OSI/Billing Records Collection for WATT 3.3
Functional Requirements

1.0 Goal

To use OSI Interface to collect and analyze specific Billing Records content in real-time.

2.0 Requirements

- The WATT box shall send a request for Billing Records collection to the OSI Billing Server.
- The OSI Billing Server shall fetch billing records generated by the switches.
- The OSI Billing Server shall first 'parse' and then 'filter' the billing records.
- The OSI Billing Server shall send the filtered billing records over the network to the WATT box.
- The WATT box shall receive the filtered billing records from the OSI Billing Server, match these records against script line requests.

3.0 Details

The Messages, Control Information as well as Data, shall be processed in the following order:

1. The WATT initiates Dialog through the WATT OSI Controller (WOC) to get OSI Billing Records transferred. The following Filtering Parameters shall be passed to the OSI Billing Server (OBS):

    - Set of circuits to watch
    - Switch identifier
    - Software load

2. Acknowledgment received by the WATT from the WOC.

3. The Filtering Parameters forwarded by the WOC to the OBS Parser/Filter.

4. Initiation of OSI/FTAM Billing Records from the WOC to the Switch Under Test (SUT) via the OSI Processor (OSI/P).

5. Billing Records transferred by the SUT to the OSI Parser, and FTAM acknowledgment to WOC.

6. The OSI Parser parses the Billing Records and passes down to the OSI Filter. The OSI Parser shall be able to perform 'shifting' and 'masking' operations on the raw binary data received from the SUT and shall be able to parse:

    - Record Types
    - Circuit Number

7. The OSI Filter filters the Billing Records as per the Parameters and ships the filetered records down to the WATT parser over LAB/CORPORATE LAN. The OSI Filter shall be able to decide as to what records one needs to capture, and what records one needs to throw away. The Ouput Processor of the OBS shall perform a byte-swap on the received data due to high-endian to low-endian conversion. The processed records shall be sent to the WATT.

8. Stop Sending Records message sent from the WATT to the WOC.

9. Stop Transfer message from the WOC to the SUT upon which the FTAM session is suspended.

3.1 Components

The various physical components of the OSI/Billing Records Collection are:

3.1.1 WATT

1. Send START message with Filtering Parameters to WOC.
2. Send RUNNING message to WOC to indicate WATT is ON and ready to receive the billing records.
3. Send STOP message to WOC.
4. Receive START ACK (Acknowledgment for receiving START message) from WOC upon successul FTAM transfer of Billing Records from SUT to OBS Parser.
4. Receive FTAM ERROR from WOC if FTAM Failure Message received by WOC from OSI/P.

3.1.2 WATT OSI Controller (WOC)

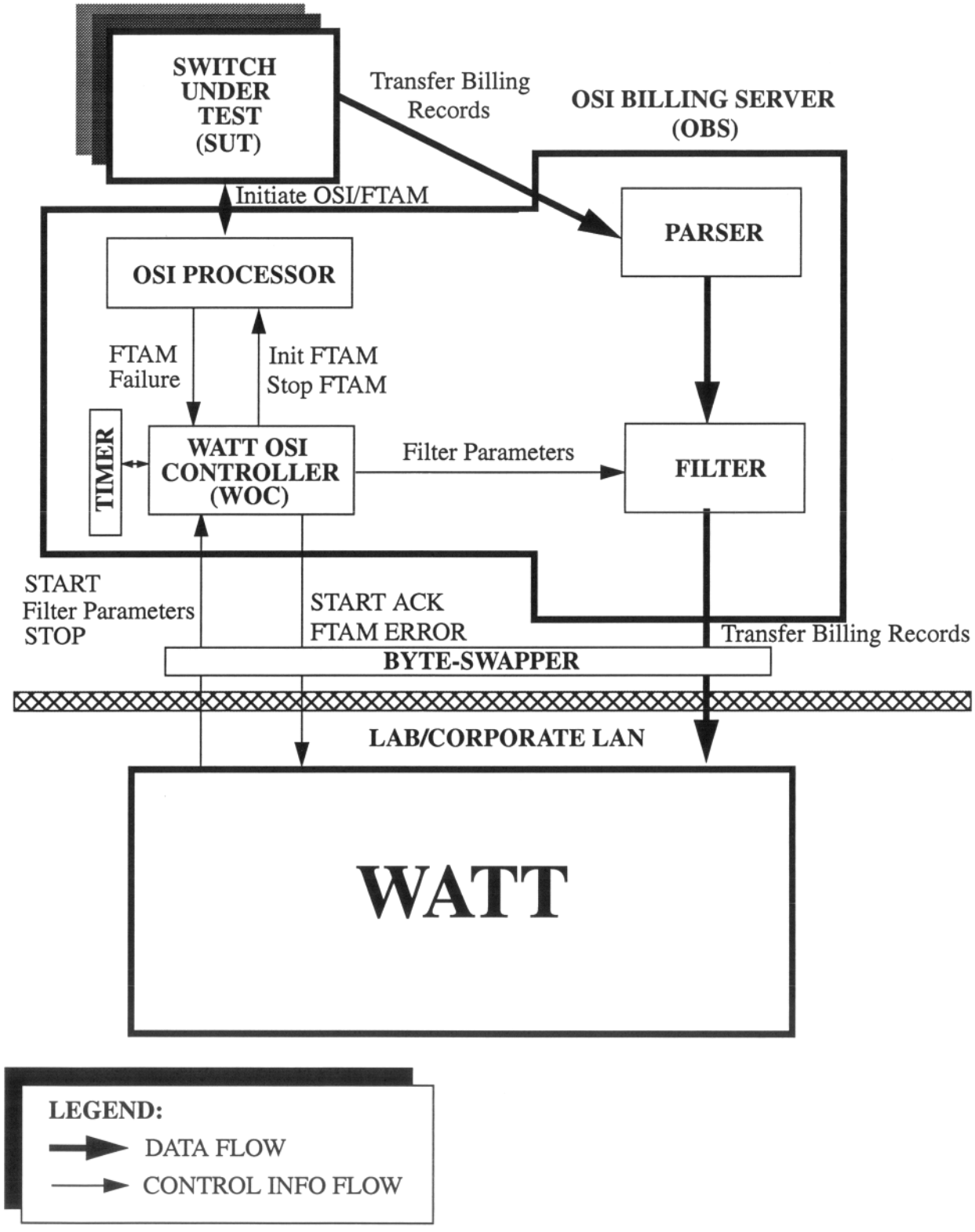
1. Receive START message with Filtering Parameters from WATT.
2. Receive RUNNING message from WATT.
3. Receive STOP message from WATT.
4. Send START ACK to WATT.
5. Forward Filtering Parameters to OBS Filter.
6. Initiate OSI/FTAM transfer by sending Init FTAM to OSI/P.
7. Stop FTAM transfer by sending Stop FTAM to OSI/P.
8. Send FTAM ERROR to WATT if FTAM Failure Message received by WOC from OSI/P.

3.1.3 OSI Billing Server (OBS) Parser/Filter

1. Receives Billing Records from SUT.
2. Filter in only the records as per the Filtering Parameters.
3. Byte-swap the records.
4. Send Records to WATT.

3.1.4 OSI Processor (OSI/P)

1. Receive Init FTAM message from WOC.
2. Send Billing Records to OBS Parser.
3. Send FTAM Failure Message to WOC if problems with FTAM initiation.
3. Receive Stop FTAM (stop OSI/FTAM transfer) message from WOC.



OSI/BILLING RECORDS COLLECTION

3.2 Operating Modes

Various Operating Modes are:

3.2.1 START Mode

1. START Message, with Forwarding Parameters, from WATT to WOC.
2. START ACK from WOC to WATT.
3. Init FTAM from WOC to OSI/P.
4. Billing Records transferred from SUT to OBS Parser.

3.2.2 WORKING Mode

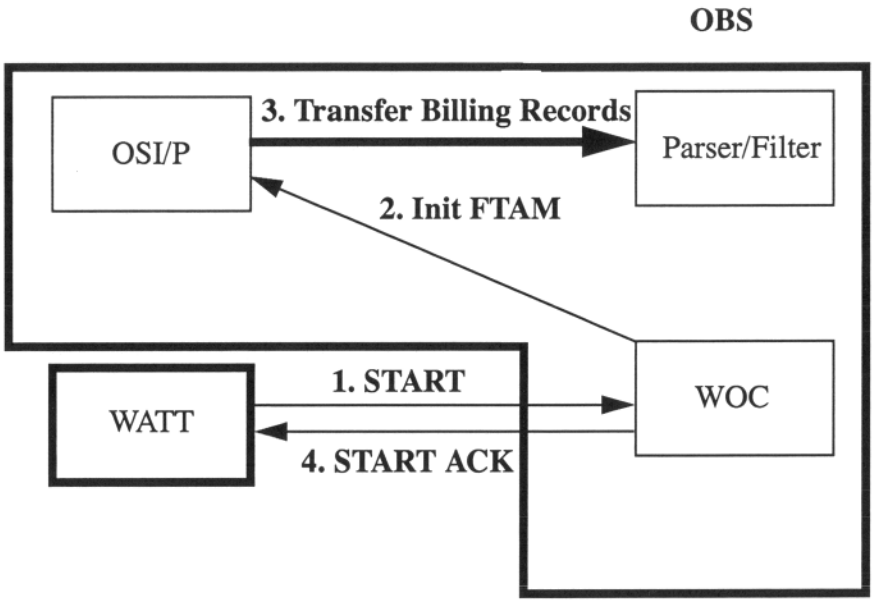
1. RUNNING Message from WATT to WOC.
3. OBS first parses and then filters Billing Records as per Filtering Parameters and then byte-swaps.
3. OBS Filter sends the filtered Billing Records to WATT.

