IBM Software Group

Delivering e-business access to CICS
IP CICS Sockets

Kenneth M Porter

IBM z/OS Communications Server Development
kmporter@us.ibm.com

ON DEMAND BUSINESS™

IP CICS Sockets Abstract

- **Business Needs**
- **Components**
- **Programming**
- **Configuration and Setup**
- **Recommendations**
- **Summary**

During this presentation we will explore the reasons why successful enterprises architect e-business solutions to leverage the value of their CICS applications, the choices available to access CICS from the many different e-business clients, and the merits of one choice over another. Customers frequently ask which access method is 'best' to access CICS. Whilst there is no one answer, we will discuss the environmental, functional and non-functional factors that have a bearing on the answer to these questions.

Over the past 35 years, part of the success of CICS has been due to the fact it does supports many types of clients, and in recent years the support of a choice of protocols, interfaces and APIs to connect to and from CICS servers. SOAP, JCA, Java RMI, WebSphere MQ, HTTP, TCP/IP sockets… the choice seems endless. However, choice can be overwhelming, and not making the right choice can lead to over-taxing demands on clients, less than optimised networks and systems, and reduced flexibility for future access and re-use.

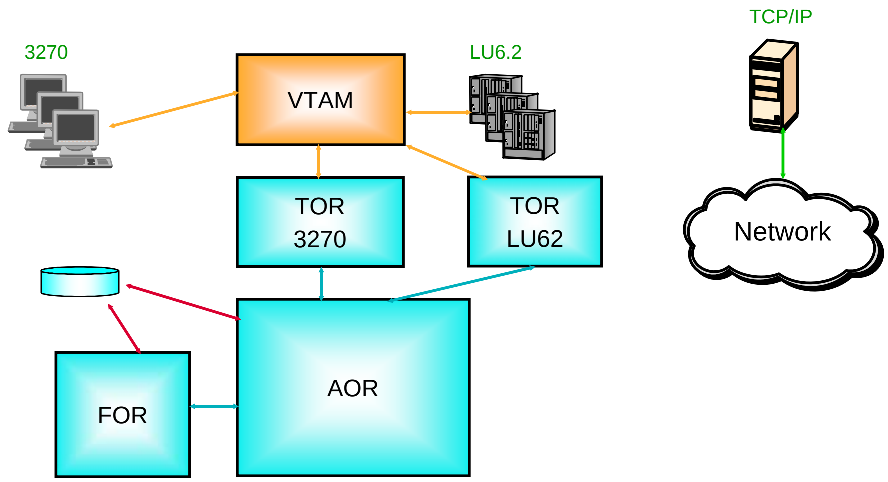
This presentation aims to outline which of these choices are industry best practice – ie. what are the strategic options?

There are a number of redbooks and whitepapers referred to at the end of this presentation which will help you explorer this topic in more detail.

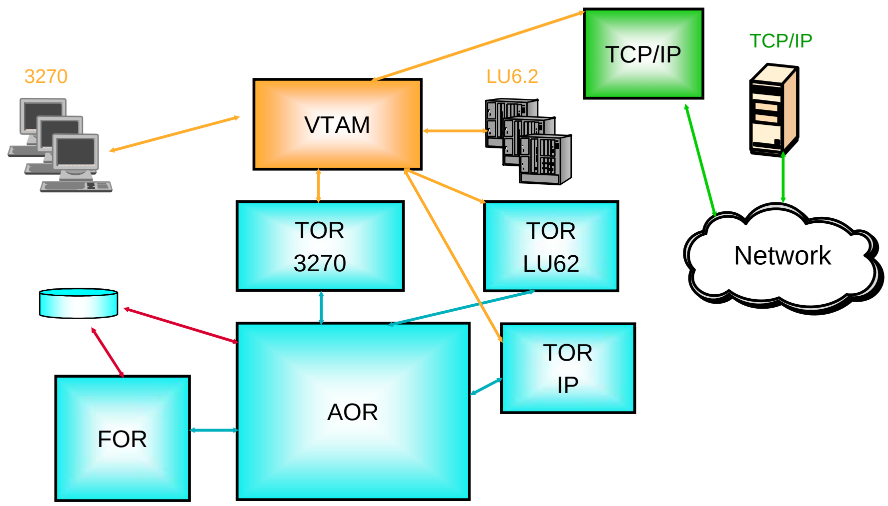
So, lets get started. Moving to slide 2….

Business Needs

The Need for TCP based connections



The Need for TCP based connections



Components

The components supporting IP CICS Sockets based applications

- ✓ **VTAM**
- ✓ **TCP/IP**
- ✓ **CICS/TS**
- ✓ **IP CICS Sockets**
- ✓ **VSAM**
- ✓ **Compiler/Assembler/Linkedit**
- ✓ **CICS programming skills**
- ✓ **Socket programming skills**

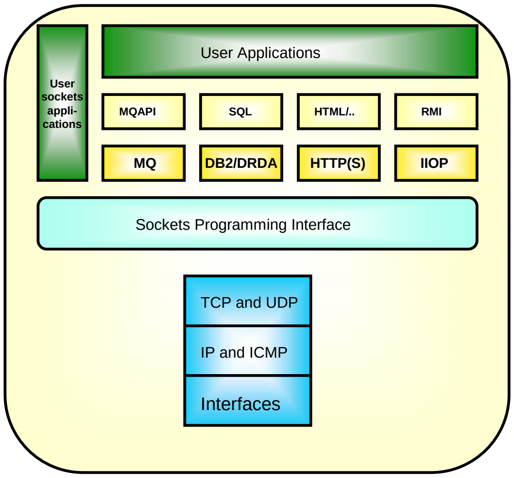
Components

What is IP CICS Sockets ?

- **IP CICS Sockets is a component of the Communications Server for OS/390 and z/OS, not CICS/TS itself.**
- **It is a general-purpose sockets programming API to be used by CICS application programmers for implementing native (low-level) sockets communication in OS/390 and z/OS CICS transaction programs.**
- **None of the CICS TS TCP/IP related services (CWI, IIOP, ECI, etc.) are based on CICS Sockets. They all use specialized implementations that are based on the native TCP/IP sockets APIs or UNIX System Services sockets.**

Components

Applications and their relationship to the sockets programming interfaces

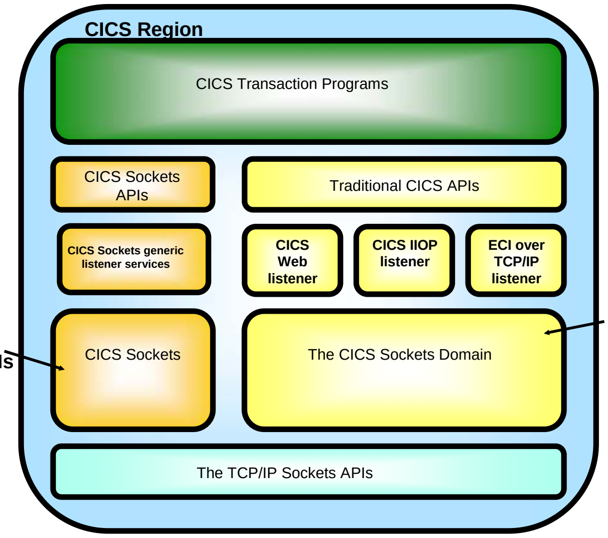


- The higher in the protocol stack the user application is located, the more abstract the programming interfaces become and the more services are done by the protocol layers on behalf of the application.
- The price is loss of detailed control over the communication channel, which for the majority of applications is of no concern.

- The sockets programming interface is a rather low-level and sometimes complex programming interface where the user's application program must handle many tedious details, such as converting data between ASCII and EBCDIC on z/OS.
- The advantage of using the socket API directly is the opportunity for developing high-performing applications with a very low software overhead.

Components

TCP/IP Communication into the CICS region

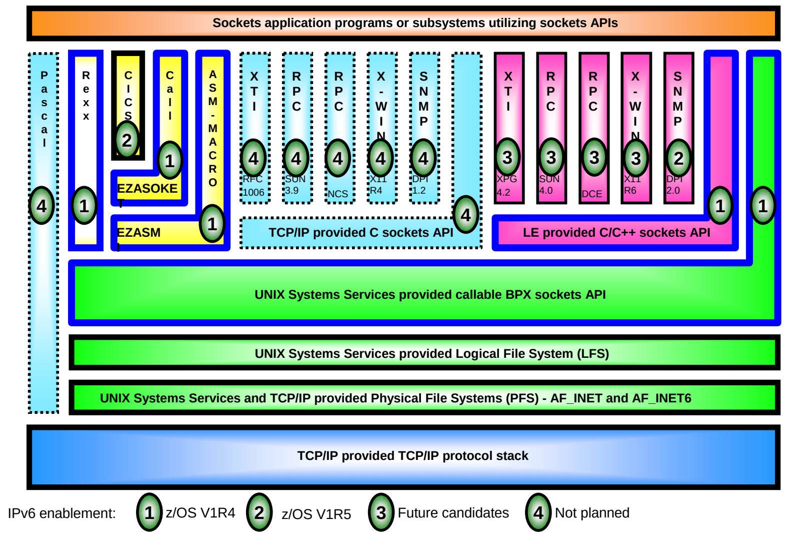


These services are based on the Sockets Extended sockets APIs (provided by Communications server)

The listeners are the 'servers' as seen from a TCP/IP perspective.

