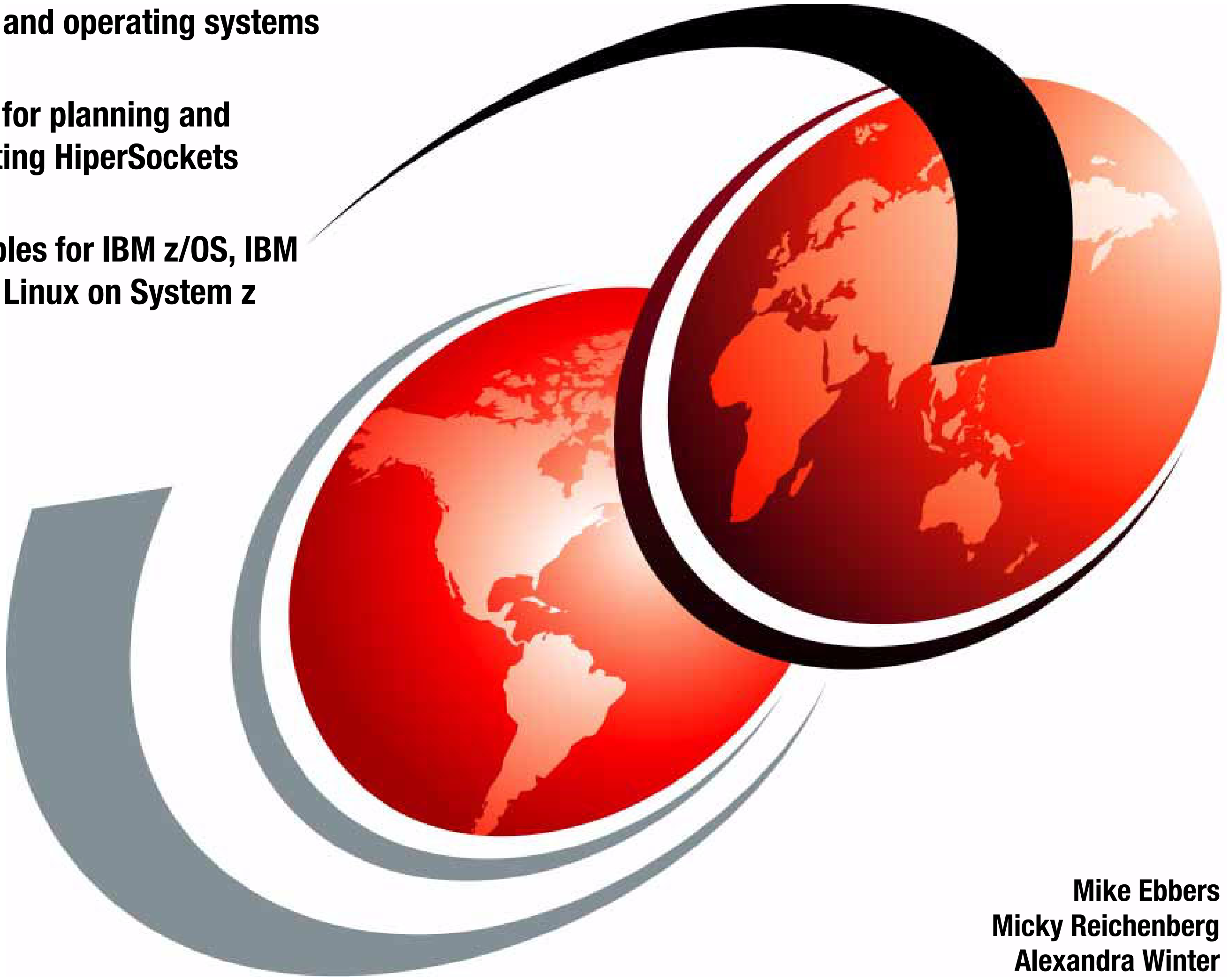
IBM HiperSockets Implementation Guide

Understand HiperSockets architecture, functions, and operating systems

Learn tips for planning and implementing HiperSockets

See examples for IBM z/OS, IBM z/VM, and Linux on System z



Mike Ebbers
Micky Reichenberg
Alexandra Winter

Redbooks

ibm.com/redbooks

International Technical Support Organization

IBM HiperSockets Implementation Guide

June 2014

SG24-6816-02

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

Third Edition (June 2014)

This edition applies to IBM zEnterprise systems.

Contents

Notices

Trademarks

Preface

This IBM® Redbooks® publication provides information about the IBM System z® HiperSockets™ function. It offers a broad description of the architecture, functions, and operating systems support. This publication will help you plan and implement HiperSockets. It provides information about the definitions needed to configure HiperSockets for the supported operating systems.

This book is intended for system programmers, network planners, and systems engineers who want to plan and install HiperSockets. A solid background in network and Transmission Control Protocol/Internet Protocol (TCP/IP) is assumed.

Authors

This book was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Poughkeepsie Center.

Mike Ebbers is a Consulting IT Specialist and Project Leader at the ITSO, Poughkeepsie Center. He has worked with IBM mainframe hardware and software products since 1974, in the field, in education, and in the ITSO.

Micky Reichenberg is an independent consultant for IBM z/OS® networking, Systems Network Architecture (SNA), and TCP/IP. He is based in Tel Aviv, Israel. Micky has over 35 years of experience in the industry, and focuses primarily on z/OS Communications Server, introducing and implementing new technologies, high availability (HA), problem determination, open systems connectivity to the IBM System z Enterprise Extender design and implementation. Before becoming a consultant, Micky was a systems engineer with IBM Israel for 17 years. During his assignment with the ITSO Raleigh, he published five networking-related IBM Redbooks publications. He holds a bachelor's degree in aeronautical engineering from the Techion Israel Institute of Technology.

Alexandra Winter is the System z HiperSockets Architect. She works in the IBM development laboratory in Boeblingen, Germany, continuously adding to the value of System z by designing and implementing new HiperSockets features and improvements. She has 16 years of experience in System z firmware development, and has worked on various System z, zSeries, and IBM S/390® projects (including Cryptographic Coprocessor, External Time Reference, Capacity on Demand, Concurrent Book Add, and HiperSockets). She holds a master's degree in electrical engineering.

Thanks to the following people for their contributions to this project:

Dave Bennin
Rich Conway
Roy Costa
Bob Haimowitz
International Technical Support Organization, Poughkeepsie Center

Wonjin Chung
IBM South Korea

Dody Kurniadi
IBM Indonesia

Joselito Manoto
IBM Australia

Rene Trumpp
IBM z/VSE® development, IBM Boeblingen

Thanks to the authors of the previous editions of this book:

- Authors of the first edition, *IBM HiperSockets Implementation Guide*

  Rama Ayyar and Velibor Uskokovic

- Authors of the second edition, *IBM HiperSockets Implementation Guide*

  Bill White, Roy Costa, Michael Gamble, Franck Injey, Giada Rauti, and Karan Singh

Now you can become a published author, too!

Comments welcome

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks
- Follow us on Twitter:

  http://twitter.com/ibmredbooks
- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806
- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm
- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Overview

This chapter provides a high-level overview of IBM System z HiperSockets and its benefits to System z customers. It describes the basic concepts, and explains how HiperSockets work internally.

1.1 Overview

The HiperSockets function, also known as internal queued direct input/output (iQDIO or internal QDIO), is an integrated function of the firmware of the System z central processor complexes (CPCs). It provides users with attachment to high-speed *logical* local area networks (LANs) with minimal system and network resource usage. HiperSockets 3provides internal *virtual* local area networks, which are Internet Protocol networks in the System z system.

Therefore, HiperSockets provides the fastest Transmission Control Protocol/Internet Protocol (TCP/IP) communication between consolidated Linux, IBM z/VM®, IBM z/VSE, and IBM z/OS virtual servers on the same system. Figure 1-1 shows a typical mainframe configuration, where multiple workloads are running in different logical partitions (LPARs), or in z/VM guests and in different operating systems. HiperSockets provide an efficient and secure internal network for all of these workloads to communicate with each other.

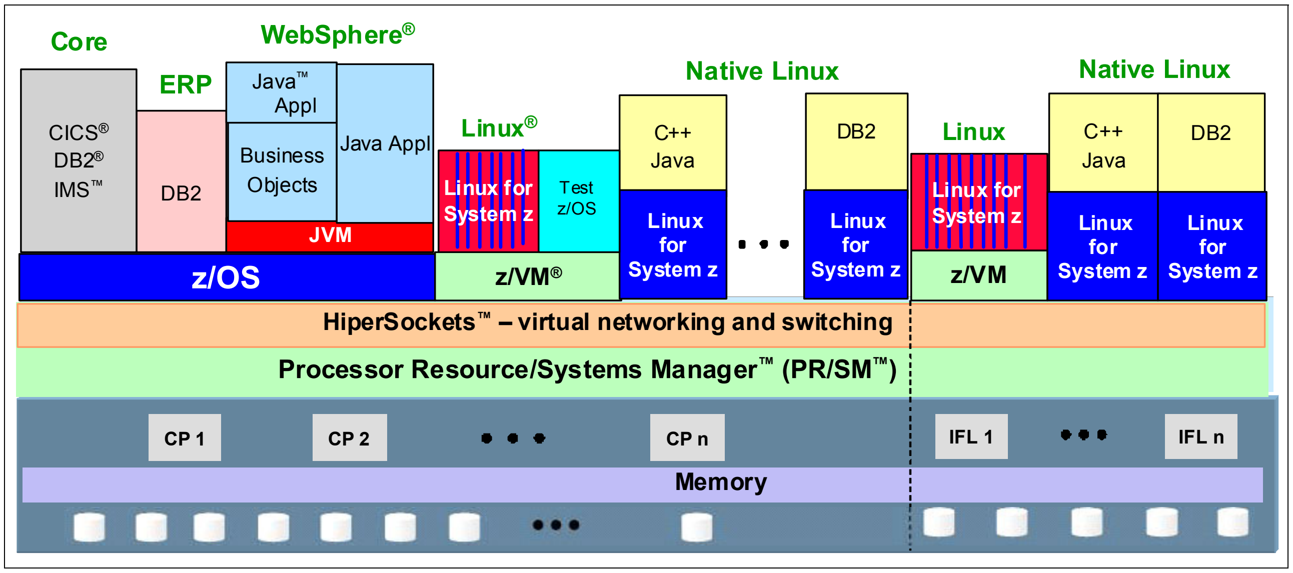


Figure 1-1 Multiple workloads on the mainframe

The virtual servers form a virtual LAN (VLAN). This technology eliminates the requirement for any physical cabling or external networking connection among them. It works as an internal LAN and moves data at memory speed between these virtual servers, providing a high-throughput and low-latency communication path.

HiperSockets is a function that emulates the Logical Link Control (LLC) layer of an Open Systems Adapter (OSA)-Express QDIO interface. It is implemented in System z firmware, which is also called Licensed Internal Code (LIC)[1]. This firmware function, coupled with supporting operating system device drivers, establishes a higher level of network availability, security, simplicity, performance, and cost effectiveness than is available when connecting single servers or LPARs together using an external Internet Protocol network.

HiperSockets is used by the following operating systems:

- z/OS
- z/VM
- z/VSE
- Linux on System z

[1] Sometimes known as microcode.

1.2 Server integration with HiperSockets

Many data center environments today are multi-tiered server applications, with a variety of middle-tier servers surrounding the System z data and transaction server. Interconnecting this multitude of servers requires the cost and complexity of many networking connections and components. The performance and availability of the inter-server communication is dependent on the performance and stability of the set of connections.

The more servers involved, the greater the number of network connections and complexity to install, administer, and maintain. Figure 1-2 shows two configurations. The configuration on the left shows a server farm surrounding a System z server, with its corporate data and transaction servers. This configuration is extremely complex, involving the backup of the servers and network connections. It is expensive and has a high administrative cost.

The configuration on the right consolidates the mid-tier workload onto multiple Linux virtual servers running on a System z server using the exceptionally reliable high-speed network that HiperSockets provides. In addition, these consolidated servers also have direct high-speed access to database and transaction servers running under z/OS on the same System z server. The external network connection for all servers is concentrated over a few high-speed OSA-Express interfaces.

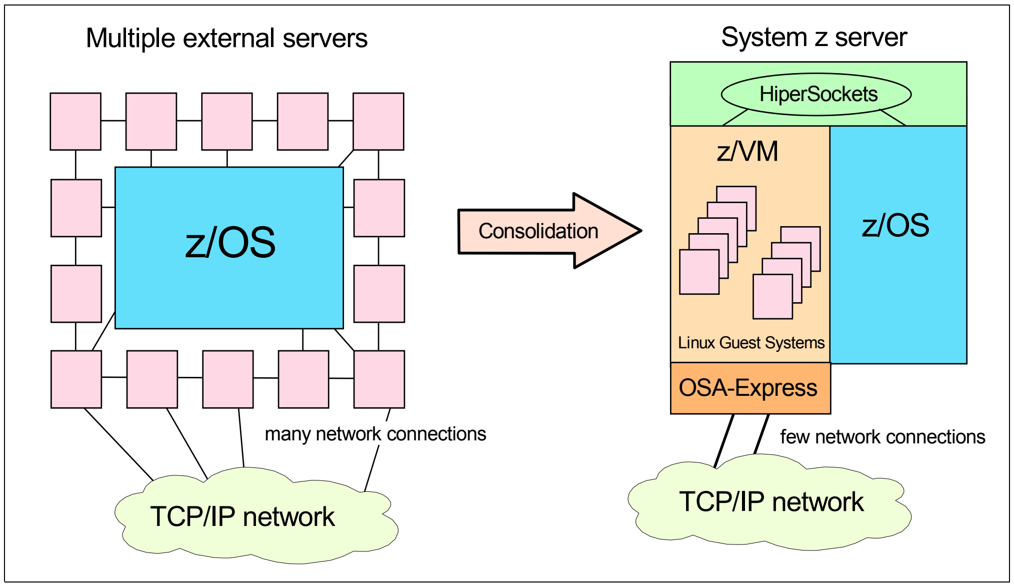


Figure 1-2 Server consolidation

1.3 HiperSockets benefits

The following list describes HiperSockets benefits:

- High performance

  Consolidated servers that have to access corporate data on the System z can do so at memory speeds with the lowest possible latency, bypassing network traffic and delays.

  Also, you can customize HiperSockets to accommodate varying traffic sizes. With HiperSockets, you can define a maximum frame size (MFS) according to the traffic characteristics for each HiperSockets. In contrast, Ethernet LANs have a MFS predefined by their architecture.

The bandwidth of your Ethernet interfaces becomes fully available for external network traffic when the internal traffic is running over HiperSockets.

- Sysplex connection improvement

  HiperSockets can also improve TCP/IP communications in a sysplex environment when the DYNAMICXCF facility is used.

- Cost savings

  You can use HiperSockets to communicate among consolidated servers in a single processor. Therefore, you can eliminate all of the hardware boxes running these separate servers, reducing floor space and power consumption. With HiperSockets, there are zero external components or cables to pay for, to replace, to maintain, or to wear out.

- Consolidation

  System z can be used to consolidate multiple servers into a single machine. With HiperSockets, this reduces the amount of server hardware, and can also reduce the complexity of your physical network, because subnets between the consolidated servers can be turned into virtual HiperSockets networks. The more you consolidate your servers, the greater your savings potential for costs related to external servers and their associated networking components.

- Flexibility

  Any changes to a HiperSockets network can easily be done without the need for installation of physical components or re-cabling.

- Simplicity

  HiperSockets is part of IBM z/Architecture® technology, including QDIO and advanced adapter interrupt handling. The data transfer is handled much like a cross-address space memory move, using the memory bus. HiperSockets is application-neutral, and it appears as a typical TCP/IP device. Its configuration is simple, making installation easy. It is supported by existing, known management and diagnostic tools.

- Security

  Cables or external components are vulnerable to physical attacks. Because HiperSockets does not have any physical interfaces, it is protected against any attacks from the outside. This might enable you to run network traffic without encryption, which results in additional performance improvements.

  For isolation purposes, you can connect servers to different HiperSockets. All security features, such as firewall filtering, are available for HiperSockets interfaces in the same way as they are with other Internet Protocol network interfaces. In a Sysplex environment, subplexing enables you to define security zones. Therefore, only members in the same security zone can communicate with each other.

- Availability

  With HiperSockets, there are no network switches, routers, adapters, or wires that can break or that have to be maintained. The absence of mechanical components greatly improves availability.

1.4 HiperSockets mode of operation

HiperSockets implementation is based on the OSA-Express QDIO protocol, therefore HiperSockets is called *internal QDIO*. In many ways, the firmware emulates the behavior of an OSA-Express QDIO interface, so common device drivers and control mechanisms can be used in the operating systems for both interfaces.

An OSA channel provides access to a physical LAN; likewise, a HiperSockets channel represents a VLAN. All servers that have access to this HiperSockets channel also have access to this VLAN.

Because HiperSockets represents virtual internal LANs, its components were implemented in a simpler and therefore better-performing way. All endpoints are inside the System z CPC, so all target addresses are known to the firmware, and no Address Resolution Protocol (ARP) is necessary.

The System z firmware maintains a lookup table of target addresses for each HiperSockets network. This table represents an *internal LAN*. At the time at which a TCP/IP stack starts a HiperSockets device, the device is registered in the target address lookup table. If a TCP/IP device is stopped, the entry for this device is deleted from the target address lookup table.

Also there is no need to fragment messages or encapsulate them in Ethernet frames. Messages are directly copied to the target device.

1.4.1 Unicast operations

The QDIO protocol operates on output queues and input queues that are in the main memory. The OSA channel operates asynchronously to the device driver. In contrast, HiperSockets unicast operations copy data synchronously from the output queue of the sending TCP/IP device to the input queue of the receiving TCP/IP device by using the memory bus to copy the data through an input/output (I/O) instruction.

HiperSockets operations are run on the CP where the I/O request is initiated. The completion of a data move is indicated to the receiving side with a Signal Adapter interrupt. Optionally, the receiving side can use dispatcher polling rather than handling Signal Adapter interrupts, also called *thin interrupts*. The data transfer is handled much like a cross-address space memory transfer using the memory bus, not the server I/O bus. HiperSockets does not contend with other system I/O activity, as shown in Figure 1-3.

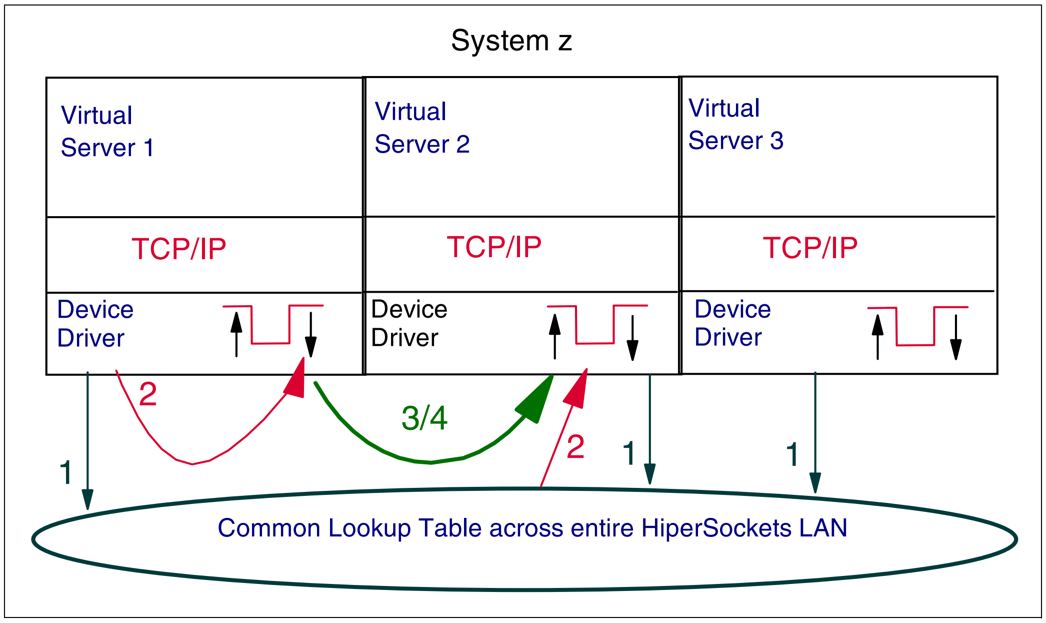


Figure 1-3 HiperSockets basic operation

The HiperSockets operational flow consists of these four steps:

1. Each virtual server's TCP/IP stack registers its target addresses into a HiperSockets Common Address Lookup table. There is one lookup table for each HiperSockets internal LAN. The scope of each LAN is the LPARs that are defined to share the HiperSockets internal queued direct communication (IQD) channel-path identifier (CHPID).
2. When data is being transferred, the send operation of HiperSockets performs a table lookup for the addresses of the sending and receiving TCP/IP stacks and their associated send and receive buffers.
3. HiperSockets firmware running on the sending processor copies the data from the sending device driver's send buffers into the target device driver's receive buffers.
4. HiperSockets firmware optionally delivers an interrupt to the target TCP/IP stack. This optional interrupt uses the thin interrupt support function of the System z server, which means the receiving host is going to look ahead, detecting and processing inbound data. This technique reduces the frequency of real I/O or external interrupts.

1.4.2 Multicast and broadcast

Multicast and broadcasts work similarly to the unicasts described previously. Virtual servers can register multicast addresses with the HiperSockets firmware, which are used to deliver messages to the targets. Broadcasts are delivered to all devices on the same IQD CHPID.

The major difference from unicast is that the memory copies for multicast and broadcast are not done synchronously inside the I/O instruction on the sending processor, but asynchronously by a system assist processor (SAP). Therefore, copying the message to multiple targets uses resources on the device driver and general-purpose processors.

2

HiperSockets environment definitions

This chapter describes how to define the IBM System z HiperSocket environment on your system. Hardware configuration definition (HCD) or input/output configuration program (IOCP) must be used to create an input/output configuration data set (IOCDS) with the HiperSocket channel-path identifier (CHPID) and subchannels (I/O device) definitions.

This chapter also provides information about how to define the environment for IBM z/VM guests, so that they can be attached to HiperSockets networks. Configuration definitions are discussed, and step-by-step instructions are described in detail. The following list includes topics covered in this chapter:

- System configuration considerations
- HCD definitions
- z/VM definitions

2.1 System configuration considerations

This chapter begins with a list of things to consider during configuration.

Important: Because this task of creating the IOCD defines the hardware configuration, it is used by all operating systems (z/OS, z/VM, z/VSE, and Linux on System z).

Consider the following information during configuration:

- Every CHPID that is shared among logical partitions (LPARs) can be seen as one local area network (LAN) interconnecting the LPARs.
- HiperSockets requires the definition of a CHPID with a type of internal queued direct communication (IQD). This CHPID is treated similar to any other CHPID, and is counted as one of the available channels.
- Because HiperSockets are shared among LPARs, the CHPIDs must be defined as shared in the hardware definition.
- HiperSockets channels can be configured to multiple channel subsystems (CSSs). They are indiscernibly shared by any or all of the configured LPARs without regard for the CSS to which the partition is configured (spanned channels).
- A minimum of three subchannel addresses must be configured:
  - One read control device
  - One write control device
  - One data device
- Each Transmission Control Protocol/Internet Protocol (TCP/IP) stack (maximum of eight on z/OS) on a single LPAR requires a single data device.
- The read device must be an even number. The write device must be the read device number plus 1.
- The data device can be any device number. It does not need to be the next sequential number after the read or write device numbers.

The HiperSockets firmware on IBM zEnterprise® or IBM z10™ central processor complexes (CPCs) supports the following components:

- Up to 32 independent HiperSockets on zEnterprise CPCs (previous models had 16) with a maximum of 12,288 I/O devices (valid subchannels) across all HiperSockets.
- A maximum total of 12,288 IP addresses across all HiperSockets. These IP addresses include the HiperSockets interface, virtual IP addresses (VIPAs), and dynamic virtual IP addresses (DVIPA) that are defined to the TCP/IP stack.
- 12,288 message authentication code (MAC) addresses per CPC.
- Each HiperSocket LAN has its own CHPID, type IQD:
  - It is a good practice to start assigning address xFF to the first CHPID, and work your way backward through the CHPID numbers, picking addresses from the high range to avoid addressing conflicts.
  - The CHPIDs can be shared by all defined LPARs.
  - The CHPIDs are delivered as object code only (OCO).
  - HiperSockets CHPIDs do not physically exist in the hardware, but these CHPID numbers cannot be used by other devices.

– The CHPIDs have no physical media constraint, so no priority queuing or cabling is required.
– Each operating system image configures its own usage of available HiperSockets CHPIDs.

2.1.1 Channel parameters for HiperSockets

Each CHPID has a configurable frame size (16 KB, 24 KB, 40 KB, or 64 KB) that supports optimizing per HiperSocket LAN for small packets versus large packets. The maximum transmission unit (MTU) size is 8 KB, 16 KB, 32 KB, or 56 KB. Table 2-1 summarizes the available frame sizes and the MTU.

Table 2-1 IQD CHPID maximum frame size (MFS) and MTU size

CHPID parameter	MFS	MTU size
`CHPARM=0x`	16 KB	8 KB
`CHPARM=4x`	24 KB	16 KB
`CHPARM=8x`	40 KB	32 KB
`CHPARM=Cx`	64 KB	56 KB

Note: On zEnterprise 196 (z196) and later processors, the channel parameter is also used to identify the usage function. The first character still indicates frame size. The second character indicates the function:

- x0 indicates normal HiperSockets.
- x2 indicates HiperSockets for intraensemble data network (IEDN).
- x4 indicates HiperSockets for z/VM External Bridge.

All “x” characters are wildcards.

2.2 HCD definitions

HiperSockets emulates an Open Systems Adapter (OSA) card. As with all channel-attached devices, an IQD CHPID must be defined by a channel path, a control unit, and I/O devices in your system configuration. This section shows the steps needed to define an IQD CHPID, using the z/OS HCD tool. It includes examples of the following definitions:

- Spanned channel path
- Control unit
- Devices

Channel IDs, control unit addresses, and device numbers do not represent physical entities for HiperSockets. They all are virtual entities. It does not matter which values you chose for these virtual entities. Just make sure that they do not conflict with other values that are used in your configuration. A simple approach is to keep the values that are proposed by HCD.

2.2.1 Dynamic Channel Path Management

As part of the z/OS Intelligent Resource Director, Dynamic Channel Path Management (DCM) enables the system to dynamically re-assign channel paths to connect control units based on the current workload and its service goals. HCD enables channel paths to be designated as static (fixed) or managed (re-assignable) when they are defined.

Note: HiperSockets can use DCM, but it is not used in the following examples.

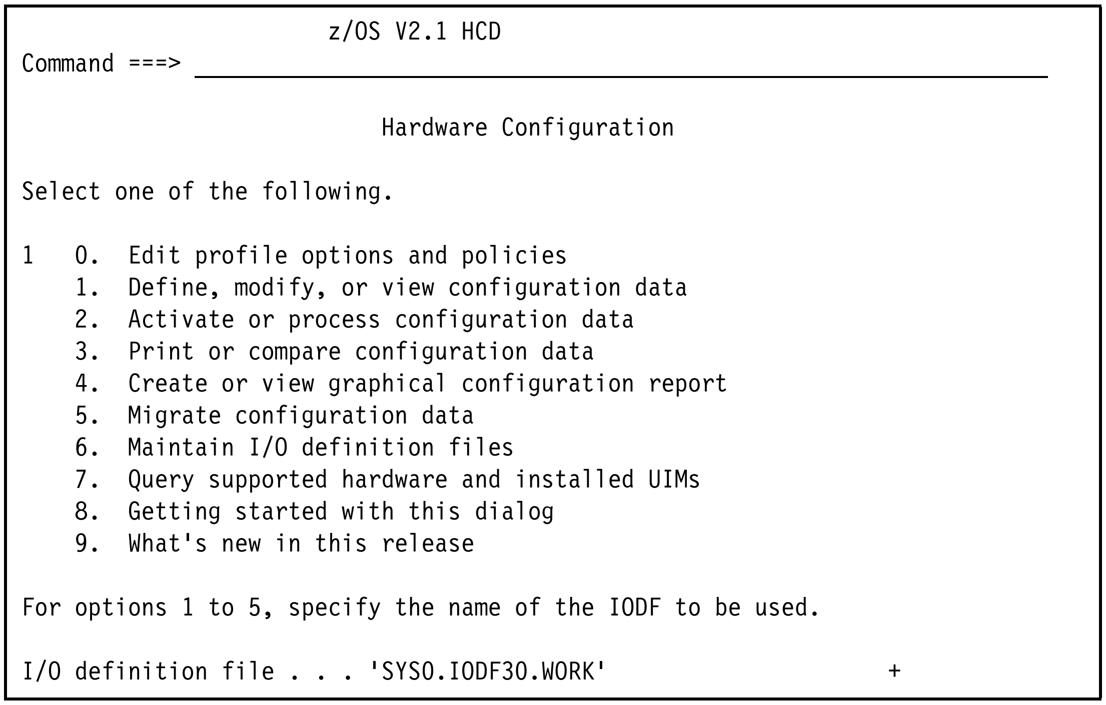
2.2.2 Channel path definitions

The process of defining a HiperSockets channel, control unit, and device is similar to defining any other set of channel, control unit, and device on z/OS using the HCD Interactive System Productivity Facility (ISPF) application. The following list describes the differences:

- During channel definition, a panel will be displayed to set the MFS and the function for the HiperSockets channel.
- A HiperSockets channel has no associated physical channel ID (PCHID).
- A minimum of three IQD devices must be defined for an IQD control unit.

Follow your installation's procedure to access the HCD main panel and enter the appropriate input/output definition file (IODF) file to begin the definition process:

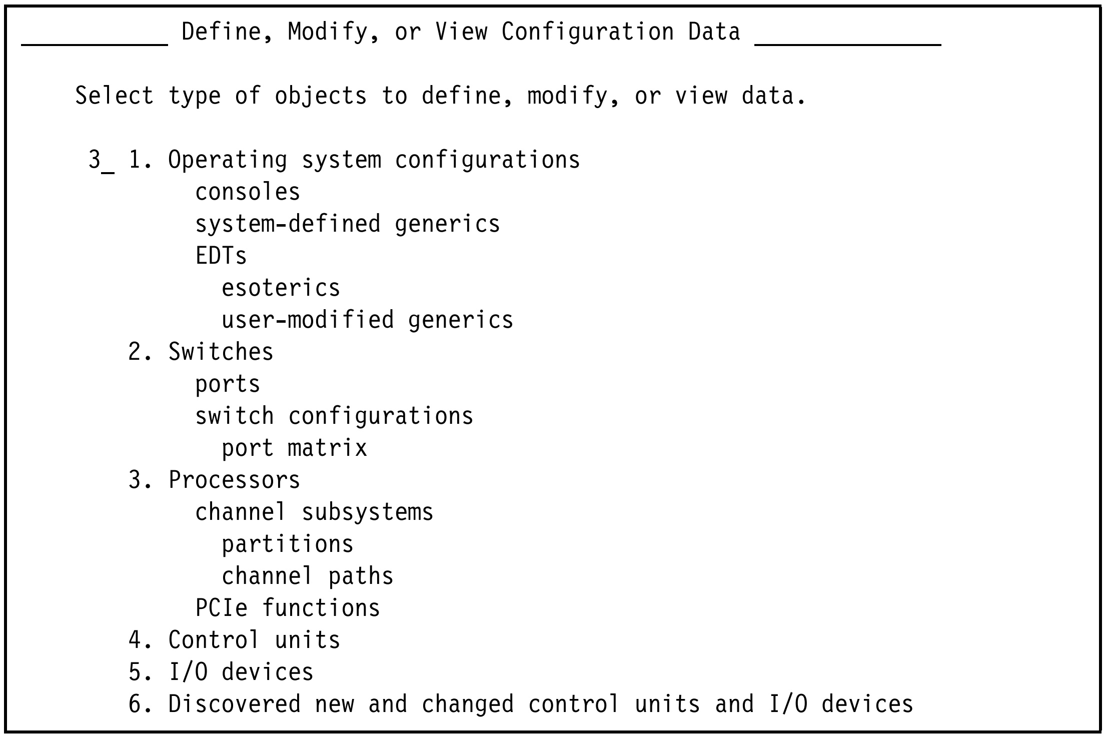
1. Starting from the HCD main menu panel, select 1, as shown in Figure 2-1.



```
                       z/OS V2.1 HCD
Command ===> ________________________________________________________________

                           Hardware Configuration

Select one of the following.

1   0.  Edit profile options and policies
    1.  Define, modify, or view configuration data
    2.  Activate or process configuration data
    3.  Print or compare configuration data
    4.  Create or view graphical configuration report
    5.  Migrate configuration data
    6.  Maintain I/O definition files
    7.  Query supported hardware and installed UIMs
    8.  Getting started with this dialog
    9.  What's new in this release

For options 1 to 5, specify the name of the IODF to be used.

I/O definition file . . . 'SYSO.IODF30.WORK'                     +
```

Figure 2-1 HCD main menu panel

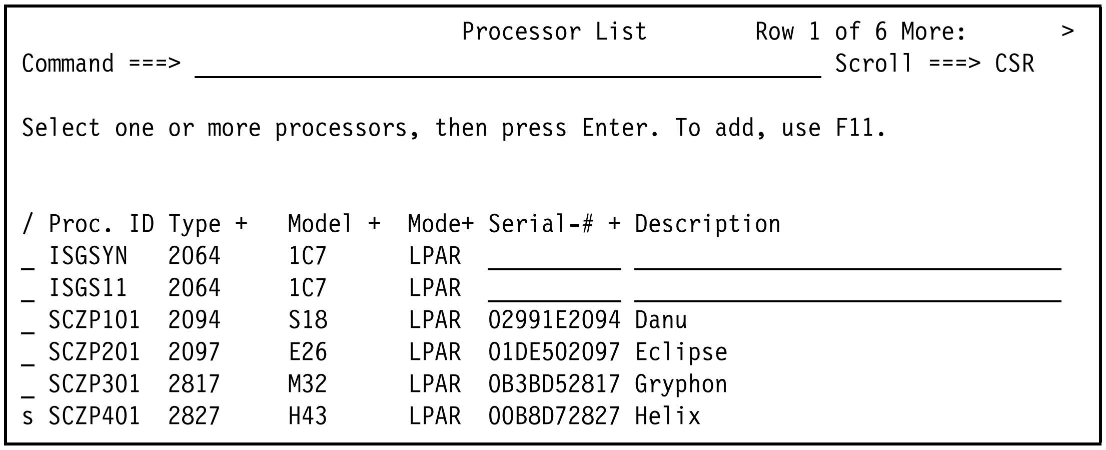
2. On the next panel, entitled Define, Modify, or View Configuration Data, select Option 3 - Processors, as shown in Figure 2-2.



```
___________ Define, Modify, or View Configuration Data ______________

    Select type of objects to define, modify, or view data.

     3_ 1. Operating system configurations
             consoles
             system-defined generics
             EDTs
               esoterics
               user-modified generics
        2. Switches
             ports
             switch configurations
               port matrix
        3. Processors
             channel subsystems
               partitions
               channel paths
             PCIe functions
        4. Control units
        5. I/O devices
        6. Discovered new and changed control units and I/O devices
```

Figure 2-2   Define, Modify, or View Configuration Data panel

3. Figure 2-3 shows the Processor List defined in the IODF data set. Select the processor to update and press Enter, as shown in Figure 2-3.



```
                                 Processor List       Row 1 of 6 More:      >
Command ===> _______________________________________________ Scroll ===> CSR

Select one or more processors, then press Enter. To add, use F11.


/ Proc. ID Type +   Model +  Mode+ Serial-# + Description
_ ISGSYN   2064     1C7      LPAR  __________ ________________________________
_ ISGS11   2064     1C7      LPAR  __________ ________________________________
_ SCZP101  2094     S18      LPAR  02991E2094 Danu
_ SCZP201  2097     E26      LPAR  01DE502097 Eclipse
_ SCZP301  2817     M32      LPAR  0B3BD52817 Gryphon
s SCZP401  2827     H43      LPAR  00B8D72827 Helix
```

Figure 2-3   Processor List

4. On a System z processor, this will display the Channel Subsystem list. Select a channel subsystem where the HiperSockets channel will be defined, as shown in Figure 2-4.

```
                            Channel Subsystem List    Row 1 of 4 More:       >
Command ===> _______________________________________________ Scroll ===> CSR

Select one or more channel subsystems, then press Enter.  To add, use F11.

Processor ID . . . : SCZP401      Helix

  CSS Devices in SS0    Devices in SS1    Devices in SS2
/ ID  Maximum + Actual  Maximum + Actual  Maximum + Actual
_ 0   65280     10667   65535     9612    65535     0
s 1   65280     10998   65535     9612    65535     0
_ 2   65280     10768   65535     9612    65535     0
_ 3   65280     11654   65535     9612    65535     0
```

Figure 2-4 Channel Subsystem List

5. This displays the Channel Path List panel (see Figure 2-5).

```
                              Channel Path List    Row 99 of 105 More:       >
Command ===> _______________________________________________ Scroll ===> CSR

Select one or more channel paths, then press Enter. To add use F11.

Processor ID . . . . : SCZP401      Helix
Configuration mode . : LPAR
Channel Subsystem ID : 1

        PCHID            Dyn Entry +
/ CHPID AID/P Type+ Mode+ Sw+ Sw Port Con Mng Description
_ F0    ____  IQD   SPAN  __  __ __       No  ________________________________
_ F1    ____  IQD   SPAN  __  __ __       No  ________________________________
_ F2    ____  IQD   SPAN  __  __ __       No  ________________________________
_ F3    ____  IQD   SPAN  __  __ __       No  IQDCHPID
_ F4    ____  IQD   SPAN  __  __ __       No  zAware
_ F8    ____  IQD   SPAN  __  __ __       No  IEDN Access (IQDX)
_ F9    ____  IQD   SPAN  __  __ __       No  External Bridge
```

Figure 2-5 Channel Path List

6. Press **F11** to add a channel path.

7. Enter all of the required information, as shown in Figure 2-6.

   This scenario uses the following values:

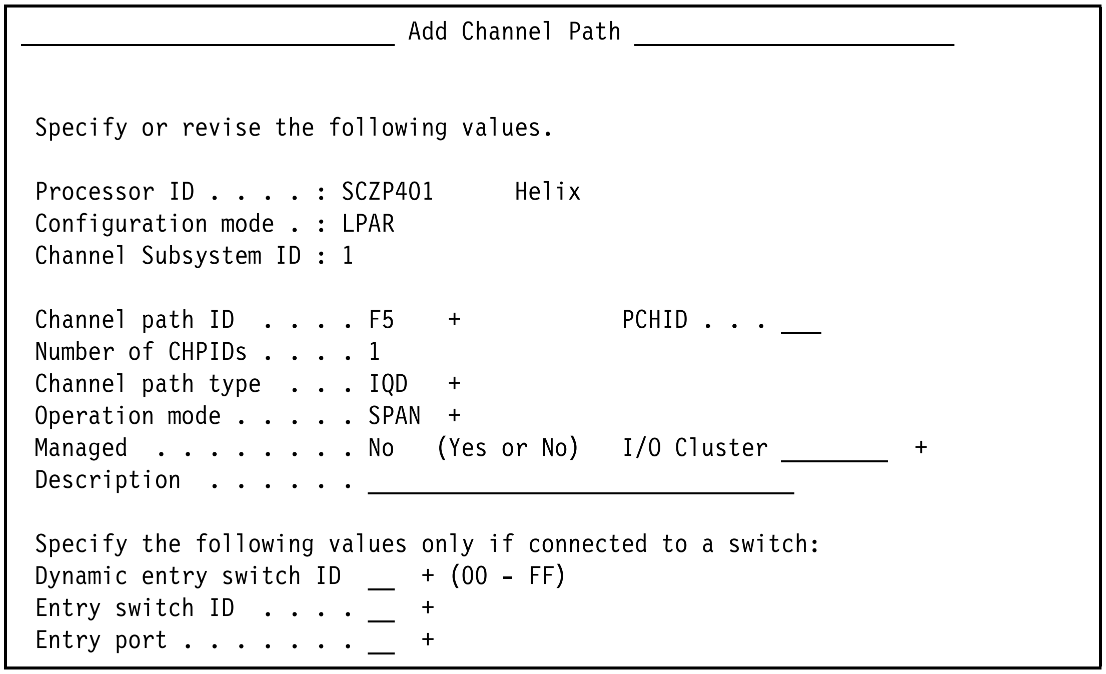
   – Enter a Channel path ID of `F5`.

