57 TCP/SNA update

March 2005

In this issue

A xephon PUBLICATION

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

Colin Smith
E-mail: info@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.

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.

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.

Printed in England.

FICON – a basic guide

In this article I will present a basic overview of what FICON is, and try to detail some of the general concepts behind it.

FICON stands for FIbre CONnectivity and is based on standard fibre channel protocols. However, from Layer 4 up FICON has been developed to be unique to the IBM mainframe environment. It is a proprietary I/O protocol developed by IBM for use between IBM (and compatible) mainframes and storage devices. It takes the higher-layer ESCON protocol, analogous to SCSI, and maps it onto a fibre channel transport. FICON offers improved capability over ESCON in the following areas:

- Increased number of concurrent connections – 4,000 channels per control unit.
- Increased distance – maximum of 100 kilometres.
- Increased link bandwidth – full-duplex data rates of 200 and 400MB/sec.
- Increased channel device address support – 16,000 addresses per channel.

FICON, like all other IBM products, has a number of basic terms that are key to its understanding. These follow in the next few paragraphs.

In FICON a node is an endpoint that contains information. It can be a computer, a disk device controller, or a peripheral disk or tape device. Each node will have a unique 64-bit identifier known as a Node_Name.

Each node must have at least one port to connect the node to the FICON topology. This node port is referred to as an N_Port. Each N_Port has a Port_Name, which is a unique 64-bit identifier that is assigned at the time it is manufactured. N_Port is not the only type of port. The others are detailed below:

- E_Port – an expansion port is used to interconnect switches and build a switched fabric.
- F_Port – a fabric port is used to connect an N_Port to a switch that is not loop-capable.
- FL_Port – a fabric loop port is used to connect NL_Ports to a switch in a loop configuration.
- G_Port – a generic port is a port that has not assumed a role in the fabric.
- L_Port – a loop port is a port in a Fibre Channel Arbitrated Loop (FC-AL) topology.
- NL_Port – a node loop port is an N_Port with loop capabilities.

FICON directors are normally referred to as switches. There are, of course, a number of terms pertaining to these, depending on where the director is in the scheme's topology. An entry switch is the FICON Director that is directly connected to the processor's FICON channel and to the control unit that is its destination or another FICON Director.

A cascaded switch is the FICON Director that connects to the control unit and the entry switch.

The entry switch and cascaded switch are connected with an Inter-Switch Link (ISL).

The Switch ID and switch address are used to address a FICON Director. These are 1 byte in length.

A port address, another 1 byte value, is used to address the physical port on the FICON Director.

One or more switches are normally connected to create what is termed a 'fabric'. This is what the N_Ports will be connected to. A switched fabric takes advantage of aggregated bandwidth via switched connections between N_Ports. The port connects to the topology through an FC link. The FC link is a fibre optic cable that has two strands – one is used to transmit a signal

and the other to receive a signal. The FC link is used to interconnect nodes and switches. The port-to-port connection options are shown below:

- Node-to-node link (N_Port-to-N_Port).
- Node-to-switch link (N_Port-to-F_Port).
- Loop node-to-switch link (NL_Port-to-FL_Port).
- Switch-to-switch link (E_Port-to-E_Port).

As mentioned previously, both nodes and ports have unique 64-bit addresses that are used to identify them in a FICON topology. These addresses are assigned by the manufacturer, with a vendor-specific portion defined by the IEEE standards committee. These addresses are called Node_Names and Port_Names, and they are unique world-wide, they are referred to as World-Wide Node_Name (WWNN) and World-Wide Port_Name (WWPN). These unique names are normally written as two sets of hexadecimal numbers, separated by a colon. An example would be 09:43:11:49:41:01:D6:A9.

FICON on IBM zSeries mainframes can operate in one of three modes. These are:

1 FICON conversion mode (FCV).

2 FICON native mode (FC).

3 Fibre Channel Protocol (FCP).

FICON enables multiple concurrent I/O operations to occur simultaneously to multiple control units. This is one of the differences between FICON and ESCON. FICON channels also permit intermixing of large and small I/O operations on the same link. This is a major change when compared with ESCON channels. The data centre I/O configuration now has increased connectivity flexibility owing to the increased I/O rate, increased bandwidth, and multiplexing of mixed workloads. When an application performs an I/O operation to a device represented by a Unit Control Block (UCB), it initiates an I/O request using macros or a supervisor call to the Input

Output Supervisor (IOS). The application or access method also provides the channel program (Channel Command Words, CCWs) and additional parameters in the Operation Request Block (ORB). This request is queued on the UCB. The IOS will service the request from the UCB on a priority basis. A Start Subchannel (SSCH) instruction will be issued by the IOS, with the Subsystem Identification word (SSID) representing the device and the ORB as operands. The Channel Subsystem (CSS) is signalled to perform the operation.

The most appropriate FICON channel is selected by the CSS. The FICON channel will fetch channel programs (CCWs) prepared by the application, fetch data from memory, or store data into memory and present the status of the operation to the application (I/O interrupt). The z/Architecture channel commands, the data, and the status are packaged by the FICON channel into FC-SB-2 (FC-4 layer) Information Units (IUs). IUs from several different I/O operations to the same or different control units and devices are multiplexed or demultiplexed by the FC-2 layer (framing). Another fundamental difference with ESCON is the CCW chaining capability of the FICON architecture. While ESCON channel program operation requires a Channel End/Device End (CE/DE) after execution of each CCW, FICON supports CCW chaining without requiring a CE/DE at the completion of each CCW operation.

On a FICON channel, CCWs are transferred to the control unit without waiting for the first command response (CMR) from the control unit or for a CE/DE after each CCW execution.

The medium of transmission for a FICON interface is a fibre optic cable. This is a pair of optical fibres that provide two dedicated, unidirectional, serial-bit transmission lines. Information in a single optical fibre flows, bit by bit, in one direction. One optical fibre is used to receive data while the other is used to transmit data. Full duplex capabilities are exploited for data transfer. The Fibre Channel Standard (FCS) protocol specifies that for normal I/O operations, frames flow serially in both directions, allowing several concurrent read

and write I/O operations on the same link. You can begin to see how much faster this type of transfer can be. The FC link data rate is 1Gbps (100 MBps) for FICON feature ports, and 1Gbps or 2Gbps (200MBps) for FICON Express feature ports. The 2Gbps link capability will be auto-negotiated by the zSeries server and FICON Director, as well as the Director and Control Units. This is transparent to the operating system and application. With devices in general, the zSeries FICON Express, FICON Director, and/or CU communicate and agree on either a 1Gbps or 2Gbps (100MBps or 200MBps) link speed. This is determined based on the supported speeds in the zSeries server feature, FICON Director, and control unit, as well as the fibre optic cable infrastructure capabilities.

The actual support for FICON requires certain levels of hardware, software, and additional fixes. These are covered in the next few paragraphs.

Native FICON channels on zSeries servers will support cascaded Fibre Channel Directors. This is for a two-Director configuration only. With cascading, a FICON native channel, or a FICON native channel with channel-to-channel (FCTC) function, can connect a server to a device or other server via two native connected Fibre Channel Directors. This cascaded Director support is for all Native FICON channels implemented on FICON features on z900 servers, and FICON Express features on z800, z900, z890, and z990 servers. No additional zSeries hardware is required; however, you do need to have the hardware enabled via an MCL that is a non-cost option from IBM. FICON channels require z/OS V1R3 with PTFs applied, or later releases of z/OS to allow the enablement of cascaded FICON Directors. z/VM V4R4 or later also provides this support. FICON support of cascaded Directors, sometimes referred to as cascaded switching or two-switch cascaded fabric, is for single-vendor fabrics only.

As with anything, cost savings at sites will vary depending on infrastructure, workloads, and size of data transfers. However, IBM reports that in general its customers with data centres

located at two sites may reduce the number of cross-site connections by using cascaded Directors. More savings could be obtained by reducing the number of channels and switch ports. Another important value of FICON support of cascaded Directors is its ability to provide high integrity data paths. The high integrity function is an integral component of the FICON architecture when configuring FICON channel paths through a cascaded fabric.

Operating system support for FICON is provided by IBM across the range. Any zSeries server in Basic mode and LPAR mode also provides support for FICON. The actual operating system base levels are detailed below:

- z/OS – all in-service releases of z/OS.
- z/VM – all in-service releases of z/VM.
- VSE, VSE/ESA Version 2 Release 6, and higher.
- TPF Version 4 Release 1.
- Linux on zSeries.

The actual zSeries servers vary in what support for FICON is offered. There are two types of FICON features available: the zSeries FICON Express and the zSeries FICON features. Both types support a long wavelength (LX) laser version and a short wavelength (SX) laser version – see below:

- z800 servers support a maximum of 16 zSeries FICON Express features (32 ports).
- z900 servers support a maximum of 48 FICON features (96 ports). These can be zSeries FICON Express features, zSeries FICON features, or a combination of both.
- z990 servers support a maximum of 60 features of zSeries FICON Express totalling 120 ports. A maximum of 48 features are supported on a 2084-A08.
- z890 servers support a maximum of 20 features of zSeries FICON Express totalling 40 ports. A maximum of 16 features are supported on a 2086-110.

Note: all of the FICON features occupy one slot in an I/O cage.

The maximum combined number of FICON or FICON Express, OSA-Express, Crypto Express2, PCIXCC, PCICC, and PCICA features supported by a server family are:

- 16 features for the z800 server.
- 48 features for the z900 server.
- 20 features for the z890 server, 16 features are supported on a 2086-110.
- 60 features for the z990 server, 48 features are supported on a 2084-A08.

The main advantages of FICON can be seen by comparing it with ESCON. The following shows how it reduces the need for more ESCON channels but provides similar connectivity. The z900 server can have the equivalent concurrent I/O connectivity of 928 ESCON channels, with 160 ESCON channels and 96 FICON channels installed. The z800 server can have the equivalent concurrent I/O connectivity of 256 ESCON channel connectivity equivalence with 32 FICON channels installed. The z990 server can have the equivalent concurrent I/O connectivity of 1320 ESCON channels, with 360 ESCON channels and 120 FICON Express channels installed. The z890 server can have the equivalent concurrent I/O connectivity of 440 ESCON channels, with 120 ESCON channels and 40 FICON Express channels installed.

Overall, for the large mainframe environment, FICON provides many advantages. It is an essential tool if you have dual site copy in place using PPRC or GDPS.

More information can be obtained about FICON from these IBM publications:

- *FICON Introduction* – SG24-5176
- *zSeries Connectivity Handbook* – SG24-5444
- *IBM zSeries 900 Technical Guide* – SG24-5975

- *Getting Started with the IBM 2109 M12 FICON Director* – SG24-6089
- *IBM zSeries 890 Technical Guide* – SG24-6310
- *IBM zSeries 800 Technical Introduction* – SG24-6515
- *IBM zSeries 990 Technical Guide* – SG24-6947.

John Bradley
Systems Programmer
Meerkat Computer Services (UK)

A C program generic socket listener for OS/390 or z/OS

Communication is a big part of everyday life. We begin a conversation expecting that someone is listening and will respond to our question or request for information. We send out e-mails and we make phone calls, hoping to communicate with friends or colleagues – sometimes in the office just down the hall, sometimes to a business partner across the city, or sometimes to head office located in another country halfway around the world. Many of the millions of computers in existence today function in an analogous fashion – that is, they try to communicate with other computers across the hall, across the city, and at locations around the world. Just as it is necessary for humans to have a common basis for communication, usually the spoken language, for computer systems to communicate, they must also agree on the communication protocol. Today, in most cases, computers will communicate across Internet connections using TCP/IP (Transmission Control Protocol/Internet Protocol) as the common communication protocol and, as with human verbal communication, the target of a computer request for feedback must be listening for requests before there can be a response.

This is commonly referred to as a TCP/IP listener program. Chances are, if you are using a Web browser to surf the Internet or if you use e-mail to communicate with friends and family, you are making use of TCP/IP and TCP/IP listener programs.