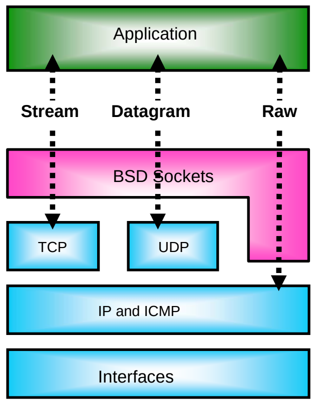
These services are based on the UNIX System Services C/C++ sockets API (provided by Language Environment) and the UNIX System Services callable APIs

z/OS sockets API overview



Socket types and their characteristics

An application programmer can use any of three types of sockets: a stream socket (the most widely used socket type), a datagram socket, and a raw socket.



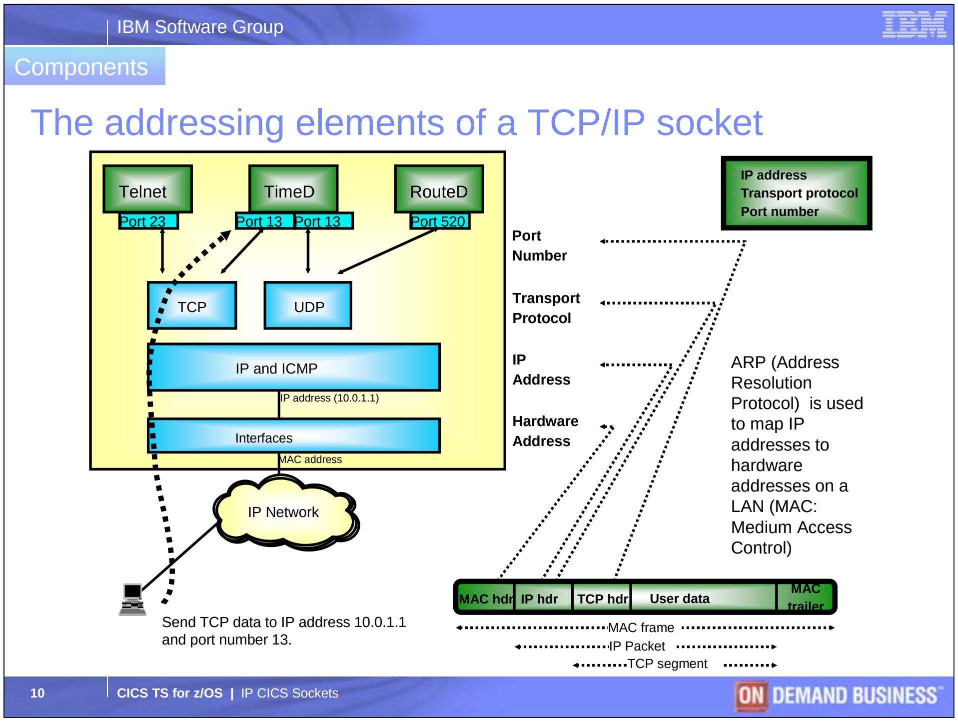
	STREAM	DATA GRAM	RAW
Reliable	Yes (connection oriented)	Application responsibility	Application responsibility
Data size	Large amounts	Fixed datagram size	Fixed network packet size
TCP/IP protocol interface	***TCP***	***UDP***	***IP***

Stream socket characteristics:
Connection oriented, full duplex, reliability is part of transport protocol layer, byte stream without transport protocol imposed boundaries, and flow control.

Datagram socket characteristics:
Connectionless, no transport protocol reliability - if required it must be part of the application protocol, no flow control, and a datagram has a maximum size.

Raw socket characteristics:
Direct application access to network layer protocols. Other characteristics resemble datagram sockets.

Most sockets applications use stream sockets (the TCP transport protocol in a TCP/IP network environment).



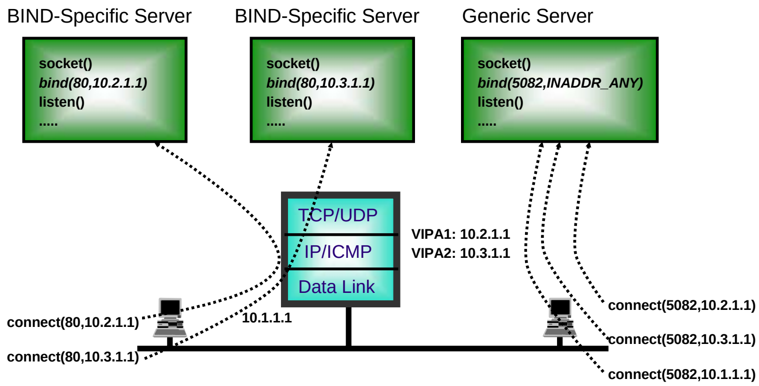
An incoming IP packet is passed from the networking layer to either the TCP or UDP protocol layer, or to a raw socket or to a part of the TCP/IP protocol stack that handles a given protocol, depending on the PROTOCOL field in the IP header. In the TCP or UDP layer, the correct socket to deliver input to is selected based on the TCP or UDP header PORT field, which holds the destination port number.

An application may use the same port number for both a stream socket and a datagram socket.

One application may have many sockets open, and each socket may be set up to serve different port numbers, if required.

Both server and client program sockets have port numbers. The server's port number is generally pre-defined (most often referred to as well-known port numbers) so client programs know how to contact a given server program by server IP address, server port number and protocol (typically TCP or UDP). Client programs are often assigned a port number by their transport protocol layer (a so-called ephemeral or short-lived port number that only has meaning while the client program is active).

Bind-specific or Generic servers



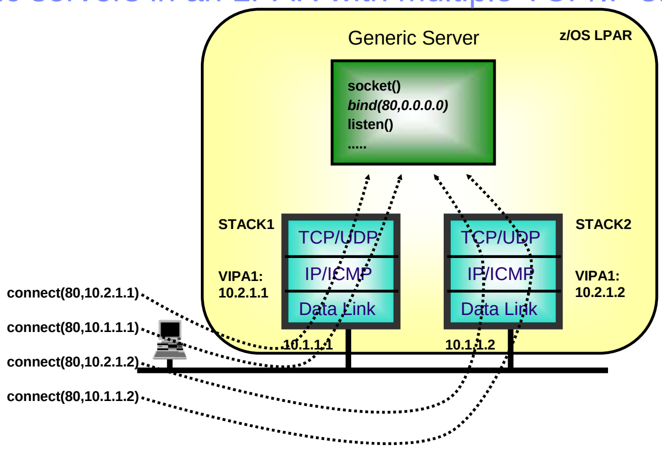
If you try to start two or more servers that both bind to the same port number on the same IP address (incl. INADDR_ANY), only the first server will succeed in starting up, the others will end with address_in_use errors. Multiple servers can bind to the same port number, if they bind to different specific HOME IP addresses.

Bind-specific or Generic servers - Notes

- **A generic server (also known as an INADDR_ANY server) will accept requests for all the HOME IP addresses of the stack, in this case: 10.1.1.1, 10.2.1.1, and 10.3.1.1**
- **If the HOME list is dynamically extended through dynamic configuration changes, the server will begin accepting requests for the added IP address in addition to those that were active when the server was started.**
- **INADDR_ANY is a symbol for the IPv4 address 0.0.0.0**
- **If a server binds its listening socket to a specific IP address (that must be in the HOME list of the TCP/IP stack that this server application interfaces to), then that server will only receive connection requests that arrive for that specific IP address and none of the other IP addresses in the HOME list.**
- **If you need to start more servers on the same standard port number, such as port 80, you can do so by making each of the servers bind-specific to their individual IP addresses - typically virtual IP addresses (VIPA).**

Components

Generic servers in an LPAR with multiple TCP/IP stacks



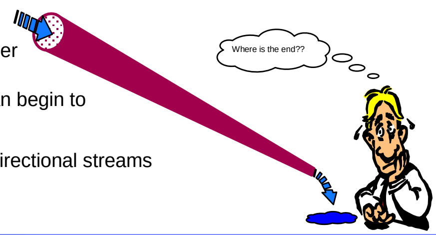
- If a server is generic, then it will accept incoming connections to any IP address in the HOME list of any TCP/IP stack that is active in that LPAR.
- Sometimes that is desirable, and sometimes it isn't
  - Consider that the reason for two stacks most likely is separation of different network security zones, and you most likely do not want users from both security zones accessing the same services