   – Enter a Channel path type of `IQD` (required for a HiperSockets channel).

   – Enter an Operation mode of `SPAN`, because the IQD CHPID is shared among LPARs across CSSs.

   – Enter a description.

   – All other parameters can default. These are not relevant to IQD CHPIDs.



```
 ____________________________ Add Channel Path ________________________


 Specify or revise the following values.

 Processor ID . . . . : SCZP401      Helix
 Configuration mode . : LPAR
 Channel Subsystem ID : 1

 Channel path ID  . . . . F5    +           PCHID . . . ___
 Number of CHPIDs . . . . 1
 Channel path type  . . . IQD   +
 Operation mode . . . . . SPAN  +
 Managed  . . . . . . . . No   (Yes or No)   I/O Cluster ________   +
 Description  . . . . . . ________________________________

 Specify the following values only if connected to a switch:
 Dynamic entry switch ID  __  + (00 - FF)
 Entry switch ID  . . . . __  +
 Entry port . . . . . . . __  +
```

Figure 2-6   Add a channel path

8. When the definitions are completed, press Enter. If the definition process is started from a production IODF, a panel opens enabling you to create a work IODF. Enter the appropriate information to create a work IODF.

9. The next panel that opens is Specify IQD Channel Parameters, as shown Figure 2-7 on page 14. Select an MFS.

   Important: The MFS is directly related to the MTU used by TCP/IP.

   Press **F4** for a list of the four possible options. Choose the default size of `16 KB`. Because this is the default, no operating system (OS) value will appear in the IOCP. Table 2-1 on page 9 shows OS (**`CHPARM`**) and MTU values for maximum frame sizes other than 16 KB in IOCP. Press Enter.

10. Select the IQD function:

   – Select `External Bridge` if you intend to deploy a z/VM Virtual Switch with HiperSockets Bridge port on this channel. See 8.4, “The z/VM Virtual Switch with HiperSockets bridge port” on page 112 for more information about the HiperSockets Bridge port.

– Select `IEDN Access (IQDX)` if this channel is intended to be part of a zEnterprise IEDN. There can only be one internal queued direct input/output extensions (IQDX) channel per CPC. See Chapter 9, “HiperSockets in an IBM zEnterprise ensemble” on page 125 for more information about the IQDX.

– Select `Basic HiperSockets` in all other cases.

Figure 2-7 shows the Specify IQD Channel Parameters panel.

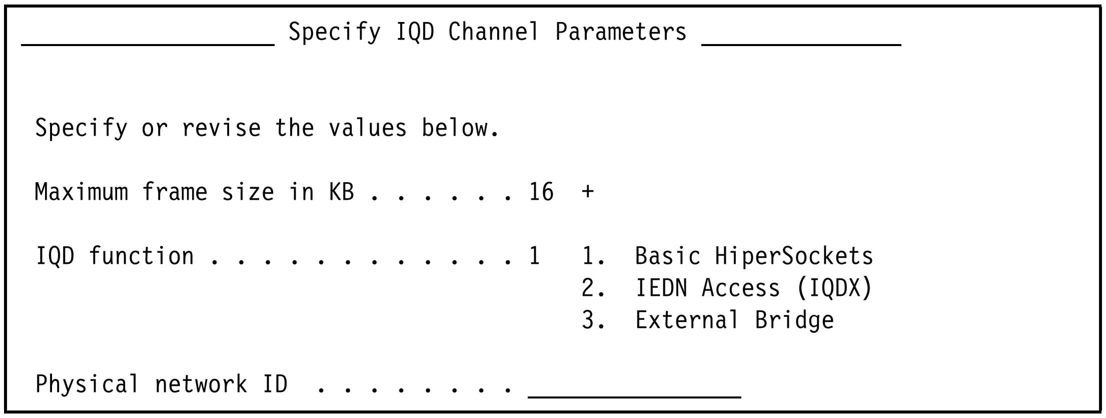

```
___________________ Specify IQD Channel Parameters _______________


 Specify or revise the values below.

 Maximum frame size in KB . . . . . . 16  +

 IQD function . . . . . . . . . . . . 1   1.  Basic HiperSockets
                                          2.  IEDN Access (IQDX)
                                          3.  External Bridge

 Physical network ID  . . . . . . . . ________________
```

Figure 2-7 Define the MFS and function

11. Optionally specify Physical network IDs.

The **`PNETID= keyword`** parameter is optional, and specifies the physical network IDs (up to four). A HiperSockets interface does not have multiple ports, so it makes sense to define only one physical network ID.

The following rules apply to the physical network IDs:

– They must be 1 - 16 characters in length.
– Characters allowed are A-Z and 0-9.
– There can be 1 - 4 physical network IDs corresponding to the ports on the card.

Physical network IDs are used for network cards to establish the affinity to a particular physical network:

– Remote Direct Memory Access (RDMA) over Converged Ethernet (10 Gb RoCE)
– OSA-Express queued direct input/output (QDIO), called OSD
– Internal queued direct communication (IQD)

When network connections require two interfaces to be coordinated, as with 10 Gb RoCE and an associated OSD, the physical network ID should match.

If you intend to bridge the HiperSockets channel to one or more OSD channels, it is a good practice to assign the same physical network ID to the IQD channel and the OSD channels, because they will form one network. However, this is not required.

For function type IQDX, the physical network ID will automatically be assigned to `IEDN`.

12. In the Define Access List panel, complete the access list for the partitions sharing the channel, and press Enter.

In this example, the IQD CHPID was defined as shared by two LPARs on channel subsystem 1 and one LPAR on channel subsystem 2, as shown in Figure 2-8.

```
__________________________ Define Access List ___________________________
                                                            Row 22 of 53
Command ===> _________________________________________ Scroll ===> CSR

Select one or more partitions for inclusion in the access list.

Channel subsystem ID : 1
Channel path ID  . . : F5     Channel path type  . : IQD
Operation mode . . . : SPAN   Number of CHPIDs . . : 1

/ CSS ID Partition Name   Number Usage Description
/ 1      A11              1      OS    COMMPLEX SC30
_ 1      A12              2      OS    VMLINUX9
/ 1      A13              3      OS    COMMPLEX SC31
/ 2      A2E              E      OS    VMLINUX1
```

Figure 2-8 Define Access List - panel 1

13. Return to Channel Path List, which shows CHPID `F5` defined (Figure 2-9).

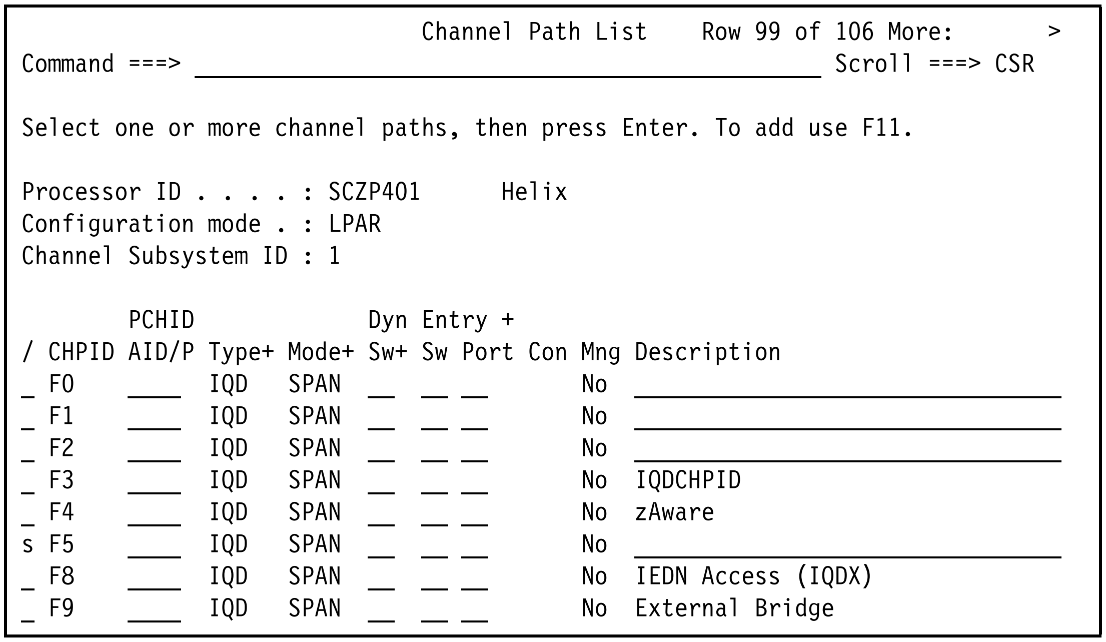
```
                              Channel Path List    Row 99 of 106 More:      >
Command ===> _______________________________________________ Scroll ===> CSR

Select one or more channel paths, then press Enter. To add use F11.

Processor ID . . . . : SCZP401      Helix
Configuration mode . : LPAR
Channel Subsystem ID : 1

       PCHID            Dyn Entry +
/ CHPID AID/P Type+ Mode+ Sw+ Sw Port Con Mng Description
_ F0    ____  IQD   SPAN  __  __ __       No  ________________________________
_ F1    ____  IQD   SPAN  __  __ __       No  ________________________________
_ F2    ____  IQD   SPAN  __  __ __       No  ________________________________
_ F3    ____  IQD   SPAN  __  __ __       No  IQDCHPID
_ F4    ____  IQD   SPAN  __  __ __       No  zAware
_ F5    ____  IQD   SPAN  __  __ __       No  ________________________________
_ F8    ____  IQD   SPAN  __  __ __       No  IEDN Access (IQDX)
_ F9    ____  IQD   SPAN  __  __ __       No  External Bridge
```

Figure 2-9 Channel Path after the CHIPID is defined

2.2.3 Control unit definitions

Follow these steps to define control units:

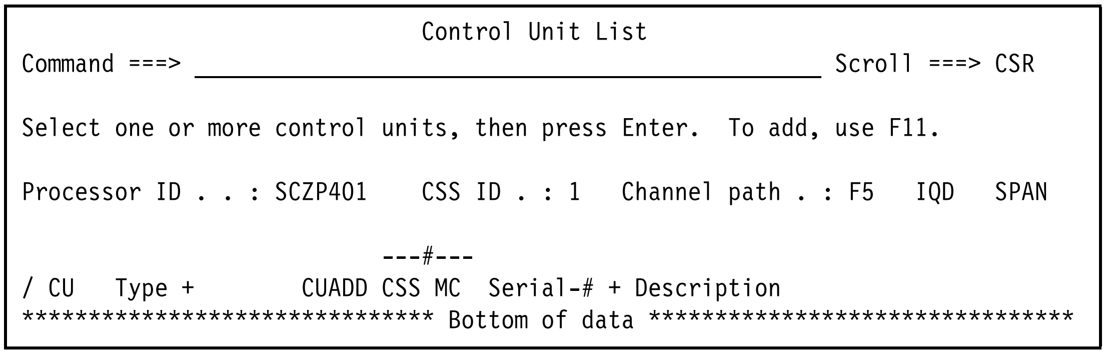
1. Starting at the Channel Path List panel, select the CHPID to get to the control unit list, as shown in Figure 2-10.



```
                              Channel Path List    Row 99 of 106 More:      >
Command ===> _______________________________________________ Scroll ===> CSR

Select one or more channel paths, then press Enter. To add use F11.

Processor ID . . . . : SCZP401      Helix
Configuration mode . : LPAR
Channel Subsystem ID : 1

       PCHID            Dyn Entry +
/ CHPID AID/P Type+ Mode+ Sw+ Sw Port Con Mng Description
_ F0    ____  IQD   SPAN  __  __ __       No  ________________________________
_ F1    ____  IQD   SPAN  __  __ __       No  ________________________________
_ F2    ____  IQD   SPAN  __  __ __       No  ________________________________
_ F3    ____  IQD   SPAN  __  __ __       No  IQDCHPID
_ F4    ____  IQD   SPAN  __  __ __       No  zAware
s F5    ____  IQD   SPAN  __  __ __       No  ________________________________
_ F8    ____  IQD   SPAN  __  __ __       No  IEDN Access (IQDX)
_ F9    ____  IQD   SPAN  __  __ __       No  External Bridge
```

Figure 2-10 Select a control unit list

2. Press F11 to add a control unit when the Control Unit List panel displays, as shown in Figure 2-11.



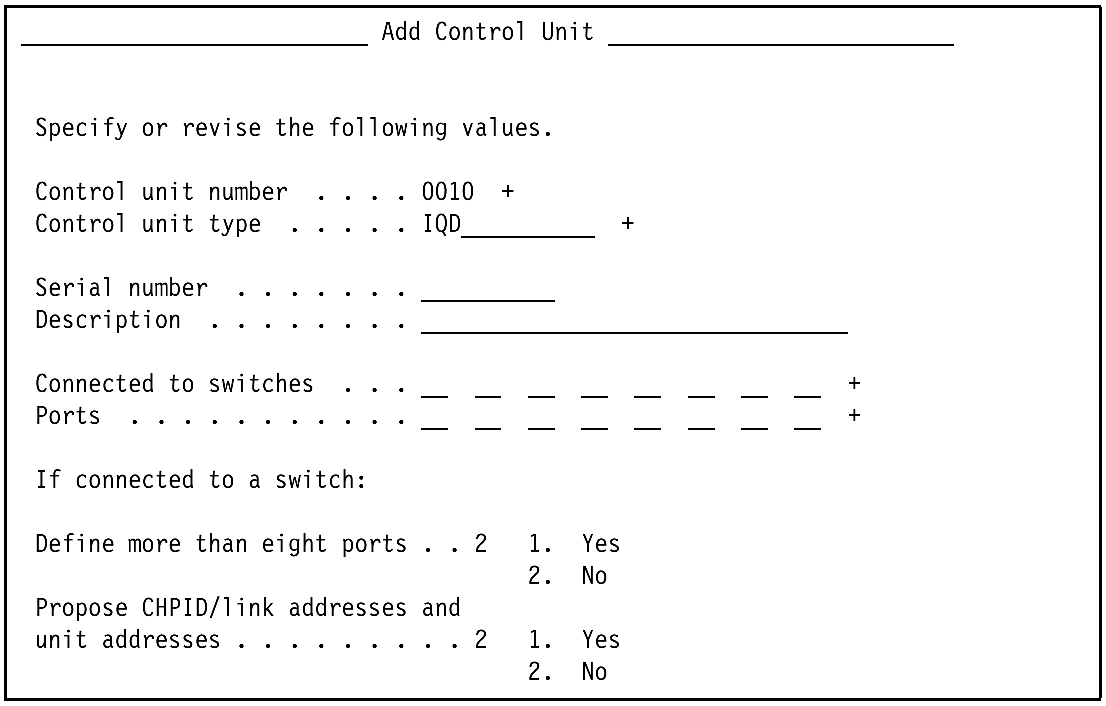
```
                              Control Unit List
Command ===> _______________________________________________ Scroll ===> CSR

Select one or more control units, then press Enter.  To add, use F11.

Processor ID . . : SCZP401    CSS ID . : 1   Channel path . : F5   IQD   SPAN

                           ---#---
/ CU   Type +       CUADD CSS MC  Serial-# + Description
******************************* Bottom of data ********************************
```

Figure 2-11 Add a control unit

3. Enter the required information, as shown in Figure 2-12, and then press Enter.

   In this example, the following values were set:

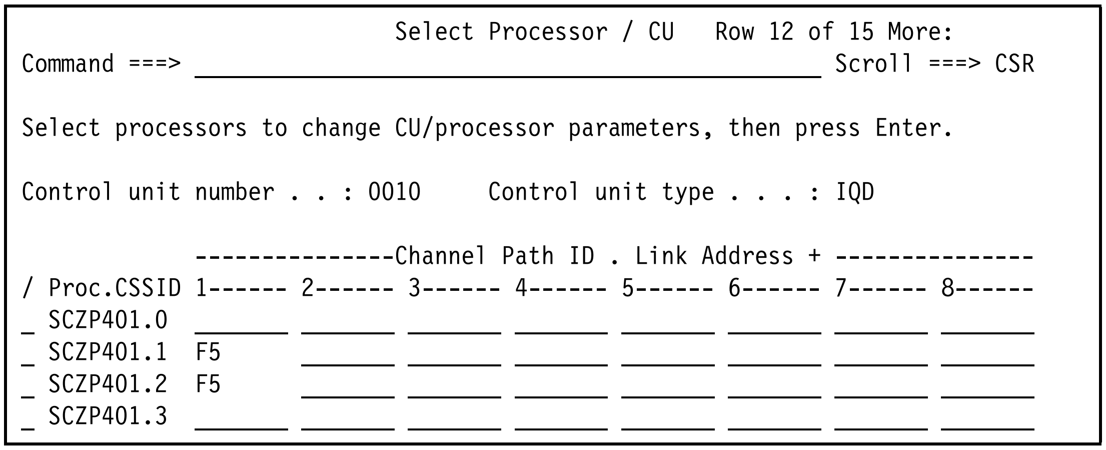
   – Control unit number to `0010`. You can choose any value, as long as it does not conflict with any HCD rules.
   – Control unit type to `IQD` (required for a HiperSockets control unit).



```
 _________________________ Add Control Unit _________________________


 Specify or revise the following values.

 Control unit number  . . . . 0010  +
 Control unit type  . . . . . IQD__________   +

 Serial number  . . . . . . . __________
 Description  . . . . . . . . ________________________________

 Connected to switches  . . . __  __  __  __  __  __  __  __  +
 Ports  . . . . . . . . . . . __  __  __  __  __  __  __  __  +

 If connected to a switch:

 Define more than eight ports . . 2   1.  Yes
                                      2.  No
 Propose CHPID/link addresses and
 unit addresses . . . . . . . . . 2   1.  Yes
                                      2.  No
```

Figure 2-12   Define a control unit

4. Attach the control unit to a processor channel subsystem. In this example, the control unit was attached to channel subsystem 1 and 2 of the selected processor. Define CHPID `F5` as the CHPID that connects to the control unit, as shown in Figure 2-13, and then press **F20**.

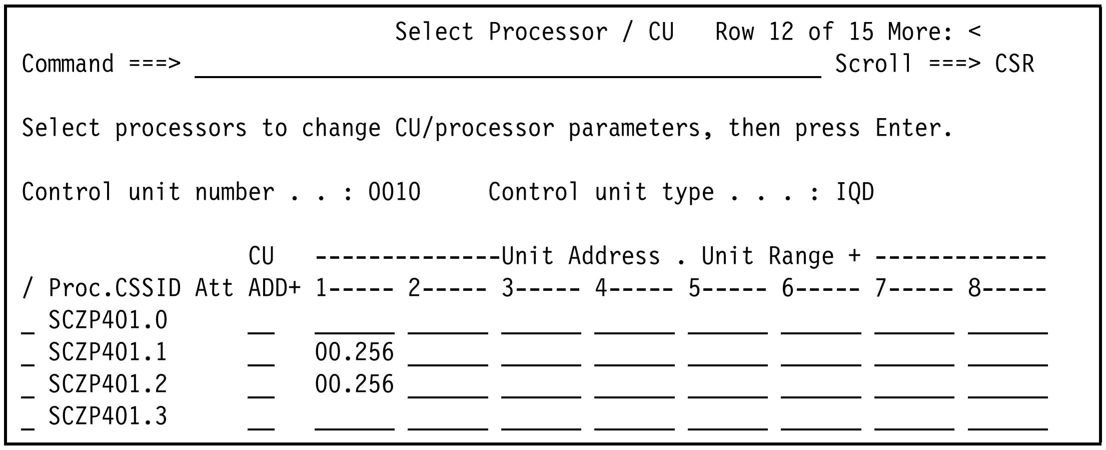


```
                            Select Processor / CU   Row 12 of 15 More:
Command ===> _______________________________________________ Scroll ===> CSR

Select processors to change CU/processor parameters, then press Enter.

Control unit number . . : 0010     Control unit type . . . : IQD

             ---------------Channel Path ID . Link Address + ---------------
/ Proc.CSSID 1------ 2------ 3------ 4------ 5------ 6------ 7------ 8------
_ SCZP401.0  _______ _______ _______ _______ _______ _______ _______ _______
_ SCZP401.1  F5      _______ _______ _______ _______ _______ _______ _______
_ SCZP401.2  F5      _______ _______ _______ _______ _______ _______ _______
_ SCZP401.3  _______ _______ _______ _______ _______ _______ _______ _______
```

Figure 2-13   Select a processor to the control unit

5. Define the control unit starting address as `00` for a range of `256` devices (see Figure 2-14) and then press `Enter`.

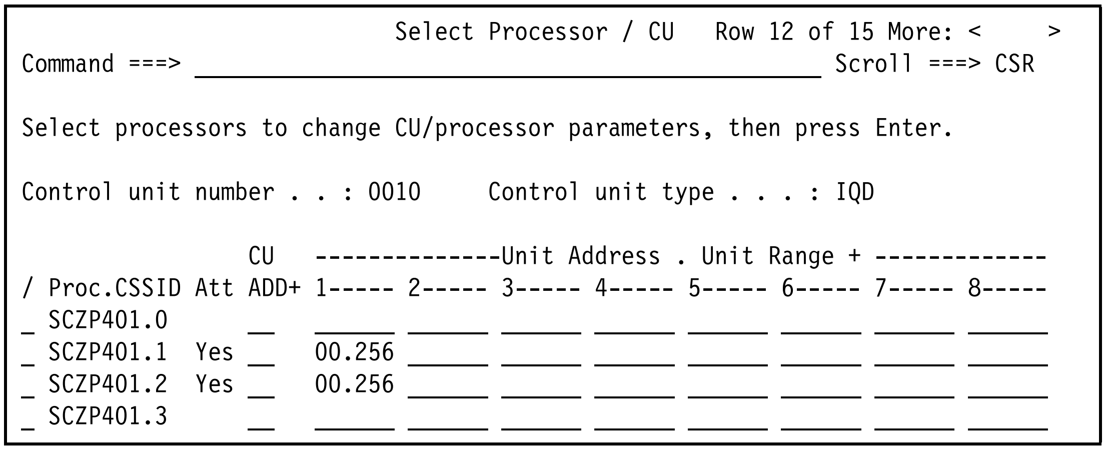
Tip: The control unit address range defines the number of devices that you can define for this control unit. You can define a smaller address range, but 256 gives you the maximum number of devices per control unit. If you need to define more than 256 devices per channel, you need to define more control units for that channel.



```
                            Select Processor / CU   Row 12 of 15 More: <
Command ===> _______________________________________________ Scroll ===> CSR

Select processors to change CU/processor parameters, then press Enter.

Control unit number . . : 0010     Control unit type . . . : IQD

                CU   --------------Unit Address . Unit Range + ------------
/ Proc.CSSID Att ADD+ 1----- 2----- 3----- 4----- 5----- 6----- 7----- 8-----
_ SCZP401.0      __   ______ ______ ______ ______ ______ ______ ______ ______
_ SCZP401.1      __   00.256 ______ ______ ______ ______ ______ ______ ______
_ SCZP401.2      __   00.256 ______ ______ ______ ______ ______ ______ ______
_ SCZP401.3      __   ______ ______ ______ ______ ______ ______ ______ ______
```

Figure 2-14 Define the control unit address range

6. Verify that the control unit is attached to the channel path (see Figure 2-15), and then press Enter.



```
                            Select Processor / CU   Row 12 of 15 More: <     >
Command ===> _______________________________________________ Scroll ===> CSR

Select processors to change CU/processor parameters, then press Enter.

Control unit number . . : 0010     Control unit type . . . : IQD

                CU   --------------Unit Address . Unit Range + ------------
/ Proc.CSSID Att ADD+ 1----- 2----- 3----- 4----- 5----- 6----- 7----- 8-----
_ SCZP401.0      __   ______ ______ ______ ______ ______ ______ ______ ______
_ SCZP401.1  Yes __   00.256 ______ ______ ______ ______ ______ ______ ______
_ SCZP401.2  Yes __   00.256 ______ ______ ______ ______ ______ ______ ______
_ SCZP401.3      __   ______ ______ ______ ______ ______ ______ ______ ______
```

Figure 2-15 Verify that the control unit is attached to the processor

2.2.4 I/O device definitions

Follow these steps to define I/O devices:

1. Starting at the Control Unit List panel shown in Figure 2-16, select the control unit and press Enter to get to the I/O device list.

```
                              Control Unit List                     Row 1 of 1
Command ===> ______________________________________________ Scroll ===> CSR

Select one or more control units, then press Enter.  To add, use F11.

Processor ID . . : SCZP401    CSS ID . : 1   Channel path . : F5   IQD   SPAN

                           ---#---
/ CU   Type +       CUADD CSS MC  Serial-# + Description
s 0010 IQD                2       __________ ________________________________
```

Figure 2-16 Select a device list

2. From the I/O Device List panel shown in Figure 2-17, press **F11** to add a device.

```
                             I/O Device List
Command ===> ___________________________________________ Scroll ===> CSR

Select one or more devices, then press Enter. To add, use F11.

Control unit number  : 0010     Control unit type  . : IQD

  ----------Device------ --#--- --------Control Unit Numbers + --------
/ Number   Type +       CSS OS 1--- 2--- 3--- 4--- 5--- 6--- 7--- 8---
******************************* Bottom of data ******************************
```

Figure 2-17 Add a device

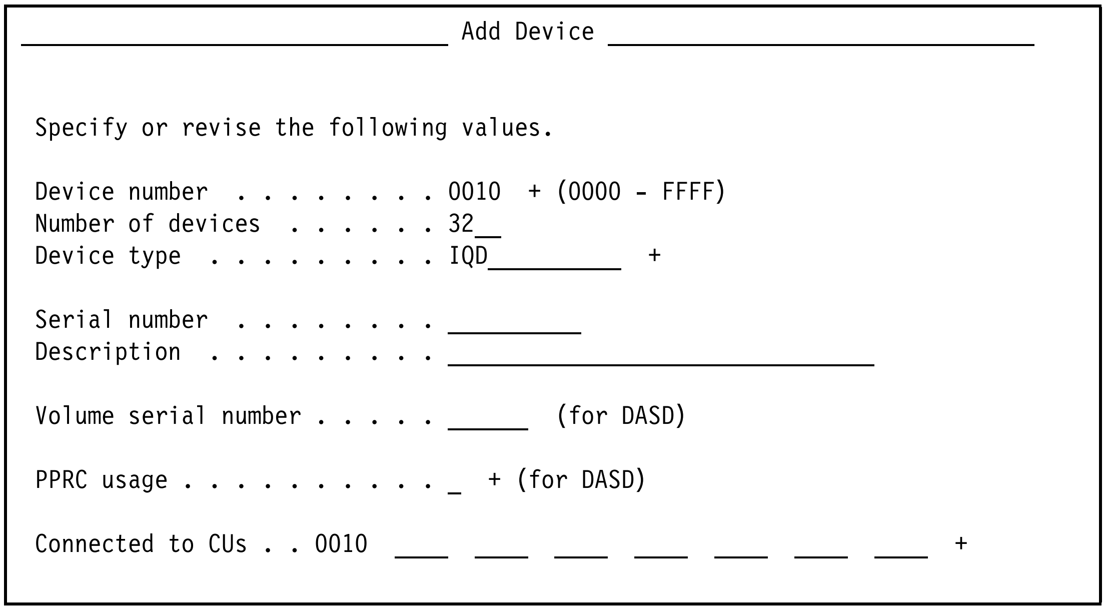
3. Enter the required information, as shown in Figure 2-18 on page 20, and then press Enter.

   In this example, the following values were set:

   – Device number `0010`
   – Number of devices `32`
   – Device type `IQD` (required for a HiperSockets device)

   > **Tip:** You need to define different device numbers for IQD interfaces on other IQD channels. You can choose any value for the range of device numbers, provided that it is not already used. You could, for example, choose `F500` to have the device number resemble the CHPID number.

4. Add a device, as shown in Figure 2-18.



```
________________________________ Add Device ________________________________


Specify or revise the following values.

Device number . . . . . . . . 0010  + (0000 - FFFF)
Number of devices  . . . . . . 32__
Device type  . . . . . . . . . IQD__________  +

Serial number  . . . . . . . . __________
Description  . . . . . . . . . ________________________________

Volume serial number . . . . . ______  (for DASD)

PPRC usage . . . . . . . . . . _  + (for DASD)

Connected to CUs . . 0010  ____  ____  ____  ____  ____  ____  ____  +
```

Figure 2-18 Define a device

Note: How many devices should be defined for z/OS LPARs?

The answer depends on the number of TCP/IP stacks running in the LPAR. Each TCP/IP stack requires one data device and IBM VTAM® requires two devices (read/write control). At most, eight TCP/IP stacks can be active on a z/OS LPAR, so 10 IQD devices is the maximum that can be used for a single HiperSockets channel on a single z/OS LPAR (one read control and one write control plus one data times eight TCP/IP stacks).

A minimum of three devices must be defined to an IQD control unit, and a maximum of 256 can be defined. 160 is the maximum number of IQD devices that can be online to a z/OS LPAR.

Devices can be shared between LPARs, so you can use the same range of 10 IQD devices per channel for all z/OS LPARs that have access to this IQD channel.

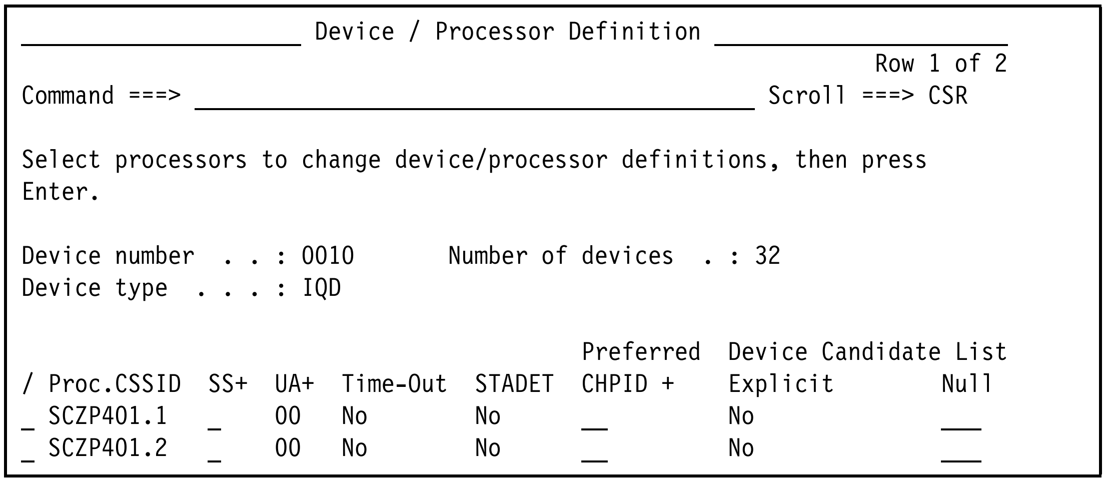
Note: How many devices should be defined for z/VM LPARs?

To connect z/VM guests directly to a HiperSockets channel, you need a different set of devices for each guest. Linux and z/VSE device drivers use three devices per HiperSockets interface. So if you intend to run Linux or z/VSE in your z/VM guests, define a number of IQD devices that is three times the maximum number of z/VM guests that you anticipate. You can add more devices to a channel using the dynamic channel path method at a later point in time.

See 2.3, “IBM z/VM definitions” on page 24 for more information about how to define HiperSockets interfaces for z/VM guests.

Devices can be shared between LPARs, so you can use the same range of IQD devices per channel for all z/VM LPARs that have access to this IQD channel.