THE SOCKET LISTENER

This article provides an example socket listener program written in IBM C and designed for use on OS/390 or z/OS systems. The fundamental listener requirements are all demonstrated. These include the request for a socket assignment by the socket() function call, providing a name for the socket using the bind() function call, and an indication to TCP/IP that the environment is ready to process requests via the listen() function call. Once the application is successfully listening for requests, it will acknowledge receipt of those requests through either a select() or accept() function call. To facilitate testing, this socket listener program is set up as a trivial HTTP server so that a Web browser can be used to send a request to this server program and receive simple responses.

PROGRAM COMPILATION, LINKAGE, AND EXECUTION

The HTTPSRV program is designed to be compiled and pre-linked in IBM C mode. A reentrant (RENT) compile option is recommended. Be sure to convert '[' to X'AD' and ']' to X'BD' in the HTTPSRV C source code before running the compile. See below for an example C compile job:

```
//PROCS    JCLLIB ORDER=(CBC.SCBCPRC)
//STEP1   EXEC EDCC,CPARM='LIST',
// CPARM2='RENT,NOSEARCH',
// CPARM3='NOMAR,NOSEQ,NOOPT,LANGLVL(EXTENDED),SOURCE,LONGNAME,SSCOMM',
//          INFILE=c.source.pds(HTTPSRV),
//          OUTFILE='object.code.pds(HTTPSRVO),DISP=SHR',
//          SYSLBLK=8000
```

The example below provides a sample prelink job. The *c.source.pds* is the dataset containing the HTTPSRV C program, *object.code.pds* represents the target object code

dataset, and *SYSLBLK* should specify the block size of the target object code dataset. The example compile job assumes a pre-existing object code dataset with a BLKSIZE of 8000.

```
//PLKED1   EXEC PGM=EDCPRLK,PARM='UPCASE,OMVS',REGION=2Ø48K
//SYSMSGS   DD  DSNAME=CEE.SCEEMSGP(EDCPMSGE),DISP=SHR
//SYSLIB    DD  DSN=TCPIP.SEZARNT1,DISP=SHR
//SYSOBJ    DD  DSN=object.code.pds,DISP=SHR
//SYSMOD    DD  DSN=object.code.pds(HTTPSRV),DISP=SHR
//SYSOUT    DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSIN     DD  *
  INCLUDE SYSOBJ(HTTPSRVO)
```

A linkedit job that can be used to create the HTTPSRV load module is shown below:

```
//IEWL      EXEC  PGM=HEWLHØ96,PARM='XREF,LIST,MAP,RENT'
//SYSPRINT DD    SYSOUT=*
//SYSUT1    DD    UNIT=SYSDA,SPACE=(CYL,(2,1))
//OBJECT    DD    DSN=object.code.pds,DISP=SHR
//SYSLIB    DD    DSN=TCPIP.SEZACMTX,DISP=SHR
//          DD    DSN=CEE.SCEELKEX,DISP=SHR
//          DD    DSN=CEE.SCEELKED,DISP=SHR
//          DD    DSN=SYS1.CSSLIB,DISP=SHR
//          DD    DSN=SYS1.LINKLIB,DISP=SHR
//SYSLMOD   DD    DSN=load.library,DISP=SHR
//SYSLIN    DD    *
   INCLUDE OBJECT(HTTPSRV)    OBJ MODULE AFTER HTTPSRV PRELINK
   ENTRY   CEESTART
   NAME    HTTPSRV(R)
```

Before the HTTPSRV program is started, an appropriate listener port should be selected. Consult your site's TCP/IP support person to determine an appropriate port number. Also, the HTTPSRV program makes use of USS (Unix System Services) supplied services so the userid that your HTTPSRV program runs under should have an OMVS security segment defined.

HTTPSRV should now be ready to be activated as a TCP/IP listener. Sample JCL to invoke HTTPSRV is shown below – in the example, the listener port is specified as 9010:

```
//HTTPSRV   EXEC PGM=HTTPSRV,PARM='9Ø1Ø'
//STEPLIB   DD   DSN=load.library,DISP=SHR
//SYSPRINT DD   SYSOUT=*
```

When the HTTPSRV program has been activated with this JCL, either through a started task or a batch job, you can test your new listener. To validate the listener's status, you can issue the NETSTAT ALLC command from TSO. If the HTTPSRV listener program is properly active, and assuming that the jobname it is running under is HTTPSRV, you should see output from the NETSTAT ALLC command similar to the following:

```
EZZ2350I MVS TCP/IP NETSTAT CS V2R10       TCPIP NAME: TCPIP
10:56:52
EZZ2585I User Id  Conn     Local Socket    Foreign Socket         State
EZZ2586I -------  ----     ------------    --------------         -----
EZZ2587I HTTPSRV  00001FB0 0.0.0.0..9010   0.0.0.0..0             Listen
```

Depending on the number of TCP/IP applications your site is running, you will probably see several other active jobnames in this list. The existence of the HTTPSRV program's jobname with a state of Listen indicates that the program is successfully listening for requests.

On your workstation, activate your favourite Web browser. For example purposes, let's assume the IP address of the OS/390 or z/OS system that HTTPSRV is running on is 10.0.2.2. To test the HTTPSRV program, issue the following on your browser's *Address* line:

```
http://10.0.2.2:9010/httptest
```

If HTTPSRV is properly contacted, it should return a response to your browser and a window should be displayed that contains the following message:

```
Server test request
```

The only 'valid' request other than httptest that the HTTPSRV program is set up to accept is quit. Any request other than httptest and quit will result in a window display containing the following message:

```
Unknown request type
```

To terminate the HTTPSRV program, issue:

```
http://10.0.2.2:9010/quit
```

That should cause the HTTPSRV job to end on your OS/390 or z/OS system and your browser should display a window with the following message:

```
Server termination request
```

Obviously, you will need to provide the IP address and port number consistent with your system's environment for the browser-initiated requests.

THINGS TO KEEP IN MIND

A few things you should keep in mind when using the HTTPSRV program or other similar tools are:

- Data transmitted via TCP/IP is almost always in ASCII format. For OS/390 or z/OS listeners, expect the client that initiated the request to be talking ASCII and format the response accordingly. That will most likely necessitate ASCII-to-EBCDIC and EBCDIC-to-ASCII translation. See the __atoe() and __etoa() function calls in the HTTPSRV source code.
- If data will be transmitted on an external network, consider using encryption.
- The HTTPSRV program included with this article is a single-threaded listener server. If your server will be handling a high volume, you must consider multi-tasking the listener and using the givesocket()/takesocket() function pair.

CONCLUSION

The HTTPSRV program in this article makes use of some very simple HTML to respond to browser requests directed to it. To be more practical, you could add support to accept commands that request a list of running jobs or the current CPU utilization, or any one of a number of pieces of MVS information you would normally be interested in. This program is meant to

show how easily your OS/390 or z/OS server can fit into today's client/server world.

