58 TCP/SNA update

June 2005

In this issue

A tc 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.

Object creation options for composite applications

Service-Oriented Architecture-based (SOA) solutions are now, incontrovertibly, the topic *du jour* in the TCP/SNA world – with IBM in particular now couching everything to do with application integration in terms of being an SOA opportunity. Host integration, the technology for creating objects out of data and business logic from mission-critical host applications (in particular mainframe and AS/400 SNA applications) for reuse in new software development initiatives, has now become *passé*. Though host integration is a *bona fide* SOA methodology, IBM and others no longer want to have technology that is limited just to 'legacy' host applications. Instead, with SOA, the goal is to extend the object capture and reuse principles advocated by host integration to apply to any and all applications – rather than just to 'legacy' applications. This in general is good news to the software developers in the TCP/SNA community, since many of them, thanks to their exposure to host integration technology (eg IBM's Host Publisher), now have a distinct head-start when it comes to SOA.

SOA, though presented as an architecture, is, in reality, more a manifesto than a technical specification. The real 'deliverables' in the context of SOA are the so-called SOA-based composite applications – where a composite application is one that relies on functionality from other applications. When it comes to a composite application, the 'other applications' from which it borrows functionality can in theory be a mix of contemporary applications (eg SAP R/3, Siebel, PeopleSoft, etc) and legacy 'green screen' applications. Irrespective of the nature and vintage of these 'other applications' (ie the source applications), the functionality to be reused by the new composite applications is expected to be extractable and available in the form of remote invocable software routines. Hence this sometimes confusing notion of reusing software functionality in the form of services; services in this context being nothing other than remotely invoked software subroutines or procedures. Thus, in essence, SOA-based composite applications can be thought of as being built around well-understood, conventional Remote Procedure Call (RPC) disciplines.

XML Web services, the latest iteration of standard-based RPC technology, is thus invariably positioned as the prerequisite basis for composite applications. In reality, this emphasis on Web services is nothing but posturing and propaganda by companies (including IBM) that are desperate to justify their backing and investment in XML technology. Software developers in the TCP/SNA world need always to remember that SOA is essentially just the latest ‘politically-acceptable’ means of referring to RPC. Thus, any and all RPC mechanisms, including Unix RPCs, are genuinely valid options for creating SOA-based composite applications. Therefore, if you already have experience and objects in CORBA, Java Beans, EJBs, COM, or .NET, do not in any way feel obliged to restart and get everything in the form of Web services. The beauty of true object technology is the ability to easily mix-and-match. It is important not to lose sight of that.

THE OPTIONS FOR OBJECT CREATION

Unless you already have all the required software functionality in some object form, the objects to be invoked by a new composite application have to be created (and, obviously, tested) before one can deploy that new application for mission-critical production use. The rationale and justification for developing composite applications hinges on the profitable reuse of existing software functionality. If that is not the case, one is back to dealing with new application development as opposed to creating composite applications. So the issue here boils down to how one goes about isolating and capturing ‘business logic’ from within existing applications – so that this business logic can later be reinvoked (in isolation) from a new composite application.

In reality, there are still only three ways to capture such business logic from existing applications and have them invocable in the form of objects:

1 Programmatic schemes, typically involving the use of application-specific adapters or connectors (such as those available from IBM or iWay Software).

2 Screen-scraping (or 3270/5250/VT datastream interception/decoding), either because there are no appropriate adapters for the applications in question (the object developer has prior experience with this approach – given its ubiquitous use in most Web-to-host solutions), or because the object developer intends to exploit the capability that is possible with this approach to easily 'combine, filter, and skip' I/O fields .

3 Extracting the relevant source code segments if one is confident that the application source code has been diligently maintained and is up to date.

Screen-scraping, historically, has been a technique widely used by the TCP/SNA community. The current trend, however (particularly when dealing with CICS and IMS applications), is to use adapters whenever possible. Adapters, in general, are now available in both Java and .NET variants, with many of the Java adapters now conforming to the J2EE Connector Architecture (JCA). One, however, has to be extremely diligent when evaluating and selecting adapters – especially for mainframe-resident applications. As TCP/SNA developers will readily appreciate, the design and the exact APIs accessed by an adapter can have a profound impact on the performance and scalability of an adapter in terms of the number of transactions it can extract from a mainframe application over a given period of time. Suffice it to say, some adapters can be a couple of orders of magnitude faster than others! Thus, if performance is a concern, as it invariably is with TCP/SNA applications, make sure that you check the exact throughput of an adapter before you commit to using it to realize your new mission-critical application.

Objects (including Web services) that are to be used in composite applications do not have to be, and in most cases they will not be, self-contained units of software functionality. While it is indeed possible to have totally self-contained objects, such objects invariably fall into the 'utility' function category (eg create a new display window), as opposed to those that perform a complete business process. This is why, especially in the TCP/SNA world, transactions will continue to be the preferred 'unit of work' when it comes to object creation.

TRANSACTIONS AS THE BASIS FOR SOA

Software functionality being sought for reuse from an existing application may never have been developed in a form conducive to (or even accessible for) reuse in the form of an object. That is an inescapable fact of life *vis-à-vis* SOA and composite applications. In many cases the required functionality may not even be implemented by a contiguous 'block' of software within the source application. Instead, there could be significant branching and linking within the parent application in order to deliver that functionality. This is invariably the case with older 'SNA' applications, with the situation often further exacerbated with on-the-fly fixes applied to these applications in the form of object-code patches that involve branching off to a new patch routine.

Hence the emphasis on transactions. Thus software functionality, for example that representing a particular business process, is invariably identified and extracted in terms of a clearly defined and demarcated transaction. This is prudent and pragmatic. End users and line-of-business management, as well as programmers, can relate to specific transactions performed by an application. Thus there is no ambiguity when attempting to describe business processes (and the software functionality that performs them) in terms of transactions.

A transaction performs a predefined (and describable) process; it has specific input/output characteristics, and one can typically determine whether it completed its designated task successfully (or otherwise). Consequently, transactions (eg extract customer record, update quantity in stock for this item) are the basic, as well as the smallest, units-of-work when it comes to SOA-based composite applications. Hence the granularity of the functionality that can be extracted from a source application will be governed by the nature of the transactions it supports. Adapter-based programmatic access or screen-scraping can be used to capture a transaction and represent it in the form of an object or Web service.

Thus, contrary to a common misconception, one is not always forced to rely on screen-scraping in order to create objects using a transaction-oriented paradigm. It is, however, crucial to note that an object or Web service representing a transaction also has to contain all the necessary application access, user authentication, and transaction location (ie navigation) data – in addition to the I/O fields used by the transaction. Thus, when dealing with host applications, many TCP/SNA developers may still prefer to deal with a proven screen-scraping scheme – particularly if they wish to capture 'complex transactions' involving I/O fields from multiple screens – even spanning multiple disparate applications.

It is, however, important never to lose sight of the fact that SOA-based solutions, by definition, rely on RPC – in other words, a run-time execution model. This means that the source applications need to be up and running when a new composite application invokes an object that sets out to obtain functionality from one or more or those source applications. Therefore, there always has to be a tight coupling between the object sourcing phase and the application execution phase. This explains the recent talk about 'composite application servers'. A composite application server is an extension and refinement to traditional application servers *à la* IBM's WebSphere or BEA's WebLogic. Composite application servers cater for inescapable coupling between object creation and application execution, and provide all the necessary automation to greatly simplify and expedite object creation, application synthesis, and the subsequent application deployment.

BOTTOM LINE

There is nothing overtly new or magical about composite applications. They are applications that make use of functionality from other existing applications using the well understood RPC paradigm – though the trend today is to present the RPC mechanism in the form of XML Web services. Transactions tend to be the optimum unit of work when it comes to capturing existing application functionality for reuse within contemporary composite applications. Programmatic access or screen-scraping can be used to capture transactions and create reusable objects or Web services. Such objects and Web services, however, need to contain all the necessary application access, user authorization, and transaction location 'linkages' in addition to the logic required to invoke the transaction with the appropriate I/O fields. This leads to the notion of composite application servers. At a minimum, a good composite application server will provide comprehensive and proven functionality for automated application connectivity, stringent access control and user authentication, invoking 'complex transactions' involving I/O fields from multiple screens, transaction coordination, process orchestration, process automation (eg scripting), and supporting multiple object technologies.

Anura Gurugé
Strategic Consultant (USA)

Case study in TCP application performance using SSL

We recently assisted a large public university with a problem in using a mainframe TCP application using Secure Sockets Layer (SSL). At times, when students registered for a dormitory room using their CICS Web-enabled application, the CICS region became ‘hung’ and had to be restarted. Also, students were unable to use the application and many calls were received by the Help Desk. Clearly, this was not a situation that could be allowed to continue.

We will use this case to illustrate some basic issues in troubleshooting TCP/IP problems:

1 Understand the exact nature of the problem.

2 Understand the architecture of the problem.

3 Be able to recreate the problem.

The path to a solution is not linear – much trial and error goes into the process and I will describe the trials and errors that we made in the hope that this will assist you in your own troubleshooting efforts.

UNDERSTAND THE EXACT NATURE OF THE PROBLEM

This application allowed students to select accommodation on the campus. They could choose residence hall, room-mate, meal plan, and other related matters. The application was available for one week out of the year. The university tried to stage the use of the application by having honours students and Seniors sign up the first day, Juniors the following day, and so forth. The application worked well until the last day, when students with poor grades, freshmen, and all others were allowed in. Then, chaos erupted.

When I was first called in to investigate this problem, I was told that on this last day the CICS region used so many resources that it ‘hung’ and had to be restarted. At first glance, it seemed that this issue was with the high number of resources being used. So, my first approach was to try to understand how much time the application was taking at the host and in the network so that we could approach the application people to discuss tuning efforts. I expected to see something like a transaction taking 1 second average time with a light load and increasing to 10 seconds when the load increased.

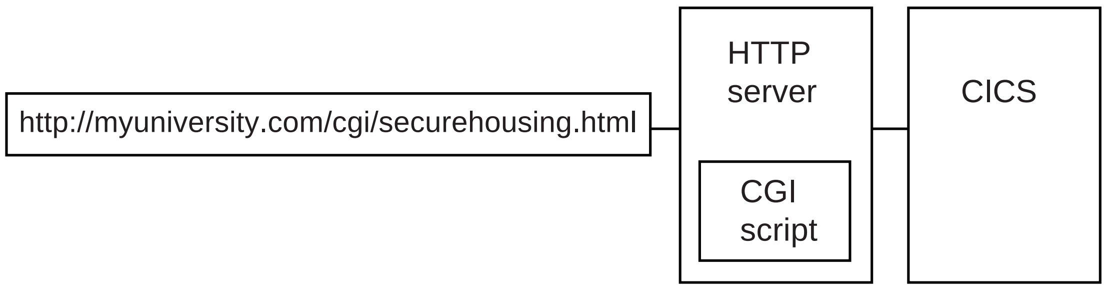

Figure 1: Housing application architecture

UNDERSTAND THE ARCHITECTURE OF THE PROBLEM: INITIAL APROACH

The next step was to understand the architecture of this application. Students used this application via a Web page using secure sockets. The secure sockets, or SSL, was required because social security numbers, financial data, and other private information might be shown. So the student would log on to a Web page such as http://myuniversity.com/cgi/securehousing.html.

This Web page was actually a CGI script running under the HTTP Web server on the mainframe. The port used was port 443 for secure sockets. This CGI script then initiated a connection to the CICS application to get the needed data and pass it back – see Figure 1.

BE ABLE TO RECREATE THE PROBLEM: INITIAL APPROACH

Our first step in this stage was to see exactly how many concurrent users we could get on the application before a serious event such as a CICS lock-up occurred. We used a load creation program that could log on to the application and use various screens. This way, we could vary how many users we had on and see the impact. We also logged on to the application ourselves during this time using test student IDs so that we could see first-hand what a potential user might experience.

Test 1

We then proceeded to our first test. Three of us tried to log on and use the application while running the load simulator. We watched the load on the HTTP server, TCP/IP, and the mainframe CPU time. For the HTTP server, we watched how many SSL threads were open. On the TCP/IP level, we ran a TCP packet trace that we could later analyse for any problems. For mainframe time, we collected SMF record type 30s to tell us address space CPU time, paging, etc.

On our first test, we reached about 20 simultaneous users before we could do no more. We three 'real' users experienced serious delays in application usage. In fact, we could not log in to the application at all at this point. This was our first benchmark. It seemed that indeed the reports of the CICS region becoming overloaded were true.

