) &snoop_disable, sizeof(snoop_disable)) < 0){
```

```
            sys_msg("\n\t Unable to Disable Snoop Option at Basestation!");
            return;
        }
        snoop_disable = 0;
        option_len = sizeof(snoop_disable);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_DISABLE,
                (char *) &snoop_disable, &option_len) < 0){
            sys_msg("\n\t Unable to Get Snoop Enabled/Disabled Status!");
            return;
        }

        if(snoop_disable == 0)
            sys_msg("\n\t Error:  Snoop Was Not Turned Off Within The Kernel");
        else
            sys_msg("\n\t Snoop Option Has Been Disabled at Basestation!");
    }
}


/******************************************************************************
 *
 *  The function sys_err takes a string and displays it for the user.
 *  Then it terminates the program with the error.  It is intended to be used
 *  with fatal errors which must terminate the program, not for program
 *  warnings.
 *      Inputs:    Error Message
 *      Outputs:   Message to The Screen
 *
 ******************************************************************************/
void sys_err(char *str)
{
    printf("%s\n",str);
    exit(1);
}


/******************************************************************************
 *
 *  The function sys_msg takes a string and displays it for the user.  It then
 *  pauses until the user presses the return key.  Unlike sys_err, it is not
 *  intended to be used with fatal errors.  It displays non-fatal errors and
 *  other general information.
 *      Inputs:    Error Message
 *      Outputs:   Message to The Screen
 *
 ******************************************************************************/
void sys_msg(char *str)
{
    int response;

    printf("%s\n",str);
    printf("\t Press the return key to continue\n");
    response = getchar();
    response = getchar();
}


/******************************************************************************
 *
 *  The function tcp_model_menu allows the user to choose what enhancements if
 *  any to add to the tcp/ip protocol suite.
 *      Input:    User's Choice
 *      Output:   Menu Item Number Selected By The User
 *
 ******************************************************************************/
int tcp_model_menu(void)
{

    int response=0;
    int snoop_disable;
    int ackp_enable;
    int option_len;

    while((response < 1) || (response > 5)){
        system("clear");
        printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
        printf("\t\t      Enhancement Options \n\n\n");

        snoop_disable=0;
        option_len = sizeof(snoop_disable);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_DISABLE,(char *)
                &snoop_disable, &option_len) < 0)
            sys_msg("\n\t Unable to Get Snoop Enabled/Disabled Status!\n");

        if(snoop_disable == 0)
            printf("\t Snoop Is Currently Enabled at The Basestation\n\n");
        else
            printf("\t Snoop Is Currently Disabled at The Basestation\n\n");

        ackp_enable=0;
        option_len = sizeof(ackp_enable);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_ACKP_ENABLE,(char *)
                &ackp_enable, &option_len) < 0)
            sys_msg("\n\t Unable to Get Ackp Enabled/Disabled Status!\n");

        if(ackp_enable == 0)
            printf("\t Ackp Is Currently Disabled at The Basestation\n\n");
        else
            printf("\t Ackp Is Currently Enabled at The Basestation\n\n");

        printf("\t Note:  Snoop must be enabled for Ackp to work\n\n");

        printf("\t 1) Exit to Main Menu\n");
        printf("\t 2) Enable Snoop Option at Basestation\n");
        printf("\t 3) Disable Snoop Option at Basestation\n");
        printf("\t 4) Enable Ackp Option at Basestation\n");
        printf("\t 5) Disable Ackp Option at Basestation\n");
        printf("\n\n\t Enter Selection (1 - 5) --> ");