HTTPSRV.C

```
/*
 *  This program provides a very basic TCP/IP socket listener for
 *  OS/390 or z/OS systems.
 *
 *  Once a connection has been accepted on the socket, any number of
 *  actions are possible. In the case of this sample program, it
 *  acts as a very basic HTTP server. The two operations that the
 *  program can 'act' on are:
 *
 *    GET /httptest
 *    GET /quit
 *
 *  The 'httptest' request will send a simple 'Server test request'
 *  message back to the initiating Web browser.  The 'quit' request
 *  will send a 'Server termination request' message back to the
 *  initiating Web browser and cause termination of the HTTP server
 *  program.  Any other requests targeted to this server program will
 *  cause an 'Unknown request type' message to be sent back to the
 *  initiating Web browser.
 *
 *  This program is not a multi-tasking server and all data is
 *  received and sent in plain text ASCII. Changing this program to
 *  a multi-tasking server with encrypted data transmission would be
 *  an obvious enhancement, however the primary objective of this
 *  program is to demonstrate a socket listener environment for
 *  OS/390 and z/OS.
 *
 *  Once you have compiled, prelinked, and linked this program, you
 *  can activate the HTTP server on your OS/390 or z/OS system. The
 *  TCP/IP port that the server will listen on is specified in the
 *  PARM value passed to the program at start-up as in:
 *
 *  //HTTPSERV EXEC PGM=HTTPSERV,PARM='9010'
 *
 *  In the above case, this HTTP server program will be listening on
 *  port 9010. From a Web browser, you can direct a request to this
 *  HTTP server as follows:
 *
 *  http://ipaddr:port#/requesttype
 *
 *  where 'ipaddr' is the IP address of the system the HTTP server
 *  program is running on, 'port#' is the port number the server is
 *  listening on, and 'requesttype' is either httptest or quit. If
```

```
 *  this HTTP server program is running on a system with an IP address
 *  of 10.0.2.2 and was using port 9010, a browser request would
 *  look like:
 *
 *  http://10.0.2.2:9010/httptest
 *
 *  If you wanted to terminate the server from the browser, enter
 *  the following (from the browser):
 *
 *  http://10.0.2.2:9010/quit
 *
 *  This sample program uses HTML for feedback to the browser, but
 *  you could add whatever you desire based on the command input and
 *  feedback technique of your preference.
 *
 *  Don't forget - the userid that this program is running under will
 *  require an OMVS security segment for successful operation.
 */

#pragma runopts("POSIX(ON)")

#define  MVS

#include <manifest.h>
#include <bsdtypes.h>
#include <socket.h>
#include <in.h>
#include <netdb.h>

#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <tcperrno.h>
#include <stdio.h>
#include <unistd.h>

#define ERROR_SOCKET_CREATE      -1000
#define ERROR_SOCKET_CONNECT     -1001
#define ERROR_SOCKET_PORT_USED   -1002
#define ERROR_SOCKET_BIND_DENIED -1003
#define ERROR_SOCKET_BIND        -1004
#define ERROR_SOCKET_LISTEN      -1005
#define ERROR_ACCEPT             -1007

#define SELECT_WAIT 5
#define BUFSIZE 1023
#define SOCKET_BACKLOG   5

/*********************************************************************/
/*                                                                   */
```

```
/*   doaccept(int *socket#)                                              */
/*                                                                       */
/*   doaccept() waits for a client connection to occur.  A select()      */
/*   is performed waiting for socket connection to become active.        */
/*   The select() has a wait time option that causes the funcion         */
/*   to wake up after a period of none activity.  Modify the wait        */
/*   time value specified in time.tv_sec as appropriate for your         */
/*   environment.                                                        */
/*                                                                       */
/*************************************************************************/
int doaccept(int *s)
{
  int ret_code;
  char msg_buff[256];

  int temps;
  int clsocket;
  struct sockaddr clientaddress;
  int addrlen;
  int maxfdpl;
  struct fd_set readmask;
  struct fd_set writmask;
  struct fd_set excpmask;
  int rc;
  struct timeval time;

  temps = *s;
  time.tv_sec = SELECT_WAIT;              // SELECT_WAIT is 5 seconds
  time.tv_usec = 0;
  maxfdpl = temps + 1;

  FD_ZERO(&readmask);
  FD_ZERO(&writmask);
  FD_ZERO(&excpmask);
  FD_SET(temps, &readmask);

  rc = select( maxfdpl, &readmask, &writmask, &excpmask, &time);

  if ( rc < 0 )
  {
    sprintf(msg_buff, "Error from select\n");
    tcperror(msg_buff);
    printf("doaccept() select() errno: %d...%d\n",
           errno,temps);
    return rc;
  }
  else if ( rc == 0 )                     // Time limit has expired
  {
    return rc;
  }
```

```
  else
  {
    addrlen = sizeof(clientaddress);
    clsocket = accept( temps, &clientaddress, &addrlen );
    if ( clsocket < 0 )
    {
      printf(" doaccept() accept() errno: %d...%d\n",
             errno,temps);
    }
    return( clsocket );
  }
}

/*********************************************************************/
/*                                                                   */
/*   Listener(int portNo, int backLog)                               */
/*                                                                   */
/*   Generic socket listener:                                        */
/*                                                                   */
/*   This subroutine provides a basic, generic socket listener.      */
/*   The port number to be used is passed by the calling routine     */
/*   and this routine acquires a socket, applies a name to the       */
/*   socket using bind(), and readies the socket to accept client    */
/*   connection requests using listen(). At that point, requests     */
/*   sent by clients can be processed.                               */
/*                                                                   */
/*********************************************************************/

int Listener(int portNo, int backLog)
{
  int listener, caller;
  int portArg;
  struct sockaddr_in address;
  int rc;
  struct clientid cid;
  char myname[8];
  char mysname[8];
  int clsocket;

  int ret_code;
  char msg_buff[256];

  int option_value;
  int option_len;

  int accept_fail_count;

  char line[32768] = {0};
  char out_line[8092] = {0};
  int i, n, good_request;
```

```
  i = 256;
  ret_code = maxdesc(&i, &i);
  ret_code = getdtablesize();

  accept_fail_count = 0;

  portArg = htons( portNo );

  memset( &address, 0, sizeof(address) );
  address.sin_family       = AF_INET;
  address.sin_port         = portArg;
  address.sin_addr.s_addr  = htonl(INADDR_ANY);

  memset(&cid, 0, sizeof(cid));
  rc = getclientid(AF_INET, &cid);
  memcpy(myname, cid.name, 8);
  memcpy(mysname, cid.subtaskname, 8);

/*
 *  Acquire a socket.
 */
  listener = socket( AF_INET, SOCK_STREAM, 0 );

  if( listener < 0 )
  {
    printf("socket() failed rc %d errno %d\n",listener,errno);
    return ERROR_SOCKET_CREATE;
  }

/*
 *  Set the socket option to allow reuse of the specified port if
 *  it's for the same application.
 */

  option_value = 1;
  option_len = sizeof(option_value);
  rc = setsockopt(listener, SOL_SOCKET, SO_REUSEADDR,
                  (char *) &option_value, option_len);

/*
 *  Apply a unique local name to the socket.
 */
  rc = bind(listener, (struct sockaddr *)(&address), sizeof(address));

  if ( rc < 0 )
  {
    if ( errno != EADDRINUSE )
    {
      i = errno;
      printf("bind() failed rc %d errno %d\n",rc,errno);
```

```
      close(listener);

      if( i==EINVAL )
        return ERROR_SOCKET_PORT_USED;
      else if( errno==EACCES )
        return ERROR_SOCKET_BIND_DENIED;
      else
        return ERROR_SOCKET_BIND;
    }

/*
 *  Loop for up to a minute if EADDRINUSE is being returned by the
 *  bind() request.
 */
    for ( i=0; i<30; i++)
    {
      sleep(2);
      rc=bind(listener, (struct sockaddr *)(&address), sizeof(address));
      if ( rc == 0 )
      {
        break;
      }
    }

    if ( rc < 0 )
    {
      close(listener);
      return ERROR_SOCKET_BIND;
    }
  }

/*
 *  Ready the socket to accept client connection requests.
 */
  if( listen(listener, backLog) < 0 )
  {
    i = errno;
    close(listener);
    return ERROR_SOCKET_LISTEN;
  }

  for(;;)
  {

/*********************************************************************/
/*                                                                   */
/*   The doaccept() function performs a select(), which will allow   */
/*   this program to "wake up" periodically.  Using doaccept() can   */
/*   provide for more sophisticated operator command processing or   */
/*   management of a multi-tasking support routine environment,      */
```

```
/*   which has not been demonstrated in this example program.        */
/*                                                                    */
/**********************************************************************/

    caller = doaccept(&listener);   /* Do either this call or the    */
                                    /* accept() call below - not both*/

/**********************************************************************/
/*                                                                    */
/*   Enable only one of the doaccept() or accept() function call     */
/*   statements.  doaccept() uses a select() timer to periodically   */
/*   wake up.  accept() will wait indefinitely for an incoming       */
/*   client connection.                                               */
/*                                                                    */
/**********************************************************************/

/*  caller = accept(listener,NULL,NULL); */  /* Do either this call  */
                                             /* or the doaccept()    */
                                             /* call above - not both*/

    if( caller<0 )
    {
      if( errno==EINTR || errno==EMFILE || errno==ENFILE ||
          errno==24 || errno==23 )
      {
        if(errno==EINTR)
        {
          printf("System call ACCEPT interrupted.  Trying again\n");
        }
        if(errno==EMFILE || errno==ENFILE || errno==24 || errno==23)
        {
          accept_fail_count = accept_fail_count + 1;
          if(accept_fail_count >= 15)
          {
            printf("accept() fail limit reached.  Trying again\n");
            accept_fail_count = 0;
          }
        }
        continue;
      }
      else
      {
        break;
      }
    }

    accept_fail_count = 0;

/**********************************************************************/
/*                                                                    */
```

```
/*   Determine whether we have a real accept condition or if we just */
/*    did a timed select() wake up.                                  */
/*                                                                   */
/*********************************************************************/

    if ( caller == 0 )
    {
/*
 *  Use this 'if' block for internal processing.  This code block will
 *  be entered only if the select() function in doaccept() returns
 *  on a time condition rather than an event condition.
 */
    }
    else
    {
      good_request = 1;
      n = read( caller, line, BUFSIZE );
      line[n] = 0;
      __atoe(line);
      printf("Inbound request: %s\n",line);
      if (strncmp(line+5,"quit",4) == 0)
      {
        printf("termination request\n");
/*
 *  Build the HTML response string to indicate that the termination
 *  request has been received.
 */
        strcpy(out_line,
               "<html><head><title>Term request acknowledged</title></
head>");
        strcat(out_line,
               "<b><font face=\"Verdana\" size=\"2\">");
        strcat(out_line,
               "<p align=\"left\">Server termination request</b></
font>");
/*
 *  Convert the response to ASCII and send it back to the browser.
 */
        __etoa(out_line);
        n = write( caller, out_line, strlen(out_line) );

        shutdown(caller,2);
        close(caller);
        shutdown(listener,2);
        close(listener);
        return 0;
      }
      else if (strncmp(line+5,"httptest",8) == 0)
      {
        printf("request type is httptest\n");
```

```
/*
 *  Build the HTML response string to indicate that the test
 *  request has been received.
 */
        strcpy(out_line,
               "<html><head><title>Test request acknowledged</title></
head>");
        strcat(out_line,
               "<b><font face=\"Verdana\" size=\"2\">");
        strcat(out_line,
               "<p align=\"left\">Server test request</b></font>");
/*
 *  Convert the response to ASCII and send it back to the browser.
 */
        __etoa(out_line);
        n = write( caller, out_line, strlen(out_line) );

        shutdown(caller,2);
        close(caller);
      }
      else
      {
        good_request = 0;
      }
      if (good_request == 0)
      {
        printf("Unknown request %s\n",line);
/*
 *  Build the HTML response string to indicate that an invalid
 *  request has been received.
 */
        strcpy(out_line,
              "<html><head><title>Unknown request type</title></head>");
        strcat(out_line,
               "<b><font face=\"Verdana\" size=\"2\">");
        strcat(out_line,
               "<p align=\"left\">Unknown request type</b></font>");
/*
 *  Convert the response to ASCII and send it back to the browser.
 */
        __etoa(out_line);
        n = write( caller, out_line, strlen(out_line) );

        shutdown(caller,2);
        close(caller);
      }
    }
  }

  sprintf(msg_buff, "accept failed\n");
```

```
  perror(msg_buff);

  return ERROR_ACCEPT;
  }

/*
 *  The main routine extracts the program PARM representing the
 *  port number to be used by this listener.  It then calls the
 *  Listener() function to perform the listener dialogue
 *  processing.
 */
main(int argc, char *argv[])
{
  int err;
  int listener_port;

  if (argc != 2)
  {
    return(8);
  }
  listener_port = atoi(argv[1]);
  printf("Listen port is %d\n",listener_port);

  err = Listener(listener_port, SOCKET_BACKLOG);

  return err;
  }
```

Rudy Douglas
System Programmer (Canada)

TCP – some future directions

The success of the Internet has in part been due to the maturity and stability of TCP/IP. The protocol is well-understood and ubiquitous. This article looks at how TCP is measuring up and likely to evolve in the near and mid-term future.

SECURITY

Although TCP is predominantly secure there are some security

issues with both TCP and TCP/IP. A critical vulnerability was discovered in TCP by the UK's National Infrastructure Security Co-ordination Centre in April 2004. The exposure could have allowed hackers to crash vulnerable routers and as a result disrupt Internet traffic. The key issue was that it is much easier to reset TCP/IP sessions using spoofed packets than previously thought. The most seriously-affected infrastructure was routers running Border Gateway Protocol (BGP) because the protocol requires a persistent TCP session between BGP peers. DNS (Domain Name System), SSL (Secure Sockets Layer), and other application protocols are also potentially vulnerable. It should be noted that although the exposure was serious, the Internet did not grind to a halt. Many workarounds existed and fixes were rapidly deployed.

PERFORMANCE

There are signs that some of the largest users are beginning to encounter performance problems with the protocol. This has led some analysts to suggest that poor performance could begin to impact smaller users in the mid-term future, as processor capability and bandwidth increases.

A number of vendors have been working on technologies that aim to improve the TCP architecture. However, most of these are still experimental. Some of the issues with TCP can't be easily corrected, so there may be a need for new protocols or possibly forks in TCP. The IETF is still some way off formally identifying and evaluating these issues, so widespread support for TCP fixes are still a long way off.

SOME RECENT EXPERIMENTAL TCP EXTENSIONS

A number of experimental TCP extensions have been developed. Obviously these are not recommended for deployment, but they may at some point be included in the standard. They include:

- February 2003 (RFC 3465) *TCP Congestion Control with*

Appropriate Byte Counting ABC. Rather than using the number of acknowledgements received, this methodology for congestion control uses the number of bytes acknowledged. It can be applied to Linux.

- April 03 (RFC 3522) *The Eifel Detection Algorithm for TCP*. This experimental extension is used to detect spurious timeouts. This is achieved through the use of timestamps.
- December 2003 (RFC 3649) *HighSpeed TCP for Large Congestion Windows*. This extension modifies TCP's steady state behaviour, allowing very large windows to be used efficiently.
- March 2004 (RFC 3742) *Limited Slow-Start for TCP with Large Congestion Windows*. This extension uses a modified slow-start behaviour in order to reduce data loss when connections use extremely large windows.

For those wishing to view a complete list of RFCs and other documents that define TCP and various TCP extensions that have accumulated in the RFC series, a *Roadmap for TCP Specification Documents* can be found on the IETF Web site (www1.ietf.org/proceedings_new/04nov/IDs/draft-ietf-tcpm-tcp-roadmap-00.txt). The IETF also produces a list of current Internet-drafts, which can be accessed at www.ietf.org/ietf/1id-abstracts.txt.

CONCLUSION

New and faster applications and next-generation network topologies are all impacting transport protocols like TCP. It is conceivable that TCP will need to be changed radically – this could take place within a decade. Of course, if the migration issues and problems that have accompanied the TCPIPv4-TCP/IPv6 changeover are anything to go by, the implications for a significant change to TCP would be monumental.

However, if the performance of TCP degrades, it could be envisaged that there would be sufficient motivation from both

the vendor and use communities to change TCP. However, if there is no reason for change then there will be no change of protocols – it is too big a job.

John Edwards
Network Administrator (UK)

The netstat command

The netstat command displays the network connections, routing tables, the statistics on the interfaces, and other information.

ACTIVE TCP/IP

To know how many TCP/IP stacks are active on the system, from the SDSF (Spool Display and Search Facility) type the following command:

```
D TCPIP
```

An example of output from the command is:

```
COUNT        TCPIP NAME           VERSION           STATUS
----------   ------------------   -------------     -----------
         1   TCP00001             CS V1R4           ACTIVE
         2   TCP00002             CS V1R4           ACTIVE
         3   TCP00003             CS V1R4           ACTIVE
         4   TCP00004             CS V1R4           ACTIVE
```

As shown above, in this system there are four TCP/IP stacks, each of which has a different primary IP address.