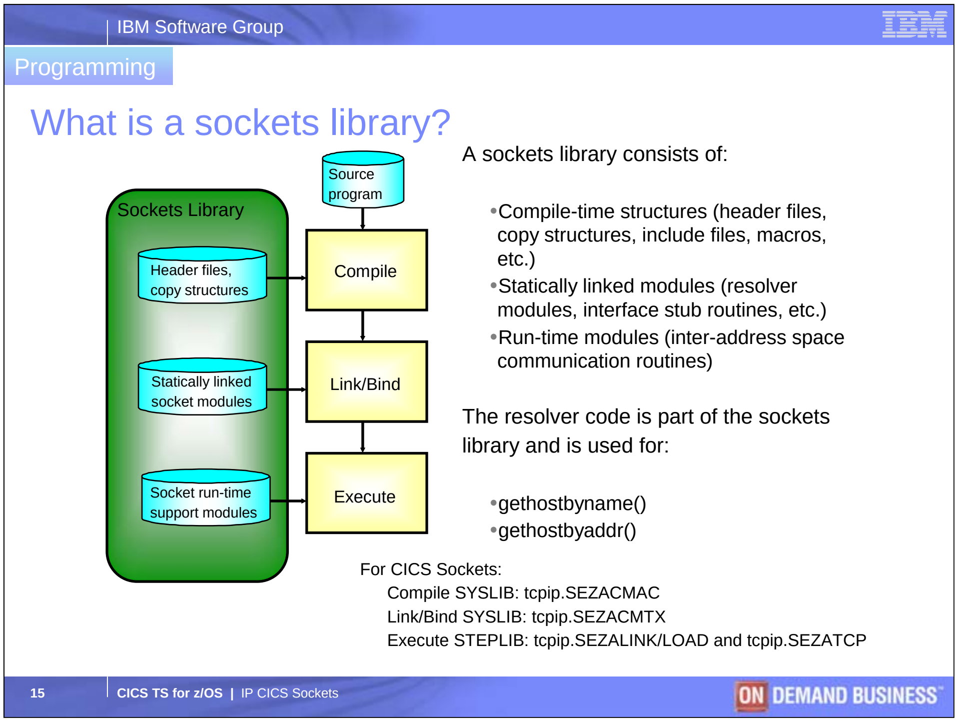
ON DEMAND BUSINESS™

The nitty-gritty of sockets programming

The sockets API is a relatively low-level API that leaves many details to be handled by the application programmer:

- Clients must have knowledge about the server's location before they can contact the server. There is no discovery mechanism at the sockets API layer.
- Socket programs must deal with different data formats and include logic to do any necessary data conversion, such as ASCII / EBCDIC translation, little Endian / big Endian fixed integer conversions, IEEE floating point / proprietary floating point format conversions, etc.
- User authentication (the plain standard sockets API does not include any functions to authenticate end users before these are given access to a server program). Secure sockets layer (SSL) may optionally authenticate a client and/or a server during connection establishment. SSL bases authentication on X.509 certificates.
- Transport layer buffer utilization may affect
- certain aspects of the API. A read() or a write()
- may not always read or write the expected number of bytes.
- Servers must be started manually before they can begin to
- serve clients.
- Stream sockets transport data in continuous bi-directional streams
- without any notion of record boundaries.





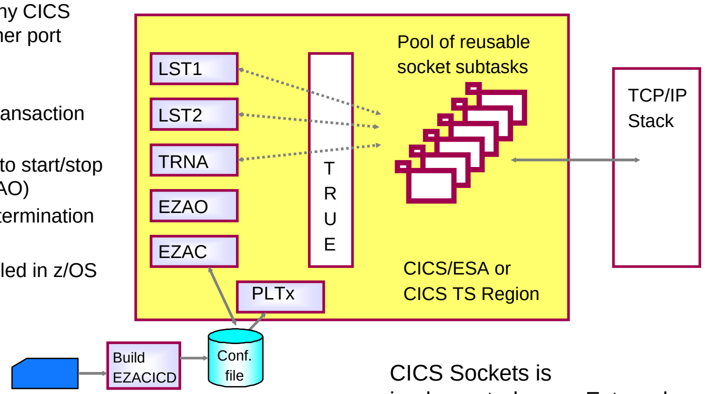
When we create a sockets program, we use something which generally is called a *sockets library*. A sockets library consists of both compile-time structures, statically linked support modules, and run-time support modules.

OS/390 uses a so-called *stub-resolver* approach, where the resolver code is part of the sockets library and is either statically linked into the application or is invoked as run-time modules as part of the application process.

The resolver is used by certain functions in the socket library, which are known as resolver calls, such as *gethostbyname* (which is used to translate aTCP/IP host name to one or more IP addresses). The resolver library routines are generally considered part of a sockets library, but they are not basic socket functions in the sense that a call to such a routine generates a single interaction to the TCP or UDP transport functions. A call to a resolver function may generate no calls to the transport protocol layers, or it may generate a series of calls to the transport protocol layer. A call to gethostbyname() may result in basic sockets calls to socket(), send(), receive(), and close() for communication with a name server.

IP CICS sockets structural overview

- Multiple listeners – each instance separately configurable
- Multiple listeners in many CICS regions can share listener port number
- User ID security
- Configuration file and transaction (EZAC)
- Operations transaction to start/stop individual listeners (EZAO)
- PLT-enabled start and termination
- Reusable subtasks
- IPv6 (AF_INET6) enabled in z/OS V1R5



CICS Sockets is implemented as an External Resource Manager in CICS.

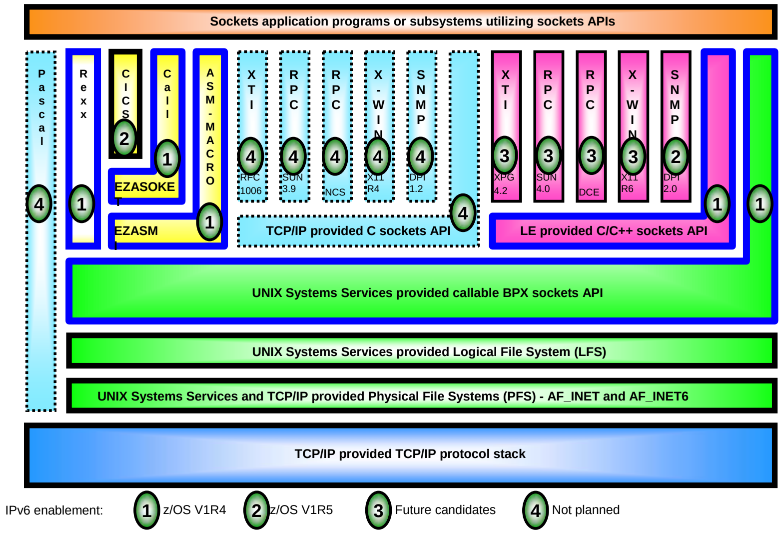
Functional enhancements: configuration file with info on multiple listeners per CICS image. Each listener can be controlled via port number, queue length, ASCII/EBCDIC translation, security exit, timeouts. Security exit may use CICS user ID security instead of terminal-related security (requires CICS V4 and write own EZACICSE security exit routine, that returns the user ID to start the transaction under). Configuration file may be built by batch utility and/or maintained with the EZAC transaction. Start/Stop can be done via CICS PLT processing - incl. enabling TRUE and starting listener transactions. Individual listeners can be stopped/started via the EZAO transaction. Special gethostbyname() module (EZACIC25) should be used by CICS transactions. A DNS cache file is maintained (may be a CICS data table). Initial content can be loaded by batch utility.

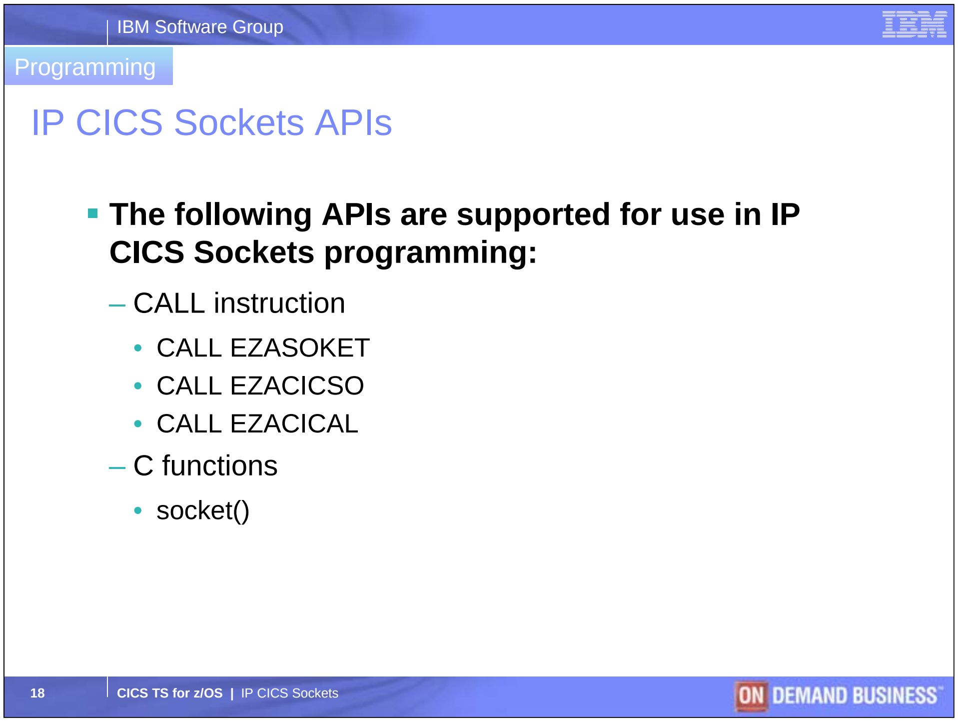
Performance enhancements: Subtasks are not started/stopped per transactions; but are started when the CICS socket environment initializes. A configuration parameter specifies the number of reuseabale subtasks to start (excl. listeners - they each have a permanent subtask). Parallel listeners allow a higher number of transaction initiations. The DNS cache improves performance for frequently resolved names. Subtasks use HPNS and not IUCV for communication with the TCP/IP V3R2 stack.