        scanf("%d", &response);
    }

    return response;
}
```

Appendix C

FreeBSD Snoop Upgrades
FreeBSD Error Model Upgrades

Main Call Tree for Berkeley Snoop Coding



Misc. Calls for Berkeley Snoop Coding

snoop_data - sets timeout for new data packet if none pending

snoop_persist - sends persist packet and sets timeout for packet if none pending

snoop_cleanbufs, snoop_mip_change clear all retransmit timers and then set one new one if the buffer isn't empty

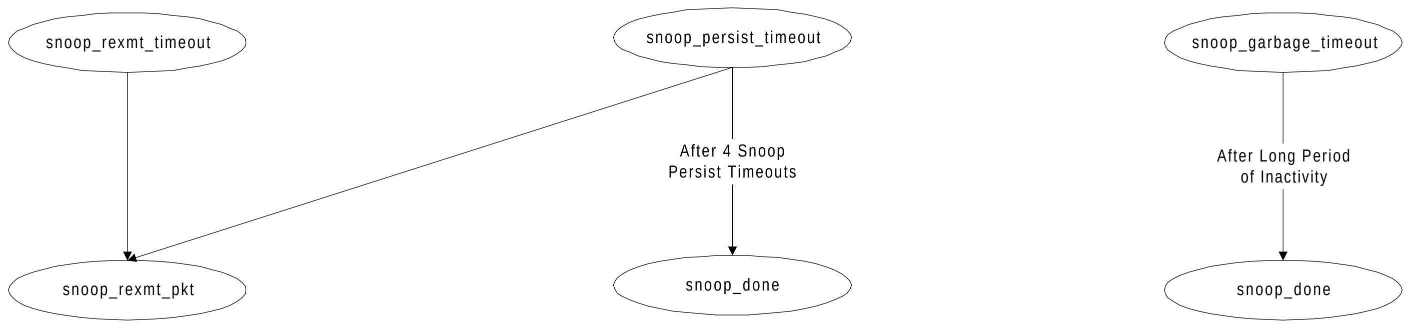
snoop_done - clears all retransmit timers

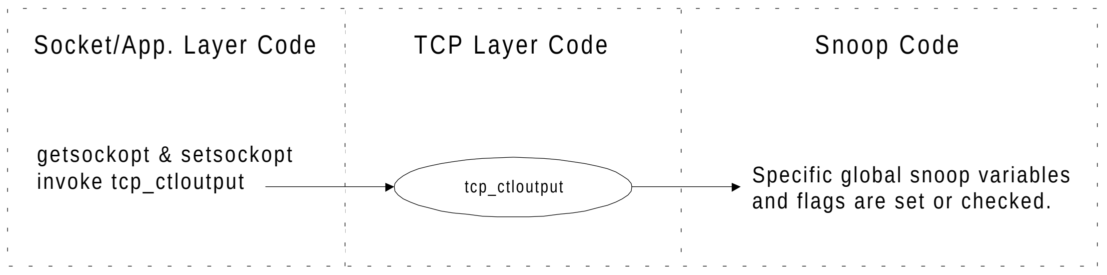
<u>Set (timeout call) by:</u>
- snoop_ctrl
- snoop_mipchange
- internally (inside snoop_persist_timeout)

<u>Cleared (untimeout call) by:</u>
- snoop_done
-snoop_mip_change

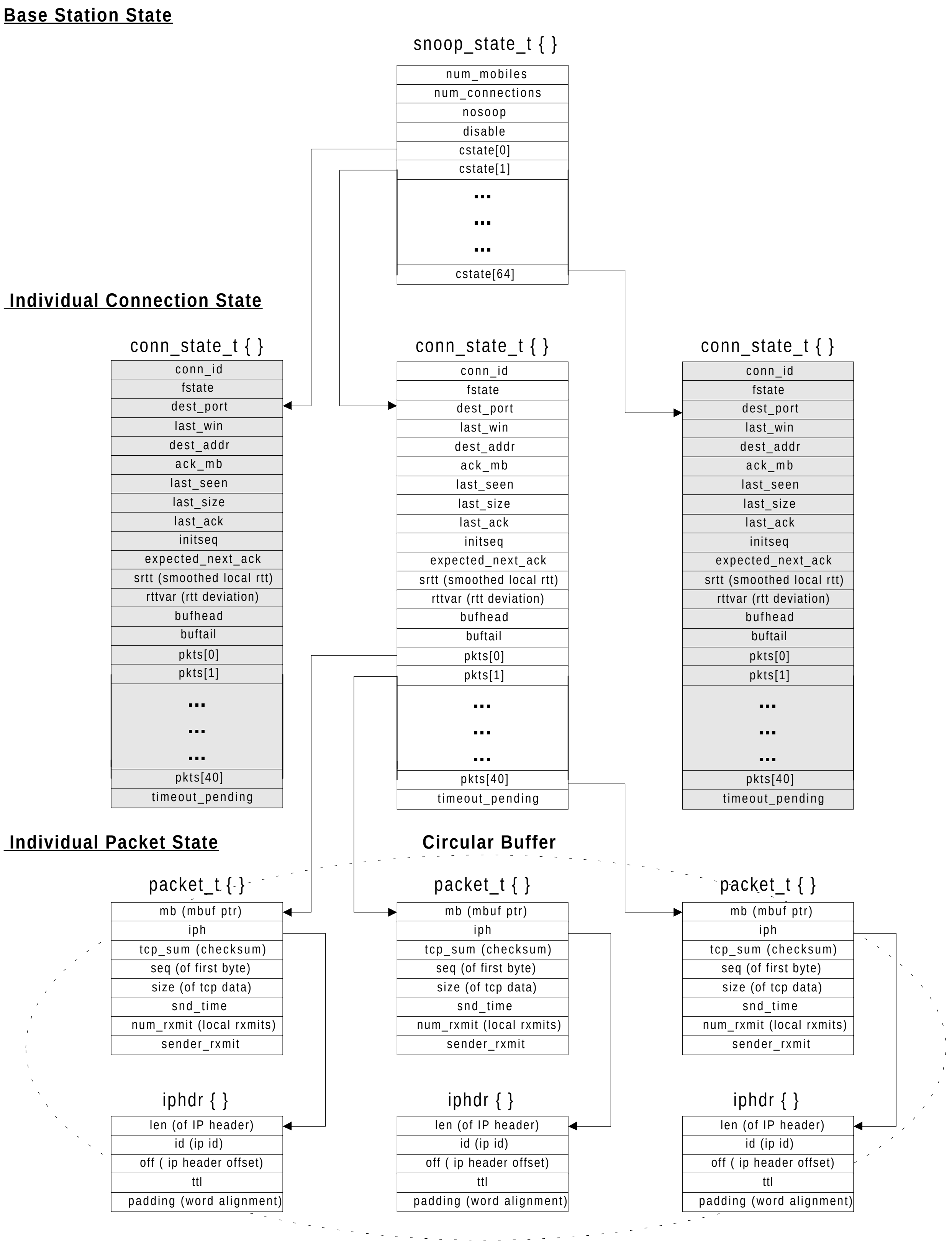
<u>Set (timeout call) by:</u>
- snoop_ctrl
- internally (inside snoop_garbage_timeout)

<u>Cleared (untimeout call) by:</u>
- Never cleared by untimeout
- SNOOP_ALIVE flag set in snoop_ctrl on each packet received




Socket/App. Layer Code
TCP Layer Code
Snoop Code
getsockopt & setsockopt invoke tcp_ctloutput
tcp_ctloutput
Specific global snoop variables and flags are set or checked.

Berkeley Snoop Data Structures



FreeBSD Code Modifications for Snoop & Error Model

Option Flag Settings in /i386/conf/ MYKERNEL

- The code used on any of the FreeBSD machines should be the same. Setting the flags as shown below allows the same version of code to function differently depending on the type of machine (i.e. fixed host, mobile host, base station). It is the option flags set within the file MYKERNEL which determines what code will be included when the kernel is compiled. MOBILE defines a machine as having a wireless interface. BASESTATION defines a machine as being capable of forwarding packets between wired and wireless links. NOMIP defines a machine as having Mobile-IP disabled. BER_EXPT defines a machine as having the bit error injection code enabled.

- The following option flags should be set in order to enable snoop at a base station:
  MOBILE
  BASESTATION
  NOMIP

- The following option flags should be set at a mobile host:
  MOBILE
  NOMIP

- The following flag should be set in order to enable bit error injection at a wireless receiver (i.e. either a base station or a mobile host):
  BER_EXPT

File Additions in /i386/conf/ files.i386

- The files netinet/snoop.c, netinet/ber.c, and netinet/rand.c should be added to the file files.i386 so that these new source code files will be compiled when the kernel is rebuilt. The FreeBSD format is slightly different than the BSDI format (as shown by Hari in the files.i386.add file on UC Berkley's daedalus ftp site). The new files should be added as follows:

  ```
  netinet/snoop.c     optional    basestation
  netinet/ber.c       optional    mobile
  netinet/rand.c      optional    mobile
  ```

  Note: The third value on each line above corresponds to the option flag added in the file MYKERNEL (i.e. basestation goes with BASESTATION, and mobile goes with MOBILE)

Note: On the base station (verdi) the files have been included as standard. This is fine as long as the files will always be there. If the files should be included based on whether the flag is defined in MYKERNEL (the third parameter above) then optional should be used in the second column above rather than standard.

SNOOP mbuf Allocation

- Snoop allocates a new type of mbuf called M_SNOOP. This requires a change in malloc.h within the /usr/src/sys/sys/ directory. The type was defined as 79 and M_LAST was moved from 79 to 80 (M_LAST must always be 1 greater than the last type defined). These definitions are used as indices into an array by kern_malloc.c which is in the /usr/src/sys/kern/ directory and contains the kernel's malloc() and free() functions.

- A constant called INITKMEMNAMES is also defined in malloc.h. Within this constant a list of strings is given to match each of the types defined. For snoop the string "snoop" was added in position 79. This large constant is used to initialize an array of names within kern_malloc.c

- The mbuf.h file within the /usr/src/sys/sys/ directory was not modified during this process.

snoop.c

- This file temporarily has the definition of SNOOP_IPTOS commented out so that the fastq option will not be used. The WaveLAN driver does not currently have a fastq and the if_ethersubr.c code does not attempt to queue anything on a fastq.

- All of the #ifdef DEBUG statements were changed to #ifdef SNOOP_DEBUG since the kernel also uses #ifdef DEBUG statements and we wanted to be sure that all the information printed while in debugging mode is snoop specific.

snoop.h

- The prototype for snoop_close_pkt was added to the snoop.h file. This was to ensure proper snoop operation and to get rid of a compiler warning message. The Berkeley code has prototypes for most of the functions in snoop.h, but some such as save_ippkt which is called only by snoop_insert doesn't have a prototype since it is defined in snoop.c before snoop_insert.

ip_input.c

- The only modifications that appear to be snoop related rather than merely differences between BSDI and FreeBSD occur in the functions ip_init() and ipintr().

- The function ip_init() calls snoop_init() when the machine is first booted to initialize the snoop data structures. It also calls ber_init() if errors are going to be injected.

- The function ipintr() is called when a new packet arrives as input. If errors are being injected and the packet arrived on the wireless link a call is made to wireless_drop() which is the main gateway into the Berkeley error code. Immediately after this the IP checksum is verified. If the IP header has been damaged, the checksum calculation will fail and the packet will be dropped without further processing. If the IP data portion (i.e. the TCP header and application layer data) has been damaged, this will be detected by the TCP checksum later.

  Provided the packet's checksum is valid, a check is made to determine if the packet has reached its final destination or should be forwarded. The plain version of ipintr() without snoop running will call ip_forward() if the packet has not reached its final destination and the machine has ipforwarding (i.e. routing) enabled. When snoop is running, snoop_ctrl() is called instead. After doing the necessary buffering of the packet, the snoop code then calls ip_forward().

- Specific code fragments which should be added by line number:

Lines 60-62	#include snoop.h
Lines 66-68	#include ber.h
Line 85	Defines extern struct ber_state ber - Grouped with lines 66-68 so only included if BER_EXPT defined.
Lines 161-166	Calls to snoop_init() and ber_init() inside ip_init()
Lines 233-248	Inject errors for wireless packets received by calling wireless_drop()
Lines 387-441	Forward packets using snoop_ctrl()

tcp_usrreq.c

- All of the modifications made in tcp_usrreq.c occur within the function tcp_ctloutput(). This function is called as a result of setsockopt() and getsockopt() system calls. There are two major snoop additions to this function. The first section allows someone at the application level to set snoop and error model parameters (enabling and disabling snoop, enabling and disabling error generation, etc.) by setting socket options. The second section allows someone at the application level to view the current setting of parameters. This is done by setting and returning the values of global variables in tcp_usrreq.c which are used in snoop.c, and ber.c. Note that other TCP improvement methods such as ELN, SMART, etc. have also added global variables which are set and checked within tcp_usrreq.c. These options are added merely by placing additional cases into the switch statement already in the code. The only options which have been added to the TAMU code are related to bit error model parameters, basic snoop enabling/disabling, and snoop options such as snoop_linkemu, snoop_eln, snoop_rexmt, and snoop_smart (i.e. SACK, etc. have not been included in the TAMU code). For our initial testing purposes, all of the snoop options should be set to zero. This will enable snoop and local retransmission and it will disable LINKEMU, ELN, and SMART.

- All of the global constants set within tcp_usrreq.c and used by snoop.c are defined right before tcp_ctloutput().

- Specific code fragments which should be added by line number:

Lines 66, 68-71	Include header files for snoop and bit error model.
Lines 353-359	Global ber and snoopstate variables used by snoop
Lines 360-371	Snoop options not initially used
Lines 372-375	Global variables for non-snoop TCP improvements
Lines 376-379	Snoop options not initially used
Lines 380-399	Global variables for non-snoop TCP improvements
Lines 451-484	Set bit error model parameters
Lines 485-491	Set snoop to enabled/disabled
Lines 492-512	Set snoop options not initially used
Lines 513-519	Set options for non-snoop TCP improvements
Lines 520-525	Set snoop options not initially used
Lines 526-595	Set options for non-snoop TCP improvements
Lines 596-611	Set statistics collection to enabled/disabled
Lines 612-634	Set options for other TCP improvements
Lines 655-679	Get bit error model parameters
Lines 680-685	Get snoop status (enabled/disabled)
Lines 686-703	Get snoop status for options not initially used
Lines 704-709	Get option status for non-snoop TCP improvements
Lines 710-715	Get snoop status for options not initially used
Lines 716-751	Get option status for non-snoop TCP improvements
Lines 752-756	Get statistics collection status (enabled/disabled)
Lines 757-768	Get round trip timing values (i.e. srtt, rtt)
Lines 769-788	Get option status for other TCP improvements

  The matching #endif which goes with each of the #ifdef MOBILE must be included. These three #endif statements occur on lines 400, 540, and 727.

<u>tcp.h</u>

- Note: all of the header files accessed by user level code are also in the directory /usr/include/. Therefore, tcp.h has to be modified there also if application programs are to recognize the additional socket options.

- The changes within tcp.h are directly related to the additional socket options used within tcp_usrreq.c. They define the constant values which will be used for the additional branches within the case statements for the get and set options.

- Line 163 should be moved up so that its definition is with the other snoop error model parameters.

- Other than the bit error model, we need TCP_SNOOP_DISABLE, and we may also want TCP_SNOOP_LINKEMU_ENABLE, TCP_SNOOP_ELN_ENABLE, and TCP_SNOOP_REXMT_DISABLE. Also, if we want round trip time information, we should add TCP_GET_SRTT and TCP_GET_RTTVAR

- Specific code fragments which should be added by line number in snoop files:

Lines 141-148	Define constants for ber case statements
Line 163	Define const BURSTRATE for ber case statement
Line 149	Define const TCP_SNOOP_DISABLE
Line 157	Define const TCP_SNOOP_LINKEMU_ENABLE
Line 158	Define const TCP_SNOOP_ELN _ENABLE
Line 159	Define const TCP_SNOOP_REXMT _DISABLE
Line 160	Define const TCP_SNOOP_SMART _ENABLE
Line 165	Define const TCP_GET_SRTT
Line 166	Define const TCP_GET_RTTVAR

ber.c

- The only change that has been made to ber.c is that the ip_mobile.h header file will only be included if NOMIP is not set (i.e. if Mobile-IP is in use). The Berkeley code had unconditionally included this header file which caused a compilation error.

- This code calls either the Markov error model or the Poisson error model depending on which is enabled. The Poisson model maintains static variables err_byte and first_time. The variable first_time ensures that no errors are injected at byte 0 the first time the function is called after reboot (errors may be injected the first time the function is called, but it will be with some offset from byte zero). The Poisson error model increments the err_byte variable by the amount calculated from the Poisson distribution function. In most cases the value incremented is larger than the size of the packet and therefore no error is injected. If the value is within the size of the packet, an error is injected by either changing the TCP checksum or the IP checksum (depending on the value of err_byte).

- The sequence of numbers returned by random() in the Poisson distribution function is the same each time the machine is rebooted. However, the sequence of packets injected with an error will be different each time because the err_byte value is static and it will vary depending on the packet sizes (and not all packet sizes will be the same).

- The value of error_prob[0] determines the mean packet error injection rate. The default is 65,536. This means that on average one out of every 65,536 bytes will be injected with an error. The variance is significant however, and sometimes the errors may be injected roughly 2000 bytes apart while other times they are injected almost 500,000 bytes apart.

- Note: Because IP only does a header checksum, it will not discover errors which have been injected within its data portion (i.e. the TCP header or the application data). TCP on the other hand does a checksum over the entire packet and therefore discovers errors in the TCP header and its data.

- With the default byte error rate of once every 65,536th packet, the bit error rate is $1/(8*65{,}536) = 1.9\text{x}10^{-6}$. Also, the packet error rate is 1 in every 1460/65,536 or about 2.3%.

- One of the problems with using the burst error method employed for this error model is that it destroys a burst of the next N packets where N is the burst size variable. This is probably not realistic when packets are not being transmitted continuously because it can mean that the burst lasts for a very long time. Thus, if the first packet sent in slow start mode is dropped as the first damaged packet in a burst, it will time out and be sent again (thus becoming the second packet in the burst), and then time out a second time, etc. This will continue until the burst of errors is complete (i.e. N packets have been damaged). A burst of errors in this type of situation could last for many seconds.

rand.c

- The only change that has been made to rand.c is that the ip_mobile.h header file will only be included if NOMIP is not set (i.e. if Mobile-IP is in use). The Berkeley code had unconditionally included this header file which caused a compilation error.