This means that this system works with four different networks.

ENVIRONMENT SET-UP

Before performing the commands, it is necessary to allocate the SYSTCPD DDname for one of the TCP/IP stacks obtained

with the command D TCPIP.

TSO environment

From TSO option number 6, type the following command:

```
ALLOC DD(SYSTCPD) DA(tcp-param-dataset) SHR REU
```

JCL environment

To activate the TCP/IP environment in a batch we must place inside the JCL the following card:

```
//SYSTCPD  DD  DSN=tcp-param-dataset,DISP=SHR
```

where *tcp-param-dataset* is the dataset allocated to the SYSTCPD DDname in the procedure that activates the TCP/IP during the IPL.

Here is a TCP/IP start-up example:

```
//TCP001     PROC
//TCPIP      EXEC  PGM=EZBTCPIP,REGION=8000K,TIME=1440
//SYSPRINT   DD  SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//ALGPRINT   DD  SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//SYSOUT     DD  SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//CEEDUMP    DD  SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//SYSERROR   DD  SYSOUT=*
//PROFILE    DD  DISP=SHR,DSN=TCP001.LIB.PARM(PFTCP001)
//SYSTCPD    DD  DISP=SHR,DSN=TCP001.LIB.PARM(PRTCP001)
```

ENVIRONMENTS FOR COMMANDS EXECUTION

The following environments are available for command execution:

1 SDSF environment.

   To perform a display on the TCP/IP, type the following command:

   ```
   /D TCPIP,proc-tcpname,NETSTAT ,CONN
   ```

2 TSO environment.

   From TSO option number 6 type the following command:

```
NETSTAT CONN  TCP  proc-tcpname
```

3 REXX environment.

The following is an example REXX program:

```
If syscall('ON') > 3 then
                      Do
                  Say "Unable to establish the SYSCALL environment"
                  Say "Return Code was " RC
                  Return
                        End
"ALLOC DD(SYSTCPD) DA(tcp-param-dataset) SHR REU"
address syscall
call bpxwunix 'onetstat -c -p proc-tcpname',,out.
Do I=1 to out.Ø
   Say out.i
 End
 exit
```

where *proc-tcpname* is the name of the TCP/IP that appears in the column TCPIP NAME found from the output of the command D TCPIP.

Note:

- The SYSCALL function is necessary to allow the REXX to perform the USS commands.
- The allocation of the DDname SYSTCPD is necessary to establish on which TCP/IP to address the commands.
- The bpxwunix function performs the command netstat in the USS environment.

USEFUL COMMANDS

Below we analyse the options used most frequently with the netstat command.

For simplicity, from now on we will look only at the TSO environment, for which all the following commands have been typed from option number 6.

Help

Command:

```
TSO HELP NETSTAT
```

Description: it provides a detailed description of the netstat command.

Active connections

Command:

```
NETSTAT CONN TCP proc-tcpname
```

where *proc-tcpname* is the name of the TCP/IP that appears in the column TCPIP NAME found from the output of the command D TCPIP.

Description: the CONN parameter provides information on the TCP/IP ACTIVE connections.

Example:

- NETSTAT CONN TCP TCP001
- NETSTAT CONN TCP TCP004.

In the first example the result of the command netstat refers to the active connections with TCP001, while the second example refers to the display of active connections with the TCP004.

The output of the command would look like:

```
User Id    Conn       Local Socket       Foreign Socket        State
--------   --------   ----------------   ------------------    ---------

AF001      00000015   0.0.0..13202        0.0.0.0..0           Listen
BPXOINIT   0000004E   0.0.0..10007        0.0.0.0..0           Listen
CXS00001   0000237F   9.87.230.21..3693  10.10.100.20..9966    Establsh
TCP001     000024D0   9.87.230.123..21   10.10.100.4..1089     Establsh
```

where:

- *User Id* is the name of the started task that has activated the service.

- *Conn* is a number that the system assigns to the connection.
- *Local Socket* is the local address and the port number on which the service is listening.
- *Foreign Socket* is the remote address and the port number through which the client is connected to the service.
- *State* is the connection state.

State values:

- ESTABLISHED – the port has a connection in progress.
- SYN SENT – the port tries to establish a connection.
- SYN RECV – connection initialization.
- FIN WAIT1 – the port is closed and the connection is about to end.
- FIN WAIT2 – the remote port concludes the connection and is waiting for confirmation of the other side.
- TIME WAIT – the port is waiting for the confirmation of the connection's conclusion.
- CLOSED – the port is not in use.
- CLOSE WAIT – the remote port closes the connection and is waiting for confirmation of the other side.
- LAST ACK – the remote port closes the connection and the port it closed: we are waiting for the final confirmation.
- LISTEN – the port is listening, waiting for a connection to be made.
- CLOSING – both the ports are closing the connection but the data has not been sent completely.
- UNKNOWN – the state of the port is unknown.

Local address

Command:

```
NETSTAT HOME TCP proc-tcpname
```

where *proc-tcpname* is the name of the TCP/IP that appears in the column TCPIP NAME found from the output of the command D TCPIP.

Description: the HOME parameter provides information on all addresses defined in the specific TCP/IP.

Example output from the command looks like:

```
Address               Link               Flag
---------------      -------------      -------
9.87.250.13           VLINK4
9.87.11.9             VLINK3             P
9.87.20.1             OSALINK1
9.87.1.2              EZAXCF01
127.0.0.1             LOOPBACK
```

The output shows that on this TCP/IP, four IP addresses are defined:

- 9.87.250.13
- 9.87.11.9
- 9.87.20.1
- 9.87.1.2.

This means that if we perform the command **ping ipaddress**, we get output similar to the following:

```
Pinging host 9.87.250.13
Ping #1 response took 0.000 seconds
```

Telnet connections

Command:

```
NETSTAT TELNET TCP proc-tcpname
```

where *proc-tcpname* is the name of the TCP/IP that appears in the column TCPIP NAME found from the output of the

command D TCPIP.

Description: the TELNET parameter provides information on all the connections with the telnet server inside to TCP/IP.

Example output from the command looks like:

```
Conn      Foreign Socket      State    BytesIn  BytesOut ApplName LuName

00000100 172.25.128.68..3684  Establsh  00438911 14817096 TPX001 T00T001
00002400 192.168.21.3..2752   Establsh  00055303 01026078 TPX001 T00T005
00002426 172.25.128.76..1089  Establsh  00008796 00399237 TPX001 T00T99
```

where:

- *Conn* is a number that the system assigns to the connection.
- *Foreign Socket* is the remote address and the port through which the telnet client is connected to the service.
- *State* is the connection state.
- *BytesIn* is the number of bytes sent by the client to the server.
- *BytesOut* is the number of bytes sent by the server to the client.
- *ApplName* is the application name that has performed the connection.
- *Luname* is the terminal name that has performed the connection.

TCP/IP information

Command:

```
NETSTAT UP TCP proc-tcpname
```

Description: the UP parameter provides information on the date and the time of the TCP/IP start-up.

Example output from the command looks like:

```
Tcpip started at 15:19:29 on 10/11/2004 with IPv6 disabled
```

Session drop

Command:

```
NETSTAT DROP conn-number TCP proc-tcpname
```

Description: the DROP parameter interrupts the session specified by the parameter *conn-number*.

Conn-number is shown by the column CONN in the command NETSTAT CONN or in the NETSTAT TELNET command.

Note: the DROP parameter is usable only by users belonging to the RACF group MVS.VARY.TCPIP.DROP.

Magni Mauro
System Engineer (Italy)

IBM's Communications Controller z/Linux

IBM's Communications Controller z/Linux (CCL), previewed in May 2004, when the Communications Server for Linux (CSL) was being announced, should be available to mainframe customers during the first half of 2005. It thus becomes IBM's second highly strategic z/Linux-based SNA migration offering, indicating that z/Linux is likely to become a pivotal platform for future TCP/SNA networking initiatives. CCL is a software offering that runs on a z/Linux LAPR, which is meant to serve as a replacement for the IBM 3745/3746 communications controllers that were withdrawn from the market in September 2002.

Though the initial preview tended to portray it as the much-needed migration option for SNI customers (since a 37*xx* running ACF/NCP is a mandatory prerequisite), the CCLs scope, in reality, is considerably broader than just SNI. CCL V1, from day 1, will also support other high-end (and in some cases specialized) 37*xx*/NCP capabilities such as XRF (IBM's

SNA-centric Extended Recovery Facility for sophisticated mainframe disaster recovery scenarios), NCP boundary node functionality (eg subarea/local address conversion), Boundary Network Node for Frame Relay interworking, and SSCP takeover. CCL, in effect, is a faithful emulation of 3745/46 hardware in software – to the point that it actually runs existing ACF/NCP software along with a customer's existing ACF/NCP (and for that matter ACF/VTAM) definitions.

The goal of CCL, echoing IBM's once sacrosanct backward compatibility promise when it came to all systems, is to enable 37*xx* customers to continue their investment in ACF/NCP, without any disruption or sacrifice, despite the withdrawal of the 'host' hardware. The actual hardware interface between the CCL software (emulating a 37*xx*) and the physical network will be realized, exclusively, using OSA-Express adapters. Since the latest OSA-Express adapters support only LAN connectivity (and not even ATM any more), there is obviously no direct support for serial links (eg SDLC, Frame Relay, or X.25). If serial link support is a requirement, this has to be realized using an appropriate feature on an SNA/APPN-capable router (eg DLSw), with IBM recommending 3600 family routers from its networking partner Cisco. (CCL does not plan to support X.25 NPSI. The few European and Asian customers that may still have a need for an NPSI capability will have to, yet again, seek support from Cisco, as opposed to IBM.)

Despite their commonality in terms of running on z/Linux LPARs, there is no inherent coupling between CCL and CSL. CSL is thus definitely not a prerequisite for using CCL. CCL as a 37*xx* replacement is squarely targeted at 'classic SNA' scenarios, whereas CSL, as a Communication Server gateway, addresses the needs of TCP/SNA migration and interoperability. It is, however, definitely possible to have both CCL and CSL running, in parallel, on the same z/Linux LPAR – where the CCL will be performing tasks hitherto associated with 37*xx*s and ACF/NCP, while CSL will be delivering interoperability-related functions such as tn3270(E) gateway and Enterprise Extender (EE) relaying.

SNI WITHOUT A 37*XX*

Much of the initial interest in CCL will still undoubtedly focus on its SNI replacement capabilities since it was the SNI customers who obviously felt most exposed and vulnerable when the 37*xx*s were withdrawn. But, ironically, there will be some valid resentment and recrimination, with IBM, quite rightly, getting blamed for not even hinting at the possibility of a CCL-like capability two years ago when the 37*xx* was being withdrawn. Since around 1998, IBM, until it previewed the CCL, had been steadfast in its claims that EE with the Extended Border Node (EBN) feature was the strategic replacement for SNI – even though this meant that customers at both ends of an SNI connection had to adopt APPN in order to use EE. There are those that have already adopted the EE/EBN route and quite a few that were implementing APPN just so that they could use EE/EBN.

The CCL solution for SNI avoids the need for implementing APPN, though on the other hand it requires using a z/Linux partition and the new (and as yet unproven) CCL. Thus there are valid pros and cons to each of the options – though the CCL option, in addition, has the virtue that, unlike the EE/EBN approach, it can be implemented asymmetrically; ie just on one side. This can be extremely important to SNI customers since in most cases the two interconnected SNA networks belong to separate enterprises, each with its own priorities, budgets, and networking strategies. IBM, as is to be expected, is, however, not touting CCL over EE/EBN. It does not advocate that those who have decided to opt for the EE/EBN approach now abandon it in favour of CCL – though many of those customers will obviously feel pressurized to evaluate CCL.