We could see that for the HTTP server, out of the 40 SSL threads available, many remained open. From the SMF type 30, we could see the TCB and SRB usage for CICS growing to about four times the normal level. The normal level was about 300 CPU milliseconds per 15-minute interval; during the test this increased to about 1,200 milliseconds per 15-minute interval. Still, this did not seem so excessive that it would cause a problem such as the CICS lock up and failure to log in.

From the TCP packet trace, however, we started seeing a very interesting phenomenon that led us away from our initial hypothesis. We saw many TCP resets from ephemeral ports to the CICS region. The resets appeared to come from the same address as our TCP stack! The TCP reset packets are shown below:

```
IP SRC = 123.45.123.6               IP DST = 123.45.123.6
   HDLEN = 5     TOS = XØØ  TOTLEN = 4Ø     ID = 19321  FLAGS = (none)
   FRAGOFF = Ø       TTL = 64    PROTOCOL = TCP     CHECKSUM = X8Ø9B   FFFF
  TCP SRC PORT = 1273                 TCP DST PORT = 3333
   SEQ NUM = 3985579497  ACK NUM = 3985622536  FLAGS = RST
   HDLEN = 5    WINDOW = Ø      CHECKSUM = X3157  FFFF  URGENT PTR = Ø
   HEADER LENGTH = XØØ28
```

This was quite interesting – who was sending all those resets and why?

UNDERSTAND THE ARCHITECTURE OF THE PROBLEM: REFINE UNDERSTANDING

We then sought to refine our understanding of the architecture. We thought that possibly these resets were responsible for the high CPU time and inability to log in. Who could be sending these resets? Our first ideas as to the culprits were the HTTP server or the CGI script. You can see from the output above that the TCP source port is an ephemeral port 1273 going to our CICS region 3333.

We never saw any session initiate (SYN-ACK) sequences from this (or any other) ephemeral port. Nearly 100 packets such as these were sent in a 2-minute period from various ephemeral ports. These traces were taken on an OS/390 system, which does not tell us the address space ID that the packet was sent on. On z/OS systems, the address space ID information is available for TCP packets and might have made our job somewhat easier.

HTTP configuration investigation

We looked at the HTTP configuration and found a number of timeout and performance directives.

The HTTPD configuration performance directives are shown below:

```
# =============================================================== #
#        Performance directives.
# =============================================================== #
#        MaxActiveThreads directive:
#
#        Defines the maximum number of threads in system thread pool.
```

```
#
#        Default:  4Ø
#        Syntax:   MaxActiveThreads  <num>
MaxActiveThreads   4Ø

#        MaxPersistRequest directive:
#
#      Maximum number of request to receive on a persistent connection.
#
#        Default:  5
#        Syntax:   MaxPersistRequest <num>
# changed by xxx from 4 to 1 to turn off persistent connections
MaxPersistRequest 1
```

The HTTPD configuration timeout directives are shown below:

```
# ================================================================== #
#        Timeout directives
# ================================================================== #
#
#        Use these directives to:
#           * limit the time to wait for the client to send a request
#       after connecting to the server before cancelling the connection.
#           * limit the time to allow for sending output to the client.
#           * limit the time to allow for server scripts to finish.
#       (If the program does not finish within allotted time, the server
#        will send a TERM signal and then a KILL signal 5 seconds later
#              to stop the program.)
#           * limit the time to wait for the client to send a request
#             after establishing a persistent connection to the server
#             before cancelling the connection.
#
#        Default:  InputTimeout    3Ø secs
#        Default:  OutputTimeout   2 minutes
#        Default:  ScriptTimeout   2 minutes
#        Default:  PersistTimeout  5 secs
#        Syntax:   <directive> <time-spec>
# changed by xx to "maybe" boost speed for housing application
# changed by xxx inputTimeout back to 3Ø from 15
# changed by xxx PersistTimeout from 1 secs to 6Ø seconds
InputTimeout    3Ø secs
OutputTimeout   2 minutes
ScriptTimeout   2 minutes
PersistTimeout       6Ø  secs
```

We could see the description of the timeout directives doing things such as:

- Limiting the time to wait for the client to send a request after connecting to the server before cancelling the connection.

- Limiting the time to allow for server scripts to finish. If the program does not finish within an allotted time, the server will send a TERM signal and then, 5 seconds later, a KILL signal (to stop the program.)
- Limiting the time to wait for the client to send a request after establishing a persistent connection to the server before cancelling the connection.

This looked very suspicious to us. The university systems programmers had tried to modify these parameters themselves before calling me in for consultation. They had turned off the MaxPersistRequest directive, which is the maximum number of requests to receive on a persistent connection. The default is 5; they had set it to 1 to turn off persistent connections. Since there was also a PersistTimeout value of 60 seconds, which will cancel the connection, we thought this might be a problem.

Test 2

Then we proceeded to our second set of tests. We changed the MaxPersistRequest to 100 and varied the timeout values:

- InputTimeout 30 seconds to 1 hour.
- OutputTimeout 2 minutes to 1 hour
- ScriptTimeout 2 minutes to 1 hour
- PersistTimeout 5 seconds to 1 hour.

We did not intend to leave the timeout values as 1 hour, but just wanted to eliminate these as a problem area. When we tried our stress test with 20 users, we still had problems getting in! These timeout values did not seem to help us.

We were stumped at this point. We then turned to the CGI script. We had looked at it before and could see a number of places where it closed a socket for various reasons such as errors – see the socket close in CGI script below:

```
/**********************************************************************
 * die --- Print a message and die.
 **********************************************************************/
void die(const char *mesg, int sock)
    {
    printf("<body bgcolor=#AAFFFF>");
    printf("<br><i>Error! : </i> %s.  Try reloading, or contact
support.\n",mesg);
    fputs(mesg, stderr);
    fputc('\n', stderr);
    shutdown(sock, 2);
    exit(Ø);
    }
```

We did not see any error messages, but, just for 'kicks', we disabled the socket closes. We thought it possible that the script might be sending a close to the wrong socket.

Test 3

This actually seemed to help! We were now able to log in even when 20 concurrent users were on. Log in was a little slow, but once we got into the application, we had no problems. Then, our systems programmers remembered that when the initial problem had occurred, students had called in because they were unable to log in and not because they had problems when actually using the application! This led us to think about what might be going on during the initial log in. Since the log in was done using SSL, it was possible that there was a problem with the SSL handshake.

We then thought to look at the SSL security directives. They had coded the SSL Cipher Specs to go from highest strength to lowest. We found some documentation for another Web server that led us to believe that this may make the SSL handshake longer. The SSL security directives are shown below:

```
# ================================================================== #
#        Security directives.
# ==================================================================
#        SSLCipherSpec directive
#
#        Specify the methods of encryption that an SSL connection will
#        support. Each encoded cipher specification is tested in the
#        order specified for compatibility with the requester. If the
#        requester supports a method specified here, an SSL connection
#        can be established. If not, the connection is refused.
#
#        Default:  All available cipher specifications are enabled by
#                  default (see directives below)
#
#        Syntax:   SSLCipherSpec <code>
#
#                  where <code> is one of:
#
#                  SSL V2:
#
#                  Code    Meaning                Note     Strength
#                  ====    =============          ====     ========
#                   21     RC4 (128 bit)            *      (weaker)
#                   22     RC4 (40 bit)
#                   23     RC2 (128 bit)            *          |
#                   24     RC2 (40 bit)                        V
#                   26     DES (56 bit)             *
#                   27     Triple DES (192 bit)     *     (stronger)
#
#                  SSL V3:
#
#                  Code    Meaning                Note     Strength
#                  ====    =============          ====     ========
#                   33     RC4 MD5 (128 bit)               (weaker)
#                   34     RC4 MD5 (128 bit)        *
#                   35     RC4 SHA (128 bit)        *          |
#                   36     RC2 MD5 (40 bit)                    V
#                   39     DES SHA (56 bit)
#                   3A     Triple DES SHA (192 bit) *     (stronger)
#
#                   * Note: Not supported in versions available
#                           outside North America.
#
# Examples:
# SSLCipherSpec 24
# SSLCipherSpec 22
SSLCipherSpec 39
SSLCipherSpec 27
SSLCipherSpec 21
SSLCipherSpec 23
SSLCipherSpec 26
SSLCipherSpec 22
SSLCipherSpec 24
SSLCipherSpec 3A
SSLCipherSpec 35
SSLCipherSpec 34
#SSLCipherSpec 39
SSLCipherSpec 33
SSLCipherSpec 36
```

Final test

So then we proceeded to what became our final test. The SSL cipher specs were set from lowest strength to highest instead of the way they had been coded – highest strength to lowest. This way the first cipher to be attempted would be the lowest strength cipher. This would take less time than the higher-strength cipher:

- SSLCipherSpec 33
- SSLCipherSpec 21
- SSLCipherSpec 22
- SSLCipherSpec 23
- SSLCipherSpec 24
- SSLCipherSpec 35 etc.

We also disabled all socket closes in the CGI script and increased the MaxPersistRequests to 100.

This allowed over 50 simultaneous users before any slowdowns occurred! We were able to connect to the application, but started slowing down when actually going from one page to the next within the application. This was an increase of nearly 150% without causing problems with the CICS region. So, now, any further improvements that could be made would seem to be in the realm of the application.

CONCLUSION

The lessons learned from this case came from truly understanding the problem and the architecture. As we said initially, for problem diagnosis, these are the key issues:

1 Understand the exact nature of the problem

2 Understand the architecture of the problem

3 Be able to recreate the problem.

If, at the outset, we had truly understood the nature of the problem, ie students had problems logging in to the application, instead of the symptom, ie CICS CPU usage was high, we might have followed the right track earlier.

Nalini Elkins
Inside Products (USA)

IP tour

In this article I will explain how one can determinate the network address and broadcast address from a given IP address and subnet address. I will also demonstrate the basics of how IP packets get routed. The principles given here apply to the old Class A, B, and C as well as CIDR (Classless Inter-Domain Routing), that is, 32-bit IP addresses.

IP AND BINARY

An IP address is made up of four numbers – these are called octets. Each octet has eight binary bits (8 bits=1 byte) that represent each number. So, 8 * 4 = 32-bit address. A typical IP address is split into two – the left part is used to hold the network address, the right side is used to hold the host address part. A Class C address format is:

```
<network>.<network>.<network>.<host>
  192    .  168    .   4     .  1Ø
```

Each byte of an IP address can contain a maximum value of 255. This is because the building blocks of an IP address are binary – remember that there are 8 bits in 1 byte and 4 bytes make up a 32-bit address. Each byte is represented using the ^2 (power of 2), with the MSB (most significant bit on the left) as follows:

```
128 64 32 16 8 4 2 1   decimal
 1   1  Ø  Ø Ø Ø 1 Ø   binary
```

In the example above, the bits that are set on are 128 + 64 + 2, which equates to 194.

With the old Class network addresses, one was quite restricted as to how the allotted IP range could be split into a network and a host address part. With CIDR this is not the case – with a few exceptions one is not limited as to how one can split the 32-bit IP address. How subnets are used deserves an article by itself; however, suffice it to say, the splitting of an IP address is done via the subnet. The subnet mask is used to split a large network into smaller network chunks. Within the IP, the subnet mask is used to achieve this. The mask is used to determine within the IP address which is the network part and which is the host part. The left-most side is the network address part, the right-most side is the host address – where the split occurs depends on how many (sub) networks and hosts one wishes to have. One can think of this process as a sliding scale: the more subnets one has, the fewer hosts one has to play with; the more hosts one wants, the fewer subnets one can have – get the picture? The subnet mask is sometimes appended to the address, for notational purposes, like:

192.168.8.0/24

This would indicate that the network address part is 192.168.8.0; the /24 means that the subnet mask is allocated 24 bits, in other words the subnet is 255.255.255.0.

Deciding on a subnet mask is down to your current and, more importantly, future networking needs. It is a good idea to have subnets; in fact it is insane not to if the company is geographically split. However, it is not imposed on you to have one.

CALCULATING THE IP NETWORK ADDRESS PART

When presented with an IP address how does one know what the network address is? Assume we have an IP address of 192.168.4.10. One cannot assume that the network part is 192.168.4. To figure out the network part, one must use the subnet mask address and work out some binary arithmetic.

The IP address of 192.168.4.10 is represented in binary as follows:

11000000 . 10101000 . 00000100 . 00001010

Let’s assume the subnet is set to 255.255.255.0.

In binary this is represented as follows:

11111111 . 11111111 . 11111111 . 00000000

Notice that all the 1s amount to 24 – that's /24 in notation.

