54 TCP/SNA update

June 2004

In this issue

TCP/SNA Update

Published by

Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

Phone: 214-340-5690
Fax: 214-341-7081

Editor

Trevor Eddolls
E-mail: trevore@xephon.com

Publisher

Nicole Thomas
E-mail: nicole@xephon.com

Subscriptions and back-issues

A year's subscription to *TCP/SNA Update*, comprising four quarterly issues, costs $190.00 in the USA and Canada; £130.00 in the UK; £136.00 in Europe; £142.00 in Australasia and Japan; and £140.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the March 2000 issue, are available separately to subscribers for $49.50 (£33.00) each including postage.

Editorial panel

Articles published in TCP/SNA Update are reviewed by our panel of experts. Members include John Bradley (UK), Carlson Colomb (Canada), Anura Gurugé (USA), Jon Pearkins (Canada), and Tod Yampel (USA).

Contributions

When Xephon is given copyright, articles published in *TCP/SNA Update* are paid for at the rate of $160 (£100 outside North America) per 1000 words and $80 (£50) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of $32 (£20) per 100 lines. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

TCP/SNA Update on-line

Code from *TCP/SNA Update*, and complete issues in Acrobat PDF format, can be downloaded from http://www.xephon.com/tcpsna; you will need to supply a word from the printed issue..

Printed in England.

zSeries OSA-Express Fast Ethernet (non-QDIO) port sharing made simple

INTRODUCTION

OSA-Express is a hardware component that helps integrate mainframes into a LAN. OSA-Express Fast Ethernet (FENET) is one of the commonly-used type/features of OSA-Express. It operates both in QDIO and non-QDIO mode and it runs both SNA and TCP/IP. Sometimes it may be necessary to share FENET ports between LPARs, perhaps because the numbers of FENET ports and LPARs do not map one-to-one and TCP/IP is required in all the LPARs. OSA-Express can be shared in both QDIO and non-QDIO mode. Here, we consider the sharing of FENET ports between LPARs in non-QDIO modes, with TCP/IP in passthrough mode.

NON-QDIO MODE

To operate OSA-Express FENET CHIPID in non-QDIO mode, FENET CHIPID should be defined in the I/O definitions (IODF) as type OSE. The TCP/IP stack in non-QDIO mode requires two devices to function. To share FENET in non-QDIO mode requires an additional definition in the IODF and the use of the OSA/SF (Open System Adapter/Support Facility) tool. The additional IODF definition is to enable OSA/SF to interface with FENET CHIPID. An additional device with UA=FE and of type OSAD is defined for OSA/SF to gain control of the CHIPID.

IODF

The required IODF is:

```
CHPID PATH=(1Ø), SHARED, PARTITION= (( LPAR1, LPAR2,) , ( LPAR1, LPAR2 )
```

```
) , TYPE=OSE

CNTLUNIT  CUNUMBR=0009,PATH=(10),UNIT=OSA

IODEVICE  ADDRESS=(100,032),CUNUMBR=(0009),UNIT=OSA
IODEVICE  ADDRESS=18F,UNITADD=FE,CUNUMBR=(0009),UNIT=OSAD
```

OSA/SF AND OAT

Without OSA/SF, it would not be possible to share ports in non-QDIO mode. Installing and operating OSA/SF is quite simple. OSA uses a table called the OSA Access Table (OAT) for registering details of the device/ports used by different LPARs and configuration files. It has details like port MAC address. When there is no OSA port sharing, OSA does not use the OAT, it has just a default dummy OAT. But for port sharing OSA does use the OAT, which can easily be customized for port sharing. OSA/SF has a couple of interfaces, the most commonly used being the IOACMD REXX interface. The steps to set up port sharing are:

1 Customize the OAT and the configuration table.

2 Install the OAT and the configuration table on the OSA FENET CHIPID.

An OAT template from samples can be used for customization. This is how an OAT template looks:

```
*******************************************************
 *UA  Mode     Port   Default      IP Address
  *****************************************************

*******************************************************
```

If it is planned to share CHIPID 10 between LPAR1 and LPAR2, TCP/IP would use OSA device 100-101 on LPAR1 with IP address xxx.xx.xx.101, and device 102-103 of LPAR2 with IP address xxx.xx.xx.102. The customized OAT for this would look like:

```
*******************************************************
 *UA  Mode     Port   Default      IP Address
```

```
*****************************************************
                        LP 1 (LPAR1)
00  passthru   00        pri      xxx.xx.xx.101
                        LP 2 (LPAR2)
02  passthru   00        sec      xxx.xx.xx.102
```

A configuration file template can be obtained from the IOACMD GET configuration file. The default configuration file itself will do if there is no need to change the MAC address and for TCP/IP passthrough mode. Using the IOACMD install command, the OAT and the configuration file can be loaded to OSA-Express from any of the LPARs, but the devices need to be kept off-line in all LPARs. After installation of the above OAT, TCP/IP for LPAR1 can be brought up with device 100 and IP address xxx.xx.xx.101 with the following TCP/IP profile definition:

```
DEVICE     OSA2DEVD00         LCS           100
LINK       OSA2ENLINK1        ETHEROR802.3  0     OSA2DEVD00
xxx.xx.xx.101   OSA2ENLINK1
```

TCP/IP for LPAR2 with device 102 and IP address xxx.xx.xx.102 can be brought up with the following TCP/IP profile definition:

```
DEVICE     OSA2DEVD00         LCS           102
LINK       OSA2ENLINK1        ETHEROR802.3  0     OSA2DEVD00
xxx.xx.xx.102   OSA2ENLINK1
```

Arun Kumaar R
MVS Systems Programmer (India)

TCP/IP programming with CICS PL/I server and VB6 client

TCP/IP BASICS

With CICS TCP/IP, remote client systems can invoke CICS transactions. This is the usual mode of operation. The opposite way is also possible, where a CICS transaction is the client and a remote system is the server.

TCP/IP provides a reliable connection between different applications, and a connection is made before sending and receiving any data. Data is sent without errors and is received in the same order that it was sent. For TCP the data is a stream of bytes. A TCP/IP host can communicate with any remote CICS or non-CICS system that has TCP/IP installed.

CICS TCP/IP SERVER PROCESS CONCEPTS

When you have CICS on the server side you can choose either a concurrent or an iterative server. In this article we will mostly discuss using a concurrent server.

Iterative server

With an iterative server we can process only one socket at a time. The server handles the request for connection and the transaction, which should be executed. Iterative servers are simpler and are appropriate for transactions that don't take long to complete.

If the transaction takes more time, a concurrent server would be a better solution because, with the iterative server, once one client has started a transaction no other client can make a call until the first client has finished.

Listener

The Listener transaction, CSKL, is provided as part of CICS TCP/IP. Listener performs several operations.

It will 'listen' on the port specified in the configuration file and wait for incoming connection requests from clients. When a connection request arrives, the Listener accepts it and obtains a new socket to pass to the CICS child server application program. It starts the CICS child server transaction and waits for the child server to take the socket and then issues the close call. When this is done, the child server program owns the socket and the Listener has nothing to do with that socket any more. The Listener can process 49 child servers simultaneously.

Security link module for Listener

Listener provides the way for security checking to be performed before a CICS transaction is invoked. If a security module is not provided, all transactions can be executed.

If you want to write your own security module, you can call it anything you like, but you have to define it in the configuration dataset. You can write the program in PL/I, COBOL, or Assembler language, but you must define that program in the CICS Program Processing Table (PPT).

Just before the task creation process, the Listener invokes the security module by a CICS LINK, passing a COMMAREA. The Listener passes a data area to the security module that contains information for the module to use for security checking and a 1-byte switch. Your security module should perform a security check and set the switch properly.

When the security module returns, the Listener checks the value of the switch and initiates the transaction if the switch has a value of 1. In the module you can use any CICS statement and function because this is a real CICS LINK module. Remember, excessive programming could cause performance degradation.

An example of a security module:

```
TCPSEC:PROC(POINT) OPTIONS(MAIN);
  /*================================================================*/
 /* FUNCTION    : SECURITY MODULE                                  */
  /*================================================================*/

 %INCLUDE DFHAID;
 %INCLUDE DFHBMSCA;

 DCL (VERIFY,TIME,DATE,ADDR,CSTG,STG,SUBSTR) BUILTIN;

 DCL POINT       PTR;
 DCL 1 COMAREA                BASED(POINT),
     2 TRAN_ID   CHAR(4),  /*CICS transaction requested by the client*/
     2 USERDATA  CHAR(40), /* Data received from the client          */
     2 ACTION    CHAR(2),  /* Method of starting the task:           */
                           /* IC Interval control                    */
                           /* KC Task control                        */
```

```
                          /* TD Transient data                      */
     2 INTERVAL  CHAR(6),
      /* Interval requested for IC start control time format hhmmss  */
     2 ADR_FAMILY BIN FIXED(15),
                /* Network address family. A value of 2 must be set.*/
     2 PORT      BIN FIXED(15),
                          /* The port number of the requester's port*/
     2 ADDRESS   BIN FIXED(31),
                          /* The IP address of the requester's host */
     2 SWITCH1   CHAR(1),  /*1 Pass the socket Not 1 Close connection*/
     2 SWITCH2   CHAR(1),  /* 1 Listener sends message to client     */
          /* Not 1 Security Exit program sends message to client     */
     2 TERMINAL  CHAR(4),  /* Terminal ID                            */
     2 SOCK_ID   BIN FIXED(15),    /* Current socket descriptor      */
     2 USERID    CHAR(8);   /* User ID                               */

 DCL 1 DATA                   BASED(ADDR(USERDATA)),
     2 USER_NAME  CHAR(8),
     2 ACCOUNT    PIC'(11)9',
     2 PASSWORD   PIC'(5)9',
     2 FILLER     CHAR(16);

 SWITCH1='Ø';
 SWITCH2='1';
 SELECT (TRAN_ID);
   WHEN('XXØ1')
   DO;
     IF USER_NAME='XXXXXXØ1 ' & ADDRESS = REQUIRED_ADDRESS1  THEN
SWITCH1='1';
   END;
   WHEN('XXØ2')
   DO;
     IF USER_NAME='XXXXXXØ2 ' & ADDRESS = REQUIRED_ADDRESS2  THEN
SWITCH1='1';
   END;
   OTHERWISE;
 END;
 EXEC CICS RETURN;

END TCPSEC;
```

Conversion routines

CICS uses the EBCDIC data format, TCP/IP networks use ASCII. When exchanging data between CICS and the TCP/IP network, your application programs must use the necessary data conversion modules. CICS TCP/IP provides several conversion routines:

1 An EBCDIC-to-ASCII conversion routine used to convert EBCDIC data within CICS to the ASCII format used in TCP/IP networks and workstations. This is module EZACIC04:

```
CALL EZACICØ4('TCPIPTOASCIIXLAT',TCP_BUF,RETCODE);
```

2 A corresponding ASCII-to-EBCDIC conversion routine, EZACIC05:

```
CALL EZACICØ5('TCPIPTOEBCDICXLT',TCP_BUF,RETCODE);
```