The initial release of CCL will not support DLSw – maintaining the *status quo* that DLSw has to be terminated (and started) at a router outside the mainframe. Much of this had to do with IBM's hitherto backing of EE as the strategic means for routing (as opposed to just transporting) SNA traffic over IP. Thus with CCL V1, customers wishing to use IP connectivity for their SNI connections are told to use external routers in order to obtain

DLSw. IBM, however, plans to support DLSw in a subsequent release of CCL! This is good news, but as with CCL's SNI capability it will be greeted with some level of consternation by some – for, again, very valid reasons. In essence, with CCL, IBM, more or less out the blue, is trying to rewrite its TCP/SNA migration recommendations. It would have been useful to have mainframe DLSw, which unlike EE is not contingent on using APPN, five years ago. It would have been a great adjunct to the OSA Express adapters. Now, when nearly everybody already has a working TCP/SNA infrastructure in place, with DLSw being terminated externally etc, IBM is ready to change its mind.

MINIMAL CHANGES TO SNA DEFINITIONS

IBM claims that a paramount consideration during the design and development stages of CCL was to ensure that any and all changes to VTAM and NCP definitions would be kept to an absolute minimum. Recognizing the strategic importance of the SNI feature, there was particular emphasis on ensuring that cut-over to CCL-based SNI connectivity could be achieved, in all instances, on an asymmetrical basis; ie all changes restricted to one side (the side with CCL).

Obviously, one cannot escape the fact that the CCL as a mainframe resident software package is not a true, one-for-one replacement for a 37*xx* – particularly in that it does not support any serial links whatsoever. Hence all of the NCP definitions have to be modified and updated to reflect total LAN connectivity, via an OSA-Express adapter, with DLSw on remote routers being used to support any serial links still required within the network. Supporting LANs through ACF/NCP is no longer new, with 37*xx* Token Ring support having been viable as of the early 1990s. However, converting the ACF/NCP definitions of a 37*xx* with even some serial ports to work on a CCL, with just OSA connectivity, will still be a tedious exercise requiring the specification of MAC addresses.

In addition to eliminating all serial connection definitions in

NCP, a new PTF will be required to ACF/VTAM. The PTF is to enable VTAM to activate and take ownership of an ACF/NCP that is accessible across an OSA-Express LAN (rather than a channel connection). There will also be a need to ensure that the OSA-Express microcode is relatively recent; ie post-May 2004. It is definitely not as bad as it could have been, but it is not going to be trivial either.

BOTTOM LINE

Overall, the CCL is a great addition to the TCP/SNA arsenal of options, especially with the demise of the 37*xx*s. The primary criticism that one can validly level against IBM is that it was extremely late in informing the customer base of the possibility of such a product. When the 37*xx*s were formally withdrawn there was no talk whatsoever of IBM planning to invest even a dime on a radically new replacement product. All that IBM talked about was making do with existing solutions such as EE, OSA-Express, DLSw etc. Then IBM discovered z/Linux. The plans to introduce mainframe DLSw is a noteworthy about-turn on IBM's part.

Against this background one can indeed also be cynical, with reason, about CCL as well as CSL. With these offerings IBM is trying to generate interest in z/Linux. In essence these are meant to be the 'killer applications' that motivate mainframe customers to start embracing z/Linux. This said, one has to admit that it is good to have a 37*xx* replacement, even if it is totally software-based. It at least gives TCP/SNA customers another option. So the bottom line has to be that one cannot ignore CCL. At a minimum you have to evaluate it to see whether it can play a meaningful role in the future of your TCP/SNA network.

Anura Gurugé
Strategic Consultant (USA)

Diagnosing routing problems via the TCP open sequence

INTRODUCTION

From analysing the TCP open sequence, I have obtained clues that led to detecting improperly configured routes and improving response times and network throughput. It may come as a surprise that a simple thing like examining the TCP Open can help solve such problems. When I do network consulting and diagnostics at various companies, one of the first things I do is examine the Open sequence for some of the important applications for the installation.

First, let's think about TCP architecture and where the Open fits in. Then, I will show some sample TCP Opens from packet traces. Finally, I will conclude by describing a case where we found improper routing and were able to fix it.

TCP VIRTUAL CIRCUIT CREATION

The way TCP works is by creating a virtual circuit between the two ends of the connection – the remote host and the local host. The remote and local hosts are also known as 'client' and 'server' or 'local address' and 'foreign address'. When the two ends need to talk to each other using the TCP protocol, a connection is established, which lasts for some period of time. In fact, the connection lasts for the period of time bounded by the Open and Close. To allow us to talk about this connection, it is called a 'virtual circuit'. All the TCP protocol functions take place in the context of this virtual circuit.

Notice also that this virtual circuit has only two endpoints. You may remember the SNA world with its multi-point circuits. In SNA, a network circuit to an end user site could potentially have a number of endpoints or drops. In TCP, all the circuits are point-to-point. Conceptually, this makes flow control easier to understand.

Everything in TCP happens over this virtual circuit between the two endpoints. By everything, I mean data transmission, flow control, and reliability. The reason TCP is different from UDP is because it provides a 'reliable' method of transmission – that is, it will try to ensure that your packet gets to the other side completely and uncorrupted.

An open sequence and a close sequence, creating this enduring connection, are a temporary marriage, if you will. The TCP open sequence will establish the connection (perform the marriage ceremony) and the TCP close sequence will terminate the connection (perform the divorce). In UDP, a connectionless protocol, none of this happens – you may make your own analogies for this situation!

So, the first thing that has to happen in a virtual circuit is the connection is opened.

TCP OPEN ARCHITECTURE

During the open sequence, TCP packets flow back and forth with various bits of the header turned on. The header is the first 20 or so bytes of the TCP packet, which contains various pieces of control information. Of course, IP adds a header as well. We will look at exactly what the TCP header looks like in a moment. To get back to the open sequence – first, a TCP packet is sent from one side; then, in response, the other side allocates buffers and other resources. This is called the SYN–SYN/ACK sequence, or the TCP three-way handshake. At the end of the handshake, if it is properly concluded, the connection or virtual circuit is ready for data transmission.

In Figure 1, you will see the TCP open sequence. Take a look and then we will discuss how this works.

The first packet sent is the SYN – Host1 sets the SYN bit in the TCP header to request a TCP connection. The sequence number is set to some random number (x). Since the SYN bit is set, this sequence number (x) is used as the initial sequence number.

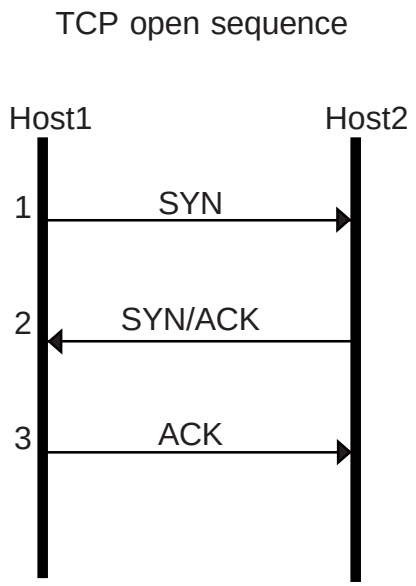

Figure 1: TCP open sequence

The next packet is the SYN/ACK – Host2 sets the SYN and the ACK bits in the TCP header. Host2 sets its initial sequence number as another random number (y). Host2 sets its window to *nn* bytes. The assumption is that it has buffer space for *nn* bytes of data. The ACK sequence number is set to x +1. This says that Host2 expects a next byte sequence number of x+1.

The last packet is the ACK – Host1 acknowledges the segment, completing the three-way handshake. The receive window is set to the receiver's buffer size. The ACK sequence number is set to y + 1 to indicate the next expected sequence number. At this point the client assumes that the TCP connection has been established. Upon receipt of the ACK, Host2 also sets the session to a state of established.

TCP HEADER

The TCP header is shown in Figure 2. The important pieces for us at this point are the Sequence number, Acknowledge number, the bits in the Code or Flag bits, and the Maximum Segment Size (MSS) that is set in the TCP options field.

Octet	*Bits*	*Len*	*Name*	*Comment*
0-1	-	2	Source port	-
2-3	-	2	Destination port	-
4-7	-	4	**Sequence number**	**Position of the last byte we sent**
8-11	-	4	**Acknowledgement number**	**Next byte we expect to receive**
12	0-3	-	Header length	4 bits. TCP header length including Options
12	4-7	-	Reserved	
13	-	1	Code or Flag bits	8 bits (6 are used) bit 0 (URG) Urgent **bit 1 (ACK) Acknowledgement** bit 2 (PSH) PUSH bit 3 (RST) Reset connection **bit 4 (SYN) Synchronize** bit 5 (FIN) Close connection
14-15	-	2	Window	Amount of data we can accept
16-17	-	2	Checksum	Checksum
18-19	-	2	Urgent pointer	Points to urgent data
			TCP options	**Includes Maximum Segment Size**

Figure 2: TCP header

TCP Send and Receive buffers are set in TCP profile and can affect the speed of transmission. The size of the maximum segment that can be sent over the connection is also negotiated here.

TCP MAXIMUM SEGMENT SIZE NEGOTIATION

A TCP host is supposed to send an indication of how large a segment it can transmit in the SYN packet if it is larger than 536. If the MSS is not sent, 536 is assumed. TCP segment size can have a large impact on the network in terms of performance. Remember, each packet has 40+ bytes for the TCP and IP headers. Small segments will suffer from overhead. On the other hand, if you send too large a segment, it may be fragmented by IP because the routers cannot support such a large packet size.

Let’s look at two TCP Opens and the MSS negotiation in each.

SAMPLE TCP OPEN

In the example below, you will see the first packet of an open sequence between two ports:

```
1Ø HOST1     PACKET   ØØØØØØØ4 2Ø:39:Ø4.439ØØ9 Packet Trace
 From Interface    : CTCLINK1          Device: CTC              Full=48
  Tod Clock        : 2ØØ5/Ø1/Ø8 2Ø:39:Ø4.439899
  Sequence #       : Ø                 Flags: Pkt
 IpHeader: Version : 4                 Header Length: 2Ø
  Tos              : ØØ                QOS: Routine Normal Service
  Packet Length    : 48                ID Number: CD1E
  Fragment         : DontFragment      Offset: Ø
  TTL              : 118               Protocol: TCP            CheckSum:
A72Ø FFFF
  Source           : xxx.xxx.xxx.1Ø8
  Destination      : yyy.yyy.yyy.3
 TCP
  Source Port      : 4147  ()          Destination Port: 1Ø28  ()
  Sequence Number  : 58ØØØ253          Ack Number: Ø
  Header Length    : 28                Flags: Syn
  Window Size      : 65535             CheckSum: 9F1C FFFF Urgent Data
Pointer: ØØØØ
   Option          : Max Seg Size Len: 4 MSS: 1412
   Option          : NOP
   Option          : NOP
   Option          : SACK Permitted
IP Header          : 2Ø
ØØØØØØ 45ØØØØ3Ø CD1E4ØØØ 7606A72Ø 41718A6C  C0A8Ø4Ø3
Protocol Header    : 28
ØØØØØØ 1Ø33Ø4Ø4 Ø3763BFD ØØØØØØØØ 7ØØ2FFFF  9F1CØØØØ Ø2Ø4Ø584 Ø1Ø1Ø4Ø2
```

The address is xxx.xxx.xxx.108 with a source port 4147, which is asking to start a connection with destination address yyy.yyy.yyy.3, port 1028. You may be wondering what ephemeral port 1028 is – it so happens that this was a passive FTP transfer to my PC. Notice that the SYN flag is set on. Notice also that the ACK number starts with 0 and the Sequence number starts with a random high number. This will be used as the ‘seed’ for future ACKs. In this entry, you can also see that the MSS or Maximum Segment Size is set to 1,412. This address, xxx.xxx.xxx.108 is suggesting that the traffic between them flows at this size.

