# HXECOM
# Network Exerciser
# User's Guide

James D. Miles

May 2, 1995

## Contents

## List of Figures

## 1 Overview

HXECOM is an HTX exerciser that conforms to the requirements of "HTX Hardware Exerciser Programmer's Guide." HXECOM is intended to be a high stress exerciser for testing communication adapters in the HTX environment. It supports an arbitrary number of adapters and host machines. The objective is to fully utilize the bandwidth available on the adapter and on the physical transmission medium. It is intended to be able to write and read to all network layers supported by the adapter hardware and driver. The current version supports testing using the TCP/IP or UDP/IP socket interface [1].

The key features of HXECOM are:

- Communication channel between hosts. This requires that one adapter or integrated network device on each host be dedicated to test coordination and synchronization [2].
- An equivalent read/write thread at the remote site for each local write/read thread.
- Automatic spawning and configuration of test threads as implied by the rule file.
- Robust recovery and identification of errors.
- Ability to shutdown all network activity quickly if rule-specified errors happen.
- Ability to test networks between AIX and NT. Big endian and little endian translations are handled as needed [3].

[1] HXECOM can be compiled to support 3.2 style processes or 4.1 threaded processes. For the rest of this document, reference to *threads* may be replaced with *process* for the nonthreaded version.

[2] Sometimes it is desired to only use only 1 network. This is possible with constraints. It should only be done if testing at the TCP layer and the adapter is relatively stable. If the network goes down, it may not be possible to obtain the usual diagnostic messages about the failure.

[3] Current path to NT binaries is /afs/austin/projects/htx/htxnt.src.

## 2 Executable Files

This exerciser consists of the following executable files:

**hxecom** You will have this file for non-threaded AIX 3.2 style processes.

**hxecom_t** This is the threaded version and is only available for AIX 4.1 and later.

**pscheck.sh** This is a bourne shell script used by above executables.

**hxecom2** This is the coordinator process discussed in Section 9.

The minimum set of files for a working exerciser is hxecom or hxecom_t, pscheck.sh, hxecom2, a pattern file and a rule file. If you are running under the HTX lpp, these should be installed. A bare bones rules file and mdt file can be found in Appendix C.

If running stand-alone as explained in Section 6, all of these files may be put in the same directory. This exerciser requires root user authority.

# 3 Input files

The following subsections refer to fields defined in Appendix B and to exit codes defined in Appendix A. If you are not familiar with rules file fields, read the above appendix sections first. The rules file "rules/reg/hxecom/default" contains an example utilizing multiple stanzas.

## 3.1 Rules File

The rules file name is always specified as an input variable on the command line (see Section 6).

The *rules file* is used to specify the various customizable parameters. These parameters consist of 2 types: parameters which can only be set once and parameters which are reset with each stanza of the rules file. The parameters that can be reset in each stanza are BUFMIN, BUFMAX, BUFINC, NUM_OPER, WRITE_SLEEP, WRITE_AHEAD, ACK_TRIG and IO_ALARM_TIME.

The rules file consist of stanza(s) separated by a blank line. Each stanza is made up of lines consisting of definitions of various parameters.

## 3.2 Pattern File

The *pattern file* specifies the stream of characters that is used to initialize the write buffer. The size of the file can be less than or greater than the write buffer size (see NUM_CHARS). The pattern is either repeated or truncated till it fits the write buffer. The pattern file is specified as a field PATTERN_ID in the rules file.

## 3.3 Input File Interactions

### 3.3.1 Device Name and TESTNET_NAME

The first argument argv[1] is normally /dev/tok0, /dev/ent0, et cetera(see Section 6). It should specify the device being tested. The TESTNET_NAME should map to the IP address that is bound to the network and device implied by argv[1].

If LAYER=UDP or LAYER=TCP is specified in the rules file, then TESTNET_NAME represents the actual device tested. If argv[1] specifies a different device then the information messages may appear to be inconsistent. This inconsistency can be avoided by making sure that the correct device is specified in the mdt file or on the command line. Failure to be consistent will also result in the wrong data being saved when using the "hxecom.save" script.

### 3.3.2 Rules Files

When HXECOM is invoked as specified in Section 6, a rules file is specified. This means that 2 potentially different files might be used locally and remotely. However, the each reader thread uses the local rule file and provides this same rule file to its remote writer thread.

All keywords in the rules file are converted to upper-case when read. The following keyword-values are case-sensitive: RULE_ID, PATTERN_ID, COMNET_NAME, and TESTNET_NAME.

### 3.3.3 Multiple Stanzas

If multiple stanzas are used, parameters PATTERN_ID, COMNET_NAME, TESTNET_NAME, COM_STREAM_PORT, COM_DGRAM_PORT, REPLICATES, MASTER, LAYER, OPER, TCP_NODELAY, SO_LINGER and SHUTDOWN_FLAGS should only be defined in the first stanza. If these parameters are defined multiple times, error messages will be generated.

### 3.3.4 NUM_OPER and Buffer size

The default rules provide a reasonable update time for devices such as Ethernet and Token ring. However, if testing slower devices such as tty's using SLIP, then the devices may appear to be hung when the update time is just too long. The algorithms in Appendix B for read and write process should indicate why this can happen.

NUM_OPER can be adjusted in this case by shutting down the test in the normal fashion, determining the number of read/writes for the test time, and adjusting NUM_OPER accordingly. The number of reads or writes can be found in either the file /tmp/htxstats or /tmp/htxstats2.

### 3.3.5 Multiple Test Networks and Port Numbers

If multiple test networks are installed and the rule files refer to the same COM_DGRAM_PORT and COM_STREAM_PORT, then test connections will be formed between test networks. This is probably not what was intended (see Section 8.2).

### 3.3.6 OPER and MASTER rules

At least 1 node must contain a master. A master can form network connections with any other node running HXECOM but a non-master can only form connections with a master.

To form a connection, one side must support reading and the other side must support writing. If both nodes have OPER=R or OPER=W, no connection will form.

## 4 Output Files

In addition to the files normally associated with HTX, the following files are also generated:

**/tmp/htxstats2** This file is similar to /tmp/htxstats and is mentioned in Section 6.3.2.

**/tmp/htxcom.config** This file documents all hosts that participated in the testing and can be used with scripts to retrieve information. See Appendix D.

## 5 Environmental variables

HXECOM recognizes the following environmental variables: HTXPATTERNS and HTXRULES. If HXECOM can't locate the rules file or the pattern file by the specified path search (see Section 6.1), then it will try the path specified by the appropriate environmental variable.

Example: To run HXECOM on a machine that doesn't have the HTX lpp, set the environmental variables. If the rules file is in directory $HOME/rule and the pattern file is in directory $HOME/test then set the environmental variables as follows:

```
export HTXRULES=$HOME/rule
export HTXPATTERNS=$HOME/test
```

Once these variables are set, HXECOM can be invoked as described in Section 6.

## 6 Starting HXECOM

If HXECOM is invoked without any arguments it responds with:

```
usage: hxecom /dev/xxxxx [ REG EMC] rule_path
  or: hxecom /dev/xxxxx   OTH  rule_path
```

OR

```
usage: hxecom_t /dev/xxxxx [ REG EMC] rule_path
  or: hxecom_t /dev/xxxxx   OTH  rule_path
```

/dev/xxxxx represents the device to be tested[4]. In Section 8, /dev/xxxxx would be the Test_Devices (tok1, tok0, or tok1) for each of the respective machines (Lab20, IO21, and Lab22).

The first call with [REG] or [EMC] must be started under HTX (see Section 6.2).

The second call with OTH is the normal startup when not running under HTX (see Section 6.3).

### 6.1 Rules File Name Path Searching

HXECOM takes the rule_path it is given and searches for the file as follows:

1. try <<rule_path>>,
   IF the file is not found AND <<rule_path>> doesn't start with a '.' or a '/' try item 2 and if necessary try item 3.

2. IF type is EMC[5]
   try ../rules/emc/hxecom/<<rule_path>>
   ELSE
   try ../rules/reg/hxecom/<<rule_path>>

3. try $HTXRULES/<<rule_path>> where $HTXRULES is an exported environmental variable as explained in Section 5.