Child server

```
CSØ1: PROC OPTIONS(MAIN);
  /*********************************************************************/
 /* FUNCTION : EXAMPLE OF CHILD CICS/PL1 CHILD SERVER                 */
/*********************************************************************/

 %INCLUDE DFHAID;
 %INCLUDE DFHBMSCA;
DCL (TIME,DATE,ANY,ADDR,CSTG,VERIFY,STG,SUBSTR,LENGTH) BUILTIN;

 DCL EZASOKET ENTRY   OPTIONS(ASSEMBLER,RETCODE) EXTERNAL;
 DCL EZACICØ4 ENTRY   OPTIONS(ASSEMBLER,RETCODE) EXTERNAL;
 DCL EZACICØ5 ENTRY   OPTIONS(ASSEMBLER,RETCODE) EXTERNAL;

 DCL RES     BIN FIXED(15);

 DCL 1  SOKET_FUNCTIONS,
     2 SOKET_ACCEPT       CHAR(16) INIT('ACCEPT          '),
     2 SOKET_BIND         CHAR(16) INIT('BIND            '),
     2 SOKET_CLOSE        CHAR(16) INIT('CLOSE           '),
     2 SOKET_CONNECT      CHAR(16) INIT('CONNECT         '),
     2 SOKET_FCNTL        CHAR(16) INIT('FCNTL           '),
     2 SOKET_GETCLIENTID  CHAR(16) INIT('GETCLIENTID     '),
     2 SOKET_GETHOSTBYADDR   CHAR(16) INIT('GETHOSTBYADDR   '),
     2 SOKET_GETHOSTBYNAME   CHAR(16) INIT('GETHOSTBYNAME   '),
     2 SOKET_GETHOSTID       CHAR(16) INIT('GETHOSTID       '),
     2 SOKET_GETHOSTNAME     CHAR(16) INIT('GETHOSTNAME     '),
     2 SOKET_GETPEERNAME     CHAR(16) INIT('GETPEERNAME     '),
     2 SOKET_GETSOCKNAME     CHAR(16) INIT('GETSOCKNAME     '),
     2 SOKET_GETSOCKOPT      CHAR(16) INIT('GETSOCKOPT      '),
     2 SOKET_GIVESOCKET      CHAR(16) INIT('GIVESOCKET      '),
     2 SOKET_INITAPI         CHAR(16) INIT('INITAPI         '),
     2 SOKET_IOCTL           CHAR(16) INIT('IOCTL           '),
     2 SOKET_LISTEN          CHAR(16) INIT('LISTEN          '),
     2 SOKET_READ            CHAR(16) INIT('READ            '),
```

```
     2 SOKET_RECV            CHAR(16) INIT('RECV            '),
     2 SOKET_RECVFROM        CHAR(16) INIT('RECVFROM        '),
     2 SOKET_SELECT          CHAR(16) INIT('SELECT          '),
     2 SOKET_SEND            CHAR(16) INIT('SEND            '),
     2 SOKET_SENDTO          CHAR(16) INIT('SENDTO          '),
     2 SOKET_SETSOCKOPT      CHAR(16) INIT('SETSOCKOPT      '),
     2 SOKET_SHUTDOWN        CHAR(16) INIT('SHUTDOWN        '),
     2 SOKET_SOCKET          CHAR(16) INIT('SOCKET          '),
     2 SOKET_TAKESOCKET      CHAR(16) INIT('TAKESOCKET      '),
     2 SOKET_TERMAPI         CHAR(16) INIT('TERMAPI         '),
     2 SOKET_WRITE           CHAR(16) INIT('WRITE           ');

 DCL RETR_LENG    BIN FIXED(15)       INIT(72);
 DCL SOCK_ID      BIN FIXED(15);
 DCL SOCK_ERR     BIN FIXED(31);
 DCL SOCK_RET     BIN FIXED(31);
 DCL SOCK_RECV    BIN FIXED(31);
 DCL SOCK_LEN     BIN FIXED(31)       INIT(10000);
 DCL SOCK_BUF     CHAR(10200);
 DCL SOCK_ERRPIC PIC'(8)Z9';

 DCL 1 SOCK_CLIENT,
     2 CLI_DOMAIN   BIN FIXED(31)   INIT(2),
     2 CLI_NAME     CHAR(8),
     2 CLI_TASK     CHAR(8),
     2 FILLER       CHAR(20);

 DCL 1 SOCK_SERVER,
     2 SOCKET_ID           BIN FIXED(31),
     2 LSTN_NAME           CHAR(8),
     2 LSTN_SUBTASKNAME    CHAR(8),
     2 CLIENT_DATA         CHAR(35),
     2 FILLER              CHAR(1),
     2 SOCKADDR_IN,
       3 FAMILY            BIN FIXED(15),
       3 PORT              BIN FIXED(15),
       3 IP_ADDR           BIN FIXED(31),
       3 RESERVE           CHAR(8);

EXEC CICS IGNORE CONDITION LENGERR;
/* This retrieves the data passed by the START command    */
/* in the concurrent server (Listener) program. This      */
/* data includes the socket descriptor and the concurrent */
/* server client ID as well as optional additional data   */
/* from the client and we are now using that technology   */
/* because in most cases this is enough                   */

 EXEC CICS RETRIEVE INTO(SOCK_SERVER) LENGTH(RETR_LENG) RESP(RES);
 IF RES¬=DFHRESP(NORMAL)
```

```
    THEN CALL SEND_DATA('ERROR ON RETRIEVE STATEMENT_END',Ø);

/* This acquires the newly-created socket from the             */
/* concurrent server. The TAKESOCKET parameters must           */
/* specify the socket descriptor to be acquired and            */
/* the client id of the concurrent server. This information    */
/* was obtained by the EXEC CICS RETRIEVE command.             */

 SOCK_ID=SOCKET_ID;
 CLI_NAME=LSTN_NAME;
 CLI_TASK=LSTN_SUBTASKNAME;
 CALL  EZASOKET(SOKET_TAKESOCKET,SOCK_ID,SOCK_CLIENT,SOCK_ERR,SOCK_RET);
 IF SOCK_RET<Ø THEN CALL SEND_DATA(' ERROR ON TAKESOKET
STATEMENT_END',Ø);
 SOCK_ID=SOCK_RET;


 IF  VERIFY(SUBSTR(CLIENT_DATA,5,5),'1234567890')¬=Ø
    THEN CALL SEND_DATA('ERROR ON CLIENT DATA_END',Ø);
 TRKEY = SUBSTR(CLIENT_DATA,5,5);

 EXEC CICS READ DATASET('FILE_NAME') RIDFLD(TRKEY) INTO(RECORD_VAR)
RESP(RES);
 IF RES=DFHRESP(NOTFND) THEN CALL SEND_DATA('ERROR READING FILE_END',1);
 IF RES¬=DFHRESP(NOTFND) & RES¬=DFHRESP(NORMAL)
    THEN CALL SEND_DATA('FILE CLOSED_END',1);

CALL  SEND_DATA(SUBSTR(RECORD_VAR,56,15));

 SEND_DATA:PROC(MESSAGE,LOG);
   DCL MESSAGE CHAR(1ØØØØ);
   DCL LOG    DEC FIXED(1);
   SOCK_LEN = 15;
   CALL EZACIC04(MESSAGE,SOCK_LEN);
   /* Conversation with the client                                */
   SOCK_BUF=MESSAGE;
   CALL
EZASOKET(SOKET_WRITE,SOCK_ID,SOCK_LEN,SOCK_BUF,SOCK_ERR,SOCK_RET);
   IF SOCK_RET<Ø THEN EXEC CICS ABEND ABCODE('CSØ1');

   /* Terminates the connection and releases  */
   /* the socket resources when finished.     */
   DELAY(1ØØ);
    CALL EZASOKET(SOKET_CLOSE,SOCK_ID,SOCK_ERR,SOCK_RET);
   IF SOCK_RET<Ø THEN EXEC CICS ABEND ABCODE('CSØ1');

   EXEC CICS RETURN;

 END SEND_DATA;
```

```
END CSØ1;
```

Sockets

The socket API is a collection of socket calls that enable you to:

- Perform the communication functions between application programs.
- Set up and establish connections to other users on the network.
- Send and receive data to and from other users.
- Close down connections.

A socket is an end point for communication that can be named and addressed in a network. From an application program perspective, a socket is a resource that is allocated by the TCP/IP address space. A socket is represented to the program by an integer called a socket descriptor.

MVS supports three socket types – stream, datagram, and raw. While CICS supports stream and datagram sockets, stream sockets provide the most reliable form of data transfer offered by TCP/IP. Stream sockets transmit data between TCP/IP hosts that are already connected to one another. Data is transmitted in a continuous stream. There is no record length or newline character between data. Communicating processes must agree on a scheme to ensure that both client and server know that all data has been received. One way of doing this is for the sending process to send the length of the data followed by the data itself or we can send something like '_END' to notify the client or server that this is the end of the data.

An address family defines a specific addressing format. Applications that use the same addressing family have a

common scheme for addressing sockets. TCP/IP for CICS supports the AF_INET address family:

```
DCL 1 SOCKADDR_IN,
    2 FAMILY       BIN FIXED(15),  /* Always 2 AF_INET family */
    2 PORT         BIN FIXED(15),  /* Application port number */
    2 IP_ADDR      BIN FIXED(31),
 /* Internet address of the network interface used by the application*/
    2 RESERVE      CHAR(8);        /* All zeros               */
```

A port is a 16-bit integer that defines a specific application within an IP address in which several applications use the same network interface. The port number is a qualifier that TCP/IP uses to route incoming data to a specific application within an IP address.

Tasks use the GIVESOCKET and TAKESOCKET functions to pass sockets from parent to child. The task passing the socket uses GIVESOCKET, and the task receiving the socket uses TAKESOCKET.

Once a client has been connected to the server, and the socket has been transferred from the main task (parent listener) to the subtask (child server), the client and server exchange application data using READ/WRITE calls.

VB client

```
Private Sub cmdsend_Click()

  DIM As WinsockKlijent AS mswinsock

  ALL_RECEIVED_DATA =  ""
  WinsockKlijent.Protocol = sckTCPProtocol
  WinsockKlijent.RemoteHost = "xxx.xx.x.x"
  WINSOCKKLIJENT.REMOTEPORT = YY  ' Integer

  WinsockKlijent.Connect

  WINSOCKKLIJENT.SENDDATA "CSØ1,XXXXXXØ1" & ACCOUNT ' String

  WinsockKlijent.Close

End Sub
```

```
Private Sub WinsockKlijent_DataArrival(ByVal bytesTotal As Long)
     WinsockKlijent.GetData RECEIVED_DATA, vbString
     ALL_RECEIVED_DATA = ALL_RECEIVED_DATA & RECEIVED_DATA
End Sub
```

Nebojsa Cosic
System Analyst
Ivan Bugarinovic
System Analyst
Pinkerton Computer Consultants Inc (USA)

FTP efficacy checking in batch processing

INTRODUCTION

File transfer protocol is a commonly-used technique in batch processing. It facilitates transfer of data between a local computer and a remote one. It is very important for batch processing using FTP to have accurate information on whether the file transfer ended successfully or not.

PROBLEM DESCRIPTION

The return code from a job step using FTP is always 0, even if FTP has not successfully ended. It would be wrong then to rely solely on the return code to indicate problems. Further analysis is required.

However, it is inefficient to check and analyse FTP report(s) all the time. A better way is to create a procedure that will do it automatically.

The procedure should not only compare the FTP report(s) against the amount of data that is expected to be transferred, but should also produce an adequate return code and a message.

SOLUTION

In batch processing, FTP is invoked by using the JCL execute statement.

Here is an example of a job step that invokes FTP:

```
//FTPSTEP   EXEC PGM=FTP,
//            PARM='remote_computer_ip_address',other_optional_parameters
//OUTPUT    DD DSN=FTPOUT,DISP=(NEW,CATLG,DELETE),
//             DCB=(RECFM=FB,LRECL=160),
//             SPACE=(160,(5,1),RLSE),AVGREC=K
//SYSPRINT  DD SYSOUT=*
//INPUT     DD *
   ftp_subcommands
/*
```

The FTP OUTPUT file could have been defined in a several different ways.

In this example the OUTPUT file, named FTPOUT, is a sequential file with a qualified name. Its DCB parameters are recommended values. To ensure that FTPOUT does not exist, a delete step should precede FTPSTEP:

```
//STEPDEL  EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
      DELETE FTPOUT
      SET MAXCC=0
/*
```

The DD statement of the INPUT file in most cases might consist of many FTP commands and subcommands. In this example we will consider a case of logging and using any number of PUT, MPUT, GET, and MGET subcommands – the ones that handle the data transfer:

```
ftp_subcommands =

USER_NAME
PASSWORD
PUT   local_file             remote_file
MPUT  local_files            remote_file
GET   remote_file            local_file (REPLACE
MGET  remote_files           local_file (REPLACE
QUIT
```

Remote_file in the PUT and MPUT, and *local_file* in the GET and MGET subcommands are optional, and, if omitted, they get the default names.

The *local_file* parameter in an OS/390 (MVS) environment should be a qualified dataset name on the local host.

If a job has more FTP steps, the restriction is that, in each step following the first one, OUTPUT line must be:

```
//OUTPUT DD DSN=FTPOUT,DISP=(MOD)
```

Then the whole job's FTP report will be in FTPOUT.

FTP is a program that executes on the TCP/IP network protocol, and the FTPOUT file is in fact a sequence of FTP client messages created as a result of the execution of FTP commands and subcommands.