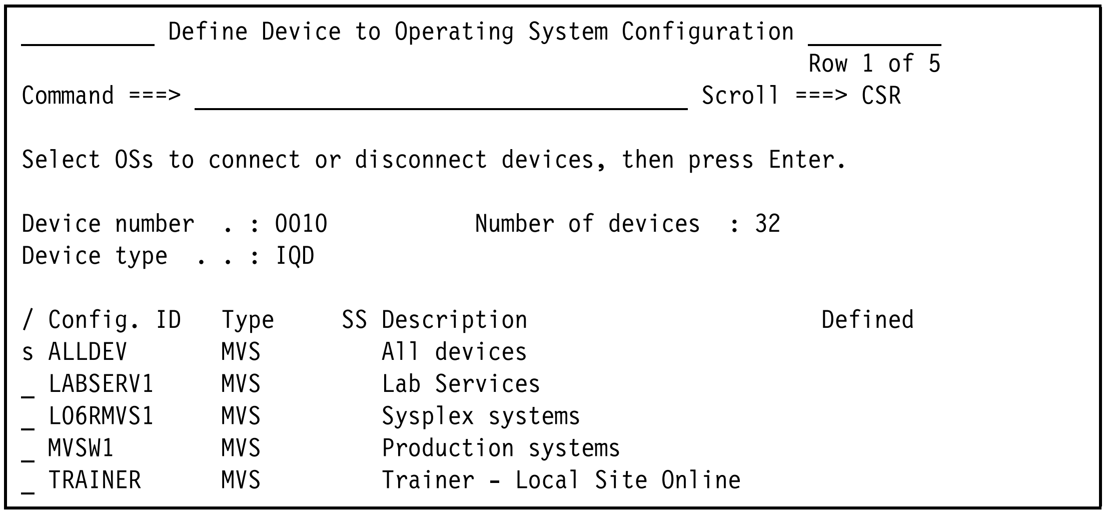
5. Set the unit address to 00 for the first device for both CSSs, as shown in Figure 2-19, and then press Enter.



```
_____________________ Device / Processor Definition ______________________
                                                                Row 1 of 2
Command ===> __________________________________________ Scroll ===> CSR

Select processors to change device/processor definitions, then press
Enter.

Device number  . . : 0010       Number of devices  . : 32
Device type  . . . : IQD

                                          Preferred  Device Candidate List
/ Proc.CSSID  SS+  UA+  Time-Out  STADET  CHPID +    Explicit       Null
_ SCZP401.1   _    00   No        No      __         No             ___
_ SCZP401.2   _    00   No        No      __         No             ___
```

Figure 2-19   Define the first device address

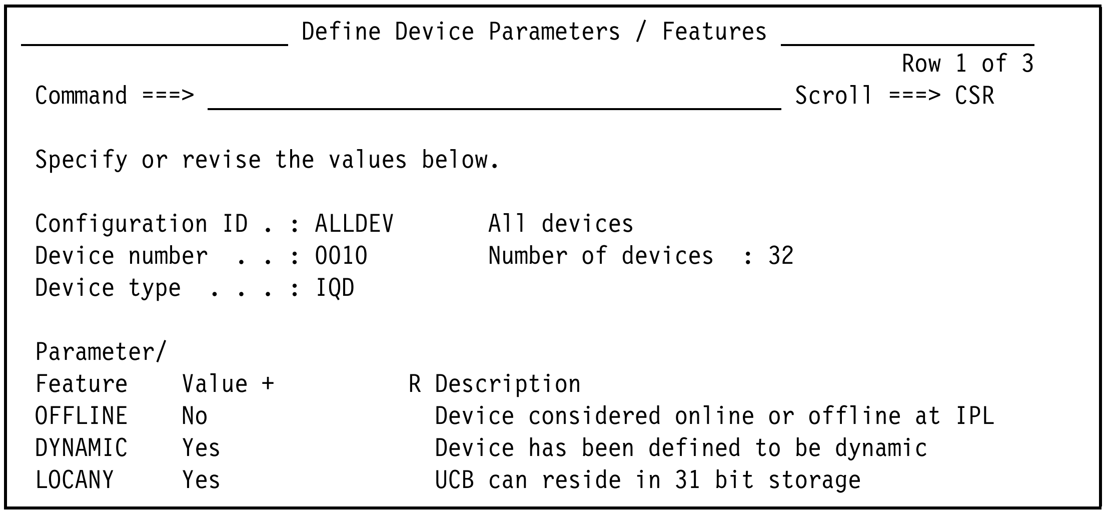
6. Select the OS to be defined for the device, as shown in Figure 2-20, and then press Enter.



```
__________ Define Device to Operating System Configuration __________
                                                           Row 1 of 5
Command ===> _____________________________________ Scroll ===> CSR

Select OSs to connect or disconnect devices, then press Enter.

Device number  . : 0010          Number of devices  : 32
Device type  . . : IQD

/ Config. ID   Type     SS Description                     Defined
s ALLDEV       MVS         All devices
_ LABSERV1     MVS         Lab Services
_ LO6RMVS1     MVS         Sysplex systems
_ MVSW1        MVS         Production systems
_ TRAINER      MVS         Trainer - Local Site Online
```

Figure 2-20   Select the operating system

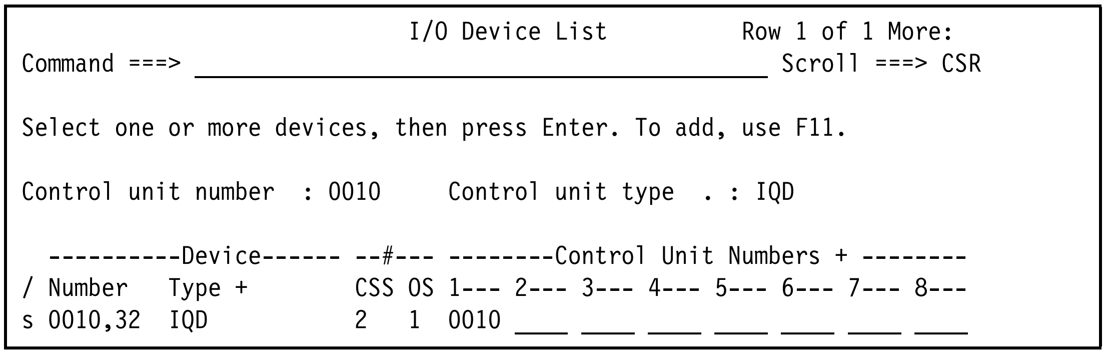
7. Select the OS parameters for the device, as shown in Figure 2-21, and then press Enter. Use the defaults.



```
 ____________________ Define Device Parameters / Features ___________________
                                                                  Row 1 of 3
 Command ===> ___________________________________________ Scroll ===> CSR

 Specify or revise the values below.

 Configuration ID . : ALLDEV       All devices
 Device number  . . : 0010         Number of devices  : 32
 Device type  . . . : IQD

 Parameter/
 Feature    Value +         R Description
 OFFLINE    No                Device considered online or offline at IPL
 DYNAMIC    Yes               Device has been defined to be dynamic
 LOCANY     Yes               UCB can reside in 31 bit storage
```

Figure 2-21 Select the operating system parameters

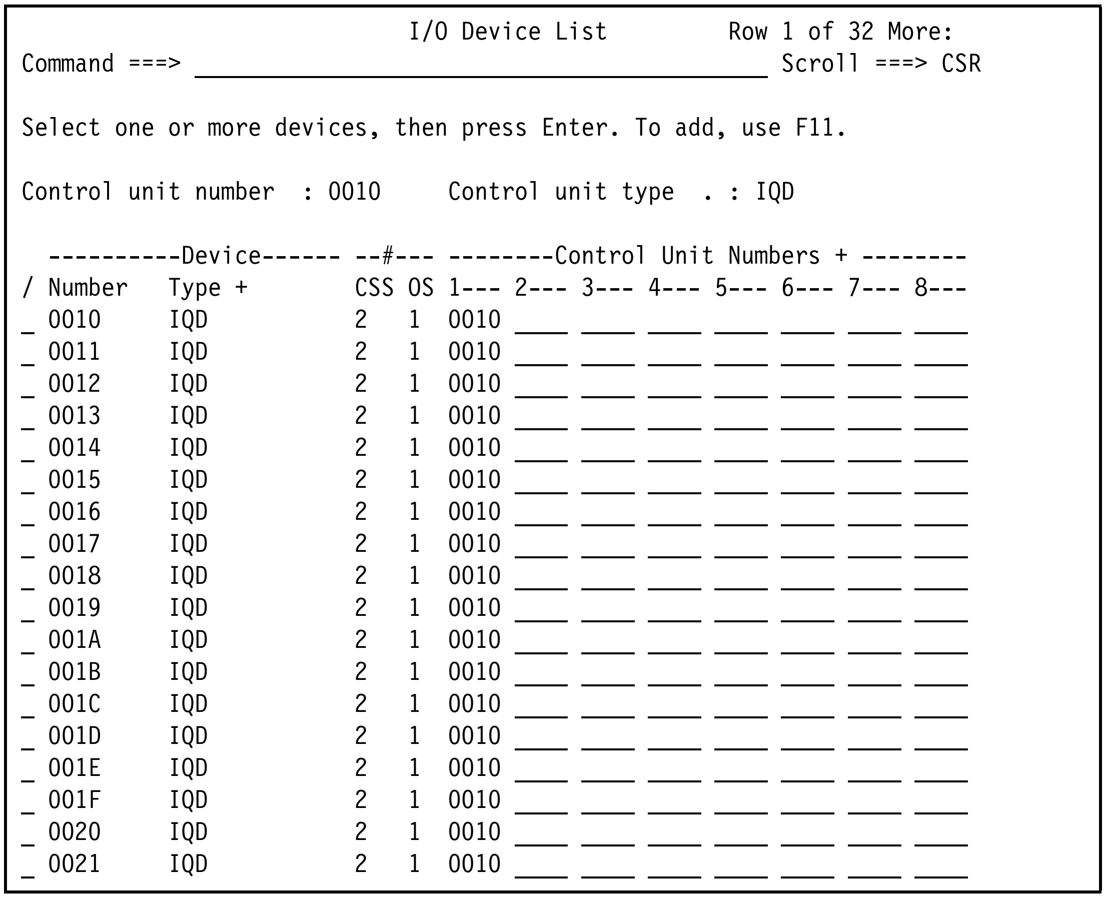
8. You are not defining the IQD devices as esoteric, so press Enter to bypass the Assign/Unassign Device to Esoteric panel. Select the devices to verify, as shown in Figure 2-22, and press Enter.



```
                             I/O Device List         Row 1 of 1 More:
Command ===> ___________________________________________ Scroll ===> CSR

Select one or more devices, then press Enter. To add, use F11.

Control unit number  : 0010     Control unit type  . : IQD

  ----------Device------ --#--- --------Control Unit Numbers + --------
/ Number   Type +       CSS OS 1--- 2--- 3--- 4--- 5--- 6--- 7--- 8---
s 0010,32  IQD          2   1  0010 ____ ____ ____ ____ ____ ____ ____
```

Figure 2-22 I/O device List

9. Next, the panel in Figure 2-23 displays, and you can verify your definitions.



```
                             I/O Device List        Row 1 of 32 More:
Command ===> __________________________________________ Scroll ===> CSR

Select one or more devices, then press Enter. To add, use F11.

Control unit number  : 0010     Control unit type  . : IQD

  ----------Device------ --#--- --------Control Unit Numbers + --------
/ Number   Type +        CSS OS 1--- 2--- 3--- 4--- 5--- 6--- 7--- 8---
_ 0010     IQD           2   1  0010 ____ ____ ____ ____ ____ ____ ____
_ 0011     IQD           2   1  0010 ____ ____ ____ ____ ____ ____ ____
_ 0012     IQD           2   1  0010 ____ ____ ____ ____ ____ ____ ____
_ 0013     IQD           2   1  0010 ____ ____ ____ ____ ____ ____ ____
_ 0014     IQD           2   1  0010 ____ ____ ____ ____ ____ ____ ____
_ 0015     IQD           2   1  0010 ____ ____ ____ ____ ____ ____ ____
_ 0016     IQD           2   1  0010 ____ ____ ____ ____ ____ ____ ____
_ 0017     IQD           2   1  0010 ____ ____ ____ ____ ____ ____ ____
_ 0018     IQD           2   1  0010 ____ ____ ____ ____ ____ ____ ____
_ 0019     IQD           2   1  0010 ____ ____ ____ ____ ____ ____ ____
_ 001A     IQD           2   1  0010 ____ ____ ____ ____ ____ ____ ____
_ 001B     IQD           2   1  0010 ____ ____ ____ ____ ____ ____ ____
_ 001C     IQD           2   1  0010 ____ ____ ____ ____ ____ ____ ____
_ 001D     IQD           2   1  0010 ____ ____ ____ ____ ____ ____ ____
_ 001E     IQD           2   1  0010 ____ ____ ____ ____ ____ ____ ____
_ 001F     IQD           2   1  0010 ____ ____ ____ ____ ____ ____ ____
_ 0020     IQD           2   1  0010 ____ ____ ____ ____ ____ ____ ____
_ 0021     IQD           2   1  0010 ____ ____ ____ ____ ____ ____ ____
```

Figure 2-23 Verify the device definitions

The IOCP statements generated by these HCD steps are shown in Example 2-1.

Example 2-1 IOCP statements

```
CHPID PATH=(CSS(1,2),F5),SHARED,                                 *
      PARTITION=((CSS(1),(A11,A13),(=)),(CSS(2),(A2E),(=))),    *
      TYPE=IQD
CNTLUNIT CUNUMBR=0010,PATH=((CSS(1),F5),(CSS(2),F5)),UNIT=IQD
IODEVICE ADDRESS=(0010,32),UNITADD=00,CUNUMBR=(0010),UNIT=IQD
```

10. Example 2-2 shows what the IOCDS statements might look like. Choose a **CHPARM** value for maximum frame sizes other than the default of 16 KB.

Example 2-2 CHPARM values for maximum frames sizes other than 16 KB

```
CHPID PATH=(CSS(1,2),F5),SHARED,CHPARM=40,                       *
      PARTITION=((CSS(1),(A11,A13),(=)),(CSS(2),(A2E),(=))),    *
     TYPE=IQD
CCHPID PATH=(CSS(1,2),F5),SHARED,CHPARM=80,                       *
      PARTITION=((CSS(1),(A11,A13),(=)),(CSS(2),(A2E),(=))),    *
     TYPE=IQD
```

```
CHPID PATH=(CSS(1,2),F5),SHARED,CHPARM=C0,                          *
      PARTITION=((CSS(1),(A11,A13),(=)),(CSS(2),(A2E),(=))),  *
      TYPE=IQD
```

2.2.5 Dynamic reconfiguration

Dynamic reconfiguration management is the ability to select a new I/O configuration during normal processing, without the need to complete a power-on reset (POR) of the hardware, or an initial program load (IPL) of the z/OS operating system. The ability of HCD to provide equivalent hardware and software I/O definitions, and to detect when they are not in sync, is essential for dynamic I/O reconfiguration management.

HCD compares both the old and the new configuration, and informs the hardware and software about the differences. You can add, delete, and modify definitions for channel paths, control units, and I/O devices without having to complete a POR or an IPL.

A system programmer or other authorized person can use the **Activate or verify configuration dynamically** HCD option, or the **`ACTIVATE IODF=xx`** operator command, to make changes to a running configuration. On the HCD panel, the administrator specifies the name and volume serial number, if applicable, for the production IODF.

2.2.6 References

The following list includes publications that provide additional information about the topic in this chapter:

- *System z Stand-Alone Input/Output Configuration Program User's Guide*, SB10-7152
- *z/OS Hardware Configuration Definition (HCD) Planning*, GA22-7525
- *z/OS Hardware Configuration Definition (HCD) User's Guide,* SC33-7988

2.3 IBM z/VM definitions

IBM z/VM provides a range of HiperSockets support for use by guest operating systems and the z/VM TCP/IP virtual machine (VM). HiperSockets devices are fully supported and managed by z/VM in the same manner as real devices. The use of HiperSockets for the z/VM TCP/IP VM is discussed in 3.4, "HiperSockets definitions for a z/VM host system" on page 44.

HiperSockets guest LAN

In z/VM, you can define a HiperSockets Guest LAN. A guest LAN is a LAN segment that is fully simulated by z/VM, and therefore has no connection to real HiperSockets CHPIDs defined in IOCDS. Guest LANs exist only inside a single z/VM LPAR. For more information about guest LANs, see *Introduction to the New Mainframe: z/VM Basics*, SG24-7316.

Attach z/VM guests to HiperSockets

Typically if a system administrator wants a guest to have connectivity to a HiperSockets network, the administrator will dedicate a set of HiperSockets devices to the guest. You use the **`ATTACH`** command or the DEDICATE directory statement to accomplish this. The results of the **`ATTACH`** command are transient, and will be undone when the guest logs off. By adding a DEDICATE statement to the guest's directory entry, the assignment becomes persistent.

2.3.1 Hardware assists

A complementary virtualization technology is available for HiperSockets that includes the following items:

- QDIO Enhanced Buffer-State Management (QEBSM) extensions are two new hardware instructions designed to help eliminate the resource usage of hypervisor interception.
- Host Page-Management Assist (HPMA) is an interface with the z/VM main storage management function designed to enable the hardware to assign, lock, and unlock page frames without z/VM hypervisor assistance.

These hardware assists enable a cooperating guest OS to initiate QDIO operations directly to the applicable channel, without interception by z/VM, thereby helping to provide additional performance improvements.

These hardware assists use support in the HiperSockets Firmware and z/VM, but also require support in the guest OSs. Support for QEBSM and HPMA is currently available in the following OSs:

- Linux for System z
- z/VSE

If the guest OS does not support or use QEBSM and HPMA, the z/VM host will emulate the virtual HiperSockets devices by passing all communication through to the real HiperSockets devices that are attached to the respective virtual devices.

2.3.2 Implementation example

Figure 2-24 shows the configuration that was used for the z/VM setup. The z/VM host system, VMLINUX1, is z/VM version 6 release 3 in a zEnterprise EC12 (zEC12) LPAR A2E with two Linux guests systems. LNXRH1 is Red Hat Enterprise Linux 6 (RHEL6.4). LNXSU1 is Novell SUSE Linux Enterprise Server 11 (SLES 11-SP2).

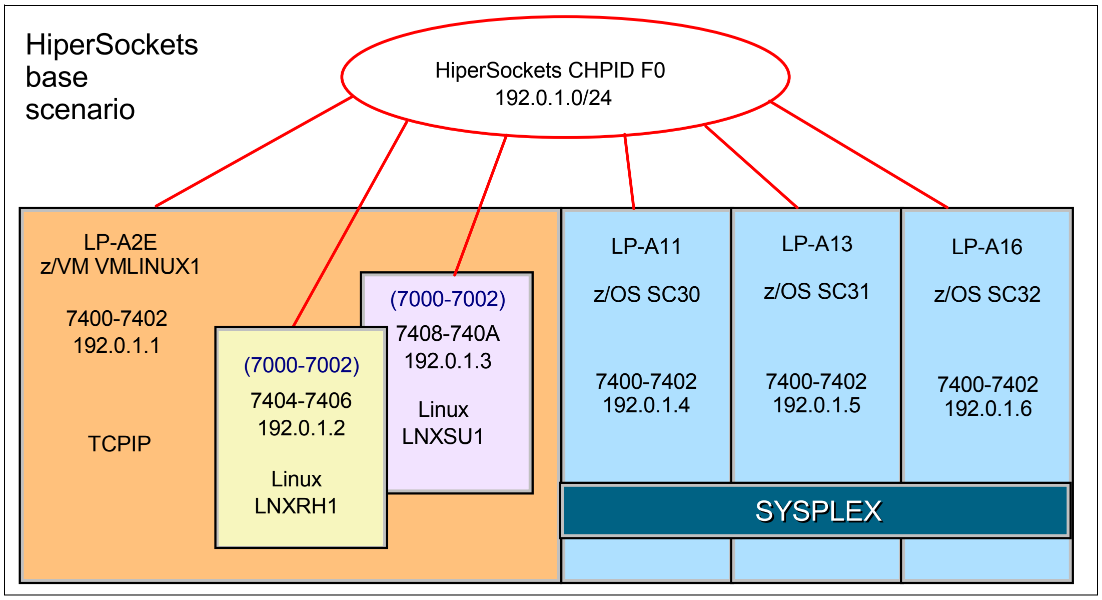

HiperSockets base scenario
HiperSockets CHPID F0 192.0.1.0/24
LP-A2E z/VM VMLINUX1
7400-7402 192.0.1.1
TCPIP
(7000-7002) 7404-7406 192.0.1.2 Linux LNXRH1
(7000-7002) 7408-740A 192.0.1.3 Linux LNXSU1
LP-A11 z/OS SC30 7400-7402 192.0.1.4
LP-A13 z/OS SC31 7400-7402 192.0.1.5
LP-A16 z/OS SC32 7400-7402 192.0.1.6
SYSPLEX

Figure 2-24 LPAR A12 z/VM with two Linux guest systems

HCD and IOCP definitions

See 2.2, “HCD definitions” on page 9 for how to define the HiperSockets devices to an LPAR.

Note: OSs in LPARs can share the same real device numbers (LP-A11, A13, and A16 in Figure 2-24 on page 25). However, you need to define separate real device numbers for OSs that run in VM guests in the same VM LPAR (LNXRH1 and LNXSU1 in Figure 2-24). However, you can define the same virtual device numbers for VM guests.

The following list includes a few notes about Figure 2-24 on page 25:

- For the z/VM TCP/IP stack, real device addresses 7400-7402 were used (setup is described in 3.4, “HiperSockets definitions for a z/VM host system” on page 44).
- For the RHEL guest, real devices 7404-7406 were connected as virtual devices 7000-7002.
- For the SLES guest, real devices 7408-740A were connected as virtual devices 7000-7002.

The CHPID F0 is defined as shared with 16 devices, and the default frame size of 16 KB. This results in a TCP/IP MTU size of 8 KB. The **CHPARM** parameter on the IOCP CHPID statement determines the frame size and subsequent MTU size values. See 2.2, “HCD definitions” on page 9 for additional information.

2.3.3 IBM z/VM I/O verification

To verify that the path to the HiperSockets devices is in an `ONLINE` status, use the **CP QUERY CHPID** command, as shown in Example 2-3.

Example 2-3 Query CHPID F0

```
q chpid f0

Path F0 online to devices 7400 7401 7402 7403 7404 7405 7406 7407
Path F0 online to devices 7408 7409 740A 740B 740C 740D 740E 740F
```

Display the HiperSockets devices to confirm that they are available to the z/VM system guests. See Example 2-4.

Example 2-4 Query devices

```
q 7400-740f

OSA  7400 FREE    , OSA  7401 FREE    , OSA  7402 FREE    , OSA  7403 FREE
OSA  7404 FREE    , OSA  7405 FREE    , OSA  7406 FREE    , OSA  7407 FREE
OSA  7408 FREE    , OSA  7409 FREE    , OSA  740A FREE    , OSA  740B FREE
OSA  740C FREE    , OSA  740D FREE    , OSA  740E FREE    , OSA  740F FREE
```

Example 2-5 shows the status of the specific paths for the HiperSockets devices (CHPID, availability, and status).

Example 2-5 Query paths for I/O devices

```
q paths 7400-7402

Device 7400, Status ONLINE
 CHPIDs to Device 7400 (PIM)  : F0
```

```
  Physically Available (PAM)  : +
  Online               (LPM)  : +
Device 7401, Status ONLINE
 CHPIDs to Device 7401 (PIM)  : F0
  Physically Available (PAM)  : +
  Online               (LPM)  : +
Device 7402, Status ONLINE
 CHPIDs to Device 7402 (PIM)  : F0
  Physically Available (PAM)  : +
  Online               (LPM)  : +
                       Legend     + Yes - No
```

2.3.4 IBM z/VM definitions for guest systems

To attach an IQD device to a guest, log on as the MAINT user and use the following command:

```
ATTACH xxx-yyy TO <guestname>
```

Where *xxx* and *yyy* are a three-device range of HiperSockets device addresses that will be attached to the specified guest. In this example, no virtual address range is defined, in which case z/VM will use the same values for the real addresses and the virtual addresses that are presented to the guest operating systems.

Some system programmers like to attach real hardware using virtual device addresses to have consistent hardware environments. This is particularly important in disaster recovery situations, where the hardware at the backup location might not be the same as the primary. By using virtual device addresses, you can make the backup system look similar to the original, enabling you to IPL your operating systems.

Temporary definitions

To dynamically attach the device addresses to a running guest system, use the **CP ATTACH** command from a privileged z/VM user ID. This is a temporary definition that will be lost when the guest is logged off, or when the z/VM system is restarted.

The following example shows the syntax for the **CP ATTACH** command:

```
CP ATTACH <real_address> <guest_id> <virtual_address>.
```

Use the following z/VM commands to allocate I/O devices to your running guest Linux systems:

```
attach 7404 lnxrh1 7000
attach 7405 lnxrh1 7001
attach 7406 lnxrh1 7002
attach 7408 lnxsu1 7000
attach 7409 lnxsu1 7001
attach 740A lnxsu1 7002
```

Displaying the HiperSockets devices, as shown in Example 2-6, provides information about how the devices are allocated to the VMs for TCP/IP and the two Linux guests.

Example 2-6 Query I/O devices

```
q 7400-740f

OSA  7400 FREE    , OSA  7401 FREE    , OSA  7402 FREE    , OSA  7403 FREE
OSA  7404 ATTACHED TO LNXRH1   7000 DEVTYPE HIPER      CHPID F0 IQD
OSA  7405 ATTACHED TO LNXRH1   7001 DEVTYPE HIPER      CHPID F0 IQD
OSA  7406 ATTACHED TO LNXRH1   7002 DEVTYPE HIPER      CHPID F0 IQD
OSA  7407 FREE
OSA  7408 ATTACHED TO LNXSU1   7000 DEVTYPE HIPER      CHPID F0 IQD
OSA  7409 ATTACHED TO LNXSU1   7001 DEVTYPE HIPER      CHPID F0 IQD
OSA  740A ATTACHED TO LNXSU1   7002 DEVTYPE HIPER      CHPID F0 IQD
OSA  740B FREE    , OSA  740C FREE    , OSA  740D FREE    , OSA  740E FREE
OSA  740F FREE
```

Note that the allocated devices show up as an OSA-type device driver. This is normal because the device driver is the same for QDIO and iQDIO in z/VM. However, the DEVTYPE is identified as HIPER rather than OSA, because it is in an OSA device.

Permanent definitions

To make the change permanent, you need to edit the guest's directory entry in the `USER DIRECT` file, and add a line similar to the following statement for each of the three HiperSockets device addresses:

```
DEDICATE <virtdev> <realdev>
```

In this case, <*virtdev*> is the virtual device address that you want the real device (<*realdev*>) address attached at. This is done for all guest systems running in a z/VM host.

The following z/VM user directory statements were used to attach the HiperSockets I/O devices permanently to the RHEL guest LNXRH1:

```
DEDICATE 7000 7404
DEDICATE 7001 7405
DEDICATE 7002 7406
```

The following statements were used for the SUSE guest LNXSU1:

```
DEDICATE 7000 7408
DEDICATE 7001 7409
DEDICATE 7002 740A
```

Now you need to bring your modified USER DIRECT online by entering the following command:

```
wodirectxa USER DIRECT
```

3

Software configurations for HiperSockets

This chapter introduces the basic software configurations that were used for this book. It provides you with examples showing you how to set up elementary HiperSockets configurations for your operating system (OS). The following OSs support HiperSockets:

- z/OS
- z/VM
- z/VSE
- Linux on System z

3.1 Test configuration

Figure 3-1 shows the HiperSockets base configuration used throughout the example scenarios in this book. Based on this configuration, implement the new functions and place the definition for each system in the specific session. Additional setup for specific scenarios, such as DYNAMICXCF, HiperSockets Accelerator, and so on, are documented in the relevant sections.

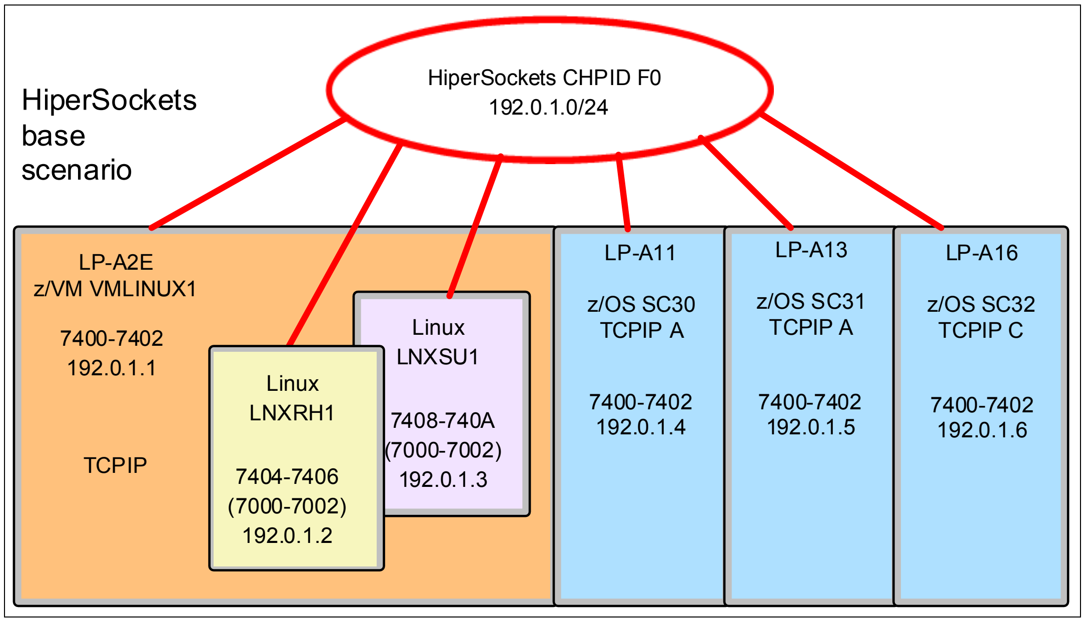


Figure 3-1 HiperSockets base configuration scenario

Use four logical partitions (LPARs) to set up and verify HiperSockets support with z/OS, z/VM, and Linux. Because HiperSockets are shared among LPARs, you must define the channel-path identifiers (CHPIDs) as shared in the hardware definitions. For HiperSockets F0, use the IP network address `192.0.1.0/24`. This configuration does not include virtual IP addresses (VIPAs), which are also supported.

The LPARs are configured in the following ways:

- In LPAR A2E, there are two Linux systems:
  - A Red Hat Enterprise Linux (RHEL) instance named LNXRH1
  - A SUSE Linux Enterprise Server (SLES) instance named LNXSU1

  They are running as guests under z/VM, along with the z/VM system (VMLINUX1). The Linux systems use DEDICATE and control their HiperSockets connections directly. Each of the two systems has one interface with HiperSockets through CHPID F0.

  For LNXRH1, allocate real addresses 7404-7406 that map virtual unit addresses 7000-7002 to the three I/O devices. For LNXSU1, use the next available unit addresses in a similar way, in ascending order.
- LPAR A11 runs a z/OS image (SC30). This image connects to HiperSockets F0 by 7400-7402 devices.
- LPAR A13 runs a z/OS image (SC31). This image connects to HiperSockets F0 by 7400-7402 devices.
- LPAR A16 runs a z/OS image (SC32). This image connects to HiperSockets F0 by 7400-7402 devices.

Table 3-1 shows the details of the test HiperSockets configuration.

Table 3-1 Details of the test HiperSockets configuration

LPAR name	Environment	System name	CHPID	Device address	IP address
A2E	z/VM	VMLINUX1	F0	7400-7402	192.0.1.1
A2E	Linux under z/VM	LNXRH1	F0	7404-7406	192.0.1.2
A2E	Linux under z/VM	LNXSU1	F0	7408-740A	192.0.1.3
A11	z/OS sysplex	SC30	F0	7400-7402	192.0.1.4
A13	z/OS sysplex	SC31	F0	7400-7402	192.0.1.5
A16	z/OS sysplex	SC32	F0	7400-7402	192.0.1.6

Tip: Each LPAR can use the same device addresses. When multiple Transmission Control Protocol/Internet Protocol (TCP/IP) stacks are present in a single LPAR, then the device addresses must be unique to each TCP/IP stack, as shown for LPAR A2E in Table 3-1.

3.2 HiperSockets in z/OS

IBM z/OS can use HiperSockets in two different ways:

1. Directly as visible HiperSockets interfaces, as described in this section.
2. As part of a Dynamic cross-system coupling facility (XCF) network. This is described in 3.3, "DYNAMICXCF HiperSockets implementation" on page 36.

Restriction: The z/OS TCP/IP stack does not support HiperSockets and DYNAMICXCF traffic over the same HiperSockets CHPID. TCP/IP stacks that have both types of traffic must use two separate CHPIDs.

3.2.1 HiperSockets implementation environment

The first implementation configured a HiperSockets channel. One TCP/IP stack on each LPAR (SC30, SC31, and SC32) was configured to use HiperSockets CHPID F0, as shown in Figure 3-2.

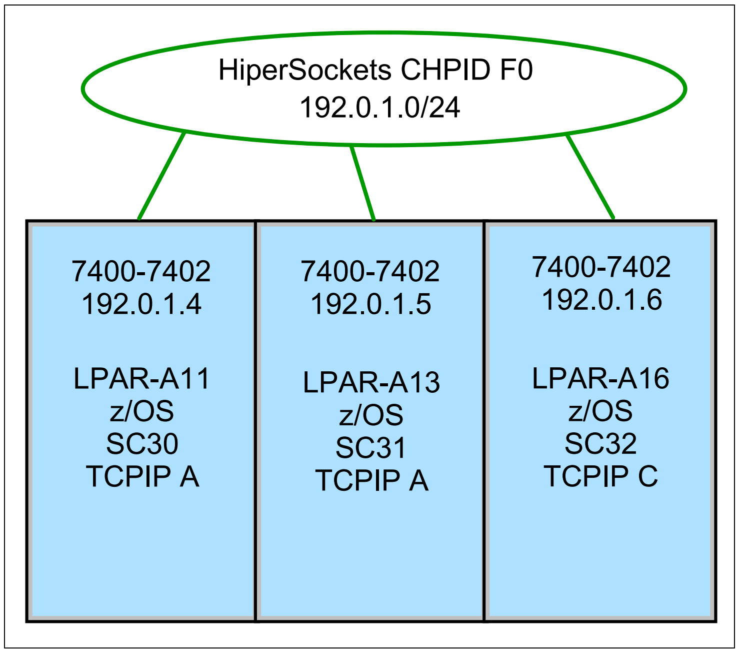


Figure 3-2 HiperSockets implementation

3.2.2 Implementation steps

Follow these steps to implement HiperSockets:

- Define the HiperSockets channel, control unit, and device.
- Make the required configuration changes to the TCP/IP profile.
- Start the TCP/IP stacks.

The following examples use the definitions shown in Figure 3-2. The HiperSockets channel and devices must be online before starting the first TCP/IP stack configured to use the devices. For this example, CHPID F0 and device numbers 7400, 7401, and 7402 must be active in z/OS before starting the first TCP/IP stack that will use HiperSockets channel F0.

Important: After the 160th IQD device is brought online to an LPAR, subsequent attempts to vary an IQD device online in z/OS will fail and generate the following message:

```
IOS577I IQD INITIALIZATION FAILED, COMPLETION TABLE FULL
```

3.2.3 No IBM VTAM setup for HiperSockets

No Virtual Telecommunications Access Method (VTAM) setup is required. VTAM dynamically creates the necessary Transport Resource List Element (TRLE). When the first TCP/IP stack is started, VTAM will build a single multipath channel (MPC) group using the subchannel devices associated with the internal queued direct communication (IQD) CHPID. VTAM will use two subchannel devices (for the read and write control devices) and 1 - 8 subchannel devices for the data device.

3.2.4 TCP/IP profile setup for HiperSockets

HiperSockets interfaces are defined in the TCP/IP profile using the INTERFACE statement introduced in z/OS Communications Server V2R1. Additionally, specify BEGINROUTES and START statements. If required, BSDROUTINGPARMS can be specified (it is not specified in this implementation example).

Update the profile member for each TCP/IP stack following the configuration rules. The customization for TCPIPA on LPAR SC30 is shown in Example 3-1.

Example 3-1 TCPIPA profile for LPAR SC30 for HiperSockets

```
INTERFACE HIPERLF0          1
    DEFINE IPAQIDIO
    IPADDR 192.0.1.4/24
    CHPID F0

BEGINROUTES
ROUTE 192.0.1.0/24 = HIPERLF0 MTU 8192 2
ENDROUTES

START HIPERLF0    3
```

1 z/OS Communications Server V2R1 introduced the INTERFACE statement for the definitions of HiperSockets. You can still use the DEVICE and LINK statements. The syntax for DEVICE and LINK is described in *IBM z/OS V2R1 Communications Server TCP/IP Implementation Volume 1: Base Functions Connectivity and Routing*, SG24-8096.

The INTERFACE name can be user-defined. This example uses HIPERLF0.

When a TCP/IP device is started, the IP address contained in the TCP/IP stack's INTERFACE statement is registered in the HiperSockets firmware's IP address lookup table. The z/OS TCP/IP stack becomes part of HiperSockets F0 with the IP address 192.0.1.4.

2 The ROUTE entry of the BEGINROUTES statement is specified as `ROUTE` *destination gateway_addr link_name* `MTU` *mtu_size*. The destination ought to specify a valid host, network or subnetwork. Specifying the Equal sign (=) for the gateway_addr means packets are routed directly to destinations on that network or host. The *link_name* must match the *link_name* specified on the `LINK` statement.

Because you are using static routes in your environment, you have to define a Route statement. If you are using dynamic routing (RIP or OSPF), omit this statement. Also note that you specified an maximum transmission unit (MTU) size of 8 KB to accommodate the maximum frame size (MFS) of the IQD CHPID F0, which you defined in the hardware configuration definition (HCD) with the default value (16 KB).

3 The START statement is specified as START *interface_name*. The START statement is used to start an interface. The interface_name must match the interface_name specified on the INTERFACE statement for the HiperSockets CHPID.

Optionally, a **`VARY TCPIP,`*`tcpipproc`*`,START,`*`interface_name`*** command can be issued or the START statement can be used in a **`VARY TCPIP,`*`tcpipproc`*`,OBEYFILE,`*`datasetname`*** command to start the wanted interface.

The TCP/IP profile for TCPIPA was also updated on LPAR SC31and TCPIPC for LPAR SC32. All parameters are the same as defined for LPAR SC30, except for the IP address, as shown in Example 3-2 for LPAR SC31 and Example 3-3 for LPAR SC32.