[4]In the NT environment this argument only has meaning for the identification of test device. A good choice would be to use the name of the test network

[5]This item is included for compatibility with AIX version 3.x only.

## 6.2 Starting under HTX

This is the first option referred to in Section 6 and is covered in "HTX User's Guide."

Note: Starting up HXECOM on one machine will not cause any activity when testing at the TCP or UDP layer. HXECOM must be running on every machine that is to be tested. If HXECOM is only started on one machine, eventually the HTX hang monitor will print hung messages. These messages will stop as soon as HXECOM is started on another machine. Section 9 should provide more insight into this behavior.

## 6.3 Starting Stand-alone

This is the second option referred to in Section 6. This mode can be used anytime it is desired to run HXECOM outside of the HTX environment. This mode provides all of the capability present when running under HTX.

Note: Starting up HXECOM on one machine will not cause any activity when testing at the TCP or UDP layer. HXECOM must be running on every machine that is to be tested. Section 9 should provide more insight into this behavior.

### 6.3.1 Stand-alone Messages

All messages are written to the screen. Messages with severe errors that would normally be written to the htxerr log are written to stderr. All other messages are written to stdout. In addition to the usual messages generated under HTX, the exerciser writes additional information messages to stdout.

### 6.3.2 Stand-alone Statistics

When an exerciser is terminated, it writes its summary statistics to the file "/tmp/htxstats2." This file is only written when the exerciser is terminated using the methods described in Section 6.3.3 below.

### 6.3.3 Killing Stand-alone Exercisers

When it is time to terminate a stand-alone exerciser, it is recommended that it be sent a SIGTERM signal[6]. This will allow the exerciser to cleanup properly. There will be multiple processes started. By using the command
`ps -ef|grep hxecom`
the parent exercisers can be determined. The PPID field of the children will point to the parent exerciser. The parent exerciser when running stand-alone will show a PPID of 1. If a SIGTERM signal is sent to a parent exerciser, all child processes should exit within a short time. If a child exerciser is hung in a system call, it may take an indeterminate amount of time. Before using a SIGKILL signal or rebooting, make sure all exercisers on other hosts have also been terminated by this method. Also try sending a SIGTERM signal directly to the child process.

A SIGTERM signal can be sent as follows:
`kill PID_of_process`

A SIGKILL signal can be sent as follows:
`kill -9 PID_of_process`

## 6.4 Fails to Start/No Activity

If the exerciser is invoked by any of the methods above and test connections fail to happen or connections fail, check the following:

- Has HXECOM been started on more than 1 machine?
- If testing slow devices, see Section 3.3.4.
- Make sure all nodes can be "pinged" as outlined in section 8.
- Check that all old processes have been terminated (see section 6.3.3). A limited number of shutting down and bringing up one host's exercisers should work without without shutting down the remaining test nodes. However if the shutdown had problems, this may not be the case.

[6] In the NT environment this can be accomplished by executing a Ctrl-C in the MSDOS window of the running exerciser. In case of a hang, close the MSDOS window for the exerciser that doesn't want to terminate.

- Make sure that the COMNET_NAME and TESTNET_NAME specified in the rules are correct (see section 8).
- Make sure that the COM_STREAM_PORT and COM_DGRAM_PORT are the same in the corresponding rule file(s) on each host (see section 8.2).
- Make sure that the netmask for the comnet is Class C. Issue an "ifconfig comnet_interface" command and verify that the netmask field and broadcast field are as follows:
  `netmask 0xffffff00 broadcast XXX.XXX.XXX.255`
- Make sure that the device represented by COMNET_NAME is not being tested — that it isn't represented in an mdt stanza. The only exception is if you have specifically used the TESTNET_NAME for the COMNET_NAME as outlined in footnote 2.
- Check the system error log for pertinent information.
- See Section 3.3.6.
- If the above items don't identify the problem, then use any error messages printed to the screen or the the message log (htxmsg) to identify the problem as explained in section 7.

### 6.5 Memory Cleanup

Because of the way UNIX processes can be killed and the fact that the HTX supervisor sends SIGKILL signals, it is possible that the shared memory and semaphores do not get removed when the exercisers exit. Normally this should not be a problem in that the first exerciser on a restart will identify and correct the problem.

## 7 Exit Numbers

Exit numbers are encoded in the messages. The first 4 digits of the err= in the message is the exit number in HEX. The last four digits contain the errno if appropriate. For the line below, the exit number is 0001.

```
/dev/MPSF63S0     Dec 22 08:25:11 1993 err=00010000 sev=7 hxecom
```

Normally the messages contain enough information to determine a course of action. Information messages may also be written to the message log which can help determine the error's nature. While looking up the exit number, be sure to note what was happening prior to the exit. Exit numbers can be cross-referenced to Appendix A for additional information.

# 8 Setup/Tutorial

## 8.1 Setting Up a Single Test Network

This section provides the steps that would normally be necessary to setup a 3 machine network.

The table below contains sample data required to start configuring a network.

The *HostName* column is not useful in configuring for test except to indicate which machine we are referring — it may be the same as other network names in the table but this is not a requirement.

The *Com_Device, Com_Name, and Com IP* columns refer to the 3 node communication network that is required. This network and its associated adapters should not be tested. Note that for IO21 machine the Com_Name is not IO21 but Lab21. Only use Lab21 in the setup. This is because each of the names has an IP address bound to it. If IO21 is used, we will attempt to test the wrong network.

The *Test_Device, Test_Name, and Test IP* columns refer to the 3 node communication network under test.

| HostName | Com_Device | Com_Name | Com IP | Test_Device | Test_Name | Test IP |
|---|---|---|---|---|---|---|
| Lab20 | tok0 | Lab20 | 192.178.100.20 | tok1 | test20 | 193.178.100.20 |
| IO21 | tok2 | Lab21 | 192.178.100.21 | tok0 | test21 | 193.178.100.21 |
| Lab22 | tok0 | Lab22 | 192.178.100.22 | tok1 | test22 | 193.178.100.22 |

Figure 1: Network Table

To run HXECOM, the network must be functional. This can be indicated by the abilty to ping all remote devices connected to the local one. For the above table, the following commands must work:

```
From Lab20: ping Lab20, ping Lab21, ping Lab22
            ping test20, ping test21, ping test22
```

To setup Lab21:

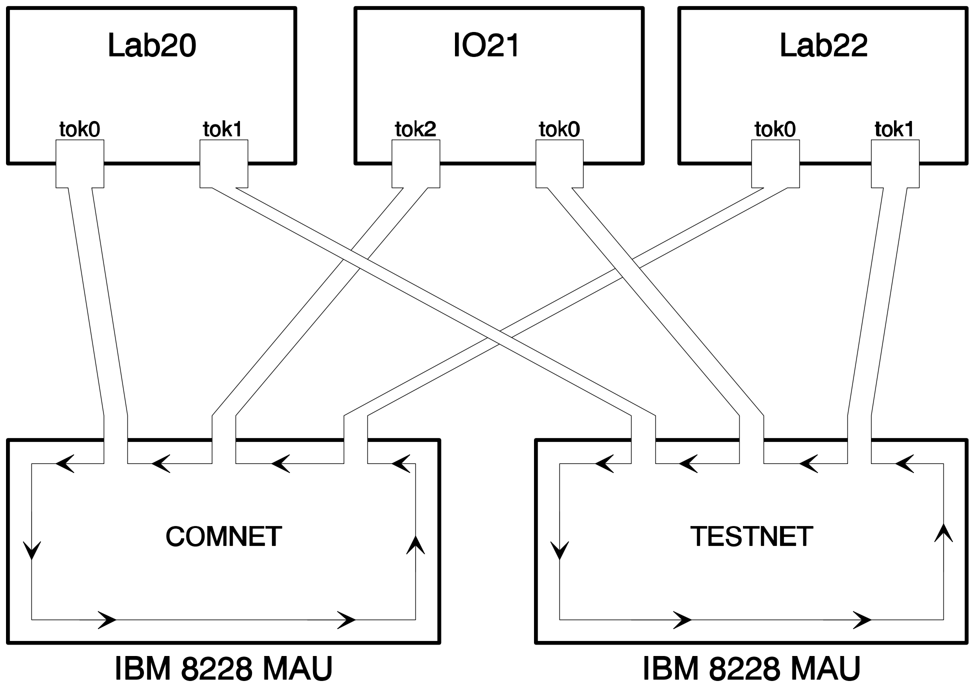


Figure 2: Network Setup

*smit* tcpip
*select minimum configuration*
*select network interface* tr2
*Hostname* Lab21
*Internet Address* 192.178.100.21
*Network Mask* 255.255.255.0
*Nameserver* - leave this section blank
*Ring speed* XX
*Start now* yes
Note: Lab21 is the name associated with the Com_Net not with the actual hostname which may or may not be the same.