The enhanced CICS socket was made available late June 97.

CICS sockets registers with WLM during initialization.

z/OS sockets API overview





The Macro API is implemented by the use of the EZASMI assembler macro.

The CALL Instruction (CALL) API is implemented by calling EZASOKET.

Notes:
1 - Implemented internally.
See *z/OS V1R4.0 Communication Server: IP Application Programming Interface Guide* for details on Macro, CALL and REXX APIs.

Programming

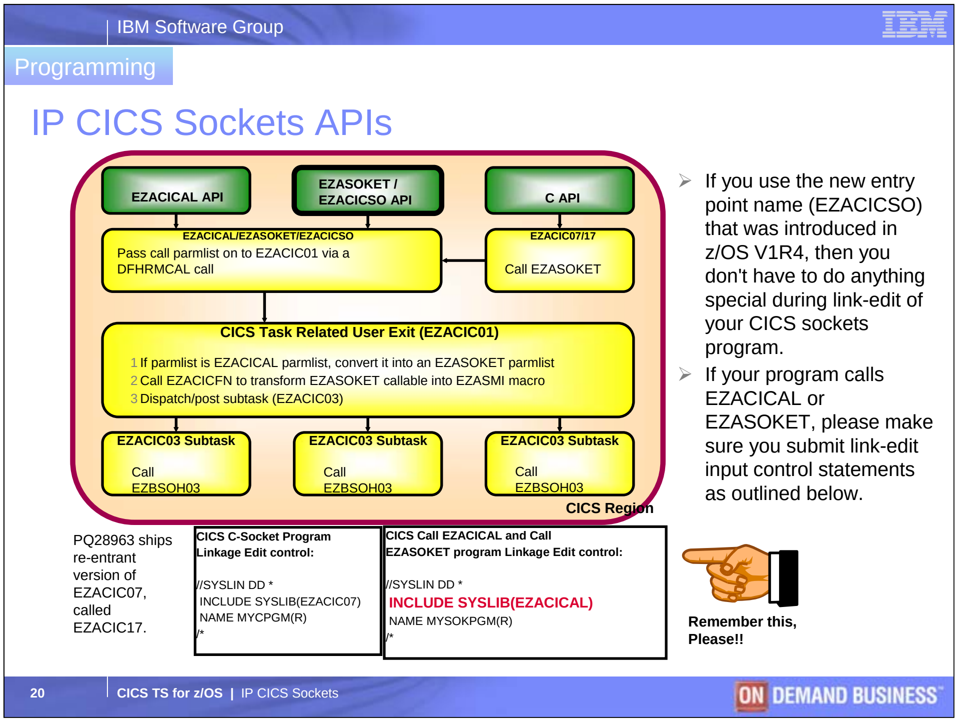
IP CICS Sockets API commands

- **The following API commands are supported for use in IP CICS Sockets programming:**
  - ACCEPT, BIND, CLOSE, CONNECT, FCNTL, GETCLIENTID, GETHOSTBYADDR, GETHOSTBYNAME, GETHOSTID, GETPEERNAME, GETSOCKNAME, GETSOCKOPT, GIVESOCKET, INITAPI, INITAPIX, IOCTL, LISTEN, READ, READV, RECVFROM, RECVMSG, SELECT, SELECTEX, SEND, SENDMSG, SENDTO, SETSOCKOPT, SHUTDOWN, SOCKET, TAKESOCKET, TERMAPI, WRITE, WRITEV
- **The following API functions are supported for use in IP CICS C Sockets programming:**
  - accept(), bind(), close(), connect(), fcntl(), getclientid(), gethostbyaddr(), gethostbyname(), gethostid(), gethostname(), getpeername(), getsockname(), getsockopt(), givesocket(), initapi(), ioctl(), listen(), read(), recv(), recvfrom(), select(), send(), sendto(), setsockopt(), shutdown(), socket(), takesocket(), write()

The Macro API is implemented by the use of the EZASMI assembler macro.

The CALL Instruction (CALL) API is implemented by calling EZASOKET.

Notes:
1 - Implemented internally.
See *z/OS V1R4.0 Communication Server: IP Application Programming Interface Guide* for details on Macro, CALL and REXX APIs.



Both CICS C-sockets and Call EZACICAL socket programs are transformed into calls to the sockets extended callable API before the socket calls are passed down to the socket communicating subtasks, making the full CICS socket implementation much more streamlined. The subtasks now only have to do call routing on behalf of the CICS task.

Really, EZACICAL calls are transformed directly into EZASMI macro calls by EZACIC01, there's not a transform to EZASOKET first. (According to Bill Kelsey, Oct 2001).

A CICS task may use sockets extended callable sockets, including assembler callable sockets; but not the sockets extended assembler macro API.

There is no change in the linkageedit control statements from V3R1 to V3R2 - for a CICS C-socket program you still need to include EZACIC07, and for both sockets extended and EZACICAL callable programs, you need to include the EZACICAL module (the EZACICAL module includes both an EZACICAL and an EZASOKET entry point for CICS sockets).

There are no changes in the definition of CICS sockets to CICS.

The CICS Listener has been changed to using sockets extended calls only.

There are no additions to the CICS C-socket functions. The sockets extended callable API in CICS has been extended with readv(), recvmsg(), selectex(), sendmsg(), and writev(). - getibmopt() and setibmopt() are not supported by CICS sockets.

The implicit initapi() or explicit with TCPNAME=space is supported in CICS and a search in TCPIP.DATA will be performed by the socket subtasks (not the CICS task!).

Gethostbyname() (Sockets extended only) in CICS works with a name server, but not with a local hosts file.

Establishing stack-affinity

- Affinity to one of more stacks in an LPAR can be established at an address-space level and will be in effect for all TCP/IP access from that address space.
- The simplest way to establish stack affinity is to add a small job step to your CICS start up procedure:

```
//DFHSTART PROC START='INITIAL',
// INDEX1='USER1.CICS130',
// INDEX2='CICSTS.V1R3M0.CICS',
// REGNAM='',
// REG='64M',
// DUMPTR='YES',
// RUNCICS='YES',
// OUTC='*',
// JVMMEMBR='DFHJVM',
// SIP=1
//*
//* Set affinity to TCPCS stack on mvs098
//*
//AFFINITY EXEC PGM=BPXTCAFF,PARM='TCPCS'
//*
........
//CICS    EXEC PGM=DFHSIP,REGION=&REG,TIME=1440,
// COND=(1,NE,CICSCNTL),
// PARM='START=&START,SYSIN'
```

- All sockets activity from this address space will only use the TCPCS TCP/IP stack on this LPAR

Setting up IP CICS Sockets

1 Modify CICS JCL
  a Add TCPIP.SEZALINK/LOAD to STEPLIB
  b Add TCPIP.SEZATCP to DFHRPL
  c Add SYSTCPD DD statement pointing to your TCP/IP resolver data set
  d Add TCPDATA DD statement for messages from CICS sockets

2 Define CICS resources
  e Add TCPDATA DCT entries
  f Add PLT entries for automatic start/stop of CICS sockets
  g Run a DFHCSDUP job with TCPIP.SEZAINST(EZACICCT) as input

3 Reserve port(s) in TCP/IP for listener(s)
  h Add a PORT reservation statement for your CICS listener(s)

4 Build CICS Sockets configuration data set
  i Create a batch job to define the IP CICS Sockets configuration and load to a VSAM data set as EZACONFG
  j Start CICS and use the EZAC transaction to update the configuration data set

Refer to *IP CICS Sockets Guide*, SC31-8518, Chapter 2 for details (easy step-by-step guide).