- The file contains an array of 50,000 numbers. The minimum value in the table is the 42,876th entry with a value of 3087 and the maximum value in the table is the 13,922nd entry with a value of 2,135,942,571 and the mean is 266,299,707. The numbers are randomly ordered within the table, but if they are sorted and plotted, the exponential distribution of the numbers is evident.

ber.h

- Currently no modifications have been made to ber.h.

if_ethersubr.c

- No modifications have been made at this time to our FreeBSD if_ethersubr.c file. There are actually no changes made to if_ethersubr.c by Berkeley which are directly related to snoop. The majority of the differences are related to a reliable link layer protocol tested at Berkeley. Changes for the reliable link layer were made in ether_input(), ether_output(). The functions ll_rxmit(), and ll_sendack() were also added for the reliable link layer protocol.

- The main thing which may be useful to add to if_ethersubr.c is a second fastq which allows packets with the IPTOS_LOWDELAY option set to be queued in front of other packets in the main output queue. Snoop attempts to use the fastq to send packets being locally retransmitted ahead of other packets. To see the syntax of how a fastq is used with a FreeBSD driver, look at the files if_ppp.c, if_sl.c, and if_pppsubr.c which are in the /usr/src/sys/net/ directory. Addition of the fastq will require several modifications:
  1) The ifnet structure must be modified to include a fastq member.
  2) The code to queue the outgoing packet on the fastq must be added to ether_output() in the if_ethersubr.c file (lines 329-332).
  3) Constants such as IPTOS_LOWDELAY, IPTOS_RELIABILITY, and IPTOS_THROUGHPUT are already defined in ip.h and may be used.

- Initially, it is easier not to use the fastq option. Therefore, IPTOS_LOWDELAY will be disabled from use in snoop.c.

- Fastq is really intended for small interactive packets or control data and not large volumes of data. Thus it is normally used with telnet, rlogin, etc. and commands within other applications, but not with the data transfer packets for ftp, etc. See TCP/IP Ill. Vol. 1 Section 3.2 for a list of usage.

tcp_var.h

- It does not appear that there are any changes in tcp_var.h which are directly related to snoop. There are many changes related to SACK and SMART. Also, many of the structure members defined in BSDI are defined as short rather than int like in FreeBSD. No modifications have been made at this time to our FreeBSD tcp_var.h file.

Detailed Notes:

ip_input.c

- The functions within ip_input are as follows:
  ip_init - Initializes info. by filling in the IP protocol switch table.
  ipintr - Main function to process incoming packets. Calls most other IP functions.
  ip_reass - Reassembles incoming fragments.
  ip_freef - Handle IP fragment buffering utilities.

ip_enq - Handle IP fragment buffering utilities.
ip_deq - Handle IP fragment buffering utilities.
ip_slowtimo - Handle IP fragment buffering utilities.
ip_drain - Handle IP fragment buffering utilities.
ip_dooptions - Process options.
ip_rtaddr - Returns routing info given internet address.
save_rte - Saves incoming source routes for use in replies.
ip_srcroute - Uses saved source routes from save_rte for its replies.
ip_stripoptions - Strips out IP options.
ip_forward - Forwards packets that haven't reached destination, calls ip_output.
ip_sysctl - Set/get IP control parameters

- Note: ip_doing_reass is only used in ip_drain.

- Note: One main difference between BSDI and FreeBSD is the use of bcopy by BSDI and the use of memcopy by FreeBSD in similar situations.

- Note: There is no queuing between the transport layer and IP layer for incoming packets.

<u>tcp_usrreq.c</u>

- The functions within ip_input are as follows:
  tcp_usrreq
  tcp_connect - This opens the TCP connection, but isn't isn't in BSDI or TCP/IP Ill.
  tcp_ctloutput - Sets and gets socket options.
  tcp_attach
  tcp_disconnect
  tcp_usrclosed
  tcp_sysctl - BSDI and FreeBSD versions are different, but accomplish same thing.

- When an option is being set, the value of the option (i.e. the error probability, enable/disable, etc.) is passed into tcp_ctloutput() as an mbuf pointer variable called mp. A local mbuf pointer variable m is then set to this address. The statement "i = *mtod(m, int *)" is used to pull the desired option value from the mbuf and store it as an integer in a local variable. The global variables are then set equal to the local variable i. The option values are always integers.

- When an option is being read, an mbuf is allocated and pointed to by both mp and m. The value of the desired global variable is then copied into the mbuf as an integer using the statement "*mtod(m, nit *) = <global variable>".

- Note: The code under ITCPEMU is only defined for setting and clearing parameters. The results of these can not be viewed with getsockopt like all of the other TCP improvements. This code is not related to snoop.

- Note: The code under TCP_Monitor which checks the round trip timing information is only used by getsockopt(). The round trip timing parameters can not be set by using a setsockopt() call.

tcp.h

- Note: Some code at the beginning of the file is added for ELN and delayed ACK code. I don't think we need any of it at this time.

- Note: There are additional options which are for use with SACK and SMART. These are not part of snoop, but they give a good model of how to add options if we need to do this as part of our future research.

if_ethersubr.c

- The functions within ip_input are as follows:
  ether_output - Called by if_output, encapsulate data and put on device queue
  ether_input - Place incoming data on IP input queue or arp queue, etc.
  ether_sprinf - Converts ethernet address to printable representation.
  ether_ifattach - Attach an interface to the interface list and set it up.
  *ether_attach - BSDI version used instead of ether_ifattach
  ether_addmulti - Adds multicast address to the list.
  ether_delmulti - Removes multicast address from the list.
  *ll_rxmit - Experimental reliable link layer protocol function from Berkeley.
  *ll_sendack - Experimental reliable link layer protocol function from Berkeley.

- Note: To use the reliable link layer protocol from Berkeley a number of other structures must be modified in other files. Also, the LLPROTO flag must be defined.

snoop.c

- Calling ip_forward - The primary calls to ip_forward from snoop are when a new piece of data has been received by snoop_data and it must be forwarded from the base station to the mobile host, when a new ACK has been received by the base station from the mobile host and it must be forwarded to the wired host, and when a packet must be locally retransmitted by the base station to the mobile host. Snoop_ctrl directly calls ip_forward in unusual situations such as when: the packet isn't a TCP packet (i.e. it is UDP or other protocols), snoop is in bypass mode, the packet has the SYN bit set, or a handoff has occurred.

- The fastq option at the ethernet level will not initially be used, so the SNOOP_IPTOS option will not be defined in snoop.c. Since IPTOS_LOWDELAY, IPTOS_RELIABILITY, and IPTOS_THROUGHPUT are defined in ip.h of FreeBSD, nothing else related to IP tos was changed in snoop.c.

- The functions within snoop.c are as follows:
  snoop_ctrl - Main function of snoop. It is called by ipintr() in ip_input.c
  snoop_init - Initializes snoop at boot time. Called from ip_init() in ip_input.c
  snoop_conninit - Called when a SYN is received to initialize connection info.
  snoop_getmobconn - Returns -1 or the ID associated with mobile addr/port
  snoop_data - Used for data from fixed host to mobile host.
  save_ippkt - Just used by snoop_insert to save packet info.
  snoop_insert - Places packet into the snoop cache
  snoop_smartoption - Called from snoop_ack. We won't use it initially
  snoop_ack - Used for ACKs from mobile host to fixed host
  snoop_close_pkt - Called by ACK
  snoop_freebuf - Frees a single packet buffer
  snoop_cleanbufs - Called by snoop_ack to clear data which has been ACKed
  snoop_sendnack - Not used. No function body defined.
  snoop_done - Clears snoop state on FIN from fixed host or RST from either end.
  snoop_mip_change - Used for handoffs. Not initially used by us.
  snoop_malloc - Allocates a M_SNOOP type of mbuf for snoop structures.
  snoop_rexmt_pkt - Retransmits a single packet
  snoop_rexmt_timeout - Local retransmission timer
  snoop_persist_timeout - Local zero window probe timer
  snoop_garbage_timeout - Removes connection after long period of inactivity

- There are several main data structures within snoop. First, snoop_state_t maintain state information which is specific to the entire base station. Second, conn_state_t maintains state information which is specific to a given connection. The snoop_state_t structure contains an array of 64 connection pointers. Thus a base station may have up to 64 TCP connections that are snooped at the same time. Third, packet_t maintains state information which is specific to a given packet. Each conn_state_t structure has an array of 40 packet pointers. This allows each connection to have a circular buffer with up to 40 packets. Finally, each packet_t structure also has a iphr substructure which keeps selected IP header information.

- When the packet buffer reaches a high water mark (90 % full), it will only accept packets which fall within its current sequence range. This allows packets which have been delayed slightly within the wired network to be accepted into the snoop cache even at the high water mark. Any packets received at this point which have a sequence number above the highest one currently cached will be directly forwarded without being cached by snoop.

- When the packet buffer is completely full, any new packets received will be forwarded without caching unless a special LRU method is used. If the LRU method is used, the oldest packet is overwritten by the incoming packet.

- Packets are always placed in the circular buffer by order of sequence number. If a packet is received out of order, it is placed in sequence, but no holes are left in the buffer for packets occurring before it which have not yet been received. Adding packets to the end or tail of the buffer is simple. Adding packets in the middle of the buffer involves the shifting of packet pointers.

- Note: The timeout() and untimeout() functions used by snoop are standard kernel timers which are defined in sys/systm.h and are used to set timers and stop (un-set) timers. The arguments for timeout() are the function which should be called upon expiration of the timer, any structures/variables to be passed to the function called upon timer expiration (in snoop's case the connection state structure), and the interval to use for the timer. The arguments for untimeout() are the same except no interval is given.

- The version of snoop that we are using only allows data transfer from the base station to the mobile host.

- Snoop_persist_timeout has a small bug. It seems that a set of brackets is missing. This bug only becomes apparent if the debugging code within this function is defined. Otherwise, the function works fine.

- The comments given at the start of snoop_ack and snoop_ctrl are very good.

- Note: The type timev has been defined in snoop.h as struct timeval.

- Note: Since we will not initially be using handoffs, snoop will always be functioning in forwarding mode rather than buffering mode.

Snoop Debug Statements