Example 3-2 SC31 TCPIPA INTERFACE statement

```
INTERFACE HIPERLF0          1
    DEFINE IPAQIDIO
    IPADDR 192.0.1.5/24
    CHPID F0
```

Example 3-3 SC32 TCPIPC INTERFACE statement

```
INTERFACE HIPERLF0          1
    DEFINE IPAQIDIO
    IPADDR 192.0.1.6/24
    CHPID F0
```

3.2.5 Verification of the HiperSockets configuration

This section shows the display commands used to verify the HiperSockets configuration.

TCP/IP startup

Start the TCP/IP stack TCPIPA on LPAR SC30. Because you specified a START parameter in the tcp profile data set for the HiperSockets interface HIPERLF0, the initialization will occur when you start the TCPIPA stack. Successful initialization of the HiperSockets interface can be verified by checking the SYSLOG messages when the IP stack is started, as shown in Example 3-4.

Example 3-4 TCPIPA startup message for HiperSockets device

```
EZZ4340I INITIALIZATION COMPLETE FOR INTERFACE HIPERLF0
```

Also start the TCP/IP stacks on LPAR SC31 and SC32.

Verify TCP/IP device and link status

To verify the status of interfaces defined to the TCP/IP stack, use the **D TCPIP,*procname*,NETSTAT,DEVLINKS** command to request NETSTAT information. The output for this example is shown in Example 3-5.

Example 3-5 Verify interface information HiperSockets

```
D TCPIP,TCPIPA,NETSTAT,DEVL
INTFNAME: HIPERLF0         INTFTYPE: IPAQIDIO   INTFSTATUS: READY  1
    TRLE: IUTIQ4F0  DATAPATH: 7402     DATAPATHSTATUS: READY       2
    CHPID: F0
    IPBROADCASTCAPABILITY: NO
    ARPOFFLOAD: YES                  ARPOFFLOADINFO: YES
    CFGMTU: NONE                     ACTMTU: 8192                  3
    IPADDR: 192.0.1.4/24
    VLANID: NONE
    READSTORAGE: GLOBAL (2048K)
    SECCLASS: 255                    MONSYSPLEX: NO
    IQDMULTIWRITE: ENABLED (ZIIP)
  MULTICAST SPECIFIC:
    MULTICAST CAPABILITY: YES
    GROUP            REFCNT       SRCFLTMD
    -----            ------       --------
    224.0.0.1        0000000001
```

Verify that the interface name and type match your tcp profile definitions, and that the device is in a ready status 1. Verify that the datapath is in a ready status 2. Also verify that the MTU definition matches what was defined in the tcp profile 3.

Verify VTAM TRLE

VTAM will automatically create a TRLE (IUTIQ4F0) 1 for the HiperSockets channel F0 as part of the transport resource list major node (ISTTRL) 2, as shown in the output of the **D NET,TRL** command in Example 3-6.

Example 3-6 D NET,TRL display for HiperSockets example (partial output)

```
D NET,TRL
IST097I DISPLAY ACCEPTED
IST350I DISPLAY TYPE = TRL 013
IST1954I TRL MAJOR NODE = ISTTRL 2
IST1314I TRLE = IUTIQ4F0 1 STATUS = ACTIV       CONTROL = MPC
```

For further information about the TRLE, issue **D NET,TRL TRLE**. The results are shown in Example 3-7.

The display shows that for the IUTIQ4F0 TRLE, devices 7400 and 7401 are used for read and write control 1. For TCPIPA, data device 7402 is assigned 2. When a datapath device is active, it indicates which TCP/IP stack is using it in message IST1717I (ULPID = *jobname*) 3. For example, TCPIPA (jobname) is using device 7402. Note that if you have multiple TCP/IP stacks running in the same LPAR and using CHPID F0, datapath devices 7403 through 7409 are used 4. No additional read/write devices are needed.

Example 3-7 Verify VTAM TRLE for HiperSockets

```
D NET,TRL,TRLE=IUTIQ4F0
IST097I DISPLAY ACCEPTED
IST075I NAME = IUTIQ4F0, TYPE = TRLE
IST1954I TRL MAJOR NODE = ISTTRL
IST486I STATUS= ACTIV, DESIRED STATE= ACTIV
IST087I TYPE = LEASED            , CONTROL = MPC , HPDT = YES
IST1715I MPCLEVEL = QDIO       MPCUSAGE = SHARE
IST2263I PORTNAME =            PORTNUM =   0   OSA CODE LEVEL = *NA*
IST2337I CHPID TYPE = IQD      CHPID = F0  PNETID = **NA**
IST2319I IQD NETWORK ID = 0716
IST1577I HEADER SIZE = 4096 DATA SIZE = 16384 STORAGE = ***NA***
IST1221I WRITE DEV = 7401 STATUS = ACTIVE     STATE = ONLINE        1
IST1577I HEADER SIZE = 4092 DATA SIZE = 0 STORAGE = ***NA***
IST1221I READ  DEV = 7400 STATUS = ACTIVE     STATE = ONLINE        1
IST924I -------------------------------------------------------------
IST1221I DATA  DEV = 7402 STATUS = ACTIVE     STATE = N/A           2
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1717I ULPID = TCPIPA ULP INTERFACE = HIPERLF0                    3
IST2309I ACCELERATED ROUTING ENABLED
IST2331I QUEUE   QUEUE     READ             QUEUE
IST2332I ID      TYPE      STORAGE          STATUS
IST2205I ------  --------  ---------------  ----------------------
```

```
IST2333I RD/1    PRIMARY   2.0M(126 SBALS)  ACTIVE
IST2305I NUMBER OF DISCARDED INBOUND READ BUFFERS = 0
IST2386I NUMBER OF DISCARDED OUTBOUND WRITE BUFFERS = 0
IST1757I PRIORITY1: UNCONGESTED PRIORITY2: UNCONGESTED
IST1757I PRIORITY3: UNCONGESTED PRIORITY4: UNCONGESTED
IST1801I UNITS OF WORK FOR NCB AT ADDRESS X'23DF7010'
IST1802I P1 CURRENT = 0 AVERAGE = 1 MAXIMUM = 1
IST1802I P2 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST1802I P3 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST1802I P4 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST924I -------------------------------------------------------------
IST1221I DATA  DEV = 7403 STATUS = RESET      STATE = N/A          4
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST924I -------------------------------------------------------------
IST1221I DATA  DEV = 7404 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST924I -------------------------------------------------------------
IST1221I DATA  DEV = 7405 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST924I -------------------------------------------------------------
IST1221I DATA  DEV = 7406 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST924I -------------------------------------------------------------
IST1221I DATA  DEV = 7407 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST924I -------------------------------------------------------------
IST1221I DATA  DEV = 7408 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST924I -------------------------------------------------------------
IST1221I DATA  DEV = 7409 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST924I -------------------------------------------------------------
IST314I END
```

Verify HiperSockets connections

The simplest and most straightforward verification is to ping the HiperSockets interface from one LPAR to another. Example 3-8 shows the result of one successful ping.

Example 3-8 HiperSockets connectivity test

```
Ping from TCPIPA on LPAR SC30 to LPAR SC31
ping 192.0.1.5 (tcp tcpipa) 1
CS V2R1: Pinging host 192.0.1.5
Ping #1 response took 0.000 seconds.
```

3.3 DYNAMICXCF HiperSockets implementation

From an IP topology perspective, DYNAMICXCF establishes fully meshed IP connectivity to all other z/OS TCP/IP stacks in the Sysplex. You only need one endpoint specification in each stack for fully meshed connectivity to all other stacks in the Sysplex. When a new stack gets started, Dynamic XCF connectivity is automatically established.

Remember: Only one dynamic XCF network is supported per Sysplex.

Dynamic XCF uses Sysplex Sockets support, enabling the stacks to communicate with each other and exchange information, such as VTAM CPNAMEs, IBM MVS™ SYSCLONE value, and IP addresses. The dynamic XCF definition for IPv4 is activated by coding the IPCONFIG DYNAMICXCF parameter in the TCP/IP profile. For IPv6, use the IPCONFIG6 DYNAMICXCF parameter to configure dynamic XCF definitions.

Dynamic XCF creates definitions for the DEVICE, LINK, HOME, and BSDROUTINGPARMS statements, and the START statement, dynamically. When activated, the dynamic XCF devices and links appear to the stack as though they had been defined in the TCP/IP profile. They can be displayed using standard commands, and they can be stopped and started.

During TCP/IP initialization, the stack joins the XCF group, ISTXCF, through VTAM. When other stacks in the group discover the new stack, the definitions are created automatically, the links are activated, and the remote IP address for each link is added to the routing table. After the remote IP address has been added, IP traffic can flow across one of the following interfaces:

- IUTSAMEH (in the same LPAR)
- HiperSockets (in the same server)
- XCF signaling (different server, either using the coupling facility link or a channel-to-channel (CTC) connection)

Figure 3-3 shows the Dynamic XCF support implementation.

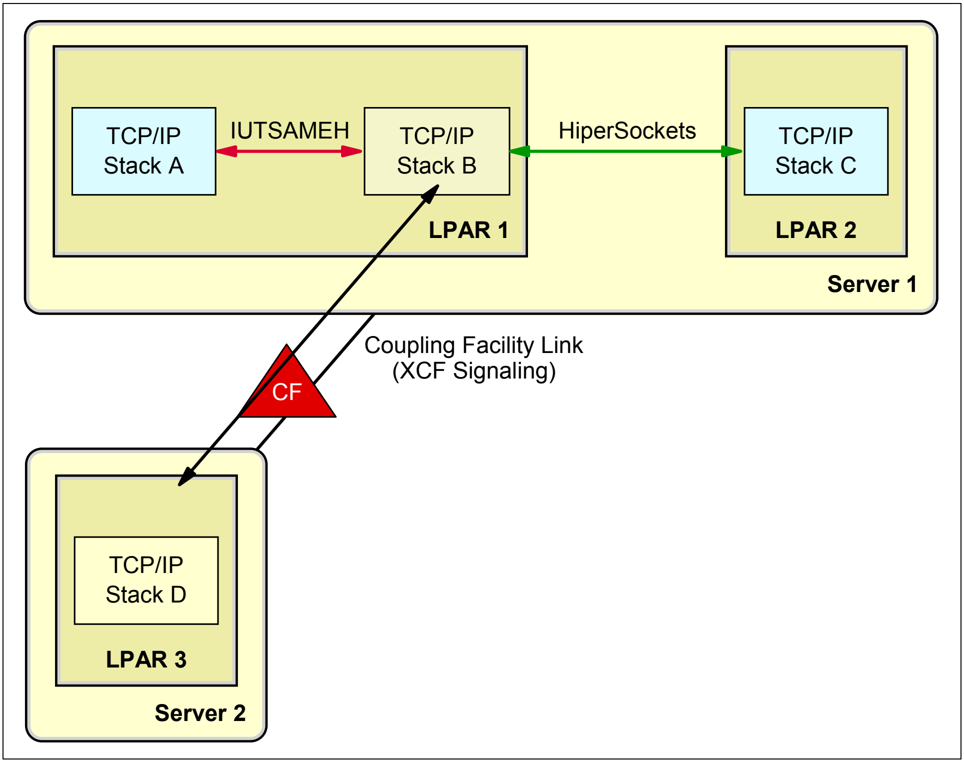


Figure 3-3 Dynamic XCF support

Tip: HiperSockets DYNAMICXCF supports dividing a Sysplex into multiple subplexes. See Appendix B, “IBM z/OS Sysplex subplexing and HiperSockets” on page 151 for more information about this topic.

3.3.1 HiperSockets DYNAMICXCF connectivity

The z/OS images in the same server with DYNAMICXCF coded will use the HiperSockets DYNAMICXCF connectivity rather than the standard XCF connectivity, under the following conditions:

- ► The TCP/IP stacks must be on the same server.
- ► For the DYNAMICXCF HiperSockets device (IUTIQDIO), the stacks must be using the same IQD CHPID, even with different channel subsystems (spanning).
- ► The stacks must be configured (through HCD) to use HiperSockets.
- ► The initial HiperSockets activation must complete successfully.

When an IPv4 DYNAMICXCF HiperSockets device and link are created and successfully activated, a subnetwork route is created across the HiperSockets link. The subnetwork is created by using the DYNAMICXCF IP address and mask.

This enables any LPAR in the same server to be reached, even ones that are not in the sysplex. To achieve this functionality, the LPAR that is outside of the sysplex environment must define at least one IP address for the HiperSockets endpoint that is in the subnetwork defined by the DYNAMICXCF IP address and mask.

When multiple stacks exist in the same LPAR that supports HiperSockets, both IUTSAMEH and HiperSockets links or interfaces will coexist. In this case, it is possible to transfer data across either link. Because IUTSAMEH links have better performance, it is always better to use them for intra-stack communication. A host route will be created by DYNAMICXCF processing across the IUTSAMEH link, but not across the HiperSockets link.

3.3.2 DYNAMICXCF implementation environment

For this test implementation, configure a single TCP/IP stack on LPARs SC30, SC31, and SC32 for DYNAMICXCF connectivity using CHPID F3, as shown in Figure 3-4 on page 39. The test environment was configured based on the following requirements:

- ► All z/OS hosts must belong to the same sysplex.
- ► VTAM must have XCF communications enabled by specifying XCFINIT=YES or XCFINIT=DEFINE as a VTAM startup parameter, or by activating the VTAM XCF local SNA major node, ISTLSXCF. For details about configuration, see *z/OS Communications Server: SNA Network Implementation*, SC31-8777.
- ► DYNAMICXCF must be specified in the TCP/IP profile of each stack.
- ► The IQD CHPID being used for the DYNAMICXCF device cannot be the same CHPID that is used for direct HiperSockets communication (which is IQD CHPID F0 in this environment). To avoid this, a VTAM start option, the IQDCHPID parameter, is used to identify which IQD CHPID will be used by DYNAMICXCF (which is IQD CHPID F3 in this environment).

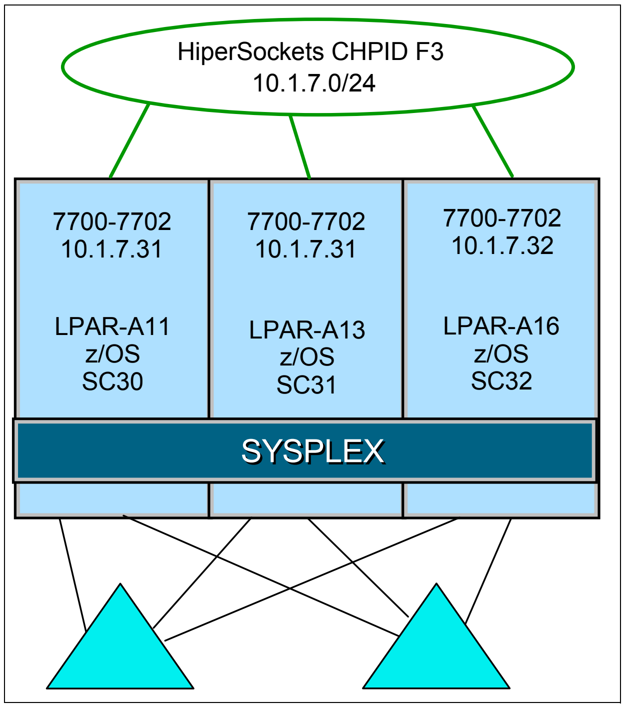


Figure 3-4 DYNAMICXCF implementation environment

Table 3-2 is the lookup table for this configuration.

Table 3-2 IP address lookup table for HiperSockets DYNAMICXCF in CHPID F3

IP address	LPAR number	System name	Device addresses
10.1.7.11	A11	SC30	7700-7702
10.1.7.32	A13	SC31	7700-7702
10.1.7.31	A16	SC32	7700-7702

All sysplex systems can connect to each other.

3.3.3 Implementation steps

Take the following steps to implement DYNAMICXCF HiperSockets:

1. Define the HiperSockets channel, control unit, and device.
2. Configure VTAM to specify XCFINIT=YES and to specify the IQDCHPID parameter.
3. Add the DYNAMICXCF statement to the TCP/IP profile.
4. Start the TCP/IP stacks.

The HiperSockets channel and devices must be online before starting the first TCP/IP stack configured to use the devices. For this example, CHPID F3 and device numbers 7700, 7701, and 7702 must be active to z/OS before starting the first TCP/IP stack configured to use HiperSockets channel F3.

3.3.4 VTAM configuration for DYNAMICXCF

To enable DYNAMICXCF over HiperSockets, the VTAM start options XCFINIT=YES and IQDCHPID must be specified. Modify the VTAM start options (ATCSTRxx) to add the required parameters. Add the XCFINIT=YES line to the VTAM start options 1, as shown in Example 3-9.

Example 3-9 VTAM XCFINIT statement for DYNAMICXCF

```
XCFINIT=YES 1
```

This tells VTAM to dynamically build the ISTLSXCF link station major node, and to establish connections to other VTAMs in the sysplex. Add the IQDCHPID statement to the VTAM start options 2, as shown in Example 3-10.

Example 3-10 VTAM IQDCHPID statement for DYNAMICXCF

```
IQDCHPID=F3 2
```

This tells VTAM to use CHPID F3 as the DYNAMICXCF link for the sysplex. In this configuration, multiple HiperSockets CHPIDs are defined (F0,F3). For this reason, the definition of IQDCHPID is mandatory to ensure iQDIO with the appropriate CHPID will be selected. Otherwise, if multiple IQD CHPIDs are defined, VTAM will use the first acceptable HiperSockets CHPID detected (having at least three subchannel devices defined).

We suggest specifying the IQDCHPID statement to reserve the IQD channel for DYNAMICXCF use when multiple HiperSockets are defined. VTAM will dynamically create a IUTIQDIO TRLE for the DYNAMICXCF HiperSockets link.

In this configuration, the VTAM START definitions are the same for LPAR SC31 and LPAR SC32.

3.3.5 TCP/IP configuration for DYNAMICXCF

In order for a TCP/IP stack to use the DYNAMICXCF HiperSockets connection, the DYNAMICXCF parameter must be specified on the IPCONFIG statement in the tcp profile. 1 The DYNAMICXCF parameter is specified as DYNAMICXCF *ipv4_address subnet_mask cost_metric*.

Specifying DYNAMICXCF will enable the creation of the DEVICE, LINK, HOME, BSDROUTINGPARMS statements, and (for IUTIQDIO device) the START statement. Modify the IPCONFIG statement to add the DYNAMICXCF parameter with an IP address, a subnet mask, and a cost metric in the TCP/IP profile (PROFF30) for SC30, as shown in Example 3-11 1.

Example 3-11 IPCONFIG statement for DYNAMICXCF connection for LPAR SC30

```
IPCONFIG DYNAMICXCF 10.1.7.11 255.255.255.0 1     1
```

This IP address (10.1.7.11) is the interface with HiperSockets CHPID F3. This address will be automatically appended to the TCP/IP profile HOME list. At TCP/IP stack initialization time, the stack is registered with its IP address 10.1.7.11 in the IP address lookup table, as shown in Table 3-2 on page 39.

The definitions for LPAR SC31 and LPAR SC32 are identical to the definition for SC30, except for the IP address of (10.1.7.32) for SC31, as shown in Example 3-12, and (10.1.7.31) for SC32, as shown in Example 3-13.

Example 3-12 IPCONFIG statement for DYNAMICXCF connection for LPAR SC31

```
IPCONFIG DYNAMICXCF 10.1.7.32 255.255.255.0 1
```

Example 3-13 IPCONFIG statement for DYNAMICXCF connection for LPAR SC32

```
IPCONFIG DYNAMICXCF 110.1.7.31 255.255.255.0 1
```

3.3.6 Verification of the DYNAMICXCF configuration

This section shows the display commands that are used to verify the DYNAMICXCF configuration.

Verify the VTAM support for XCF

Before starting a TCP/IP stack to use the DYNAMICXCF connection, verify that the VTAM configuration is active. Use the **`display VTAM options`** command to make sure that IQDCHPID=F3 **1** and XCFINIT=YES **2** were defined. Example 3-14 shows the results of the **`D NET,VTAMOPTS`** command. Issue the command to each VTAM node in the sysplex to verify that the required parameters are correct.

Example 3-14 Verify VTAM options

```
D NET,VTAMOPTS,OPTION=(IQDCHPID,XCFINIT)
IST097I DISPLAY ACCEPTED
IST1188I VTAM CSV2R1 STARTED AT 09:45:16 ON 11/21/13
IST1349I COMPONENT ID IS 5695-11701-210
IST1348I VTAM STARTED AS NETWORK NODE
IST1189I IQDCHPID = F3         XCFINIT  = YES
IST314I END
```

TCP/IP startup

Start the TCP/IP stack TCPIPF on LPAR SC30. SYSLOG messages issued during the start of the TCP/IP stack indicate if DYNAMICXCF is enabled, and if the VTAM TRLE definition was generated. To ensure that DYNAMICXCF was enabled, examine the system log for the EZZ0624I message **1**, which is generated at TCP/IP startup time. Also verify that the dynamically defined VTAM TRLE IUTIQDIO device was started **2**, as shown in Example 3-15.

Example 3-15 TCP/IP startup messages for DYNAMICXCF configuration

```
EZZ0624I DYNAMIC XCF DEFINITIONS ARE ENABLED 1
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTIQDIO 2
```

Also start the TCP/IP stacks on LPAR SC31 and SC32, which were configured for DYNAMICXCF.

Verify TCP/IP DEVICE and LINK status

To verify the status of devices and links defined to the TCP/IP stack, use the **D TCPIP,*procname*,NETSTAT,DEVLINKS** command to request NETSTAT information. The output is shown in Example 3-16.

Example 3-16 Verify device and link status for DYNAMICXCF

```
D TCPIP,TCPIPA,NETSTAT,DEV
...
DEVNAME: IUTIQDIO 1 DEVTYPE: MPCIPA
  DEVSTATUS: READY
  LNKNAME: IQDIOLNK0A01070B 2 LNKTYPE: IPAQIDIO   LNKSTATUS: READY
    IPBROADCASTCAPABILITY: NO
    ARPOFFLOAD: YES                  ARPOFFLOADINFO: YES
    ACTMTU: 8192
    VLANID: NONE
    READSTORAGE: GLOBAL (2048K)
    SECCLASS: 255
    IQDMULTIWRITE: ENABLED (ZIIP)
  ROUTING PARAMETERS:
    MTU SIZE: 8192             METRIC: 110
    DESTADDR: 0.0.0.0          SUBNETMASK: 255.255.255.0
  MULTICAST SPECIFIC:
    MULTICAST CAPABILITY: YES ...
```

TCP/IP will automatically generate a DEVICE IUTIQDIO **1** with LINK name IQDIOLNKxxxxxxxx, where xxxxxxxx is the IP address in hexadecimal format. In this case, the IP address was 10.1.7.11 for SC30. The LINK name created was IQDIOLNK0A01070B **2**.

Table 3-3 shows the hexadecimal value in the LINK name converted to a dotted IP address.

Table 3-3 LINK name-to-IP address conversion

LINK name 0A01070B	0A	01	07	0B
IP address 10.1.7.11	10	1	7	11

Tip: The LINK name generated by TCP/IP can be used in conjunction with static routes. However, you must first start the stack, then issue the **VARY TCPIP** command to add static routes. Also be aware that the LINK name will change whenever the IP address defined in the DYNAMICXCF statement changes.

Verify TRLE definitions

The first stack in an LPAR to initialize dynamic XCF makes VTAM generate a TRLE with a name of IUTIQDIO **1**. Use the **D NET,TRL,TRLE=*trle_name*** command, as shown in Example 3-17 on page 43, to verify that the TRLE is active **2**.

You can see from the display that the IUTIQDIO MPC group has been assigned 7700 for the read control device **3**, and 7701 for the write control device **4**. TCPIPA, the stack configured to use DYNAMICXCF **5**, is assigned data device 7702 **6**. Note that if you have multiple TCP/IP stacks running in the same LPAR and using CHPID F3, then datapath devices 7703 through 7709 are used. No additional read and write control devices are required.

The IUTIQDIO device is assigned a PORTNAME IUTIQD*xx*, where xx is the IQD CHPID that was specified in the IQDCHPID VTAM statement. For this example, the PORTNAME is IUTIQDF3 7, because you specified IQDCHPID=F3 in the VTAM start options.

Example 3-17 Verify TRLE

```
D NET,TRL,TRLE=IUTIQDIO
IST097I DISPLAY ACCEPTED
IST075I NAME = IUTIQDIO, TYPE = TRLE                    1
IST1954I TRL MAJOR NODE = ISTTRL
IST486I STATUS= ACTIV, DESIRED STATE= ACTIV             2
IST087I TYPE = LEASED            , CONTROL = MPC , HPDT = Y
IST1715I MPCLEVEL = QDIO       MPCUSAGE = SHARE
IST2263I PORTNAME = IUTIQDF3 7 PORTNUM =   0   OSA CODE LEVEL = *NA*
IST2337I CHPID TYPE = IQD      CHPID = F3  PNETID = **NA**
IST2319I IQD NETWORK ID = 0719
IST1577I HEADER SIZE = 4096 DATA SIZE = 16384 STORAGE = ***NA***
IST1221I WRITE DEV = 7701 STATUS = ACTIVE     STATE = ONLINE 4
IST1577I HEADER SIZE = 4092 DATA SIZE = 0 STORAGE = ***NA***
IST1221I READ  DEV = 7700 STATUS = ACTIVE     STATE = ONLINE 3
IST924I ---------------------------------------------------- 6
IST1221I DATA  DEV = 7702 STATUS = ACTIVE     STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1717I ULPID = TCPIPA 5 ULP INTERFACE = IUTIQDIO
IST2309I ACCELERATED ROUTING ENABLED
IST2331I QUEUE   QUEUE     READ             QUEUE
IST2332I ID      TYPE      STORAGE          STATUS
IST2205I ------  --------  ---------------  ----------------
IST2333I RD/1    PRIMARY   2.0M(126 SBALS)  ACTIVE
IST2305I NUMBER OF DISCARDED INBOUND READ BUFFERS = 0
IST2386I NUMBER OF DISCARDED OUTBOUND WRITE BUFFERS = 0
IST1757I PRIORITY1: UNCONGESTED PRIORITY2: UNCONGESTED
IST1757I PRIORITY3: UNCONGESTED PRIORITY4: UNCONGESTED
IST1801I UNITS OF WORK FOR NCB AT ADDRESS X'23C4B010'
IST1802I P1 CURRENT = 0 AVERAGE = 1 MAXIMUM = 1
IST1802I P2 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST1802I P3 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST1802I P4 CURRENT = 0 AVERAGE = 1 MAXIMUM = 2
IST924I ----------------------------------------------------
IST1221I DATA  DEV = 7703 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST924I ----------------------------------------------------
IST314I END
```

If multiple TCP/IP stacks are running in the same z/OS LPAR, the DYNAMICXCF connection creates a SAMEHOST TRLE named IUTSAMEH 1. DYNAMICXCF will provide connectivity between stacks under the same LPAR by using the SAMEHOST device, and not use the IUTIQDIO HiperSockets connection. Because you have additional TCP/IP stacks active that are not a part of this configuration example, use the **D NET,TRL,TRLE=*trle_name*** command, as shown in Example 3-18, to verify the TRLE status.

Example 3-18 Display IUTSAMEH TRLE

```
D NET,TRL,TRLE=IUTSAMEH

IST097I DISPLAY ACCEPTED
```

```
IST075I NAME = IUTSAMEH, TYPE = TRLE   1
IST1954I TRL MAJOR NODE = ISTTRL
IST486I STATUS= ACTIV, DESIRED STATE= ACTIV
IST087I TYPE = LEASED            , CONTROL = MPC , HPDT = YES
IST1715I MPCLEVEL = HPDT       MPCUSAGE = SHARE
IST1717I ULPID = TCPIPF
IST314I END
```

Verify DYNAMICXCF HiperSockets connections

Use the **ping** command to verify the HiperSockets connection between the LPARs. If multiple stacks are active, you must route the **ping** command to the correct stack. In this example, TCPIPA is using the DYNAMICXCF HiperSockets link, so specify **(tcp tcpipa)** as a ping command option 1. The results are shown in Example 3-19.

Example 3-19 DYNAMICXCF HiperSockets connectivity test

```
Ping from TCPIPA on LPAR SC30 to LPAR SC31
ping 10.1.7.32 (tcp tcpipa) 1
```

3.4 HiperSockets definitions for a z/VM host system

This section provides the TCP/IP definitions needed to support the HiperSockets environment on the z/VM system shown in Figure 3-1 on page 30. By default, the TCP/IP service machine is user ID TCPIP, and the TCP/IP configuration files are located on TCPMAINT’s 198 disk.

3.4.1 Permanent TCP/IP definitions for a z/VM host system

As with QDIO devices, each z/VM TCP/IP HiperSockets connection requires three I/O devices:

- One device is used for read control, and must be an even-numbered device.
- One device is used for write control, and must be the read control device plus one, which makes it odd-numbered.
- One device is used for data exchange, and must be one greater than the write control device.

All three I/O devices must be in a group of three contiguous device addresses.

The z/VM TCP/IP profile, PROFILE TCPIP, definitions for this configuration are shown in Example 3-20. The device type on the DEVICE statement for HiperSockets is HIPERS, as highlighted in the example. The DEVICE and LINK names, HIPERDF0 and HIPERLF0, are arbitrary. Choose a naming convention to identify the use of HiperSockets, along with the applicable CHPID F0.

Example 3-20 IBM z/VM TCP/IP definitions

```
   ...
   DEVICE HIPERDFO HIPERS 7400
   LINK HIPERLFO QDIOIP HIPERDFO
   ...
   HOME
   192.0.1.1 HIPERLFO
```

```
...
GATEWAY
; Network       Subnet          First            Link      MTU
; Address       Mask            Hop              Name      Size
; ------------- --------------- ---------------- --------- -----
192.0.1.0       255.255.255.0   =                HIPERLFO 8192
; ------------- --------------- ---------------- --------- ----
...
START HIPERDFO
```

Because you are using static routes in this environment, you have to define a GATEWAY statement. Also note that a maximum packet size (MTU size) of 8 KB was specified, to accommodate the MFS of the IQD CHPID.

Tip: If you are using a dynamic routing protocol, such as Routing Information Protocol (RIP) or Open Shortest Path First (OSPF), omit this GATEWAY statement.

Note: IBM z/VM can be connected to a HiperSockets network that a z/OS sysplex is using for DynamicXCF. However, z/VM cannot use the DynamicXCF protocol, but will establish a communication connection using the HiperSockets firmware.

To establish a connection, you need to add a port name to the TCPIP PROFILE file device statement, as shown in the following example:

```
DEVICE HIPERDF3 HIPERS 7700 PORTNAME IUTIQDF3
```

The value for the port name is the device identifier from the z/OS sysplex network configuration.

In the DEVICE statement, only the first of three I/O device addresses is defined, because the addresses are contiguous. IBM z/VM will determine the role of each device. These addresses are attached to the TCP/IP virtual machine (VM) with the SYSTEM DTCPARMS file.

The following SYSTEM DTCPARMS entry attaches the HiperSockets I/O devices permanently to the z/VM LPAR:

```
:nick.TCPIP    :type.server
               :class.stack
               :attach. 7400-7402
```

If other network I/O device addresses are defined to the TCP/IP stack, then add the HiperSockets I/O devices:

```
:nick.TCPIP    :type.server
               :class.stack
               :attach. C200-C203, 7400-7402
```

3.4.2 Dynamically define HiperSockets for a z/VM host system

Updating the PROFILE TCPIP and SYSTEM DTCPARMS files will permanently define the device to TCPIP. To dynamically bring the new device online without having to take TCPIP down, follow these steps:

1. From a privileged z/VM user ID, such as MAINT, attach the HiperSockets devices to TCPIP:

   ```
   CP ATTACH 7400-7402 TCPIP
   ```

2. From a user ID authorized to use the TCPIP OBEYFILE command, such as MAINT or TCPMAINT, issue the command for TCPIP to re-read the PROFILE TCPIP file. You will need to have access to that file before issuing the command:

   ```
   CP LINK TCPMAINT 198 198 rr
   CP ACC 198 F
   OBEYFILE PROFILE TCPIP F
   ```

   If there is a link password for accessing TCPMAINT's 198 disk, then TCPIP will have to be authorized to link to the disk, or a password needs to be included in the command:

   ```
   OBEYFILE PROFILE TCPIP F (read_password
   ```

3. If you want to stop the TCPIP service, log on to the TCPIP user ID and issue the following command:

   ```
   #CP EXTERNAL
   ```

4. To restart TCPIP, issue **TCPRUN** or **IPL CMS**.

   At device start, the z/VM TCP/IP stack is registered in the HiperSockets IP address lookup table with its IP address (192.0.1.1) and becomes part of the HiperSockets network on CHPID F0.

3.4.3 TCP/IP verification

Example 3-21 shows information pertinent to the TCP/IP HiperSockets device, which is displayed by using the NETSTAT DEVLINKS command. It ensures that the proper I/O device and MFS were defined to TCP/IP, and verifies that the device (HIPERDF0) and link (HIPERLF0) are in a READY status.

Important: To use these TCP/IP commands, the z/VM user ID has to be linked to TCPMAINT's 592 disk.

Example 3-21 Display TCP/IP DEVICE and LINK

```
NETSTAT DEVL

Device HIPERDF0 Type: HIPERS         Status: Ready
  Queue size: 0     CPU: 0     Address: 7400       Port name: UNASSIGNED
  IPv4 Router Type: NonRouter  Arp Query Support: No
    Link HIPERLF0 Type: QDIOIP        Net number: 0
      BytesIn: 4008           BytesOut: 1328
      Forwarding: Disabled     MTU: 8192
      Maximum Frame Size  : 16384
      Broadcast Capability: Yes
      Multicast Capability: Yes
```

```
      Group                                       Members
      -----                                       -------
      224.0.0.1                                      1
```

The NETSTAT GATEWAY command, shown in Example 3-22, indicates that the correct routing table entry has been added for HiperSockets.

Example 3-22 Display routing table

```
NETSTAT GATE

Subnet Address  Subnet Mask      FirstHop        Flgs PktSz Metric Link
--------------  -----------      --------        ---- ----- ------ ------
192.0.1.0       255.255.255.0    <direct>        US   8192  <none> HIPERLF0
```

Example 3-23 shows similar information about the HiperSockets connection using the IFCONFIG command.

Example 3-23 Display configuration

```
ifconfig hiperlf0

HIPERLF0 inet addr: 192.0.1.1 mask: 255.255.255.0
         UP BROADCAST MULTICAST MTU: 8192
         vdev: 7400 rdev: 7400 type: HIPERS
         vlan: 1 cpu: 0 forwarding: DISABLED
         RX bytes: 1040 TX bytes: 2888
```

For IP connectivity verification, use ping commands. In this example, we were able to successfully ping all of the other TCP/IP stacks that participate in HiperSockets CHPID F0 (192.0.1.2, 192.0.1.3, 192.0.1.4, 192.0.1.5, and 192.0.1.6).

3.4.4 References

The following list includes publications that provide additional information about the topic in this section:

- *z/VM V6R3 CP Commands and Utilities Reference,* SC24-6175
- *z/VM V6R3 CP Planning and Administration,* SC24-6178
- *z/VM V6R3 TCP/IP Planning and Customization,* SC24-6238
- *z/VM V6R3 TCP/IP User's Guide,* SC24-6240
- *z/VM V6R3 TCP/IP Messages and Codes,* GC24-6237
- *z/VM V6R3 TCP/IP Diagnosis Guide,* GC24-6235

These and other z/VM publications are available in PDF format on the IBM z/VM Internet Library:

http://www.vm.ibm.com/library/index.html

You can also find more information on the IBM Publication Center:

http://www-05.ibm.com/e-business/linkweb/publications/servlet/pbi.wss

3.5 HiperSockets in Linux on System z

Linux uses the QETH device driver to support HiperSockets on System z. It is delivered as part of the current Red Hat Enterprise Linux (RHEL) and SUSE Linux Enterprise Server (SLES) distributions. The QETH device driver also supports Open Systems Adapter (OSA)-Express ports operating in queued direct input/output (QDIO) mode. The HiperSockets internal QDIO (iQDIO) support is an extension to the OSA-Express QDIO support.

Each Linux HiperSockets connection requires three I/O devices. One device is used for read control, one for write control, and one for data exchange. The device number for the control write device must be the device number for the read control device plus one. The device number for the data exchange device can be any number.

This section provides information to use HiperSockets with Linux systems. To enable HiperSockets support, perform the following tasks:

1. Confirm that the correct input/output configuration program (IOCP) definitions are in place for the HiperSockets channels:
   - One or more IQD channels with access to a native Linux partition.
   - One or more IQD channels with access to a z/VM partition that is hosting the Linux guest. In this case, make sure that there are enough devices defined, so that each Linux guest can be attached to three different devices.
2. If Linux is hosted by z/VM, define the virtual environment in z/VM for the Linux guest system to use HiperSockets.
3. Configure the HiperSockets interfaces in the Linux system in one of the following ways:
   - Permanently, when installing the Linux system for the first time.
   - Add a permanent HiperSockets interface with an already existing Linux system.
   - Add a temporary HiperSockets interface with an existing Linux system. This interface will disappear when the system is rebooted.