The idea is to read FTPOUT sequentially and do the following:

- Check whether a connection has been established; if not, raise an appropriate error message and return code.
- Find lines with PUT, MPUT, GET, or MGET subcommands, and find the names of the files on the local host that have been copied to (or from) the remote computer.
- Find the transfer type (binary, ASCII, EBCDIC).
- Determine the record type – variable or fixed length – of a copied file.
- Find the number of bytes that have been transferred.
- If any transfer took place, invoke the sort procedure on the copied file in order to get its number of records and calculate its number of bytes, by sequentially reading the sort sysout file.
- Compare (by taking into consideration the transfer type) the number of bytes calculated and the number of bytes that have been transferred.

- Show the resulting line for each subcommand.
- Signal any inconsistency by raising an appropriate error message and return code.

This is all done by the REXX FTPCHECK procedure, which should be executed as the last step of the job.

Because of a problem with sort within REXX, when FTP works with files with fixed and variable length records, FTPCHECK is executed twice – once for files with fixed record lengths, and then for files with variable record lengths. It depends on the record type argument value passed to FTPCHECK. V is used for variable record lengths; anything else is used for fixed record lengths.

```
//FTPCHKF  EXEC PGM=IKJEFT01,step_parameters
//SYSEXEC  DD DSN=library_name,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//FTP      DD DSN=FTPOUT,DISP=SHR
//SYSIN    DD DSN=FTPSYSIN,DISP=SHR
//SYSOUT   DD DSN=sysout_file,DISP=SHR
//SORTOUT  DD DUMMY
//SYSTSIN DD   *
 %FTPCHECK
/*
//FTPCHKV  EXEC PGM=IKJEFT01,step_parameters
//SYSEXEC  DD DSN=library_name,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//FTP      DD DSN=FTPOUT,DISP=SHR
//SYSIN    DD DSN=FTPSYSIN,DISP=SHR
//SYSOUT   DD DSN=sysout_file,DISP=SHR
//SORTOUT  DD DUMMY
//SYSTSIN DD   *
 %FTPCHECK V
/*
```

The FTP file is passed from previous step(s).

The SYSIN file, *FTPSYSIN*, is a PDS file.

```
FTPSYSIN = "SORT FIELDS=(1,1,CH,A),WORK=3".
```

Since the *SORTOUT* file is DUMMY, the *SORTIN* file is not sorted at all.

Sysout_file could be any sequential file.

CODE

```
/* Rexx */
ARG record_type
IF record_type¬="V"  THEN record_type = "F"

connection_OK = Ø
bytes_transfered = -2
unsuccessfull = Ø

do while rc¬=2
  "execio 1 diskr ftp"
  if rc=Ø then
  do
    parse pull rec1
    if connection_OK=Ø then
    DO  /* searches 23Ø FTP reply code meaning - User logged on */
      if (word(rec1,1)="23Ø") then connection_OK = 1
    end
    else do
      if (word(rec1,1)="EZA1736I") & ((word(rec1,2)="PUT") | ,
         (word(rec1,2)="GET") | (word(rec1,2)="MGET") | ,
         (word(rec1,2)="MPUT")) then
      do    /* new subcommand */
        if bytes_transfered = -1 then
        do  /* previous subcommand not executed */
          say "ERROR "ftp_subcommand file_name "has not been executed"
          unsuccessfull = unsuccessfull + 1
        end
        ftp_subcommand = word(rec1,2)
        bytes_transfered = -1
        select
          when ftp_subcommand = "PUT" then file_name = word(rec1,3)
          when ftp_subcommand = "GET" then
          do
            if word(rec1,4)=null | word(rec1,4)="(REPLACE"
            then file_name = word(rec1,3)
                        /* local_file not entered, gets default name */
     else file_name = word(rec1,4)
                                                /* local_file entered */
     end
      /* for "MGET" file_name is in following EZA17Ø1I >>> RETR line */
      /* for "MPUT" file_name is in following EZA17Ø1I >>> STOR line */
          otherwise file_name = " "
        end
      end    /* new subcommand */
    if word(rec1,1)¬="EZA1736I" & (bytes_transfered=-1 | file_name= " ")
then
      do     /* other information */
```

```
        if file_name = " " & word(rec1,1)="EZA17Ø1I" ,
             & (word(rec1,3)="STOR" |  word(rec1,3)="RETR") then
        do   /* file_name for MPUT or MGET */
          file_name = word(rec1,4)
          bytes_transfered=-1
        end
        if word(rec1,1)="15Ø" then transfer_type = word(rec1,3)
                                          /* BINARY, ASCII, EBCDIC */
        if word(rec1,1)="EZA1617I" then
        do   /* transfer information */
          bytes_transfered = word(rec1,2)
          call determine_file_type  /* F/V */
          if file_type=record_type then
          do
            call calculate_nbf
            if number_of_bytes ¬= bytes_transfered then
            do   /* raise ERROR message */
              unsuccessfull = unsuccessfull + 1
              say "ERROR" file_name "has" number_of_bytes "bytes"
            end
            else say file_name "has" number_of_bytes "bytes"
                                                  /* ok, no ERROR */
            say substr(rec1,9)  /* subcommand resulting line */
          end
          if (ftp_subcommand = "MPUT" | ftp_subcommand = "MGET")
          then file_name = " "  /* for next file in MPUT, MGET */
        end   /* transfer information */
      end  /* other information */
    end   /* else do */
  end    /* if rc=Ø */
end
"execio Ø diskr ftp (finis"

if connection_OK=Ø then
do      /* raises appropriate message and return code */
  say "Connection not established"
  RC=1Ø
  EXIT RC
end

if bytes_transfered = -1 then
do      /* if last subcommand failed */
  say "ERROR "ftp_subcommand file_name "has not been executed"
  unsuccessfull = unsuccessfull + 1
end

if unsuccessfull>Ø then
do   /* message and return code */
  RC=4
```

```
  EXIT RC
end

EXIT

/************************ subroutines ***************************/
determine_file_type:
                          /* determine file_type by invoking listds tso
                            command which returns
                            VB, VBA, FB or FBA as a first word in one
                            of its lines */
  t=OUTTRAP('level.')
  "listds" file_name
  t=OUTTRAP('OFF')
  do i=1 to level.Ø
     if substr(Word(level.i,1),1,1) = 'V' | ,
        substr(Word(level.i,1),1,1) = 'F'
     then file_type = substr(Word(level.i,1),1,1)
  end
  RETURN

calculate_nbf: /* calculate number of bytes in file_name */
  number_of_bytes = Ø
  number_of_records = Ø
  number_of_bytes_sorted = Ø

  "alloc fi(sortin) da("file_name") shr reu"
  if rc>Ø then say "ERROR alloc fi(sortin) "file_name

  "call 'sys1.sortlpa(sort)'"
  if rc>Ø then say "ERROR call sort"file_name rc

  "free fi(sortin)"
  if rc>Ø then say "ERROR free fi(sortin) "file_name

  do while rc¬=2
    "execio 1 diskr sysout"
    if rc=Ø then
    do
      parse pull rec2
      if (word(rec2,1)="ICEØ54I") ,
      then number_of_records=word(rec2,8)
      if (word(rec2,1)="ICE134I") then
      do
        number_of_bytes_sorted=word(rec2,7)
        /* variable file sort adds 4 bytes to each record */
        if file_type="V"
        then number_of_bytes = ,
            (number_of_bytes_sorted - (number_of_records * 4))
```

```
        else number_of_bytes = number_of_bytes_sorted
      end
    end
  end
"execio Ø diskr sysout (finis"

/* ASCII and EBCDIC transfer types have carriage return and line feed
   characters at the end of each record, while BINARY do not. */
if transfer_type ¬= "BINARY" ,
then number_of_bytes = number_of_bytes + (2 * number_of_records)
RETURN
```

EXAMPLE

Here is an example of FTP from an IBM host to my personal computer.

- The first subcommand copies two files from the host to my PC.
- The second copies a file with variable record length to my PC.
- The third copies a file with the BINARY option to my PC.
- The fourth invokes a file that does not exist.
- The fifth subcommand copies back files from my first subcommand, from my PC to the host.
- The sixth one copies one of them into a file with a new name on the host.

JCL

```
//MNUPT231 JOB CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//FTPMYPC  EXEC PGM=FTP,PARM=my_pc_ip_address,REGION=4Ø96K
//OUTPUT   DD DSN=MNUPT.FTPOUT.PROL,
//            DISP=(NEW,CATLG,DELETE),DCB=(RECFM=FB,LRECL=16Ø),
//            SPACE=(16Ø,(5,1),RLSE),AVGREC=K
//SYSPRINT DD  SYSOUT=*
//INPUT    DD  *
POSTSTED\VAntonijevic
password
MPUT 'MNUP*.KURSLIST.*L'
PUT  'MNUPT.VARRECLE.STAL'
BINARY
```

```
PUT  'MNUPT.ISPLATE.PROL'
PUT  'MNUPT.NOTEXIST.PROL'
ASCII
MGET 'MNUP*.KURSLIST.*L' (REPLACE
GET  'MNUPT.KURSLIST.PROL' 'MNUPT.TEST.PROL' (REPLACE
QUIT
/*
//FTPCHKF  EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
//SYSEXEC  DD DSN=POSTT.MNUP.CNTL,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//FTP      DD DSN=MNUPT.FTPOUT.PROL,DISP=SHR
//SYSIN    DD DSN=POSTT.MNUP.CNTL(FTPSYSIN),DISP=SHR
//SYSOUT   DD DSN=MNUPT.SOUT.PROL,DISP=SHR
//SORTOUT  DD DUMMY
//SYSTSIN DD   *
 %FTPCHECK
/*
//FTPCHKV  EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
//SYSEXEC  DD DSN=POSTT.MNUP.CNTL,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//FTP      DD DSN=MNUPT.FTPOUT.PROL,DISP=SHR
//SYSIN    DD DSN=POSTT.MNUP.CNTL(FTPSYSIN),DISP=SHR
//SYSOUT   DD DSN=MNUPT.SOUT.PROL,DISP=SHR
//SORTOUT  DD DUMMY
//SYSTSIN DD   *
 %FTPCHECK V
/*
```