CICS entry in IP CICS Sockets configuration file - EZAC trans

```
EZAC,DISplay,CICS                                            APPLID = DBDCCICS

APPLID      ===> DBDCCICS            APPLID of CICS System
TCPADDR     ===> TCPCS               Name of TCP Address Space
NTASKS      ===> 100                 Number of Reusable Tasks
DPRTY       ===> 010                 DPRTY value for ATTACH
CACHMIN     ===> 010                 Minimum Refresh Time for Cache
CACHMAX     ===> 020                 Maximum Refresh Time for Cache
CACHRES     ===> 005                 Maximum number of Resolvers
ERRORTD     ===> CSMT                TD Queue for Error Messages
SMSGSUP     ===> NO                  Suppress Task Started Messages
TERMLIM     ===> 000                 Subtask Termination Limit



PF                  3 END                                 12 CNCL
```

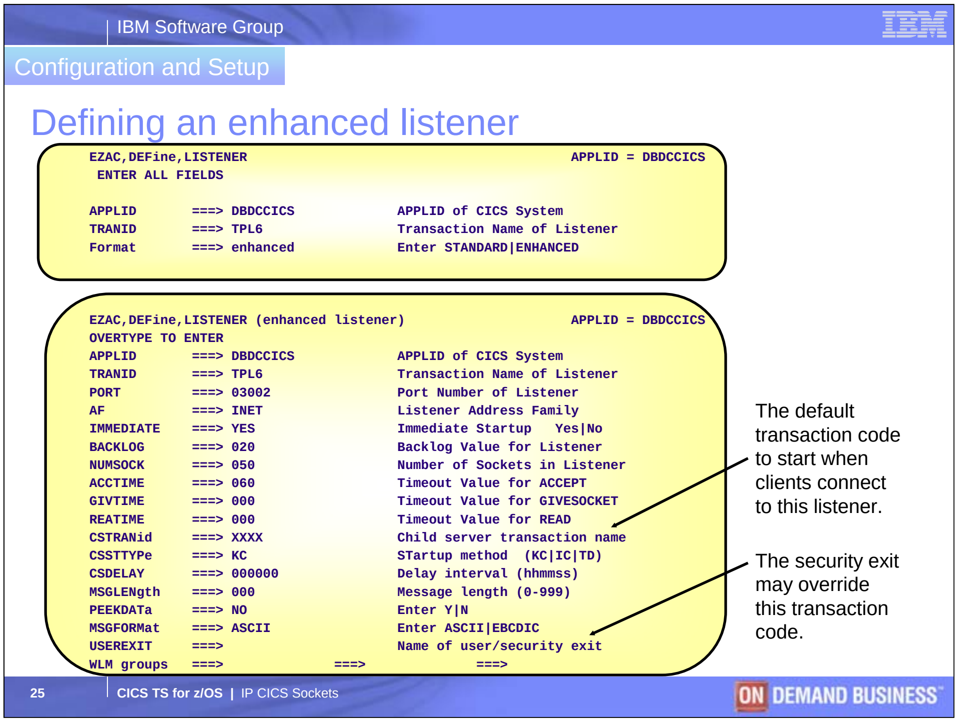
Listener entry in IP CICS Sockets configuration file - EZAC trans

```
EZAC,DISplay,LISTENER (standard listener)                APPLID = DBDCCICS


APPLID       ===> DBDCCICS              APPLID of CICS System
TRANID       ===> CSKL                  Transaction Name of Listener
PORT         ===> 03001                 Port Number of Listener
AF           ===> INET                  Listener Address Family
IMMEDIATE    ===> YES                   Immediate Startup   Yes|No
BACKLOG      ===> 040                   Backlog Value for Listener
NUMSOCK      ===> 100                   Number of Sockets in Listener
MINMSGL      ===> 004                   Minimum Message Length
ACCTIME      ===> 060                   Timeout Value for ACCEPT
GIVTIME      ===> 000                   Timeout Value for GIVESOCKET
REATIME      ===> 000                   Timeout Value for READ
TRANTRN      ===> NO                    Translate TRNID     Yes|No
TRANUSR      ===> NO                    Translate User Data Yes|No
SECEXIT      ===>                       Name of Security Exit
WLM groups   ===>          ===>           ===>


PF                 3 END                                  12 CNCL
```

You specify if the listener is an IPv4 or an IPv6 listener (INET or INET6)



MSGFORM tells te enhanced listener if errors should be reported in EBCDIC or ASCII to the client

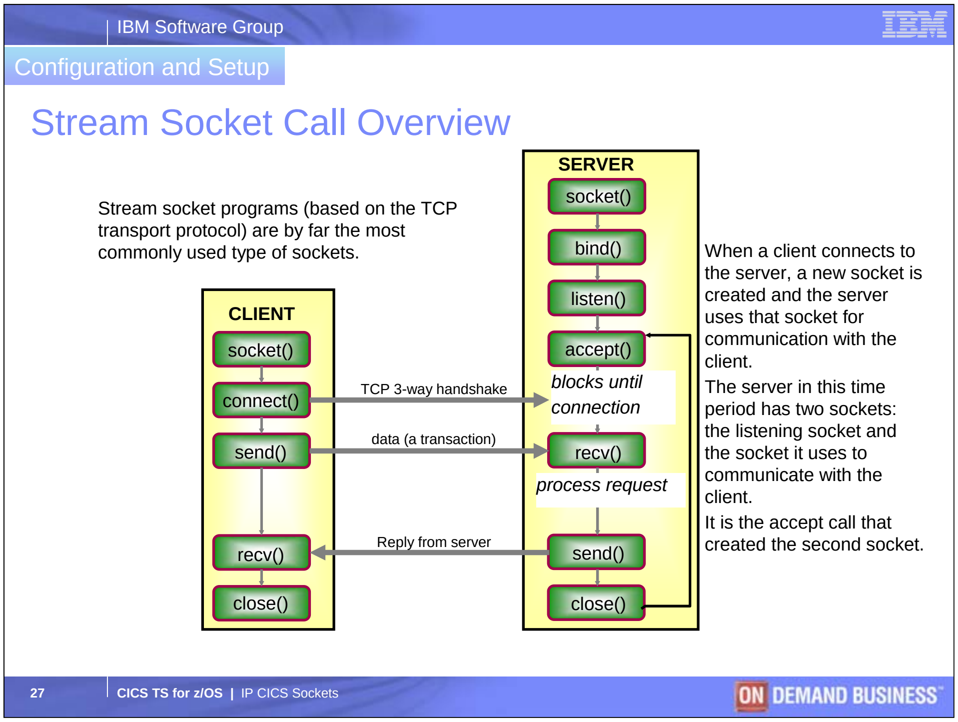
CSTTYPE and CSDELAY are also for the enhanced listener and defines IC delay

Common setup gotchas

- **CICS started task user ID must have an OMVS segment**
- **SYSTCPD DD statement must be defined in CICS startup JCL**
- **TCPIP.SEZATCP must be in DFHRPL concatenation**
- **If storage protection has been enabled, transactions EZAO, EZAP, and CSKL must be defined with TASKDATAKEY(CICS) and programs EZACIC00, EZACIC01, and EZACIC02 must have EXECKEY=CICS**
- **TASKDATALOC for EZAO, EZAP, and CSKL must be the same**

Look for error messages in the transient data queue identified by the ERRORTD configuration option and MSGUSR output from your CICS address space. IP CICS Sockets messages all start with prefix EZY:

```
EZY1224I 02/21/02 21:15:51 CICS/SOCKETS INITIALIZATION SUCCESSFUL
```



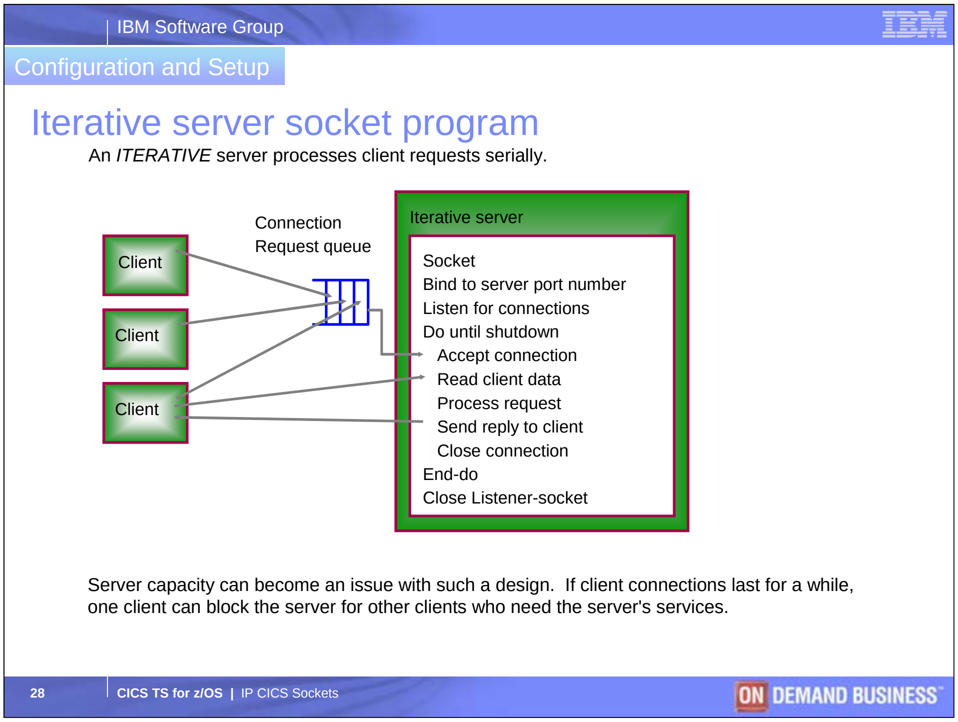
TCP connection establishment is performed using what is known as the TCP 3-way handshake, which consists of a SYN segment from client TCP layer to server TCP layer, a SYN+ACK segment from server TCP layer back to client TCP layer, and a final ACK segment from the client TCP layer.

During close() processing, FIN segments are exchanged to break the connection again.

The client and server may exchange any amount of data in any number of iterations during a connection.

If there is no traffic on the connection, the TCP layers may, at regular intervals, send KEEPALIVE segments to learn if the other end is still around or has vanished.

If any data is outstanding (not acknowledged), the TCP layers will retry and keep retrying for up to 3.5 minutes (TCP/IP MVS figures, other implementations may differ) using some rather complicated retransmission algorithms to calculate how long to wait before a retry should be done.