The Linux support for HiperSockets on System z is the same as for Linux running in a stand-alone LPAR and running as a guest under z/VM. With this in mind, we chose to work with Linux guests in these test configurations. This section describes how to define the most simple HiperSockets interface. Additional functionality, such as layer 2 interfaces, virtual local area network (VLAN) definitions, and others, is described in subsequent sections.

3.5.1 Software requirements

The System z QETH Linux network device driver is required, and is available in all Linux distributions for System z.

3.5.2 Linux configuration example

The configuration used in this example is shown in Figure 3-5.

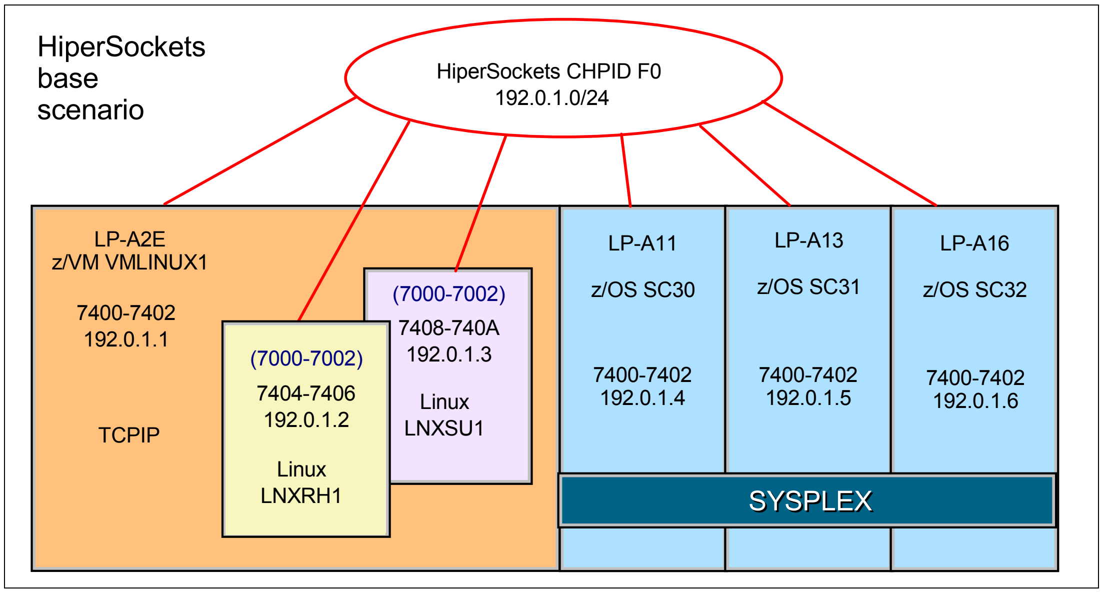


Figure 3-5 Linux test configuration

For the HiperSockets network, use CHPID F0 devices connected as I/O addresses 7000-7002 for both Linux environments. See 2.3, “IBM z/VM definitions” on page 24 for a description how to define the interfaces for the Linux guests in z/VM.

3.5.3 Linux I/O definitions for the initial installation of the Linux system

If your server environment will only be using the HiperSockets network, define the HiperSockets devices as your primary network when installing your Linux system. The installation process for both the SLES and RHEL distributions is much the same as the one that you follow using OSA or other network devices. The primary difference is identifying the type of network device as HiperSockets.

Note: Because the Linux systems were already installed, we did not define the HiperSockets interfaces during installation for this version of this book. The following paragraphs describe how this was done for a previous version of this book.

Follow these steps to complete the initial install:

1. Using a minimal parameter configuration file (`parmfile`), make the following choices at the network device prompt:

   Attention: For command formats for your specific Linux distribution, see `http://www.ibm.com/developerworks/linux/linux390/documentation_dev.html`

   a. For the SLES installation, select HiperSockets. Example 3-24 shows the prompt from an SLES install, selecting option 2.

   Example 3-24 SUSE installation prompt

   ```
   Please select the type of your network device.

   1) OSA-2 or OSA-Express
   2) HiperSockets
   --------------------
   3) Channel To Channel (CTC)
   4) ESCON
   5) Inter-User Communication Vehicle (IUCV)

   > 2
   ```

   b. For the RHEL installation, select the qeth driver and at the bus ID and Device Number prompt, enter the HiperSockets devices, such as `0.0.7000,0.0.7001,0.0.7002`.

   Both SUSE (SLES10 and later) and Red Hat (RHEL5 and later) allow the use of installation parameter files (parmfiles) to identify installation resources, rather than having to input these parameters during the installation. The format of the parmfile is the same as in an installation using a different type of network device, identifying HiperSockets as the installation device.

2. Configure the parmfile:

   a. For an SLES parmfile, include the line shown in Example 3-25.

   Example 3-25 SLES PARMFILE

   ```
   ....
   InstNetDev=hsi OsaInterface=eth OsaMedium=qdio
   ....
   ```

   b. For an RHEL parmfile, point to the configuration file to be used and include the lines in Example 3-26 and Example 3-27.

   Example 3-26 RHEL PARMFILE

   ```
   ...
   CMSCONFFILE=rhel4.conf
   ...
   ```

   Example 3-27 RHEL4 CONF

   ```
   ...
   DEVICE="hsi0"
   NETTYPE="qeth"
   ...
   ```

Important: To use the HiperSockets network during the initial installation, the installation code must be accessible from the HiperSockets network.

3.5.4 Linux I/O definitions for adding to an existing Linux system

This section shows the Linux setup from the perspective of the `/sysfs` file system. As such, the definitions are temporary, and will be lost after a reboot. 3.5.5, "Permanent Linux definitions" on page 54 describes how to make the definitions permanent.

Linux I/O verification

Ensure that your Linux system has the HiperSockets devices available:

1. From the Linux guest, issue the **`lscss`** command to display network devices, as shown in Example 3-28.

   Example 3-28 The lscss command

   ```
   lscss | grep 1732

   0.0.7000 0.0.0000  1732/05 1731/05      80  80  ff f0000000 00000000
   0.0.7001 0.0.0001  1732/05 1731/05      80  80  ff f0000000 00000000
   0.0.7002 0.0.0002  1732/05 1731/05      80  80  ff f0000000 00000000
   ```

   This shows that the Linux system knows of the devices, but the absence of the word *yes* between the `1731/05` and the `80` indicates that the devices are not configured or online. The first two characters of the second to the last column show the CHPID, `f0` in the example.

2. On an RHEL system, the HiperSockets devices might not show up, although they configured correctly in the input/output configuration data set (IOCDS) and in z/VM. In this case, to remove the HiperSockets devices from the list of ignored devices and make them visible to Linux, issue the following command:

   ```
   cio_ignore -r 0.0.7000,0.0.7001,0.0.7002
   ```

Hardware definition using znetconf

Use the **`znetconf`** command to sense and list candidate configurations for network devices. Example 3-29 shows how to use the znetconf command to list unconfigured network devices (the `-u` option), to add the HiperSockets devices (the `-a` option), and to list the configured network devices (the `-c` option).

Example 3-29 Use znetconf to configure HiperSockets devices

```
lnxsu1:~ # znetconf -u
Scanning for network devices...
Device IDs                 Type    Card Type      CHPID Drv.
------------------------------------------------------------
0.0.7000,0.0.7001,0.0.7002 1731/05 HiperSockets      f0 qeth

lnxsu1:~ #znetconf -a 7000
Scanning for network devices...
Successfully configured device 0.0.7000 (hsi0)

lnxsu1:~ #znetconf -c
```

```
Device IDs                Type    Card Type      CHPID Drv. Name       State
--------------------------------------------------------------------------------
0.0.c200,0.0.c201,0.0.c202 1731/01 GuestLAN QDIO     03 qeth eth0       online
0.0.8000,0.0.8001,0.0.8002 1731/01 GuestLAN QDIO     05 qeth eth1       online
0.0.7000,0.0.7001,0.0.7002 1731/05 HiperSockets      F0 qeth hsi0       online
```

Hardware definition using sysfs attributes

Alternatively you can use the sysfs attributes to set the devices online:

1. Create a qeth group device, as shown in the following code:

   ```
   echo 0.0.7000,0.0.7001,0.0.7002 > /sys/bus/ccwgroup/drivers/qeth/group
   ```

   This defines the device as part of the qeth group. The three devices are specified in device bus-ID form, with `0.0` proceeding each device address (the device number). Address 7000 is the read control I/O device, address 7001 is the write control I/O device, and address 7002 is the data exchange device.

   The driver handles the distribution for read and write buffer space dynamically. The qeth device driver uses the device bus-ID of the read subchannel to create a directory for the device, for example, this file should now exist:

   ```
   /sys/devices/qeth/0.0.7000
   ```

   Also, the driver creates additional directories that are symbolic links to the device directory:

   ```
   /sys/bus/ccwgroup/drivers/qeth/0.0.7000
   /sys/bus/ccwgroup/devices/0.0.7000
   ```

   This directory contains one file for each of the attributes that control the device. The driver automatically senses the device as a HiperSockets device, and sets all device attributes to their default values. Example 3-30 shows a list of these attributes.

   Example 3-30 List of device attributes

   ```
   [root@lnxrh1 ~]# ls /sys/devices/qeth/0.0.7000
   blkt               driver          net                rxip
   broadcast_mode     fake_broadcast  online             sniffer
   buffer_count       hsuid           performance_stats  state
   canonical_macaddr  hw_trap         portname           subsystem
   card_type          if_name         portno             uevent
   cdev0              inbuf_size      power              ungroup
   cdev1              ipa_takeover    priority_queueing  vipa
   cdev2              isolation       recover
   checksumming       large_send      route4
   chpid              layer2          route6
   ```

   Writing the new value to the appropriate file changes the attribute. The *Linux on System z, Device Drivers, Features, and Commands,* SC33-8281 publication describes the HiperSockets device attributes and the qeth device driver.

2. To bring the HiperSockets device online, write a `1` to the online attribute file:

   ```
   echo 1 > /sys/devices/qeth/0.0.7000/online
   ```

   Remember: Linux is case-sensitive. When using hardware addresses with hex alphabetic characters in commands and files, keep the case the same.

Network definition

The device driver associates the device with an interface name:

1. Find the name in the `if_name` attribute file for the device:
   a. For SLES, use the following command:
   ```
   lnxsu1:~ # cat /sys/devices/qeth/0.0.7000/if_name
   hsi1
   ```
   b. For RHEL, use the following command:
   ```
   [root@lnxrh1 ~]# cat /sys/devices/qeth/0.0.7000/if_name
   hsi0
   ```
2. With the name, use the **`ifconfig`** command to define the network interface, as shown in this example for RHEL:
   ```
   ifconfig hsi0  192.0.1.2 netmask 255.255.255.0 up
   ```

Verification of the setup

This section shows how the configuration can be verified regardless of the way that it is defined. Example 3-31 shows how the **`lsqeth`** command can be used to display the attributes of an interface.

Example 3-31 Using lsqeth to display attributes

```
[root@lnxrh1 ~]# lsqeth hsi0
Device name                  : hsi0
---------------------------------------------
        card_type            : HiperSockets
        cdev0                : 0.0.7000
        cdev1                : 0.0.7001
        cdev2                : 0.0.7002
        chpid                : F0
        online               : 1
        portname             : no portname required
        portno               : 0
        route4               : no
        route6               : no
        checksumming         : sw checksumming
        state                : UP (LAN ONLINE)
        priority_queueing    : always queue 2
        fake_broadcast       : 0
        buffer_count         : 128
        layer2               : 0
        large_send           : no
        isolation            : none
        sniffer              : 0
```

Example 3-32 shows the **`ifconfig`** command issued to verify the network setup.

Example 3-32 Verifying Linux HiperSockets setup with ifconfig

```
ifconfig

hsi0      Link encap:Ethernet  HWaddr 06:00:F0:2E:00:06
          inet addr:192.0.1.2  Bcast:192.0.1.255  Mask:255.255.255.0
          inet6 addr: fe80::400:f0ff:fe2e:6/64 Scope:Link
          UP BROADCAST RUNNING NOARP MULTICAST  MTU:8192  Metric:1
```

```
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:0 (0.0 b)  TX bytes:210 (210.0 b)
```

Tip: You might want to check the kernel messages (displayable by using the **`dmesg`** command) for any problems that could have occurred. Most distributions will also trigger a message in `/var/log/messages` in that case.

3.5.5 Permanent Linux definitions

This section provides the Linux information to permanently define the same configuration used in Figure 3-5 on page 49. Although the definition can be completed using the Linux distribution setup software (for example, SUSE YAST), this section shows you how to do it manually.

Hardware definition for SLES

SLES11 differs from SLES10 in that it no longer uses hardware definition files in `/etc/sysconfig/hardware`, but instead uses udev rules defined in `/etc/udev/rules.d`.

To define the new network interface, follow these steps:

1. Copy an existing OSA hardware definition file.
2. Modify the new file to change all of the device addresses to the new one:

   ```
   cd /etc/udev/rules.d
   sed 51-qeth-0.0.c200.rules -e ‘s/c20/700/g’ > 51-qeth-0.0.7000.rules
   ```

   Now the file looks like Example 3-33.

Example 3-33 The `/etc/udev/rules.d/51-qeth-0.0.7000.rules` *file*

```
# Configure qeth device at 0.0.7000/0.0.7001/0.0.7002
ACTION=="add", SUBSYSTEM=="drivers", KERNEL=="qeth", IMPORT{program}="collect
0.0.7000 %k 0.0.7000 0.0.7001 0.0.7002 qeth"
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.7000", IMPORT{program}="collect
0.0.7000 %k 0.0.7000 0.0.7001 0.0.7002 qeth"
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.7001", IMPORT{program}="collect
0.0.7000 %k 0.0.7000 0.0.7001 0.0.7002 qeth"
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.7002", IMPORT{program}="collect
0.0.7000 %k 0.0.7000 0.0.7001 0.0.7002 qeth"
ACTION=="remove", SUBSYSTEM=="drivers", KERNEL=="qeth",
IMPORT{program}="collect --remove 0.0.7000 %k 0.0.7000 0.0.7001 0.0.7002 qeth"
ACTION=="remove", SUBSYSTEM=="ccw", KERNEL=="0.0.7000",
IMPORT{program}="collect --remove 0.0.7000 %k 0.0.7000 0.0.7001 0.0.7002 qeth"
ACTION=="remove", SUBSYSTEM=="ccw", KERNEL=="0.0.7001",
IMPORT{program}="collect --remove 0.0.7000 %k 0.0.7000 0.0.7001 0.0.7002 qeth"
ACTION=="remove", SUBSYSTEM=="ccw", KERNEL=="0.0.7002",
IMPORT{program}="collect --remove 0.0.7000 %k 0.0.7000 0.0.7001 0.0.7002 qeth"
TEST=="[ccwgroup/0.0.7000]", GOTO="qeth-0.0.7000-end"
ACTION=="add", SUBSYSTEM=="ccw", ENV{COLLECT_0.0.7000}=="0",
ATTR{[drivers/ccwgroup:qeth]group}="0.0.7000,0.0.7001,0.0.7002"
ACTION=="add", SUBSYSTEM=="drivers", KERNEL=="qeth",
ENV{COLLECT_0.0.7000}=="0",
ATTR{[drivers/ccwgroup:qeth]group}="0.0.7000,0.0.7001,0.0.7002"
```

```
LABEL="qeth-0.0.7000-end"
ACTION=="add", SUBSYSTEM=="ccwgroup", KERNEL=="0.0.7000", ATTR{portno}="0"
ACTION=="add", SUBSYSTEM=="ccwgroup", KERNEL=="0.0.7000", ATTR{layer2}="0"
ACTION=="add", SUBSYSTEM=="ccwgroup", KERNEL=="0.0.7000", ATTR{online}="1"
```

Network definition for SLES

In a similar manner, the `/etc/sysconfig/network` directory contains the network configuration files. The files are named with the interface name. For example, the file name for this configuration is `ifcfg-hsi1`. The contents of this file are shown in Figure 3-6. This file can be built from a copy of an existing OSA definition file, or from the sample `/etc/sysconfig/network/ifcfg.template` file.

```
BOOTPROTO='static'
IPADDR='192.0.1.3'
BROADCAST='192.0.1.255'
STARTMODE='auto'
NAME='ITSO HiperSockets Network CHPID F0 (0.0.7000)'
USERCONTROL='no'
NETMASK='255.255.255.0'
```

Figure 3-6 Example ifcfg-hsi1

To activate the new file after it has been properly customized, you use the following **`ifup`** command:

```
lnxsu1:/etc/sysconfig/network # ifup hsi1
hsi1      name: ITSO HiperSockets Network CHPID F0 (0.0.7000)
```

You can also use the **`rcnetwork restart`** command shown in Example 3-34.

Example 3-34 The rcnet work restart command

```
lnxsu1:~ # rcnetwork restart
Shutting down network interfaces:
    eth0      name: OSA Express Network card (0.0.c200)           done
    hsi1      name: ITSO HiperSockets Network CHPID F0 (0.0.7000)    done
Shutting down service network  .  .  .  .  .  .  .  .  .          done
Hint: you may set mandatory devices in /etc/sysconfig/network/config
Setting up network interfaces:
    eth0      name: OSA Express Network card (0.0.c200)
    eth0      IP address: 9.12.7.16/20                             done
    hsi1      name: ITSO HiperSockets Network CHPID F0 (0.0.7000)
    hsi1      IP address: 192.0.1.3/24                             done
Setting up service network  .  .  .  .  .  .  .  .  .  .          done
```

This will permanently define the HiperSockets interface configuration for your SLES11 system.

Hardware and network definition for Red Hat Enterprise Linux

The `/etc/sysconfig/network-scripts` directory contains the network configuration files. These files contain hardware information as well. These files are named with the interface type and number. For example, your file name is `ifcfg-hsi0`.

The contents shown in this file can be built from a copy of an existing configuration file. Figure 3-7 shows the example `ifcfg-hsi0` file.

```
DEVICE=hsi0
BOOTPROTO=static
IPADDR=192.0.1.2
NETMASK=255.255.255.0
NETTYPE=qeth
ONBOOT=yes
SUBCHANNELS=0.0.7000,0.0.7001,0.0.7002
TYPE=Ethernet
```

Figure 3-7 Example ifcfg-hsi0 file

To activate the interface, you can issue the **`ifup`** command:

```
ifup hsi0
```

3.5.6 References

The following list includes publications that provide additional information about the topic in this section:

- *Device Drivers, Features, and Commands*, SC33-8281. The most recent version can be found in the *What's new link*:

  http://www.ibm.com/developerworks/linux/linux390/index.html

- Additional information is available at the following sites:

  http://www-03.ibm.com/systems/z/os/linux/
  http://www.vm.ibm.com/linux

- For details about Linux distributions and support, go to the following website:

  http://www.ibm.com/servers/eserver/zseries/os/linux/

3.6 HiperSockets in z/VSE

This section covers the basic setup and usage of HiperSockets in a z/VSE environment. HiperSockets have been supported since IBM VSE/ESA 2.7.

Note: We did not have a z/VSE system for this version of the book. The information about z/VSE was supplied by the IBM z/VSE developers. For additional information, see *Introduction to the New Mainframe: z/VSE Basics*, SG24-7436 and *Enhanced Networking on IBM z/VSE*, SG24-8091 for instructions about how to set up HiperSockets interfaces on z/VSE.

3.6.1 HiperSockets Support in z/VSE

HiperSockets devices are defined in the IOCDS with channel path identifier (CHPID type IQD). IBM z/VSE supports HiperSockets spanned channels. These channels are HiperSockets that connect LPARs using different logical channel subsystems (LCSSs).

For HiperSockets devices, the z/VSE QDIO network driver supports the following items:

- TCP/IP layer 3
- IPv4 protocols
- IPv6 protocols (z/VSE 5.1 and later)
- VLAN (z/VSE 5.1 and later)

IPv6 support is provided by IPv6/VSE.

The TCP/IP configuration for HiperSockets devices is provided by the following software:

- TCP/IP for VSE/ESA (licensed from Connectivity Systems, Inc., or CSI)

  This is a well-known TCP/IP stack with applications for IPv4 traffic. It provides secure transmission of data using the Secure Sockets Layer (SSL) protocol. For more information about CSI's TCP/IP for VSE/ESA, see the following website:

  `http://www.csi-international.com`

- IPv6/VSE (licensed from Barnard Software Inc., or BSI)

  This is a full-function TCP/IP stack with applications for IPv4 and IPv6 network traffic. Since December 2012, secured transmission of data using the Secure Socket Layer (SSL) Protocol, Hypertext Transfer Protocol Secure (HTTPS), File Transfer Protocol Secure (FTPS), Simple Mail Transfer Protocol Secure (SMTPS), and TN3270E over SSL has been supported. It is available for z/VSE 4.3 and z/VSE 5.1.

  For more information about BSI's IPv6/VSE, see their Internet home page:

  `http://www.bsiopti.com`

If you run z/VSE under z/VM, you can also use virtual HiperSockets. For details about HiperSockets support under z/VM, see the corresponding z/VM documentation.

3.6.2 *Configuring HiperSockets devices in z/VSE*

For each HiperSockets link, you require three CHPID type IQD devices. In z/VSE, CHPID type IQD devices have a corresponding z/VSE device type: `OSAX`.

The HiperSockets devices must be added with the z/VSE IPL **`ADD`** statement using type OSAX. To distinguish CHPID type IQD devices from CHPID type OSA-Express QDIO (OSD) devices, a mode of 1 must be specified, as shown in the following example:

```
ADD 1500:1515 AS 500:515,OSAX,1
```

3.6.3 Configuring a HiperSockets link in TCP/IP

The statements for using a HiperSockets connection vary depending upon the TCP/IP solution that you have chosen. Under TCP/IP for VSE/ESA, to define a layer 3 IPv4 link you must specify device and link information in the TCP/IP DEFINE LINK command:

```
DEFINE LINK,ID=...,TYPE=OSAX,
      DEV=cuu1,          (or DEV=(cuu1,cuu2))
      DATAPATH=cuu3,
      IPADDR=addr,
      MTU=xxxx,          (default: as specified in the OS parameter)
      FRAGMENT={NO|YES}  (default: NO)
                         (YES not supported by HiperSockets)
```

In this example, cuu1 and cuu2 are VSE addresses that correspond to physical addresses. These definitions are the same as those for OSA Express, except in the following ways:

- HiperSockets do not require a PORTNAME.
- The MTU size must not exceed the MTU size specified in the OS parameter (CHPID definition). The default MTU size is the size specified in the OS parameter (CHPID definition).

Under IPv6/VSE, for details of how to specify the equivalent statements to those described previously, see the following website:

http://www.ibm.com/systems/z/os/zvse/documentation/#tcpip

3.6.4 Related publications

For more details about TCP/IP in z/VSE, see the following publications:

- *Enhanced Networking on IBM z/VSE*, SG24-8091
- *Introduction to the New Mainframe: z/VSE Basics*, SG24-7436
- *z/VSE Planning*, SC33-8301
- *z/VSE Administration*, SC33-8304
- *z/VSE Operation*, SC33-8309
- *z/VSE V5R1 e-business Connectors User's Guide*, SC34-2629
- *z/VSE V5R1 TCP/IP Support*, SC34-2640

4

Performance considerations

This chapter describes the factors that influence IBM System z HiperSockets performance. It provides insights on which tuning parameters are available, and can help you decide how to configure HiperSockets for your scenario.

4.1 HiperSockets for highest performance

HiperSockets moves network data from one virtual server to another. This movement is synchronous to the device driver and uses memory copies, therefore resulting in the lowest possible latency and highest throughput. So, although it emulates an Open Systems Adapter (OSA) channel in many aspects, with respect to performance it behaves more like a software program.

4.2 Processor considerations

The firmware that copies the data runs inside the send instruction, so a server that is processor-constrained will not be able to send as much data per second as a server with more processing power available. This is especially true for processors shared between logical partitions (LPARs), or between z/VM guests. In the time slices where a processor is not running for a server, HiperSockets will not move data for that server.

The operating systems (OSs) can distribute HiperSockets traffic over multiple processors, so a server with more physical processors available is able to send more data per second over HiperSockets. This is not true if the processors are *only* virtual, and not backed by physical resources, because then no true parallelism can be achieved.

Upgrading to a new model of System z will typically result in higher HiperSockets throughput and lower latency, because of the improved capacity of the processor hardware.

HiperSockets firmware on central processors (CPs) of subcapacity models runs at the same speed as on full capacity models or Integrated Facilities for Linux (IFLs). However, the subcapacity CPs will affect the end-to-end network performance, because it affects the Transmission Control Protocol/Internet Protocol (TCP/IP) stack and the device driver on these servers.

4.3 Physical memory structure

HiperSockets firmware uses the same hardware as other System z software memory moves use. The best overall performance is achieved when processors accessing the same memory location are in close physical proximity, sharing a certain level of cache. Processors that are physically separated, on different processor books, achieve less optimal overall performance. The same is true for the processors that communicate over HiperSockets. Depending on their physical location, HiperSockets performance can vary.

4.4 Maximum transmission unit size

HiperSockets can be defined with four different maximum transmission units (MTUs): 8 KB, 16 KB, 32 KB, or 56 KB (see 2.1, “System configuration considerations” on page 8). Transferring large units of data in one I/O operation can result in very high throughput, especially for streaming-type workloads. For request-response type workloads, smaller MTU sizes are usually sufficient.

A large MTU can improve the throughput, and it can also result in less processor use, because the TCP/IP stack and device driver functions are called less often. It is important to adjust both the HiperSockets channel definition and the MTU definitions in the TCP/IP stacks, to benefit from the large MTU capabilities of HiperSockets.

Note that the MTU size affects the size of the input and output buffers that are allocated by the operating system. A larger MTU size will result in higher memory consumption. If a server is severely memory-constrained, larger MTU sizes, and therefore larger buffers, might deteriorate the situation, lead to increased paging activity, and therefore to less than optimal overall system performance.

4.5 Input buffer count

Important: Because HiperSockets transfers data synchronously, successful data delivery depends on the available queued direct input/output (QDIO) input buffers of the target system. The device drivers or TCP/IP stacks are able to handle conditions when there are no available target buffers, but these situations affect the overall performance.

Optimal performance can be achieved if the device drivers always try to provide the maximum number of 128 available input buffers. Each input buffer has the size of the maximum frame size (MFS) that is defined for the internal queued direct communication (IQD) channel (not the MTU that is defined for this TCP/IP stack). Therefore, in a scenario with a large MFS and servers that are memory-constrained rather than processor-constrained, a smaller number of input buffers might be appropriate.

4.5.1 Input buffer count in IBM z/OS

The default number of input buffers for HiperSockets for z/OS is 126. To configure the number of input buffers for HiperSockets, the Virtual Telecommunications Access Method (VTAM) Start Option **`IQDIOSTG`** or the **`READSTORAGE`** keyword on the LINK or INTERFACE statements can be used. For more information, go to the z/OS V2R1 Information Center:

`http://pic.dhe.ibm.com/infocenter/zos/v2r1/index.jsp`

4.5.2 Input buffer count in Linux for System z

A count of 128 input buffers has been the default since Red Hat Enterprise Linux (RHEL) V6.2 and SUSE Linux Enterprise Server (SLES) V11 SP2. It used to be 16 in previous versions. Your current input buffer count can be checked using the **`lsqeth -p`** command. The input buffer count can be set in the appropriate config file:

- SUSE SLES 10

  In `/etc/sysconfig/hardware/hwcfg-qeth-bus-ccw-0.0.`*`device_number`*, add **`QETH_OPTIONS="buffer_count=128"`**.

- SUSE SLES 11

  In `/etc/udev/rules.d/51-qeth-0.0.`*`device_number`*`.rules`, add **`ACTION=="add"' SUBSYSTEM=="ccwgroup"' KERNEL=="0.0.`*`device_number`*`", ATTR{buffer_count}="128"`**.

- RHEL 6.2

  In `/etc/sysconfig/network-scripts/ifcfg-`*`interface_name`*,
  add **`OPTIONS="buffer_count=128"`**.

4.5.3 Input buffer count in z/VSE

The default number of input buffer for HiperSockets on z/VSE is 8. To configure the number of QDIO input buffers for HiperSockets, you can use the configuration skeleton SKOSACFG in ICCF library 59. Valid values are: 8, 16, 32, or 64. For more information see *Enhanced Networking on IBM z/VSE*, SG24-8091.

4.6 References

Look for the current IBM performance benchmarks at IBM developerWorks®:

http://www.ibm.com/developerworks/linux/linux390/perf/index.html

5

Layer 2 and layer 3 modes

HiperSockets can operate either as layer 3 interfaces (network or Internet Protocol (IP) layer), or as layer 2 interfaces (link layer).

This chapter describes what the layer concept means for HiperSockets, and explains how HiperSockets use IPv4, IPv6, and message authentication code (MAC) addresses to communicate.

5.1 Concept of layer modes for HiperSockets

For flexible and efficient data transfer for IP and non-IP workloads, HiperSockets can support two transport modes. These are layer 3 (network or IP layer) and layer 2 (link layer).

As described in 1.4, "HiperSockets mode of operation" on page 4, the virtual servers register target addresses for the HiperSockets interfaces with the firmware. A HiperSockets device can be configured by the device driver either in layer 3 mode or in layer 2 mode. In layer 3 mode, IP addresses are used as target addresses. In layer 2 mode, MAC addresses are used as target addresses.

Layer 3 mode is used for messages in IP datagram format (either IPv4 or IPv6), identifying a next-hop IP address for each message. An internal queued direct communication (IQD) channel-path identifier (CHPID) represents an IP subnet in this case.

Layer 2 mode is intended to be used for messages in Ethernet format (Ethernet II Digital, Intel, and Xerox (DIX) or Institute of Electrical and Electronics Engineers (IEEE) 802.3 frames). An IQD CHPID represents an Ethernet broadcast domain in this case.

The layer used by the guests is not apparent to IBM z/VM, if the devices are directly connected to the guest.

Tip: Devices in layer 3 mode and devices in layer 2 mode cannot communicate with each other. It is a good practice to either define only layer 3 devices or only layer 2 devices on one HiperSockets CHPID.

5.2 Layer 3 mode

When a HiperSockets device is defined in layer 3 mode, both IPv4 or IPv6 protocols can be used. An IQD CHPID represents an IP local area network (LAN) segment in this case.

Software enqueues IP datagrams into the HiperSockets queues, and sets the next-hop IP address as the destination. HiperSockets firmware delivers these datagrams into the target queues without the involvement of any lower-level protocols. The software device drivers also operate at the IP layer. The lower levels of the Transmission Control Protocol/Internet Protocol (TCP/IP) stack, such as Ethernet, are not required at all.

Different upper-layer protocols, such as TCP, User Datagram Protocol (UDP), and others, can be used by the software stacks on HiperSockets layer 3 devices. IBM z/OS, Linux on System z, z/VM (host) and z/VSE support HiperSockets layer 3 devices. As of the writing of this book, layer 3 is the default mode for all operating systems (OSs).

In z/VSE, layer 3 is the default. It is supported by both TCP/IP software packages:

- TCP/IP for VSE/ESA (licensed from Connectivity Systems, Inc. (CSI))
- IPv6/VSE (licensed from Barnard Software Inc. (BSI))

5.2.1 IPv4

Four-byte IPv4 addresses are used as target addresses. IBM z/OS, Linux on System z, z/VM (host), and z/VSE support IPv4 over HiperSockets layer 3 devices. IBM z/VSE support has been present since VSE/ESA V2.7.

5.2.2 IPv6

16-byte IPv6 addresses are used as target addresses. IPv6 support for layer 3 devices has been available since IBM System z9®.

IPv6 address generation

If stateless auto-configuration is used on IPv6 interfaces, the IP stack tries to define a *link-local address* by obtaining the MAC address of the interface. HiperSockets firmware provides a computer-generated virtual MAC address for that purpose. See 5.3.1, "MAC address generation" on page 66 for more information.

IPv6 on layer 3 devices software support

IPv6 on layer 3 devices supports the following software:

- All current z/OS releases support IPV6.
- HiperSockets accelerator does not support IPV6.
- Linux support has been present since Red Hat Enterprise Linux (RHEL) 5.2 and Novell SUSE Linux Enterprise Server (SLES) 10 SP2.
- Virtual machine (VM) TCP/IP stack support has been present since z/VM V5.2.
- IBM z/VSE 4.2 and later support includes IPv6/VSE (licensed from BSI).

5.2.3 IP takeover

An IP address is registered with its HiperSockets interface by the TCP/IP stack when the TCP/IP device is started. IP addresses are removed from an IP address lookup table when a HiperSockets device is stopped. Under OS control, you can reassign IP addresses to other HiperSockets interfaces on the same HiperSockets LAN. This enables flexible backup of TCP/IP stacks.

Important: Reassignment is only possible in the same HiperSockets LAN. A HiperSockets channel is one network or subnetwork. Reassignment is only possible for the same OS type. The following list includes some examples:

- An IP address originally assigned to a Linux TCP/IP stack can only be reassigned to another Linux TCP/IP stack.
- A z/OS dynamic virtual IP address (VIPA) can only be reassigned to another z/OS TCP/IP stack.
- A z/VM TCP/IP VIPA can only be reassigned to another z/VM TCP/IP stack.

The firmware performs the reassignment in force mode. It is up to the OS's TCP/IP stack to control this change.

IP address takeover in Linux

Linux supports IP address takeover. Only IP addresses from another Linux system on the same System z can be taken over. HiperSockets checks the type of OS before a change in the IP address lookup table is made. IP address takeover must be initiated on the system that takes over an IP address (add). IP address takeover must be enabled on *both* systems when loading the driver. The devices are *disabled* by default. To enable the device for takeover, write to the following file:

```
/sys/devices/qeth/<device_number>/ipa_takeover/enable
```

For example, you could use the following write:

```
echo 1 > /sys/devices/qeth/0.0.7000/ipa_takeover/enable
```

5.3 Layer 2 mode

When a HiperSockets device is defined in layer 2 mode, software implements the layer 2 functionality of the network stack (in this case, Ethernet including Address Resolution Protocol, or ARP), registers virtual MAC addresses as target addresses, and enqueues Ethernet frames into the internal queued direct input/output (iQDIO) queues. Firmware will then deliver these Ethernet frames based on the destination MAC address. An IQD CHPID represents an Ethernet broadcast domain in this case.

This enables the use of standard networking stacks over HiperSockets devices, without having to separate the IP layer from the Ethernet layer. It also enables the use of applications that depend on the existence of an Ethernet layer: NetBIOS, Dynamic Host Configuration Protocol (DHCP) servers, some firewalls, and others. Network stacks can implement IPv4, IPv6, or other protocols over HiperSockets layer 2 devices.

Layer 2 for HiperSockets requires IBM System z10® or later.

5.3.1 MAC address generation

HiperSockets is a virtual network without any MAC addresses burnt into real hardware adapters. Therefore, firmware generates a default virtual MAC address for each device. These generated addresses can be read by the software stack and can be used for layer 2 Ethernet addresses, or to generate IPv6 link-local addresses. It is not mandatory to use these default MAC addresses, because user-defined virtual MAC addresses can also be used. Firmware-generated MAC addresses have been available since IBM System z9.

These generated MAC addresses are locally administered. They are guaranteed to be unique in the same central processor complex (CPC) with respect to MAC addresses generated by other firmware or z/VM. Because HiperSockets networks connect only to other HiperSockets interfaces in the same CPC, no duplicate MAC addresses can occur on the same HiperSockets network. Usually, HiperSockets interfaces on different CPCs will get different MAC addresses, but this is not guaranteed.

The default MAC address of a HiperSockets device is persistent, which means that each data device will get the same firmware-generated MAC address again after the server is restarted, or if the HiperSockets CHPID is configured off and on, or even if the CPC is restarted. If the server is restarted in a different logical partition (LPAR), or on a different CPC, it will get a different default MAC address. If it is a requirement to keep the same MAC address, user-defined MAC addresses must be used.

A variation of this mechanism is used for IBM zEnterprise intra-ensemble data networks (IEDNs). There, the network management component of the zEnterprise Unified Resource Manager (URM) orchestrates the generation of MAC addresses for the IEDN. See 9.4, “MAC management by the URM” on page 129 for more information about this topic.

5.3.2 HiperSockets layer 2 mode software support

HiperSockets provides the following layer 2 mode software support:

- z/OS, as of today, supports only HiperSockets layer 2 for internal queued direct Input/output extensions (IQDX) channels. See 9.7, “The z/OS converged interface” on page 139 for more details.
- Linux for System z is supported by Novell SLES 10 SP2 or RHEL 5.2 and later.
- z/VSE provides no layer 2 support for HiperSockets as of the writing of this book.
- z/VM provides the following support:
  - The layer of direct-connected guest devices is not apparent to z/VM.
  - A z/VM TCP/IP stack does not support layer 2, as of the writing of this book.
  - HiperSockets Bridge Port on z/VM VSwitch is always a layer 2 device. See Chapter 8, “Connect HiperSockets to other networks” on page 95 for more information.