FTPOUT = MNUPT.FTPOUT.PROL looks like this:

```
EZA1736I FTP my_pc_ip_address
EZA1450I IBM FTP CS V1R2
EZA1554I Connecting to:  my_pc_ip_address port: 21.
220 PC235-5 Microsoft FTP Service (Version 5.0).
EZA1459I NAME (my_pc_ip_address:PIPPG04):
EZA1701I >>> USER POSTSTED\VAntonijevic
331 Password required for POSTSTED\VAntonijevic.
EZA1789I PASSWORD:
EZA1701I >>> PASS
230 User POSTSTED\VAntonijevic logged in.
EZA1460I Command:
EZA1736I MPUT 'MNUP*.KURSLIST.*L'
EZA1701I >>> SITE FIXrecfm 31 LRECL=31 RECFM=FB BLKSIZE=27993
500 'SITE FIXrecfm 31 LRECL=31 RECFM=FB BLKSIZE=27993': command not
understood
EZA1701I >>> PORT port_address
200 PORT command successful.
EZA1701I >>> STOR 'MNUPP.KURSLIST.STAL'
150 Opening ASCII mode data connection for 'MNUPP.KURSLIST.STAL'.
226 Transfer complete.
```

```
EZA1617I 990 bytes transferred in 0.005 seconds.  Transfer rate 198.00
Kbytes/sec.
EZA1701I >>> SITE FIXrecfm 31 LRECL=31 RECFM=FB BLKSIZE=27993
500 'SITE FIXrecfm 31 LRECL=31 RECFM=FB BLKSIZE=27993': command not
understood
EZA1701I >>> PORT port_address
200 PORT command successful.
EZA1701I >>> STOR 'MNUPT.KURSLIST.PROL'
150 Opening ASCII mode data connection for 'MNUPT.KURSLIST.PROL'.
226 Transfer complete.
EZA1617I 1980 bytes transferred in 0.005 seconds.  Transfer rate 396.00
Kbytes/sec.
EZA1460I Command:
EZA1736I PUT  'MNUPT.VARRECLE.STAL'
EZA1701I >>> SITE VARrecfm LRECL=1028 RECFM=VB BLKSIZE=6144
500 'SITE VARrecfm LRECL=1028 RECFM=VB BLKSIZE=6144': command not
understood
EZA1701I >>> PORT port_address
200 PORT command successful.
EZA1701I >>> STOR 'MNUPT.VARRECLE.STAL'
150 Opening ASCII mode data connection for 'MNUPT.VARRECLE.STAL'.
226 Transfer complete.
EZA1617I 63 bytes transferred in 0.020 seconds.  Transfer rate 3.15
Kbytes/sec.
EZA1460I Command:
EZA1736I BINARY
EZA1701I >>> TYPE I
200 Type set to I.
EZA1460I Command:
EZA1736I PUT  'MNUPT.ISPLATE.PROL'
EZA1701I >>> SITE FIXrecfm 16 LRECL=16 RECFM=FB BLKSIZE=27984
500 'SITE FIXrecfm 16 LRECL=16 RECFM=FB BLKSIZE=27984': command not
understood
EZA1701I >>> PORT port_address
200 PORT command successful.
EZA1701I >>> STOR 'MNUPT.ISPLATE.PROL'
150 Opening BINARY mode data connection for 'MNUPT.ISPLATE.PROL'.
226 Transfer complete.
EZA1617I 30512 bytes transferred in 0.010 seconds.  Transfer rate
3051.20 Kbytes/sec.
EZA1460I Command:
EZA1736I PUT  'MNUPT.NOTEXIST.PROL'
EZA1684W Local file not found
EZA1460I Command:
EZA1736I ASCII
EZA1701I >>> TYPE A
200 Type set to A.
EZA1460I Command:
```

```
EZA1736I MGET 'MNUP*.KURSLIST.*L' (REPLACE
EZA1701I >>> PORT port_address
200 PORT command successful.
EZA1701I >>> NLST 'MNUP*.KURSLIST.*L'
150 Opening ASCII mode data connection for file list.
226 Transfer complete.
EZA1701I >>> PORT port_address
200 PORT command successful.
EZA1701I >>> RETR 'MNUPP.KURSLIST.STAL'
150 Opening ASCII mode data connection for 'MNUPP.KURSLIST.STAL'(990
bytes).
226 Transfer complete.
EZA1617I 990 bytes transferred in 0.010 seconds.  Transfer rate 99.00
Kbytes/sec.
EZA1701I >>> PORT port_address
200 PORT command successful.
EZA1701I >>> RETR 'MNUPT.KURSLIST.PROL'
150 Opening ASCII mode data connection for 'MNUPT.KURSLIST.PROL'(1980
bytes).
226 Transfer complete.
EZA1617I 1980 bytes transferred in 0.010 seconds.  Transfer rate 198.00
Kbytes/sec.
EZA1460I Command:
EZA1736I GET 'MNUPT.KURSLIST.PROL' 'MNUPT.TEST.PROL' (REPLACE
EZA1701I >>> PORT port_address
200 PORT command successful.
EZA1701I >>> RETR 'MNUPT.KURSLIST.PROL'
150 Opening ASCII mode data connection for 'MNUPT.KURSLIST.PROL'(1980
bytes).
226 Transfer complete.
EZA1617I 1980 bytes transferred in 0.020 seconds.  Transfer rate 99.00
Kbytes/sec.
EZA1460I Command:
EZA1736I QUIT
EZA1701I >>> QUIT
221
```

Port_address and *my_pc_ip_address* are substituted for real IP address values.

The whole job return codes look like this:

```
JOBNAME  STEPNAME PROCSTEP    RC
MNUPT231 FTPMYPC              00
MNUPT231 FTPCHKF              04
MNUPT231 FTPCHKV              04
```

SYSTSPRT FTCPCHKF looks like this:

```
READY
 %FTPCHECK
'MNUPP.KURSLIST.STAL' has 990 bytes
 990 bytes transferred in 0.005 seconds.  Transfer rate 198.00
Kbytes/sec.
'MNUPT.KURSLIST.PROL' has 1980 bytes
 1980 bytes transferred in 0.005 seconds.  Transfer rate 396.00
Kbytes/sec.
'MNUPT.ISPLATE.PROL' has 30512 bytes
 30512 bytes transferred in 0.010 seconds.  Transfer rate 3051.20
Kbytes/sec.
ERROR PUT 'MNUPT.NOTEXIST.PROL' has not been executed
'MNUPP.KURSLIST.STAL' has 990 bytes
 990 bytes transferred in 0.010 seconds.  Transfer rate 99.00
Kbytes/sec.
'MNUPT.KURSLIST.PROL' has 1980 bytes
 1980 bytes transferred in 0.010 seconds.  Transfer rate 198.00
Kbytes/sec.
'MNUPT.TEST.PROL' has 1980 bytes
 1980 bytes transferred in 0.020 seconds.  Transfer rate 99.00
Kbytes/sec.
READY
END
```

SYSTSPRT FTCPCHKV looks like this:

```
READY
 %FTPCHECK V
'MNUPT.VARRECLE.STAL' has 63 bytes
 63 bytes transferred in 0.020 seconds.  Transfer rate 3.15 Kbytes/sec.
ERROR PUT 'MNUPT.NOTEXIST.PROL' has not been executed
READY
END
```

CONCLUSION

In order to avoid controlling FTP steps using execution reports within JCL, it is possible to check FTP success with a REXX procedure in the last step. This procedure provides complete information and facilitates the control of such batch processing.

Vladimir Antonijevic
IT Analyst
Postal Savings Bank (Yugoslavia)

Simple TCP/IP commands

I produced this article for some people who had only just joined the networking team at a site I was working at. Although it's fairly basic, it might help other newcomers.

Here are some useful TCP/IP commands that can be used with Windows to check the connection.

You can find out the name of the system on which you are running the TCP/IP commands by typing in:

```
hostname
```

You can find out your current TCP/IP or UDP connection's status and get statistics using the **netstat** command. It shows both the local and remote name and port of your active network connections. If you want to know what all the available **netstat** switches are, you can add a '?' to the end of the command. If you want to see all the active connections in port order, enter:

```
netstat -n
```

The **ipconfig** command will display your system's current TCP/IP configuration settings. It can also tell you your current DNS servers' addresses. The command is:

```
ipconfig /all
```

The **route** command allows you to view or edit the IP routing table from the command prompt. Windows uses the routing table when it needs to find a path to another TCP/IP host. You can use '?' to display all the available options for the command. To view a system's routing table, enter:

```
route print
```

If you think you've got DNS problems, the command to use is **nslookup**. It's an interactive utility that displays a special

command prompt. To find out all the commands available, enter:

```
help
```

at the command prompt. To get a list of the DNS domain information, you enter **ls**.

To verify that a router path exists between your system and a remote system you can use the **tracert** command. **Tracert** reports the number of jumps necessary to reach a specified destination. To trace a route to www.xephon.com, you would enter:

```
tracert www.xephon.com
```

To view, add, or delete entries in the IP-address-to physical-network-address translation table used by the Address Resolution Protocol (ARP), use the **arp** command. To display the cached IP and MAC addresses on a system, enter:

```
arp -a
```

The command to display a remote Line Print Daemon (LPD) print queue is **lpq**. To display the status of a printer called HPLJ2 (identified by the -P switch on a system named noro2 (identified by the -S switch), you would enter:

```
lpq -Snoro2 -PHPLJ2
```

To test whether a system is connected use the **ping** command. You need to enter the IP address of the system you're trying to communicate with. So to **ping** a system whose IP address is 192.168.100.2, enter:

```
ping 192.168.100.2
```

There's also a command that combines **ping** and **tracert**, it's **pathping**. **Pathping** will ping each router between the originating computer and a target destination. It then records each hop's duration and packet loss. To use **pathping** on www.xephon.com, you would enter:

```
pathping www.xephon.com
```

I hope this starts you off with some ideas on how to diagnose where any problems might be occurring.

Nick Nourse
Independent Consultant (UK)

Different usage of Host On-Demand

If you have to give a browser solution to your Internet users for mainframe access and you don't want to use a TCP/IP stack on an OS/390 mainframe, there is still another option for mainframe access. You can use IBM Host On-Demand and Microsoft HIS Server together to implement a browser-based SNA solution.

This article presents browser-based SNA access to an OS/390 mainframe and gives some detail session configurations.

The following statements describe the applications we will use:

1 IBM WebSphere Host On-Demand – it gives users secure browser access to host applications with Java-based emulation, and supports TN3270 telnet emulation.

2 Microsoft HIS server – which connects SNA networks with PC-based LANs. Host Integration Server allows users to access resources on mainframes.

3 IBM HTTP Server for Windows – IBM Web server (see Figure 1).

Figure1 illustrates how the components work together. We placed the application server in the DMZ and installed IBM HTTP Server, Host On-Demand Server, and Microsoft HIS Server on it.

The main steps in Figure1 are:

1 First of all a remote PC sends a request to the IBM HTTP Server using the URL address https://x.x.x.x/hod/Test.html.

2 The Host On-Demand cache client applet is downloaded to the PC.

3 The remote PC accesses the HIS Server using HOD3270 telnet emulation, and takes a TN3270LU (LUA).

4 An SNA request is sent to the host with LUA (TCP0001).

Now, how should we customize our applications?

IBM WEBSPHERE HOST ON-DEMAND 7.0 CONFIGURATION

Host On-Demand provides a variety of clients for mainframe access (download client, cache client, thin client, etc). First you must decide which type of client you will use. We used HOD cache client and a configuration server model in this case:

1 Configuration Server Definitions – user preferences (macros, keyboard, etc) are saved on the HOD Configuration Server. So we created a new user using https://x.x.x.x/hod/HODAdmin.html, and defined a new telnet session.

Some important 3270 telnet definitions are:

```
Session Name: TEST
       Destination Address: 192.254.x.x
                                        //* SNA server legal ip address
       Destination Port:  4713          //* Telnet port
       LU or POOL name: TCP0001
              //*  We defined TCP0001 as a TN3270 LU in the SNA server.
       Enable Security:  No
```

2 Deployment Wizard – we created a Test.html file and copied it to the HOD publish directory.

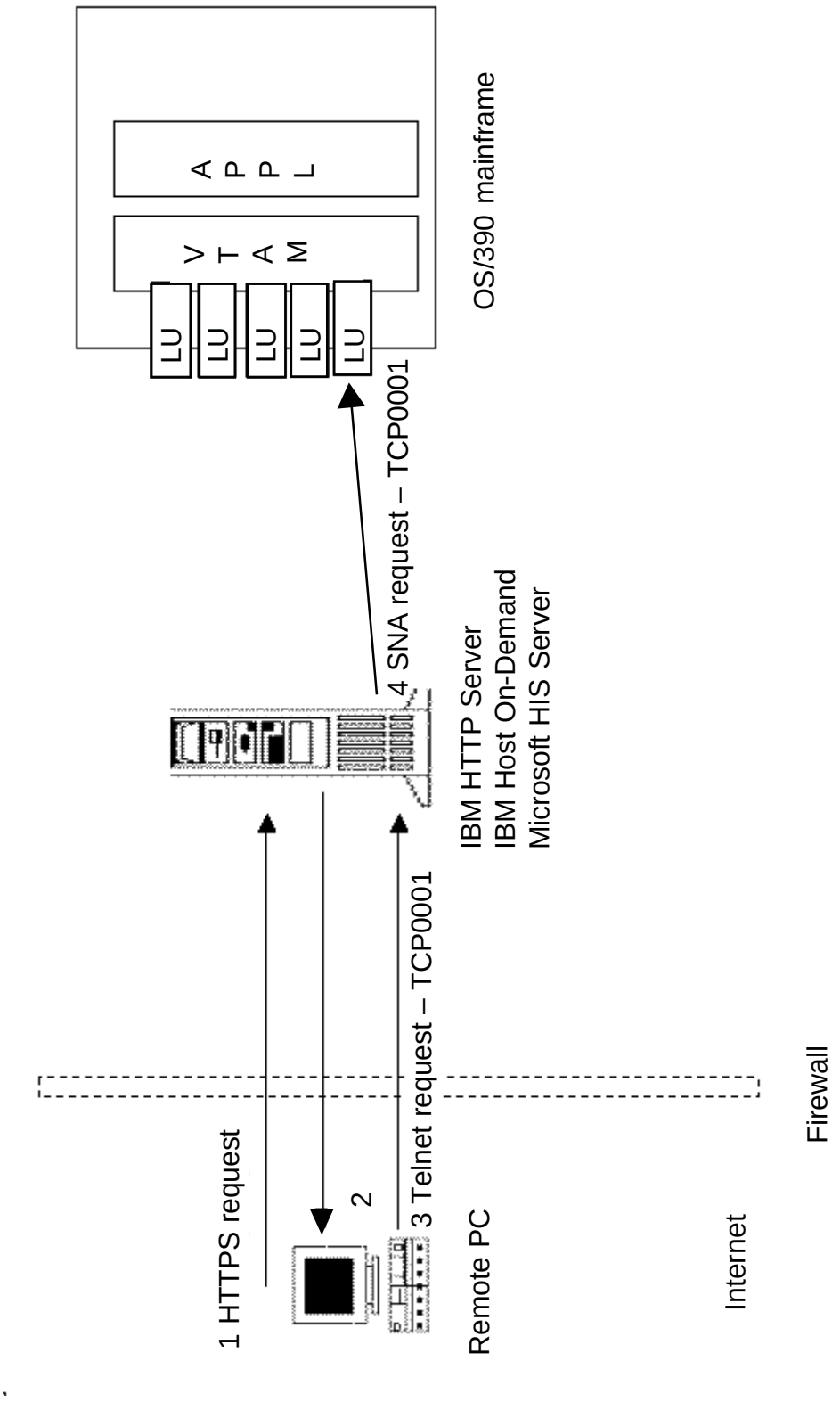

Figure 1: SNA connection to host

MICROSOFT HIS SERVER CONFIGURATION

We configured HIS Server to use TN3270 services. TN3270 communicates with Host Integration Server 2000 using the LUA API. Therefore, all LUs configured for use with the TN3270 service must be LUA LUs.