Add the following line to /etc/hosts
192.178.100.21 Lab21
The easiest, safest method is to add a line for each comnet and testnet node

to the /etc/hosts file. Then replicate the additions to this file to every host involved in the testing.

Copy the default rules file to "another_rule_name" and modify the following lines:

```
comnet_name  = Lab21
testnet_name = test21
```

Change any other appropriate parameters.

Copy the mdt.all file to "another_mdt_name" and modify/add the stanza for the device tok0. Make sure there is no stanza for the Com_Device tok2.

The modified stanza should include the following lines for the process version of HXECOM:

```
tok0:
  HE_name   = "hxecom"
  reg_rules = "hxecom/another_rule_name"
```

The modified stanza should include the following lines for the threaded version of HXECOM:

```
tok0:
  HE_name   = "hxecom_t"
  reg_rules = "hxecom/another_rule_name"
```

Note that the only difference between the above stanzas is in the 'HE_name =' field. There is no restriction on which stanzas/nodes specify the process or threaded version of HXECOM.

Once the above steps have been performed on all 3 hosts, testing may begin as outlined in Section 6.

## 8.2 Setting Up Multiple Test Networks and SLIP

Additional test networks may be added. Each communication adapter added must represent an additional network if testing at the TCP or UDP layer. Each adapter/network added would require an additional stanza in the mdt file and a new rules file.

The default behavior of HXECOM is to try to form connections between each local adapter and all remote adapters. This is probably not the behavior desired. To form connections only between adapters on the same network requires modification of the rule parameters COM_DGRAM_PORT and COM_STREAM_PORT. Any port numbers greater than 5000 will suffice. However, be aware that potential conflicts may happen if other applications have assigned port numbers.

A satisfactory assignment scheme would be:

| | COM_STREAM_PORT | COM_DGRAM_PORT |
|---|---|---|
| Network/SLIP 1 rules | 5100 | 5101 |
| Network/SLIP 2 rules | 5102 | 5103 |
| Network/SLIP 3 rules | 5104 | 5105 |

etc.
Where SLIP represents a connection between any 2 tty ports.

## 9 Design/Functionality of HXECOM

This section contains an overview of the basic design. For more detail, review the source code.

When an exerciser is started by method described in Section 6, a separate unique *coordinator* process is created to handle messages at the ports specified in the rules file. This coordinator process broadcasts on the COMNET a REMOTESERVER message about the new HXECOM exerciser (see Figure 3).

The coordinator process keeps 3 tables — a list of local exercisers, a list of remote exercisers, and a list of coordinators. When a REMOTESERVER message is received, the coordinator scans the remote exerciser list. If the exerciser address is found in the list, the message is discarded. If it is not in the list, all local exercisers in the coordinators list are notified of the new remote server (see Figure 4).

If the coordinator forwards a REMOTESERVER message to an exerciser, the exerciser will look at other information in the message and react accordingly. To form a test connection, either the remote server or the local server must be a master. In addition, the local server rules must support writers and the remote server must support readers (see Figure 5). It is up to each of these spawned writer threads to communicate with the remote server. Figure 6 shows the communications between each write thread and the remote server.

If TCP/IP is used as the test protocol, the writer writes as fast as allowed since this protocol is reliable. However, when a UDP/IP is used, a method to limit overruns is needed. The rule file parameter WRITE_AHEAD serves this purpose. As can be seen from Figure 7, the total outstanding packets can be no greater than the number specified by the WRITE_AHEAD parameter. This parameter when combined with other network parameters can be used to detect abnormal data discards. UDP/IP testing should only be performed by testers willing to invest the time required to understand the interaction between network parameters.

```
Rule file
   COMNET_NAME=host1
   COMSTREAM_PORT=5100
   COMDGRAM_PORT=5101

/etc/hosts
   host1     192.178.100.1


Nomenclature
   xxx.xxx.xxx.xxx.pppp

   xxx.xxx.xxx.xxx = dotted decimal IP address
   pppp = 4 digit port number
```