3.2.3 SHUTDOWN Mode

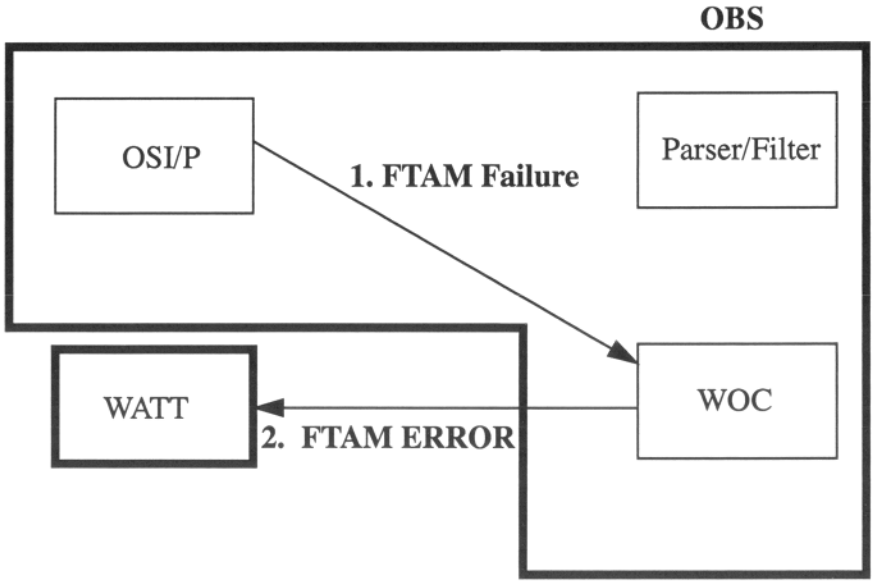
1. STOP Message from WATT to WOC.
2. Stop FTAM Message from WOC to SUT.
3. WATT performs cleanup on its end.

3.2.4 FAILURE Mode

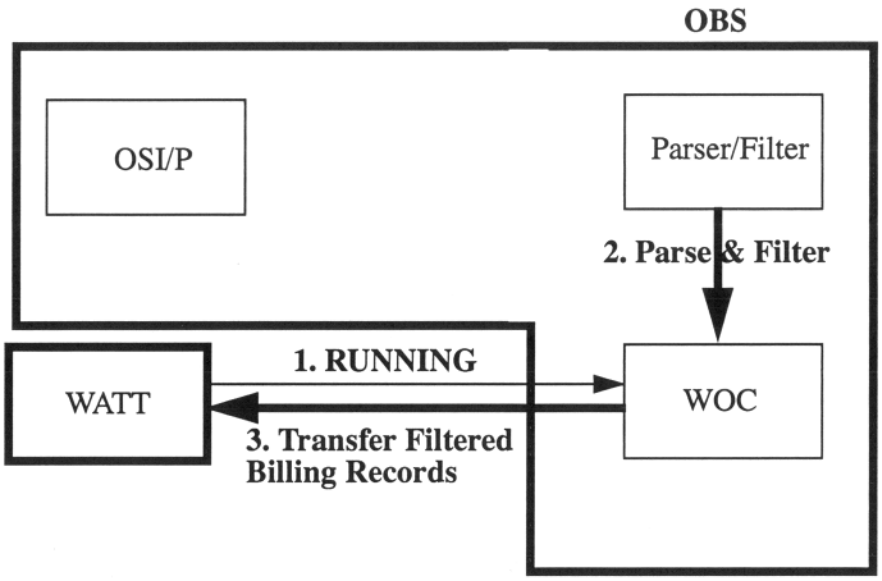
SUT does not send records in some specified time period. In that case, OSI/P sends FTAM Failure Message to WOC, and WOC sends FTAM ERROR across the network to WATT.



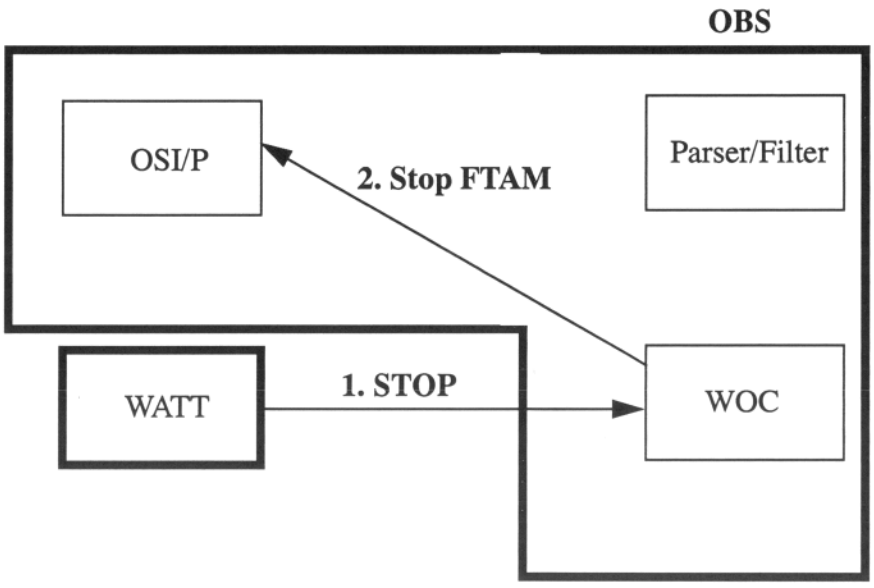
START Mode



FAILURE Mode



WORKING Mode



SHUTDOWN Mode

OSI/Billing Records Collection for WATT 3.3
Detailed Design

The OSI Billing Server (OBS) running on a Sun Workstation (**sun1209** Sparc) has to be launched before WATT is turned ON to request an OSI CDR. Once the OSI server is launched as a background process, it waits for a request from a client (WATT). The socket Datagrams are used for receiving/sending messages from/to the client.

I. On OBS side:

- Open a socket connection to communicate with WATT.
- If successful, wait for OSI_START message from WATT.
- If OSI_START message received, collect the Filtering Parameters (Switch identifier, set of circuits to watch and software load), and send Init FTAM message to OSI/P.
- Send OSI_START_ACK to WATT and initiate FTAM transfer of raw binary data from SUT to OBS. If FTAM failure message is received, send OSI_FTAM ERROR to WATT.
- If OSI_STOP message received from the WATT, immediately send Stop FTAM message to OSI/P and suspend WOC and OBS and stop receiving the Billing Records.

I.1. The OSI Server

The main file on the OSI Server to perform duties of an OSI server is **osi_server.c**. It **parses** the *ftamfile* and strips records off of it, **filters** the ones that happen to be on the CDR circuits that we are watching, and then ships over the CDRs to WATT. It has all the necessary code to bring up (down) socket communication, and be able to receive/send messages over the network. Since it is a server, it is capable of handling requests from more than one client. All the definitions, `#define`'s and `#include`'s are contained in the header file **osi_cdr.h**.

I.1.1 New Functions

read_client()

This function performs the initial handshake with WATT (client), handles all its requests, and dumps a CDR as soon as it finds one after parsing and filtering the billing records. It kicks off appropriate shell scripts to initiate as well as to stop OSI/FTAM transfer of raw binary data from SUT.

parse()

To break a string buf, with white spaces in it, into an array of individual strings.

break_tokens()

This program reads the CDRCKTS line from startup.ats file and creates an array containing each circuit number or range of circuit numbers to test CDRs to be passed to WATT.

test_cktnum()

This function tests the input Circuit number against the array.

filter()

This function filters out CDRs based on Circuit number they are observed on.

non_block_read()

This function performs a non-block read operation on an open socket.

I.1.2. Existing Functions

readBR()

This function reads a billing record from the current place in the billing file.

disassemble_record()

This function disassembles a Billing Record.

determine_defaults()

This function validates and sets defaults associated with the parsed invocation-command line.

initBilling()

This function opens the billing file, reads the volume record (if input source is tape), then goes to the 1st record position in the billing file.

I.1.3. Pseudo-Code for the OSI Server

+ Open a Datagram Socket and wait for a request from a client (WATT).
+ IF OSI_START is received (non-block read), THEN
    - Read in Filtering Parameters
    - Send OSI_START_ACK to the client
    - Initiate FTAM transfer of AMA file from SUT to OBS. If any error, then send OSI_FTAM_ERROR to the client and cancel the request.
+ ELSE IF OSI_RUNNING is received (after OSI_START is processed), THEN
    - Read the Billing Records from the FTAM file
    - Disassemble only CDRs (Parser Stage), ignore other billing records
    - Filter out only those that are observed on the given CDRs, ignore others
    - Send the Filtered CDRs to the client
    - Perform non-block read on the opened connection for a possible OSI_STOP request from the client. If OSI_STOP is received, then do the following, otherwise continue.
+ ELSE IF OSI_STOP is received, THEN
    - Terminate FTAM transfer
    - Close Socket connection with the client and clean up
    - Wait for next request from any client.

I.2. Shell Scripts

The following scripts have been written:

I.2.1. osicdr

This script launches an OSI Billing Records server application as a background process. To run this command, enter the following at the prompt:

`osicdr` {*filename*}

where *filename* is optional, the default being **ftamfile**, which is a file, stored locally, that AMA file from SUT is copied over to, using FTAM transfer (see **dir_ama** script below).