1 Link Services Definitions – we created a new link service, for example SNADLC1.

2 Connections – we defined a new 802.2 connection (PU) and created a new Application LU (LUA).

   We assigned it to TN3270.

   ```
   LU(LUA) name: TCP0001
   ```

3 TN3270 Service – we configured this service to use telnet port 6531.

   Note: you should also add TCP0001 LUs under VTAM PU definitions.

IBM HTTP SERVER CONFIGURATIONS

We configured HTTP Server to use the https protocol. The following changes were made in the httpd.conf file to make secure https connections and access the Test.html file. If you want to use the https protocol, first you must get a secure certificate from a certificate authority, or you must create your own self-signed certificate.

Sample customization of the httpd.conf file:

```
Alias /hod/   "D:\hostondemand/HOD/"  //* Pass rules for host on demand


Listen 443
//* for https connection
FileETag none
<VirtualHost 192.168.x.x:443>
                          //* illegal ip address is used as virtual host
ServerName  www.yourservername.com
DocumentRoot D:\hostondemand/HOD/
DirectoryIndex test.html
```

```
ErrorLog "|rotatelogs d:/HTTPServer/logs/xx_11255_error.log 14400"
TransferLog "|rotatelogs d:/HTTPServer/logs/xx_11255_transfer.log 14400"
Keyfile d:/httpserver/ssl/test.kdb                  //*keyfile directory
SSLV2Timeout 100
SSLV3Timeout 1000
SSLEnable
SSLClientAuth none
</VirtualHost>
```

Firewall rules

We had to configure the following rulesets on the firewall:

- HOD ports – you should open 8999 and 8989 ports if you want to use the HOD Administrator panel or Configurations server.

Service	Port	Connection Direction
Https	443	Internet => DMZ
Telnet	4713	Internet => DMZ
HODConfigSrvr	8989	Internet => DMZ
HODAdmin	8999	Internet <= DMZ

Pinar Erdim
Network Systems Programmer
Yapi Kredi Bank (Turkey)

If you have ever experienced any difficulties with TCP or SNA, or made an interesting discovery, you could receive a cash payment simply by telling us about it.

More information about contributing an article, plus an explanation of our terms and conditions, can be found at www.xephon.com/nfc.

If you have an idea for an article, please contact the editor, Trevor Eddolls, at trevore@xephon.com.

TCP/IP diagnostic assistance using case studies

INTRODUCTION

Many years are needed to gain expertise in troubleshooting the kind of complex enterprise systems in place today. In this article we will discuss the specifics of resolving TCP/IP network problems under the z/OS operating system. It is not enough just to know the rules of the TCP/IP protocol; different kinds of network equipment and applications (DB2, CICS, MQSeries) interact with the protocol to create unique situations.

Many systems programmers have years of experience in various areas of technology, but they are such specialists that they may be the only one, or one of a handful, in their company with such expertise. They may be experiencing a network problem another systems programmer, thousands of miles away, may have resolved only the previous week.

We are gathering case studies to serve as a repository of knowledge to assist systems programmers in resolving problems. The case study format is interesting because people may be trained to learn in this way. Stories bring learning to life. By telling a good story, you can give someone a piece of your experience in a form they can use. When people go to conferences, their best learning experience may be the ‘war stories’ they share over drinks at the bar at night.

In this article, we will discuss some case studies involving z/OS performance diagnostics. They include:

- TCP Listener throughput
- FTP CSM drops
- IP fragmentation leading to poor response time.

We will also discuss a methodology of categorizing cases and future implementation possibilities.

CASE STUDIES

1: TCP Listener throughput

Problem

We have two CICS Listeners that have performance problems. Both show the same problem. During stress testing, when they reach a transaction rate of 15 per second or so, they start to degrade. As a result, the testing is never able to exercise them at more than about 15 per second.

Suggestions

The following are suggestions to resolve this problem:

1 One place to look is at the 'backlog' number in the Listener. The backlog number may be too small and may be filling up with queued sessions. A small 'backlog' number leads to retransmits in session start-up. Whatever the number is, try increasing it.

2 Check the TCP/IP stack definition for SOMAXCONN. The default is 10. This is the maximum number of connections for the stack that will be used for the listen.

3 You may want to check whether your Listeners are multitasking the incoming requests (ie givesocket(), attach sub task, takesocket(), etc). If not, then the accept queue could reach its maximum because each request was being processed sequentially. If you are already multitasking, then the problem may be on the CICS side.

Variables to examine

In this case, the z/OS SNMP MIB is one source of information on the CICS backlog and the stack configuration parameters:

CICS backlog:

- Accept count – the total number of connections accepted by the CICS Listener.
- Exceed backlog – the total number of connections dropped by the CICS Listener because the backlog is exceeded.
- Current backlog – the current number of connections in backlog.
- Maximum in backlog – the maximum number of connections allowed in the backlog at one time.

You may want to view these parameters not just in real time, but also monitor them over time (historical trend) so you can see whether any connections are being dropped by the Listener during the course of testing.

Stack configuration

The value for the maximum number of socket connections for the stack is available from the z/OS MIB as well as the TCP/IP profile.

2: FTP CSM drops

Problem

A user is GETting a very large file with FTP and failed midway with a timeout error. In the FTP log, the following message may be seen:

```
TO0583 read_ascii_as_image: CSM,GET_BUFFER,failed,rc(fe) reason(12)
```

Some time later, the FTP terminated with the timeout error shown above. I understand the FE reason code means that FTP could not get some of the frames in the buffer request and the second number (12) was the number of frames FTP couldn't get. Is this a problem with CSM?

Suggestions and SNMP MIB variables

FTP uses Communications Storage Manager (CSM) buffers as well as other resources. There may be a temporary shortage of the CSM or the particular type of CSM buffers involved. In this case, the z/OS MIB can provide information on whether and when z/OS IP has discarded packets because of CSM shortages. The fields to check are in the z/OS IP MIB and are as follows:

- In discards memory – the number of IP inbound packets discarded because of a CSM storage shortage.
- Out discards memory – the number of IP outbound packets discarded because of a CSM storage shortage.

You may wish to monitor these variables over time (historical trend) so you can see whether any IP packets are being dropped at times of peak workload. You may also want to set alerts on these fields so that you will be immediately notified if problems occur at any time.

3: IP fragmentation leading to poor response time

Problem

We have an application that uses an Oracle database. Every once in a while, the application seems to send a large burst of traffic and the response time goes up tremendously. We think this is a fragmentation issue. How can we trap this? And, if we can't do anything about it, can we at least see if it is going on so our Help Desk can tell people who call what the problem might be?

Suggestions and MIB variables

Fragmentation is described in the IP public MIB. In this particular case, you need to think about whether the fragmentation is occurring on the z/OS system or somewhere in the network at a router. The IP public MIB is supported by

routers as well as the z/OS system.

CASE STUDY ORGANIZATION

TCP/IP problems are numerous. We have proposed the following initial categories:

- FTP
- Listeners
- Network components
- OSA
- Policy agent
- Print
- Response time
- Routing
- SMTP
- Stack
- TN3270-TCP connections

SUMMARY

We have only scratched the surface of the many problems that may occur on TCP/IP networks. We are actively gathering case studies, and as the number of studies grows, a different organizational structure may be required. Perhaps artificial intelligence techniques such as case-based reasoning, knowledge structuring, and knowledge indexing methods may be the future of this approach. Please feel free to contact us at info@inside-products.com to discuss cases at your shop or to contribute to the casebase.

Nalini Elkins
Inside Products (USA)

Moving datasets between systems

There is often a need to copy datasets from one z/OS system to another. What follows could work between organizations, but the most common situation is between LPARs within the same organization, when the LPARs do not share the same catalog. For example, you may install software to a Test system LPAR and later migrate it to Production LPARs.

ALTERNATIVES

The TSO TRANSMIT (XMIT) command is one possible alternative. But, for large datasets, XMIT has size limitations and performs poorly because it uses the JES*x* spool.

If the source and destination LPARs share at least one DASD volume with sufficient free space, you could move the dataset to the DASD volume, but the dataset will appear uncatalogued to the destination LPAR. This can cause catalog problems.

Even if you catalog it on the destination LPAR, deleting the dataset on the shared volume from either LPAR will leave a non-existent catalogued dataset on the other LPAR. Given that the most likely use of a shared DASD volume is as a common IPL volume, with IODF/IOCDS datasets, its integrity is critical.

Part of Communications Server's TCP/IP component, FTP, is ideal for copying datasets between LPARs. It runs very efficiently, and, between LPARs sharing the same processors, it is extremely fast and can be tuned to be even faster.

FTP

FTP can copy non-VSAM datasets, including those on tape, between LPARs. VSAM datasets can be EXPORTed to sequential datasets, FTPed between LPARs, then IMPORTed

on the destination LPAR. Where FTP really shines is large load libraries with many members; all load module attributes are correctly copied, so long as both LPARs are running at least OS/390 2.10.

FTP can be used as a TSO command from the TSO READY state without ISPF running, or in Option 6 of ISPF/PDF. Like most TSO commands, FTP can also be used from any Option or command line in ISPF/PDF. FTP is also a Unix System Services (USS) command and can be used in batch, either within a batch TSO session (PGM=IKJEFT01) or directly via EXEC PGM=FTP. However, experienced users favour interactive use of FTP as being more intuitive.

There are two ways to use FTP to copy datasets. You can preallocate the dataset and use the REPLACE option on the FTP GET command (although REPLACE is not available for the PUT command), or you can use the LOCSITE or SITE command to set the attributes of the dataset and let FTP create the dataset with the GET or PUT command specified without REPLACE. The process is slightly different for a PDS.

For small datasets, a third option is to just GET or PUT, and live with the FTP defaults. This is often the fastest way to get text, including JCL, from another mainframe or non-mainframe system, or even a workstation. You end up with a variable-blocked sequential dataset, which you can then copy with ISPF Option 3.3 to a new member of a library: a fixed-block PDS. Awkward? Not necessarily, because it usually requires less typing and less thought than doing the GET or PUT directly into the PDS member.

GETTING STARTED

Assuming you are currently logged on to TSO in one of the LPARs, the first thing you will need is the IP address (or domain name, if one is assigned) of the other LPAR. Type the TSO command, FTP, followed by the IP address or domain name, and hit *Enter*. You will then be prompted for your TSO

user ID and password on the other LPAR. Just hit *Enter* if you want to log on to the same user ID on the other LPAR as you are currently logged on to.

```
FTP 10.1.4.16{Enter}
 IBM FTP CS V1R2
 FTP: using TCPIPOE
 Connecting to:  10.1.4.16 port: 21.
 220-FTPD1 IBM FTP CS V1R2 at mvsa800.wcb.ab.ca, 03:07:13 on 2004-03-21.
 220 Connection will close if idle for more than 5 minutes.
 NAME (10.1.4.16:E667800):
{Enter}
 >>> USER E667800
 331 Send password please.
 PASSWORD:
password{Enter}

 >>> PASS
 230 E667800 is logged on.  Working directory is "E667800.".
 Command:
```

The only exception is if you choose to preallocate datasets outside of FTP, rather than using FTP's LOCSITE or SITE command. You may want to do this first, before starting FTP. Three obvious preallocation methods are ISPF Option 3.2, the TSO ALLOC command, or batch JCL with a DD DISP=(NEW,CATLG) statement. Note that the TSO ALLOC command can also be used from within FTP: TSO is a subcommand you can use within FTP to execute a TSO command such as ALLOC.