```
conninit - return
```

Printed each time that snoop_conninit is called to create a new connection entry within the snoop cache. This message is printed at the top of the function without any previous processing.

```
data: seq = 501bbae1
```

Printed each time that snoop_data is called. It unconditionally prints the TCP sequence number in hex of the first byte of data within the new packet that has just arrived. This message does not have a carriage return.

```
way out-of-order pkt 501bbae1, lastack 501bbaef
```

Printed only when the packet which has just been received has a sequence number that is before the left edge of the receiver's window (i.e. the mobile host's flow control window). The first number printed is the sequence number of the packet received in hex and the second number is the sequence number of the last packet acked in hex. This message occurs right before an abnormal exit from snoop_insert.

```
have pkt 501bbae1 at 3
```

Printed only when the packet received is already currently cached. The first number is the sequence number of the packet just received in hex and the second number is the index where the packet has already been cached. This message occurs right before an abnormal exit from snoop_insert.

```
      cache reorg; pkt 501bbae1, head 1, tail 5
```

Printed only when the packet received must be inserted into the middle of the cache (i.e. it is an out of order packet that will fill a hole in the receiver's window). The sequence of the new packet is printed in hex and the index for the head and tail of the circular buffer are printed. This message does not have a carriage return.

```
                                   501bbae1 at 5
```

This is the typical message called right after the new packet has been inserted. It gives the packet sequence number in hex, and the index of the newly inserted packet in the cache.

```
              pkt 501bbae1 out of order, last 501bbaef
```

This is printed whenever the packet just inserted into the snoop cache does not have a higher sequence number than all other packets currently cached. Last is the sequence number of the packet having the highest sequence number.

```
ack 501bbae1, expect 1, dacks 0  cached 501bbae1-0
```

This message is printed unconditionally at the start of snoop_ack each time an ACK is received. It prints the ACK value of the ACK packet just received. It also prints the number of duplicate ACKs expected and the sequence numbers for the packets at the head and tail of the snoop cache.

```
        spurious ack 501bbae1
```
This is printed when the ACK is for a packet which has previously been ACKed. The ACK will be passed though with no further processing. <u>This is part of case 1 for an ACK.</u>

```
                don't have 501bbae1 letting thru
```
This is printed when the packet being ACKed has not been cached by snoop. The ACK will be passed though with no further processing. <u>This is part of case 2 for an ACK - duplicate ACKs.</u>

```
        in cache 501bbae1
```
This is printed when the packet being ACKed has been cached by snoop and is a duplicate acknowledgment. This message does not have a carriage return. <u>This is part of case 2 for an ACK - duplicate ACKs.</u>

```
                501bbae1 sender rxmit
```
This is printed when the packet being ACKed has been cached by snoop and was retransmitted by the sender. The sequence number of the packet retransmitted is printed. <u>This is part of case 2 for an ACK - duplicate ACKs.</u>

```
                ack 501bae1 expect 1 more
```
This is printed when the packet being ACKed has been cached by snoop and we are expecting further duplicate ACKs. The number of duplicate ACKs expected is printed as the second number above. <u>This is part of case 2 for an ACK - duplicate ACKs.</u>

```
                sample 2753 srtt 51040
```
This is printed when the packet being ACKed has been cached by snoop and we are expecting duplicate ACKs. The number of duplicate ACKs expected is printed as the second number above. <u>This is part of case 3 for an ACK - normal new ACK sequence.</u>

```
snoop_done: already closed connection 47
```
This is printed when snoop_done is called after a connection has already been closed. The connection id number is printed at the end.

```
Untimeout
```
This is printed from within snoop_done once for each retransmit time-out that is reset.

```
srtt 58987
```
This is printed within snoop_done as the connection is removed from the snoop cache.