Example: `osicdr myftamfile`

I.2.2. getfile.scr

This script is launched as soon as the OSI server receives a request from WATT to initiate the FTAM transfer of OSI CDRs. To run this command, enter the following at the prompt:

`getfile.scr` (*switchname*) {*filename*}

The *switchname* (mandatory) to get CDRs from is received from WATT, using socket datagram, and the *filename* (optional) is same **ftamfile** file as above.

Example: `getfile.scr mci2 myftamfile`

I.2.3. stop.scr

This script stops the FTAM transfer of the AMA file and then removes the *filename*. To run this command, enter the following at the prompt:

`stop.scr` {*filename*}

Example: `stop.scr myftamfile`

I.2.4. killit

This script performs the following tasks:

a. Kills the FTAM transfer process.
b. Removes {*filename*}.
c. Kills the OSI server process.

This script is run for cleanup purposes as well as to drastically end the OSI transfer process. This script does not affect the WATT side - the WATT simply stops receiving the CDRs. To run this command, enter the following at the prompt:

`killit` {*filename*}

Example: `killit  myftamfile`

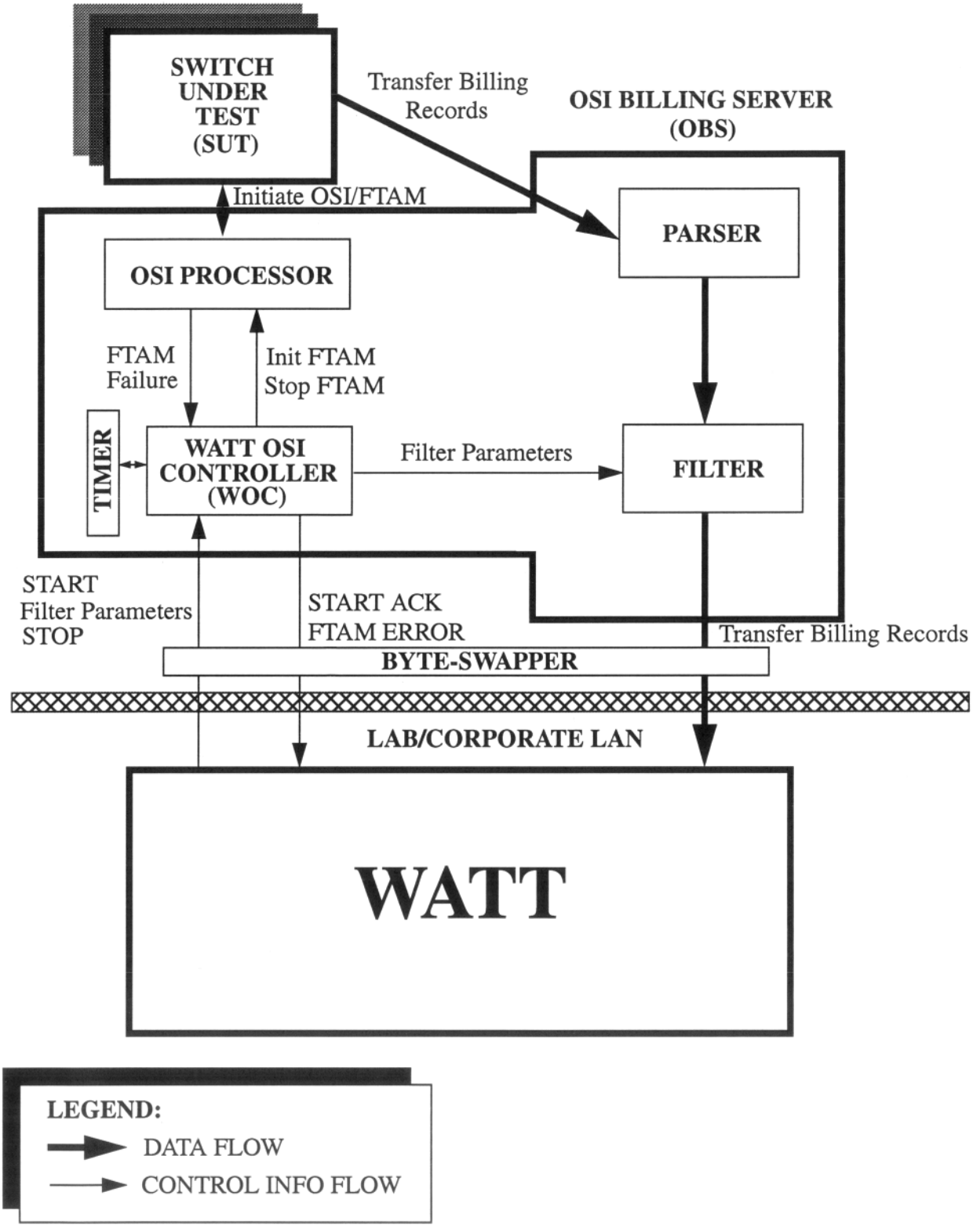
I.2.5. dir_ama

This script, that already existed on the server prior to this work, is run to get directory listing of AMA files on SUT. To run this command, enter the following at the prompt:

`dir_ama` (*switchname*) {*filename*}

The *switchname* (mandatory) to get CDRs from is received from WATT, using socket datagram, and the *filename* (optional) is same **ftamfile** file as above.

Example: `dir_ama mci2  myftamfile`



OSI/BILLING RECORDS COLLECTION

II. On WATT side:

- Open a socket connection to communicate with OBS.
- Send OSI_START message with Filtering Parameters to WATT OSI Controller (WOC).
- Receive OSI_START_ACK message from WOC.
- Send OSI_RUNNING to WOC when WATT is ON.
- When WATT is stopped, send OSI_STOP message to WOC.
- Close the socket connection.

The following modules have been modified:

II.1. admin.c:

i. Read pertinent CDR startup label (N or CAMP or OSI or BOTH).

II.2. cdr_init.c:

i. Initialize Socket communication parameters and initiate dialog with the OSI server by sending it an OSI_START message.
ii. Fork off camp_if process if and only if CDR=CAMP | BOTH
iii. Cleanup after OSI_STOP message has been sent to the OSI server.

II.3. cdrmgr.c:

i. Send OSI_RUNNING to the OSI server when the WATT is ON.
ii. Receive the OSI CDRs from the OSI server.
iii. Put the received OSI CDR in its own message queue, CDR_QKEY, for further processing - as if it came from a **camp_if** process.
iv. Send OSI_STOP to the OSI server when the WATT is turned OFF.

II.4. Startup File (startup.ats):

i. The following field has been **added**:
   a. SWITCH=xxx
      Example: SWITCH=mci2

ii. The following field has been **modified** to take one of four possible values:
   a. CDR=n | CAMP | OSI | BOTH
   where
      n,N : No CDRs desired
      CAMP, camp : CDRs using DIGIBOARD
      OSI : CDRs using OSI/FTAM
      BOTH : CDRs using both OSI/FTAM and DIGIBOARD
      Example: CDR=BOTH

III. Operating Modes

The various operating modes are:

III.1. START Mode

The OSI server receives OSI_START message, along with the following filtering parameters, from WATT:

i. Switchname, to get OSI CDRs from.
ii. The CDR circuits, to observe the CDRs on.
iii. Software Load.

The OSI server, upon the receipt of OSI_START, sends OSI_START_ACK to the WATT in order to acknowledge the receipt of START message and the Filtering Parameters, gets a directory listing of the AMA files on the switch by running the script **dir_ama**, initiates FTAM transfer of an active AMA file to a local file *ftamfile* by running the script **getfile.scr**.

III.2. WORKING Mode

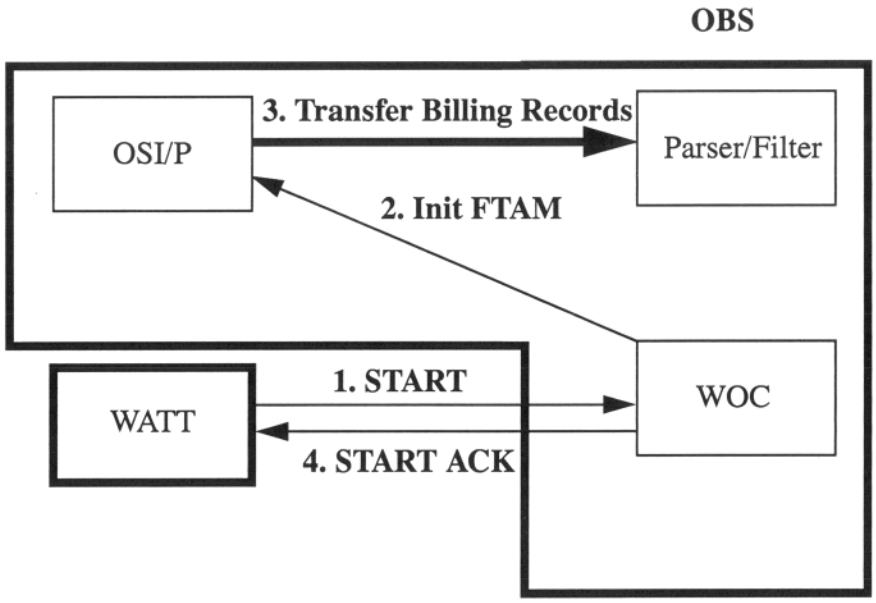
The OSI server receives OSI_RUNNING from WATT as soon as WATT is up and running. It signals the transfer of OSI CDRs from the OSI server to the WATT.