TRANSFER METHOD

Since defaults are installation-dependent, it is wise to set the transfer method:

```
Command:
EB
 >>> TYPE E
 200 Representation type is Ebcdic NonPrint
 Command:
MO B
 >>> MODE B
 200 Data transfer mode is Block
 Command:
```

EB sets the transfer type to EBCDIC: as-is, without translation. *MO B* sets the transfer mode to block.

These settings work best for all types of data. For example, you could use stream mode for text, but you would lose the ISPF statistics for a PDS member.

THE NEXT STEP – ATTRIBUTES

Unless the destination dataset already exists, the next step is to define its attributes so that FTP will automatically allocate it correctly. The LOCSITE command defines a local dataset to be allocated on the LPAR where you are logged on to TSO. The SITE command defines a dataset allocated on the remote site, with the IP address or domain name you entered with the FTP command.

With a couple of exceptions, LOCSITE and SITE share the same list of possible parameters. The commonly-used allocation parameters are:

- ASAtrans
- BLKsize=size
- BLocks
- CYlinders
- DATAClass=dataclass
- DCbdsn
- DIrectory=size
- FILEtype=SEQ – SEQ means sequential or PDS
- ISPFSTATS
- LRecl=length
- MGmtclass=managementclass
- NOASAtrans

- NOISPFSTATS
- PDSTYPE=PDS or PDSE
- PRImary=amount
- Qdisk – displays available space on a DASD volume
- RECfm=FB, VB, U, FBA, etc
- RETpd=days
- SECondary=amount
- STORclass=storageclass
- TRacks
- UCOUNT=units – number of devices to concurrently allocate; P specifies parallel mount request
- Unit=unittype
- VCOUNT=volumecount
- VOLume=volser or (volser,volser).

Upper-case is the shortest abbreviation. As shown in the following example, FTP does not share the block size allocation defaults of ISPF, so you will want to specify them too:

```
Command:
LOCSITE CY DI=35 LR=8Ø PRI=45 SEC=15 REC=FB
 BLOCKSIze must be a multiple of LRecl for FB data sets.  BLOCKSIze set
to 616
 Ø.
```

To determine the attributes of an existing dataset, you can use the DIR command:

```
DIR
 >>> PORT 1Ø,1,4,19,7,67
 2ØØ Port request OK.
 >>> LIST
 125 List started OK
 Volume Unit    Referred Ext Used Recfm Lrecl BlkSz Dsorg Dsname
 APUB34 339Ø   2ØØ4/Ø3/21  1   15  FB      8Ø 2792Ø  PO  ACTIVE.CNTL
```

```
 APUB02 3390   2004/01/02  1   15  VB     255 27998  PO  ACTIVE.DATA
 Migrated                                              ACTIVE.EXEC
 250 List completed successfully.
 Command:
DIR 'SYS1.PARMLIB'
 >>> PORT 10,1,4,19,7,70
 200 Port request OK.
 >>> LIST 'sys1.parmlib'
 125 List started OK
 Volume Unit    Referred Ext Used Recfm Lrecl BlkSz Dsorg Dsname
 ASYSL1 3390   2004/03/19  1  150  FB      80  3120  PO  'SYS1.PARMLIB'
 250 List completed successfully.
```

DIR alone displays datasets on the remote LPAR having a High-Level Qualifier (HLQ) of the user ID you entered at FTP's NAME prompt when you first connected. Initially, when you connect, and unless you have navigated the FTP directory structure, the dataset name will be displayed without the high-level qualifier.

For datasets with other HLQs, specify the DSN in single quotes. You can also use the asterisk (*) and per cent sign (%) as wildcard characters. In contrast to DIR, the LS command lists just the DSNs, unlike non-mainframe platforms' FTP.

If you are replacing a load library, you may have to allocate the new one on the same DASD volume as the old, if the APF, link list, or other entry in SYS1.PARMLIB specifies the DASD volume serial number (VOLSER) rather than SMS.

SEQUENTIAL

For sequential datasets, you would use a PUT command after a SITE command to allocate and copy the dataset from the local LPAR to the remote one. GET after LOCSITE allocates and copies the dataset from the remote LPAR to the local one you are logged on to in TSO.

```
 Command:
LOCSITE REC=FB
 BLOCKSIze must be a multiple of LRecl for FB data sets.  BLOCKSIze set
to 614
 4.
```

```
 Command:
GET MY.DATASETS
 >>> PORT 10,1,4,19,7,73
 200 Port request OK.
 >>> RETR MY.DATASETS
 125 Sending data set E667800.MY.DATASETS FIXrecfm 132
 250 Transfer completed successfully.
 4288 bytes transferred in 0.280 seconds.  Transfer rate 15.31 Kbytes/
sec.
 Command:
SITE TR PRI=1
 >>> SITE TR PRI=1
 200 SITE command was accepted
 Command:
PUT 'E667800.DOCTEXT.DATA'
 >>> SITE FIXrecfm 80 LRECL=80 RECFM=FB BLKSIZE=7200
 200 SITE command was accepted
 >>> PORT 10,1,4,19,7,75
 200 Port request OK.
 >>> STOR 'E667800.DOCTEXT.DATA'
 125 Storing data set E667800.DOCTEXT.DATA
 250 Transfer completed successfully.
 4207 bytes transferred in 0.005 seconds.  Transfer rate 841.40 Kbytes/
sec.
```

If you preallocated the dataset, use the REPLACE option rather than SITE or LOCSITE.

```
 Command:
GET MY.DATASETS (REPLACE
 >>> PORT 10,1,4,19,7,76
 200 Port request OK.
 >>> RETR MY.DATASETS
 125 Sending data set E667800.MY.DATASETS FIXrecfm 132
 250 Transfer completed successfully.
 4288 bytes transferred in 0.260 seconds.  Transfer rate 16.49 Kbytes/
sec.
```

ALLOCATE THE PDS

Unfortunately, you cannot GET or PUT a PDS, but you can MGET * and MPUT * all members of a PDS. You could:

- Preallocate the destination PDS outside of FTP.
- Preallocate using TSO ALLOC within FTP.

- Use the SITE and MKDIR commands to allocate the dataset on the remote LPAR.
- Use the LOCSITE and LMKDIR commands to allocate the dataset on the local LPAR.

For example:

```
 Command:
LOCSITE CY DI=35 LR=8Ø PRI=45 SEC=15 REC=FB BLK=2792Ø
 Command:
LMKDIR ARCH.CNTL
 E667800.ARCH.CNTL created.
CD or LCD into the PDS
```

Keeping track of where you are in the FTP directory hierarchies on both the local and remote systems is essential. CD, CDU, and LCD allow you to move around. Assuming that your current positions on both the local and remote LPAR are the default, just below your HLQ, here is what happens with the following sequence of commands:

- CD ARCHIVE.CNTL moves down two levels to hlq.ARCHIVE.CNTL on the remote LPAR.
- LCD ARCHIVE.CNTL moves down two levels to hlq.ARCHIVE.CNTL on the local LPAR.
- LCD ARCHIVE.CNTL moves up one level to hlq.ARCHIVE on the local LPAR.
- CD ARCHIVE.CNTL or CDU moves you up one level to hlq.ARCHIVE on the remote LPAR.
- CD ‘SYS1.PARMLIB’ goes directly into the remote LPAR’s system parameter library, SYS1.PARMLIB.

Here is how FTP responds:

```
 Command:
LCD ARCH.CNTL
 Local directory name set to partitioned data set E667800.ARCH.CNTL
 Command:
CD ARCHIVE.CNTL
 >>> CWD ARCHIVE.CNTL
 25Ø The working directory "E667800.ARCHIVE.CNTL" is a partitioned data
```

```
set
```

LOAD LIBRARIES

Load libraries have the best support within FTP. By default, the attributes of each load module are transferred correctly. Even aliases are automatically copied with MGET * or MPUT *. Best of all, for load libraries with large numbers of members, FTP is very quiet by default, not displaying one or more messages per member, as it does with other PDSs.

```
DIR DB2D.LOADLIB
 >>> PORT 10,1,4,19,7,92
 200 Port request OK.
 >>> LIST DB2D.LOADLIB
 125 List started OK
 Volume Unit    Referred Ext Used Recfm Lrecl BlkSz Dsorg Dsname
 APUB11 3390   2004/03/21  1   15  U    32760 32760  PO  DB2D.LOADLIB
 250 List completed successfully.
 Command:
LOCSITE REC=U DI=15 LR=32760 BLK=32760
 Command:
LMKDIR DB2D.LOADLIB
 E667800.DB2D.LOADLIB created.
 Command:
LCD DB2D.LOADLIB
 Local directory might be a load library
 Local directory name set to partitioned data set E667800.DB2D.LOADLIB
 Command:
CD DB2D.LOADLIB
 >>> CWD DB2D.LOADLIB
 250-The working directory may be a load library
 250 The working directory "E667800.DB2D.LOADLIB" is a partitioned data
set
 Command:
MGET *
 >>> XLMT PDS 0 *
 250 PDS 1400 - send next command for load module transfer
 >>> PORT 10,1,4,19,7,93
 200 Port request OK.
 >>> RETR *
 125-Transferring load module
 125 DCB 32760 32760
 250 Transfer completed successfully.
 2598 bytes transferred in 0.005 seconds.  Transfer rate 519.60 Kbytes/
sec.
```

There are a couple of restrictions. You cannot copy from PDS to PDSE or *vice versa*. And if FTP renames a load module during a copy, the renamed module will not be executable. For example, GET ALLOC ALLOC1 copies load module ALLOC and renames it to ALLOC1 in the destination load library.

OTHER PDS

Other than load libraries, transferring large numbers of PDS members can be tedious. Even with the PROMPT command to prevent FTP from asking you whether each member should be transferred, multiple lines of display are generated for each member transferred:

```
Command:
PROMPT
 Interactive mode is off
 Command:
MGET *
 >>> PORT 10,1,4,19,7,82
 200 Port request OK.
 >>> NLST *
 125 List started OK
 250 List completed successfully.
 >>> PORT 10,1,4,19,7,83
 200 Port request OK.
 >>> RETR ARCDEL
 125 Sending data set E667800.ARCHIVE.CNTL(ARCDEL) FIXrecfm 80
 250 Transfer completed successfully.
 4920 bytes transferred in 0.210 seconds.  Transfer rate 23.43 Kbytes/
sec.
 >>> PORT 10,1,4,19,7,84
 200 Port request OK.
 >>> RETR ARCHIVE
 125 Sending data set E667800.ARCHIVE.CNTL(ARCHIVE) FIXrecfm 80
 250 Transfer completed successfully.
 984 bytes transferred in 0.005 seconds.  Transfer rate 196.80 Kbytes/
sec.
```

If you forget to type the PROMPT command, you can respond to any prompt with an S to Stop prompting. But I'm still looking for a better answer than repeatedly hitting *Enter* to clear screen after screen of displayed messages. Neither obvious answer is that much less tedious:

- Run it in batch
- Use IEBCOPY to create a sequential dataset from the PDS.

TAPE AND TIMEOUTS

Datasets on tape can also be copied from one LPAR to another. Listed above, the commonly-used allocation parameters of the SITE (remote LPAR) and LOCSITE (local LPAR) commands include tape-related parameters like Unit and VOLume. Not listed above, SITE AUTOMount and LOCSITE AUTOM should also be specified to ensure FTP can mount tapes on both systems.

You may also need SITE AUTORecall and LOCSITE AUTOR to allow the automatic recall of DFSMShsm-migrated datasets, though it is normally the installation default. Recalls from tape can see FTP timeout the connection; five minutes is a common installation default. If you want a 20-minute timeout, begin your FTP session with the TImeout parameter:

```
FTP 10.1.4.16 (TI 1200
```

And finally, to end your FTP session, type the **quit** command.