Layer 2 for Linux

To dynamically put a HiperSockets interface into layer 2 mode, you first need to unconfigure the device. A sequence that worked with both SLES and RHEL is shown in Example 5-1.

Example 5-1 Dynamically setting an HiperSockets interface with layer 2 using znetconf

```
lnxsu1:~ # lsqeth hsi1
Device name                  : hsi1
---------------------------------------------
        card_type            : HiperSockets
        cdev0                : 0.0.7000
        cdev1                : 0.0.7001
        cdev2                : 0.0.7002
        chpid                : F0
        online               : 0
        portname             : no portname required
        portno               : 0
        route4               : no
        route6               : no
        checksumming         : sw checksumming
        state                : UP (LAN ONLINE)
        priority_queueing    : always queue 2
        fake_broadcast       : 0
        buffer_count         : 128
        layer2               : 0
        large_send           : no
        isolation            : none
        sniffer              : 0
lnxsu1:~ # znetconf -r 7000
Remove network device 0.0.7000 (0.0.7000,0.0.7001,0.0.7002)?
Warning: this may affect network connectivity!
Do you want to continue (y/n)?y
Successfully removed device 0.0.7000 (hsi1)
lnxsu1:~ # znetconf -a 7000 -o layer2="1"
Scanning for network devices...
Successfully configured device 0.0.7000 (hsi1)
lnxsu1:~ # lsqeth hsi1
Device name                  : hsi1
---------------------------------------------
```

```
        card_type           : HiperSockets
        cdev0               : 0.0.7000
        cdev1               : 0.0.7001
        cdev2               : 0.0.7002
        chpid               : F0
        online              : 1
        portname            : no portname required
        portno              : 0
        state               : UP (LAN ONLINE)
        priority_queueing   : always queue 2
        buffer_count        : 128
        layer2              : 1
        isolation           : none
```

For RHEL, it seems to be sufficient to set the device offline. The sequence shown in Example 5-2 only worked for RHEL, not for SLES.

Example 5-2 Dynamically set a HiperSockets interface with layer 2 using sysfs

```
[root@lnxrh1 ~]# cd /sys/devices/qeth/0.0.7000
[root@lnxrh1 0.0.7000]# echo 0 > /sys/devices/qeth/0.0.7000/online
[root@lnxrh1 0.0.7000]# echo 1 > /sys/devices/qeth/0.0.7000/layer2
[root@lnxrh1 0.0.7000]# echo 1 > /sys/devices/qeth/0.0.7000/online
```

For a permanent layer 2 definition in SLES11, define the following **ATTR** parameter in `/etc/udev/rules.d/51-qeth-0.0.`*`device_number`*`.rules`:

```
ACTION=="add", SUBSYSTEM=="ccwgroup", KERNEL=="0.0.device_number",
ATTR{layer2}="1"
```

For a permanent layer 2 definition in RHEL, define the following **OPTIONS** parameter in `/etc/sysconfig/network-scripts/ifcfg-`*`interface_name`*:

```
OPTIONS="layer2=1"
```

6

Virtual local area network support

This chapter describes the concept of a virtual local area network (VLAN), and how it is implemented in the HiperSockets environment. Examples are provided to illustrate its use.

6.1 Overview

The HiperSockets VLAN can logically subdivide the internal LAN for a HiperSockets channel-path identifier (CHPID) into multiple LANs. Two or more stacks that configure the same VLAN ID for the same CHPID can communicate over HiperSockets, but stacks that configure different VLAN IDs cannot.

A VLAN configuration provides many benefits, such as improved network performance by distributing traffic among VLANs, enhanced security by isolating traffic, and more flexibility in configuring networks. See Figure 6-1 for an example of how a HiperSockets LAN is subdivided into two VLANs.

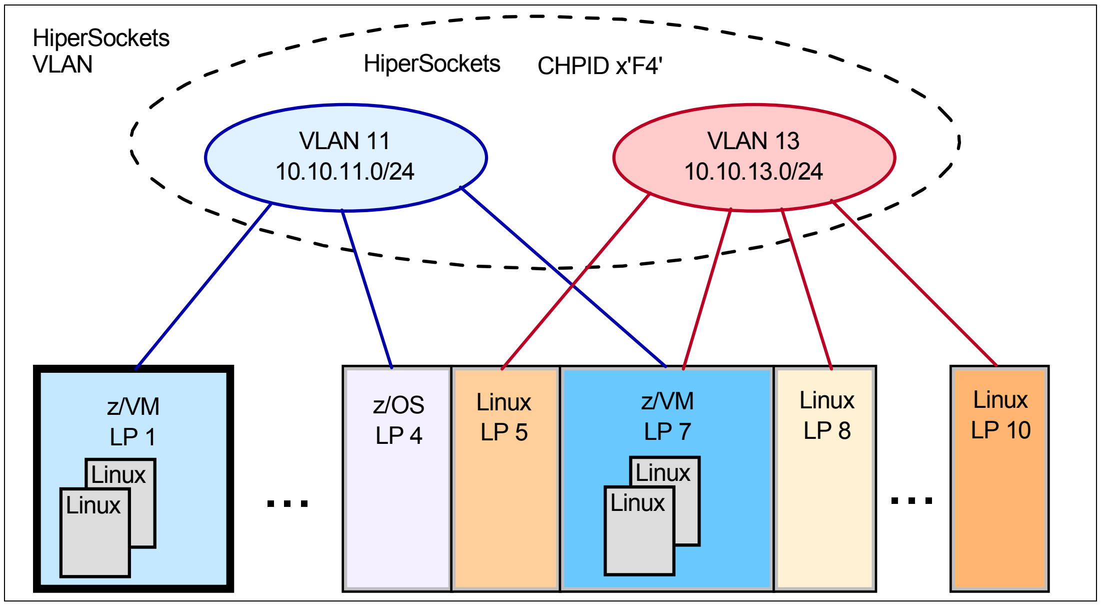


Figure 6-1 HiperSockets VLAN

6.2 Types of connections according to IEEE

Institute of Electrical and Electronics Engineers (IEEE) 802.1p and IEEE 802.1q VLANs operate by defining switch ports as members of VLANs. Devices on a VLAN can be connected in three modes (trunk, access, and hybrid), based on whether the connected devices are VLAN-aware or VLAN-unaware. VLAN-aware devices use tagging and understand VLAN memberships (which users belong to a particular VLAN) and VLAN formats.

Ports used to attach VLAN-unaware equipment are called *access ports*, and ports used to connect to other switches or VLAN-aware servers are known as *trunk ports*. Network frames generated by VLAN-aware equipment are marked with a *tag* that identifies the frame as belonging to a particular VLAN.

Trunk mode

Trunk mode indicates that the switch ought to allow all VLAN ID-tagged packets to pass through the switch port without altering the VLAN ID. This mode is intended for servers that are VLAN-capable. It filters and processes all VLAN ID-tagged packets. In trunk mode, the switch expects to see VLAN ID-tagged packets inbound to the switch port.

Access mode

Access mode indicates that the switch ought to filter on specific VLAN IDs, and only allow packets that match the configured VLAN IDs to pass through the switch port. The VLAN ID is then removed from the packet before it is sent to the server. That is, VLAN ID filtering is controlled by the switch. In access mode, the switch expects to see packets without VLAN ID tags inbound to the switch port.

Hybrid mode

Hybrid mode is a combination of the previous two modes. This is a port where both VLAN-aware and VLAN-unaware devices are attached. A hybrid port can have both tagged and untagged frames.

6.2.1 HiperSockets as a virtual switch in trunk mode with VLAN control

A HiperSockets LAN is not connected to any physical switches or network. However, the HiperSockets CHPID acts as a virtual switch, and its interfaces into the servers can be viewed as virtual switch ports.

As of this writing, HiperSockets does *not* support access mode. The servers' operating systems (OSs) have to be VLAN-aware. However, HiperSockets firmware will verify the VLAN tags of the messages, and only deliver a message if the tag matches one of the VLAN IDs defined by the server.

Although no Ethernet frames are constructed when HiperSockets are used in layer 3 mode, it is still possible to define VLAN IDs per server for layer 3 HiperSockets devices. As for layer 2, HiperSockets will only deliver messages when the source and target servers have defined the same VLAN ID.

6.3 Out-of-band VLAN management using the IBM zEnterprise Unified Resource Manager Network Virtualization Manager

Chapter 9, "HiperSockets in an IBM zEnterprise ensemble" on page 125 provides information about the IBM zEnterprise Unified Resource Manager (URM), also called zManager, and its component, the *Network Virtualization Manager* (NVM). NVM is used to divide an intra-ensemble data network (IEDN) HiperSockets channel (internal queued direct Input/output extensions, or IQDX) into VLANs, and define per server to which VLAN it belongs.

IQDX HiperSockets enforce NVM settings, and prevent any communication that does not concur with these settings. However, HiperSockets still operates in trunk mode, which means that the servers still need to be VLAN-aware.

See 9.4, "MAC management by the URM" on page 129 for more details about NVM VLAN management.

6.4 Benefits of HiperSockets VLAN

The following list describes some of the benefits of using VLANs with HiperSockets:

- Optimization of traffic flow

  HiperSockets makes sure that no message with an incorrect VLAN ID goes into the input queue of an interface, and therefore prevents the OSs from unnecessary handling of these messages. This is especially true for broadcast messages. Broadcasts will only be sent to a target that belongs to the same VLAN.

- Isolation

  An important point about VLANs in general is that they provide isolation. VLANs behave like separate networks, even though they are contained in the same HiperSockets LAN.

- Flexibility

  A server can be moved from one VLAN to another by just changing the configuration in its OS, without changing the hardware configuration of the central processor complex (CPC).

6.5 An example of HiperSockets VLAN in z/OS

To demonstrate how HiperSockets VLANs are defined and operated, create two VLANs across a single HiperSockets channel. Logical partition (LPAR) SC31 is defined to VLAN 2, and LPARs SC30 and SC32 are defined to VLAN 3, as shown in Figure 6-2 on page 73.

Additionally, a z/VM LPAR and Linux system are configured to VLAN 2. Another Linux system is configured to VLAN 3. The z/VM and Linux configuration details are covered in the two sections that follow, 6.6, “HiperSockets VLAN for a z/VM host system” on page 77 and 6.7, “HiperSockets VLAN in Linux on System z” on page 78.

Configure the TCP/IP stacks using the following configuration rules:

- The VLANID parameter is specified on the INTERFACE statement for the HiperSockets device in the TCP profile data set.
- For TCP/IP stacks communicating over the same HiperSockets channel, each stack must specify the same VLANID parameter value.

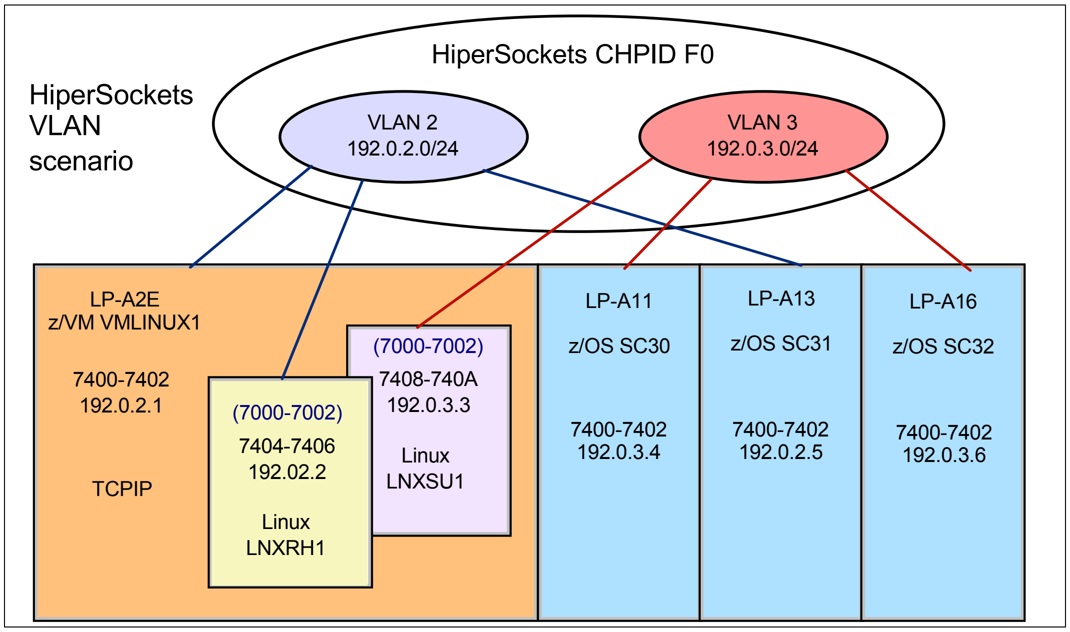

HiperSockets VLAN scenario
HiperSockets CHPID F0
VLAN 2 192.0.2.0/24
VLAN 3 192.0.3.0/24
LP-A2E z/VM VMLINUX1
7400-7402 192.0.2.1
TCPIP
(7000-7002) 7404-7406 192.02.2
Linux LNXRH1
(7000-7002) 7408-740A 192.0.3.3
Linux LNXSU1
LP-A11 z/OS SC30 7400-7402 192.0.3.4
LP-A13 z/OS SC31 7400-7402 192.0.2.5
LP-A16 z/OS SC32 7400-7402 192.0.3.6

Figure 6-2 HiperSockets VLAN configuration

Although the scenario described in the following pages uses IPv4, VLAN for HiperSockets is supported in IPv6 as well.

6.5.1 Implementation steps

Take the following steps to implement VLAN on HiperSockets:

1. Use the configuration detailed in Figure 3-2 on page 32.
2. Add the VLANID to the INTERFACE statement in the TCP profile data set for each TCP/IP stack participating in the VLAN. Assign each VLAN a separate subnet address.
3. Start the TCP/IP stacks.

6.5.2 Virtual Telecommunications Access Method setup for VLAN HiperSockets

No additional Virtual Telecommunications Access Method (VTAM) customization is required.

6.5.3 TCP/IP profile customization for VLAN HiperSockets

The VLANID keyword must be coded on the INTERFACE statement associated with the HiperSockets CHPID. For this scenario, add the VLANID and assign a value to the TCPIP profile on each LPAR:

1. For SC30, specify VLANID 3 **1** and assign an address **2** in a subnet **3** for the VLAN, as shown in Example 6-1.

Example 6-1 VLAN configuration for TCPIPF on SC30

```
INTERFACE HIPERLFO              1
    DEFINE IPAQIDIO
    IPADDR 192.0.3.4/24         2
    CHPID F0
    VLANID 3


   BEGINROUTES
   ROUTE 192.0.3.0/24 = HIPERLFO MTU 8192  3
   ENDROUTES

START HIPERLFO
```

2. Because LPAR SC32 is to participate in VLAN 3 (with LPAR SC30), configure its TCPIPC stack to use the same VLANID 3 **1** and assign it an address **2** in the same subnet **3** as TCPIPA on LPAR SC30, as shown in Example 6-2.

Example 6-2 VLAN configuration for TCPIPC on SC32

```
INTERFACE HIPERLFO
    DEFINE IPAQIDIO
    IPADDR 192.0.3.6/24      2
    CHPID F0
    VLANID 3
1
BEGINROUTES
ROUTE 192.0.3.0/24 = HIPERLFO MTU 8192    3
ENDROUTES

START HIPERLFO
```

3. Configure the TCP/IP stack on LPAR SC31 to use a different VLANID and subnet from LPARs SC30 and SC32. Although they are all using the same HiperSockets channel (F0), they will not be able to communicate because of the different VLANIDs. For SC31, specify VLANID 2 **1** and assign an address **2** in a subnet **3** for the VLAN, as shown in Example 6-3.

Example 6-3 VLAN configuration for TCPIPA on SC31

```
INTERFACE HIPERLFO
    DEFINE IPAQIDIO
    IPADDR 192.0.2.5/24      2
    CHPID F0
    VLANID 2 1


BEGINROUTES
ROUTE 192.0.2.0/24 = HIPERLFO MTU 8192   3
ENDROUTES

START HIPERLFO
```

6.5.4 Verify VLAN implementation

This section shows the display commands used to verify the configuration.

TCP/IP startup

There are no messages issued during the TCP/IP stack initialization to indicate that the IP stack is part of a VLAN.

Verify the device and link

After the TCP/IP stack has started, issue the command **D TCPIP,procname,NETSTAT,DEV** to verify the VLANID 3 for the HiperSockets INTERFACE. In this example, it is HIPERLF0 2. For SC30, the display command output is shown in Example 6-4.

Example 6-4 Verify VLANID for SC30 (partial output)

```
INTFNAME: HIPERLF0     2     INTFTYPE: IPAQIDIO   INTFSTATUS: READY
    TRLE: IUTIQ4F0  DATAPATH: 7403     DATAPATHSTATUS: READY
    CHPID: F0
    IPBROADCASTCAPABILITY: NO
    ARPOFFLOAD: YES                  ARPOFFLOADINFO: YES
    CFGMTU: NONE                     ACTMTU: 8192
    IPADDR: 192.0.3.4/24
    VLANID: 3                            1
READSTORAGE: GLOBAL (2048K)
    SECCLASS: 255                    MONSYSPLEX: NO
    IQDMULTIWRITE: DISABLED
  MULTICAST SPECIFIC:
    SECCLASS: 255                    MONSYSPLEX: NO
```

For SC31, the output of the **D TCPIP,procname,NETSTAT,DEV** command is shown in Example 6-5.

Example 6-5 Verify VLANID for SC31 (partial output)

```
INTFNAME: HIPERLFO         INTFTYPE: IPAQIDIO   INTFSTATUS: READY
    TRLE: IUTIQ4FO  DATAPATH: 7402     DATAPATHSTATUS: READY
    CHPID: FO
    IPBROADCASTCAPABILITY: NO
    ARPOFFLOAD: YES                  ARPOFFLOADINFO: YES
    CFGMTU: NONE                     ACTMTU: 8192
    IPADDR: 192.0.2.5/24
    VLANID: 2
    READSTORAGE: GLOBAL (2048K)
    SECCLASS: 255                    MONSYSPLEX: NO
```

For SC32, the output of the **D TCPIP,procname,NETSTAT,DEV** command is shown in Example 6-6.

Example 6-6 Verify vlanid for SC32 (partial output)

```
INTFNAME: HIPERLFO         INTFTYPE: IPAQIDIO   INTFSTATUS: READY
    TRLE: IUTIQ4FO  DATAPATH: 7402     DATAPATHSTATUS: READY
    CHPID: FO
    IPBROADCASTCAPABILITY: NO
    ARPOFFLOAD: YES                  ARPOFFLOADINFO: YES
    CFGMTU: NONE                     ACTMTU: 8192
    IPADDR: 192.0.3.6/24
    VLANID: 3
    READSTORAGE: GLOBAL (2048K)
    SECCLASS: 255                    MONSYSPLEX: NO
    IQDMULTIWRITE: DISABLED
```

VLAN connectivity test

To test that no connectivity exists between the two VLANs with different VLANIDs, attempt to ping across the VLANs. There is no connectivity between VLAN 1 and VLAN 3, as shown in Example 6-7.

Example 6-7 Ping from SC30 (vlan 3) to SC31 (vlan 2)

```
===> ping 192.0.2.5 (tcp tcpipa
 CS V2R1: Pinging host 192.0.2.5
 Ping #1 timed out
```

There is connectivity between the two systems on the same VLAN (SC30 and SC32 are in VLAN 3), as shown in Example 6-8.

Example 6-8 Ping from SC30 (vlan 3) to SC32 (vlan 3)

```
===> ping 192.0.3.6 (tcp tcpipa)
CS V2R1: Pinging host 192.0.3.6
Ping #1 response took 0.000 seconds.
```

6.6 HiperSockets VLAN for a z/VM host system

IBM z/VM can use VLAN in a HiperSockets network to logically subdivide the network without needing additional hardware resources or HiperSockets networks.

6.6.1 VLAN definitions

The VLAN is defined and started like any other network. The VLAN parameter is added to the TCPIP PROFILE file LINK statement, as shown in Example 6-9.

Example 6-9 IBM z/VM TCPIP definition for VLAN

```
   ...
   DEVICE HIPERDF0 HIPERS 7400
   LINK HIPERLF0 QDIOIP      HIPERDF0 NOFWD VLAN 2
   ...
   HOME
   192.0.2.1 HIPERLF0
   ...
   GATEWAY
   ; Network       Subnet          First            Link      MTU
   ; Address       Mask            Hop              Name      Size
   ; ------------- --------------- ---------------- --------- -----
   192.0.2.0       255.255.255.0   =                HIPERLF0 8192
   ; ------------- --------------- ---------------- --------- ----
   START HIPERDF0
   ...
```

6.6.2 VLAN verification

To verify the connection to the VLAN, use the **NETSTAT DEVLINKS** command. In Example 6-10, the device HIPERDF0 is connected to VLAN 2. Only systems with devices connected to VLAN 2 will be able to communicate with each other.

Example 6-10 Display TCP/IP VLAN connection

```
netstat devl
VM TCP/IP Netstat Level 520

Device HIPERDF0 Type: HIPERS         Status: Ready
  Queue size: 0     CPU: 0     Address: 7400       Port name: UNASSIGNED
  IPv4 Router Type: NonRouter  Arp Query Support: No
    Link HIPERLF0 Type: QDIOIP         Net number: 0
      BytesIn: 0                BytesOut: 308
      Forwarding: Disabled      MTU: 8192
      Maximum Frame Size  : 16384
      VLAN ID: 2
      Broadcast Capability: Yes
      Multicast Capability: Yes
      Group                                   Members
      -----                                   -------
      224.0.0.1                                  1
```

6.7 HiperSockets VLAN in Linux on System z

This section demonstrates that Linux on System z can use VLAN in a HiperSockets network to logically subdivide a network. Start with a HiperSockets network as described in 3.1, “Test configuration” on page 30. You recall that the device addresses being assigned to the various systems are on the same CHPID, and therefore on the same HiperSockets network.

This section shows the commands needed to set up a VLAN for the environment shown in Figure 6-3 for the Red Hat Enterprise Linux (RHEL) and SUSE Linux Enterprise Server (SLES) guest servers.

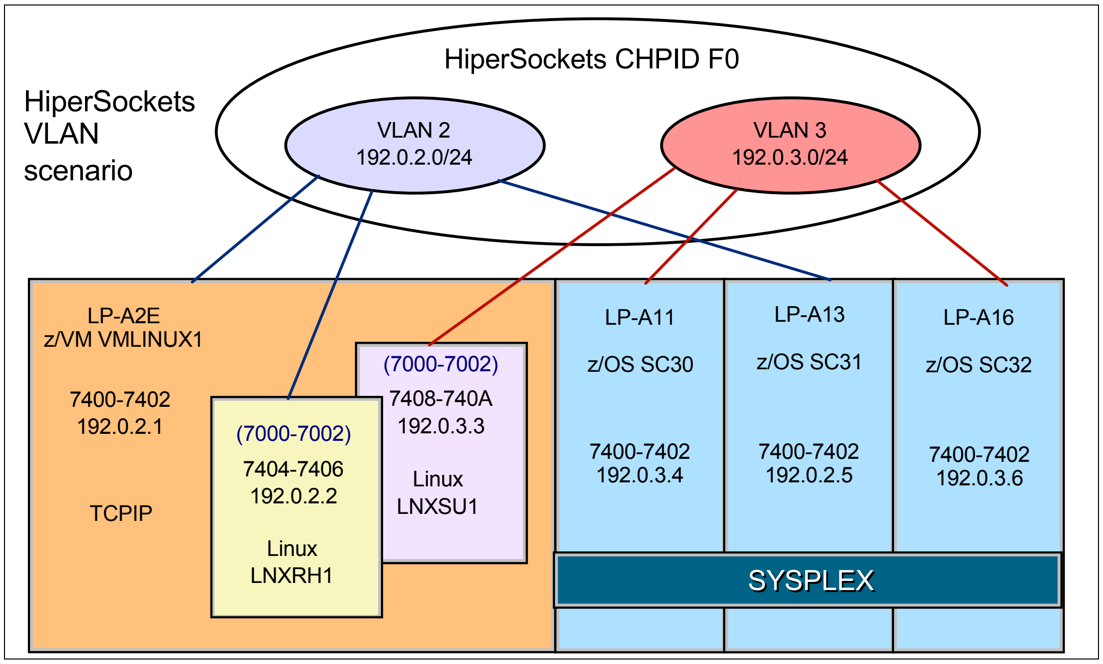


Figure 6-3 VLAN test configuration

The only difference for the two Linux guest systems is specifying the VLAN number on the **vconfig** and **ifconfig** commands. The **vconfig** command defines VLAN 2 on the RHEL server, and VLAN 3 on the SLES server, to the HiperSockets interfaces.

The important thing to note in the examples is that the HiperSockets network is up before the VLAN is defined (that is, before the network is subdivided) and started. The other commands show the process of defining and starting the VLAN. Start with the LNXRH1 system.

6.7.1 Temporary VLAN for RHEL—VLAN 2 on LNXRH1

As described in 3.5.2, “Linux configuration example” on page 49, you have defined `hsi0` as HiperSockets interface by using the **znetconf** command, or added it to the `ifcfg-hsi0` file. Example 6-11 shows the configuration of `hsi0` at this point.

Example 6-11 Displaying HiperSockets network definition

```
[root@lnxrh1 ~]# ifconfig hsi0
hsi0      Link encap:Ethernet  HWaddr 06:00:F0:2E:00:06
          inet addr:192.0.1.2  Bcast:192.0.1.255  Mask:255.255.255.0
          inet6 addr: fe80::400:f0ff:fe2e:6/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:8192  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:830 (830.0 b)
```

To create the VLAN, follow these steps:

1. In addition to this standard HiperSockets interface, create a new interface for VLAN 2 by using the **vconfig** command, as shown in Example 6-12.

   Example 6-12 Adding a VLAN interface

   ```
   [root@lnxrh1 ~]# vconfig add hsi0 2
   Added VLAN with VID == 2 to IF -:hsi0:-
   ```

   The new VLAN interface can be seen in the Linux VLAN configuration file and in the interface configuration, as shown in Example 6-13.

   Example 6-13 Displaying the VLAN interface

   ```
   [root@lnxrh1 ~]# cat /proc/net/vlan/config
   VLAN Dev name    | VLAN ID
   Name-Type: VLAN_NAME_TYPE_RAW_PLUS_VID_NO_PAD
   hsi0.2 | 2| hsi0
   [root@lnxrh1 ~]# ifconfig hsi0.2
   hsi0.2    Link encap:Ethernet  HWaddr 06:00:F0:2E:00:06
             BROADCAST MULTICAST  MTU:8192  Metric:1
             RX packets:0 errors:0 dropped:0 overruns:0 frame:0
             TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:0
             RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
   ```

2. If you compare Example 6-13 with Example 6-11 on page 78, you will note the missing second and third lines, in addition to the fields missing from the fourth line from the hsi0 information. This is because hsi0.2 is defined only and needs to be started:

   ```
   ifconfig hsi0.2 192.0.2.2 netmask 255.255.255.0 up
   ```

   Now you see that the VLAN 2subnet is up and running. See Example 6-14.

   Example 6-14 VLAN 2 is running

   ```
   [root@lnxrh1 ~]# ifconfig hsi0.2
   hsi0.2    Link encap:Ethernet  HWaddr 06:00:F0:2E:00:06
             inet addr:192.0.2.2  Bcast:192.0.2.255  Mask:255.255.255.0
             inet6 addr: fe80::400:f0ff:fe2e:6/64 Scope:Link
             UP BROADCAST RUNNING MULTICAST  MTU:8192  Metric:1
             RX packets:0 errors:0 dropped:0 overruns:0 frame:0
             TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:0
             RX bytes:0 (0.0 b)  TX bytes:492 (492.0 b)
   ```

6.7.2 Temporary VLAN for SLES11—VLAN 3 on LNXSU1

Further subdivide the HiperSockets network on the LNXSU1 system by setting up another VLAN. Example 6-15 shows the existing HiperSockets interface.

Example 6-15 HiperSockets is up

```
lnxsu1:~ # ifconfig hsi1
hsi1      Link encap:Ethernet  HWaddr 06:00:F0:2E:00:0A
          inet addr:192.0.1.3  Bcast:192.0.1.255  Mask:255.255.255.0
          inet6 addr: fe80::400:f0ff:fe2e:a/64 Scope:Link
          UP BROADCAST RUNNING NOARP MULTICAST  MTU:8192  Metric:1
          RX packets:4 errors:0 dropped:0 overruns:0 frame:0
          TX packets:50 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:336 (336.0 b)  TX bytes:4800 (4.6 Kb)
```

To create a temporary VLAN, follow these steps:

1. Create a new interface for VLAN 3 by using the **vconfig** command, as shown in Example 6-16.

   Example 6-16 Adding a VLAN

   ```
   lnxsu1:~ # vconfig add hsi1 3
   Added VLAN with VID == 3 to IF -:hsi1:-
   ```

2. Define the network address for `hsi1.3`, as shown in Example 6-17.

   Example 6-17 Defining a network address

   ```
   lnxsu1:~ # ifconfig hsi1.3 192.0.3.3 netmask 255.255.255.0 up
   ```

3. Example 6-18 shows the new interface `hsi1.3`.

   Example 6-18 Displaying the interface

   ```
   lnxsu1:~ # cat /proc/net/vlan/config
   VLAN Dev name    | VLAN ID
   Name-Type: VLAN_NAME_TYPE_RAW_PLUS_VID_NO_PAD
   hsi1.3         | 3  | hsi1
   lnxsu1:~ # ifconfig hsi1.3
   hsi1.3    Link encap:Ethernet  HWaddr 06:00:F0:2E:00:0A
             inet addr:192.0.3.3  Bcast:192.0.3.255  Mask:255.255.255.0
             inet6 addr: fe80::400:f0ff:fe2e:a/64 Scope:Link
             UP BROADCAST RUNNING NOARP MULTICAST  MTU:8192  Metric:1
             RX packets:0 errors:0 dropped:0 overruns:0 frame:0
             TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:0
             RX bytes:0 (0.0 b)  TX bytes:210 (210.0 b)
   ```

6.7.3 Verifying your setup

Using **ping**, confirm that the VLAN addresses `192.0.2.2` and `192.0.3.3` cannot be reached by the other Linux guest. They both can reach the VLAN-defined IP addresses of the z/OS LPARs and the z/VM stack that were defined in the previous sections, if they are on the same HiperSockets network and the same VLAN.

Note that the two Linux guests can still communicate without VLAN: `192.0.1.2` and `192.0.1.3` can still ping each other. If you want to remove these non-VLAN interfaces, you can remove their IP addresses from the respective config files, as shown in the next sections.

6.7.4 Permanent VLAN definition for Red Hat Enterprise Linux

To define a permanent VLAN for RHEL, follow these steps:

1. To make your VLAN interface permanent, create a new file `/etc/sysconfig/network-scripts/ifcfg-hsi0.2`, as shown in Example 6-19.

Example 6-19 Content of `/etc/sysconfig/network-scripts/ifcfg-hsi0.2`

```
DEVICE=hsi0.2
BOOTPROTO=static
IPADDR=192.0.2.2
NETMASK=255.255.255.0
ONBOOT=yes
VLAN=yes
```

2. If you want to change the definition of the base interface in `/etc/sysconfig/network-scripts/ifcfg-hsi0`, you can do so, as shown in Example 6-20.

Example 6-20 Content of `/etc/sysconfig/network-scripts/ifcfg-hsi0`

```
DEVICE=hsi0
BOOTPROTO=static
NETTYPE=qeth
ONBOOT=yes
SUBCHANNELS=0.0.7000,0.0.7001,0.0.7002
TYPE=Ethernet
```

3. Now only VLAN traffic is possible over the HiperSockets network, as shown in Example 6-21.

Example 6-21 Result of ifconfig

```
[root@lnxrh1 ~]# ifconfig
[...]
hsi0      Link encap:Ethernet  HWaddr 06:00:F0:2E:00:06
          inet6 addr: fe80::400:f0ff:fe2e:6/64 Scope:Link
          UP BROADCAST RUNNING NOARP MULTICAST  MTU:8192  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:780 (780.0 b)

hsi0.2    Link encap:Ethernet  HWaddr 06:00:F0:2E:00:06
          inet addr:192.0.2.2  Bcast:192.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::400:f0ff:fe2e:6/64 Scope:Link
          UP BROADCAST RUNNING NOARP MULTICAST  MTU:8192  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:390 (390.0 b)
[...]
```

6.7.5 Permanent VLAN definition for SLES11

To define a permanent VLAN for SLES, follow these steps:

1. To make your VLAN interface permanent, create a new file `/etc/sysconfig/network/ifcfg-hsi1.3`, as shown in Example 6-22.

   Example 6-22 Contents of `/etc/sysconfig/network/ifcfg-hsi1.3`

   ```
   BOOTPROTO='static'
   IPADDR='192.0.3.3'
   BROADCAST='192.0.3.255'
   STARTMODE='auto'
   NAME='ITSO HiperSockets Network CHPID F0 (0.0.7000)'
   USERCONTROL='no'
   NETMASK='255.255.255.0'
   VLAN='yes'
   ETHERDEVICE='hsi1'
   ```

2. If wanted, the IP address information for the base interface in `/etc/sysconfig/network/ifcfg-hsi` can be commented out, as shown in Example 6-23.

   Example 6-23 Contents of `/etc/sysconfig/network/ifcfg-hsi1`

   ```
   BOOTPROTO='static'
   #IPADDR='192.0.1.3'
   #BROADCAST='192.0.1.255'
   STARTMODE='auto'
   #NAME='ITSO HiperSockets Network CHPID F0 (0.0.7000)'
   USERCONTROL='no'
   #NETMASK='255.255.255.0'
   ```

3. Now only VLAN traffic is possible over the HiperSockets network, as shown in Example 6-24.

   Example 6-24 Result of ifconfig

   ```
   lnxsu1:/etc/sysconfig/network # ifconfig
   [...]
   hsi1      Link encap:Ethernet  HWaddr 06:00:F0:2E:00:0A
             inet6 addr: fe80::400:f0ff:fe2e:a/64 Scope:Link
             UP BROADCAST RUNNING NOARP MULTICAST  MTU:8192  Metric:1
             RX packets:25 errors:0 dropped:0 overruns:0 frame:0
             TX packets:123 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:1000
             RX bytes:2100 (2.0 Kb)  TX bytes:11318 (11.0 Kb)

   hsi1.3    Link encap:Ethernet  HWaddr 06:00:F0:2E:00:0A
             inet addr:192.0.3.3  Bcast:192.0.3.255  Mask:255.255.255.0
             inet6 addr: fe80::400:f0ff:fe2e:a/64 Scope:Link
             UP BROADCAST RUNNING NOARP MULTICAST  MTU:8192  Metric:1
             RX packets:0 errors:0 dropped:0 overruns:0 frame:0
             TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:0
             RX bytes:0 (0.0 b)  TX bytes:210 (210.0 b)
   [...]
   ```

6.8 HiperSockets VLAN in z/VSE

IBM z/VSE has provided VLAN support for Open Systems Adapter (OSA)-Express CHPID types OSD (OSA-Express for queued direct input/output (QDIO)) and OSX (OSA for IEDN), and HiperSockets devices since version 5.1:

- In a Layer 3 configuration, VLANs can be transparently used by IPv6/VSE and Transmission Control Protocol/Internet Protocol (TCP/IP) for VSE/ESA.
- If you want to configure VLANs for OSA-Express (CHPID types OSD and OSX) devices in a Layer 2 configuration that carries IPv6 traffic, you require the IPv6/VSE product.

You can use one of the following ways to configure your system to use VLAN:

- Configure one or more VLANs in the TCP/IP stack of IPv6/VSE using the LINK command. For details about IPv6/VSE commands, see *IPv6/VSE Installation Guide*, SC34-2616.
- Generate and catalog phase IJBOCONF containing the global VLANs to be used with your OSAX devices. IBM z/VSE provides skeleton SKOSACFG to generate phase IJBOCONF. The VLANs contained in IJBOCONF can be transparently used for Layer 3 links by IPv6/VSE and TCP/IP for VSE/ESA. See *z/VSE Planning,* SC34-2635 for details.

Example 6-25 shows a sample configuration setting VLAN ID 200 for Device D00.