III.3. SHUTDOWN Mode

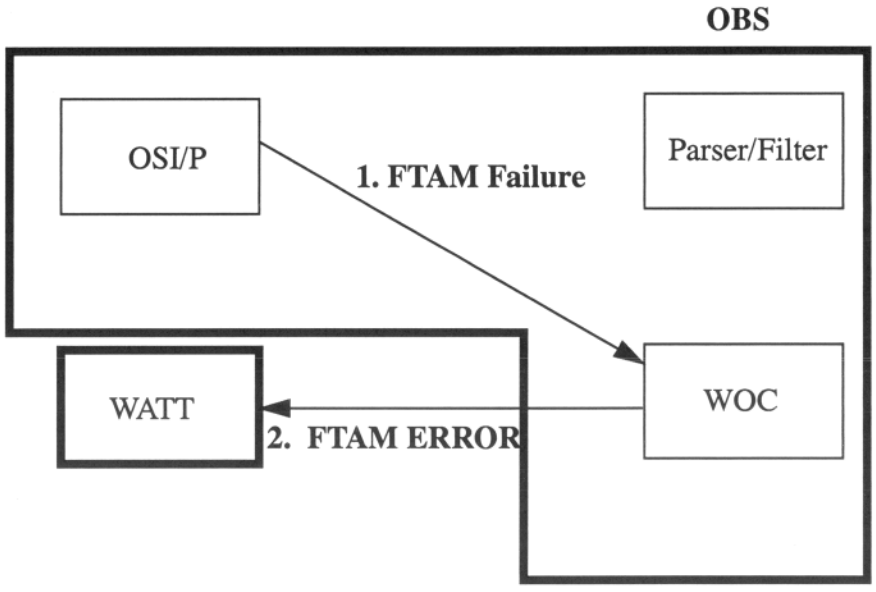
The OSI server receives the OSI_STOP from WATT when WATT wants the OSI server to discontinue sending the OSI CDRs, especially when WATT (Re)Starts or Stops. The script **stop.scr** is launched.

III.4. FAILURE Mode

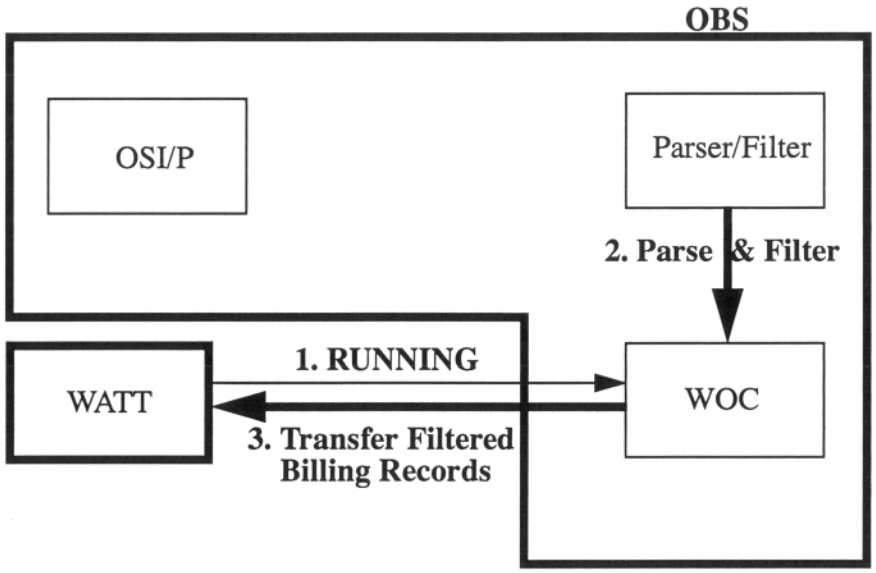
The OSI server sends OSI_FTAM message to WATT upon a failure to initiate an FTAM transfer.



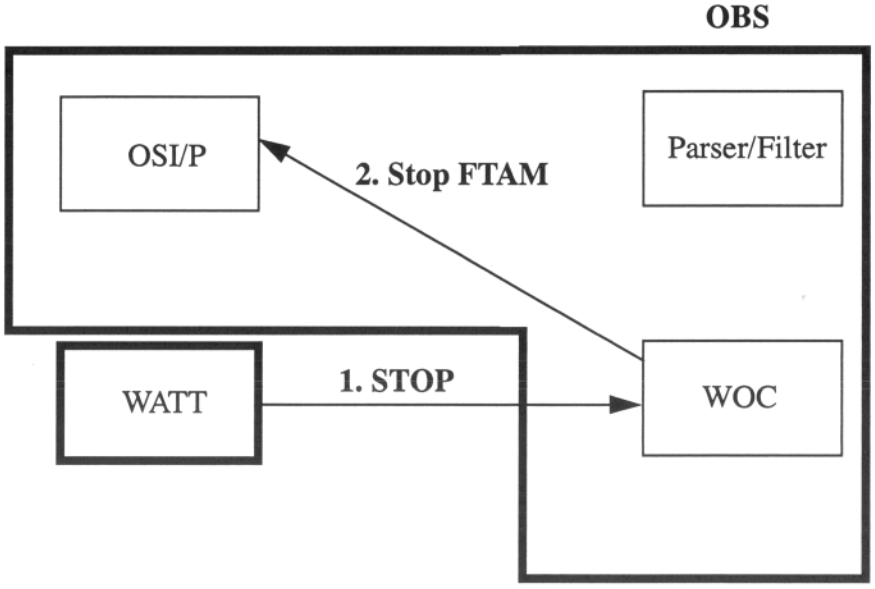
START Mode



FAILURE Mode



WORKING Mode



SHUTDOWN Mode

I. OSI BILLING RECORDS SERVER (OBS) SIDE

I.1. The OSI Server

The main file on the OSI Server to perform duties of an OSI server is **osi_server.c**. It **parses** the *ftamfile* and strips records off of it, **filters** the ones that happen to be on the CDR circuits that we are watching, and then ships over the CDRs to WATT. It has all the necessary code to bring up (down) socket communication, and be able to receive/send messages over the network. Since it is a server, it is capable of handling requests from more than one client. All the definitions, #define's and #include's are contained in the header file **osi_cdr.h**.

I.1.1 New Functions

read_client()

This function performs the initial handshake with WATT (client), handles all its requests, and dumps a CDR as soon as it finds one after parsing and filtering the billing records. It kicks off appropriate shell scripts to initiate as well as to stop OSI/FTAM transfer of raw binary data from SUT.

parse()

To break a string buf, with white spaces in it, into an array of individual strings.

break_tokens()

This program reads the CDRCKTS line from startup.ats file and creates an array containing each circuit number or range of circuit numbers to test CDRs to be passed to WATT.

test_cktnum()

This function tests the input ckt number against the array.

filter()

This function filters out CDRs based on Circuit number they are observed on.

non_block_read()

This function performs a non-block read operation on an open socket.

I.1.2. Existing Functions

readBR()

This function reads a billing record from the current place in the billing file.

disassemble_record()

This function disassembles a Billing Record.

determine_defaults()

This function validates and sets defaults associated with the parsed invocation-command line.

initBilling()

This function opens the billing file, reads the volume record (if input source is tape), then goes to the 1st record position in the billing file.