In the example below, you will see the second packet of this open sequence:

```
11 HOST1     PACKET   00000004 20:39:04.440825 Packet Trace
 To Interface      : CTCLINK1         Device: CTC              Full=44
  Tod Clock        : 2005/01/08 20:39:04.440816
  Sequence #       : 0                Flags: Pkt Out
 IpHeader: Version : 4                Header Length: 20
  Tos              : 00               QOS: Routine Normal Service
  Packet Length    : 44               ID Number: 869A
  Fragment         :                  Offset: 0
  TTL              : 64               Protocol: TCP            CheckSum:
63A9 FFFF
  Source           : yyy.yyy.yyy.3
  Destination      : xxx.xxx.xxx.108
 TCP
  Source Port      : 1028  ()         Destination Port: 4147  ()
  Sequence Number  : 247810133        Ack Number: 58080254
  Header Length    : 24               Flags: Ack Syn
  Window Size      : 32768            CheckSum: DCC7 FFFF Urgent Data
Pointer: 0000
   Option          : Max Seg Size Len: 4 MSS: 1460
IP Header          : 20
000000 45000020 869A0000 400663A9 C0A80403  41718A6C
Protocol Header    : 24
000000 04041033 0EC54855 03763BFE 60128000  DCC70000 020405B4
```

Notice that the SYN and ACK flags are set on. Notice also that the ACK number, 58080254, is one more than the preceding sequence number of 58080253. This indicates that the previous packet containing sequence number 58080253 was received and the next byte this host expects is 58080254. Host yyy.yyy.yyy.3 now seeds his ACKs with the random high number 247810133, as shown in the Sequence number field. This will be used as the ‘seed’ for future ACKs from the other host. In this entry, you can also see that the MSS is set to 1,460. This is larger than the 1,412 size suggested by address xxx.xxx.xxx.108. This host is saying that it is capable of transmitting larger packets.

In the example below, you will see the last packet of this open sequence:

```
13 HOST1     PACKET   00000004 20:39:04.746849 Packet Trace
 From Interface    : CTCLINK1         Device: CTC              Full=40
  Tod Clock        : 2005/01/08 20:39:04.746838
  Sequence #       : 0                Flags: Pkt
 IpHeader: Version : 4                Header Length: 20
  Tos              : 00               QOS: Routine Normal Service
```

```
  Packet Length    : 40               ID Number: CD20
  Fragment         : DontFragment     Offset: 0
  TTL              : 118              Protocol: TCP            CheckSum:
A726 FFFF
  Source           : xxx.xxx.xxx.108
  Destination      : yyy.yyy.yyy.3
 TCP
  Source Port      : 4147  ()         Destination Port: 1028  ()
  Sequence Number  : 58080254         Ack Number: 247810134
  Header Length    : 20               Flags: Ack
  Window Size      : 65535            CheckSum: 7485 FFFF Urgent Data
Pointer: 0000
IP Header          : 20
000000 45000028 CD204000 7606A726 41718A6C  C0A80403
Protocol Header    : 20
000000 10330404 03763BFE 0EC54856 5010FFFF  74850000
```

Notice the ACK flag is set on. The virtual circuit has successfully been created and data transmission can now start. In this case, we will now see the data file requested begin to be transferred.

ROUTING PROBLEM FOUND

Now, we look at the TCP Open, which led us to discover a routing problem:

```
757 HOST2     PACKET   00000001 07:50:10.150650 Packet Trace
 From Interface    : GBE2             Device: QDIO Ethernet    Full=60
  Tod Clock        : 2004/12/03 07:50:10.150649
  Sequence #       : 0                Flags: Pkt Ver2
  Source Port      : 3886             Dest Port: 5023  Asid: 0066 TCB:
00000000
 IpHeader: Version : 4                Header Length: 20
  Tos              : 00               QOS: Routine Normal Service
  Packet Length    : 60               ID Number: 0F74
  Fragment         :                  Offset: 0
  TTL              : 64               Protocol: TCP            CheckSum:
CACC FFFF
  Source           : xxx.xxx.xxx.5
  Destination      : yyy.yyy.yyy.241

 TCP
  Source Port      : 3886  ()         Destination Port: 5023  ()
  Sequence Number  : 3392023214       Ack Number: 0
  Header Length    : 40               Flags: Syn
  Window Size      : 65535            CheckSum: 6EDE FFFF Urgent Data
Pointer: 0000
```

```
   Option          : Max Seg Size Len: 4 MSS: 8952
   Option          : NOP
   Option          : Window Scale OPT Len: 3 Shift: 1
   Option          : NOP
   Option          : NOP
   Option          : Timestamp        Len: 10 Value: DA182CA6 Echo:
00000000
```

First, notice that in the example above the SYN to open the sequence flows over interface GBE2 while the SYN/ACK, shown below, flows over GBE1:

```
758 HOST1     PACKET   00000001 07:50:10.150761 Packet Trace
 To Interface      : GBE1             Device: QDIO Ethernet    Full=60
  Tod Clock        : 2004/12/03 07:50:10.150761
  Sequence #       : 0                Flags: Pkt Ver2 Out
  Source Port      : 5023             Dest Port: 3886  Asid: 0066 TCB:
00000000
 IpHeader: Version : 4                Header Length: 20
  Tos              : 00               QOS: Routine Normal Service
  Packet Length    : 60               ID Number: F585
  Fragment         :                  Offset: 0
  TTL              : 64               Protocol: TCP            CheckSum:
E4BA FFFF
  Source           : yyy.yyy.yyy.241
  Destination      : xxx.xxx.xxx.5

 TCP
  Source Port      : 5023  ()         Destination Port: 3886  ()
  Sequence Number  : 1441719441       Ack Number: 3392023215
  Header Length    : 40               Flags: Ack Syn
  Window Size      : 65535            CheckSum: 4AD2 FFFF Urgent Data
Pointer: 0000
   Option          : Max Seg Size Len: 4 MSS: 1460
   Option          : NOP
   Option          : Window Scale OPT Len: 3 Shift: 0
   Option          : NOP
   Option          : NOP
   Option          : Timestamp        Len: 10 Value: DA182CA7 Echo:
DA182CA6
```

So, the virtual circuit actually travels over two different interfaces or physical paths. The installation wanted all traffic to and from port 5023, which is used in this virtual circuit, to flow over GBE2. They were not aware that the traffic was actually split over two interfaces.

Notice also that the MSS in the first example starts out at

8,952, but is negotiated down to 1,460! To see the new MSS, take a look at the second example. This was definitely not what they wanted to have happen. After examining the open sequence, we were able to correct the routing and eliminate many communications errors for these connections. The port 5023 was a critical production DB2 application, so this was quite an important solution we were able to implement.

SUMMARY

We have discussed one way to find difficult problems on the TCP/IP network. Many problems may occur on large TCP/IP networks. Other ways exist to find such information, but analysing traces will lead to a deep and fundamental knowledge of your network, so you will be in a good position to resolve problems.

Nalini Elkins
Inside Products (USA)

How to use HPRIP under OS/390

HPR nodes use the basic APPN CP-CP sessions unchanged. HPR network nodes and basic APPN network nodes share a common topology database. Nodes in the APPN network see the nodes in the HPR portion of the network as basic APPN nodes. Nodes in the HPR portion of the network can distinguish between the APPN and HPR TGs and nodes.

HPR employs a route set-up protocol in order to obtain ANR and RTP connection information about the selected path. Any processing of packets required at the network connection and transport connection sublayers is the responsibility of the origin and destination endpoints of the packets. Endpoint processing includes flow control, segmentation and reassembly, and recovery of lost packets.

The procedure is as follows:

1 Change the VTAM options in VTAMLST(ATCSTRxx) so they look like:

```
CONNTYPE = LEN
CPCP = YES
DYNLU=YES
IOPURGE = 6Ø
INITDB = ALL
IPADDR = 192.DD.XXX.ZZ (* Address IP VIPA)
NODETYPE=NN
SORDER = APPN
SSEARCH=YES
TCPNAME = TCPIP
```

2 Use the new members in VTAMLST(ATCCONxx) to start the network. Add members COSAPPN (from SYS1.SAMPLIB) and IBMTGPS (from SYS1.SAMPLIB).

3 Change the OSA card:

- Activate node VTAM OSASF:

  ```
  V NET,ACT,ID=xxxxxx
  ```

- Start OSASF:

  ```
  S OSASF
  ```

- Get a table of the OSA card OSA (OAT) under TSO in command mode:

  ```
  ex 'sys1.sioasamp(ioacmd)'
  ```

- Choose option *Get OSA Address Table*:

  ```
  6
  ```

- When message *enter 0 to get help information for get...* appears, press *Enter*.

- When message *enter chpid ...*:

  ```
  b4
  ```

- When message *enter MVS dataset name...*:

  ```
  reseau.getoat.dujjmmaa
  ```

- When message *enter volser....* appears, press *Enter* and wait for the end of the EXEC.
- For example, to change the file created for addresses 40- 42 and 50-52 (units 600, 601, 602, 603 for each partition):

```
Ø2(Ø6Ø2) passthru Ø1 no Ø192.194.254.Ø4Ø S ALL
Ø192.194.49.Ø4Ø 255.255.255.Ø

Ø2(Ø6Ø2) passthru Ø1 no Ø192.194.254.Ø5Ø S ALL
Ø192.194.48.Ø5Ø 255.255.255.Ø
```

- Stop TCPIP:

```
P TCPIP ,.....
```

- Link the OSA card with the file created in the last step, ‘reseau.getoat.dujjmmaa’:

```
sys1.sioasamp(ioacmd)
```

- Choose option *Put OAT Address Table*:

```
8
```

- When message *enter 0 to get help information for get...* appears, press *Enter*.
- When you see message *enter chpid ...*:

```
b4
```

- When message *enter MVS dataset name...*:

```
reseau.getoat.dujjmmaa
```

- When you see message *enter volser....* press *Enter* and wait for the end of the EXEC.

Note: if there are two active partitions, the following commands must be entered on both systems:

- Take off-line the units in the OSA list:

```
V (6ØØ-6Ø3,6FE),OFFLINE
```

- Take the chpids off-line:

```
CF CHP(B4), OFFLINE
```

- Bring the chpids on-line:

```
CF CHP(B4), ONLINE
```

- Bring the units online:

```
V (600-603,6FE),ONLINE
```

- Restart TCP/IP and its associated tasks:

```
S TCPIP .....
```

The physical units 600/601 on the production partition are:

```
0192.DD.254.40
0192.DD.49.40 255.255.255.0
```

The physical units 600/601 on the test partition are:

```
0192.DD.254.50
0192.DD.48.50 255.255.255.0
```

The physical units 602/603 on the production partition are:

```
0192.DD.254.42
0192.DD.49.40 255.255.255.0
```

The physical units 602/603 on the test partition are:

```
0192.DD.254.52
0192.DD.48.50 255.255.255.0
```

4 IPL:

- Change TCP/IP:
  - for TCPIP.PARMLIB(DEV):

```
; OSA Port 1
DEVICE OSA1 LCS 602
LINK ETH1 ETHERNET 1 OSA1
; EE
DEVICE IUTSAMEH MPCPTP
LINK EELINK MPCPTP IUTSAMEH
; VIPA
DEVICE VIPA VIRTUAL 0
LINK LVIPA VIRTUAL 0 VIPA
```

  - for TCPIP.PARMLIB(START):

```
START OSA1
START IUTSAMEH
```

– for TCPIP.PARMLIB(HOME):

  - for the production partition:

```
HOME
 192.194.49.40 LVIPA
 192.194.254.40 ETH1
 192.194.49.41 EELINK
 ;PRIMARYINTERFACE ETH1
```

  - for the test partition:

```
HOME
 192.194.48.50 LVIPA
 192.194.254.50 ETH1
 192.194.48.51 EELINK
 ;PRIMARYINTERFACE ETH1
```

  - for TCPIP.PARMLIB(GEN):

```
ARPAGE 20
IPCONFIG NODATAGRAMFWD ; Dans Stay Cool on OS/390
; FIREWALL ; Activation firewall
 ARPTO 3600
 IGNOREREDIRECT ; Necessary if OROUTED is utilized
 VARSUBNETTING ; Necessary if OROUTED is utilized
 SOURCEVIPA ; VIPA
SACONFIG DISABLE
TCPCONFIG RESTRICTLOWPORTS
 TCPSENDBFRSIZE 32K
 TCPRCVBUFRSIZE 32K
UDPCONFIG RESTRICTLOWPORTS
 UDPQUEUELIMIT
TRANSLATE
ITRACE OFF
```

  - for OSA Menu (table OAT) (under TSO: 'SYS1.SIOASAMP(IOACMD)':

```
02(0602) passthru 01 no 0192.194.254.040 SIU ALL
 0192.194.49.040 255.255.255.0
------------------------------------------------------------
02(0602) passthru 01 no 0192.194.254.050 SIU ALL
 0192.194.48.050 255.255.255.0
```

Before you PUTOAT, you should stop TCP/IP and all the TCP/IP subtasks (nfs, ipprintwy, tcpdns, tcpsntp, and tcprouted), and after PUTOAT you have to put the units (600-603,6fe) and the CHPID B4 offline. After putting

them online, restart TCP/IP.