Calculating the network address one has to carry out an 'AND' operation, that is, if both values are '1', then the result will be '1', otherwise the result is '0'. The AND is done on the IP and subnet mask. Using the above IP and subnet address the result is shown below:

11000000 . 10101000 . 00000100 . 00001010 (IP address)

11111111 . 11111111 . 11111111 . 00000000 (subnet mask)

11000000 . 10101000 . 00000100 . 00000000 (resulting AND)

This converts to 192.168.4.0, which is the network part address.

It is more commonly written as 192.168.4.0/24 (remember the /24 results from the amount of 1s in the subnet).

The above example was an easy one; after all, the subnet mask 255.255.255.0 is quite common. But what if the subnet was 255.255.255.240? Using the same IP address, 192.168.4.10:

11000000 . 10101000 . 00000100 . 00001010 (IP address)

11111111 . 11111111 . 11111111 . 11110000 (subnet mask)

11000000 . 10101000 . 00000100 . 00000000 (resulting AND)

This converts to 192.168.4.0, which is the network part address. It is more commonly written as 192.168.4.0/28.

CALCULATING THE IP BROADCAST ADDRESS PART

The broadcast address can also be calculated. The broadcast is an address in 'each' subnet that can, literally, broadcast a single packet to all IP-based hosts. It is used for DHCP and initial IP connections. Using the examples above, all that is required is to first negate or invert the subnet, usually called the 1s complement. This means turn all 1s to 0s and *vice versa*. Using this new address, an OR operation is performed against the original IP address (192.168.4.10). An OR operation means if either value is 1 or 0, or if both values are 1, the result is 1; otherwise, the result is 0.

The inverted subnet mask (255.255.255.0) in binary now becomes:

00000000 . 00000000 . 00000000 . 11111111

Now, using the OR operation:

11000000 . 10101000 . 00000100 . 00001010 (IP address)

00000000 . 00000000 . 00000000 . 11111111 (inverted subnet)

11000000 . 10101000 . 00000100 . 11111111 (resulting OR)

The above result converts to 192.168.4.255. This is the broadcast address. Notice that the last octet, which is the host section in the resulting 'OR' operation, has all 1s. This must be the case for all broadcast addresses.

For good measure and completeness, one can also calculate the host part of an IP by ANDing the inverted subnet against the IP address.

11000000 . 10101000 . 00000100 . 00001010 (IP address)

00000000 . 00000000 . 00000000 . 11111111 (inverted subnet mask)

00000000 . 00000000 . 00000000 . 00001010 (resulting AND)

This converts to 0.0.0.10, which is the host part address of 192.168.4.10.

Though this article is not focusing on subnets, the subnet mask is used on all the calculations. Thus far, you may be asking yourself how one can tell how many networks one has, or can have, within a given subnet mask. To determine this, take the decimal value from the last octet in the subnet mask and subtract that figure from 256. Looking at an example, assume the subnet mask is 255.255.255.224 and the network is 192.168.4.0 /27, so:

256 – 224 = 32

Using this number one can now progress. Simply add 32 to each previous last octet, starting at the network address 192.168.4.0:

- 192.168.4.0
- 192.168.4.32
- 192.168.4.64
- 192.168.4.96
- 192.168.4.128
- 192.168.4.160
- 192.168.4.192
- 192.168.4.224

In the above example eight networks can be used using the subnet mask of 255.255.255.224.

SENDING PACKETS ONWARDS

When sending data across a network, the IP packet needs to know where to go. The destination of the packet may well be on a different network. Without doubt, the packet will eventually arrive at the local default Gateway (unless the packet is sent via a point-to-point host route). When a packet reaches its first IP host/gateway/forwarder it will first AND the destination address with its own subnet mask. If the result is the same subnet mask, the gateway will forward it directly to the destination host. If the subnet mask is different, it will forward it to the next gateway. That gateway will then test to see whether any of its routes has the same subnet mask. This process is repeated until a gateway can match the subnet mask of the packet. In short, the packet is being routed.

Assume host A has a subnet mask of 255.255.240.0/20. An IP packet is sent to host B, which has the IP address 192.168.4.49.

The following example in binary compares the IPs:

11111111 . 11111111 . 11110000 . 00000000 (subnet mask of host A)

11000000 . 10101000 . 00000100 . 00110001 (IP of host B)

11000000 . 10101000 . 00000000 . 00000000 (resulting AND)

11000000 . 10101000 . 00000000 . 00000000 (network address of host A)

The resulting AND result is compared with the network address of host A. If they match, the packet is sent to host B. There is a match in the above example so the packet is accepted or sent, depending on the operation.

Each host with a network connection will have a routing table. This can be displayed by issuing netstat commands.

Routes can be manually added or deleted using the route command. The most basic format of the command is:

```
route <add|del> <-net|-host> <target IP> <netmask> <gw> <interface>
```

To add a point-to-point route from the local host to another host only with an IP of 192.168.6.12, one could use:

```
# route add -host 192.168.6.12 eth0
```

Notice that there is no need to specify the netmask; a host route implies a mask of 255.255.255.255.

To add a direct network route to the 192.168.8.0 network, one could use:

```
# route add -net 192.168.8.0 netmask 255.255.255.0 eth0
```

Alternatively, the mask notation method could be used:

```
# route add -net 192.168.8.0/24 eth0
```

To add the route 172.22.10.0, with a mask of 255.255.255.0 that requires the gateway 192.168.8.1 to access it, one could use:

```
# route add -net 172.22.10.0/24 gw 192.168.8.1 eth0
```

To delete a route, simply use the command one used to add the route, but replace the ‘add’ with ‘del’.

There are two types of routing one can use: static or dynamic. Static is the manual (or script) insertion of a route, which is what has been demonstrated in the above examples. Dynamic routing is a self-discovery daemon run by ‘routed’ or ‘gated’. These daemons will self-managed the routing tables. Generally I would use static routing because one has total control over the route table one creates.

Understanding IP addresses requires binary arithmetic and Boolean algebra (that’s the AND and OR operations). If this in not your ‘cup of tea’, there are IP calculators/converters on the Web that will do the conversions for you. I recommend using these utilities anyway as a back-up/confirmation of one’s own manual calculations.

David Tansley
Global Operations
ACE Overseas General (UK)

Using REXEC

The REXEC client in TCP/IP is a very powerful tool. It can be used to execute commands or scripts on remote platforms. We use this as a command and control mechanism to administer and manage Unix and XP platforms from the mainframe. We have two approaches to this: batch and on-line.

The first uses REXEC in batch to coordinate activities on Unix hosts with batch activities on the mainframe. For this I wrote a batch REXX EXEC to run REXEC. This was done to implement a level of security and to implement an approach to determine whether the remote command succeeded or failed. The basic return code from REXEC simply lets you know whether the command was successfully launched on the target host. It is not an indicator of whether the command actually worked or not. To solve that problem, my batch REXX EXEC called REXECX appends an echo of $? to the command string.

As you probably know, the $? variable in Unix is the closest thing Unix has to a return code. This appears as the last line of the output and REXECX parses the command output and will exit using the $? value as REXECX's return code. This allows JCL condition code processing to depend on success or failure of instream Unix steps.

The second issue was security. Nobody wanted the userid and password for each host hardcoded in a REXX EXEC or in a dataset that was accessible to the world. For this we implemented a NETRC dataset. REXEC supposedly supports NETRC, but I found it has problems, so REXECX will read the syntactically valid NETRC member and parse it internally. Using the dataset allows special access rules and read access to only authorized users. Using a NETRC PDS also avoids multiple copies of the same userid and password data for target hosts. Our host names are all of eight characters, so we chose to create a member in the NETRC PDS for each host.

The second approach uses REXEC interactively from TSO/ISPF. For this I wrote a REXEC dialog that prompts for the remote command and displays the output in ISPF browse. It stores the last used parameters in the ISPF profile and uses an internal crypto routine to encrypt/decrypt any passwords stored in the ISPF profile to avoid seeing any passwords in clear text. It will also generate an audit log by user to keep track of all commands executed.

REXECX REXX EXEC

```
/*********************************************************************/
/*                              REXX                                 */
```

```
/*******************************************************************/
/* Purpose: REXEC in batch to return a valid RC from the Remote CMD  */
/*-----------------------------------------------------------------*/
/* Syntax:  REXECX host command                                    */
/*-----------------------------------------------------------------*/
/* Parms: HOST       - IP Address or DNS name for target host        */
/*        COMMAND    - Command to execute on the remote host         */
/*                                                                 */
/* Notes: Requires a valid NETRC file in the JCL                   */
/*                                                                 */
/*******************************************************************/
/*                      Change Log                                 */
/*******************************************************************/
/* Accept parms                                                    */
/*******************************************************************/
parse arg host command
/*******************************************************************/
/* If NETRC is allocated, open and parse                           */
/*******************************************************************/
if listdsi("NETRC" "FILE") = Ø then
    do
    "EXECIO * DISKR NETRC (STEM NETRC. FINIS"
     do n=1 to netrc.Ø
        parse var netrc.n . hostname . uid . pw .
        if hostname = host then leave
     end
    end
else
    do
     say 'NETRC file is missing RC=2Ø'
     exit 2Ø
    end
/*******************************************************************/
/* Format the REXEC command and append the ECHO RC=$?              */
/*******************************************************************/
"REXEC -l" uid "-p" pw host command";echo RC=$?"
if RC <> Ø then exit RC
/*******************************************************************/
/* Read the contents of the OUTPUT DD (Remote output)              */
/*******************************************************************/
"EXECIO * DISKR OUTPUT (STEM OUTPUT. FINIS"
/*******************************************************************/
/* Display the output                                              */
/*******************************************************************/
do o=1 to output.Ø
    say strip(output.o)
end
/*******************************************************************/
/* Check last line for RC= string and EXIT with the CMDRC value ($?) */
/*******************************************************************/
```

```
last = output.Ø
if left(output.last,3) = 'RC=' then
    do
     parse var output.last 'RC=' cmdrc .
     exit cmdrc
    end
else
    do
     say 'Error: Did not find RC= in the last line'
     say 'Last line was:' output.last
     exit 12
    end
```

REXECX JCL

```
//jobcard…
//*****************************************************************
//* REXEC COMMANDS                                                *
//*****************************************************************
//REXEC    EXEC PGM=IKJEFTØ1,
//         PARM='REXECX myhost.com ps -ef'
//SYSEXEC  DD   DSN=my.exec.pds,DISP=SHR
//OUTPUT   DD   UNIT=VIO,SPACE=(TRK,(1,1Ø),RLSE),RECFM=VB,LRECL=1ØØØ
//NETRC    DD   DSN=my.netrc.pds(myhost),DISP=SHR
//SYSPRINT DD   SYSOUT=*
//SYSTSPRT DD   SYSOUT=*
//SYSTSIN  DD   DUMMY
```