OSI_SERVER.C

```
/*********************************************************************
* OSI_SERVER.C
*
* PURPOSE:
*               This is the main file on the OSI Server to perform duties of an
*               OSI Billing Records Server. It parses the ftamfile and strips
*               records off of it, filters the ones that happen to be on the CDR
*               circuits that we are watching, and then ships over the CDRs to
*               WATT. It has all the necessary code to bring up (down) socket
*               communication, and be able to receive/send messages over the
*               network. Since it is a server, it is capable of handling
*               requests from more than one client.
*
* REVISION HISTORY:
*
*  Date         Author            Reason
*  --------    ------------  ------------------------------
*  5/20/1997   Sunil Fotedar     ORIGINAL
*              David Colvin
*********************************************************************/
#include "osi_cdr.h"

main(int argc, char *argv[])
{
                int             sockfd, newsockfd, clilen, childpid;
                struct          sockaddr_in    cli_addr, serv_addr;

                if((sockfd=socket(AF_INET,SOCK_DGRAM,0))<0) {
```

```
                fprintf(stderr,"OSI server: problem with opening datagram socket\n");
                exit(1);
            }

            memset((char *) &serv_addr, '\0', sizeof(serv_addr));
            serv_addr.sin_family = AF_INET;
            serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
            serv_addr.sin_port= htons(SERV_UDP_PORT);

            if(bind(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr))<0) {
                        fprintf(stderr,"OSI server: problem with bind()\n");
                        exit(1);
            }

            /* process the request */
            read_client(sockfd, (struct sockaddr *)&cli_addr, sizeof(cli_addr), argv);

}
/*****************************************************************************
 * ROUTINE:  read_client()
 *
 * PURPOSE: This function performs the initial handshake with WATT (client),
 *              handles all its requests, and dumps a CDR as soon as it finds
 *              one after parsing and filtering the billing records. It kicks
 *              off appropriate shell scripts to initiate as well as to stop
 *              OSI/FTAM transfer of raw binary data from SUT.
 * AUTHOR:  Sunil Fotedar
 *
*****************************************************************************/
read_client(int sockfd, struct sockaddr *cli_addr, int maxclilen, char *argv[])
{
    int    i, index, index_size, clilen;
    int    n=0, done=1, osi_started=0, system_rtn;
```

```
        char            buffer[256], cdrbuffer[256], switchname[20], temp[10];
char    *args[10];
static  OSI_MSG msg, recvmsg;
        BREC            bill_struct;
        billRec *record;
        long            type;

        for(;;) {
                        clilen = maxclilen;
           if(non_block_read(sockfd))
           {

                        memset((char *) &recvmsg, 0, sizeof(recvmsg));
                        memset((char *) &msg, 0, sizeof(msg));
                        n = recvfrom(sockfd, (char *)&recvmsg,sizeof(recvmsg),
                                                0, cli_addr, &clilen);
                        if (n<0) {
        fprintf(stderr,"OSI server: recvfrom error");
        exit(1);
    }
                        fprintf(stderr,"\nType:%u, Message: %s\n",
                                                ntohs(recvmsg.msg_hdr.msgtype),
                                                recvmsg.data_buf.contents);

                        switch((int)ntohs(recvmsg.msg_hdr.msgtype))
                        {
                                case OSI_START:

                                                parse(recvmsg.data_buf.contents, args);

                                                strcpy(recvmsg.switchname,args[0]);
                                                strcpy(recvmsg.filter_params.swgeneric,args[1]);
                                                strcpy(recvmsg.filter_params.cdrckts,args[2]);
```

```
                                        fprintf(stderr,"The Filtering Parameters are:\n\t \
Switch Name    : %s\n\t \
SWGENERIC      : %s\n\t \
CDR Circuits   : %s\n\n",
                                                recvmsg.switchname,recvmsg.filter_params.swgeneric,
recvmsg.filter_params.cdrckts);

/* Send an acknowledgment to WATT that you have received START message
  with Filtering Parameters */
                                        memset((char *) &msg, 0, sizeof(msg));
                                        msg.msg_hdr.msgtype = htons(OSI_START_ACK);
                                        strcpy(msg.data_buf.contents,"OSI_START_ACK");

                                if(sendto(sockfd, (char *)&msg, sizeof(msg), 0, cli_addr, clilen)!=sizeof(msg))
                                        {
                                                fprintf(stderr,"OSI server: Error sending
OSI_START_ACK message to WATT\n");
                                                exit(1);
                                        }

                                        /* Break cdr string into tokens */
                                        strcpy(cdrbuffer,recvmsg.filter_params.cdrckts);
                                        break_tokens(cdrbuffer);

                                        /* get billing file */
                                        buffer[0]='\0';
                                        sprintf(buffer,"getfile.scr %s %s",
                                                        recvmsg.switchname, argv[1]);
                                        system_rtn = system(buffer)>>8;
                                        if(system_rtn==OSI_FTAM_ERROR)
                                        {
                                                fprintf(stderr,"OSI server: Error in FTAM transfer\n");
```

```
                    /* Send OSI_FTAM_ERROR message to WATT to give it a chance to clean up */
                                        memset((char *) &msg, 0, sizeof(msg));
                                msg.msg_hdr.msgtype = htons(OSI_FTAM_ERROR);
                                        strcpy(msg.data_buf.contents,"OSI_FTAM_ERROR");

                                if(sendto(sockfd, (char *)&msg, sizeof(msg), 0,cli_addr,
clilen)!=sizeof(msg))
                                        {
                                                fprintf(stderr,"OSI server: Error sending
OSI_FTAM_ERROR message to WATT\n");
                                        }
                                        exit(1);
                                } /* end of if(system_rtn==OSI_FTAM_ERROR) */

                                /* give some time to have ftamfile ready */
                                sleep(8);
                                osi_started=1;
                    if (!infile)
                    /*  here if 1st input file not yet defined  */
                    {
                                 if ((infile = fopen (argv[1], "r")) == NULL)
                                {
                                        fprintf (stderr, "Failed to open input file:  '%s'.\n", argv[1]);
                                        my_exit (1);
                                }
                    } /* end of if (!infile) */

                        determine_defaults ();
                        if(initBilling(input_source_device,infile)!=S_OK)
                                {
                                        fprintf (stderr, "OSI server: Problem with billing file\n");
                                        exit (1);
                                }
```

```
                done=0;
                break;

case OSI_STOP:
   if(osi_started)
   {
                sprintf(buffer,"stop.scr %s", argv[1]);
                 system(buffer);
                done=1; /* received STOP from WATT */
                osi_started=0;
   }
                break;
case OSI_RUNNING:
    while(!done)
    {
                while ((type=readBR(&record))==CDR_TYPE)
                {
                  if (type < MIN_EXT_TYPE)
                                index_size = BILLREC_SIZE;
                  else
                                index_size = EXT_BILLREC_SIZE;

                  /* Parser */
                  disassemble_record(record, &bill_struct);

                  /* Filter - based on orig. ckt */
                  if(filter(&recvmsg, &bill_struct))
                  {
                  /* ship out the filtered record to WATT */
                msg.msg_hdr.msgtype = htons(CDR_CAMP_RSP);
                                for(index=0;index<index_size;index++)
                                {
                                msg.data_buf.record[index] = htons((*record)[index]);
```

```
                                                fprintf(stderr,"0x%x\t",(*record)[index]);
                                                }
                                        if(sendto(sockfd, (char *)&msg, sizeof(msg), 0, cli_addr,
clilen)!=sizeof(msg))
                                                {
                                                        fprintf(stderr,"OSI server: Error sending
CDR to WATT\n");
                                                        exit(1);
                                                }

                                        } /* end of filter */

        /*
        * make sure that we don't receive a STOP message from WATT in the middle of a CDR dump
        */
                                        if(non_block_read(sockfd))
                        {
                                        memset((char *)&recvmsg,0,sizeof(recvmsg));
                                        n = recvfrom(sockfd,(char *)&recvmsg,
                                                sizeof(recvmsg), 0, cli_addr, &clilen);

                                if (n<0)
                                        {
                                        fprintf(stderr,"OSI server: recvfrom error\n");
                                                exit(1);
                                }

                                        fprintf(stderr,"\nType:%u, Message: %s\n",
                                                ntohs(recvmsg.msg_hdr.msgtype),
                                                recvmsg.data_buf.contents);

                        if((ntohs(recvmsg.msg_hdr.msgtype)==OSI_STOP)
                                                && osi_started)
```

```
                        {
                                                    sprintf(buffer,"stop.scr %s", argv[1]);
                                                    system(buffer);
                                                    done=1; /* received STOP from WATT */
                                                    osi_started=0;
                                                    break;
                        }
                                } /* end of non_block_read */
                              } /* end of while(readBR) */
                        } /* end of while(!done) */

                                                    break;
                                default:
                                                    fprintf(stderr,"The Message from WATT not understood.\n");
                                                    break;
                        } /* end of switch() */
                } /* end of non_block_read */
        } /* end of for(;;) */
}
/*****************************************************************************
 * ROUTINE:  parse()
 *
 * PURPOSE: To break a string buf, with white spaces in it, into an array
 *                  of individual strings
*****************************************************************************/

parse(char *buf, char *args[])
{
        while (*buf != '\0') {
                while((*buf == ' ') || (*buf == '\t') || (*buf == '\n'))
                        {
                                *buf++ = '\0';
                        }
```

```
            *args++ = buf;
            while ((*buf != '\0') && (*buf != ' ')
              && (*buf != '\t') && (*buf != '\n')) buf++;
        }
        *args = '\0';
}
/*****************************************************************************
 * ROUTINE:  break_tokens()
 *
 * PURPOSE:
 *                This program reads the CDRCKTS line from startup.ats file
 *                and creates an array containing each circuit number or range of
 *                circuit numbers to test CDRs to be passed to WATT.
 *
 * AUTHOR: David Colvin
 *
*****************************************************************************/
void break_tokens(char *cdrckts)
{
  int i=1 ;
  int j=1 ;
  char buffer[60];
  char *temp_buff1 ;
  char *temp_buff2 ;
  char *token[10] ;    /*Holds tokens.*/

  strcpy(buffer,"CDRCKTS=");
  strcat(buffer,cdrckts);
  strcpy(cdrckts, buffer);
  /************************************************
  * Point to cdrckts & strip off 'CDRCKTS=' string:
  ************************************************/
  memset(&token[0],0,sizeof(token)) ;
```

```
token[0] = strtok(cdrckts,"=") ;

/*************************************************
* Separate token strings delimited by commas:
* (example: 1234,4567-6789, ...)
*************************************************/
while ((token[i] = strtok(NULL,",")) != NULL)
{
/*   printf("token[%d]= %s \n",i,token[i]) ;*/
  ++i ;
}

/*************************************************
* Place the range numbers in a structure array. Also,
* place single numbers in both start & stop positions.
*************************************************/
while (j < i)
{
  /*************************************************
  * See if the token is a ckt number range (seperated by -)
  * or an individual circuit number.
  *************************************************/
  temp_buff1 = token[j] ;

  if (strstr(temp_buff1,"-") != 0)
  {
    /*************************************************
    * Token is a range of ckt numbers. Copy the
    * start and stop numbers to the array.
    *************************************************/

    /***** Get the interval start number *****/
    temp_buff2 = strtok(temp_buff1,"-") ;
```

```
        range[j].start_num = atoi(temp_buff2) ;

        /***** Get the interval stop number *****/
        temp_buff2 = strtok(NULL,"-") ;
        range[j].stop_num = atoi(temp_buff2) ;
      }
      else
      {
        /************************************************
        * Token is a single number. Copy the number to
        * start and stop numbers of the array.
        ************************************************/
        range[j].start_num = atoi(temp_buff1) ;
        range[j].stop_num = atoi(temp_buff1) ;

      }
      ++j ;

    }/*end while*/

   return;
}/*end break_tokens()*/
/*****************************************************************************
 * ROUTINE:  test_cktnum()
 *
 * PURPOSE:
 *              This function tests the input ckt number against the array.
 *
 * AUTHOR: David Colvin
 *
*****************************************************************************/
int test_cktnum(int test_num)
{
```

```
  int i=1;

  while(range[i].start_num > 0)
  {
    /************************************************
    * See if the current CDR origckt number is in the array.
    ************************************************/
    if ((test_num >= range[i].start_num) && (test_num <= range[i].stop_num))
    {
                return 1;
    }

    ++i;

  }/*end while*/

  return 0; /* nothing matched */

}/*end test_cktnum*/
/*****************************************************************************
 * ROUTINE: filter()
 *
 * PURPOSE:
 *     This function filters out CDRs based on:
 *                          a. Circuit number they are observed on.
 *
 * AUTHOR: Sunil Fotedar
 *
*****************************************************************************/
int filter(OSI_MSG *recvmsg, BREC *bill_struct)
{
                CDR_PNR         *cdr_rec;
```

```
                cdr_rec = (CDR_PNR  *) bill_struct;

                if(!test_cktnum((int)cdr_rec->op))
                {
                        fprintf(stderr, "\nOSI server: No match found on ckt no. %ld\n",
                                                        cdr_rec->op);
                        sleep(2);
                        return 0;
                }
                fprintf(stderr,"\nOSI Filter: Match found for ckt no. %ld,(Hex 0x%x)\n",
                                        cdr_rec->op, cdr_rec->op);
                sleep(2);
                return 1; /* everything matched, send the CDR to WATT */
}
/*****************************************************************************
 * ROUTINE:  non_block_read()
 *
 * PURPOSE:
 *      This function performs a non-block read operation on an open socket.
 *
 * AUTHOR: Sunil Fotedar
 *
*****************************************************************************/
int non_block_read(int sockfd)
{
      int     n=0, nfds;
      fd_set  readfds;
      struct  timeval tv;

      FD_SET(sockfd,&readfds);

                /* Set the timeout */
      tv.tv_sec = 0;
```

```
        tv.tv_usec = 0;

        nfds = select(sockfd+1,&readfds,NULL,NULL,&tv);
        if(nfds>0) {
                        if(FD_ISSET(sockfd,&readfds))
                                {
                return 1; /* something to be read */
                                }
                }

                return 0; /* nothing to be read */

}
/*****************************************************************************/
```