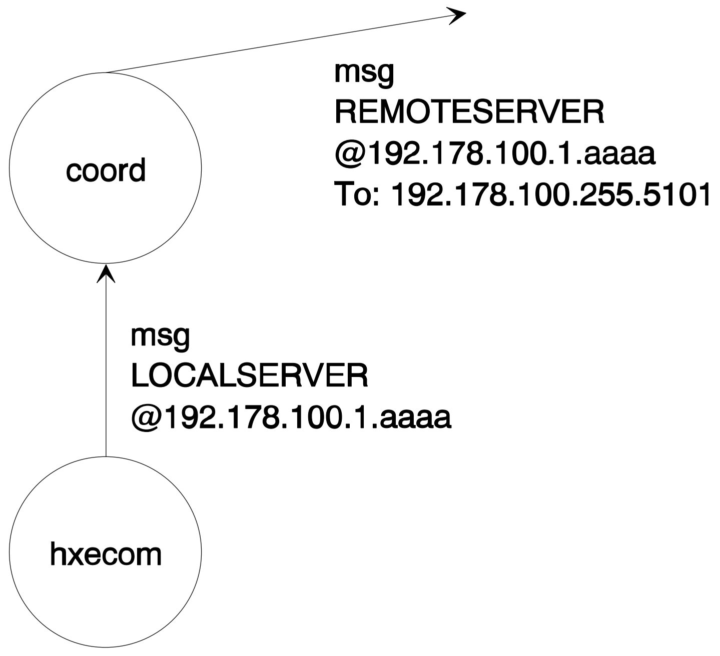


Figure 3: Start Up Broadcast

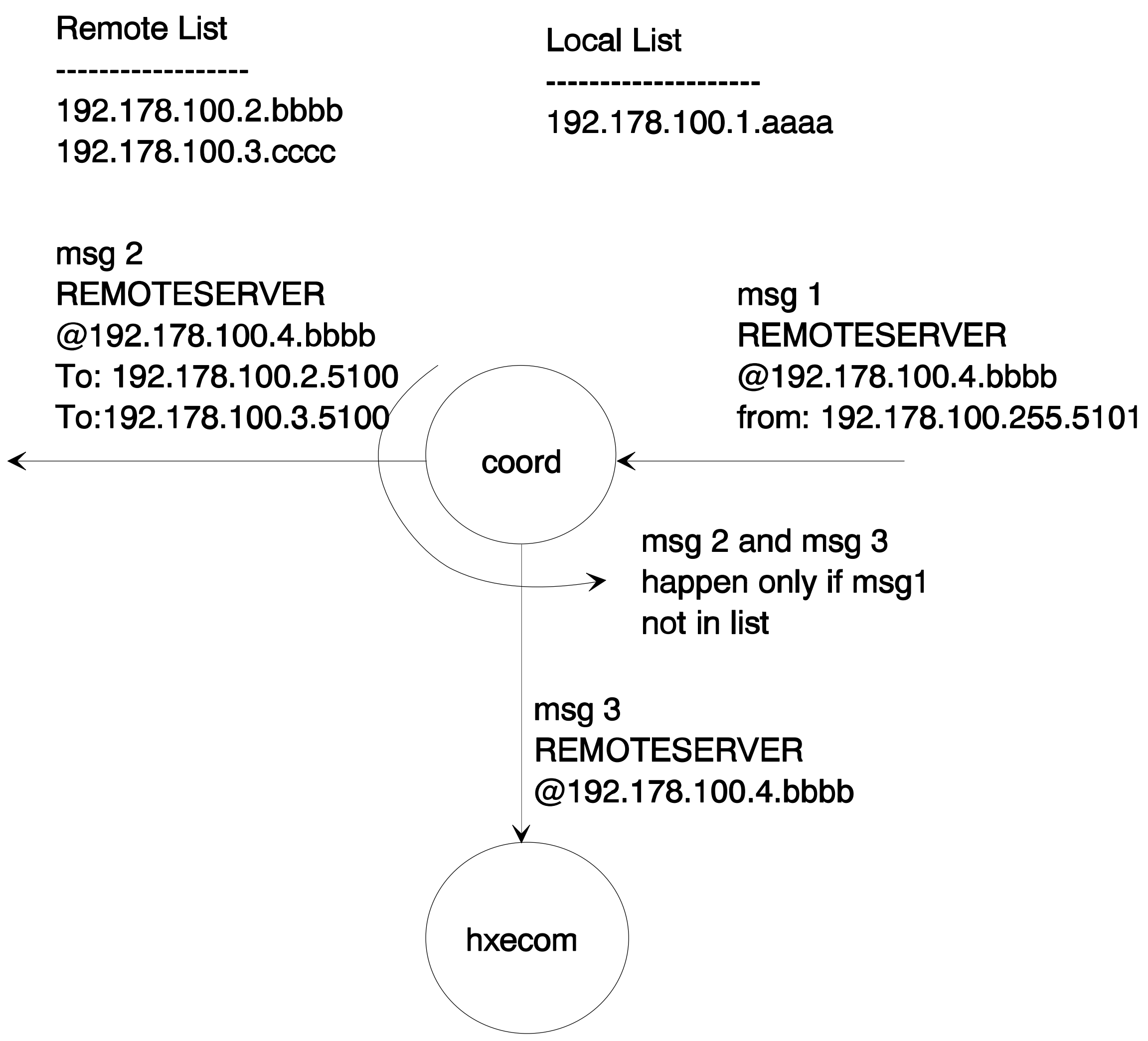


Figure 4: Receiving a Broadcast

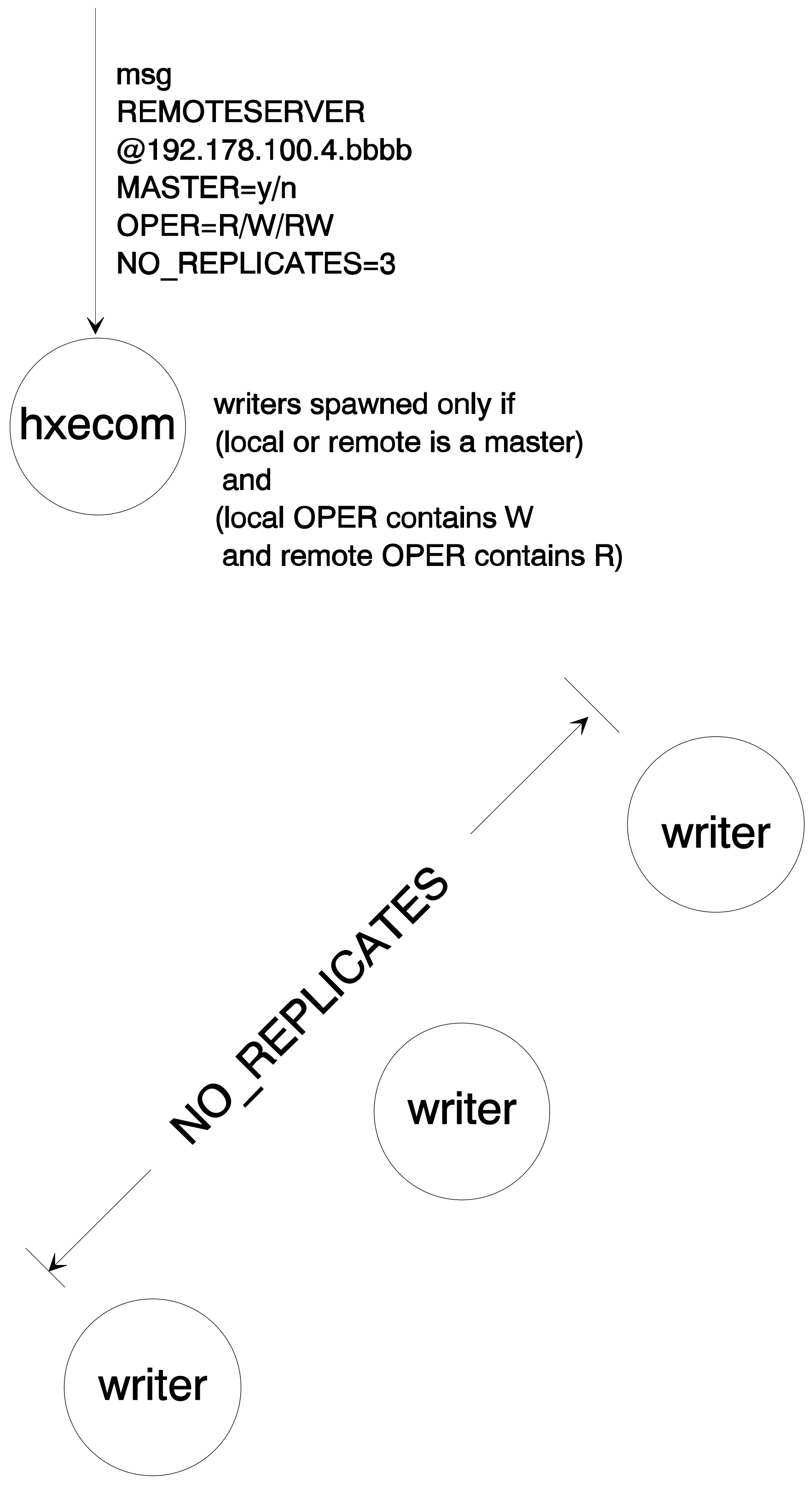


Figure 5: Spawning Writer Threads

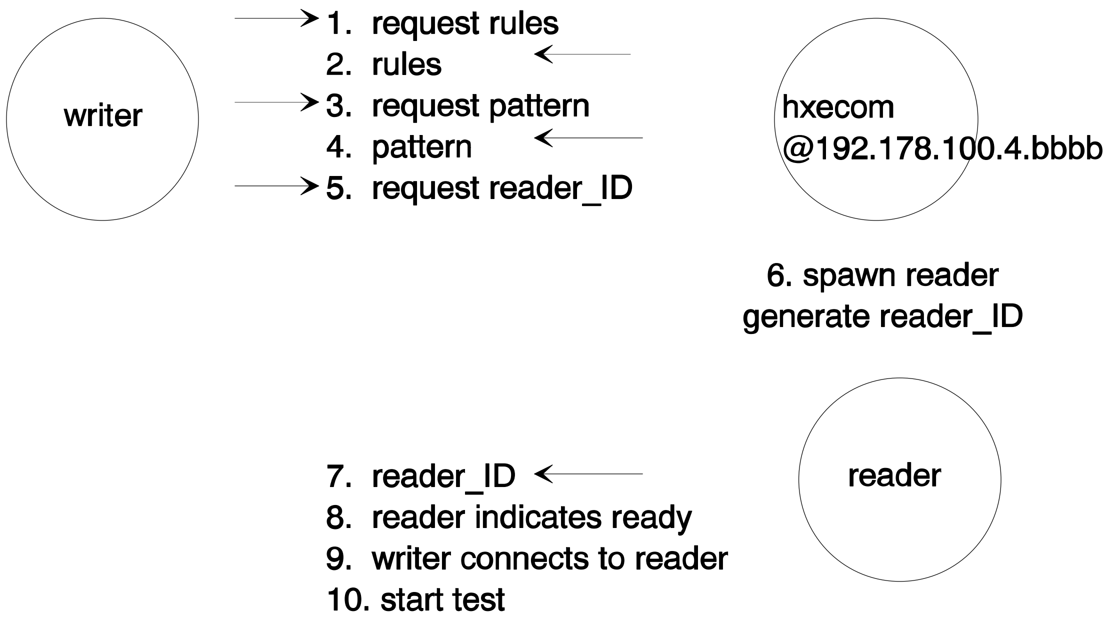
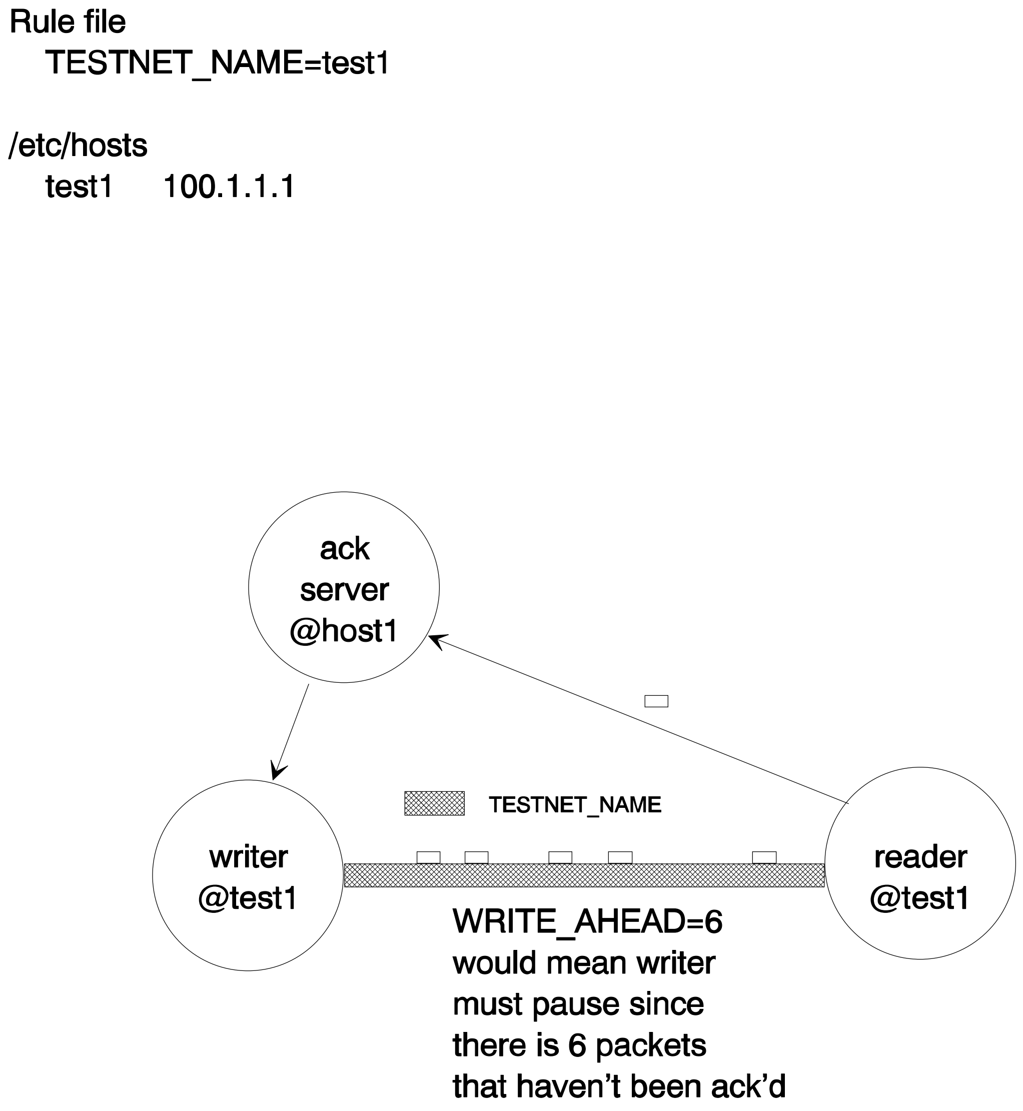


Figure 6: Reader/Writer Initialization



```
Rule file
   TESTNET_NAME=test1

/etc/hosts
   test1     100.1.1.1
```

Figure 7: Flow Control with UDP

## 10 Limitations/Exceptions

Testing at the device driver level has not been implemented yet.

Only a class C network can be used for the communications network. Netmask should be set to 255.255.255.0 (0xffffff00).

If a test machine is connected to a nameserver, timeouts may be encountered during the connect operation.

Because of IP constraints and the fact that the device driver level interface is not implemented, testing between adapters on the same machine is not possible.

Halting a device through the HTX interface may cause threads connected to the halted thread to timeout. This will remain an unsupported feature until HTX provides a better interface for handling halts.

Because of the nature and intent of this exerciser, stand-alone mode doesn't exit but continues indefinitely.

## A Exit Codes

These exit numbers are in decimal. The numbers extracted as explained in Section 7 are in hexadecimal.

```
/*****************************************************************************/
/* When the htxerr file is referred to below and you are testing in          */
/* standalone mode, we are refering to stderr.                               */
/*****************************************************************************/

/*****************************************************************************/
/* An invalid number of arguments were passed at the invocation of "hxecom".  */
/* Correct invocation of hxecom can be displayed by executing hxecom without  */
/* any arguments.                                                            */
#define EX_INV_ARGV     1
/*****************************************************************************/

/*****************************************************************************/
/* There was an error reading the rules file.                                */
/* Specific information should have been printed in the htxerr log file.     */
#define EX_RULEREAD1    2
/*****************************************************************************/

/*****************************************************************************/
/* The exerciser may have been unable to generate a valid pattern file name.  */
/* Check the following:                                                      */
/* htxerr message  - There are more than a few failure modes and this will    */
/*                   help isolate problem.                                   */
/* The file name   - Specified in rule file and should be a valid file in the */
/*                   pattern directory. The name must be all upper-case       */
/*                   letters.  If the file name contains some lower-case      */
/*                   letters, rename the file.                               */
/* The path      1 - Path will be assumed to be part of the file name.        */
/*               2 - If file name begins with a '.' or a '/'. no other paths  */
/*                   will be tried (This is hard to do with a limit of 8      */
/*                   characters for the filename.)                           */
/*            or 2 - Next a hardwired path '../pattern' will be tried.        */
/*               3 - Last, path HTXPATTERNS from environment will be tried.   */
#define EX_RULEREAD2    3