- for TCPIP.PARMLIB(ROUT):

```
; orouted Routing Information
BSDROUTINGPARMS TRUE
; Interface Max MTU Metric Subnet Mask
 ETH1 1500 0 255.255.255.0 0
 LVIPA defaultsize 0 255.255.255.0 0
 EELINK defaultsize 0 255.255.255.0 0
ENDBSDROUTINGPARMS
```

- for TCPIP.PARMLIB(PROFILE):

```
 ;
INCLUDE TCPIP.PARMLIB(ESA1GEN) ; Options generales
INCLUDE TCPIP.PARMLIB(ESA1TELN) ; Telnet
INCLUDE TCPIP.PARMLIB(ESA1AUTO) ; Autolog
INCLUDE TCPIP.PARMLIB(ESA1PORT) ; Ports
INCLUDE TCPIP.PARMLIB(ESA1DEV) ; Devices & Links
INCLUDE TCPIP.PARMLIB(ESA1HOME) ; Adresses
;INCLUDE TCPIP.PARMLIB(ESA1GWAY) ; Gateway / BSDRouting
INCLUDE TCPIP.PARMLIB(ESA1ROUT) ; Gateway / BSDRouting
INCLUDE TCPIP.PARMLIB(ESA1STRT) ; Start
```

- Change Unix files:
  - for file */etc/gateways*:

```
options interface.scan.interval 90
options interface.poll.interval 15
options interface EELINK 192.194.49.41 ripoff
net 0.0.0.0 gateway 192.194.254.7 metric 1 passive
```

  - for file */etc/routed.profile*:

```
; Parametres: cf IP configuration
RIP_SUPPLY_CONTROL: RIP2
RIP_RECEIVE_CONTROL: RIP2
RIP2_AUTHENTICATION_KEY:
```

- Accept changes in TCP/IP without stopping it:

```
V TCPIP,, O, TCPIP.PARMLIB(xxxGEN)
V TCPIP,,O, TCPIP.PARMLIB(xxxDEV)
V TCPIP,,O, TCPIP.PARMLIB(xxxHOME)
V TCPIP,,O, START , IUTSAMEH
V TCPIP,,O, TCPIP.PARMLIB(xxxROUT)
S TCPROUTED
```

- Changes in VTAM:

– for PROCLIB VTAM (net):

```
//* ADDITIONAL DATASETS FOR NN
//*
//DSDB1 DD DISP=SHR,DSN=ESA1.DSDB1
//DSDB2 DD DISP=SHR,DSN=ESA1.DSDB2
//DSDBCTRL DD DISP=SHR,DSN=ESA1.DSDBCTRL
//TRSDB DD DISP=SHR,DSN=ESA1.TRSDB
```

(NB: for DSDB1, DSDB2, and TRSDB (LREC =19200, and DSDBCTRL (20))

– changes to OPTIONS in VTAM (VTAMLST(ATCSTRxx)):

```
CDSERVR = YES (for the 2 unique site servers)
CONNTYPE = LEN
CPCP = YES
DYNLU=YES
IOPURGE = 60
INITDB = ALL
IPADDR = 192.dd.49.40 (VIPA Production)
                              or 192.dd.48.50 (VIPA Test)
NODETYPE=NN
SORDER = APPN
SSEARCH=YES
TCPNAME = TCPIP
```

– add a model for TG (VTAMLST(HPRTRL)):

```
AHPRTRL VBUILD TYPE=MODEL
ISTP* PU TRLE=*,TGP=XCF,CONNTYPE=APPN
```

– add a major node XCA (VTAMLST(HPRXCA)):

```
AHPRXCA VBUILD TYPE=XCA
PHPRXCA PORT MEDIUM=HPRIP
GHPRXCA GROUP DIAL=YES,CALL=INOUT,DYNPU=YES,AUTOGEN=20
```

– create a swnet node (VTAMLST(HPRyzz), eg:

o for the Toulouse production system use this in VTAMLST:

```
XHPR131  VBUILD TYPE=SWNET,MAXGRP=192,MAXNO=192
PHPR131  PU    ISTATUS=ACTIVE,CPNAME=TOULOUSE,NETID=CP,                *
               TGN=6,TGP=ETHERNET,CONNTYPE=APPN
         PATH  IPADDR=192.31.49.40,GRPNM=GHPRXCA,PID=1,GID=1,USE=YES,  *
               CALL=INOUT
```

o for the Toulouse test system use this in

VTAMLST:

```
XHPR231  VBUILD TYPE=SWNET,MAXGRP=192,MAXNO=192
PHPR231  PU    ISTATUS=ACTIVE,CPNAME=*****,NETID=CP,                    *
               TGN=6,TGP=ETHERNET,CONNTYPE=APPN
         PATH  IPADDR=192.31.48.5Ø,GRPNM=GHPRXCA,PID=1,GID=1,USE=YES,   *
               CALL=INOUT
```

- for the Bordeaux production system use this in VTAMLST:

```
XHPR133  VBUILD TYPE=SWNET,MAXGRP=192,MAXNO=192
PHPR133  PU    ISTATUS=ACTIVE,CPNAME=BORDEAUX,NETID=CP,                 *
               TGN=6,TGP=ETHERNET,CONNTYPE=APPN
         PATH  IPADDR=192.33.49.4Ø,GRPNM=GHPRXCA,PID=1,GID=1,USE=YES,   *
               CALL=INOUT
```

- for the Bordeaux test system use this in VTAMLST:

```
XHPR233  VBUILD TYPE=SWNET,MAXGRP=192,MAXNO=192
PHPR233  PU    ISTATUS=ACTIVE,CPNAME=AN2M33,NETID=CP,                   *
               TGN=6,TGP=ETHERNET,CONNTYPE=APPN
         PATH  IPADDR=192.33.48.5Ø,GRPNM=GHPRXCA,PID=1,GID=1,USE=YES,   *
               CALL=INOUT
```

etc.

- take into account the new members when starting the two systems (VTAMLST(ATCCONxx)):

```
COSAPPN (Recovery file on SYS1.SAMPLIB)
IBMTGPS (Recovery file on SYS1.SAMPLIB)
HPRTRL
HPRXCA
HPR194 *
HPR294 *
HPR131 *
HPR231 *
HPR133 * ....
```

NB: do not code the node of the system of the site you are working at.

- DISPLAY under NetView:
  - node TRL:

```
DISPLAY NET,ID=HPRTRL,SCOPE=ALL
```

```
 NAME = TRLCP1 , TYPE = MODEL MAJOR NODE
 VTAMTOPO = REPORT , NODE REPORTED - YES
 MODELS:
 ISTP* TYPE = PU_T2 , RESET
 END
```

– node XCA:

```
DISPLAY NET,ID=HPRXCA,SCOPE=ALL
NAME = AHPRXCA , TYPE = XCA MAJOR NODE
STATUS= ACTIV , DESIRED STATE= ACTIV
 MEDIUM = HPRIP
 TCP/IP JOB NAME = TCPIP
 LOCAL IP ADDRESS 192.194.49.40
I/O TRACE = OFF, BUFFER TRACE = OFF
 VTAMTOPO = REPORT , NODE REPORTED - YES
LINES:
00000000 ACTIV
00000001 ACTIV
00000002 ACTIV
00000003 ACTIV
00000004 ACTIV
00000005 ACTIV
00000006 ACTIV
00000007 ACTIV
00000008 ACTIV
00000009 ACTIV
END
```

– node HPR:

```
 DISPLAY NET,ID=HPR294,SCOPE=ALL
 NAME = XHPR294 , TYPE = SW SNA MAJ NODE
 STATUS= ACTIV , DESIRED STATE= ACTIV
 VTAMTOPO = REPORT , NODE REPORTED - YES
 NETWORK RESOURCES:
 PHPR294 TYPE = PU_T2 , ACTIV
 STATE TRACE = OFF
 END
```

– node RTP (dynamic node):

```
DISPLAY NET,ID=ISTRTPMN,SCOPE=ALL
 IST097I DISPLAY ACCEPTED
 IST075I NAME = ISTRTPMN , TYPE = RTP MAJOR NODE
 IST486I STATUS= ACTIV , DESIRED STATE= ACTIV
 IST1486I RTP NAME STATE DESTINATION CP MNPS TYPE
 IST1487I CNR0000A CONNECTED CP.D94ESA2 NO LULU
 IST1487I CNR00009 CONNECTED CP.D94ESA2 NO LULU
 IST1487I CNR00005 CONNECTED CP.D94ESA2 NO RSTP
 IST1487I CNR00004 CONNECTED CP.D94ESA2 NO LULU
```

```
 IST1487I CNR00003 CONNECTED CP.D94ESA2 NO RSTP
 IST1487I CNR00002 CONNECTED CP.D94ESA2 NO CPCP
 IST1487I CNR00001 CONNECTED CP.D94ESA2 NO CPCP
 IST314I END
```

– node LCL:

```
 DISPLAY NET,ID=ISTLSXCF,SCOPE=ALL
 IST097I DISPLAY ACCEPTED
 IST075I NAME = ISTLSXCF , TYPE = LCL SNA MAJ NODE
 IST486I STATUS= ACTIV , DESIRED STATE= ACTIV
 IST084I NETWORK RESOURCES:
IST1316I PU NAME = ISTPA1A2 STATUS = ACTIV--X- TRLE = ISTTA1A2
 IST1500I STATE TRACE = OFF
 IST314I END
```

– display topology:

```
DELAND D NET,TOPO,ID=D94ESA2,LIST=ALL
 ' ACP1N DELAND
 IST350I DISPLAY TYPE = TOPOLOGY
 IST1295I CP NAME NODETYPE ROUTERES CONGESTION CP-CP WEIGHT
 IST1296I CP.D94ESA2 NN 128 NONE YES *NA*
 IST1579I ----------------------------------------
 IST1297I ICN/MDH CDSERVR RSN HPR
 IST1298I YES YES 2304892 RTP
 IST1579I ----------------------------------------
 IST1223I BN NATIVE TIME LEFT LOCATE SIZE
 IST1224I NO YES 11 16K
 IST1299I TRANSMISSION GROUPS ORIGINATING AT CP CP.D94ESA2
 IST1357I CPCP
 IST1300I DESTINATION CP TGN STATUS TGTYPE VALUE WEIGHT
 IST1301I CP.CRETEIL 21 OPER INTERM YES *NA*
 IST1301I CP.CRETEIL 6 OPER INTERM YES *NA*
 IST314I END
```