IREXEC REXX EXEC

```
/*********************************************************************/
/*                               REXX                                */
/*********************************************************************/
/* Purpose: REXEC a command to a host                                */
/*-------------------------------------------------------------------*/
/* Syntax:  IREXEC                                                   */
/*-------------------------------------------------------------------*/
/* Parms: N/A        - N/A                                           */
/*                                                                   */
/* Notes: Will append an echo for $? after the command               */
/*                                                                   */
/*********************************************************************/
/*                      Change Log                                   */
/*********** @REFRESH BEGIN START    2ØØ4/Ø3/Ø6 13:16:32 *************/
/* Standard housekeeping activities                                  */
/*********************************************************************/
call time 'r'
parse arg parms
```

```
signal on syntax name trap
signal on failure name trap
signal on novalue name trap
probe = 'NONE'
modtrace = 'NO'
modspace = ''
call stdentry 'DIAGMSGS'
module = 'MAINLINE'
push trace() time('L') module 'From:' Ø 'Parms:' parms
if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
call modtrace 'START' Ø
/*********************************************************************/
/* Set local estoeric names                                          */
/*********************************************************************/
@vio   = 'VIO'
@sysda = 'SYSDA'
/*********** @REFRESH END   START    2ØØ4/Ø3/Ø6 13:16:32 *************/
/* Get all ISPF Profile variables                                    */
/*********************************************************************/
call ispwrap 8 "VGET (RHOST RID RCMD1 RCMD2) PROFILE"
/*********************************************************************/
/* If the password was found decrypt it                              */
/*********************************************************************/
if ispwrap(8 "VGET (RPASS) PROFILE") = Ø then rpass = crypt(rpass)
/*********************************************************************/
/* Display panel to get REXEC details                                */
/*********************************************************************/
mem.1  = ")ATTR                                                        "
mem.2  = " @ TYPE(TEXT) COLOR(TURQ)                                    "
mem.3  = " # TYPE(INPUT) CAPS(OFF) COLOR(GREEN)                        "
mem.4  = " $ TYPE(INPUT) CAPS(OFF) INTENS(NON)                         "
mem.5  = ")BODY EXPAND(//) WINDOW(76,5)                                "
mem.6  = "@Host     : #Z                                               "
mem.7  = "@UserID   : #Z                                               "
mem.8  = "@Password : $Z                                               "
mem.9  = "@Command  : #Z                                               "
mem.1Ø = "          : #Z                                               "
mem.11 = ")INIT                                                        "
mem.12 = " .ZVARS = '(RHOST RID RPASS RCMD1 RCMD2)'                    "
mem.13 = ")PROC                                                        "
mem.14 = " VER (&RHOST,NB)                                             "
mem.15 = " VER (&RID,NB)                                               "
mem.16 = " VER (&RPASS,NB)                                             "
mem.17 = " VER (&RCMD1,NB)                                             "
mem.18 = ")END                                                         "
/*********************************************************************/
/* Display the Dynamic Panel                                         */
/*********************************************************************/
call popdyn 'MEM' 4 'Enter' execname 'Parameters'
/*********************************************************************/
```

```
/* ALLOC the OUTPUT VIO DSN                                          */
/*********************************************************************/
"ALLOC F(OUTPUT) UNIT("@vio") SPACE(1 1) NEW CYLINDERS"
/*********************************************************************/
/* REXEC Command and append an echo for $?                           */
/*********************************************************************/
rcmd = rcmd1||rcmd2||';echo RC=$?'
call lock 'Executing command:' rcmd
"REXEC -l" rid "-p" rpass rhost rcmd
call rcexit RC 'REXEC Failure, reconfirm host, userid and password'
/*********************************************************************/
/* Encrypt the password and store all ISPF variables                 */
/*********************************************************************/
rpass = crypt(rpass)
call ispwrap "VPUT (RHOST RID RPASS RCMD1 RCMD2) PROFILE"
/*********************************************************************/
/* Scrape the RC= lines from the last line                           */
/*********************************************************************/
call tsotrap "EXECIO * DISKR OUTPUT (STEM OUTPUT. FINIS"
/*********************************************************************/
/* Check last line for RC= string and EXIT with the CMDRC value ($?) */
/*********************************************************************/
last = output.Ø
if left(output.last,3) = 'RC=' then
    do
     parse var output.last 'RC=' cmdrc .
     call msg 'Remote Command: "'rcmd'" executed on' rhost output.last
    end
else
    do
     msg1 = 'Remote Command Error: Did not find RC= in the last line'
     msg2 = 'Last line was:' output.last
     call msg msg1 msg2
    end
/*********************************************************************/
/* Log the command                                                   */
/*********************************************************************/
call logger 'RC='CMDRC rid rhost rcmd
/*********************************************************************/
/* Browse the OUTPUT                                                 */
/*********************************************************************/
call unlock
call brwsdd 'OUTPUT'
/*********************************************************************/
/* Shutdown                                                          */
/*********************************************************************/
shutdown: nop
/*********************************************************************/
/* Put unique shutdown logic before the call to stdexit              */
/*********** @REFRESH BEGIN STOP     2ØØ2/Ø8/Ø3 Ø8:42:33 *************/
```

```
/* Shutdown message and terminate                                     */
/**********************************************************************/
           call stdexit time('e')
/*********** @REFRESH END   STOP     2002/08/03 08:42:33 *************/
/*********** @REFRESH BEGIN SUBBOX   2004/03/10 01:25:03 *************/
/*                                                                    */
/* 31 Internal Subroutines provided in IREXEC                         */
/*                                                                    */
/* Last Subroutine REFRESH was 25 Jan 2005 23:31:17                   */
/*                                                                    */
/*                                                                    */
/* RCEXIT   - Exit on non-zero return codes                           */
/* TRAP     - Issue a common trap error message using rcexit          */
/* ERRMSG   - Build common error message with failing line number     */
/* STDENTRY - Standard Entry logic                                    */
/* STDEXIT  - Standard Exit logic                                     */
/* MSG      - Determine whether to SAY or ISPEXEC SETMSG the message */
/* DDCHECK  - Determine whether a required DD is allocated            */
/* DDLIST   - Returns number of DDs and populates DDLIST variable     */
/* DDDSNS   - Returns number of DSNs in a DD and populates DDDSNS     */
/* QDSN     - Make sure there are only one set of quotes              */
/* UNIQDSN  - Create a unique dataset name                            */
/* TEMPMEM  - EXECIO a stem into a TEMP PDS                           */
/* ISPWRAP  - Wrapper for ISPF commands                               */
/* TSOTRAP  - Capture the output from a TSO command in a stem         */
/* SETBORD  - Set the ISPF Pop-up active frame border colour          */
/* LOCK     - Put up a popup under foreground ISPF during long waits */
/* UNLOCK   - Unlock from a popup under foreground ISPF               */
/* PANDSN   - Create a unique PDS(MEM) name for a dynamic panel       */
/* POPDYN   - Addpop a Dynamic Panel                                  */
/* SAYDD    - Print messages to the requested DD                      */
/* JOBINFO  - Get job-related data from control blocks                */
/* PTR      - Pointer to a storage location                           */
/* STG      - Return the data from a storage location                 */
/* STACK    - UNLOAD, RELOAD, or LIST the Stack                       */
/* BRWSDD   - Invoke ISPF Browse on any DD                            */
/* LOGGER   - Append messages to a dynamic log                        */
/* CRYPT    - Encryption/Decryption routine                           */
/* MODTRACE - Module Trace                                            */
/*                                                                    */
/*********** @REFRESH END   SUBBOX   2004/03/10 01:25:03 *************/
/*********** @REFRESH BEGIN RCEXIT   2004/11/09 23:54:19 *************/
/* RCEXIT   - Exit on non-zero return codes                           */
/*--------------------------------------------------------------------*/
/* EXITRC   - Return code to exit with (if non-zero)                  */
/* ZEDLMSG  - Message text for it with for non-zero EXITRCs           */
/**********************************************************************/
rcexit: parse arg EXITRC zedlmsg
         EXITRC = abs(EXITRC)
         if EXITRC <> 0 then
```

```
            do
             trace 'o'
/*********************************************************************/
/* If execution environment is ISPF then VPUT ZISPFRC                */
/*********************************************************************/
             if execenv = 'TSO' | execenv = 'ISPF' then
                do
                 if ispfenv = 'YES' then
                    do
                     zispfrc = EXITRC
/*********************************************************************/
/* Does not call ISPWRAP to avoid obscuring error message modules    */
/*********************************************************************/
                     address ISPEXEC "VPUT (ZISPFRC)"
                    end
                end
/*********************************************************************/
/* If a message is provided, wrap it in date, time and EXITRC        */
/*********************************************************************/
             if zedlmsg <> '' then
                do
                 zedlmsg = time('L') execname zedlmsg 'RC='EXITRC
                 call msg zedlmsg
                end
/*********************************************************************/
/* Write the contents of the Parentage Stack                         */
/*********************************************************************/
             stacktitle = 'Parentage Stack Trace ('queued()' entries):'
/*********************************************************************/
/* Write to MSGDD if background and MSGDD exists                     */
/*********************************************************************/
             if tsoenv = 'BACK' then
                do
                 if subword(zedlmsg,9,1) = msgdd then
                    do
                     say zedlmsg
                     signal shutdown
                    end
                 else
                    do
                     call saydd msgdd 1 zedlmsg
                     call saydd msgdd 1 stacktitle
                    end
                end
             else
/*********************************************************************/
/* Write to the ISPF Log if foreground                               */
/*********************************************************************/
                do
                 zerrlm = zedlmsg
                 address ISPEXEC "LOG MSG(ISRZØØ3)"
                 zerrlm = center(' 'stacktitle' ',78,'-')
                 address ISPEXEC "LOG MSG(ISRZØØ3)"
                end
/*********************************************************************/
/* Unload the Parentage Stack                                        */
/*********************************************************************/
             do queued()
                pull stackinfo
                if tsoenv = 'BACK' then
                   do
                    call saydd msgdd Ø stackinfo
                   end
                else
                   do
                    zerrlm = stackinfo
                    address ISPEXEC "LOG MSG(ISRZØØ3)"
                   end
             end
/*********************************************************************/
/* Put a terminator in the ISPF Log for the Parentage Stack          */
/*********************************************************************/
             if tsoenv = 'FORE' then
                do
                 zerrlm = center(' 'stacktitle' ',78,'-')
                 address ISPEXEC "LOG MSG(ISRZØØ3)"
                end
/*********************************************************************/
/* Signal SHUTDOWN.  SHUTDOWN label MUST exist in the program        */
/*********************************************************************/
             signal shutdown
            end
         else
            return
/*********** @REFRESH END   RCEXIT   2ØØ4/11/Ø9 23:54:19 *************/
/*********** @REFRESH BEGIN TRAP     2ØØ4/12/13 14:ØØ:48 *************/
/* TRAP     - Issue a common trap error message using rcexit         */
/*-------------------------------------------------------------------*/
/* PARM     - N/A                                                    */
/*********************************************************************/
trap: trace 'off'
       traptype = condition('C')
       if traptype = 'SYNTAX' then
          msg = errortext(RC)
       else
          msg = condition('D')
       trapline = strip(sourceline(sigl))
       msg = traptype 'TRAP:' msg', Line:' sigl '"'trapline'"'
       if trap = 'YES' & tsoenv = 'BACK' then
          do
```

```
           trap = 'NO'
           traplinemsg = msg
           say traplinemsg
           signal on syntax name trap
           signal on failure name trap
           signal on novalue name trap
           say
           say center(' Trace of failing instruction ',78,'-')
           trace 'i'
           interpret trapline
          end
       if trap = 'NO' & tsoenv = 'BACK' then
          do
           say center(' Trace of failing instruction ',78,'-')
           say
          end
       if tsoenv = 'FORE' then
          call rcexit 666 msg
       else
          call rcexit 666 traplinemsg
/*********** @REFRESH END   TRAP     2004/12/13 14:00:48 *************/
/*********** @REFRESH BEGIN ERRMSG   2002/08/10 16:53:04 *************/
/* ERRMSG   - Build common error message with failing line number    */
/*-------------------------------------------------------------------*/
/* ERRLINE  - The failing line number passed by caller from SIGL     */
/* TEXT     - Error message text passed by caller                    */
/*********************************************************************/
errmsg: nop
         parse arg errline text
         return 'Error on statement' errline',' text
/*********** @REFRESH END   ERRMSG   2002/08/10 16:53:04 *************/
/*********** @REFRESH BEGIN STDENTRY 2004/11/23 21:54:51 *************/
/* STDENTRY - Standard Entry logic                                   */
/*-------------------------------------------------------------------*/
/* MSGDD    - Optional MSGDD used only in background                 */
/*********************************************************************/
stdentry: module = 'STDENTRY'
           if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
           parse arg sparms
           push trace() time('L') module 'From:' sigl 'Parms:' sparms
           arg msgdd
           parse upper source . . execname . execdsn . . execenv .
/*********************************************************************/
/* Start-up values                                                   */
/*********************************************************************/
           EXITRC = 0
           MAXRC = 0
           trap = 'YES'
           ispfenv = 'NO'
           popup = 'NO'
           lockpop = 'NO'
           headoff = 'NO'
           hcreator = 'NO'
           keepstack = 'NO'
           lpar = mvsvar('SYSNAME')