Example 6-25 Sample SKOSACFG configuration

```
* $$ JOB JNM=IJBOCONF,CLASS=A,DISP=D
// JOB IJBOCONF GENERATE IJBOSA MODULE CONFIGURATION PHASE
// LIBDEF *,CATALOG=PRD2.CONFIG
// LIBDEF *,SEARCH=PRD1.BASE
// OPTION ERRS,SXREF,SYM,NODECK,CATAL,LISTX
PHASE IJBOCONF,*
// EXEC ASMA90,SIZE=(ASMA90,64K)
IJBOCONF CSECT
IJBOCONF AMODE ANY
IJBOCONF RMODE ANY
*
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                     *
* GLOBAL VLAN DEFINITION                                              *
*                                                                     *
* DEFGVLAN DEVNO=<CUU>, VLAN_ID=<ID>, VLAN_PRIO=<PRIO>                *
* <CUU> VSE DEVICE NUMBER IN HEX FORMAT                               *
* <ID> VLAN ID IN DECIMAL FORMAT (1 ... 4095)                         *
* <PRIO> VLAN PRIORITY (VALID VALUES 0 ... 7)                         *
*                                                                     *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*
DEFGVLAN DEVNO=0D00,VLAN_ID=200
*
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                     *
* ONLY ONE GLOBAL VLAN CAN BE DEFINED PER SUBCHANNEL.                  *
* IF GLOBAL VLAN IS DEFINED, NO USUAL VLAN(S) MAY BE DEFINED          *
* ON THE SAME SUBCHANNEL.                                             *
*                                                                     *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
END
/*
// IF $MRC GT 4 THEN
// GOTO NOLINK
// EXEC LNKEDT,PARM='MSHP'
/. NOLINK
/&
* $$ EOJ
```

The job template SKOSACFG is located in ICCF library 59.

7

More HiperSockets features

This chapter provides information about several features that are provided by IBM System z HiperSockets firmware to help you in special scenarios:

- The *HiperSockets multiple write facility (multiwrite)* provides you with even more throughput for large messages, by offering the ability to transfer more than one message per firmware instruction. It is used by IBM z/OS.
- The *HiperSockets network traffic analyzer* (HS NTA) gives you detailed insights into the behavior of your HiperSockets network, by tracing the messages that are being transferred into a Linux tool. You can trace messages from all operating systems (OSs). Special authorization through the Support Element (SE) is required to protect your security when using the HiperSockets network traffic analyzer.
- The *HiperSockets completion queue* feature enables HiperSockets firmware to transfer messages synchronously when possible, and asynchronously when needed, therefore increasing the overall system performance. It is used by the IBM z/VM Virtual Switch's HiperSockets Bridge Port, and by Linux and IBM z/VSE for inter-user communication vehicle (IUCV) Sockets over HiperSockets and Linux Fast Path (LFP).

7.1 HiperSockets multiple write facility

The HiperSockets multiple write facility moves multiple buffers of data with a single write operation. This facility was added to reduce processor use, and to improve performance for large outbound messages over HiperSockets. The receiving partition can process larger amounts of data per input/output (I/O) interrupt. The improvement is not apparent to the OS in the receiving partition.

Multiple writes with fewer I/O interrupts reduce processor use of both the sending and receiving logical partitions (LPARs). This has been available since the IBM System z10.

7.1.1 HiperSockets multiwrite for z/OS

The HiperSockets multiple write facility has been supported in z/OS since V1R9 with program temporary fixes (PTFs).

When enabled, HiperSockets multiwrite is used any time a message spans the HiperSockets frame size, consequently requiring multiple output buffers to transfer the message. Therefore, it will only be used for larger outbound messages. Spanning multiple output data buffers can be affected by several factors:

- HiperSockets frame size
- Application socket send size
- Transmission Control Protocol (TCP) send size
- Maximum transmission unit (MTU) size

The HiperSockets multiple write facility is disabled by default. To enable it on all HiperSockets interfaces, including interfaces created for dynamic cross-system coupling facility (XCF), add the `IQDMULTIWRITE` parameter to the `GLOBALCONFIG` statement. If the `GLOBALCONFIG` parameters are changed with the `VARY TCPIP,,OBEYFILE` command, the new values will not take effect for an active HiperSockets interface until you stop and restart the interface.

Restriction: HiperSockets multiwrite is not supported when running as a guest in a z/VM environment.

For more information, see the following sources:

- *IBM z/OS V2R1 Communications Server TCP/IP Implementation Volume 1: Base Functions, Connectivity, and Routing*, SG24-8096
- The z/OS V2R1 Information Center

  http://pic.dhe.ibm.com/infocenter/zos/v2r1/index.jsp

IBM System z Integrated Information Processor-assisted HiperSockets for large messages

IBM System z Integrated Information Processor (zIIP) is a speciality processor designed to make more general computing capacity available, and to lower software costs for selected workloads.

With z/OS V1R10 and later, HiperSockets multiwrite operations can also be offloaded to a zIIP for further savings. The zIIP offload is only supported for TCP traffic that originates in this host. For example, it cannot be used for sysplex distributor or Enterprise Extender (EE) traffic.

IBM z/OS application workloads that are based on Extensible Markup Language (XML), Hypertext Transfer Protocol (HTTP), SOAP, Java, and traditional file transfer methods, can benefit from zIIP enablement by lowering general-purpose processor use.

IBM zIIP-assisted HiperSockets multiwrite is disabled by default. To enable it, specify the **ZIIP IQDIOMULTIWRITE** parameter on the **GLOBALCONFIG** statement. Notice that the **IQDMULTIWRITE** parameter that turns on the function is spelled differently than the **IQDIOMULTIWRITE** parameter that enables it for zIIP assist. Figure 7-1 illustrates the configuration parameters for z/OS.

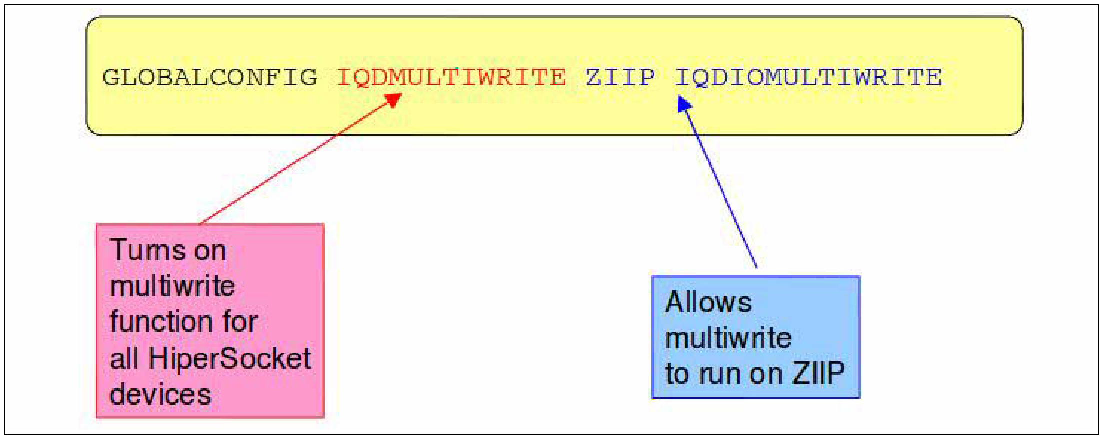

GLOBALCONFIG IQDMULTIWRITE ZIIP IQDIOMULTIWRITE
Turns on multiwrite function for all HiperSocket devices
Allows multiwrite to run on ZIIP

Figure 7-1 IBM zIIP-assisted HiperSockets multiwrite configuration example

7.2 HiperSockets network traffic analyzer

HS NTA is a function that can make problem isolation and resolution simpler by allowing layer 2 and layer 3 tracing of HiperSockets network traffic.

7.2.1 Overview

HS NTA support has been available since System z10. A Linux on System z implementation of the HS NTA has been available since:

- Novell SUSE Linux Enterprise Server (SLES) 11 SP1
- Red Hat Enterprise Linux (RHEL) 6.0

You can use HS NTA for Linux on System z to trace network traffic between any endpoints on a given HiperSockets network. The endpoints do not have to be Linux, but can be traffic between any OSs in any logical partitions (LPARs) that have access to this HiperSockets network.

HS NTA for Linux on System z captures records into host memory and storage (file systems). These records can be analyzed by system programmers and network administrators by using Linux on System z tools to format, edit, and process the trace records.

To protect the security of your HiperSockets networks, the use of the HS NTA feature needs to be authorized by the system administrator on the SE. On the SE you can specify, for each HiperSockets LAN, in which LPAR an NTA can be started and which LPARs are eligible for being traced.

A network traffic analyzer traces messages from and to devices with the same channel identifier (CHID) that the NTA belongs to. An NTA receives one copy of each message that was successfully delivered *from* a sender LPAR that is eligible to be traced by the NTA, *to* at least one target LPAR that is also eligible to be traced by the NTA. Figure 7-2 shows an example of this trace.

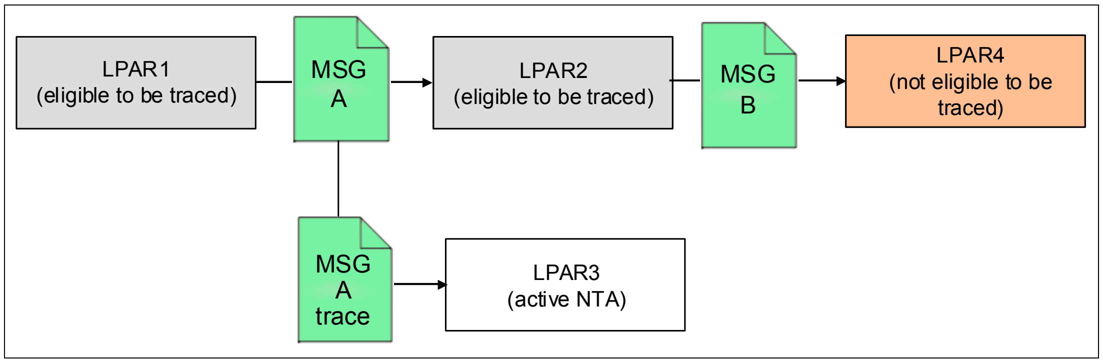


Figure 7-2 Message A is traced by NTA; Message B is not traced

An active NTA can trace layer 2 messages and layer 3 messages.

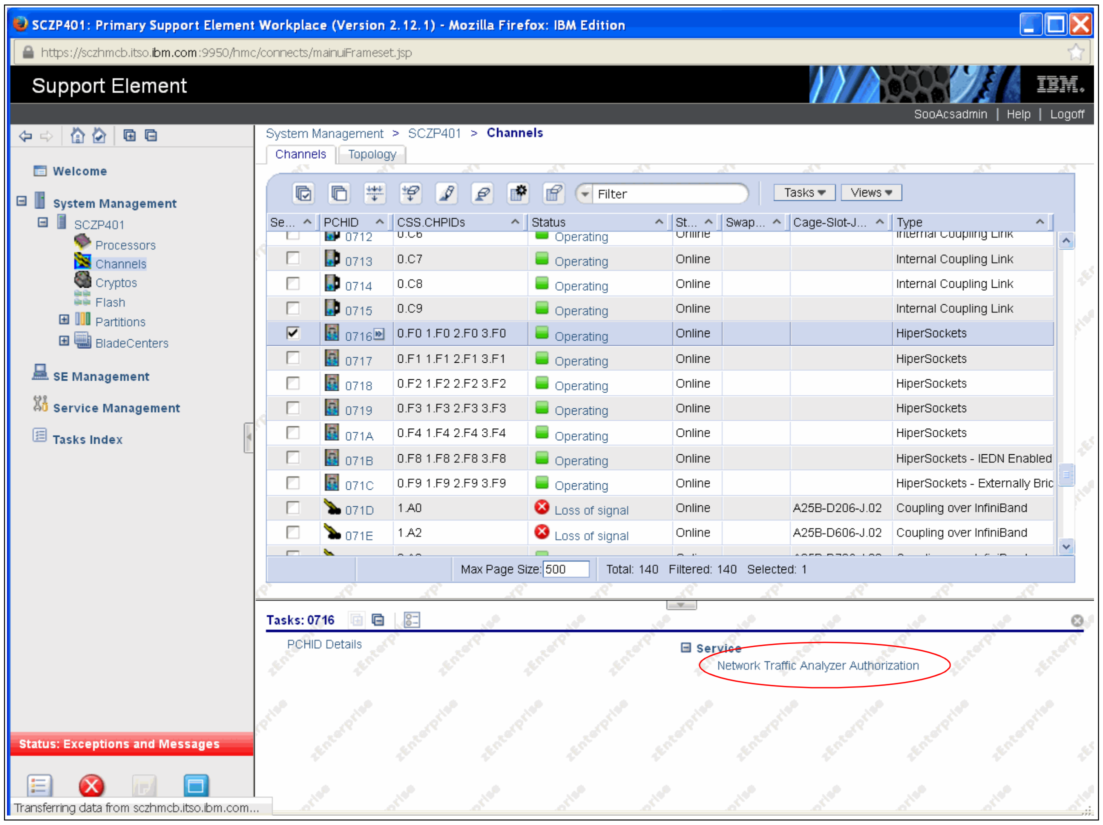
Because the HiperSockets transfer instruction (run by the processor) has to serve the Network Traffic Analyzer also, this instruction will take longer to finish. This affects performance and also uses more processor cycles at the sender side.

7.2.2 NTA authorization on the SE

HS NTA rules can be set up on the SE. Log on with the user ID `ACSADMIN` to modify HS NTA rules. Select the HiperSockets channel (internal queued direct communication (IQD) channel) for which the NTA configuration has to be modified. Figure 7-3 on page 89 shows this process. To set the rules, follow these steps:

1. In the menu of the selected IQD channel, expand **Service** and select **Network Traffic Analyzer Authorization**.

I


SCZP401: Primary Support Element Workplace (Version 2.12.1) - Mozilla Firefox: IBM Edition

https://sczhmcb.itso.ibm.com:9950/hmc/connects/mainuiFrameset.jsp

Support Element

SooAcsadmin | Help | Logoff

System Management > SCZP401 > Channels

Channels | Topology

Se...	PCHID	CSS.CHPIDs	Status	St...	Swap...	Cage-Slot-J...	Type
	0712	0.C6	🟩 Operating	Online			Internal Coupling Link
	0713	0.C7	🟩 Operating	Online			Internal Coupling Link
	0714	0.C8	🟩 Operating	Online			Internal Coupling Link
	0715	0.C9	🟩 Operating	Online			Internal Coupling Link
✓	0716	0.F0 1.F0 2.F0 3.F0	🟩 Operating	Online			HiperSockets
	0717	0.F1 1.F1 2.F1 3.F1	🟩 Operating	Online			HiperSockets
	0718	0.F2 1.F2 2.F2 3.F2	🟩 Operating	Online			HiperSockets
	0719	0.F3 1.F3 2.F3 3.F3	🟩 Operating	Online			HiperSockets
	071A	0.F4 1.F4 2.F4 3.F4	🟩 Operating	Online			HiperSockets
	071B	0.F8 1.F8 2.F8 3.F8	🟩 Operating	Online			HiperSockets - IEDN Enabled
	071C	0.F9 1.F9 2.F9 3.F9	🟩 Operating	Online			HiperSockets - Externally Bric
	071D	1.A0	✖ Loss of signal	Online		A25B-D206-J.02	Coupling over InfiniBand
	071E	1.A2	✖ Loss of signal	Online		A25B-D606-J.02	Coupling over InfiniBand

Max Page Size: 500 Total: 140 Filtered: 140 Selected: 1

Tasks: 0716

PCHID Details

Service

Network Traffic Analyzer Authorization

Status: Exceptions and Messages

Transferring data from sczhmcb.itso.ibm.com...

Figure 7-3 Channel view window

2. Select this subentry if you want to modify the HS NTA rules.

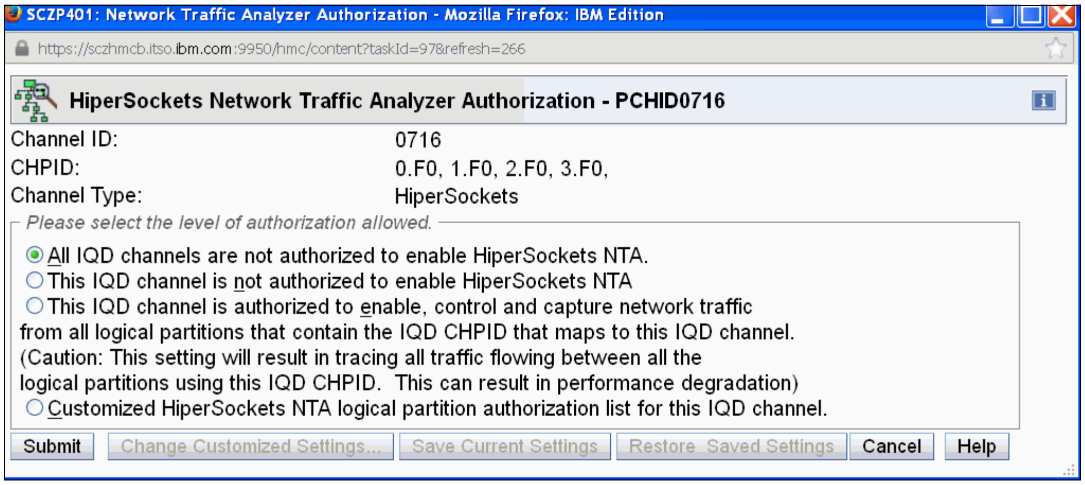
   Figure 7-4 shows that there are four types of rules for the HS NTA:

   – Tracing is disabled for all IQD channels in the system (the default rule).

   – Tracing is disabled for this IQD channel.

   – Tracing is allowed for this IQD channel. (All LPARS can be set up for NTA, and all LPARs are eligible to be traced by an active NTA.)

   – Tracing is customized by an authorization list for this IQD channel.


SCZP401: Network Traffic Analyzer Authorization - Mozilla Firefox: IBM Edition

https://sczhmcb.itso.ibm.com:9950/hmc/content?taskId=97&refresh=266

HiperSockets Network Traffic Analyzer Authorization - PCHID0716

Channel ID:	0716
CHPID:	0.F0, 1.F0, 2.F0, 3.F0,
Channel Type:	HiperSockets

Please select the level of authorization allowed.

[✓ All IQD channels are not authorized to enable HiperSockets NTA.]
○ This IQD channel is not authorized to enable HiperSockets NTA
○ This IQD channel is authorized to enable, control and capture network traffic from all logical partitions that contain the IQD CHPID that maps to this IQD channel. (Caution: This setting will result in tracing all traffic flowing between all the logical partitions using this IQD CHPID. This can result in performance degradation)
○ Customized HiperSockets NTA logical partition authorization list for this IQD channel.

Figure 7-4 HiperSockets NTA authorization window

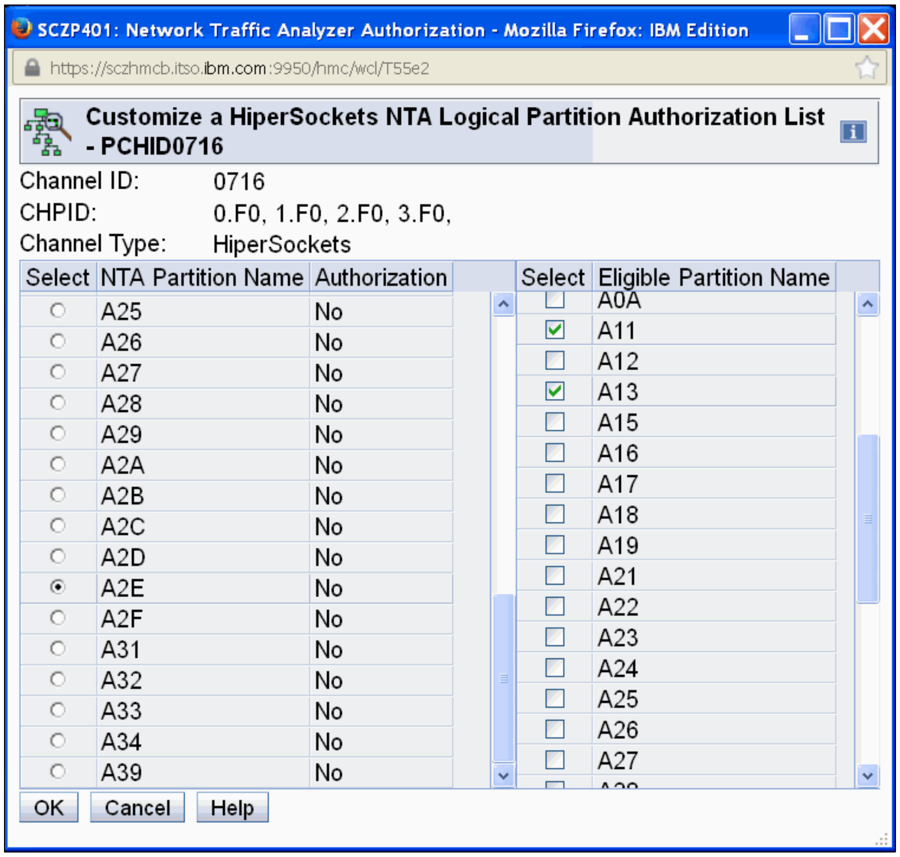
The HS NTA feature is not authorized per default. You can authorize it by channel, or define a customized authorization list. A customized authorization list enables you to authorize one or more LPARs that are authorized to start an NTA, and also define which LPARs are eligible to be traced on the selected IQD channel.

Therefore, if you select LPAR-A as an NTA partition, and LPAR-B and LPAR-C as eligible to be traced partitions, LPAR-A is only able to trace messages successfully sent between LPAR-B and LPAR-C. If LPAR-D also communicates over the selected IQD channel, the NTA will not be able to trace any of the messages sent from or to LPAR-D. At least one partition that is eligible to be traced must be selected to enable an NTA on the NTA partition.

If a z/VM hypervisor runs in an LPAR, the respective settings apply to all virtual machine (VM) guests in that LPAR.

3. You can set up a customized NTA authorization list by selecting the corresponding option in the Network Traffic Analyzer Authorization window. This enables the **Change Customized Settings** button in the same dialog box. Click **Change Customized Settings** to open the Customize a HiperSockets NTA Logical Partition Authorization List window, as shown in Figure 7-5.

   In this window, it is possible to specify which LPAR is authorized to enable an NTA, and which LPAR is eligible to be traced by this NTA. To apply modified NTA rules to the system, press **Submit**. As mentioned previously, only messages that are successfully delivered between at least two LPARs that are eligible to be traced by the NTA are also delivered to the active NTA.


SCZP401: Network Traffic Analyzer Authorization - Mozilla Firefox: IBM Edition
https://sczhmcb.itso.ibm.com:9950/hmc/wcl/T55e2

Customize a HiperSockets NTA Logical Partition Authorization List - PCHID0716

Channel ID:	0716
CHPID:	0.F0, 1.F0, 2.F0, 3.F0,
Channel Type:	HiperSockets

Select	NTA Partition Name	Authorization
○	A25	No
○	A26	No
○	A27	No
○	A28	No
○	A29	No
○	A2A	No
○	A2B	No
○	A2C	No
○	A2D	No
●	A2E	No
○	A2F	No
○	A31	No
○	A32	No
○	A33	No
○	A34	No
○	A39	No

Select	Eligible Partition Name
☐	A0A
✓	A11
☐	A12
✓	A13
☐	A15
☐	A16
☐	A17
☐	A18
☐	A19
☐	A21
☐	A22
☐	A23
☐	A24
☐	A25
☐	A26
☐	A27

Figure 7-5 Customize HS NTA LPAR authorization list

When HS NTA authorization is changed, a security log is taken on the SE. This security log contains details as to what the change was (such as what channel is authorized for NTA).

If the input/output configuration data set (IOCDS) is not changed, the HS NTA authorization will be preserved across power-on resets (PORs). If the IOCDS is changed, then another security log will be issued during the POR to say that all HS NTA authorization has been disabled.

7.2.3 HiperSockets NTA for Linux on System z

The `qeth` Linux on System z HiperSockets network interface driver provides a **`sysfs`** attribute called **`sniffer`**. If this attribute is set to a value of `1` for a HiperSockets network interface, and the NTA rules are set up accordingly for the specific LPAR, an NTA can be activated by every user space application that opens a *raw* socket on this network interface (for instance, **`tcpdump`**). The interface must be **`online`** and **`up`**, and must not be defined as a layer 2 interface.

Note: The sniffer attribute *cannot* be defined when the Linux instance is a z/VM guest. Only Linux instances running natively in an LPAR can define the sniffer attribute.

Note that it is not possible to assign Internet Protocol (IP) or message authentication code (MAC) addresses to a HiperSockets NTA.

To set up an HS NTA interface manually, follow these steps:

1. Issue the following commands:

   ```
   [root@xyz ~]# znetconf -a 1234 -o layer2=0 -o sniffer=1
   [root@xyz ~]# ip link set hsi1 up
   ```

2. Start tracing with the following command:

   ```
   [root@xyz ~]# tcpdump -i hsi1 [<further options>]
   ```

   This triggers the delivery of authorized HiperSockets-LAN-packets to the `hsi1` interface.

3. The **`tcpdump`** command can also be started for a non-NTA HiperSockets network interface configured with the **`sniffer`** attribute set to value `0`. In this case, only packets sent to and received from this interface are dumped.

   Note: If the Linux on System z you are using does not support the HS NTA function, the following command will fail with a message:

   ```
   echo 1 > /sys/bus/ccwgroup/devices/0.0.1234/sniffer
   /sys/bus/ccwgroup/devices/0.0.1234/sniffer: No such file or directory
   ```

4. Due to the synchronous nature of HiperSockets, situations can occur where a message is delivered to the actual target, but cannot be delivered to the active NTA because of lack of empty inbound buffers at the NTA. In this case, the `dropped packet counter` of the NTA interface is increased.

   ```
   [root@xyz ~]# ifconfig hsi 1 | grep "RX packets"
      RX packets:6789 errors:0 dropped:5 overruns:0 frame:0
   [root@xyz ~]# tcpdump -i hsi1
      tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
      listening on hsi1, link-type EN10MB (Ethernet), capture size 96 bytes
      ...
      5 packets dropped by kernel
   ```

7.2.4 Reference

An HS NTA FAQ document can be found at the following website:

http://ibm.biz/BdRFgz

A link to the FAQ can be found at the following website:

http://www.ibm.com/systems/z/hardware/networking/products.html#hipersockets

7.3 Completion queue function

The HiperSockets completion queue function is designed to enable HiperSockets to transfer data synchronously if possible, or asynchronously if necessary. This combines ultra-low latency with more tolerance for traffic peaks. With the asynchronous support, during high volume situations data can be temporarily held until the receiver has buffers available in its inbound queue. This provides end-to-end performance improvement for communication inside the central processor complex (CPC).

As explained in 4.1, “HiperSockets for highest performance” on page 60, the synchronous nature is a reason for HiperSockets outstanding performance and minimal latency. However, in cases where the receiving server runs out of empty inbound buffers, the sending device driver needs to handle these conditions.

This usually means special handling outside of the optimized main path. It can lead to a decrease in overall performance if it happens too often. As described in 4.5, “Input buffer count” on page 61, increasing the input buffer count on the target system helps avoid these problems.

The HiperSockets completion queue function solves these problems by buffering the messages at the sender side in cases where they temporarily cannot be delivered to the receiver, transferring the data synchronously if possible and asynchronously if necessary. The device driver software at the sender side needs to use this function. No special support on the receiver side is required. Currently, the following software components use the HiperSockets completion queue function:

- HiperSockets Bridge Port for z/VM Virtual Switch
- Inter-user communication vehicle (IUCV)

The HiperSockets completion queue function has been available since IBM zEnterprise 196 (z196). IBM z/VM 6.3 and later provides support for completion queue usage by z/VM guests with queued direct input/output (QDIO) Enhanced Buffer-State Management (QEBSM).

7.3.1 Details about completion queue

For completion queue usage, the OS’s device driver needs to provide a completion queue in addition to the internal QDIO (iQDIO) output and input queues that were described in 1.4, “HiperSockets mode of operation” on page 4. With a special instruction, the device driver informs the firmware that there is data available in the output queue that can be sent synchronously or asynchronously. If there are empty buffers at the target side, the data is transferred exactly the same way as without the HiperSockets completion queue function.

If there are no empty buffers, the data remains in the send buffer at the sender side, the sending device driver is informed that this send buffer cannot be freed up yet, and the transfer can continue with the next send buffer.

When (some time later) the receiving side provides empty inbound buffers, firmware will automatically move the pending data and then post completion events into the sender's completion queue. This informs the device driver that the output buffers can now be freed.

7.3.2 Completion queue for the z/VM HiperSockets Bridge Port

The HiperSockets Bridge Port for z/VM Virtual Switch is explained in 8.4, "The z/VM Virtual Switch with HiperSockets bridge port" on page 112.

The z/VM implementation of the HiperSockets Bridge Port is a highly efficient mechanism that moves the data directly from the OSA's input queue to the HiperSockets network. If that mechanism has to deal with target full conditions, it reduces overall performance for the other targets as well. By using the completion queue function, the z/VM Bridge Port leaves that to HiperSockets firmware and can continue to bridge the next packet from Open Systems Adapter (OSA) to HiperSockets without additional delay.

7.3.3 IUCV Sockets over HiperSockets (Linux, z/VSE)

IUCV was first implemented as a z/VM function that provides a fast, reliable communication path between two z/VM guests running under the same z/VM system. This concept of a fast, reliable, point-to-point connection was transferred to IUCV Sockets over HiperSockets, where a HiperSockets connection with completion queue support is used to create an IUCV Socket. In this case, the completion queue function is used on both sides of the connection, and is used as a congestion control mechanism to form a reliable connection.

Because IUCV Sockets are based on a reliable communication layer, they require many fewer resources than User Datagram Protocol (UDP) or TCP sockets. There is no need for sequence numbers, acknowledgements, or checksums to protect against problems such as packet loss, duplicate packets, packet sequence errors, and damaged or incomplete packets. Therefore IUCV Sockets are fast and lightweight.

For more information about the IUCV Socket type under Linux, see *Device Drivers, Features, and Commands*, SC33-8281. The most recent version can be found at the following website:

http://www.ibm.com/developerworks/linux/linux390/index.html

7.3.4 Linux Fast Path (Linux, z/VSE)

The Fast Path to Linux on System z function enables selected TCP/IP applications running under z/VSE to communicate with a TCP/IP stack on Linux on System z without using a TCP/IP stack on z/VSE. The short name is Linux Fast Path (LFP). LFP routes TCP/IP traffic to Linux on System z without the need for a z/VSE local TCP/IP stack. Instead, the Linux TCP/IP stack is used to communicate to the Internet.

Rather than using TCP/IP-based network communication, LFP uses IUCV-based communication. LFP can either use VM-defined IUCV devices for communication between guests in the same z/VM LPAR, or IUCV Sockets over HiperSockets for communication between servers running in LPARs, guests in different z/VM LPARs, or any combination.

LFP over IUCV Sockets over HiperSockets has been supported since z/VSE 5.1 and SLES 11 SP2 (or RHEL 6 Update 3). IUCV over HiperSockets requires the HiperSockets completion queue function, which has been available since z196.

IUCV over HiperSockets from z/VM guests requires z/VM 6.3 or later for completion queue support.

8

Connect HiperSockets to other networks

This chapter provides information about the different tools that are available to you to connect HiperSockets to an external network. Depending on the operating systems (OSs) that you use, the kind of IBM System z HiperSockets network that you have, and the network topology that you want to achieve, there are several options available.

This chapter provides information about, and examples for, the following functions:

- HiperSockets Accelerator or queued direct input/output (QDIO) Accelerator for z/OS
- HiperSockets Network Concentrator for Linux on System z
- HiperSockets Bridge Port for z/VM Virtual Switch

8.1 Connecting HiperSockets to external networks

Most servers require connectivity to other servers inside the central processor complex (CPC), and to external networks. There are several ways to achieve this:

- Multiple interfaces per server

  You can simply define one or more Open Systems Adapter (OSA) interfaces for external traffic, and one or more HiperSockets interfaces for access to the internal HiperSockets local area networks (LANs), per server.

- Software router

  One server can take the role of an Internet Protocol (IP) gateway, and connect a HiperSockets LAN to an OSA LAN. This is standard IP technology to connect different LANs. As for other LANs, a gateway server can be used to implement firewall technology.

  A software router connects different IP networks. The endpoints are aware that they are in different subnets. Routing information about the gateway must be set up in the endpoints.

- IBM z/OS HiperSockets Accelerator or z/OS QDIO Accelerator

  In z/OS, in addition to setting up an IP gateway, a HiperSockets Accelerator or a QDIO Accelerator can be used to connect a layer 3 IPv4 HiperSockets IP subnet to another IP subnet on OSA. These functions work like software routers, but are more efficient.

- HiperSockets Network Concentrator on Linux

  The HiperSockets Network Concentrator on Linux is a tool that connects a HiperSockets layer 3 network with an OSA network to form one single IP subnet. Therefore, it does not operate as an IP gateway, but rather as a software switch or bridge. The endpoints are all in the same IP subnet, and have no awareness whether another endpoint is reached through OSA or through HiperSockets.

- HiperSockets Bridge Port for z/VM Virtual Switch

  The HiperSockets Bridge Port for z/VM Virtual Switch provides a true layer 2 bridge that connects a layer 2 HiperSockets network with a layer 2 OSA network to create a single Ethernet broadcast domain. The OSs can define the IP topology just like they do for a LAN segment with a physical bridge. Therefore, it is natural to define one single IP subnet for this LAN segment.

8.2 HiperSockets Accelerator on z/OS

The z/OS Communications Server uses the technological advances and high-performing nature of the input/output I/O processing offered by HiperSockets with the IBM System z servers and OSA-Express, using the QDIO architecture. This is achieved by optimizing IP packet forwarding processing that occurs across these two types of technologies. This function is referred to as HiperSockets Accelerator. It is a configurable option, and is activated by defining the `IQDIORouting` option on the `IPCONFIG` statement.

HiperSockets Accelerator is supported by z/OS for IPv4 only. It enables a z/OS Transmission Control Protocol (TCP)/IP router stack to efficiently route IP packets from an OSA-Express (QDIO) interface with a HiperSockets (internal QDIO, or iQDIO) interface and vice versa. The routing is done by the z/OS Communications Server device drivers at the lowest possible software data link control level. IP packets do not have to be processed at the higher-level TCP/IP stack routing function, therefore reducing the path-length and improving performance.

Figure 8-1 represents an example of a HiperSockets Accelerator routing stack with four OSA-Express interfaces in a single System z that has multiple logical partitions (LPARs). These LPARs might be running z/OS, z/VM, or Linux on System z, and also z/VM with numerous guest systems.

You can have more than one z/OS TCP/IP stack to provide a redundant path in case one of the z/OS HiperSockets Accelerator images suffers an outage. The remaining routing stack can then connect to all of the remaining TCP/IP stacks in other images in System z that require connectivity to the OSA LANs using HiperSockets.

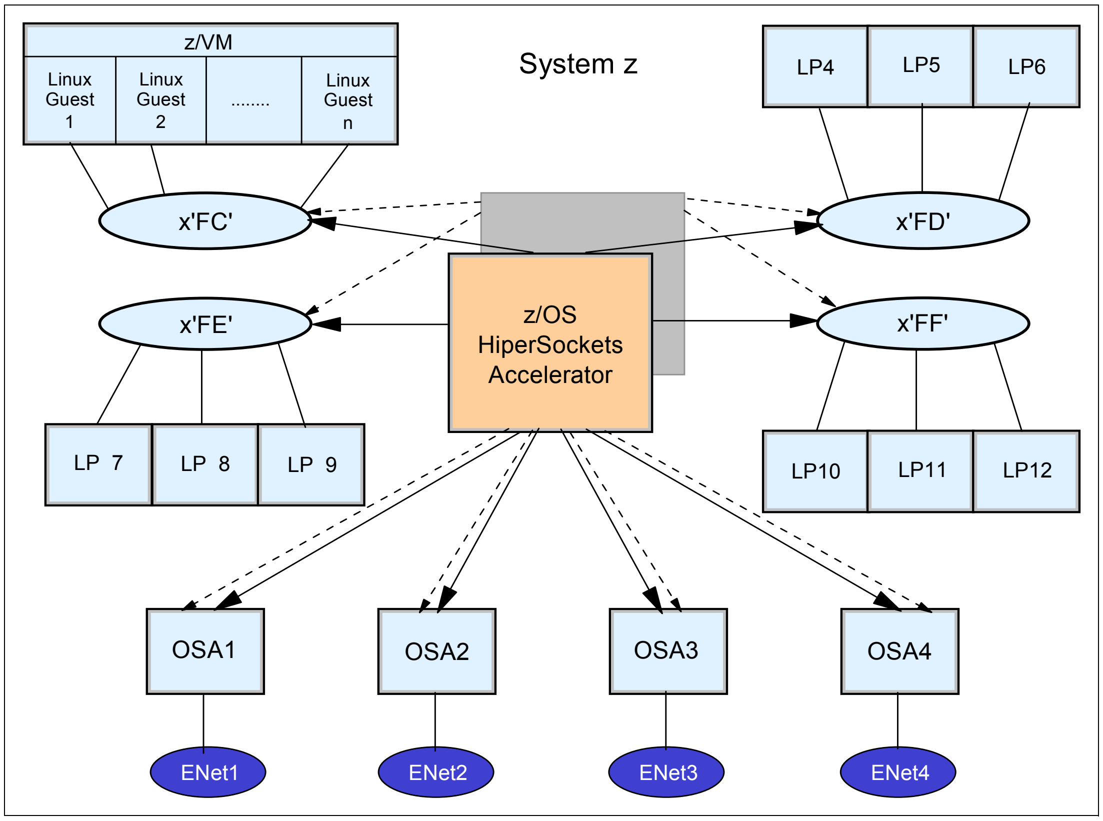


Figure 8-1 HiperSockets Accelerator on z/OS: Routing stack implementation

Figure 8-2 illustrates how HiperSockets Accelerator works. The solid line connecting TCP/IP A and TCP/IP X represents the normal path through the TCP/IP H stack's routing function, and the dotted line represents the accelerated path through the Virtual Telecommunications Access Method (VTAM) device driver.