OSI_CDR.H

```
/*********************************************************************
* OSI_CDR.H
*
* PURPOSE:
*      This is the header file for the OSI Billing Records Server.
*                It contains all the pertinent #define's, #include's and
*                definitions.
*
* REVISION HISTORY:
*
*  Date         Author             Reason
*  --------    ------------  ------------------------------
*  5/20/1997    Sunil Fotedar      ORIGINAL
*********************************************************************/
#include <stdio.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include "batch_file.h"
#include "cdr_96.h"
#include "billing_stats.h"   /* module interface definition */
#include "opt_parameter.h"   /* optional parameter data file interface */
#include "get_prog_dir.h"    /* interface for "get_program_dir" routine */
```

```
#define SERV_UDP_PORT   4567
#define SERV_HOST_ADDR  "201.23.2.106"  /* host addr for OSI server sun1209 */
#define MAXLINE         512

/* Message Types */
#define OSI_START          10
#define OSI_STOP           11
#define OSI_START_ACK      12
#define OSI_START_NAK      13
#define OSI_RUNNING    14
#define OSI_FTAM_ERROR    15            /* whenever there is an FTAM problem */
#define STRT_SWGENSIZ      30        /* sw generic load string limit */
#define STRT_CKOSIZ        50        /* circuits for protocols string limit*/
#define ID_SIZE          8       /* Element id field length */
#define CDR_CAMP_RSP       502     /* response from DEX */

#define CDR_TYPE     1                        /* Record Type for CDR */

/************************************************
* Build a structure to contain start and stop numbers
* for ckt ranges. Discrete ckt numbers not in a range
* will have the same number in both start & stop.
************************************************/
struct Range_struct
{
  int start_num ;
  int stop_num ;
};
struct Range_struct range[20];  /*Holds range of numbers or single number*/

INPUT_SOURCE input_source_device = osi;
```

```
FILE *infile = 0;
FILE *outfile = 0;

extern int errno;

typedef struct {
                unsigned char swgeneric[STRT_SWGENSIZ];
                unsigned char cdrckts[STRT_CKOSIZ];
} FilterParams;

typedef struct
{
 unsigned long msgclass;      /* message class of receiver */
 unsigned short msgtype;      /* message type */
 unsigned short sender_id;    /* sender's ID (sender's q-key) */
}MSG_HDR;

typedef struct
{
 MSG_HDR  msg_hdr;            /* message header (routing) */
 unsigned char switchname[10];
 FilterParams filter_params;
 union
 {
   unsigned short record[128];
   unsigned char contents[256];
 } data_buf;
}OSI_MSG;

int    sockfd;
char   hostname[64];
struct hostent *hp;
struct sockaddr_in    serv_addr, cli_addr;
```

```
int filter(OSI_MSG *,BREC *);
void break_tokens(char * ) ;
int test_cktnum(int ) ;
int non_block_read(int );
```

SHELL SCRIPTS

I.2. Shell Scripts

The following scripts have been written:

I.2.1. osicdr

This script launches an OSI Billing Records server application as a background process. To run this command, enter the following at the prompt:

`osicdr` {*filename*}

where *filename* is optional, the default being **ftamfile**, which is a file, stored locally, that AMA file from SUT is copied over to, using FTAM transfer (see **dir_ama** script below).

Example: `osicdr myftamfile`

I.2.2. getfile.scr

This script is launched as soon as the OSI server receives a request from WATT to initiate the FTAM transfer of OSI CDRs. To run this command, enter the following at the prompt:

`getfile.scr` (*switchname*) {*filename*}

The *switchname* (mandatory) to get CDRs from is received from WATT, using socket datagram, and the *filename* (optional) is same **ftamfile** file as above.

Example: `getfile.scr mci2 myftamfile`

I.2.3. stop.scr

This script stops the FTAM transfer of the AMA file and then removes the *filename*. To run this command, enter the following at the

prompt:

`stop.scr` {*filename*}

Example: `stop.scr  myftamfile`

I.2.4. killit

This script performs the following tasks:

a. Kills the FTAM transfer process.
b. Removes {*filename*}.
c. Kills the OSI server process.

This script is run for cleanup purposes as well as to drastically end the OSI transfer process. This script does not affect the WATT side - the WATT simply stops receiving the CDRs. To run this command, enter the following at the prompt:

`killit` {*filename*}

Example: `killit  myftamfile`

I.2.5. dir_ama

This script, that already existed on the server prior to this work, is run to get directory listing of AMA files on SUT. To run this command, enter the following at the prompt:

`dir_ama` (*switchname*) {*filename*}

The *switchname* (mandatory) to get CDRs from is received from WATT, using socket datagram,
and the *filename* (optional) is same **ftamfile** file as above.