CONCLUSION

The place to start reading about FTP is the IBM manual *z/OS Communications Server IP User's Guide and Commands Reference* (SC31-8780). As well as more details about more FTP commands, you will also find methods, such as FTP.DATA, of changing the defaults just for your user ID, in order to avoid typing the same SITE, LOCSITE, and other commands every time you use FTP.

Jon E Pearkins
Adiant (Canada)

Converting to Ethernet

Ethernet, which as recently as 1999 was still considered the 'un-IBM LAN' that clearly separated big blue IBM shops from enterprises that used other systems, has now, however, become the recommended strategic networking scheme for mainframe environments. Of that there can no longer be any doubt or debate. IBM has made it very clear over the last two years that Gigabit Ethernet is the way forward. Confirmation of this, if required, can be found on IBM's description of the OSA-Express Token Ring adapter (which can be located on IBM's Web site in the mainframe server networking section).

After describing some of the benefits that the OSA-Express Token Ring adapter can offer (eg100Mbps connectivity), IBM goes on to say: 'It should be noted that OSA-Express Gigabit Ethernet running QDIO for TCP/IP remains the optimum path into zSeries for networking traffic. Using the SNA to TCP/IP integration technologies TN3270 Server and Enterprise Extender, the customer can transport most SNA traffic over the higher speed TCP/IP Gigabit Ethernet connection into zSeries.'

IBM's total commitment to Ethernet, at the expense of Token Ring and ATM, was further confirmed in IBM's 7 April 2004 (ie the 40th birthday of mainframes) announcement of the new z890 mainframe. One of the important new features announced for the z890 and the z990 was the OSA-Integrated Console Controller (OSA-ICC). This is the ability to attach PCs, as system operator consoles, capable of IPL and multi-LPAR management, to z890s and z990s via an OSA-Express port. OSA-ICC, however, works only with OSA-Express Ethernet adapters, ideally one of the Gigabit variants. The PCs that are to act as system consoles are also expected to run tn3270(E) emulation, as opposed to end-to-end SNA.

Cisco, IBM's strategic networking hardware partner since their formal August 1999 alliance, reinforces this theme further. A

document entitled *Token Ring-to-Ethernet Migration*, which can be currently found in the *IBM/SNA Solutions* section of their Web site, has this to say on page 1: 'Cisco has already announced the "end of sale" of many Token Ring platforms, and many other Cisco Token Ring products are rapidly approaching their end of life because of a greatly diminished number of customers still using Token Ring products.'

The die is thus cast. Token Ring, including the novelty 100Mbps High Speed Token Ring (HSTR) variant, is being displaced by Fast- and Gigabit Ethernet – with Cisco, and others, already offering full-duplex, 10Gbps Ethernet ports. 40Gbps Ethernet is already being talked about, and 100Gbps is obviously the next major goal. The still relatively large Token Ring installed base within the TCP/SNA world has to, in time, accede and conform to this irrevocable trend to Ethernet-based networking.

The goal of this article is to provide pertinent information to those who will be tasked with making a success of a move to Ethernet-based SNA/TCP networking – in a mainframe-centric environment. Such a migration will inevitably necessitate evaluating, at least, issues related to:

- SNA over Ethernet.
- IBM 3745/46 replacement now that IBM has officially discontinued them.
- HPR/IP (aka Enterprise Extender) and Cisco's SNA Sw.
- Absence of OSA-2 support on z990s.
- QDIO or not to QDIO.
- OSA-Express or channel-attached Cisco router with CIP/CPA.
- Token Ring coexistence.
- Cabling and possible use of Wi-Fi.

Invariably, some of the above issues will be more pressing than

others; for example high-speed mainframe interface using OSA-Express or OSA-2.

To facilitate the 'peck-and-pick' approach to reading this article, the remainder of it will be structured in the form of modular, self-contained sections – each with a descriptive title. It is thus not necessary to read the rest of this article in a sequential manner. Just locate the section that you are most interested in.

GRADUAL MIGRATION (USING DLSW) IS OK

The good news, to begin with, is that any migration to Ethernet-based networking, whether it be to 100/10Mbps Fast Ethernet or Gigabit Ethernet, can always be achieved in a piecemeal, one-step-at-a-time, fashion. There is never a need or rationale for a wholesale, do-or-die, everything at once, cutover. There are no hard deadlines so to speak. This is not a Y2K conversion.

Ethernet (irrespective of the speed) and 4/16(/100)Mbps Token Ring LANs can coexist, whether at the data centre itself, on a campus, or a remote office. The necessary technologies for such coexistence, which begins with what is called translational bridging, have existed in a proven form since the early 1990s. Today, coexistence is best realized using a multimedia router – such as those available from Cisco (where a multimedia router is a router that supports both Token Ring and Ethernet ports). A router, in addition to handling the media coexistence, will also perform the necessary layer 2 and layer 3 conversions to guarantee optimum end-to-end results.

In a coexistence-based migration scenario, the major concern is going to be the transport of SNA/APPN and NetBIOS over the Ethernet portions of the network. This is the reason for recommending a router-based approach, rather than translational bridging (which is now *passé*) or translational switching. Data Link Switching (DLSw), a 'protocol-encapsulation' method pioneered by IBM in 1992 on its star-crossed 6611 bridge/routers, is now the best approach for safely transporting SNA/APPN and

NetBIOS, end-to-end, in a mixed-media environment. DLSw, which, as RFCs 1434, 1795, and 2216, became IETF endorsed Internet-standards in the mid-1990s, is available on most enterprise-class routers.

Routers support DLSw across WANs. Thus multimedia routers DLSw can be used to interconnect Token Ring LAN segments to Ethernet LAN segments in any and all enterprise network configurations – including those using Virtual Private Networking (VPN). Token Ring and Ethernet coexistence and interoperability, using DLSw, has been a common feature of IBM world networking since the mid-1990s and is now thought of as an 'off-the-shelf' commodity technology.

SYNERGISTICALLY COMBINING HPR/IP AND DLSW

DLSw is not supported on mainframes; ie there is no mainframe software (at least from any of the big players) that will encapsulate/de-encapsulate DLSw traffic within a mainframe. Instead, IBM's HPR/IP (also known as EE) is the strategic means to transport IP encapsulated SNA/APPN traffic in and out of a mainframe. (Transporting NetBIOS to/from mainframes is not a major requirement and most IBM customers are happy just to have NetBIOS terminated at a data centre LAN segment.)

HPR/IP is an integral feature of all of IBM's Communication Server and PComm offerings. Cisco also supports it via its SNASw capability. Thus HPR/IP could also be used, end-to-end, for Token Ring/Ethernet interoperability in place of DLSw. Typically DLSw proves to be a simpler lower-overhead approach. There are only two, increasingly esoteric, scenarios where HPR/IP will prove to be better than DLSw. These are:

1 HPR/IP will support SNA Class-Of-Service-based (COS) end-to-end traffic priority across the network.

2 HPR/IP will support APPN/HPR network node oriented end-to-end routing across a network with multiple APPN/HPR 'end nodes'.

DLSw and HPR/IP are not mutually exclusive. Thus, particularly with Cisco routers, it is possible also to realize HPR/IP-to-DLSw interoperability. This invariably proves to be the optimal solution if one needs to transport IP-encapsulated SNA/APPN to/from a mainframe. Just use HPR/IP on the last leg between the mainframe and co-located data centre router. Use DLSw between the data centre and the rest of the network. Terminate all the DLSw traffic at the data centre router and then SNASw to convert it into HPR/IP compatible form. Cisco supports this, as a standard requirement, with the so-called SNASw EE with DLSw+ feature.

WHY THE SWITCH FROM TOKEN RING?

With the advent of LAN switching in the mid-1990s, Token Ring started to lose the performance-related advantages it had over Ethernet. Ethernet, with its trademark Carrier Sense Multiple Access with Collision Detection (CSMA/CD) technology, is a contention-based communication scheme. Contention-based, in this context, means that access to the physical media (ie the actual LAN cables) is not controlled by a 'poll/select' type mechanism. Instead, multiple stations can try and access the media at the same time, resulting in data collision and corruption. This is what CSMA/CD is all about. When data collision is detected by the hardware, all the stations involved back-off, wait for random periods of time, and then try to re-transmit the traffic. This, obviously, all works, but as the traffic on a shared (ie hub-based) Ethernet increases, so would the probability of collision. This would result in performance degradation as shared Ethernet LANs got larger and busier.

IBM, with one notable exception in the late 1960s when it came to 2780/3780 RJE, has always abhorred contention based 'hit-or-miss' networking schemes. Instead, IBM advocates controlled deterministic access. Hence the master/slave polling schemes in BSC and SDLC. The token in Token Ring was also a polling mechanism of sorts. Only the station that was in

control of the token could transmit data on the LAN. This immediately precluded collisions and the resulting performance degradations.

Switched Ethernet mitigated the performance issues associated with shared Ethernet LANs. When switches started to displace hubs, in the mid-1990s, as the preferred means for realizing enterprise LANs, Token Ring could no longer justify its considerably higher cost just on the grounds that it offered better performance. This was the beginning of the end. However, Token Ring had other technical merits. It supported larger frame sizes than Ethernet or Fast Ethernet. But this, in most cases, did not translate into palpable gains in response times because neither SNA nor NetBIOS was designed to exploit these large frame sizes. Most SNA traffic rarely exceeded 1,204 bytes per block, and such blocks were well within the 1,500 byte frame size of standard Ethernet.

The ability to support multiple paths across a network, as epitomized by the 'dual-TIC' configurations (ie two concurrently active Token Ring interfaces) on channel-attached 37*xx* FEPs, was the other unassailable advantage of Token Ring over Ethernet. Ethernet, with its transparent bridging mechanism for LAN interconnection, does not permit multiple, concurrently-active paths across a network. Ethernet actually uses an algorithm referred to as 'spanning tree' to detect and eliminate duplicate paths through a network. Rather than rely on alternative paths at Layer 2, IP networks use router technology to realize alternative routes at higher levels. With today's move towards IP-networking, Ethernet's inability to support alternative paths is not viewed as a road-block.

By 1999, the laws of supply-and-demand across the IT market had resulted in 100Mbps Fast Ethernet adapters being an order of magnitude less expensive than 16Mbps Token Ring adapters. Moreover, by then, more and more PCs were being built with a standard 100/10Mbps Ethernet port – in many cases integrated on the motherboard. IBM, always a canny opportunist, was by then a major supplier of Ethernet offerings.

The market demand for 100Mbps HSTR was not compelling. Gigabit Ethernet was already being talked about. IBM saw the writing on the wall and started to de-escalate its support for Token Ring and instead started promoting Ethernet. The rest is history.

DON'T TRIP OVER THE CABLING

SNA customers invariably implemented their Token Ring LANs using Shielded Twisted Pair (STP) wiring following IBM's cabling recommendations. Most large SNA shops thus have a huge investment in STP. Category 5 Unshielded Twisted Pair (UTP), however, is the preferred medium for 100/10 Ethernet. Consequently many Token Ring customers fear that they will have to incur the sizeable cost of rewiring in order to switch from Token Ring to Fast Ethernet. Not so. Ethernet can be used over STP – which in reality is a more robust medium than UTP. The Ethernet standard supports STP, and both Cisco and IBM now certify certain Ethernet offerings for use across STP.

Thus it is indeed possible to switch from Token Ring to Fast Ethernet without having to incur the cost of rewiring. That is a given. So do not let the presence of STP stop you from considering a fast cutover to 100Mbps Ethernet. Plus this could be the perfect opportunity to cut the cord!

Wireless networking is on the ascendant. Just as with Token Ring versus Ethernet, any debate over wired versus wireless networking will be over in five years' time. The latest provisional 802.11g standard already supports 108Mbps wireless networking – which, moreover, is backward compatible with the older (and slower) 802.11b standard. Concerns about security are a red herring. Cisco, and other credible networking vendors, including IBM, now fully endorse Wi-Fi networking. Even at 54Mbps, never mind 108Mbps, Wi-Fi will give you a considerably better performance than a 16Mbps Token Ring. Plus you are never going to encounter wiring hassles.

Wireless is the way of the future. As with Ethernet and TCP/IP,

there is no point doubting and resisting. The momentum is already picking up. Thus, rather than agonize about STP versus UTP, cut your losses now and start considering a Wi-Fi infrastructure – beginning at your small, remote locations.