/*****************************************************************************/

/*****************************************************************************/
/* The file had been opened previously, check the htxerr message for clues.   */
#define  EX_RULEREAD3   4
/*****************************************************************************/
```

```
/******************************************************************************/
/* Device passed into hxecom on the command line was invalid.                */
#define EX_INVDEV       6
/******************************************************************************/

/******************************************************************************/
/* Either the paging volume is too small, not enough memory, or too many     */
/* processes/threads are running on this machine.                            */
/* "lsps -a" will show amount of paging space.  "lsdev -C|grep mem" will     */
/* show total memory in machine.                                             */
#define EX_SIGDANGER    7
/******************************************************************************/

/******************************************************************************/
/* Save htxerr log and any aditional information.                            */
#define EX_FORK0        10
#define EX_FORK1        11
#define EX_FORK2        12
#define EX_FORK3        13
#define EX_FORK4        14
#define EX_FORK5        15
#define EX_FORK6        16
#define EX_FORK7        17
#define EX_FORK8        18

#define EX_THREAD1      21
#define EX_THREAD2      22
#define EX_THREAD3      23
#define EX_THREAD4      24
/******************************************************************************/

/******************************************************************************/
/* The exerciser was unable to generate a valid rule file pathname.  Check   */
/* the following:                                                            */
/* htxerr message  - There are more than a few failure modes and this will   */
/*                   help isolate problem.                                   */
/* The file name   - If specified in mdt test file, it should be a valid file */
/*                   in the rule directory.  If running standalone, the rule */
/*                   file name is specified as the 3rd argument on the       */
/*                   command line.                                           */
/* The path      1 - Path will be assumed to be part of the file name.       */
/*               2 - If file name begins with a '.' or a '/'. no other paths */
/*                   will be tried.                                          */
/*            or 2 - A hardwired path '../rules/reg/hxecom' will be tried    */
/*                   if in "REG" mode or "OTH" mode.  If in "EMC" mode,      */
/*                   '../rules/emc/hxecom' will be tried.                    */
/*              3 - Last, path HTXRULES from environment will be tried.      */
#define EX_FNAME        25
/*****************************************************************************/

/*****************************************************************************/
/* Unable to lookup the IP address for the name represented by COMNET_NAME in */
/* the rule file.  Check :                                                   */
/*     - COMNET_NAME (TESTNET_NAME) was correct in rule file                 */
/*     - COMNET_NAME (TESTNET_NAME) is represented in the /etc/hosts file    */
/*     - If a nameserver is active, can it resolve this name? Use of         */
/*       nameserver for testing is not recommended.                          */
/*     - Review messages in htxerr and htxmsg for additionl information.     */