           jobname = mvsvar('SYMDEF','JOBNAME')
           zedlmsg = 'Default shutdown message'
/*******************************************************************/
/* Determine environment                                           */
/*******************************************************************/
           if substr(execenv,1,3) <> 'TSO' & execenv <> 'ISPF' then
              tsoenv = 'NONE'
           else
              do
               tsoenv = sysvar('SYSENV')
               signal off failure
              "ISPQRY"
               ISPRC = RC
               if ISPRC = Ø then
                  do
                   ispfenv = 'YES'
/*******************************************************************/
/* Check if HEADING ISPF table exists already, if so set HEADOFF=YES */
/*******************************************************************/
                   call ispwrap "VGET (ZSCREEN)"
                   if tsoenv = 'BACK' then
                      htable = jobinfo(1)||jobinfo(2)
                   else
                      htable = userid()||zscreen
                   TBCRC = ispwrap(8 "TBCREATE" htable "KEYS(HEAD)")
                   if TBCRC = Ø then
                      do
                       headoff = 'NO'
                       hcreator = 'YES'
                      end
                   else
                      do
                       headoff = 'YES'
                      end
                  end
               signal on failure name trap
              end
/*******************************************************************/
/* MODTRACE must occur after the setting of ISPFENV                */
/*******************************************************************/
           call modtrace 'START' sigl
/*******************************************************************/
/* Start-up message (if batch)                                     */
/*******************************************************************/
           startmsg = execname 'started' date() time() 'on' lpar
```

```
           if tsoenv = 'BACK' & sysvar('SYSNEST') = 'NO' &,
              headoff = 'NO' then
              do
               jobinfo = jobinfo()
               parse var jobinfo jobtype jobnum .
               say jobname center(' 'startmsg' ',61,'-') jobtype jobnum
               say
               if ISPRC = -3 then
                  do
                   call saydd msgdd 1 'ISPF ISPQRY module not found,',
                                      'ISPQRY is usually in the LINKLST'
                   call rcexit 2Ø 'ISPF ISPQRY module is missing'
                  end
/*********************************************************************/
/* If MSGDD is provided, write the STARTMSG and SYSEXEC DSN to MSGDD */
/*********************************************************************/
               if msgdd <> '' then
                  do
                   call ddcheck msgdd
                   call saydd msgdd 1 startmsg
                   call ddcheck 'SYSEXEC'
                   call saydd msgdd Ø execname 'loaded from' sysdsname
/*********************************************************************/
/* If there are PARMS, write them to the MSGDD                       */
/*********************************************************************/
                   if parms <> '' then
                      call saydd msgdd Ø 'Parms:' parms
/*********************************************************************/
/* If there is a STEPLIB, write the STEPLIB DSN MSGDD                */
/*********************************************************************/
                   if listdsi('STEPLIB' 'FILE') = Ø then
                      do
                       steplibs = dddsns('STEPLIB')
                       call saydd msgdd Ø 'STEPLIB executables loaded',
                           'from' word(dddsns,1)
                       if dddsns('STEPLIB') > 1 then
                          do
                           do stl=2 to steplibs
                              call saydd msgdd Ø copies(' ',31),
                                   word(dddsns,stl)
                           end
                          end
                      end
                  end
              end
/*********************************************************************/
/* If foreground, save ZFKA and turn off the FKA display             */
/*********************************************************************/
           else
              do
```

```
                 fkaset = 'OFF'
                 call ispwrap "VGET (ZFKA) PROFILE"
                 if zfka <> 'OFF' & tsoenv = 'FORE' then
                    do
                     fkaset = zfka
                     fkacmd = 'FKA OFF'
                     call ispwrap "CONTROL DISPLAY SAVE"
                    call ispwrap "DISPLAY PANEL(ISPBLANK) COMMAND(FKACMD)"
                     call ispwrap "CONTROL DISPLAY RESTORE"
                    end
                end
/*****************************************************************/
           pull tracelvl . module . sigl . sparms
           call modtrace 'STOP' sigl
           interpret 'trace' tracelvl
           return
/*********** @REFRESH END   STDENTRY 2004/11/23 21:54:51 *************/
/*********** @REFRESH BEGIN STDEXIT  2004/08/02 06:06:40 *************/
/* STDEXIT  - Standard Exit logic                                 */
/*---------------------------------------------------------------*/
/* ENDTIME  - Elapsed time                                        */
/* Note: Caller must set KEEPSTACK if the stack is valid          */
/*****************************************************************/
stdexit: module = 'STDEXIT'
          if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg endtime
          endmsg = execname 'ended' date() time() format(endtime,,1)
/*****************************************************************/
/* if MAXRC is greater then EXITRC then set EXITRC to MAXRC       */
/*****************************************************************/
          EXITRC = max(EXITRC,MAXRC)
          endmsg = endmsg 'on' lpar 'RC='EXITRC
          if tsoenv = 'BACK' & sysvar('SYSNEST') = 'NO' &,
             headoff = 'NO' then
             do
              say
              say jobname center(' 'endmsg' ',61,'-') jobtype jobnum
/*****************************************************************/
/* Make sure this isn't a MSGDD missing error then log to MSGDD    */
/*****************************************************************/
                if msgdd <> '' & subword(zedlmsg,9,1) <> msgdd then
                   do
                    call saydd msgdd 1 execname 'ran in' endtime 'seconds'
                    call saydd msgdd 0 endmsg
                   end
             end
/*****************************************************************/
```

```
/* If foreground, reset the FKA if necessary                        */
/*******************************************************************/
          else
             do
              if fkaset <> 'OFF' then
                 do
                  fkafix = 'FKA'
                  call ispwrap "CONTROL DISPLAY SAVE"
                  call ispwrap "DISPLAY PANEL(ISPBLANK) COMMAND(FKAFIX)"
                  if fkaset = 'SHORT' then
                     call ispwrap "DISPLAY PANEL(ISPBLANK)",
                                  "COMMAND(FKAFIX)"
                  call ispwrap "CONTROL DISPLAY RESTORE"
                 end
             end
/*******************************************************************/
/* Clean up the temporary HEADING table                             */
/*******************************************************************/
          if ispfenv = 'YES' & hcreator = 'YES' then
             call ispwrap "TBEND" htable
/*******************************************************************/
/* Remove STDEXIT and MAINLINE Parentage Stack entries, if there    */
/*******************************************************************/
          call modtrace 'STOP' sigl
          if queued() > Ø then pull . . module . sigl . sparms
          if queued() > Ø then pull . . module . sigl . sparms
          if tsoenv = 'FORE' & queued() > Ø & keepstack = 'NO' then
             pull . . module . sigl . sparms
/*******************************************************************/
/* if the Parentage Stack is not empty, display its contents        */
/*******************************************************************/
          if queued() > Ø & keepstack = 'NO' then
             do
              say queued() 'Leftover Parentage Stack Entries:'
              say
              do queued()
                 pull stackundo
                 say stackundo
              end
              EXITRC = 1
             end
/*******************************************************************/
/* Exit                                                             */
/*******************************************************************/
          exit(EXITRC)
/*********** @REFRESH END   STDEXIT  2ØØ4/Ø8/Ø2 Ø6:Ø6:4Ø *************/
/*********** @REFRESH BEGIN MSG      2ØØ2/Ø9/11 Ø1:35:53 *************/
/* MSG      - Determine whether to SAY or ISPEXEC SETMSG the message */
/*-----------------------------------------------------------------*/
/* ZEDLMSG  - The long message variable                             */
```

```
/*********************************************************************/
msg: module = 'MSG'
      parse arg zedlmsg
      if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
      parse arg sparms
      push trace() time('L') module 'From:' sigl 'Parms:' sparms
      call modtrace 'START' sigl
/*********************************************************************/
/* If this is background or OMVS use SAY                             */
/*********************************************************************/
      if tsoenv = 'BACK' | execenv = 'OMVS' then
         say zedlmsg
      else
/*********************************************************************/
/* If this is foreground and ISPF is available, use SETMSG           */
/*********************************************************************/
         do
          if ispfenv = 'YES' then
/*********************************************************************/
/* Does not call ISPWRAP to avoid obscuring error message modules    */
/*********************************************************************/
             address ISPEXEC "SETMSG MSG(ISRZØØØ)"
          else
             say zedlmsg
         end
      pull tracelvl . module . sigl . sparms
      call modtrace 'STOP' sigl
      interpret 'trace' tracelvl
      return
/*********** @REFRESH END   MSG      2ØØ2/Ø9/11 Ø1:35:53 *************/
/*********** @REFRESH BEGIN DDCHECK  2ØØ4/11/Ø9 22:48:36 *************/
/* DDCHECK  - Determine whether a required DD is allocated           */
/*-------------------------------------------------------------------*/
/* DD       - DDNAME to confirm                                      */
/*********************************************************************/
ddcheck: module = 'DDCHECK'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg dd
          dderrmsg = 'OK'
          LRC = listdsi(dd "FILE")
/*********************************************************************/
/* Allow sysreason=3 & 22 to verify SYSOUT and tape DD statements    */
/*********************************************************************/
          if LRC <> Ø & sysreason <> 3 & sysreason <> 22 then
             do
              dderrmsg = errmsg(sigl 'Required DD' dd 'is missing')
              call rcexit LRC dderrmsg sysmsglvl2
```

```
           end
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return
/*********** @REFRESH END   DDCHECK  2004/11/09 22:48:36 *************/
/*********** @REFRESH BEGIN DDLIST   2002/12/15 04:54:32 *************/
/* DDLIST   - Returns number of DDs and populates DDLIST variable    */
/*-------------------------------------------------------------------*/
/* N/A      - None                                                   */
/*********************************************************************/
ddlist: module = 'DDLIST'
        if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
        parse arg sparms
        push trace() time('L') module 'From:' sigl 'Parms:' sparms
        call modtrace 'START' sigl
/*********************************************************************/
/* Trap the output from the LISTA STATUS command                     */
/*********************************************************************/
        call outtrap 'lines.'
        address TSO "LISTALC STATUS"
        call outtrap 'off'
        ddnum = 0
/*********************************************************************/
/* Parse out the DDNAMEs and concatenate into a list                 */
/*********************************************************************/
        ddlist = ''
        do ddl=1 to lines.0
           if words(lines.ddl) = 2 then
              do
               parse upper var lines.ddl ddname .
               ddlist = ddlist ddname
               ddnum = ddnum + 1
              end
           else
              do
               iterate
              end
        end
/*********************************************************************/
/* Return the number of DDs                                          */
/*********************************************************************/
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return ddnum
/*********** @REFRESH END   DDLIST   2002/12/15 04:54:32 *************/
/*********** @REFRESH BEGIN DDDSNS   2002/09/11 00:37:36 *************/
/* DDDSNS   - Returns number of DSNs in a DD and populates DDDSNS    */
/*-------------------------------------------------------------------*/
```

```
/* TARGDD   - DD to return DSNs for                                   */
/*********************************************************************/
dddsns: module = 'DDDSNS'
         if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
         arg targdd
         if targdd = '' then call rcexit 77 'DD missing for DDDSNS'
/*********************************************************************/
/* Trap the output from the LISTA STATUS command                     */
/*********************************************************************/
         x = outtrap('lines.')
         address TSO "LISTALC STATUS"
         dsnnum = Ø
         ddname = '$DDNAME$'
/*********************************************************************/
/* Parse out the DDNAMEs, locate the target DD and concatentate DSNs */
/*********************************************************************/
         do ddd=1 to lines.Ø
            select
               when words(lines.ddd) = 1 & targdd = ddname &,
                    lines.ddd <> 'KEEP' then
                    dddsns = dddsns strip(lines.ddd)
               when words(lines.ddd) = 1 & strip(lines.ddd),
                    <> 'KEEP' then
                    dddsn.ddd = strip(lines.ddd)
               when words(lines.ddd) = 2 then
                    do
                     parse upper var lines.ddd ddname .
                     if targdd = ddname then
                        do
                         fdsn = ddd - 1
                         dddsns = lines.fdsn
                        end
                    end
               otherwise iterate
            end
         end
/*********************************************************************/
/* Get the last DD                                                   */
/*********************************************************************/
         ddnum = ddlist()
         lastdd = word(ddlist,ddnum)
/*********************************************************************/
/* Remove the last DSN from the list if not the last DD or SYSEXEC   */
/*********************************************************************/
         if targdd <> 'SYSEXEC' & targdd <> lastdd then
            do
             dsnnum = words(dddsns) - 1
             dddsns = subword(dddsns,1,dsnnum)
            end
/*********************************************************************/
/* Return the number of DSNs in the DD                               */
/*********************************************************************/
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return dsnnum
/*********** @REFRESH END   DDDSNS   2002/09/11 00:37:36 *************/
/*********** @REFRESH BEGIN QDSN     2002/09/11 01:15:23 *************/
/* QDSN     - Make sure there are only one set of quotes             */
/*------------------------------------------------------------------*/
/* QDSN     - The DSN                                                */
/*********************************************************************/
qdsn: module = 'QDSN'
       if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
       parse arg sparms
       push trace() time('L') module 'From:' sigl 'Parms:' sparms
       call modtrace 'START' sigl
       parse arg qdsn
       qdsn = "'"strip(qdsn,"B","'")"'"
       pull tracelvl . module . sigl . sparms
       call modtrace 'STOP' sigl
       interpret 'trace' tracelvl
       return qdsn