```
Closed 47
```

This is printed within snoop_done as the connection is removed from the snoop cache.

`rexmt seq 501bae1`

This is printed within snoop_rexmt_pkt when a packet is retransmitted due to a local retransmission time-out.

`timeout: next timeout in 20`

This is printed within snoop_rexmt_timeout. It is possible for this to be printed without having the message above printed since this message is printed even if the time-out occurs with an empty packet cache. Snoop_rexmt_pkt is not called when the cache is empty however.

`Persist timeout called`

This is printed each time the snoop_persist_timeout function is called. This timer will be called every half second. During inactive periods, this message will be printed five times in succession. After 2 seconds of inactivity the persist time-out removes the connection from the cache by calling snoop_done.

Appendix D
FreeBSD Measurement Upgrades

Available Measurement Methods

1) **tcp_trace with trpt** - see IP Ill. Vol. 2 P. 918, man pages for trpt
Use the **tcp_trace** function to gather the information and **trpt** to print the results. The structure used to store the statistics is in tcp_debug.h and the function tcp_trace() is in tcp_debug.c. To use tcp_trace() SO_DEBUG must be set. Other flags can be set to print the data directly to the screen rather than retrieving it from the buffer with trpt. This method will capture data at the TCP level, so any damaged or lost packets will not be included in the data. The data collected will include entire TCP header information, but not summary information like netstat (see below). The key disadvantage to this method is that the tcp_trace buffer is currently limited to 100 packets. Packet number 101 will overwrite packet number 1. We can change this size, but each packet requires 196 bytes of storage, so even this small default buffer is using 19,600 bytes.

2) **BPF with tcpdump** - see TCP/IP Ill. Vol. 1 App. A, man pages on tcpdump (BSD)
Use **BPF** (Berkeley Packet Filter) to gather the information and **tcpdump** to filter and print the results. This will capture data at the link layer (from ethernet taps in wlread/leread for input and lestart/wlstart for output. Therefore, any damaged packets will be included in the data although they will not reach the TCP layer. Initially, there were concerns whether BPF would work with the WaveLAN driver. In looking at the driver, all the right taps appear to be there, so we should be able to use it as a data gathering option.

   - The BPF code already appears to be included in our kernel builds.
   - The WaveLAN driver calls bpfattach correctly and uses a type of DLT_EN10MB which is the standard ethernet type.
   - Should print "bpf… …attached" at boot time if the bootverbose flag is set.
   - Files related to BPF include net/bpf.h, net/bpf.c, net/bpf_filter.c, net/bpf_compat.h, bpfdesc.h, and compile/MYKERNEL/bpfilter.h. The file bpf_compat.h has nothing to do with us and is for use with the SUN OS only. The file bpfilter.h has the sole purpose of defining NBPFILTER to be 4.

3) **netstat -s** - see TCP/IP Ill. Vol. 2 P. 797 for tcpstat structure, man pages for netstat
Using netstat with the -s option gives TCP and other protocol data. The data for TCP is taken at the TCP level by lines sprinkled throughout the TCP code which update tcpstat structure members. The structure is included within the tcp_var.h file in the source code. The main drawback to this method is that it gives summary data rather than per packet information, and it lumps the information from all TCP connections together.

4) **netperf**
Berkeley has a netperf program which will collect network performance statistics. This was used by Berkeley to collect data on their TCP experiments. The program looks too large and complex (much larger than the entire snoop code) to be a viable option. It was intended to be part of the sprite operating system and was written by one of the author's of the snoop code, so this is probably why it was used to gather statistics in the Berkeley snoop implementation.

Trpt Code Modifications

Key Modifications:

Socket Programs

- The test_sender.c program was modified to include a setsockopt() call which sets the SO_DEBUG for the socket created for the transfer if statistics collection has been enabled by the user.

  Note: tcp_trace() and trpt operate on a per connection basis, so it is necessary to set the SO_DEBUG option each time a new connection is made.

  Note: trpt will give a misleading error if you type trpt <filename> rather than trpt > <filename>. It will say no namelist which is the same thing it says when the symbolic links have not been established.

Option Flag Settings in /i386/conf/ MYKERNEL

- The following option flag should be set in order include the tcp_trace() code which captures data within tcp_input.c and tcp_timer.c
  TCPDEBUG_TR

- Some of the #ifdef TCPDEBUG statements within tcp_input.c and tcp_timer.c were changed to #ifdef TCPDEBUG_TR so that tracing could be enabled without enabling all socket debugging statements. Note that no changes have been made to capture outgoing data packets at the sender, since it is the feedback (in the form of ACKs) and time-outs that we are concerned with for our experiments.

File Additions in /i386/conf/files.i386

- The file tcp_debug.c should be added to the file files.i386 so that this source code file containing tcp_trace() will be compiled when the kernel is rebuilt. The format used is slightly different than for BSDI (as shown by Hari in the files.i386.add file on the daedalus ftp site). The new file should be added as follows:

  ```
  netinet/tcp_debug.c    optional    tcpdebug_tr
  ```

Note: The third value on each line corresponds to the option flag added in the file MYKERNEL above (i.e. tcpdebug_tr goes with TCPDEBUG_TR).

<u>tcp_debug.h</u>

- The size of the trace circular buffer was increased from 100 to 2000 packets. This is enough so that the buffer will not wrap around during a typical 2 MB data transfer.

  Note: There is a copy of tcp_debug.h in the directory /usr/include/netinet which is used by applications programs (such as trpt) which must also be updated.

- Specific code fragments which should be modified by line number:

```
Lines 78     #define TCP_NDEBUG 2000
```

<u>tcp_debug.c</u>

- The symbolic links need to be included in order for the data captured by tcp_trace() to be read by trpt.

- Specific code fragments which should be modified by line number:

```
Lines 51     #define TCPDEBUG changed to #define TCPDEBUG_TR
```

<u>tcp_input.c</u>

- The header tcp_debug.h needs to be included if tracing is to be done. Therefore, it is conditionally included whenever TCPDEBUG or TCPDEBUG_TR is defined. The lines below ensure that it is included if TCPDEBUG_TR is defined, and that it is not included twice if TCPDEBUG is also defined.

- The constant TCPDEBUG has been changed to TCPDEBUG_TR each place where the tcp_trace() code is called.

- The tcp_trace() code was originally intended for finding bugs. Therefore, it was not needed in the header prediction section of the code (since this code only executes when everything is perfect). For our purposes, we want to measure good behavior just as much as abnormal behavior. Therefore, additional calls were included in the header prediction section of the code.

- Specific code fragments which should be added by line number:

```
Line 96     #ifndef TCPDEBUG
Line 97     #ifdef TCPDEBUG_TR
Line 98     #include <netinet/tcp_debug.h>
```

Line 99   struct tcpiphdr tcp_saveti
Line 100   #endif
Line 101   #endif
Line 301   #define TCPDEBUG changed to #define TCPDEBUG_TR
Line 416   #define TCPDEBUG changed to #define TCPDEBUG_TR
Lines 541-544 Added call for header prediction of a pure ACK packet.
Lines 611-614 Added call for header prediction of a pure data packet.
Line 1655 #define TCPDEBUG changed to #define TCPDEBUG_TR
Line 1674 #define TCPDEBUG changed to #define TCPDEBUG_TR
Line 1692 #define TCPDEBUG changed to #define TCPDEBUG_TR
Line 1713 #define TCPDEBUG changed to #define TCPDEBUG_TR

tcp_timer.c

- The header tcp_debug.h needs to be included if tracing is to be done. Therefore, it is conditionally included whenever TCPDEBUG or TCPDEBUG_TR is defined. The lines below ensure that it is included if TCPDEBUG_TR is defined, and that it is not included twice if TCPDEBUG is also defined.

- The constant TCPDEBUG has been changed to TCPDEBUG_TR each place where the tcp_trace() code is called.

- Specific code fragments which should be added by line number:
  Line 91   #ifndef TCPDEBUG
  Line 92   #ifdef TCPDEBUG_TR
  Line 93   #include <netinet/tcp_debug.h>
  Line 94   struct tcpiphdr tcp_saveti
  Line 95   #endif
  Line 96   #endif
  Line 145   #define TCPDEBUG changed to #define TCPDEBUG_TR
  Line 168   #define TCPDEBUG changed to #define TCPDEBUG_TR
  Line 174   #define TCPDEBUG changed to #define TCPDEBUG_TR

trpt.c

The source code for trpt is found in the directory /usr/src/usr.sbin/trpt in the file trpt.c. A Makefile is also included which should be used to ensure that the symbolic constants are linked properly. The only portion of the code for trpt.c which should ever be changed is within the tcp_trace() function which formats and prints the data collected. This portion of the code has been changed significantly. The structure used to store the statistics used by trpt is in tcp_debug.h and the function tcp_trace() which collects those statistics is in tcp_debug.c. To use tcp_trace() SO_DEBUG must be set. This method will capture data at the TCP level, so any damaged packets will not be included in the data. The data collected will include the entire set of TCP header information and control block information. While this requires a lot of space, it makes trpt very flexible. Since the information is already captured, trpt may be modified to print data in a specific format for a given run without changing anything at the kernel level.

Appendix E
FreeBSD Partial ACK Upgrades

Partial ACK Code Modifications

Key Modifications:

Socket Programs

- The config_wireless.c program was changed so that it now has an option to enable or disable partial acknowledgments (ACKP) at the base station. Because the base station contains all of the coding to determine when to generate an Ack_p and to send the Ack_p, there is no need to enable or disable anything at the sender or receiver. Trying to set the ACKP option at the sender will have no effect (which is the proper thing to do).

- In order to use the ACKP code, the snoop code must also be enabled. The ACKP code is called from within the snoop code, so if the snoop code is disabled, the ACKP code will never be called.

- While snoop is automatically enabled by the kernel when the machine is booted, the ACKP code is automatically disabled by the kernel when the machine is booted. The user must explicitly turn ACKP on using the config_wireless program. This may be changed in the future after ACKP becomes less experimental.

/i386/conf MYKERNEL

- A new option called SNOOPACKP was defined within MYKERNEL at the sender and the base station. This option causes the additional ACKP code to be compiled into the kernel.