#define EX_GETH1        26
#define EX_GETH2        27
/* This is the same as EX_GETH1 above except it may be for TESTNET_NAME.     */
/* Review the error message to determine which network has the problem and   */
/* follow the advice for EX_GETH1 above.                                     */
#define EX_GETH3        28
/*****************************************************************************/

/*****************************************************************************/
/* Unable to bind IP address represented by COMNET_NAME in the rule file.    */
/* Check:                                                                    */
/*     - IP address in /etc/hosts or nameserver.                             */
/*     - IP address must be a valid address on THIS host.                   */
/*     - Review messages in htxerr and htxmsg for additionl information.     */
#define EX_BIND1        29
/* This is the same as EX_BIND1 above except it may be for TESTNET_NAME.     */
/* Review the error message to determine which network has the problem and   */
/* follow the advice for EX_BIND1 above.                                     */
#define EX_BIND5        30
/*****************************************************************************/

/*****************************************************************************/
/* Was unable to make a socket connection to address printed in htxerr log.  */
/* Try pinging the same address as                                           */
/* ping xx.xx.xx.xx  where xx.xx.xx.xx is address printed in the log.        */
/* Always make sure all remote connections can be pinged before using HXECOM */
#define EX_GETH4        42
/*****************************************************************************/

/*****************************************************************************/
/* There is insufficient memory to run the exerciser.  If you get these errors*/
/* and you are running under HTX, check the memory exerciser.  It may be     */
/* taking too much memory and not leaving enough for hxecom.  Depending on   */
/* the order of execution of processes, the problem may appear to be random. */
#define EX_MALLOC1      31
```

```
#define EX_MALLOC2      32
#define EX_MALLOC3      33
#define EX_MALLOC4      34
#define EX_MALLOC5      35
#define EX_MALLOC6      36
#define EX_MALLOC7      37
#define EX_MALLOC8      38
#define EX_MALLOC9      39
#define EX_MALLOC10     40
#define EX_MALLOC11     41
/****************************************************************************/

/****************************************************************************/
/* You have exceeded a maximum array limit.  Study the error message in the   */
/* htxerr log and if you think you still need more space allocated.  Contact  */
/* the exerciser owner for a change.                                           */
#define  EX_EXERNO      43
#define  EX_COORD1      44
#define  EX_COORD16     45
#define  EX_COORD22     46
#define  EX_COMSEM7     47
/****************************************************************************/

/****************************************************************************/
/* Old shared memory existed when processes were started.  These errors       */
/* occurred while removing the old memory.  Additional information should be  */
/* in the htxerr log.                                                          */
#define EX_BACK1        50
#define EX_BACK2        51
#define EX_BACK3        52
#define EX_BACK4        53
#define EX_BACK5        54
#define EX_BACK6        55
#define EX_BACK7        56
/****************************************************************************/
```