/*********** @REFRESH END   QDSN     2002/09/11 01:15:23 *************/
/*********** @REFRESH BEGIN UNIQDSN  2004/09/01 18:03:04 *************/
/* UNIQDSN  - Create a unique dataset name                           */
/*-------------------------------------------------------------------*/
/* PARM     - N/A                                                    */
/*********************************************************************/
uniqdsn: module = 'UNIQDSN'
          if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
/*********************************************************************/
/* Compose a DSN using userid, exec name, job number, date, and time */
/*********************************************************************/
          jnum = jobinfo(1) || jobinfo(2)
          udate = 'D'space(translate(date('O'),'','/'),0)
          utime = 'T'left(space(translate(time('L'),'',':.'),0),7)
          uniqdsn = userid()'.'execname'.'jnum'.'udate'.'utime
          if sysdsn(qdsn(uniqdsn)) = 'OK' then
             do
/*********************************************************************/
/* Wait 1 seconds to ensure a unique dataset (necessary on z990)     */
/*********************************************************************/
              RC = syscalls('ON')
             address SYSCALL "SLEEP 1"
             RC = syscalls('OFF')
             uniqdsn = uniqdsn()
            end
/********************************************************************/
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return uniqdsn
/*********** @REFRESH END   UNIQDSN  2004/09/01 18:03:04 *************/
/*********** @REFRESH BEGIN TEMPMEM  2004/09/01 17:20:19 *************/
/* TEMPMEM  - EXECIO a stem into a TEMP PDS                          */
/*------------------------------------------------------------------*/
/* TEMPMEM  - The member to create                                   */
/********************************************************************/
tempmem: module = 'TEMPMEM'
          if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg tempmem
/********************************************************************/
/* Create a unique DD name                                           */
/********************************************************************/
          if length(tempmem) < 7 then
             tempdd = '$'tempmem'$'
          else
             tempdd = '$'substr(tempmem,2,6)'$'
/********************************************************************/
/* If TEMPDD exists, then FREE it                                    */
/********************************************************************/
          if listdsi(tempdd 'FILE') = 0 then "FREE F("tempdd")"
/********************************************************************/
/* Generate the TEMPDSN                                              */
/********************************************************************/
          tempdsn = uniqdsn()'('tempmem')'
/********************************************************************/
/* ALLOCATE the TEMP DD and member                                   */
/********************************************************************/
          call tsotrap "ALLOC F("tempdd") DA("qdsn(tempdsn)") NEW",
                       "LRECL(80) BLKS(0) DIR(1) SPACE(1) CATALOG",
                       "UNIT("@sysda") RECFM(F B)"
/********************************************************************/
/* Write the STEM to the TEMP DD                                     */
/********************************************************************/
          call tsotrap 'EXECIO * DISKW' tempdd '(STEM' tempmem'. FINIS'
/********************************************************************/
/* DROP the stem variable                                            */
/********************************************************************/
          interpret 'drop' tempmem'.'
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return tempdd
/*********** @REFRESH END   TEMPMEM  2ØØ4/Ø9/Ø1 17:2Ø:19 *************/
/*********** @REFRESH BEGIN ISPWRAP  2ØØ2/Ø9/11 Ø1:11:43 *************/
/* ISPWRAP  - Wrapper for ISPF commands                              */
/*-------------------------------------------------------------------*/
/* VALIDRC  - Optional valid RC from the ISPF command, defaults to Ø */
/* ISPPARM  - Valid ISPF command                                     */
/*********************************************************************/
ispwrap: module = 'ISPWRAP'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          parse arg ispparm
          zerrlm = 'NO ZERRLM'
/*********************************************************************/
/* If the optional valid_rc parm is present use it, if not assume Ø  */
/*********************************************************************/
          parse var ispparm valid_rc isp_cmd
          if datatype(valid_rc,'W') = Ø then
             do
              valid_rc = Ø
              isp_cmd = ispparm
             end
          address ISPEXEC isp_cmd
          IRC = RC
/*********************************************************************/
/* If RC = Ø then return                                             */
/*********************************************************************/
          if IRC <= valid_rc then
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return IRC
             end
          else
             do
              perrmsg = errmsg(sigl 'ISPF Command:')
              call rcexit IRC perrmsg isp_cmd strip(zerrlm)
             end
/*********** @REFRESH END   ISPWRAP  2ØØ2/Ø9/11 Ø1:11:43 *************/
/*********** @REFRESH BEGIN TSOTRAP  2ØØ2/12/15 Ø5:18:45 *************/
/* TSOTRAP  - Capture the output from a TSO command in a stem        */
/*-------------------------------------------------------------------*/
/* VALIDRC  - Optional valid RC, defaults to zero                    */
/* TSOPARM  - Valid TSO command                                      */
```

```
/*********************************************************************/
tsotrap: module = 'TSOTRAP'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          parse arg tsoparm
/*********************************************************************/
/* If the optional valid_rc parm is present use it, if not assume Ø  */
/*********************************************************************/
          parse var tsoparm valid_rc tso_cmd
          if datatype(valid_rc,'W') = Ø then
             do
              valid_rc = Ø
              tso_cmd = tsoparm
             end
          call outtrap 'tsoout.'
          tsoline = sigl
          address TSO tso_cmd
          CRC = RC
          call outtrap 'off'
/*********************************************************************/
/* If RC = Ø then return                                             */
/*********************************************************************/
          if CRC <= valid_rc then
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return CRC
             end
          else
             do
              trapmsg = center(' TSO Command Error Trap ',78,'-')
              terrmsg = errmsg(sigl 'TSO Command:')
/*********************************************************************/
/* If RC <> Ø then format output depending on environment            */
/*********************************************************************/
              if tsoenv = 'BACK' | execenv = 'OMVS' then
                 do
                  say trapmsg
                  do c=1 to tsoout.Ø
                     say tsoout.c
                  end
                  say trapmsg
                  call rcexit CRC terrmsg tso_cmd
                 end
              else
/*********************************************************************/
/* If this is foreground and ISPF is available, use the ISPF LOG     */
```

```
/*********************************************************************/
                 do
                  if ispfenv = 'YES' then
                     do
                      zedlmsg = trapmsg
/*********************************************************************/
/* Does not call ISPWRAP to avoid obscuring error message modules    */
/*********************************************************************/
                      address ISPEXEC "LOG MSG(ISRZØØØ)"
                      do c=1 to tsoout.Ø
                         zedlmsg = tsoout.c
                         address ISPEXEC "LOG MSG(ISRZØØØ)"
                      end
                      zedlmsg = trapmsg
                      address ISPEXEC "LOG MSG(ISRZØØØ)"
                      call rcexit CRC terrmsg tso_cmd,
                          ' see the ISPF Log (Option 7.5) for details'
                     end
                  else
                     do
                      say trapmsg
                      do c=1 to tsoout.Ø
                         say tsoout.c
                      end
                      say trapmsg
                      call rcexit CRC terrmsg tso_cmd
                     end
                 end
             end
/*********** @REFRESH END   TSOTRAP  2ØØ2/12/15 Ø5:18:45 *************/
/*********** @REFRESH BEGIN SETBORD  2ØØ2/Ø9/11 Ø1:16:41 *************/
/* SETBORD  - Set the ISPF Pop-up active frame border colour         */
/*-------------------------------------------------------------------*/
/* COLOR    - Colour for the Active Frame Border                     */
/*********************************************************************/
setbord: module = 'SETBORD'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg color
/*********************************************************************/
/* Parse and validate colour                                         */
/*********************************************************************/
          if color = '' then color = 'YELLOW'
/*********************************************************************/
/* Build a temporary panel                                           */
/*********************************************************************/
          ispopt11.1=")BODY                       "
          ispopt11.2="%Command ===>_ZCMD        + "
          ispopt11.3=")INIT                          "
          ispopt11.4="&ZCMD = ' '                    "
          ispopt11.5="VGET (COLOR) SHARED            "
          ispopt11.6="&ZCOLOR = &COLOR               "
          ispopt11.7=".RESP = END                    "
          ispopt11.8=")END                           "
/*******************************************************************/
/* Allocate and load the Dynamic Panel                             */
/*******************************************************************/
          setdd = tempmem('ISPOPT11')
/*******************************************************************/
/* LIBDEF the DSN, VPUT @TFCOLOR, and run CUAATTR                  */
/*******************************************************************/
          call ispwrap "LIBDEF ISPPLIB LIBRARY ID("setdd") STACK"
          call ispwrap "VPUT (COLOR) SHARED"
          call ispwrap "SELECT PGM(ISPOPT) PARM(ISPOPT11)"
          call ispwrap "LIBDEF ISPPLIB"
          call tsotrap "FREE F("setdd") DELETE"
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return
/*********** @REFRESH END   SETBORD  2002/09/11 01:16:41 *************/
/*********** @REFRESH BEGIN LOCK     2004/09/01 18:00:03 *************/
/* LOCK     - Put up a popup under foreground ISPF during long waits */
/*-----------------------------------------------------------------*/
/* LOCKMSG  - Message for the pop-up screen                        */
/*******************************************************************/
lock: module = 'LOCK'
       if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
       parse arg sparms
       push trace() time('L') module 'From:' sigl 'Parms:' sparms
       call modtrace 'START' sigl
       parse arg lockmsg
       if lockmsg = '' then lockmsg = 'Please be patient'
       if tsoenv = 'FORE' then
          do
/*******************************************************************/
/* Use the length of the lockmsg to determine the pop-up size      */
/*******************************************************************/
           if length(lockmsg) < 76 then
              locklen = length(lockmsg) + 2
           else
              locklen = 77
           if locklen <= 10 then locklen = 10
/*******************************************************************/
/* Build a temporary panel                                         */
/*******************************************************************/
           lock.1 = ")BODY EXPAND(//) WINDOW("locklen",1)"
           lock.2 = "%&LOCKMSG                     "
           lock.3 = ")END                          "
/*********************************************************************/
/* Lock the screen and put up a pop-up                              */
/*********************************************************************/
           call ispwrap "CONTROL DISPLAY LOCK"
           call popdyn 'LOCK' 8 execname 'Please be patient'
           lockpop = 'YES'
          end
       pull tracelvl . module . sigl . sparms
       call modtrace 'STOP' sigl
       interpret 'trace' tracelvl
       return
/*********** @REFRESH END   LOCK     2004/09/01 18:00:03 *************/
/*********** @REFRESH BEGIN UNLOCK   2003/10/18 09:33:19 *************/
/* UNLOCK   - Unlock from a pop-up under foreground ISPF            */
/*-------------------------------------------------------------------*/
/* PARM     - N/A                                                    */
/*********************************************************************/
unlock: module = 'UNLOCK'
         if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
         if tsoenv = 'FORE' then
             do
              if lockpop = 'YES' then
                 do
                  call ispwrap "REMPOP"
                  lockpop = 'NO'
                 end
              if popup = 'YES' then
                 do
                  call setbord 'BLUE'
                  call ispwrap "REMPOP"
                  popup = 'NO'
                 end
             end
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return
/*********** @REFRESH END   UNLOCK   2003/10/18 09:33:19 *************/
/*********** @REFRESH BEGIN PANDSN   2004/04/28 00:46:04 *************/
/* PANDSN   - Create a unique PDS(MEM) name for a dynamic panel     */
/*-------------------------------------------------------------------*/
/* PANEL    - Dynamic panel name                                     */
/*********************************************************************/
pandsn: module = 'PANDSN'
         if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
         arg panel
         pandsn = uniqdsn()'('panel')'
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return pandsn
/*********** @REFRESH END   PANDSN   2004/04/28 00:46:04 *************/
/*********** @REFRESH BEGIN POPDYN   2002/09/11 01:15:11 *************/
/* POPDYN   - Addpop a Dynamic Panel                                 */
/*-------------------------------------------------------------------*/
/* DYN      - Dynamic panel name                                     */
/* DYNROW   - Default row for ADDPOP, defaults to 1                  */
/* DYNMSG   - ADDPOP Window title                                    */
/*********************************************************************/
popdyn: module = 'POPDYN'
         if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
         parse arg dyn dynrow dynmsg
/*********************************************************************/
/* Set the default ADDPOP row location                               */
/*********************************************************************/
         if dynrow = '' then dynrow = 1
/*********************************************************************/
/* Set the default ADDPOP window title to the current exec name      */
/*********************************************************************/
         if dynmsg = '' then dynmsg = execname
/*********************************************************************/
/* Check if the RETURN option is specified in the DYNMSG             */
/*********************************************************************/
         dynreturn = 'NO'
         if word(dynmsg,1) = 'RETURN' then
            parse var dynmsg dynreturn dynmsg
/*********************************************************************/
/* Allocate and load the Dynamic Panel                               */
/*********************************************************************/
         dyndd = tempmem(dyn)
/*********************************************************************/
/* LIBDEF the POPDYN panel                                           */
/*********************************************************************/
         call ispwrap "LIBDEF ISPPLIB LIBRARY ID("dyndd") STACK"
/*********************************************************************/
/* Change the Active Frame Colour                                    */
/*********************************************************************/
         call setbord 'YELLOW'
/*********************************************************************/