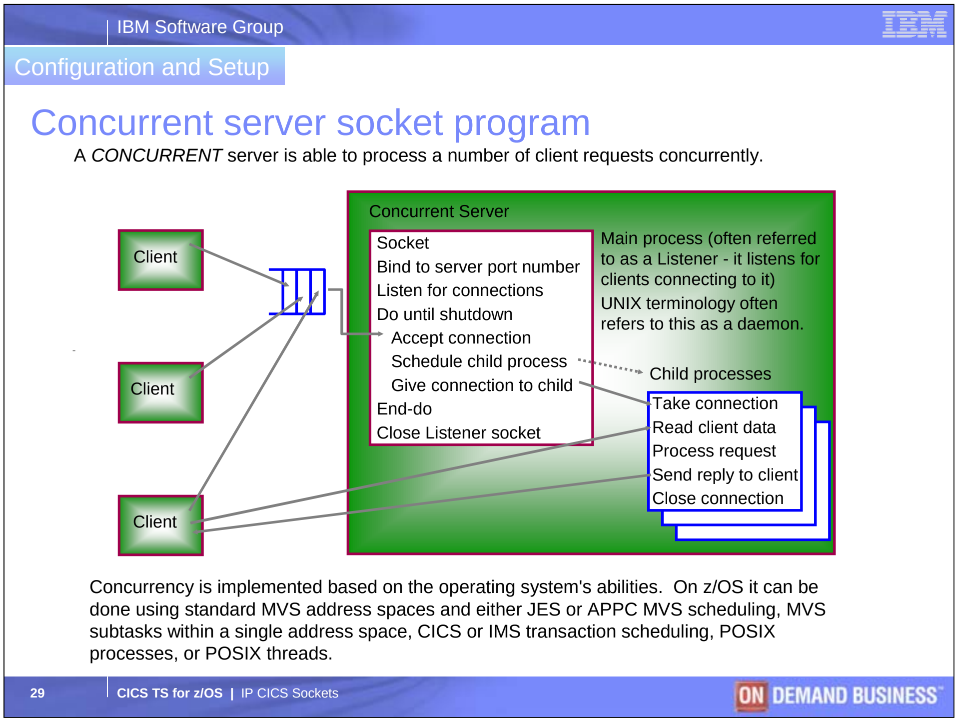
Both iterative and concurrent server are concepts that best match the stream socket application.

Good and simple design for short transactions.

Iterative servers are not good for long transactions that involve much processing or a number of iterations between server and client. Other clients will wait until this transaction has finished.

The Bind call associates this server with the preselected port number and fills in the local address part of the socket address so that the socket can be addressed from the clients. Normally the IP address is filled in as INADDR_ANY (binary zero) meaning that the server will accept connection requests from all available network interfaces.

The Listen call prepares the socket to accept connection requests from the clients. The size of the connection request queue is specified as a backlog value on this call. The maximum value is configured in TCPIP.PROFILE with the SOMAXCONN keyword (default is 10).



A concurrent server is good for high-performance, high-volume transactions where each transaction may vary in length.

The main process loop is very short, making it able to accept new connections and schedule them to parallel child processes very fast.

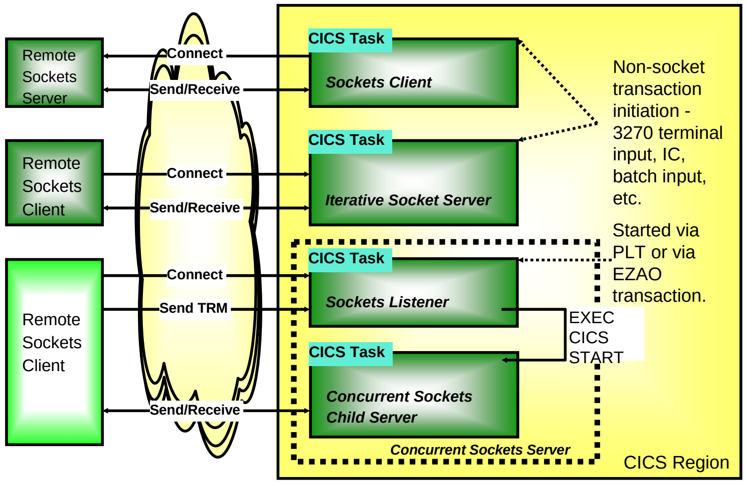
Child process scheduling depends on the environment: normal MVS address space will use subtasking (this requires either the use of Sockets Extended assembler macro API or C-sockets with C Multi Tasking Facility (MTF)). CICS will use CICS transactions. IMS will use IMS transactions. OpenEdition/MVS will use multiple forked address spaces, or the POSIX threading facilities.

In TCP/IP for MVS we use the GIVEOSOCKET/TAKESOCKET sequence to give a connection to a child process and to take it in the child process. In OpenEdition/MVS the socket is enherited by the forked child process that is able to use it as soon as the forking process has closed it. In a POSIX multithreaded environment, sockets are accessible by all threads.

The third and final category is a socket client program, which we do not decsribe in detail in this context.

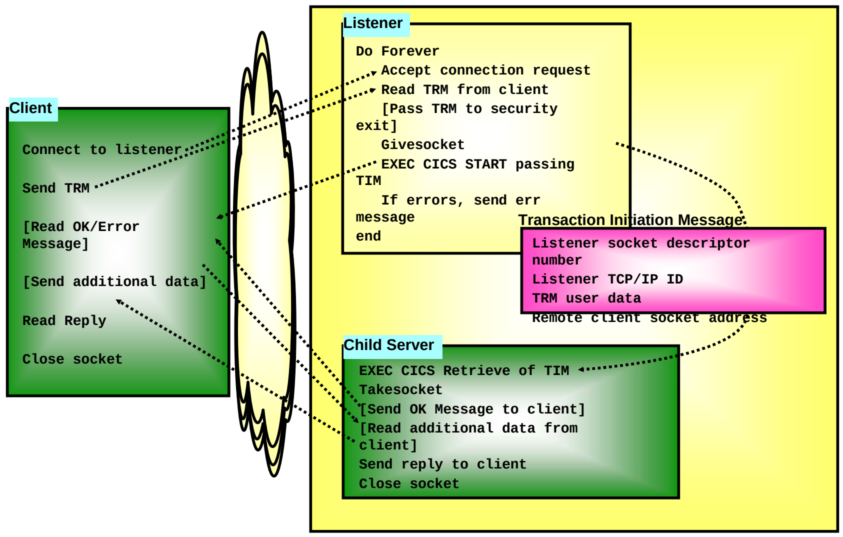
Configuration and Setup

Sockets program categories in CICS



TRM: Transaction Request Message

Concurrent CICS sockets server overview



The Transaction Request Message (TRM) format

```
Tran [,user data [,KC/IC/TD [,hhmmss]]]
```

The listener will analyze the Tran code and determine if it is ASCII or EBCDIC uppercase and perform translation of the remaining TRM accordingly - and according to your configuration of the listener transaction (TRANTRN and TRANUSR options).

Tran

- CICS transaction code in uppercase to start child server. Can be 1 to 4 bytes long.

User data

- Optionally includes up to 35 bytes of user data. The data can be input to be passed to child server in the TIM or it can be data that is required by your listener security exit routine, such as a user ID and a password.

KC/IC/TD

- Task Control (KC), Interval Control (IC) or Transient Data (TD). If nothing specified, startup is immediate using Task Control. IC or TD may also be specified in lowercase (ic or td). IF TD/td is specified, Tran is an intrapartition queue with trigger-level set to one or any value.

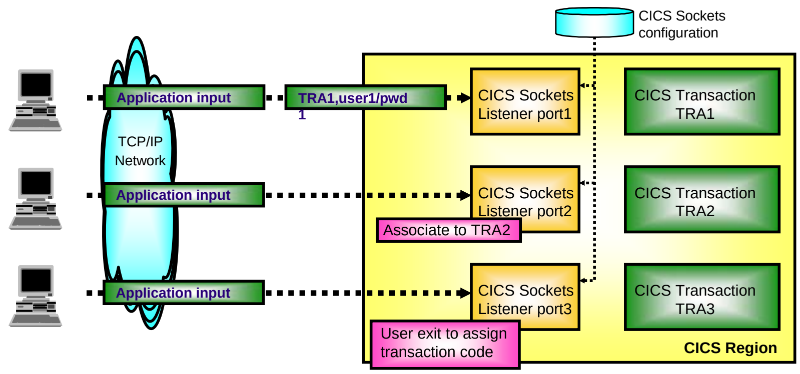
hhmmss

- If you specified IC above, you can here specify the interval time - all six digits must be specified (example: 000005 for 5 seconds interval).

Examples:

```
CICA
CICB,,KC
CIC1,,IC,000005
CIC2,MYUSER/MYPWD
CIC3
CIC4,,TD
```

Transaction initiation enhancements in CS z/OS V1R2



Three ways to launch CICS transactions:

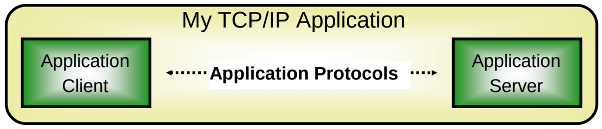
1. As before via a Transaction Request Message
2. Via a listener configuration option to associate listener instance with one specific CICS transaction code
3. Via a user exit, driven by the listener

With the two new options, data may be sent by the client in completely free format.