## B Rules

The following has been extracted from the default rule file.

```
*RULE_ID is a free form name to be used as tester sees fit.
*Maximum length is 8 characters.  Default is 00000000.
*
*PATTERN_ID for pattern file.  Default is HEX255.
*
*COMNET_NAME for identification of network used for coordination.
*This must be a class C network configured for TCP/IP.  The name
*entered here must be in the /etc/hosts file and refer to a valid
*IP address.  The device represented by this address must not be
*in the mdt file.  Required field -- NO DEFAULT.
*
*TESTNET_NAME for identification of network under test.  This must
*be a name that exists in /etc/hosts and represent a valid IP
*address.   The device represented by this address must be in the
*mdt file.  Required field -- NO DEFAULT.
*
*COM_STREAM_PORT and COM_DGRAM_PORT will normally be the defaults but
*it is possible to partition the network using these parameters.
*Default is 5100 and 5101 respectively.
*
*REPLICATES specifies multiple r-w links if desired.  Normally there
*will be 1 full-duplex or 1 half-duplex test between devices.  If
*REPLICATES=n where n=1,2,3..., there will be n full-duplex or half-duplex
*tests between devices.  Default is 1.
*
*MASTER qualifies which remote exercisers can form connections with
*this exerciser.  At least 1 of the nodes forming a connection must
*be a master.  This is used to cause slaves to connect to a master
*node without having the slaves form connections between each other.
*             S-M-S    vs.   M-M
*               |            |X|
*               S            M-M
*Values are Y and N. Default is Y.
*
*OPER specifies if this exerciser process is allowed to read, write, or
*read-and-write. Valid string values are R, W, RW, or WR.  RW and WR
*are equivalent.  Default value is "W" if MASTER=N and "RW" if MASTER=Y.
*
*LAYER specifies if testing should be performed using TCP or UDP.
*When specifying UDP, WRITE_AHEAD value is critical.  See
*WRITE_AHEAD. Default is TCP.
*
```

```
*SHUTDOWN_FLAGS will be used to shutdown all network activity when
*a specific error(s) has been detected.  This will aid in locating
*packets on the network analyzer and isolating transmit from receive
*problems.  Default is 0x0000.  This number is always specified in
*hexadecimal format.  The 0x is optional.  The following are flags
*that can be specified individually or cumulatively:
*  - 0xffff  Halt on any error.
*  - 0x0001  Halt on miscompare.  err=00e3xxxx
*  - 0x0002  Halt when exceed IO_ALARM_TIME in a read. err=00daxxxx
*  - 0x0004  Halt on unknown interrupt. err=00dbxxxx
*  - 0x0008  Halt on timestamp error. err=00dexxxx
*  - 0x0010  Halt if received packet is wrong length. err=00dfxxxx
*  - 0x0020  Halt if packet dropped (Transmitted but not received). err=00e1xxxx
*  - 0x0040  Halt if receive a duplicate or out-of-sequence packet. err=00e0xxxx
*  - 0x0080  Halt if receive unidentified packet (not above errors).err=00e2xxxx
*  - 0x0100  Halt if data header fields violate construction rules. err=00ddxxxx
*  - 0x0200  Halt if write thread receives a SIGPIPE. err=00ebxxxx
* Default is 0x0000.
*
*The following rule attributes may be changed by using multiple
*stanzas.  The algorithm indicates their use.
*
*Write processes:
*for(stanza=0; stanza<NumberOfStanzas; stanza++) {
*    for(BufSize = BUFMIN[stanza]; BufSize<BUFMAX[stanza]; i
*                                          BufSize+=BUFINC[stanza]) {
*        for(i=0; i< NUM_OPER[stanza]; i++) {
*            if( !TCP ) {
*                while(Unacknowledge packets == WRITE_AHEAD[stanza])
*                    sleep
*            }
*            fill pattern with BufSize characters from pattern file
*            set alarm for IO_ALARM_TIME[stanza]
*            write(header+pattern+trailer)
*            clear alarm
*            if(error)
*                Print/Update.
*            sleep(WRITE_SLEEP[stanza] in milliseconds)
*        }
*        Print/Update stats
*    }
*}
*
```

```
*Read processes:
*for(stanza=0; stanza<NumberOfStanzas; stanza++) {
*    for(BufSize = BUFMIN[stanza]; BufSize<BUFMAX[stanza];
*                                              BufSize+=BUFINC[stanza]) {
*        for(i=0; i< NUM_OPER[stanza]; i++) {
*            set alarm for IO_ALARM_TIME[stanza]
*            read(header+BufSize characters+trailer)
*            clear alarm
*            check packet read to pattern generated from PATTERN_ID file
*            if( !TCP && (WRITE_AHEAD[stanza] - packets_since_ack
*                                                   <= ACK_TRIG[stanza]))
*                Acknowledge packet(s)
*            if(error)
*                Print/Update.
*        }
*        Print/Update stats
*    }
*}
*
*BUFMIN - Default is 100 characters.
*BUFMAX - Default is BUFMIN characters.
*BUFINC - Default is 1 character.
*
*NUM_OPER - Default is 1000.
*
*WRITE_SLEEP allows lowering the stress level by specifying sleep
*time in milliseconds.  It will also balance the work-load between
*fast and slow machines.  Default is 0.
*
*WRITE_AHEAD specifies how many packets can be written ahead of the
*read process without acknowledgements.  Default is 2 for UDP.  Size
*of the write ahead value for TCP is ignored.
*
*For UDP
*    The value of WRITE_AHEAD, buffer size(BUFMIN, BUFMAX, and
*    BUFINC), and network parameters interact to determine behavior.
*    See SC23-2365-01, "Performance Monitoring and Tuning Guide," for
*    information on how to setup.  Network option arpt_killc should be
*    set to a large value to prevent dropping packets every 20 minutes.
*
*ACK_TRIG specifies how many of the test packets to acknowledged.
*With a heavy load, a machine can become bound by the number of
*interrupts that it can service.  Using a low ACK_TRIG value will
*alleviate this problem except for very small packets.  The default
*is to ack all test packets.  ACK_TRIG=0 specifies to ack only the
*last packet in the WRITE_AHEAD sequence (0 left to write).  If you
*are using a fairly reliable comnet then this will work.  If it is
```

```
*not then there will be some timeouts related to dropping packets on
*the comnet.  These can be identified by looking at the error messages
*printed.
*---------------------------------------------------------------------------
*This shows that packet 239 had been written, but it had never been
*received -- A TEST PROBLEM.
*
*/dev/MPSF63S0     Dec 11 17:30:30 1993 err=00e10000 sev=4 hxecom
*R(100.1.1.21.1082) connected to W(100.1.1.25.1054)
*(Packet number), read (101),  expected (100).
*Packet(s) dropped.
*For stanza 0, UDP connection, bufmin=1000, bufmax=1000, bufinc=100
*Consecutive prior good reads=   156100,  good bytes read=        157192700
*
*/dev/MPSF63S0     Dec 11 19:02:35 1993 err=00da0004 sev=4 hxecom
*R(100.1.1.21.1082) connected to W(100.1.1.25.1054)
*Read failed - Interrupted system call.
*SIGALARM signal received.
*Resetting writer to i=239, bsize=1000, timestamp=1, pak=239.
*For stanza 0, UDP connection, bufmin=1000, bufmax=1000, bufinc=100
*Consecutive prior good reads=   228138,  good bytes read=        229734966
*
*/dev/MPSF63S0     Dec 11 18:57:57 1992 err=00f20000 sev=1 hxecom
*W(100.1.1.25.1054) connected to R(100.1.1.21.1082)
*Resetting Writer to -
*i= 239, timestamp=   1, bsize=  1000, stanza=  0, pak= 239.
*Parameters for next write prior to reset -
*i= 240, timestamp=   0, bsize=  1000, stanza=  0, pak= 240.
*For stanza 0, UDP connection,  bufmin=1000, bufmax=1000, bufinc=100
*Consecutive prior good writes=   384240,  good bytes written=        386929680
*
*----------------------------------------------------------------------------
*This shows that packet 332 hadn't been written causing the reader
*to time out -- A COMNET PROBLEM not a test problem.
*/dev/MPSF63S0     Dec 11 13:56:27 1993 err=00da0004 sev=4 hxecom
*R(100.1.1.23.1034) connected to W(100.1.1.25.1027)
*Read failed - Interrupted system call.
*SIGALARM signal received.
*Resetting writer to i=332, bsize=1000, timestamp=1, pak=332.
*For stanza 0, UDP connection, bufmin=1000, bufmax=1000, bufinc=100
*Consecutive prior good reads=     7332,  good bytes read=          7383324
*
*/dev/MPSF63S0     Dec 11 13:53:06 1992 err=00f20000 sev=1 hxecom
*W(100.1.1.25.1027) connected to R(100.1.1.23.1034)
*Resetting Writer to -
*i= 332, timestamp=   1, bsize=  1000, stanza=  0, pak= 332.
*Parameters for next write prior to reset -
```

```
*i= 332, timestamp=   0, bsize=  1000, stanza=  0, pak= 332.
*For stanza 0, UDP connection,  bufmin=1000, bufmax=1000, bufinc=100
*Consecutive prior good writes=     7332,  good bytes written=          7383324
*--------------------------------------------------------------------------------
*Recommendation: Use ACK_TRIG=1 or 0.  With WRITE_AHEAD=2 use ACK_TRIG=0.
*If a few timeouts occur, they can be sorted out by method above.
*
*
*IO_ALARM_TIME specifies how long to wait in a read or write function
*before printing an error.  Default is 300 seconds.
*
*IO_IDLE_TIME allows detection of sleeping write process.  This should only be
*used for debug.  Default is USHRT_MAX (65535 seconds).  If errors are printed,
*they should be used for information only.  They are not necessarily errors.
*
*TCP_NODELAY specifies if the TCP layer is allowed to delay writing.  Delayed
*writing may improve utilization of the network through reduced packets/larger
*packets on the network.  It may also reduce the stress on the hardware and
*device driver. TCP_NODELAY should be set to check boundary conditions in the
*device driver and adapter.  When set, user packets will usually be written
*without coalescing with other user packets.  When combined with combinations
*of BUFMIN,BUFMAX,BUFINC,NUM_OPER, and mtu size, boundary conditions can be
*checked quickly. Values are Y and N.  Default is Y.
*
*SO_LINGER specifies if data buffered in TCP socket should be flushed or
*delivered when socket is closed.  Values are Y and N.  Default is N.  A value
*of N will improve ability to restart exerciser for test conditions where
*the test-network is less than reliable.  A value of N will not allow testing
*the ability to force a close of socket with data.
*
RULE_ID=00000000
PATTERN_ID=HEX255
COMNET_NAME=comnetXX
TESTNET_NAME=testnetXX
BUFMIN=5000
BUFMAX=5000
BUFINC=1
MASTER=Y
OPER=RW
LAYER=TCP
IO_ALARM_TIME=300
SHUTDOWN_FLAGS=0x0000
WRITE_SLEEP=0
REPLICATES=1
NUM_OPER=1000

BUFMIN=100
```

```
BUFMAX=1000
BUFINC=100
NUM_OPER=100
```

## C Minimum Rules File and Mdt File

The following is the minimum rule file needed. HXECOM requires the name of the test network and the communications network as explained in Section 8.

```
COMNET_NAME=iovlab20
TESTNET_NAME=testnet20
```

The following is the minimum mdt file needed. The default stanza was cut from mdt.all. It may or may not represent what is needed.

```
default:
        HE_name = ""                      * Hardware Exerciser name, 14 char
        adapt_desc = ""                   * adapter description, 11 char max.
        device_desc = ""                  * device description, 15 char max.
        reg_rules = ""                    * reg rules
        emc_rules = ""                    * emc rules
        dma_chan = 0                      * DMA channel number
        idle_time = 0                     * idle time (secs)
        intrpt_lev = 0                    * interrupt level
        load_seq = 32768                  * load sequence (1 - 65535)
        max_run_tm = 180                  * max run time (secs)
        port = "0"                        * port number
        priority = 19                     * priority (1=highest to 19=lowest)
        slot = "0"                        * slot number
        adapt_addr = "x00000"             * adapter address
        hft = 0                           * hft number
        cont_on_err = "YES"               * continue on error (YES/NO)
        halt_level = "1"                  * level <= which HE halts
        start_halted = "n"                * exerciser halted at startup

tok0:
        HE_name = "hxecom_t"              * Hardware Exerciser name, 14 char
        adapt_desc = "token_ring"         * adapter description, 11 char max.
        device_desc = "commo"             * device description, 15 char max.
        reg_rules = "hxecom/NETmps"       * reg
        emc_rules = "hxecom/NETmps"       * emc
        max_run_tm = 300                  * max run time (secs)
        slot = "0003"                     * slot number
```