/* set the POPUP variable if this is not a LOCK request              */
```

```
/*****************************************************************/
         if dyn = 'LOCK' & popup = 'NO' then
            popup = 'NO'
         else
            popup = 'YES'
/*****************************************************************/
/* Put up the pop-up                                             */
/*****************************************************************/
         zwinttl = dynmsg
         call ispwrap "ADDPOP ROW("dynrow")"
         DRC = ispwrap(8 "DISPLAY PANEL("dyn")")
         call ispwrap "LIBDEF ISPPLIB"
         call tsotrap "FREE F("dyndd") DELETE"
/*****************************************************************/
/* Change the Active Frame Colour                                */
/*****************************************************************/
         call setbord 'BLUE'
/*****************************************************************/
/* Determine how to return                                       */
/*****************************************************************/
         if dynreturn = 'NO' then
            call rcexit DRC 'terminated by user'
         if dynreturn = 'RETURN' & DRC = 8 then
            do
             call ispwrap "REMPOP"
             popup = 'NO'
            end
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return DRC
/*********** @REFRESH END   POPDYN   2002/09/11 01:15:11 *************/
/*********** @REFRESH BEGIN SAYDD    2004/03/29 23:48:37 *************/
/* SAYDD    - Print messages to the requested DD                 */
/*---------------------------------------------------------------*/
/* MSGDD    - DDNAME to write messages to                        */
/* MSGLINES - number of blank lines to put before and after      */
/* MESSAGE  - Text to write to the MSGDD                         */
/*****************************************************************/
saydd: module = 'SAYDD'
        if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
        parse arg sparms
        push trace() time('L') module 'From:' sigl 'Parms:' sparms
        call modtrace 'START' sigl
        parse arg msgdd msglines message
        if words(msgdd msglines message) < 3 then
           call rcexit 33 'Missing MSGDD or MSGLINES'
        if datatype(msglines) <> 'NUM' then
           call rcexit 34 'MSGLINES must be numeric'
/*****************************************************************/
```

```
/* If this is not background then bypass                          */
/******************************************************************/
        if tsoenv <> 'BACK' then
           do
            pull tracelvl . module . sigl . sparms
            call modtrace 'STOP' sigl
            interpret 'trace' tracelvl
            return
           end
/******************************************************************/
/* Confirm the MSGDD exists                                       */
/******************************************************************/
        call ddcheck msgdd
/******************************************************************/
/* If a number is provided, add that number of blank lines before */
/* the message                                                    */
/******************************************************************/
        msgb = 1
        if msglines > Ø then
           do msgb=1 to msglines
              msgline.msgb = ' '
           end
/******************************************************************/
/* If the linesize is too long break it into multiple lines and   */
/* create continuation records                                    */
/******************************************************************/
        msgm = msgb
        if length(message) > 6Ø & substr(message,1,2) <> '@@' then
           do
            messst = lastpos(' ',message,6Ø)
            messseg = substr(message,1,messst)
            msgline.msgm = date() time() strip(messseg)
            message = strip(delstr(message,1,messst))
            do while length(message) > Ø
               msgm = msgm + 1
               if length(message) > 55 then
                  messst = lastpos(' ',message,55)
               if messst > Ø then
                  messseg = substr(message,1,messst)
               else
                  messseg = substr(message,1,length(message))
               msgline.msgm = date() time() 'CONT:' strip(messseg)
               message = strip(delstr(message,1,length(messseg)))
            end
           end
        else
/******************************************************************/
/* Build print lines. Default strips and prefixes date and timestamp */
/* @BLANK - Blank line, no date and timestamp                     */
/* @      - No stripping, retains leading blanks                  */
/* @@     - No stripping, No date and timestamp                     */
/*******************************************************************/
           do
            select
               when message = '@BLANK@' then msgline.msgm = ' '
               when word(message,1) = '@' then
                    do
                     message = substr(message,2,length(message)-1)
                     msgline.msgm = date() time() message
                    end
               when substr(message,1,2) = '@@' then
                    do
                     message = substr(message,3,length(message)-2)
                     msgline.msgm = message
                    end
               otherwise msgline.msgm = date() time() strip(message)
            end
           end
/*******************************************************************/
/* If a number is provided, add that number of blank lines after     */
/* the message                                                       */
/*******************************************************************/
        if msglines > Ø then
           do msgt=1 to msglines
              msge = msgt + msgm
              msgline.msge = ' '
           end
/*******************************************************************/
/* Write the contents of the MSGLINE stem to the MSGDD               */
/*******************************************************************/
        call tsotrap "EXECIO * DISKW" msgdd "(STEM MSGLINE. FINIS"
        drop msgline. msgb msgt msge
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return
/*********** @REFRESH END   SAYDD    2ØØ4/Ø3/29 23:48:37 *************/
/*********** @REFRESH BEGIN JOBINFO  2ØØ4/11/23 22:11:25 *************/
/* JOBINFO  - Get job-related data from control blocks               */
/*-----------------------------------------------------------------*/
/* ITEM     - Optional item number desired, default is all           */
/*******************************************************************/
jobinfo: module = 'JOBINFO'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg item
/*******************************************************************/
/* Chase control blocks                                              */
```

```
/*******************************************************************/
          tcb      = ptr(540)
          ascb     = ptr(548)
          tiot     = ptr(tcb+12)
          jscb     = ptr(tcb+180)
          ssib     = ptr(jscb+316)
          asid     = c2d(stg(ascb+36,2))
          jobtype  = stg(ssib+12,3)
          jobnum   = strip(stg(ssib+15,5),'L',0)
          stepname = stg(tiot+8,8)
          procstep = stg(tiot+16,8)
          progname = stg(jscb+360,8)
          jobdata  = jobtype jobnum stepname procstep progname asid
/*******************************************************************/
/* Return job data                                                 */
/*******************************************************************/
          if item <> '' & (datatype(item,'W') = 1) then
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return word(jobdata,item)
             end
          else
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return jobdata
             end
/*********** @REFRESH END   JOBINFO  2004/11/23 22:11:25 *************/
/*********** @REFRESH BEGIN PTR      2002/07/13 15:45:36 *************/
/* PTR      - Pointer to a storage location                        */
/*-----------------------------------------------------------------*/
/* ARG(1)   - Storage Address                                      */
/*******************************************************************/
ptr: return c2d(storage(d2x(arg(1)),4))
/*********** @REFRESH END   PTR      2002/07/13 15:45:36 *************/
/*********** @REFRESH BEGIN STG      2002/07/13 15:49:12 *************/
/* STG      - Return the data from a storage location              */
/*-----------------------------------------------------------------*/
/* ARG(1)   - Location                                             */
/* ARG(2)   - Length                                               */
/*******************************************************************/
stg: return storage(d2x(arg(1)),arg(2))
/*********** @REFRESH END   STG      2002/07/13 15:49:12 *************/
/*********** @REFRESH BEGIN STACK    2004/05/18 09:25:09 *************/
/* STACK    - UNLOAD, RELOAD or LIST the Stack                     */
/*-----------------------------------------------------------------*/
/* OPTION   - UNLOAD, RELOAD or LIST                               */
```

```
/*********************************************************************/
stack: arg stackopt
/*********************************************************************/
/* Unload the parentage stack to avoid display problems             */
/*********************************************************************/
        if stackopt = 'UNLOAD' | stackopt = 'LIST' then
           do deq=1 to queued()
              pull stackinfo
              tempq.deq = stackinfo
           end
/*********************************************************************/
/* List the stack                                                    */
/*********************************************************************/
        if stackopt = 'LIST' then
           do req=deq-1 to 1 by -1
              say tempq.req
           end
/*********************************************************************/
/* Reload the parentage stack                                        */
/*********************************************************************/
        if stackopt = 'RELOAD' | stackopt = 'LIST' then
           do req=deq-1 to 1 by -1
              push tempq.req
           end
        return
/*********** @REFRESH END   STACK    2004/05/18 09:25:09 *************/
/*********** @REFRESH BEGIN BRWSDD   2002/09/11 01:05:08 *************/
/* BRWSDD   - Invoke ISPF Browse on any DD                           */
/*-------------------------------------------------------------------*/
/* BRWSDD   - Any DD to browse                                       */
/*********************************************************************/
brwsdd: module = 'BRWSDD'
         if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
         arg brwsdd
         if brwsdd = '' then call rcexit 90 'Browse DD missing'
         call ispwrap "LMINIT DATAID(DATAID) DDNAME("brwsdd")"
/*********************************************************************/
/* Browse the VIO dataset                                            */
/*********************************************************************/
         call ispwrap "BROWSE DATAID("dataid")"
/*********************************************************************/
/* FREE and DELETE the VIO dataset                                   */
/*********************************************************************/
         call ispwrap "LMFREE DATAID("dataid")"
         call tsotrap "FREE F("brwsdd")"
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return
/*********** @REFRESH END   BRWSDD   2002/09/11 01:05:08 *************/
/*********** @REFRESH BEGIN LOGGER   2004/09/27 14:00:29 *************/
/* LOGGER   - Append messages to a dynamic log                        */
/*--------------------------------------------------------------------*/
/* LOGTEXT  - The text to append to the log                           */
/**********************************************************************/
logger: module = 'LOGGER'
         if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
/**********************************************************************/
/* Log the supplied text                                              */
/**********************************************************************/
         parse arg logtext
         jobname = left(mvsvar('SYMDEF','JOBNAME'),8)
         logpfx = date('S') time() lpar jobname left(userid(),7)
         logger.1 = logpfx logtext
         logfile = userid()'.'execname'.LOG'
         call tsotrap "ALLOC F(LOGGER) DA('"logfile"') MOD REUSE",
                      "LRECL(200)"
         call tsotrap "EXECIO * DISKW LOGGER (STEM LOGGER. FINIS"
         call tsotrap "FREE F(LOGGER)"
/**********************************************************************/
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return
/*********** @REFRESH END   LOGGER   2004/09/27 14:00:29 *************/
/*********** @REFRESH BEGIN CRYPT    2004/09/29 16:44:54 *************/
/* CRYPT    - Encryption/Decryption routine                           */
/*--------------------------------------------------------------------*/
/* STRING   - String to encrypt or decrypt                            */
/**********************************************************************/
crypt: procedure expose probe sigl modtrace
        module = 'CRYPT'
        if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
        parse arg sparms
        push trace() time('L') module 'From:' sigl 'Parms:' sparms
        call modtrace 'START' sigl
/**********************************************************************/
/* Encryption/decryption                                              */
/**********************************************************************/
        parse arg string
        chars = xrange('00'x,'FF'x)
        do s=length(string) to 1 by -1
           univ = ''
           do c=1 to length(chars)
```

```
             if datatype(substr(chars,c,1),'S') = 1 then
                univ = univ||substr(chars,c,1)
          end
          b = random(1,length(univ),s)
          univ = substr(univ,b,length(univ)-b+1)||substr(univ,1,b-1)
          new = translate(substr(string,s,1),reverse(univ),univ)
          string = overlay(new,string,s)
       end
/*****************************************************************/
       pull tracelvl . module . sigl . sparms
       call modtrace 'STOP' sigl
       interpret 'trace' tracelvl
       return string
/*********** @REFRESH END   CRYPT    2004/09/29 16:44:54 *************/
/*********** @REFRESH BEGIN MODTRACE 2003/12/31 21:56:54 *************/
/* MODTRACE - Module Trace                                          */
/*------------------------------------------------------------------*/
/* TRACETYP - Type of trace entry                                   */
/* SIGLINE  - The line number called from                           */
/*****************************************************************/
modtrace: if modtrace = 'NO' then return
          arg tracetyp sigline
          tracetyp = left(tracetyp,5)
          sigline = left(sigline,5)
/*****************************************************************/
/* Adjust MODSPACE for START                                        */
/*****************************************************************/
          if tracetyp = 'START' then
             modspace = substr(modspace,1,length(modspace)+1)
/*****************************************************************/
/* Set the trace entry                                              */
/*****************************************************************/
          traceline = modspace time('L') tracetyp module sigline sparms
/*****************************************************************/
/* Adjust MODSPACE for STOP                                         */
/*****************************************************************/
          if tracetyp = 'STOP' then
             modspace = substr(modspace,1,length(modspace)-1)
/*****************************************************************/
/* Determine where to write the traceline                           */
/*****************************************************************/
          if ispfenv = 'YES' & tsoenv = 'FORE' then
/*****************************************************************/
/* Write to the ISPF Log, do not use ISPWRAP here                   */
/*****************************************************************/
             do
              zedlmsg = traceline
              address ISPEXEC "LOG MSG(ISRZ000)"
             end
          else
```

```
              say traceline
/*********************************************************************/
/* SAY to SYSTSPRT                                                   */
/*********************************************************************/
           return
/*********** @REFRESH END   MODTRACE 2003/12/31 21:56:54 *************/
```