```
Options SNOOPACKP # enable ackp code
```

snoop.h

- The function prototype for snoop_send_ackp() was added.

snoop.c

- All of the new code specific to the base station has been added within the snoop.c file. One new function called snoop_send_ackp has been added. This function is called anytime a partial acknowledgment needs to be generated at the base station. It is provided with an mbuf containing the packet that requires the partial acknowledgment. From this it is able to construct a partial acknowledgment packet consisting of the TCP header and IP pseudo-header within an mbuf. This mbuf is then passed to ip_output which sends the partial acknowledgment packet back to the fixed host. It is better to pass the packet to ip_output() rather than to ip_forward() because ip_output() will not set certain fields in forwarded packets (since they have presumably already been set by the sender's IP layer).

```
#ifdef SNOOPACKP
int snoop_send_ackp(struct mbuf *mdata_pkt)
{
   int error=0;
   struct tcpiphdr *ti;
   struct tcpiphdr  *data_pkt = mtod(mdata_pkt, struct tcpiphdr *);
   unsigned optlen=0, hdrlen=0;
   u_char opt[TCP_MAXOLEN];
   u_long *lp;
   struct mbuf *m;
   int flags=0;
   long len=0, recv_wnd = 17280;


/*
 * Create the ackp option to be appended to the header below.
 */

   hdrlen = sizeof(struct tcpiphdr);
   lp = (u_long *)(opt + optlen);
   *lp++ = htonl(TCPOPT_NOP << 24 |
              TCPOPT_NOP << 16 |
              TCPOPT_ACKP << 8 |
              TCPOLEN_ACKP);
   optlen += 4;
   hdrlen += optlen;


/*
 * Obtain an mbuf for the pure ack (i.e. no data) we are about to send.
 */

   MGETHDR(m, M_DONTWAIT, MT_HEADER);
   if(m == NULL){
      error = ENOBUFS;
      return(error);
   }

   m->m_data += max_linkhdr;
   m->m_len = hdrlen;
   m->m_pkthdr.rcvif = (struct ifnet *) 0;
   ti = mtod(m, struct tcpiphdr *);


/*
 * Fill in the tcpiphdr fields for the ackp segment.
 */

   ti->ti_next = ti->ti_prev = 0;
   ti->ti_x1 = 0;
   ti->ti_pr = IPPROTO_TCP;
   ti->ti_src = data_pkt->ti_dst;
   ti->ti_dst = data_pkt->ti_src;
   ti->ti_sport = data_pkt->ti_dport;
   ti->ti_dport = data_pkt->ti_sport;
```

```
        ti->ti_seq = data_pkt->ti_ack;
        ti->ti_ack = data_pkt->ti_seq;
        ti->ti_x2 = 0;
        bcopy((caddr_t) opt, (caddr_t)(ti + 1), optlen);
        ti->ti_off = (sizeof(struct tcphdr) + optlen) >> 2;
        ti->ti_flags = 0;
        ti->ti_flags = ti->ti_flags & TH_ACK;
        ti->ti_win = htons((u_short) recv_wnd);
        ti->ti_sum = 0;
        ti->ti_urp = 0;
        ti->ti_len = htons((u_short) (sizeof(struct tcphdr) + optlen + len));


    /*
     * Perform the tcp checksum for the ackp segment.
     */

        ti->ti_sum = in_cksum(m, (int) (hdrlen + len));


    /*
     * Fill in the ip header fields which tcp normally fills in for ip.
     *
     * The values of 64 for ttl and 0 for tos used here were the same as
     * values captured using printf statements within the kernel for other
     * packets.  Also, they are a single unsigned character, so there is no
     * byte order to be concerned with.
     */

        m->m_pkthdr.len = hdrlen + len;
        ((struct ip *)ti)->ip_len = m->m_pkthdr.len;
        ((struct ip *)ti)->ip_ttl = 64; /* XXX */
        ((struct ip *)ti)->ip_tos = 0; /* XXX */


    /*
     * Send the ackp by calling ip_output with only the mbuf.  No options
     * or route are specified.
     */

        error = ip_output(m, (struct mbuf *)0, (struct route *)0, 0, 0);
        return (error);

    }
    #endif /* SNOOPACKP */
```

<u>tcp.h</u>

- The new TCP option for partial acknowledgments is defined in tcp.h and so is the length of the new option.

`#define TCPOPT_ACKP   32`	Defines option kind as 32
`#define TCPOLEN_ACKP   2`	Defines option length as 2

- The constant used for the case statement in tcp_ctloutput() which sets and gets socket options was also defined as:

```
#define TCP_SNOOP_ACKP_ENABLE  0x220
```

<u>tcp_debug.h</u>

- A new type of packet and a new name in the array *tanames* were defined in tcp_debug.h to go with the "ackp". This allows Ack_p packets in the trace buffer to be distinguished from other dropped acknowledgments by trpt when it prints out the debugging results.

```
    #define TA_ACKP     5

    char    *tanames[] =
        {"input","output","user","respond","drop","ackp"};
```

<u>tcp_var.h</u>

- The tcpopt structure which is passed between tcp_input() and tcp_dooptions() has been updated to include an option flag for ACKP called TOF_ACKP which is defined as 0x0016. This flag will be set by tcp_dooptions() when the ACKP option is present. It is then tested in tcp_input() and responded to appropriately.

<u>trpt.c</u>

- The trpt.c code needed an additional case added to handle the new Ack_p packet type. This case is handled exactly the same as a TA_DROP case, except that the word “ackp” is printed rather than “drop”. This allows us to easily spot Ack_p packets in the trpt output.

<u>tcp_input.c</u>

- This code has been modified to recognize and react to the new TCP partial acknowledgment option. With the current tcp_input() flow, this happens as follows:

  1) Receive Segment
  2) Perform Checksum
  3) Locate PCB
  4) Process Options in tcp_dooptions()
  5) If ACKP option flag is set
     - Perform partial acknowledgment timer back-off and goto drop:
  6) Record partial acknowledgment receipt with tcp_trace()
  7) Drop the packet

- A case statement was added to tcp_dooptions() so that it will recognize the new ACKP option. If the ACKP option is present, the ACKP option flag will be set. Note that

FreeBSD uses a much cleaner way of passing data and flags between tcp_dooptions() and tcp_input() than the version in TCP/IP Ill. Vol. 2. It defines everything directly related to the options in the option structure discussed above.