– display a session (CLIST lusid):

```
LUSID DSYI0012
CNMKWIND OUTPUT FROM D NET,SESSIONS,SID=DA5B99370D9075B5 LINE 0 OF 27
 *-------------------------- Top of Data ----------------------------*
 IST097I DISPLAY ACCEPTED
 IST350I DISPLAY TYPE = SESSIONS
 IST879I PLU/OLU REAL = CP.TSO20002 ALIAS = ***NA***
 IST879I SLU/DLU REAL = CP.DSYI0012 ALIAS = ***NA***
 IST880I SETUP STATUS = ACTIV
 IST875I ADJSSCP TOWARDS PLU = ISTAPNCP
 IST875I ALSNAME TOWARDS PLU = CNR0000B
 IST933I LOGMODE=SNX32702, COS=*BLANK*
 IST875I APPNCOS TOWARDS PLU = £CONNECT
 IST1635I PLU HSCB TYPE: BSB LOCATED AT ADDRESS X'0723E540'
```

```
IST1635I SLU HSCB TYPE: FMCB LOCATED AT ADDRESS X'Ø691CCB8'
IST1636I PACING STAGE(S) AND VALUES:
IST1644I PLU--STAGE 1----!----STAGE 2--SLU
IST1638I STAGE1: PRIMARY TO SECONDARY DIRECTION - ADAPTIVE
IST164ØI SECONDARY RECEIVE = 7
IST1641I STAGE1: SECONDARY TO PRIMARY DIRECTION - ADAPTIVE
IST1642I SECONDARY SEND: CURRENT = 1 NEXT = 1
IST1638I STAGE2: PRIMARY TO SECONDARY DIRECTION - ADAPTIVE
IST1639I PRIMARY SEND: CURRENT = 6 NEXT = 7
IST164ØI SECONDARY RECEIVE = 7
IST1641I STAGE2: SECONDARY TO PRIMARY DIRECTION - ADAPTIVE
IST1642I SECONDARY SEND: CURRENT = Ø NEXT = 1
IST1643I PRIMARY RECEIVE = 7
IST1713I RTP RSCV IN THE DIRECTION OF THE PLU
IST146ØI TGN CPNAME TG TYPE HPR
IST1461I 21 CP.D94ESA2 APPN RTP
IST314I END
```

Claude Dunand
Systems Programmer (France)

SOAs and composite applications

IBM, particularly when announcing CICS Transaction Server for z/OS V3.1, Attachmate in the context of its new Synapta branding, and BEA in general – just to name a few of the major players – is now vociferously promoting the notion of Service-Oriented Architecture-based (SOA) solutions. Consequently host integration, a key technology within the overall Web-to-host solution spectrum, dating back to at least 1998, is now being subsumed by a new emphasis on SOA-based composite applications – with the implication that host integration is now *passé* and composite applications are the way of the future.

Before one gets too misled by what is in reality a whole bunch of new terms to describe previously-known methodologies, it is best to take stock of what SOA is all about and tie it into what went before it. For a start, one should appreciate that host integration is also an SOA-based solution, and that just

because the term 'SOA' is new does not mean that SOA concepts were not previously available. It is also important to note, at the very outset, that, despite what marketers would like us to believe, XML Web services are not the only way to realize SOA-based solutions.

BEA, for example, defines SOA as follows: 'SOA is a standards-based organizational and design methodology that more closely aligns IT with business processes using a collection of shared services on a network. Using standard interfaces that help mask the underlying technical complexity of the IT environment, SOA enables greater re-use of IT assets.' Essentially, all that this is saying is that SOA is about reusing existing IT assets (ie application functionality to be precise) using standards-based interfaces. Well, this was also what host integration, which is all about reusing existing host application business logic via technologies such as XML, EJBs, .NET assemblies, and Web services, is also all about.

However, to be fair, host integration, as denoted by its name, focused on capturing and reusing functionality from applications running on traditional 'hosts' – ie mainframes, iSeries, and Unix systems. Hence the connotation is that host integration deals with 'legacy'. In contrast, 'composite applications', the term now preferred to describe the end-result possible with SOA, has no explicit associations with legacy applications – though reusing functionality from legacy applications is definitely not ruled out, and in many cases will be a requirement. Thus composite applications will be something that the TCP/SNA community will have to come to terms with rather quickly since this will be the terminology that management would prefer to hear about.

COMPOSITE APPLICATIONS IN THE TCP/SNA WORLD

The best definition of a composite application, *vis-à-vis* the goals of SOA, is to say that it is a new application that gainfully makes use of functionality from other applications. Given the software development methodologies already in place, it is

not necessary to add any caveats to limit the scope of this definition. Thus, note that there is no need to limit composite applications to ones that insist on using only functionality that is available as XML Web services. This is an unnecessary restriction, particularly if a composite application for a particular enterprise is just going to reuse application functionality from applications currently owned by that enterprise. In many cases it will be possible to create *bona fide*, SOA-based composite applications using other object representation mechanisms such as EJBs or .NET assemblies.

The whole notion of SOA is contingent upon the run-time invocation and execution of the application functionality being reused. This is what is meant by treating existing applications as services that deliver reusable functionality for use by newer applications. Host integration, *à la* IBM's HATS V5, NetManage's OnWeb, Attachmate's Synapta Service Builder, or Seagull's Transidiom, is also totally based on this service-centric notion. With SOA one does not borrow functionality by 'cutting-and-pasting' code segments from other applications into the new one. There is also no attempt made to convert existing application code (or even application subroutines) from its original 'legacy' programming languages (eg COBOL) to more strategic, platform-independent variants (eg Java).

Experience, particularly the Y2K conversion, has proved that trying to reuse application functionality (or recreate business processes) at the source-code level is often not possible or practical – primarily because of uncertainty as to reliability, over the decades, of source-code maintenance and control. In addition, if functionality in the form of source-code is desired from third-party applications, as is invariably the case today, one can also be faced with intellectual property, copyright, and royalty issues. Hence the growing popularity of reusing application functionality in the form of run-time services – which is what SOA, in the end, is all about.

With the SOA approach, application functionality reuse is realized through the standard function-call (or procedure/ subroutine-call) paradigm. The pivotal difference, however, is

that the software functions being invoked, in real-time, will not be a part of the calling application. In most cases the functions being invoked will not even be running on the same platform or even the same data centre. That's why XML Web services are often explained as being a platform- and language-independent form of Remote Procedure Call (RPCs). Thus, with SOA, a composite application written in C# per .NET criteria and running on a Windows 2003 server could be making function calls to CICS applications running on a mainframe as well as to an RPG application running on an AS/400.

DECOUPLING WEB SERVICES FROM SOA

SOA solutions do not always have to be based entirely on Web services despite IBM (and others) being incapable of talking of the one without unfailingly mentioning the other. Web services just happen to be the newest (and most talked about) enabling technology for SOA. Just because .NET assemblies, EJBs, and for that matter CORBA, do not have the word 'services' in their names does not in any way mean that they too do not fully support the creation of SOA-based solutions. This distinction is important because there can be many cases where one, particularly in the TCP/SNA world, may (and should) use EJBs or .NET assemblies to create a Web service.

There is nothing mystical about Web services. Web services, from their inception, have always been about self-contained, modular chunks of software with standards-based (ie XML-defined) input and output. SOAP provides a way to remotely invoke these chunks of software, where WSDL is a text file that clearly defines (using XML) what that Web service is intended to do, by specifying what input it expects and the output it will generate. Note that all of the standards-related aspects of Web services pertain to the I/O definitions (via XML, WSDL), the preferred means for Web service invocation (ie SOAP), and the optional service advertising mechanism (ie UDDI).

There are no standards or even conventions as to how the actual body of a Web service (ie the business logic) should be implemented. But this is intentional. It is this flexibility *vis-à-vis* the actual implementation that makes Web services so attractive and powerful. You can create a Web service using any programming methodology – even Assembler, COBOL, or FORTRAN. All that is mandatory is that its I/O requirements are XML-based, and that it can be invoked remotely using a SOAP-like mechanism. In practice, with the current emphasis on object-oriented software development, most Web services are created using .NET assemblies or EJBs.

If a Web service consists of a .NET assembly or EJB at its core, one has to question what advantages one can derive from using the Web service as opposed to directly accessing the core object. The answer to this is very simple and somewhat anticlimactic! The Web service provides an XML-defined I/O mechanism that can be invoked using SOAP whereas the native object scheme will use a less rigorously defined I/O scheme. The issue thus boils down to the advantages of having XML-defined I/O. Obviously, having XML-defined I/O, to avoid any and all ambiguity, is extremely attractive and desirable when one is trying to source functionality from a third party – ie a service-provider.

This, in essence, should now put the potential role of Web services *vis-à-vis* composite applications into perspective. Going back to the four 'topology' categories listed above, it should now be clear that Web services are most relevant if one has to source functionality, over the Internet, from a previously foreign service provider. If, on the other hand, your composite applications are all going to be based on functionality culled from applications already available on the corporate intranet, or on an extranet with a preferred partner, you should retain the option of being able to use EJBs or .NET assemblies in addition to Web services. Forgoing that option is an unnecessary and potentially costly constraint.

BOTTOM LINE

SOA and composite applications have now become the latest terms that professionals in the TCP/SNA community have to contend with. Though newly coined, these terms, however, deal with concepts and technologies familiar to those who have been involved with Web-to-host. Host integration, a core Web-to-host methodology, is very much an SOA-oriented scheme, particularly so in that all major host integration solutions support Web services – though it is indeed possible to have SOA solutions that are not entirely based on Web services. Whereas host integration focused on reusing legacy 'assets', composite applications do not pigeonhole themselves just to legacy applications. In essence, composite applications are a super-set of what is possible with host integration. This, however, means that it is indeed possible to have a composite application that reuses only application functionality from legacy TCP/SNA applications.

Anura Gurugé
Strategic Consultant (USA)

If you would like to contribute an article to *TCP/SNA Update*, send it to the editor, Trevor Eddolls, at TrevorE@xephon.com.

Inside Products has released Inside the Stack (ItS), its interactive, graphical, Web-based management solution, which is designed to help diagnose and resolve problems in a z/OS TCP/IP network.

The product offers Performance Problem Determination Assistance (PPDA), a knowledgebase approach to enable systems programmers to benefit from the real-world experience of their peers when diagnosing and resolving problems.

Other key features of ItS include, a performance dashboard, system usage statistics, a listener performance profile, a real-time monitor, a history monitor, an alert monitor, and automatic assessment of trouble spots.

For further information contact:
Inside Products, 30 Los Helechos, Carmel Valley, CA 93924, USA.
Tel: (831) 659 8360.
URL: www.inside-products.com.

* * *

NetManage has announced the latest version of RUMBA, its solution for PC client access to applications and databases on mainframes, and other systems.

RUMBA is part of NetManage's Host Services Platform (HSP), allowing RUMBA administrators to enable optional monitoring of user host access activity and provides reports so they can observe host usage patterns or determine cost allocations.

For further information contact:
NetManage, 20883 Stevens Creek Blvd, Cupertino, CA 95014, USA.
Tel: (408) 973 7171.
URL: www.netmanage.com/pressroom/viewpress.asp?id=365.

* * *

TPS Systems has announced TPS/JES Services, allowing companies to consolidate older communications infrastructure to take advantage of newer multi-protocol communication technologies. Historically JES subsystem relied on direct mainframe connectivity or SNA networks. Companies that rely on z/OS JES-based applications would like to eliminate the need for existing SDLC WAN networks. RJE services is one where a transfer from SNA or BSC to TCP/IP is needed. TPS/JES Services provides the migration from SNA to TCP/IP.

TPS/JES Services is comprised of two components: TPS/JES Services Server and TPS/RJS (Remote JES Services) Client. The Server operates as a z/OS component executing in the MVS background to make JES2 and JES3 available to a TCP/IP based client, TPS/RJS, allowing a multitude of simultaneous Client connections while maintaining only a single instance of the Server.

For further information contact:
TPS Systems, 14100 San Pedro Avenue, Suite 600, San Antonio, TX 78232-4399, USA.
Tel: (210) 496 1984.
URL: www.tps.com.

* * *