Error messages from the listener to the sockets client

- **The listener may in certain situations not be able to start the child server program (such as: unknown CICS transaction code, or transaction request rejected by the listener security exit). In those situations, the listener will send an error response back to the remote client and close the socket communication immediately with that client. If the client sent the transaction code in ASCII, the error message will be translated to US-ASCII before it is sent.**
  - EZY1315E 04/27/04 14:13:55 INVALID TRANID=ECHO PARTNER INET ADDR=9.42.104.161 PORT= 1128
  - EZY1303I 04/27/04 14:25:32 EZACIC02 GIVESOCKET TIMEOUT TRANS SRV1 PARTNER INET ADDR =9.42.104.161 PORT= 1129
- **The remote client needs to be developed in such a way that it is prepared to receive an error message from the listener:**
  - peek first three characters in the output stream and test for "EZY"
- **If everything is OK, the listener will not send an error message but it will start your child server program.**
- **The remote client may then, based on your application protocol, send some additional input data or just issue a second read and wait for a final reply from your child server.**

Application protocol



- **An application protocol defines as a minimum:**
  - The identity of the server (port number and transport protocol).
  - The format of data and commands that are exchanged between the two parts of the application
    - ASCII/EBCDIC
    - Fixed length or variable length messages
    - Character data, fixed point integers (little/big Endian), floating point, etc.
  - The state switching mechanisms of the application: when one end is allowed to send data, when the other end is expected to create and send a reply.
  - How error status is presented to the client.
  - How the dialog is orderly terminated.
  - If user authentication is required and how it is done.

Application protocol - messages

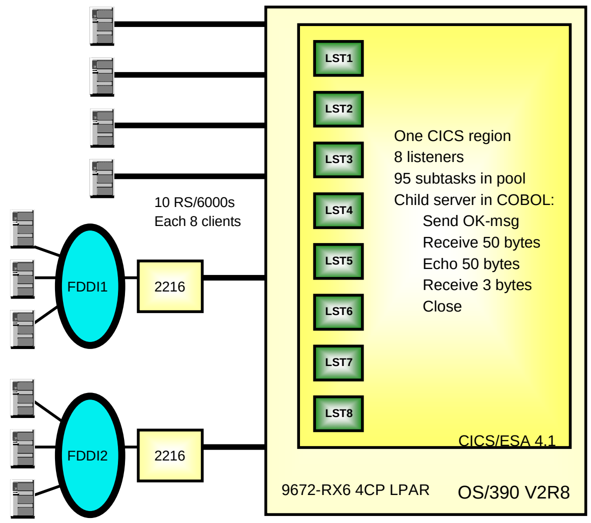
- **Three alternative message designs:**
  1. Record identifier and associated fixed message lengths. First byte holds a message type identifier, and for each message type, sender and receiver have agreed upon a fixed message length. Example: message type A is 26 bytes long, message type B is 54 bytes long.
  2. Record descriptor word in first 2 or 4 bytes in each message holding information about the length of the remaining message. If length is in binary, the application protocol should state that it must be in network byte order format (big Endian). If length is in character format, the length field should be right justified with leading zero characters: 0008 for an 8-byte long message.
  3. End-of-message markers. Typically used in C-programming, where messages often are null-terminated character strings: characters strings where the last byte is a x'00' byte.
- **The first two techniques are often implemented using a peeking read on the receiving side, where the receiver peeks at the first 1, 2 or 4 bytes to find out how much more data the receiver has to read to get the full message.**
- **Sometimes an application protocol includes a well-known character sequence in the beginning of the first message from a client. This sequence is used by the server to decide whether the client is sending character data in ASCII or in EBCDIC.**

CICS sockets callable service routines

- **Character Conversion**
  - EZACIC04: Translate from EBCDIC to ASCII
  - EZACIC05: Translate from ASCII to EBCDIC
  - Both routines are based on a variation of 8-bit US ASCII and IBM-037 US EBCDIC. If you need to use standard code sets, you have to write your own translation routines.
- **Converting between bit strings and character strings**
  - The select() call requires a bit string as argument and returns another bit string. The EZACIC06 routine can be used to convert between a bit string and a character string, which makes the use of bit strings easier for programs written in, for example, COBOL.
- **Get information out of resolver structures:**
  - Gethostbyname() and gethostbyaddr() calls return a so-called host entry structure, which is easily understood and interpreted by assembler and C programs, but not COBOL programs.
  - EZACIC08 is a callable routine that allows a COBOL program easy access to the information that is returned by the above two calls.
  - EZACIC09 is a callable routine that allows a COBOL program easy access to the information that is return by a Getaddrinfo() call. (z/OS V1R5)

Recommendations

Performance of CICS sockets



Results:

451.3 connections/sec

TCP/IP CPU: 9.68%
VTAM CPU: 0.32%
CICS CPU: 90%

NB: The transactions do not use any back-end data. If access to DB2 were needed, CICS would require more CPU and the number of connections per second would drop.

Results show the capabilities of TCP/IP and the CICS sockets API infrastructure.

Recommendations

To use it - or not to use it?

- **The CICS Sockets API is a low-level API:**
  - Detailed knowledge about sockets programming is required.
  - Even simple transactions require quite a bit of design and coding effort to implement.
  - Since it is a call API, CEDF does not get control when a sockets call is issued (It does when the call is passed to the TRUE, but not when your application calls the socket API).
  - Good performance.
- **This is NOT SNA LU6.2 programming:**
  - There is no 2-phase commit protocol built into sockets.
  - The closest you come to an APPC attach manager is the sockets listener, but there is no architected FMH5 that carries authentication information.
- **CICS Transaction Server provides many good Internet access alternatives today:**
  - CICS Transaction Gateway
  - IIOP for EJB and CORBA-initiated transactions
  - CICS Web Interface
  - Message queuing
  - 3270 transaction bridge

- **Don't decide to use CICS Sockets until you have carefully analyzed other, more functionally rich alternatives that today are offered by CICS TS. CICS Sockets is for those who need the details of a low-level API and the performance of a low-overhead solution.**

Recommendations

Advanced Topics

- Security Exit
- Writing your own listener
- TCP connection load balancing and availability
- Connection balancing
- Workload balancing with CICS sockets
- High-availability design for TCP/IP workload into the CICS environment
- Intra Sysplex CICS Transaction routing for sockets-initiated workload

Security exit

- **It is possible to write a security exit routine that is given control by the listener whenever a new request has been received from a remote client.**

- **The exit routine has access to the following information:**
  - The transaction code
  - Optional user data (based on installation standards, this could include a RACF user ID and password to be used by the security exit to authenticate the client)
  - Method of starting (IC, TD, or KC)
  - Optional interval control time
  - Client's socket address (addressing family, port, and IP address)
  - Socket descriptor

- **The exit routine returns**
  - Allow/disallow transaction
  - Listener or security exit sends error message to remote client
  - CICS terminal ID to be associated with new CICS task
  - CICS user ID to be associated with new CICS task (CICS/ESA V4 as a minimum)

- **Exit routine can modify the user data field, so a possible userID/password in the user data doesn't get sent over to the child server program in the TIM. The exit routine can also modify the transaction code, if so is desired or needed.**

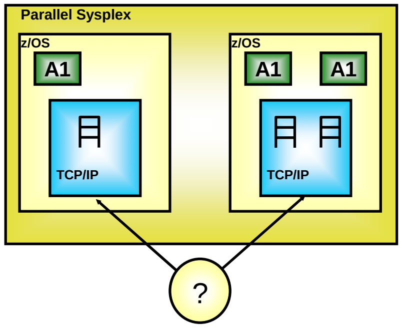
Recommendations

Writing your own listener

- The IBM-supplied standard listener requires a TRM from the remote client that has at least a CICS transaction code, and optionally more information.
- If an installation has unique requirements for the listener, it is fully possible to write your own listener. It is a normal socket program.
- Delay and throughput are important attributes in a listener, so the listener code must be developed so that it never blocks processing because a remote client is a bit slow in sending data. This requires the use of select logic, which is a special socket call that tests a list of socket descriptors for activity - allowing the listener code not to issue socket calls against sockets before it knows that the socket call will be serviced immediately.
- For efficiency, a listener can be coded in assembler - but COBOL or any other HLL is OK too.
- See IP CICS Sockets Guide, SC31-8518, Chapter 5 for details on writing your own listener.

Recommendations

TCP connection load balancing and availability



Load Balancing Technologies:

Between z/OS images:

- a DNS - DNS/WLM
- b NAT - Local Director, Alteon
- c MAC forwarders - IND, MNLB, Sysplex Distributor
- d Contents-based - Arrowpoint, BigIP, Alteon

Inside z/OS:

- e Port sharing

Application Characteristics:

- Multiple instances of the server are able to provide the exact same services to clients (will typically require data sharing)
- No state preserved at server between two connections (application protocol has to include support for such behavior)

Benefits of Intelligent Load Balancing:

- Improving response time
- Availability - If one instance goes down, connections with it break, but new connections can be established with remaining instance(s)
- Scalability - more server instances can be added on demand (horizontal growth)

Examples:

- Web server
- TN3270 server
- Some CICS applications
- FTP server (depends on data sharing)
- DB2

The load balancing technologies are the generalized ones. There are other solutions that are application-specific, such as the web servers use of WLM multiple address space support.