Example: `dir_ama mci2  myftamfile`

```
#####################################################################
# SCRIPT:  osicdr
# PURPOSE: To launch the OSI Billing Records Server application.
# USAGE:   osicdr {filename}
# EXAMPLE: osicdr myftamfile
#
# REVISION HISTORY:
#
#  Date          Author             Reason
#  --------    ------------  ------------------------------
#  5/20/1997     Sunil Fotedar    ORIGINAL
#
#####################################################################
if (($# < 1))
then
                                echo ""
                                echo "OSI/FTAM transfer of AMA file to ftamfile."
                                echo ""
                                osi_server ftamfile &
else
                                echo ""
                                echo "OSI/FTAM transfer of AMA file to $1."
                                echo ""
        osi_server $1 &
fi
```

```
#####################################################################
# SCRIPT:  getfile.scr
# PURPOSE: This script is launched as soon as the OSI server
#          receives a request from WATT to initiate the FTAM
#                    transfer of OSI CDRs.
# USAGE:   getfile.scr  (switchname) {filename}
# EXAMPLE: getfile.scr mci2 myftamfile
#
# REVISION HISTORY:
#
#  Date          Author             Reason
#  --------    -------------  -------------------------------
#  5/20/1997    Sunil Fotedar     ORIGINAL
#
#####################################################################
#!/bin/bash
#
# un-comment the next line for testing
# set -x
#
# validate invocation and environment
#
case $# in
  1) ;;
  2) ;;
  *) echo "Usage: getfile.scr {e2|mci2} {filename}"
     exit 1;;
esac

case $1 in
  e2)   ;;
  mci2) ;;
  *) echo "Usage: getfile.scr {e2|mci2} {filename}"
```

```
        exit 1;;
esac

# Grab the batch file name
#
dir_ama $1>tempfile
# Check if Error received
grep Error tempfile > temp2
wc temp2 > temp3
awk '{print $1}' temp3 > temp2
read nnn < temp2
if [ $nnn -ne 0 ]
then
        echo "Error in getting AMA listing"
                echo ""
        exit 2      # OSI_FTAM_ERROR returned to calling function
fi
# looks like everything is fine up to this point
grep AN tempfile > temp1
awk '{print $10}' temp1 > temp2
read filename < temp2
#
# Initiate FTAM
#
echo ""
echo "Initiating OSI/FTAM Transfer of AMA file $filename..."
echo ""
sleep 2 # wait to make sure that $filename is ready to be copied over
fcp $1\!$filename $2& >tempfile
grep failure tempfile > temp2
wc temp2 > temp3
awk '{print $1}' temp3 > temp2
read nnn < temp2
```

```
if [ $nnn -ne 0 ]
then
        echo "Error in OSI/FTAM Transfer"
                echo ""
        exit 2      # OSI_FTAM_ERROR returned to calling function
fi
rm tempfile temp1 temp2 temp3
exit 0 # success
```

```
#####################################################################
# SCRIPT:  stop.scr
# PURPOSE: This script stops the FTAM transfer of the AMA file
#                    and then removes the filename.
# USAGE:   stop.scr {filename}
# EXAMPLE: stop.scr myftamfile
#
# REVISION HISTORY:
#
#  Date           Author              Reason
# --------    -------------  -------------------------------
#  5/20/1997    Sunil Fotedar      ORIGINAL
#
#####################################################################
#!/bin/bash
#
# un-comment the next line for testing
# set -x
#
# validate invocation and environment
#
case $# in
  1)  ;;
  2)  ;;
  *)  echo "Usage: stop.scr {filename}"
      exit 1;;
esac

rm $1
# kill fcp proc
ps -f | awk '$8 ~ /fcp/ {print $2}' > temp
read child_pid < temp ; kill -9 $child_pid ; rm temp
```

```
#####################################################################
# SCRIPT:  killit
# PURPOSE: This script performs the following tasks:
#                       a. Kills the FTAM transfer process.
#                  b. Removes {filename}.
#                  c. Kills the OSI server process.
#                    This script is run for cleanup purposes as well as to
#                    drastically end the OSI transfer process. This script
#                    does not affect the WATT side - the WATT simply stops
#                    receiving the CDRs.
# USAGE:   killit {filename}
# EXAMPLE: killit myftamfile
#
# REVISION HISTORY:
#
#  Date          Author              Reason
# --------    ------------  -------------------------------
#  5/20/1997    Sunil Fotedar     ORIGINAL
#
#####################################################################
# kill fcp proc
ps -f | awk '$8 ~ /fcp/ {print $2}' > temp
wc temp > temp1
awk '{print $1}' temp1 > temp2
read num < temp2
if [ $num -ne 0 ]
then
                  read child_pid < temp
                  kill -9 $child_pid
                  echo ""
                  echo "OSI/FTAM transfer is stopped."
                  rm temp temp1 temp2
else
```

```
                echo ""
                echo "There was no OSI/FTAM transfer."
fi
# Remove FTAM file
if (($# < 1))
then
        if [[ -a ftamfile ]]
        then
                rm ftamfile
                        echo ""
                        echo "AMA file ftamfile is removed."
        fi
else
        if [[ -a $1 ]]
        then
                rm $1
                        echo ""
                        echo "AMA file $1 is removed."
        fi
fi
# kill osi server
ps -f | awk '$8 ~ /osi_server/ {print $2}' > temp
wc temp > temp1
awk '{print $1}' temp1 > temp2
read num < temp2
if [ $num -ne 0 ]
then
                read child_pid < temp
        kill -9 $child_pid
                echo ""
                echo "OSI Server has been killed."
        rm temp temp1 temp2
else
```

```
        echo ""
        echo "The OSI Server was not running at all."
        echo ""
fi
```

II. WATT SIDE

- Open a socket connection to communicate with OBS.
- Send OSI_START message with Filtering Parameters to WATT OSI Controller (WOC).
- Receive OSI_START_ACK message from WOC.
- Send OSI_RUNNING to WOC when WATT is ON.
- When WATT is stopped, send OSI_STOP message to WOC.
- Close the socket connection.

The following modules have been modified:

II.1. admin.c:

i. Read pertinent CDR startup label (N or CAMP or OSI or BOTH).

II.2. cdr_init.c:

i. Initialize Socket communication parameters and initiate dialog with the OSI server by sending it an OSI_START message.
ii. Fork off camp_if process if and only if CDR=CAMP | BOTH
iii. Cleanup after OSI_STOP message has been sent to the OSI server.

II.3. cdrmgr.c:

i. Send OSI_RUNNING to the OSI server when the WATT is ON.
ii. Receive the OSI CDRs from the OSI server.
iii. Put the received OSI CDR in its own message queue, CDR_QKEY, for further processing - as if it came from a **camp_if** process.
iv. Send OSI_STOP to the OSI server when the WATT is turned OFF.

II.4. Startup File (startup.ats):

i. The following field has been **added**:
   a. SWITCH=xxx
      Example: SWITCH=mci2

ii. The following field has been **modified** to take one of four possible values:

a. CDR=n | CAMP | OSI | BOTH

where

n,N : No CDRs desired

CAMP, camp : CDRs using DIGIBOARD

OSI : CDRs using OSI/FTAM

BOTH : CDRs using both OSI/FTAM and DIGIBOARD

Example: CDR=BOTH

ADMIN.C

```
int  read_startup(STARTUP_PARM *start_dat)
{

●●●●●●

        /* startup CDR label */
        else if((strsiz <= (lablsiz= strlen(CDRLABL))) &&
            (!memcmp(label,CDRLABL,lablsiz)))
        {
          strupr(tmpptr);
          /* determine CDR active status */
          if(!strcmp(tmpptr,"CAMP"))
             start_dat->cdr_run = CAMP_CDR;
          else if(!strcmp(tmpptr,"OSI"))
             start_dat->cdr_run = OSI_CDR;
          else if(!strcmp(tmpptr,"BOTH"))
             start_dat->cdr_run = BOTH_CDR;

          cdr_run_flag = start_dat->cdr_run;
        }


●●●●●●

}
```

CDR_INIT.C

```
short cdr_init()
{

●●●●●●


/*************************************************************************/
/* Socket Communications with OSI server                                 */
/*************************************************************************/
   if((atsparms->cdr_run==OSI_CDR)||(atsparms->cdr_run==BOTH_CDR))
   {
     bzero((char *) &osi_serv_addr, sizeof(osi_serv_addr));
     osi_serv_addr.sin_family = AF_INET;
     osi_serv_addr.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR);
     osi_serv_addr.sin_port      = htons(SERV_UDP_PORT);

     if((osi_sockfd=socket(AF_INET,SOCK_DGRAM,0))<0) {
          fprintf(stderr,"CDR_OSI: Problem with opening socket\n");
          return(1);
     }
     /* Bind any local address for us. */
     bzero((char *) &osi_cli_addr, sizeof(osi_cli_addr));
     osi_cli_addr.sin_family = AF_INET;
     osi_cli_addr.sin_addr.s_addr = htonl(INADDR_ANY);
     osi_cli_addr.sin_port       = htons(0);

     if(bind(osi_sockfd, (struct sockaddr *)&osi_cli_addr, sizeof(osi_cli_addr))<0)
     {
          send_rptmgr(STAT_FILE, 1,"CDR_OSI: Problem with bind\n");
```

```
            return(1);
        }
    } /* end of if */

●●●●●●

}

void cdr_done(int sig_num)
{

●●●●●●

    /* Clean up for socket communications with OSI server */
    if((atsparms->cdr_run==OSI_CDR)||(atsparms->cdr_run==BOTH_CDR) &&  osi_started)
    {
      msg.msg_hdr.msgtype = htons(OSI_STOP);
      strcpy(msg.data_buf.contents,"STOP");
      if(sendto(osi_sockfd,&msg,sizeof(msg),0,&osi_serv_addr,sizeof(osi_serv_addr))!=sizeof(msg))
      {
        send_rptmgr(STAT_FILE,1,"CDR_OSI: Problem sending STOP to OSI server\n");
      }
      close(osi_sockfd);
    }

●●●●●●

}
```

CDRMGR.C

```
main(argc, argv)
int argc;
char *argv[];
{

●●●●●●

  if (atsparms->cdr_run == BOTH_CDR || atsparms->cdr_run == OSI_CDR)
  {
    /* start socket handshake with OSI server  */
    if(commOSIserver())
    {
      send_rptmgr(STAT_FILE,1,"CDR_OSI: Trouble communicating with OSI server!\n");
      cdr_done(0);
    }
    else
      osi_started=1;
  }
  if (atsparms->cdr_run == BOTH_CDR || atsparms->cdr_run == CAMP_CDR)
  {
    /* start CAMP collection of CDRs */
    if (startCamp(&cdrMessage, &type))

    {
      send_rptmgr(STAT_FILE, 1, "Failed starting CAMP AMA CDR collection\n");
      cdr_done(0);
    }
  }
```