SNA OVER ETHERNET WITH OR WITHOUT TN3270(E)

The ability to easily and transparently transport SNA/APPN over Ethernet LANs ceased to be an issue worth talking about around 1996. By then all popular SNA offerings supported Ethernet – albeit with one notable exception. This was the Ethernet adapter on the IBM 37*xx*. That was annoying, but was not a show-stopper. By then you could easily get around this by using an IBM 3172 or 3174, both of which could act as mainframe-Ethernet gateways for SNA and TCP/IP traffic. There were also popular and proven non-IBM solutions for Ethernet-centric access to mainframes, in particular Cisco channel-attached (ie CIP) routers and Bus-Tech's NetShuttle family of offerings.

With the advent of IBM's 2216 bridge/router-based Multiaccess-Enclosure (MAE) for the 3746, even the 'not supported by 37*xx*' argument became moot. The 2216, like Cisco's equivalents, was a channel-attachable, multi-protocol, multimedia bridge/router. It permitted a channel-attached approach for getting SNA/APPN traffic to/from a mainframe using 10/100Mbps Ethernet.

Today, getting SNA/APPN traffic to/from a mainframe using 10/100/1000Mbps Ethernet is not an issue. OSA-Express supports it, albeit not in QDIO mode. So do channel-attached, in this case either CIP or CPA, Cisco routers. And as discussed above, HPR/IP can also be used to transport SNA/APPN to/from mainframes, across any LAN media – but in this instance by having the SNA/APPN traffic encapsulated within IP.

All of this said, it should be noted and acknowledged that the inescapable truth is that SNA/APPN is increasingly becoming restricted to data centres, or even just to the mainframe.

Corporations still rely heavily on their complement of SNA mission-critical applications. But the amount of SNA/APPN traffic going end-to-end, mainframe-to-client is declining. Instead of using clients that talk SNA, enterprises are opting for tn3270(E) solutions. Tn3270(E) clients, and IBM's Host On-Demand and PComm are good examples, still act as *bona fide* SNA/3270 clients. But they do not talk SNA 'on the wire'. Instead, they use 3270 data streams over TCP/IP.

A tn3270(E) server, either in the mainframe itself (as with IBM's Communications Server), or in the data centre (eg pSeries machine with IBM's Communications Server, Cisco router, or Bus-Tech NetShuttle) is then used to convert the TCP/IP-based tn3270(E) transactions into *bona fide* SNA request/response units. If a mainframe resident tn3270(E) server is used, ie Communication Server, then no SNA traffic *per se* goes in or out of the mainframe. Instead all the traffic is IP-based. Thus no allowances whatsoever have to be made for SNA traffic. This is IBM's recommended approach – particularly using Gigabit Ethernet in QDIO mode with OSA-Express. For IBM, it also has the advantage that the tn3270(E) server functions are performed using mainframe cycles and memory.

If an external tn3270(E) server is used, then one has to deal with SNA traffic to/from the mainframe. This is not an issue if one still plans to have a 37*xx* around or is using a channel-attached server (eg Cisco router or Bus-Tech NetShuttle). You could also use SNA across OSA-2 or OSA-Express. The only problem here is that OSA-Express works only with IP traffic. You can get around this, as discussed above, using HPR/IP.

3745/3746 REPLACEMENT

IBM officially stopped marketing the now extremely old 3745/3746 communications controllers as of September 2002. The large installed base of channel-attached 37*xx*s, still in the thousands, will, however, be around for a long time yet. Given

that there are Ethernet adapters for the 37*xx*(/MAE) it is possible to use these machines as a mainframe-Ethernet gateway. This could be short-sighted. For a start, any ESCON channel-attached device will have trouble accommodating full-duplex, Fast Ethernet. This is also true for OSA-2, 3172s, 3174s, and all other channel-attached solutions including Cisco channel-attached routers.

ESCON is a 17MB/sec fabric. That is 136Mbps. While this can, in theory, accommodate 100Mbps Fast Ethernet, albeit in half-duplex mode, it is obviously woefully inadequate to handle Gigabit Ethernet. So, cutting to the chase, all future debates about Ethernet attachment to mainframes will eventually boil down to bandwidth issues – particularly the desire to support 1,000Mbps (and faster) Ethernet.

If your short- to mid-term requirement is for just Fast Ethernet then a 37*xx* or any other channel-attached gateway (eg 3172) will suffice. There is a lot of pre-owned channel-attached equipment that can now be easily obtained at very affordable prices. Thus one could even engineer an Ethernet-to-mainframe scheme that uses multiple 3172-type gateways – attached to separate ESCON channels. Such a configuration, front-ended by a good Fast Ethernet switch, will provide multiple 100Mbps Ethernet connections to/from a mainframe.

Where an OSA-Express solution starts to make sense is if you are looking at Ethernet for the long haul. To this end, it is worth keeping in mind that there are no other networking methodologies waiting in the wings to potentially displace Ethernet as the networking scheme of choice. ATM now only has meaning in specialized WAN backbones – in particular the Internet backbone. IBM underlined this by discontinuing ATM support on the OSA-Express adapters for the z990s. Consequently, as with IP, Gigabit Ethernet (and its successors) is going to rule networking for at least the next decade and a half.

OSA-2 OR NOT TO OSA-2

OSA-2, like the 37*xx*, is no longer strategic. That is incontrovertible. OSA-2 was supported only on z900s in the compatibility I/O cage, rather than in the new adapter I/O cage. Z990s and the newly-announced z890 do not support OSA-2 in any form. Thus any new decision involving OSA-2s has to be made against this backdrop.

Though appearing to be an integrated adapter, the OSA-2, in marked contrast to the OSA-Express, does not attach directly to a mainframe's Self-Timed Interface (STI) bus. Instead, an OSA-2 still ends up getting attached to an ESCON channel via a built-in control unit. That is the crux of the problem. The ESCON channel becomes an insurmountable barrier when one tries to move beyond Fast Ethernet.

OSA-2 adapters are, however, installed on many 'older' (eg G5/G6) S/390 machines. Though you could not run these adapters at 1,000Mbps you could, as described above, gain multiple 100Mbps interfaces to a mainframe by using multiple adapters. Again this will all depend on projected network traffic loads in the years to come. If you currently have a G5/G6, but plan to upgrade to a z9*xx* in the next 18 months, it would make sense to make do with OSA-2. On the other hand, it would be counter-productive to think about getting a z900 with OSA-2, knowing that IBM's has already end gamed this adapter.

WHY ALL THE FUSS ABOUT OSA-EXPRESS?

OSA-Express represents the next generation of IBM mainframe I/O technology. It is as simple as that. That is why IBM incessantly promotes it. IBM wants to wean mainframe customers away from ESCON channels as soon as possible. Parallel channels are already out and IBM would like to see ESCON retired in the next five years. FICON and OSA-Express, with their Gigabit capabilities, represent the future.

There are two and a half things that make OSA-Express

different from all previous mainframe networking solutions:

1 OSA-Express attaches directly to a z9*xx* STI bus at 333MB/sec (ie 2,664Mbps) with no intervening control units.

2 OSA-Express supports QDIO-mode operation – when it comes to IP traffic.

2.5 OSA-Express, particularly when supporting the Gigabit Ethernet (GbE) adapter, uses an internal 66MHz, 64-bit PCI bus architecture, rather than the 33MHz, 32-bit architecture used with older adapters.

The bottom line is that OSA-Express, with GbE and QDIO, is the fastest, highest capacity, networking mechanism ever for mainframes. This is the way to eliminate network bottlenecks and accommodate the increasingly larger traffic volumes introduced by Web-related applications.

A z900 can support up to 24 OSA-Express ports – all of which can be GbE, while a z990 doubles that limit. Thus, with this approach, one is unlikely to run out of networking bandwidth in the foreseeable future. There are both Fast Ethernet and 10/100/1000Mbps Ethernet adapters for OSA-Express. There is also support for both fibre infrastructure connections and copper connections. So there are no dead ends. All viable options *vis-à-vis* Ethernet are covered.

QUEUED DIRECT INPUT/OUTPUT (QDIO)

QDIO is a whole new approach for handling networking I/O. It consists of five major features that work synergistically to make networking I/O more efficient and expeditious. These are:

1 Direct Memory Access (DMA) between an OSA-Express and mainframe applications – in particular Communication Servers. Thus networking traffic is moved directly from a GbE adapter to mainframe memory with no intervening steps.

2 IP assist (or IP 'off-loading') processing on behalf of the mainframe, which includes MAC address management, Address Resolution Protocol (ARP) support, routing table maintenance, IP multicast address intervention, and packet filtering.

3 Priority queueing for both inbound and outbound traffic using multiple queues.

4 IP checksum offloading.

5 Inter-LPAR communications if an OSA-Express is shared across multiple LPARs.

QDIO, which supports only IP traffic, is not new. It has been around since 1999. If you are going to use Ethernet, in particular GbE, with OSA-Express, then you might as well think about QDIO. That is what will give you the best of all worlds. The IP-only limitations, as we have seen, are no longer a show-stopper even for SNA applications. Use HPR/IP so that SNA/APPN traffic can be supported across OSA-Express with QDIO.

BOTTOM LINE

Token Ring is now an anachronistic technology. There is no merit or glory in trying to persevere with Token Ring. Given that new workstations now include a built-in 10/100 Ethernet port, trying to prolong Token Ring is just adding unnecessary cost and complexity to contemporary enterprise networking. Thanks to DLSw and HPR/IP, any and all conceivable Token Ring/ Ethernet coexistence scenarios can easily be accommodated for SNA/APPN, NetBIOS, and IP traffic. With OSA-Express, GbE adapters, and QDIO, IBM is offering mainframe customers a whole new way to realize enterprise networks. This is the future. Everything that went before is now yesterday's technology. There are no pitfalls. Ethernet networking, in mainframe environments, is not new. Some embraced it in the mid-1990s. So this is a highly proven approach.

Anura Gurugé
Strategic Consultant (USA)

Attachmate has released Version 4.0.2 of myEXTRA! Smart Connector Mainframe Edition, bringing direct access to mainframe data sources (including VSAM, IMS/DB, DB2, and Adabas), plus native transactional access to CICS and IMS/TM applications. Based on a Service Oriented Architecture (SOA), the product makes legacy assets reusable as services for new application development.

Smart Connector Mainframe Edition provides real-time access to mainframe data sources and uses standard SQL queries to join data across disparate systems. Users can encapsulate mainframe assets as Web services, access the mainframe on a mid-tier server, and/or choose direct connection to the back-end.

For further information contact:
Attachmate, 3617 131st Avenue SE, Bellevue, WA 98006, USA.
Tel: (425) 644 4010.
URL: http://www.attachmate.com/press/press_release/0,1045,4212_1,00.html.

* * *

BMC Software has announced MAINVIEW for IP Version 2.2, which offers IP pacing technology to ensure that mission-critical Internet (TCP/IP) applications receive priority access to IP bandwidth.

The solution simplifies the viewing of the entire TCP/IP stack on the mainframe, including viewing connections, TN3270 statistics, OSA information, detailed FTP statistics, trace route information, ping information, and packet and socket traces.

Mainframe data for Web applications normally travels over TCP/IP networks. MAINVIEW for IP manages these networks by simplifying the monitoring, management, and tuning of mission-critical TCP/IP applications. The IP pacing technology prioritizes the IP traffic so that the most critical applications have the network bandwidth they need to run smoothly.

For further information contact:
BMC Software, 2101 City West Blvd, Houston, TX 77042-2827, USA.
Tel: (713) 918 8800.
URL: http://www.bmc.com/products/proddocview/0,2832,19052_19429_22915_1874,00.html.

* * *

Zephyr has announced the availability of the 2004 release of its PASSPORT HLLAPI Toolkit. HLLAPI is the standard method for a Windows PC application to interface with an IBM mainframe using a TN3270, TN5250, or VT Windows terminal emulation program. Typical HLLAPI applications can present users with a friendlier interface to host applications and can be used to automate routine functions such as file transfer.

PASSPORT can be used to integrate TN3270, TN5250, VT220, and SCO/ANSI host applications with other client, server, or Web applications using HLLAPI. Most PASSPORT customers use HLLAPI to cut and paste data to and from the host session, and have the option of using the PASSPORT HLLAPI Toolkit to do the job. The toolkit contains sample source code that can be used to develop code that will improve end-user productivity.

For further information contact:
Zephyr, 8 E Greenway Plaza, Suite 1414, Houston, TX 77046, USA.
Tel: (713) 623 0089.
URL: http://www.zephyrcorp.com/News/eNews/Mar04.htm.