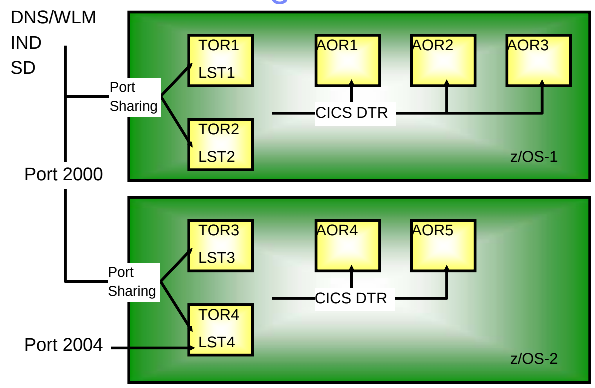
Connection balancing

- **Word of caution:**
  - Remember server instance is selected at TCP connection setup point in time.
    - If many CICS transaction requests are multiplexed over a single TCP connection, then connection balancing doesn't buy you a whole lot (ECI works this way).
  - Do not begin distributing connections across multiple server instances (CICS regions) unless you know for sure:
    - that each of the server instances can provide the exact same service
    - that your backend CICS transactions do not rely on local state data between a series of connections

- **SHAREPORT**
  - If one server instance in an LPAR can't handle all the workload, you may be able to start a second server instance on the same LPAR on the same port number. This requires that the TCP/IP systems programmer defines the port as SHARED - and TCP/IP will then send a new incoming connection request to the server in the shared pool of servers to the one that currently has the lowest number of active connections. SHAREPORT is limited to balance between servers in one LPAR.
  - Because of the SHARED attribute, multiple servers are able to bind to one and the same local IP address (optional - may also just bind to 0.0.0.0) and the same port number (for example, 80).

- **Sysplex Distributor, Network Dispatcher, OEM connection balancers (or sprayers) on outboard router or switch equipment**
  - These solutions can be deployed so that they can balance among multiple servers on different LPARs. The preferred technology today is Sysplex Distributor or some of the so-called content switch technologies from various router/switch vendors.
  - DNS/WLM is no longer a preferred technology and should not be used - use Sysplex Distributor instead in a Sysplex.

Workload balancing with CICS sockets



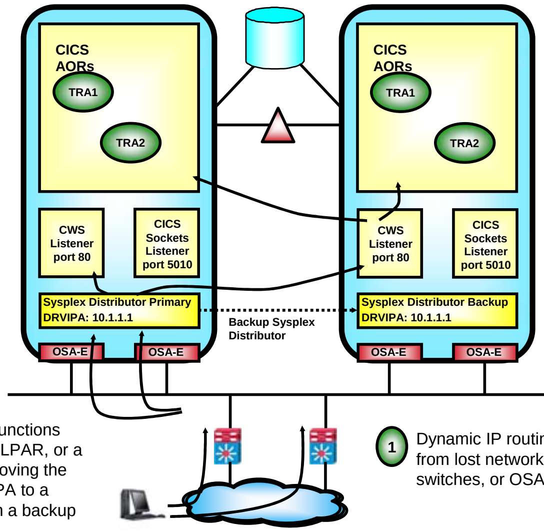
- Affinity to a certain TOR for multiple connections must be handled by the application protocol.
- An alternate port number could be defined for each TOR, and that port number could be returned to the client to use on succeeding connections that require affinity.
- Sysplex Distributor does from z/OS V1R4 support timer-based affinity

- CICS TS 1.3 supports Dynamic Transaction Routing for started transactions. Transaction must be defined as routable in region where start command is issued.
- DTRPGM=EYU9XLOP is the CICS Plex System Manager routine that will make a decision about which AOR to route to.
- The AOR must be on the same z/OS image as the TOR (givesocket/takesocket limited to same z/OS)

Recommendations

High-availability for TCP/IP workload into the CICS environment

Redundancy and use of automated recovery technologies are key to a successful high-availability design.



1 Dynamic IP routing will recover from lost network segments, switches, or OSA adapter ports

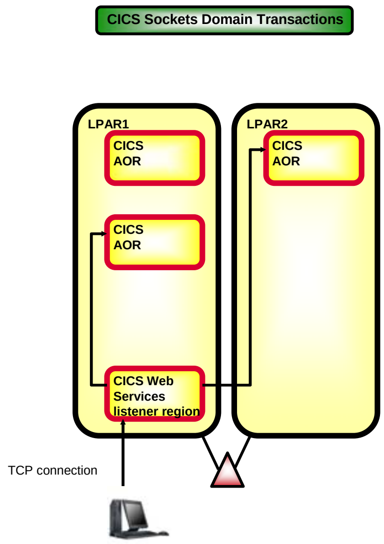
2 z/OS TCP/IP Sysplex functions will recover from a lost LPAR, or a lost TCP/IP stack by moving the distributed dynamic VIPA to a backup TCP/IP stack in a backup LPAR.

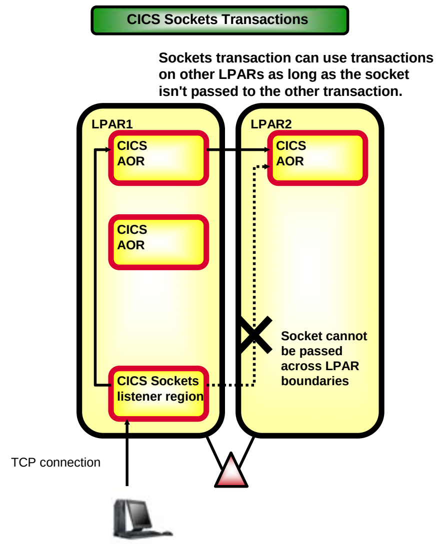
3 z/OS TCP/IP Sysplex functions will recover from a lost listener by redirecting all new connections to remaining listener(s) in the Sysplex.

4 z/OS CICS functions will recover from lost AORs

Recommendations

Intra-Sysplex CICS transaction routing for sockets-initiated workload





Summary

CICS Sockets enhancements in z/OS V1R5

- **IP CICS Sockets APIs support for the Basic Socket Interface Extensions for IPv6**
  - IP CICS C sockets
  - IP CICS sockets extended
- **Support for both IPv4 and IPv6 standard and enhanced listener**
- **Support for the IPv6 multicast options for GETSOCKOPT and SETSOCKOPT.**
  - IP CICS C sockets
  - IP CICS sockets extended
- **Support for IPv4 multicast options for getsockopt and setsockopt.**
  - IP CICS C sockets
- **Providing IPv6 and assembler samples**

Summary

CICS Sockets enhancements in z/OS V1R6

- **Focus on reliability:**
  - Improved peek-processing in the listener
    - Faulty client programs could in certain scenarios cause the listener to loop, trying to get enough data from the client
    - PQ68544 - PTFs for z/OS V1R2, V1R4, shipped with V1R5
  - Improved syntax checking in the listener
    - Faulty clients could in certain cases make the standard listener abend if some of the syntax rules for the TRM were violated
  - Improved shutdown processing where a large number of subtasks were active
    - There were some windows where subtasks weren't terminated before CICS terminated its main task - resulting in abend A03 during CICS shutdown
    - PQ76754 - PTFs for z/OS V1R2, V1R4, and V1R5

Summary

Documentation references

- **IP CICS Sockets Guide**
  - Manual SC31-8518
- **A Beginner's Guide to MVS TCP/IP Socket Programming**
  - Redbook GG24-2561
- **CICS/ESA and TCP/IP for MVS Sockets Interface**
  - Redbook GG24-4026

Summary

For More Information....

IBM eServer zSeries Mainframe Servers
http://www.ibm.com/servers/eserver/zseries

Networking: IBM zSeries Servers
http://www.ibm.com/servers/eserver/zseries/networking

IBM Enterprise Servers: Networking Technologies
http://www.ibm.com/servers/eserver/zseries/networking/technology.html

Networking & communications software
http://www.ibm.com/software/network

Communications Server
http://www.ibm.com/software/network/commserver

Communications Server white papers, product documentations, etc.
http://www.ibm.com/software/network/commserver/library

Communications Server technical Support
http://www.ibm.com/software/network/commserver/support

ITSO redbooks
http://www.redbooks.ibm.com

Advanced technical support (flashes, presentations, white papers, etc.)
http://www.ibm.com/support/techdocs/

Request For Comments (RFC)
http://www.rfc-editor.org/rfcsearch.html

Acknowledgements

- The following are trademarks of International Business Machines Corporation in the United States, other countries, or both: IBM, CICS, CICS/ESA, CICS TS, CICS Transaction Server, DB2, MQSeries, OS/390, S/390, WebSphere, z/OS, zSeries, Parallel Sysplex.
- Java, JavaBeans, and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
- Other company, product, and service names and logos may be trademarks or service marks of others.

During this presentation I may use some trademarks or abbreviations. They are shown here.

Moving on to the agenda on slide 3…