# D Script for Saving Data From Multiple Hosts

```
hxecom Save Configuration Script (multiple machines)
====================================================

Written by Gary M. Bass (x83599)
November 2, 1994


DESCRIPTION

    This program is a korn shell script, hxecom.save, that saves the
    test files created from hxecom, the LAN network exerciser.  It
    executes the script "hxecom.save.single" on all machines in the test
    including the local machine, then collects the files from each of
    the remote machines plus the local machine and places them under a
    common directory on the local (master) machine.  Please note that
    this use of the term "master" hxecom machine is not to be confused
    with the term "master" used in running hxecom.


REQUIREMENTS: PLEASE READ THIS SECTION!!!

    In order to run this program, each system in the test with files to be
    copied must have the /etc/hosts.equiv and /.rhosts files modified as
    listed below, where master_host_name is the hostname of the master
    machine invoking the hxecom.save script:

      1. /etc/hosts.equiv must contain the following line (starts on the
         first character of the line):

            +master_host_name

      2. /.rhosts must contain the following line (starts on the first
         character of the line):

            master_host_name root

    These file modifications give the (master) invoking machine
    permission to execute the necessary commands on each remote machine.


FILES SAVED

    When executed on each machine, this program copies the following files
    to the save directory and preserves the original copy in the original
```

directory:

```
    /usr/lpp/htx/mdt/mdt
    /tmp/htxerr
    /tmp/htxmsg
    /tmp/htxstats
    /tmp/htxstats2
    /tmp/htxcom.config
    mdt rules files in stanzas containing hxecom, hxecom2 and hxecom_t
```

This program moves the following files to the save directory without preserving the original copy in the original directory:

```
        /tmp/htx*.wbuf*  (htxcom write buffer miscompare files)
        /tmp/htx*.rbuf*  (htxcom read buffer miscompare files)
```

This program executes the following commands and saves the output to the indicated file in the save directory:

```
        /usr/sbin/lsdev -C                   > lsdev.out
        /usr/sbin/lscfg -v                   > lscfg.out
        /usr/lpp/htx/etc/scripts/htxversion > htxversion.out
        /usr/sbin/arp -a                     > arp.out
        what /usr/lpp/htx/bin/hxecom         > what.hxecom
        what /usr/lpp/htx/bin/hxecom_t       > what.hxecom_t
        what /usr/lpp/htx/bin/hxecom2        > what.hxecom2
        errpt -a                             > errpt.out
        netstat -s                           > netstat.out
        netstat -Z -s                        > /dev/null
        netstat -Z -m                        > /dev/null
        netstat -m                           >> netstat.out
        tokstat -r -d tr$TOK_NUM             >> tokstat.out
        errclear 0                           (no results saved)
```

Please see the USER WARNINGS section of this document for cautions on some of the above commands (moving files instead of copying them and clearing the error log and tokstat's).

## SAVE_DIRECTORY NAMING CONVENTION

All of the files collected with this script are saved in a directory under /tmp/htxcom. The directory name is composed of the date of the run followed by a sequence number indicating the run number for that day. The format is "DATE_OF_RUN.SEQ". Examples are as follows:

```
        /tmp/htxcom/070492.5     # The fifth run on 7/4/92
```

```
/tmp/htxcom/122592.19    # The 19th run on 12/25/92
/tmp/htxcom/040492.7     # The seventh run on 4/4/92
```

When these directories (ie, 070492.5, 122592.19, 040492.7 etc.) are then collected on the local machine, the naming convention is as follows:

```
/tmp/htxcom/[cur_date].[master_TCP2].[master_seq]/[remote_TCPIP]
```

where

```
[cur_date]     = current date when invoking the master script
[master_TCP2]  = last 2 digits of master's TCPIP address
[master_seq]   = sequence number of run for that day
[remote_TCPIP] = entire TCPIP address of the remote machine
```

Please note that the sequence number on the remote machine and the cumulative save directory name on the local machine do not have to match. For example, if "hxecom.save.single" is being executed for the eighth time on the remote machine, the directory name on the remote machine is is 090993.8. However, if the cumulative save script is being executed for only the fourth on the local machine (say testnet20), the cumulative save directory name will be 090993.20.4. This could be the case if the remote machine has been involved in other testing.

```
Example:
               remote directory
   machine       data saved in    testnet20 local directory to be moved to
--------------  ---------------   ----------------------------------------
193.100.178.20   /tmp/090993.2    /tmp/htxcom/090993.20.4/193.100.178.20
193.100.178.21   /tmp/090993.3    /tmp/htxcom/090993.20.4/193.100.178.21
193.100.178.22   /tmp/090993.1    /tmp/htxcom/090993.20.4/193.100.178.22
193.100.178.23   /tmp/090993.6    /tmp/htxcom/090993.20.4/193.100.178.23
                                                     ^^ ^
                           key:   [directory] [date]  | | [remote mach ]
                                                      | |
                                         machine number |
                                                        |
                                                 sequence
```

USER WARNINGS:

1. If hxecom is executed from the command line (ie, `$ hxecom /dev/tok0 reg hxecom/default.rules`) and not from

the supervisor (ie, $ runsup), an mdt file is not created. Since this save script looks to parse the mdt file and can't, no rules files are saved. This can be circumvented by creating an mdt file containing references to the appropriate exerciser (hxecom, hxecom_t or hxecom2) before the script is executed. Then this script will then save the rules files.

2. This script *moves* the read buffer and write buffer miscompare files to the save directory instead of copying them. Thus, the original copies are preserved but in the save directory. After execution of this script, the user will not find the miscompare files in the /tmp directory.

3. This script copies all miscompare files in the /tmp directory, not necessarily just the ones from the most recent run. The format of the files moved is as followed:

   ```
   $ mv /tmp/htx*.wbuf* $SAVE_DIRECTORY
   $ mv /tmp/htx*.rbuf* $SAVE_DIRECTORY
   ```

   Thus, if there are read buffer and write buffer files in /tmp that are left over from a previous run, this script will not discriminate against them and consequently move them to the save directory.

4. This script zeroes out all entries in tokstat via the command:

   ```
   tokstat -r -d tr$TOK_NUM >> $SAVE_DIR/tokstat.out
   ```

   Consequently, once this script is run, the tokstat results are irretrievably gone.

5. This script clears the error log using the command:

   ```
   errclear 0      # clear the error log
   ```

   Again, once this script is run, errors in the error log are irretrievably gone.

NODE LIST GENERATION (this is a user FYI section):

hxecom creates the file /tmp/htxcom.config on the local machine that lists the machines (by TCPIP address) with which it transferred data. As an example, say a machine with a TCPIP address of 100.178.100.24 transfers files between itself and machines with TCPIP addresses of 100.178.100.25 and 100.178.100.26, the format of /tmp/htxcom.config is as follows:

```
        100.178.100.24
        100.178.100.25
        100.178.100.26
```

Please note that the list also includes the address of the local machine. This local machine address is used by the algorithm to determine all nodes in the test.

The script creates a master node list starting with contents of /tmp/htxcom.config on the local machine. It then reads the /tmp/htxcom.config file of each of the machines listed, and adds any new nodes to the list. It then reads the /tmp/htxcom.config from the new nodes and once again, creates a new node list if any machine names found are not in the master node list. This algorithm continues until no new nodes are found.

Assumption:

When transferring files to the master machine, this script assumes that all nodes listed are accessible by the master machine (ie, they're all on the same network). In the example below, machine 1 is transferring files with machines 2, 3 and 4, while machine 4 is transferring files with machine 5.

```
Example:    file transfers: 1 <-> 2, 3, 4
                            4 <-> 5

            where 1 is the master machine
```

With this assumption, files from machine 5 need not be moved from 5 to 4 and then from 4 to 1. They can be moved directly from 5 to 1. Problems occur if there are two different networks, one connecting nodes 1 through 4 and the other connecting nodes 4 and 5 (ie, token ring for 1 thru 4 and ethernet for 4 and 5). In this example, machine one would not necessarily be able to connect to machine 5, and the files would not be saved.