●●●●●●

```
/* Since WATT is alive and well, send OSI_RUNNING to OSI server */
if ((atsparms->cdr_run == BOTH_CDR || atsparms->cdr_run == OSI_CDR)   && osi_started)
{
    msg.msg_hdr.msgtype = htons(OSI_RUNNING);
    strcpy((char *)msg.data_buf.contents,"OSI_RUNNING");

    if(sendto(osi_sockfd,&msg,sizeof(msg),0, (struct sockaddr *)&osi_serv_addr, sizeof(osi_serv_addr))!=sizeof(msg))
    {
        send_rptmgr(STAT_FILE, 1, "CDR_OSI: Problem with sendto\n");
        return 1;
    }

}

while (!done)
{
 if (atsparms->cdr_run == BOTH_CDR || atsparms->cdr_run == OSI_CDR)
 {
  /* doing OSI CDR collection and got something in the socket */
    osi_cdr_rtn = handle_osi_cdr();
 }

  done = processMessages(&cdrMessage, &type);

  for (i=0;i<MAXCIRCUITS;i++)
  {
    if (cdrMode[i]==MODE_MATCH)   /* pending requests */
    {
      time(&currentTime);
      if (currentTime-timeout[i] >= atsparms->cdrto)
```

```
        {
          /* no match or partial match, send responses to CSI */
          send_rptmgr(STAT_FILE,2, "Timed out on CDR search.\n");
          sndTCNresponses();
          break; /* out of for */
        } /* if timed out */
      } /* if looking for cdr matches on this circuit */
    } /* for loop */
    nap(1000);
  } /* end while(!done) */

  cdr_done(0);              /* clean up and get ready to die! */
} /* end main */

/*****************************************************************************
 * ROUTINE:  handle_osi_cdr()
 * PURPOSE:
 *           This procedure reads the OSI CDRs that are being read from a socket.
 *
 * Possible return values are:
 * osi_cdr_rtn = 0          Problem - get out gracefully
 * osi_cdr_rtn = 1          Nothing to read from the socket - no action
 * osi_cdr_rtn = RECV_OSI_CDR   An OSI CDR has been received
*****************************************************************************/

int handle_osi_cdr()
{
  int       chars_read;
  OSI_MSG     recvmsg;
  CAMP_MSG    campmsg;
  int i;
  unsigned short tmpval;
```

```
memset(&recvmsg, 0, sizeof(recvmsg));
if(non_block_read())
{
 chars_read = recvfrom(osi_sockfd,&recvmsg,sizeof(recvmsg),0,  (struct sockaddr *)0,(int *)0);

 if (chars_read<0)
 {
        send_rptmgr(STAT_FILE,1, "CDR_OSI: recvfrom error\n");
        return(0);
 }

 if(chars_read>0)
 {
   send_rptmgr(STAT_FILE,1, "CDR_OSI: Message Type:%u, # chars read =%d\n",
                                    ntohs(recvmsg.msg_hdr.msgtype), chars_read);

   /* Make sure you don't have FTAM problems on OSI server side */
   if(ntohs(recvmsg.msg_hdr.msgtype)==OSI_FTAM_ERROR) /* problem getting CDR */
   {
        send_rptmgr(STAT_FILE,1, "CDR_OSI: FTAM transfer error\n");
        return(0);
   }
   /* take the CDR from the socket and put it into CDR's message queue
    as if it came in from a camp_if process */

   memset((char *)&campmsg,0,CAMP_MSG_SIZE);
   campmsg.msg_hdr.msgclass= CDR_CLASS;
   campmsg.msg_hdr.msgtype= ntohs(recvmsg.msg_hdr.msgtype);
   /* it better be CDR_CAMP_RSP */
   campmsg.msg_hdr.sender_id= CDR_QKEY;
   /* copy the CDR over */

   if (chars_read > CAMPBUF_SIZ)
```

```
    {
        send_rptmgr(STAT_FILE, 1,
        "WARNING: socket data of size %d would overrun CDR data_buf, restricting to %d\n",
          chars_read, CAMPBUF_SIZ);
        chars_read = CAMPBUF_SIZ;
    }
    campmsg.byte_count = chars_read;
    memcpy(campmsg.data_buf, recvmsg.data_buf.contents, chars_read);
    send_atsmsg(CDR_QKEY, (unsigned char *)&campmsg, sizeof(campmsg), 0);
    return(RECV_OSI_CDR); /* CDR read */

   } /* if chars_read > 0 */
  } /* end of non_block_read */
  return(1); /* no chars read */
}

/*************************************************************************
 * ROUTINE:  commOSIserver()
 * PURPOSE:
 *          Initiate FTAM transfer of OSI CDRS. This procedure performs initial
 *          handshake with the OSI server
 *
 * AUTHOR: Sunil Fotedar
*************************************************************************/
int commOSIserver()
{
    int    n, i, j;
    OSI_MSG msg, recvmsg;
    char   buffer[300], temp[50];
    char dumpbuf[1000];

/* Send the Filtering Parameters to OSI server */
    memset((unsigned char *)&recvmsg, 0, sizeof(OSI_MSG));
```

```
memset((unsigned char *)&msg, 0, sizeof(OSI_MSG));
strcpy(msg.switchname, atsparms->switchname);
strcpy(msg.filter_params.swgeneric, atsparms->swgeneric);

/* remove blank space from cdrckts string */
strcpy(temp, atsparms->cdrckts);
cir_trans(temp, 50);
for (i=0,j=0; i < strlen(temp); i++)
{
      if( isgraph((int)temp[i]) )
      {
            temp[j]=temp[i];
            j++;
      }
} /* End for (isgraph) */
temp[j]=0; /* Make sure there is a NULL on the end */
strcpy(msg.filter_params.cdrckts, temp);

msg.msg_hdr.msgtype = htons(OSI_START);
sprintf(buffer,"%s %s %s",
            msg.switchname,
            msg.filter_params.swgeneric,
            temp);
strcpy(msg.data_buf.contents,buffer);

if(sendto(osi_sockfd,&msg,sizeof(msg),0,&osi_serv_addr,sizeof(osi_serv_addr))!=sizeof(msg))
{
      send_rptmgr(STAT_FILE, 1, "CDR_OSI: Problem with sendto\n");
      return 1;
}

send_rptmgr(STAT_FILE, 3,"CDR_OSI: Message sent: %s\n", msg.data_buf.contents);
```

```
    /* Try to read OSI_START_ACK from OSI server */
    n = recvfrom(osi_sockfd,&recvmsg,sizeof(OSI_MSG),0,(struct sockaddr *)0, (int *)0);

    if (n<0) {
        send_rptmgr(STAT_FILE,1, "CDR_OSI: recvfrom error %d\n", errno);
        return 1;
    }

    send_rptmgr(STAT_FILE,1, "CDR_OSI: Message Type:%u, Message received:%s\n",
                                ntohs(recvmsg.msg_hdr.msgtype), recvmsg.data_buf.contents);

    if(ntohs(recvmsg.msg_hdr.msgtype)!=OSI_START_ACK)
    {
        send_rptmgr(STAT_FILE,1, "CDR_OSI: OSI_START_ACK not received from OSI server\n");
        return 1;
    }
    return 0; /* everything worked fine */
}
/*************************************************************************
 * ROUTINE:  non_block_read()
 *
 * PURPOSE:
 *     This function performs a non-block read operation on an open socket.
 *
 * AUTHOR: Sunil Fotedar
 *
*************************************************************************/

int non_block_read()
{
    int    nfds;
    fd_set  readfds;
    struct  timeval tv;
```

```
FD_SET(osi_sockfd,&readfds);

/* Set the timeout */
tv.tv_sec = 0;
tv.tv_usec = 0;

nfds = select(osi_sockfd+1,&readfds,NULL,NULL,&tv);
 if(nfds>0) {
       if(FD_ISSET(osi_sockfd,&readfds))
       {
             return 1; /* something to be read */
       }
}

return 0; /* nothing to be read */}
```

OSI_CDR.H

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

#define SERV_UDP_PORT   4567
#define SERV_HOST_ADDR  "201.23.2.106" /* host addr for server sun1209 */
#define MAXLINE             512

/* Message Types */
#define OSI_START           10
#define OSI_STOP            11
#define OSI_START_ACK       12
#define OSI_START_NAK       13
#define OSI_RUNNING         14
#define OSI_FTAM_ERROR      15      /* whenever there is an FTAM problem */

#define STRT_SWGENSIZ       30      /* sw generic load string limit */
#define STRT_CKOSIZ         50      /* circuits for protocols string limit*/

#define RECV_OSI_CDR    2                       /* Yes, we have received a CDR to proces
s */
#define BILLREC_SIZE    32*2                    /* Number of WORDS in a Billing Record      */
#define EXT_BILLREC_SIZE    (BILLREC_SIZE * 2)  /* Number of words in an Extended Billing Record      */
```

```
extern int errno;

typedef struct {
      ELEMENT  origckt, origtime;
      unsigned char swgeneric[STRT_SWGENSIZ];
      unsigned char cdrckts[STRT_CKOSIZ];
} FilterParams;

typedef struct
{
 MSG_HDR      msg_hdr;          /* message header (routing) */
 unsigned char switchname[10];
 FilterParams  filter_params;
 union
 {
   unsigned short record[128];
   unsigned char contents[256];
 } data_buf;
}OSI_MSG;

int    osi_sockfd;
char   hostname[64];
struct  hostent *hp;
struct  sockaddr_in    osi_serv_addr, osi_cli_addr;
```

Sample STARTUP.ATS File

A sample **startup** file for DOE requesting OSI CDRs from switch MCI2:

```
USERNAME=ATS
PASSWORD=STA
ATSNAME=doe
SWITCH=mci2
FEATURESET=B
SWGENERIC=108-32.10.00
CKO=OCK:1920
CDRCKTS=OCK-TCK,1922,1923
INBAND=n
MMI=n
CDR=OSI
DOSS=n
CDRTO=50
CPRETRY=0
ERR_SHUT=n
```

●●●●●●