```
     #ifdef SNOOPACKP
          case TCPOPT_ACKP:
          if (optlen != TCPOLEN_ACKP){
               printf("Invalid ACKP option length \n");
          }
          to->to_flag |= TOF_ACKP;
          break;
     #endif
```

- The following piece of code is in tcp_input() directly after the call to tcp_dooptions(). It checks to see if the ACKP flag is set and if so, it resets the timer to a constant value and then goes to the drop label at the end of tcp_input(). The *constant_value* is a constant which is roughly two times the current time-out value. For a LAN the constant value used was 10 ticks (5 seconds) and for a low bandwidth wireless network the constant value used was 20 ticks (10 seconds). Initially the timer was just reset, but this does not prevent time-outs because it effectively changes the time-out by at most 1 tick since the Ack_p packet arrives back at the sender only milliseconds after the timer has been set.

```
     #ifdef SNOOPACKP
       if(to.to_flag & TOF_ACKP){
          tp->t_timer[TCPT_REXMT] = constant_value;
          goto drop;
       }
     #endif
```

- The following piece of code occurs directly after the drop label. It records the packet in the trace buffer as either a dropped acknowledgment or an Ack_p depending on whether the ACKP option flag is set. Then it drops the packet. If we do not want to place Ack_p packets in the debugging buffer, we can just remove the if clause below.

```
#ifdef TCPDEBUG_TR
  if (tp == 0 || (tp->t_inpcb->inp_socket->so_options &
SO_DEBUG)){
#ifdef SNOOPACKP
     if(to.to_flag & TOF_ACKP)
       tcp_trace(TA_ACKP, ostate, tp, &tcp_saveti, 0);
     else
#endif /* SNOOPACKP */
       tcp_trace(TA_DROP, ostate, tp, &tcp_saveti, 0);
  }
#endif /* TCPDEBUG_TR */
  m_freem(m);
```

tcp_timer.c

- The function tcp_xmit_timer() is called when the round trip timer is being used, and the acknowledgment is for a sequence number greater than the sequence of the packet being timed. If ts_present is true, timestamps are being used. Otherwise, the old method of timing one packet per window is being used. See page 836 of TCP/IP Ill. Vol. 2 for details. Below is the typical way that the tcp_xmit_timer() function is called:

```
if(ts_present)
   tcp_xmit_timer(tp, tcp_now - ts_ecr + 1);
if(tp->t_rtt && SEQ_GT(ti->ti_ack, tp->tp->t_rtseq))
   tcp_xmit_timer(tp, tp->t_rtt);
```

Note: We have the updated timestamp code proposed by Bob Braden. See page 870 of TCP/IP Ill. Vol. 2 for details.

tcp_subr.c

- The FreeBSD code has implemented the TCP timestamp option and the T/TCP (TCP for transaction) option. These options have been enabled by setting the global variables tcp_do_rfc1323 (timestamps) and tcp_do_rfc1644 (T/TCP) to 1. In order to prevent these options from possibly affecting our results, they have been disabled (set to 0) on Ravel, Verdi, and Chopin. Therefore, the only TCP option which is being used is the partial acknowledgment one which we have created.

tcp_usrreq.c

- The code added to this file allows the user to turn the ACKP code on and off using setsockopt() calls from an application program such as config_wireless. The current setting can also be read using a getsockopt() call.

- The following code was added to the tcp_subr.c file just before the tcp_ctloutput() function. It defines the external variable snoop_ackp_enable which is declared in the file snoop.c. This variable is set and checked with the new ACKP case statements in tcp_ctloutput().

```
#ifdef SNOOPACKP
extern int snoop_ackp_enable;
#endif /* SNOOPACKP */
```

- The following code was added to tcp_ctloutput() to set the ACKP option. Note that it is enclosed within the #ifdef MOBILE statement, so that it will only be defined on a machine with a wireless link.

```
#ifdef SNOOPACKP
  case TCP_SNOOP_ACKP_ENABLE:
    i = *mtod(m, int*);
    snoop_ackp_enable = i;
    break;
#endif /* SNOOPACKP */
```

- The following code was added to tcp_ctloutput() to read the ACKP option. Note that it is enclosed within the #ifdef MOBILE statement, so that it will only be defined on a machine with a wireless link.

```
#ifdef SNOOPACKP
  case TCP_SNOOP_ACKP_ENABLE:
    *mtod(m, int *) = snoop_ackp_enable;
    break;
#endif /* SNOOPACKP */
```

<u>Notes:</u>

- An outgoing packet calls in_cksum() with the checksum field set to zero. This causes a checksum to be calculated and the value returned by in_cksum() is placed in the checksum field (resulting in a non-zero value). An incoming packet calls in_cksum() with the value calculated by the receiver in the checksum field. For an undamaged packet, the value calculated should be the compliment of the value in the checksum field and therefore the new checksum value upon returning from in_cksum() is zero. The value for TCP's checksum recorded for trpt will always be zero since the checksum has already been calculated by the receiver at the point tcp_trace() is called.

- An attempt has been made to set the fields in the Ack_p packet header as accurately as possible. For many of the fields in the header, this is not really required since the packet will be dropped before any of them are used by tcp_input().

- Using the TCP timestamp option will not improve the granularity of the round trip time values (t_rtt). The values will still be in ticks and the value of a tick can vary depending on the machine being used (usually 200 ms or 500 ms). The advantage of the timestamp option is that the rtt for a higher fraction of the packets is measured.

- It doesn't appear that there is any reason for the partial acknowledgment code to set any IP options. These all relate to source routing, route tracing, and route timing which should not be used during a bulk data transfer.

- The variable ip_id is a global variable declared in ip_output.c and incremented each time a new segment is sent. It is used on a per-machine rather than a per-connection basis and its intent is to allow the receiver to reassemble fragmented IP segments by using the segment identification and offset. Since we know that our partial acknowledgment packets will not be fragmented (since they are very small ACK packets with no data), this is not a real issue. However, the IP layer will assign an appropriate ip_id for each of our packets.

- When the ip_output() function successfully sends a packet, it should return an error code of zero. Otherwise, it returns one of the following error codes which are defined in the file /usr/src/sys/sys/errno.h:

ENETUNREACH	51
EHOSTUNREACH	65
ENOBUFS	55
EADDRNOTAVAIL	49
EACCES	13
EMSGSIZE	40

  Other errors returned to it by the link layer.

- Initially, I had thought that our scheme would be in direct conflict with the code at the beginning of the tcp_output() function which initiates slow start if it receives new output after a period of inactivity (see page 853 of TCP/IP Ill. Vol. 2). However, our function is only applied while there is activity since it generates partial acknowledgments in response to data packets received at the base station. Also, our timer back-off is really effective when the coarse time-out used by the sender is tighter (as is being suggested for newer TCP implementations). Thus, backing off on these tight timers should be no worse in terms of congestion than the very loose timers being used today.

- Since a low bandwidth wireless link was not available, and it would be difficult to simulate this by having many connections opened at once, a delay was added to the base station's code at the IP layer to create a low bandwidth environment. This delay was then used to test plain TCP, the snoop protocol, and snoop plus ACKP. A base station that is equipped with snoop will pass every packet that is to be forwarded on the wireless link to the snoop code. If snoop is turned off, the packet will just be passed to ip_forward() with no further processing. Thus, every packet destined for the wireless link passes through the snoop code regardless of which TCP enhancements are enabled. Therefore, all of the required delays can be added to the snoop code. A delay was added in snoop_ctrl() for the case when snoop is turned off. Another delay was added in snoop_data() for new packets arriving at the base station and being forwarded by snoop. A third delay was added in snoop_rexmt_pkt() for retransmitted packets. In each case the single command "DELAY(100000);" which causes a busy wait was added.

- Other methods of creating delays were attempted at the link-layer. However, these all proved to be unsatisfactory because they caused packets to be lost due to missed interrupts (i.e. the WaveLAN card was taking so much time to send the packet that it blocked the ethernet card's interrupts for incoming packets).