Robert Zenuk
Systems Programmer (USA)

RTM with TN3270(E) servers

Ever since the advent of 3270 terminals in the early 1970s, the response times experienced by end users accessing real-time interactive applications represented the unmistakable health of mainframe-based networks. In the early days of mainframe networking, when BSC held sway and network management was, at best, embryonic, response times became the pulse of a network. In the absence of other incisive monitoring tools, response times were an infallible indicator of overall network health. Changes in response times, especially sudden degradations, were invariably a portent that something was awry. As interactive, transaction-processing, mission-critical applications gained popularity, employee productivity and even the sacrosanct corporate bottom line became inextricably linked with response times. By the 1980s, poor response times, which sapped productivity and impacted transaction volumes, were being explicitly cited as a lost opportunity cost element *vis-à-vis* mission-critical operations.

Unbeknown to most, in the mid-1970s, a time when terminal usage was still a relative novelty, IBM did some seminal research into the end-user psychology of response times. A key finding, which was constantly drummed into those of us working on 3270 projects, was the importance of consistency. End users, especially the so-called 'heads-down' users, fall into a work rhythm dictated by the response times that they are experiencing. While they would subconsciously (which today would be referred to as autonomically) compensate for minor variations, any major changes (eg of 1 second more) would result in alterations to work patterns and even habits. The research found that response times became a kind of metronome for office users. Obviously the issue here, yet again, was that of maximizing productivity.

The optional IBM 3174 Response Time Monitor (RTM) feature that was made available around 1984 (along with extensions to the SNA Management Services request/response repertoire to support this feature) was a breakthrough, not just for accurate response time measurement but also for overall mainframe network management. Now for the first time it was possible to measure actual end user response times – accurately, unambiguously, and, above all, consistently. The 3174 RTM measured actual, round-trip response times – albeit collectively (as opposed to individually) for all the 'dumb' terminals attached to a given 3174. But at a time (just prior to LANs) when terminals were coax-attached and PCs were nascent, this was acceptable and adequate.

TN3270(E) RTM IS NOT THE SAME AS 3174 RTM

Though the term 'RTM' is still widely bandied about today in the context of TCP/SNA networks (or even predominantly IP networks), it is important to always remember that today's RTM measurements are not the same as what was measured by 3174s. In general this is not an issue, especially when one realizes that what one is always looking for in terms of response times is fluctuations. Thus, as long as you have a base-line and you measure fluctuations relative to that, it doesn't in the end really matter whether today's RTM cannot even come close to emulating the true end-to-end measurements recorded by the 3174. Today's network topologies, TN server-based mainframe access, and the ubiquitous use of multitasking PCs make it extremely difficult, if not actually impossible, to come even close to mimicking the 3174 RTM methodology.

With coax-attached, ‘dumb’ (ie no real processing done at the ‘head’), 3270 terminals, where all the keystroke handling and datastream decoding was done at the 3*x*74 control unit, accurately keeping track of exact round-trip response times was easy. For each 3270 transaction, the 3174 RTM, in essence, measured the time between when the keyboard was locked (on the entry of an AID-generating key, eg *Enter*) and when the keyboard was unlocked by a Write Control Word (WCC) in the incoming 3270 datastream. This, from the perspective of a 3270 end user, really was the measure of what they perceived as their response time – and also corresponded to the appearance and disappearance of the little ‘transaction being processed’ clock icon on the 3270 status bar.

Tn3270(E) RTM does not measure its response times using this approach. The mechanics of the tn3270(E) RTM feature are actually spelled out as a part of the overall tn3270(E) standard – with the relevant standard being RFC 2562. RFC 2562 was put forward by IBM in April 1999, entitled *Definitions of Protocol and Managed Objects for TN3270E Response Time Collection Using SMIv2 (TN3270E-RT-MIB)*. Given that it is from IBM, to its credit, it actually starts off by quite emphatically spelling out the differences in response time collection methodology. Unfortunately, with the exception of the actual developers, most others that deal with tn3270(E) RTM have rarely had the time to look at what IBM clearly points out within this RFC.

THE TWO APPROACHES IN RFC 2562

The two approaches for measuring response times in a TCP/SNA network, per the RFC, are:

1. The SNA MS-based RTM method as implemented on the 3174.
2. Timestamping using definite response flows.

Tn3270(E) servers that comply to RFC 2562, with IBM’s mainframe-resident Communication Server implementation being key among these, will always use the latter approach; ie the definite response timestamping at the tn3270(E) server. The first thing to note here is that this timestamping, the basis for tn RTM calculations, is performed at the interface between the IP segment of the network and the start of what is the SNA component of the network. Given that there are now two very distinct networks involved – the TCP/IP network and the SNA network – this timestamping scheme does enable one to cleanly split out IP network transit times versus transit times within the SNA network.

Figure 1 (where DR stands for definite response requested) shows how and when the timestamps are taken by the tn3270(E) server, with each timestamp identified by the letters D, E, and F.

At this juncture it is imperative to clarify, particularly to those who grew up with SNA, what ‘request’, ‘reply’, and ‘response’ mean in the RFC 2562 context – and hence in Figure 1 as well

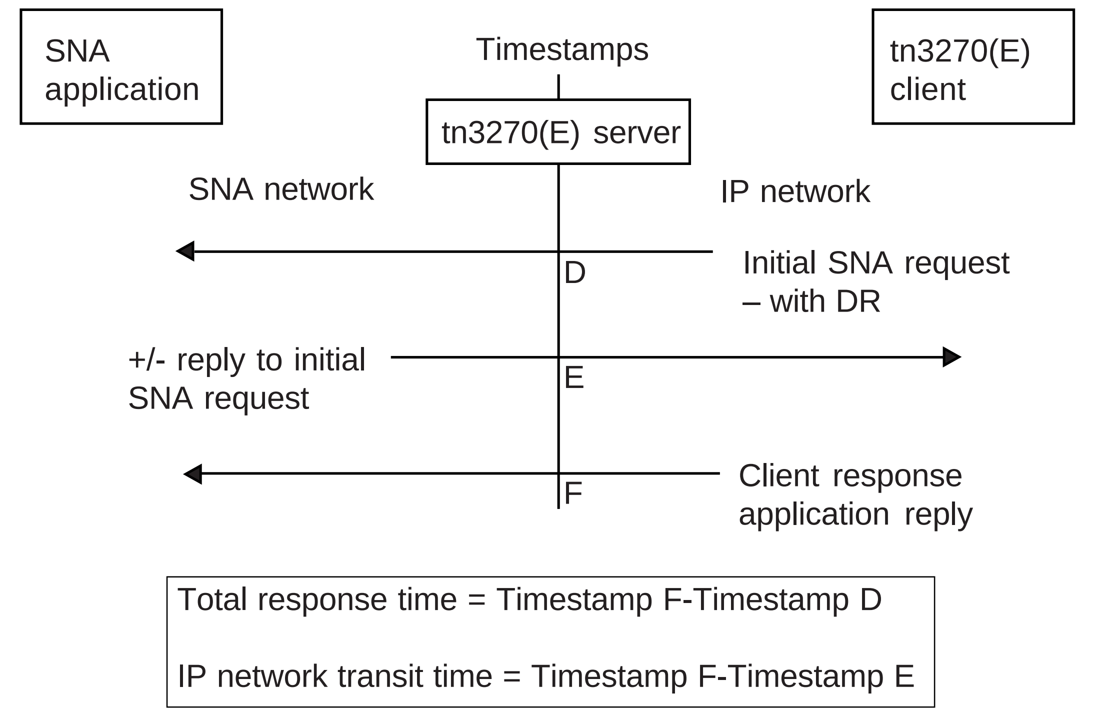
SNA application
Timestamps
tn3270(E) client
tn3270(E) server
SNA network
IP network
D
Initial SNA request – with DR
+/- reply to initial SNA request
E
F
Client response application reply
Total response time = Timestamp F-Timestamp D
IP network transit time = Timestamp F-Timestamp E

Figure 1: Timestamps

as with actual tn3270 RTM implementations. The message unit (or segment) generated when a tn user hits PF8 during a TSO session, to scroll down the screen, corresponds to the initial SNA request. TSO's 'answer' to this request is what is considered to be the 'reply' to the initial SNA request. The final 'response' is the tn client's positive or negative response to TSO's reply. Obviously this is different from a straight SNA approach and moreover requires that the tn clients conform to the RFC 2355 *Telnet 3270 Enhancements* standard so it is able to successfully negotiate the RESPONSES (or timing-mark) functions.

Though not the same as 3174 RTM, the tn3270(E) RTM is now the only real standards-based option available to us. Once we know what it measures, in terms of those timestamps, we can correlate that to the overall 'SNA' data flows between the desktop client and the mainframe SNA applications. This enables us to establish the all important base-line and furthermore split out the SNA transit times from those imposed by the IP network. After that, what we really are trying to monitor is variations from this baseline.

BOTTOM LINE

In the case of today's TCP/SNA networks, not having an incisive mainframe IP monitor means that you are flying blind most of the time when it comes to overall system visibility and awareness. Within that context, having a mainframe IP monitor that supports RTM and complies with RFC 2562 is akin to having radar. 3270 RTM, as has always been the case, invariably permits you to see potential problems 'beyond the horizon' and take evasive action before they cause service disruption. With 'zero-downtime' operation and stringent Service Level Agreements (SLAs) being the norm with most TCP/SNA networks, tn3270(E) RTM is an invaluable management tool. The fact that it uses a different methodology from that pioneered by the 3174 does not, however, mean that it is less valuable or less effective. As IBM discovered nearly

30 years ago, the key thing that end users expect with 3270 response times is consistency. The RFC 2562 scheme, once we work out what it does, definitely allows us to monitor response time consistency and detect fluctuations. Tn3270(E) RTM should thus be a vital tool in your TCP/SNA network management toolkit.

Anura Gurugé
Strategic Consultant (USA)

William Data Systems has announced Version 4.1 of EXIGENCE, its network problem determination tool for z/OS.

The product eliminates the complexity and overhead of defining, capturing, and analysing IP and SNA traces. Traces captured by EXIGENCE are formatted before being presented to users and any exception conditions are highlighted. EXIGENCE also presents an explanation of any error, its most likely cause, and a suggested solution.

Enhancements include: group trace (providing simplified tracing across the sysplex); a Java-based client; and full support for TCP/IP Version 6.

For further information contact:
URL: www.willdata.com/v2/company/news-050311.htm.

* * *

NetManage has announced Version 7.2 of OnWeb, which transforms host processes, business logic, and data into reusable components, such as Web services, .NET Assemblies, JavaBeans, Enterprise JavaBeans (EJBs), and portlets, which can be integrated with other enterprise applications or be used as components in a new application. OnWeb applications can be presented in a wide range of formats and accessible from Web browsers, PDAs, and other mobile devices.

New features include: improved SSL and SSH support, improved integration capabilities to ERP and CRM systems, databases, middleware and Microsoft BizTalk 2004; and enhanced monitoring and reporting capabilities. The product now runs on AIX and Linux (on Intel) systems.

For further information contact:
URL: www.netmanage.com/products/onweb/index.asp.

* * *

William Data Systems has also announced Version 3.3 of IMPLEX, its real-time IP network monitor for z/OS.

This version provides a new, browser-based, client interface that removes the need for customers to install additional software on their desktop. Being a two-tier client means that a separate Web server is not required.

The product also has the advantage of not impacting on bandwidth because the XML data is no greater than the equivalent 3270 datastream, and has no processor overhead because all graphical formatting is performed in the browser.

For further information contact:
URL: www.willdata.com/v2/company/news-150405.htm.

* * *

Tsarfin Computing has announced Version 5.4 of IPMonitor, its network monitoring software.

The product allows network administrators to monitor any networked device on the Internet, corporate intranet, or TCP/IP LAN and receive alerts immediately via audible alarm, message, e-mail, or third-party software when a connection fails.

For further information contact:
URL: ipmonitor.tsarfin.com/.

* * *