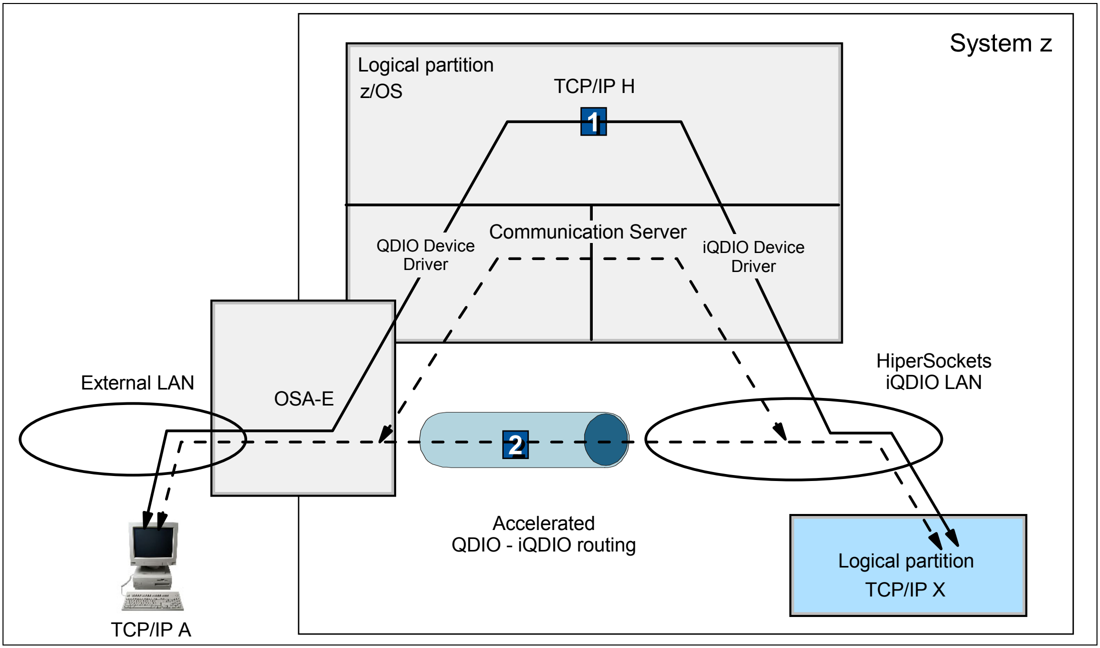


Figure 8-2 HiperSockets Accelerator flow

You can activate the HiperSockets Accelerator by configuring the iQDIO Routing option in the TCP/IP profile using the `IPCONFIG` statement. The TCP/IP stack automatically detects an IP packet prerouting across a HiperSockets Accelerator eligible route. Eligible routes are from OSA-Express (QDIO) to HiperSockets (iQDIO), and from HiperSockets (iQDIO) to OSA-Express (QDIO).

Figure 8-2 shows what happens when TCP/IP A sends something to TCP/IP X. The process is explained in the following steps:

1. The first packet goes through the TCP/IP routing stack in TCP/IP H, which creates iQDIO routing route entries for source TCP/IP A and destination TCP/IP X, the gateway for an external network. These entries are added to the iQDIO routing table. The destination stack TCP/IP X must be reachable through HiperSockets.
2. Starting with the second packet, all subsequent packets for the same destination take the optimized device driver path, and do not traverse the routing function of the TCP/IP routing stack. No change is required for target stacks. There is a timer built into the HiperSockets Accelerator function. Based on this timer, if a specific IQDIORouting entry is not used for 90 seconds, it is deleted from this table.

Therefore, for just the first time that a TCP/IP host sends the first packet of new entries, it is created in the IQDIORouting table, and it is involved in the TCP/IP H routing stack. The IP packets that follow this first packet are routed through the VTAM device driver.

Important: In order for a TCP/IP router stack to forward IP packets from an OSA-Express device to a HiperSockets device, the OSA-Express port must be defined as `PRIRouter` on the `DEVICE` statement in the TCP/IP profile. If no `PRIRouter` option is defined to the OSA-Express port, IP packets are not forwarded.

Another way to enable the OSA-Express device to forward packets is to use the VMAC parameter.

Only *one* `PRIRouter` can be defined to an OSA-Express port. However, a second TCP/IP stack can be defined as `SECRouter` to the same OSA-Express port, and serve as a backup to the `PRIRouter` TCP/IP stack.

Restriction: HiperSockets Accelerator cannot be enabled if either `IPSECURITY` or `NODATAGRAMFWD` is specified in the `IPCONFIG` statement.

If any IP packets have to be fragmented so that they can be routed between QDIO and iQDIO (or vice versa), they are not accelerated, and the normal path through the TCP/IP stack routing function is taken. You can prevent IP fragmentation conflicts by using path maximum transmission unit (MTU) discovery (`PATHMTUDISCOVERY` in `IPCONFIG`), or by coding the appropriate MTU size in the static route statement (if static routes are used).

For more details about defining MTU discovery and MTU sizes, see *z/OS Communications Server, IP Configuration Reference*, SC31-8776.

8.2.1 The QDIO Accelerator function

Since z/OS V1R11, the QDIO accelerator function is available. The QDIO Accelerator function extends the HiperSockets Accelerator function. HiperSockets Accelerator provides accelerated forwarding at the data link control (DLC) layer for the following types of packets:

- Inbound packets over HiperSockets that are forwarded outbound over OSA-Express QDIO
- Inbound packets over OSA-Express QDIO that are forwarded outbound over HiperSockets

QDIO Accelerator provides all of the functionality supported by HiperSockets Accelerator. It also provides accelerated forwarding at the DLC layer for the following types of packets:

- Inbound packets over OSA-Express QDIO that are forwarded outbound over OSA-Express QDIO
- Inbound packets over HiperSockets that are forwarded outbound over HiperSockets
- Sysplex distributor packets that are forwarded to a target stack, or that are forwarded to or from an IBM DataPower® appliance, when the route involves any of the following inbound and outbound DLC combinations:
  - Inbound over HiperSockets, forwarded outbound over OSA-Express QDIO
  - Inbound over OSA-Express QDIO, forwarded outbound over HiperSockets
  - Inbound over OSA-Express QDIO, forwarded outbound over OSA-Express QDIO
  - Inbound over HiperSockets, forwarded outbound over HiperSockets

To configure QDIO Accelerator, specify the `QDIOACCELERATOR` parameter on the `IPCONFIG` statement rather than specifying `IQDIOROUTING` for the HiperSockets Accelerator.

Just like the HiperSockets Accelerator, QDIO Accelerator is supported for IPv4 only, and can only be used with HiperSockets layer 3 interfaces.

For more information go to the z/OS V1R11 Information Center:

`http://pic.dhe.ibm.com/infocenter/zos/v1r11/index.jsp`

More information about the QDIO Accelerator, and additional details about how to set it up, are available by expanding the following topics in the information center: **Communications Server** → **IP Configuration Guide** → **Base TCP/IP system** → **IP configuration overview** → **Considerations for networking hardware attachment** → **QDIO Accelerator**.

8.2.2 HiperSockets Accelerator implementation

The test environment is shown in Figure 8-3 on page 101:

- LNXRH1, LNXSU1, and SC32 are only connected to IP subnet 192.0.1.0/24 on HiperSockets CHPID F0.
- SC31 represents the outside world. It has no direct connection to HiperSockets. It is connected to IP subnet 192.168.6.0/24 by OSA CHPID 07.

  In a real-life production scenario, the outside world might be distributed workstations, logical partitions (LPARs), or virtual machine (VM) guests running on another System z CPC. These servers can run any OS.

  Connecting SC31 using HiperSockets Accelerator does not really make sense, because it is on the same CPC and might easily be connected to the HiperSockets network directly. They are connected in this example for demonstration purposes only. It also works if SC31 is on another CPC, and if more than one server is connected to IP subnet 192.168.6.0/24 on a physical Ethernet.
- SC30 serves as a gateway between the HiperSockets network and the Ethernet connected with the OSA card. For this purpose the HiperSockets Accelerator was implemented in SC30.

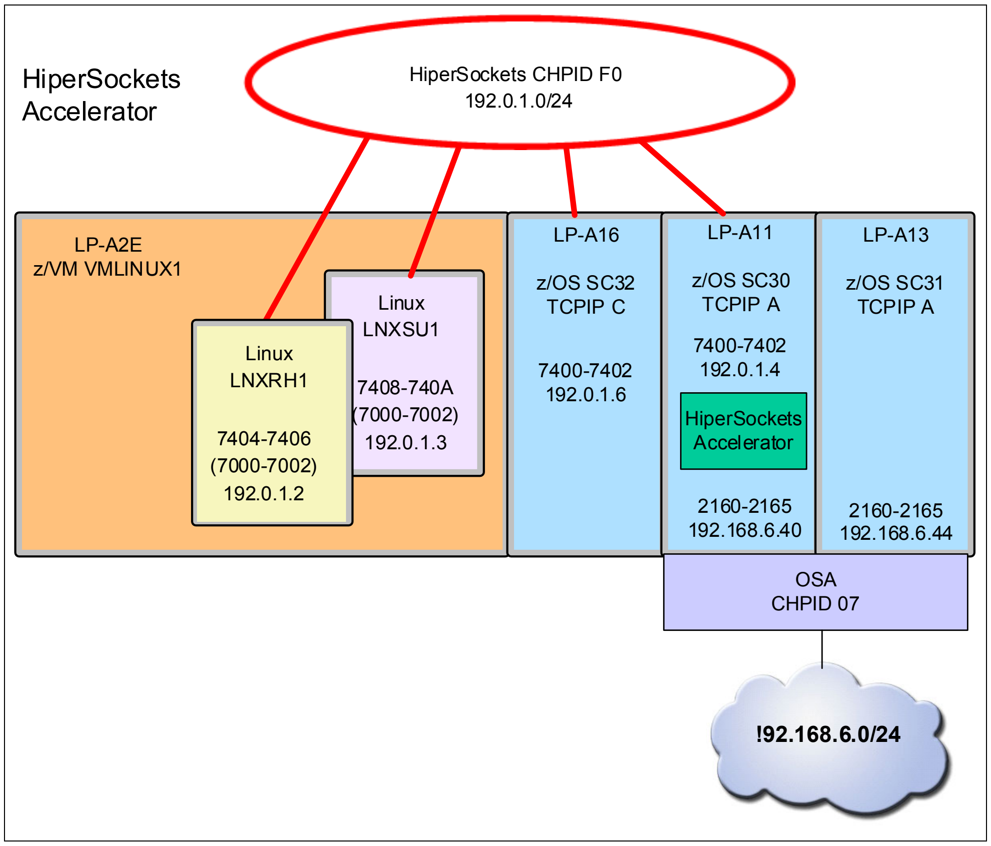


Figure 8-3 HiperSockets Accelerator implementation environment

Implement the HiperSockets Accelerator function using a single TCP/IP stack, TCPIPA on LPAR A11, to forward IP traffic from the HiperSockets LAN on CHPID F0 to an OSA-Express port defined on CHPID 07. It was configured in this example according to the following guidelines:

- The OSA-Express port must be defined as the `PRIRouter` on the `DEVICE` statement in the TCP/IP profile.
- Only one primary router (`PRIRouter`) can be defined to an OSA-Express port.
- HiperSockets Accelerator is activated by configuring the `IQDIORouting` option in the TCP/IP profile using the `IPCONFIG` statement.

8.2.3 HiperSockets Accelerator implementation steps

Take the following steps to configure your HiperSockets Accelerator test:

1. Define the HiperSockets and OSA-Express channel, control unit, and devices.
2. Configure the TCP/IP stacks to use the HiperSockets channel.
3. Configure a TCP/IP stack to use the OSA-Express device as a primary router.
4. Configure the TCP/IP using the OSA-Express device to enable forwarding of IP packets from its HiperSockets interface with the OSA-Express interface.

The OSA-Express IOCP statements are shown in Example 8-1.

Example 8-1 OSA-Express IOCP statements

```
CHPID PATH=(CSS(1,2),07),SHARED,                               *
      PARTITION=((CSS(1),(A11,A13,A16,A18),(=)),(CSS(2),(A2E),*
      (=))),PCHID=570,TYPE=OSD
CNTLUNIT CUNUMBR=2160,PATH=((CSS(1),07),(CSS(2),07)),UNIT=OSA
IODEVICE ADDRESS=(2160,015),UNITADD=00,CUNUMBR=(2160),UNIT=OSA
IODEVICE ADDRESS=(216F,001),UNITADD=FE,CUNUMBR=(2160),         *
      UNIT=OSAD
```

8.2.4 VTAM configuration

A VTAM Transport Resource List (TRL) major node must be defined for the OSA-Express device. Define the VTAM TRL as member OSA2160 in your VTAMLST data set, as shown in Example 8-2.

Example 8-2 VTAM TRLE definition for OSA-EXPRESS

```
OSA2160  VBUILD TYPE=TRL
OSA2160P TRLE  LNCTL=MPC,
               READ=2160,
               WRITE=2161,
               DATAPATH=(2162-2165),
               PORTNAME=OSA2160,
               MPCLEVEL=QDIO
```

Activate the OSA TRL by issuing this command:

```
V NET,ACT,ID=OSA2160
```

8.2.5 TCP/IP configuration

In this implementation, configure stack TCPIPA on LPAR A11 to use the HiperSockets connection on CHPID F0 by using the `INTERFACE` statements for the HiperSockets connection (see Example 8-3 on page 103). Use the `DEVICE`, `LINK`, and `HOME` statements for the OSA-Express connection.

HiperSockets Accelerator is activated by configuring the `IQDIORouting` option in the TCP/IP profile's `IPCONFIG` statement **1**. Also specify the `DATAGRAMFWD` option on the `IPCONFIG` statement to enable pack forwarding **2**. In addition, configure the OSA-Express device with the `PRIRouter` option **3**.

Example 8-3 TCPIPA profile on SC30

```
GLOBALCONFIG NOTCPIPSTATISTICS
IPCONFIG DATAGRAMFWD 2 PATHMTUDISCOVERY IQDIOROUTING    1
TCPCONFIG TCPSENDBFRSIZE 256K TCPRCVBUFRSIZE 256K SENDGARBAGE FALSE
TCPCONFIG RESTRICTLOWPORTS
UDPCONFIG RESTRICTLOWPORTS
;-------------------------------------------------------------------
;  HIPERSOCKETS
;-------------------------------------------------------------------
;
; Define HiperSockets interface on CHPID F0
  Interface HIPERLF0
  Define IPAQIDIO
  IPADDR 192.0.1.4/24
  CHPID F0

; PRIMARY ROUTING SU CHPID 07
DEVICE OSA2160 MPCIPA PRIR   3
LINK OSA2160L IPAQENET OSA2160
;
;-------------------------------------------------------------------
;  HOME DEFINITION
;-------------------------------------------------------------------
;
HOME
 192.168.6.40      OSA2160L

BEGINROUTES
ROUTE DEFAULT              192.0.1.4        HIPERLF0    MTU 1492
ROUTE 192.0.1.0       255.255.255.0 =       HIPERLF0    MTU 1492
ROUTE 192.168.6.0     255.255.255.0 =       OSA2160L    MTU 8992
ENDROUTES

START HIPERLF0
START OSA2160
```

Static routing was used in this example. However, dynamic routing with Routing Information Protocol (RIP) or Open Shortest Path First (OSPF) is supported when the HiperSockets Accelerator function is enabled.

Important: Because `NODATAGRAMFWD` is the default in `IPCONFIG`, you must explicitly code `DATAGRAMFWD` when using the HiperSockets Accelerator.

8.2.6 HiperSockets Accelerator verification

The following sections describe how to verify that the HiperSockets Accelerator is installed and working.

OSA-Express verification

Use the **D,NET,TRLE=*trle_name*** command to verify that the OSA-Express transport resource list element (TRLE) has been activated, as shown in Example 8-4.

Example 8-4 OSA TRLE display

```
D NET,TRL,TRLE=OSA2160P
IST097I DISPLAY ACCEPTED
IST075I NAME = OSA2160P, TYPE = TRLE 855
IST1954I TRL MAJOR NODE = OSA2160N
IST486I STATUS= ACTIV, DESIRED STATE= ACTIV
IST087I TYPE = LEASED            , CONTROL = MPC , HPDT = YES
IST1715I MPCLEVEL = QDIO       MPCUSAGE = SHARE
IST2263I PORTNAME = OSA2160    PORTNUM =   0   OSA CODE LEVEL = 0C8C
IST2337I CHPID TYPE = OSD      CHPID = 07  PNETID = **NA**
```

TCP/IP startup

When TCPIPA is started on LPAR A11, SYSLOG messages enable us to verify that IP forwarding is enabled 1, the HiperSockets Accelerator function is enabled 2, the HiperSockets device is initialized 3, and the OSA device is initialized 4, as shown in Example 8-5.

Example 8-5 HiperSockets Accelerator stack SYSLOG messages

```
EZZ0641I IP FORWARDING NOFWDMULTIPATH SUPPORT IS ENABLED 1
EZZ0623I PATH MTU DISCOVERY SUPPORT IS ENABLED
EZZ0688I IQDIO ROUTING IS ENABLED 2
EZZ4340I INITIALIZATION COMPLETE FOR INTERFACE HIPERLF0 3
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA2160 4
```

If this function cannot be enabled, you will receive the following message:

```
EZZ0689I CANNOT ENABLE IQDIO ROUTING
```

This might be due to one of the conditions described in the following messages:

- `IP FORWARDING IS DISABLED`
- `IP SECURITY IS ACTIVE`
- `QDIO ACCELERATOR IS ACTIVE`
- `PROCESSOR IS NOT HIPERSOCKET CAPABLE`
- `TCPIP ACTIVATED WITH NOIQDIOROUTING`

Example 8-6 shows the **NETSTAT** command with the configuration option, which you use to verify that this function is enabled. The IP configuration table portion of the **D TCPIP,procname,N,CONFIG** command also verifies that IP forwarding and routing are enabled, as shown in Example 8-6.

Example 8-6 TCPIPA configuration display

```
D TCPIP,TCPIPA,N,CONFIG
...
IP CONFIGURATION TABLE:
FORWARDING: YES   TIMETOLIVE: 00064  RSMTIMEOUT:  00060
IPSECURITY: NO
ARPTIMEOUT: 01200  MAXRSMSIZE: 65535  FORMAT:      LONG
IGREDIRECT: NO     SYSPLXROUT: NO     DOUBLENOP:   NO
STOPCLAWER: NO     SOURCEVIPA: NO
MULTIPATH:  NO     PATHMTUDSC: YES    DEVRTRYDUR:  0000000090
```

```
DYNAMICXCF: NO
QDIOACCEL:  NO
IQDIOROUTE: YES       QDIOPRIORITY: 1
TCPSTACKSRCVIPA: NO
CHECKSUMOFFLOAD: YES     SEGOFFLOAD: NO
...
```

The last command used to verify the iQDIO routing function is a display of the VTAM TRLE major node (`IUTIQ4F0`), shown in Example 8-7. A display of the QDIO TRLE also results in message ITS2309I.

Example 8-7 Verify HiperSockets Accelerator is enabled by displaying the TRLE

```
D NET,TRL,TRLE=IUTIQ4F0
IST097I DISPLAY ACCEPTED
IST075I NAME = IUTIQ4F0, TYPE = TRLE 913
IST1954I TRL MAJOR NODE = ISTTRL
IST486I STATUS= ACTIV, DESIRED STATE= ACTIV
IST087I TYPE = LEASED            , CONTROL = MPC , HPDT = YES
IST1715I MPCLEVEL = QDIO       MPCUSAGE = SHARE
IST2263I PORTNAME =            PORTNUM =   0   OSA CODE LEVEL = *NA*
IST2337I CHPID TYPE = IQD      CHPID = F0  PNETID = **NA**
IST2319I IQD NETWORK ID = 0716
IST1577I HEADER SIZE = 4096 DATA SIZE = 16384 STORAGE = ***NA***
IST1221I WRITE DEV = 7401 STATUS = ACTIVE     STATE = ONLINE
IST1577I HEADER SIZE = 4092 DATA SIZE = 0 STORAGE = ***NA***
IST1221I READ  DEV = 7400 STATUS = ACTIVE     STATE = ONLINE
IST924I -------------------------------------------------------------
IST1221I DATA  DEV = 7402 STATUS = ACTIVE     STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1717I ULPID = TCPIPA ULP INTERFACE = HIPERLF0
IST2309I ACCELERATED ROUTING ENABLED
```

Connectivity test

To have connectivity across the two subnets, the correct routing information needs to be set up in all of the endpoints. To set up routing information, follow these steps:

1. Add the following information in LNXRH1 and LNXSU1:

   ```
   route add -net 192.168.6.0 netmask 255.255.255.0 gw 192.0.1.4
   ```

   Alternatively, you can add the HiperSockets Accelerator as a default gateway:

   ```
   route add default gw 192.0.1.4
   ```

2. Add the following line (in bold) to the profile of TCPIP C in SC32:

   ```
   BEGINRoutes
   ; Direct Routes - Routes that are directly connected to my interfaces
   ; Destination Subnet Mask   First Hop Link Name     Packet Size
   ROUTE 192.0.1.0    255.255.255.0 = HIPERLF0      mtu defaultsize repl
   ROUTE 192.168.6.0  255.255.255.0 192.0.1.4 HIPERLF0 mtu defaultsize repl
   ENDRoutes
   ```

   Alternatively you could have added the following code:

   ```
   ROUTE DEFAULT 192.0.1.4 HIPERLF0 mtu defaultsize repl
   ```

3. Add the line in bold to the profile of TCPIP A in SC31:

```
ROUTE 192.168.6.0     255.255.255.0 =            OSA2160L    MTU 8992
ROUTE 192.0.1.0   255.255.255.0 192.168.6.40 OSA2160L    MTU 8992
```

4. Now you are able to ping back and forth between SC31 and the servers on the HiperSockets network. Example 8-8 shows a ping from LNXRH1 (192.0.1.2) to SC31 (192.168.6.44).

Example 8-8 Linux on HiperSockets to z/OS on OSA

```
[root@lnxrh1 ~]# ping 192.168.6.44
PING 192.168.6.44 (192.168.6.44) 56(84) bytes of data.
64 bytes from 192.168.6.44: icmp_seq=1 ttl=64 time=1.77 ms
64 bytes from 192.168.6.44: icmp_seq=2 ttl=64 time=1.78 ms
^C
--- 192.168.6.44 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1459ms
rtt min/avg/max/mdev = 1.778/1.780/1.782/0.002 ms
```

5. Example 8-9 shows a ping from SC31 (192.168.6.44) to SC32 (192.0.1.6).

Example 8-9 z/OS on OSA to z/OS on HiperSockets

```
CS V2R1: Pinging host 192.0.1.6
 Ping #1 response took 0.002 seconds.
 ***
```

All other combinations were also verified.

8.2.7 References

The following references provide additional information about HiperSockets Accelerator:

- *Communications Server for z/OS V2R1 TCP/IP Implementation Volume 1: Base Functions, Connectivity, and Routing*, SG24-8096
- *IBM System z Connectivity Handbook*, SG24-5444
- *z/OS Communications Server, IP Configuration Guide,* SC31-8775
- *z/OS Communications Server, IP Configuration Reference,* SC31-8776
- *z/OS Communications Server, SNA Resource Definition Reference*, SC31-8778

8.3 HiperSockets Network Concentrator on Linux

Traffic between HiperSockets and OSA-Express can be transparently bridged using the HiperSockets Network Concentrator, without requiring intervening network routing overhead, therefore increasing performance and simplifying the network configuration. This is achieved by configuring a connector Linux system that has HiperSockets and OSA-Express connections defined.

The HiperSockets Network Concentrator registers with HiperSockets firmware as a special network entity to receive data packets destined for an IP address on the external LAN using an OSA-Express port. The HiperSockets Network Concentrator also registers IP addresses to the OSA-Express on behalf of the TCP/IP stacks using HiperSockets, therefore providing inbound and outbound connectivity.

Restriction: The Linux implementation of the HiperSockets Network Concentrator supports only layer 3 devices, only IPv4 addresses, and does not support VLAN traffic.

The HiperSockets Network Concentrator, shown in Figure 8-5 on page 109, is a mechanism to connect systems with HiperSockets interfaces to the external network using the *same subnet*. Therefore, the server instances connected to HiperSockets appear as though they were directly connected to the physical network.

A Linux Network Concentrator system acts as a forwarder for traffic between the OSA interface and the internal HiperSockets connected systems (z/VM, z/OS, z/VSE, and Linux on z). See *Linux on System z, Device Drivers, Features, and Commands*, SC33-8281, for detailed information.

Important: IP fragmentation does not work for multicast bridging. The MTU of the HiperSockets link and OSA must be of the same size. Multicast packets not fitting in the link MTU are discarded.

HiperSockets Network Concentrator can be a useful solution if you have a Linux on System z (native LPAR or guest under z/VM), a large amount of traffic among servers inside System z, and the requirement of high-speed communications to the external network. It enables you to bridge network endpoints rather than routing them, and it does not use other subnets.

In addition, HiperSockets Network Concentrator enables you to port systems from the LAN into a System z environment without changing IP address and network routing. Therefore, HiperSockets Network Concentrator helps to simplify network configuration and administration. It can be helpful during server consolidation by enabling you to move one server after the other from an external workstation to Linux on System z without changing the network configuration.

Figure 8-4 shows such a scenario where Linux servers are ported from external workstations to Linux on System z. Linux A, B, and C are already ported. Linux D and E still reside on external workstations. At any point in time, Linux A-E can all communicate with each other, and with the z/OS LPARs over subnet 192.0.1.0/24.

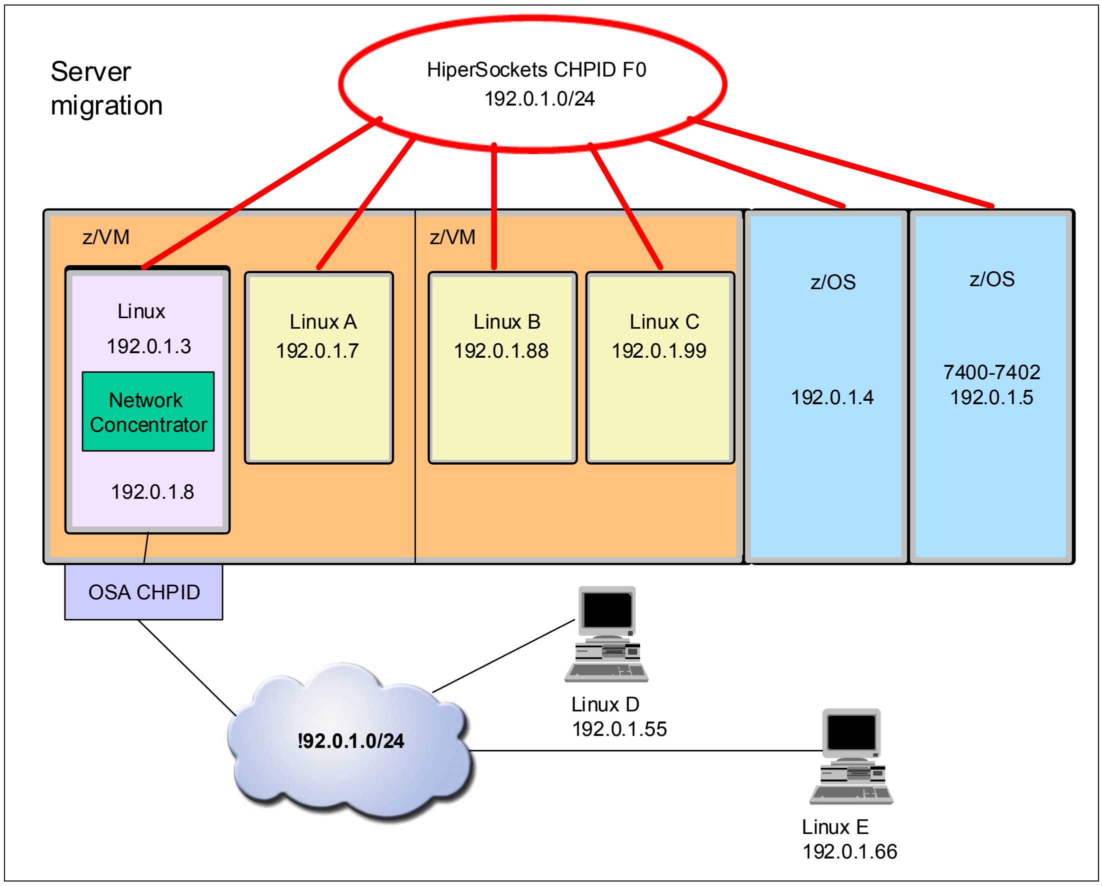


Figure 8-4 Server migration with the HiperSockets Network Concentrator

8.3.1 Example

Figure 8-5 on page 109 represents the setup used for this example. Because there were no external workstations available, LNXRH1 plays the role of an external server, and is only connected to the OSA Card. LNXRH1 is not directly connected to the HiperSockets network. LNXSU1 runs the HiperSockets Network Concentrator that connects the HiperSockets network and the OSA network. IBM z/OS SC30-32 are only connected to the HiperSockets network, and have no direct connection to the OSA network.

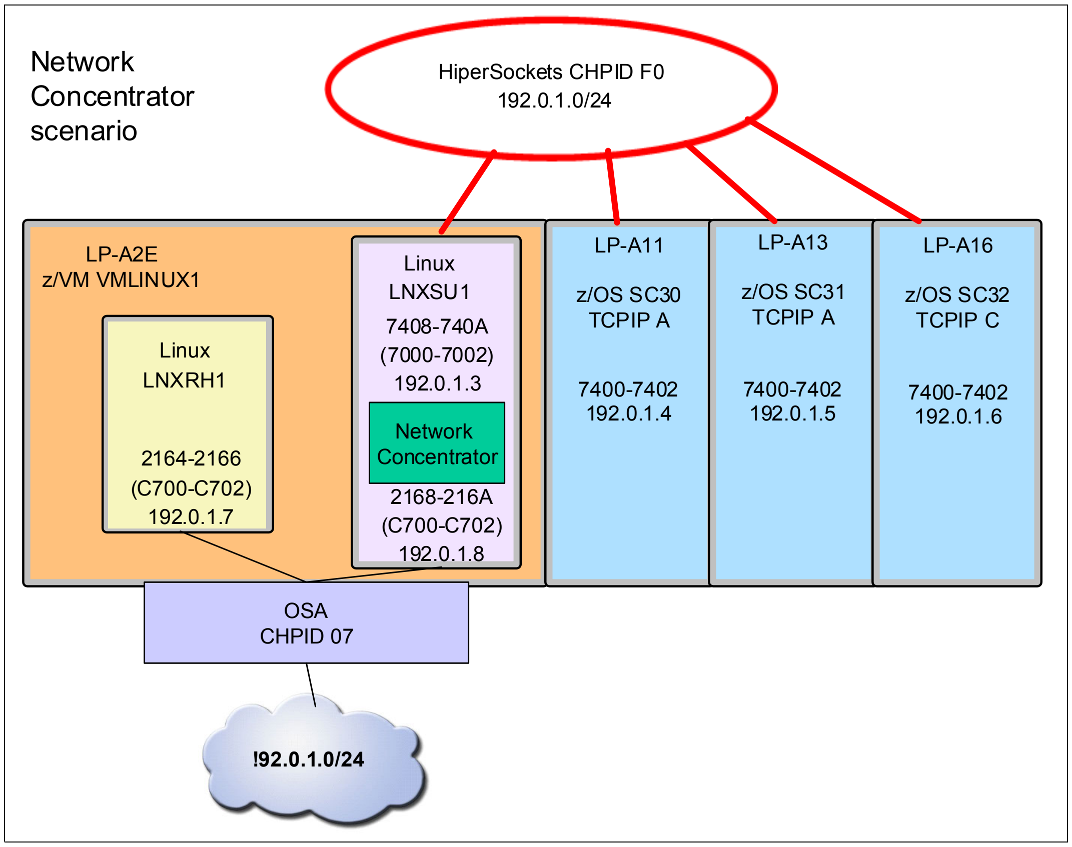


Figure 8-5 HiperSockets Network Concentrator on Linux

Although the HiperSockets Network Concentrator can be a stand-alone Linux LPAR, there is no functional difference between Linux in an LPAR and running as a z/VM guest Linux server. This configuration used a Linux guest server.

For the servers inside the HiperSockets Internal LAN, the routing parameters are the same as they were if they were connected on the external net directly. The *default gateway* setting is the same. See Table 8-1.

Table 8-1 Details of the configuration scenario

LP name	Environment	System name	CHPID	Device address	IP address
A2E	Linux under z/VM	LNXRH1	07	C700-C702	192.0.1.7
A2E	Linux under z/VM	LNXSU1	07	C700-C702	192.0.1.8
A2E	Linux under z/VM	LNXSU1	F0	7000-7002	192.0.1.3
A11	z/OS sysplex	SC30	F0	7400-7402	192.0.1.4
A13	z/OS sysplex	SC31	F0	7400-7402	192.0.1.5
A16	z/OS sysplex	SC32	F0	7400-7402	192.0.1.6

On the connector Linux server, LNXSU1, follow these steps:

1. Define an OSA-Express LAN. This is similar to the information given previously in 3.5, “HiperSockets in Linux on System z” on page 48 because the same qeth device driver is used.

Important: Both the OSA and the HiperSockets interface need to be defined as layer 3 interfaces.

Issue the following commands:

```
znetconf -a c700 -o layer2="0"
ifconfig eth4 192.0.1.8 netmask 255.255.255.0 mtu 8192 up
```

Tip: When issuing the **`ifconfig`** command for each network interface, include the `mtu` parameter with the same value.

This example uses mtu 8192, which is the HiperSockets default. Accordingly, the OSA card supports up to 8992 bytes for layer 3. An alternative is to use mtu 1492 on both interfaces, which is the default value for the OSA card.

Also, to preserve this setting across Linux boots, add an `mtu=8192` statement to the appropriate network definition file. See 3.5.5, "Permanent Linux definitions" on page 54.

2. Define a layer 3 HiperSockets LAN interface as described in 3.5, "HiperSockets in Linux on System z" on page 48:

   ```
   znetconf -a 7000 -o layer2="0"
   ifconfig hsi1 192.0.1.3 netmask 255.255.255.0 mtu 8192 up
   ```

3. Make the HiperSockets a primary concentrator; for example, issue the following command:

   ```
   echo primary_connector > /sys/bus/ccwgroup/drivers/qeth/0.0.7000/route4
   ```

4. When using multicasting, make the OSA-Express a multicast router. For example, issue the following command:

   ```
   echo multicast_router > /sys/bus/ccwgroup/drivers/qeth/0.0.c700/route4
   ```

5. Enable IP forwarding:

   ```
   sysctl -w net.ipv4.ip_forward=1
   ```

   Update `/etc/sysctl.conf` with this line to retain the setup across a restart.

6. Remove the network route for the HiperSockets interface, for example:

   ```
   route del -net 192.0.1.0 netmask 255.255.255.0 dev hsi1
   ```

7. Start the HiperSockets Network Concentrator:

   ```
   start_hsnc.sh &
   ```

   Update `/etc/init.d/boot.local` with **`start_hsnc.sh`** to retain the setup across a restart. Warnings and errors are written to the `var/log/messages` file. No messages are written if the start is successful unless multicasting is used. Then the following messages are written to `var/log/messages`:

   ```
   xcec-bridge: *** started ***
   xcec-bridge: rechecking interfaces
   xcec-bridge: added interface hsi1
   xcec-bridge: added interface eth4
   xcec-bridge: rechecking interfaces
   ```

Example 8-10 shows the results of the **ifconfig** command for the Network Concentrator. Note the identical mtu values. Otherwise, nothing else is unique in this output.

Example 8-10 Network Concentrator ifconfig

```
eth4      Link encap:Ethernet  HWaddr 6C:AE:8B:48:0A:B0
          inet addr:192.0.1.8  Bcast:192.0.1.255  Mask:255.255.255.0
          inet6 addr: fe80::6cae:8b00:348:ab0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:8192  Metric:1
          RX packets:395 errors:0 dropped:0 overruns:0 frame:0
          TX packets:25 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:40616 (39.6 Kb)  TX bytes:2330 (2.2 Kb)

hsi1      Link encap:Ethernet  HWaddr 06:00:F0:2E:00:0A
          inet addr:192.0.1.3  Bcast:192.0.1.255  Mask:255.255.255.0
          inet6 addr: fe80::400:f0ff:fe2e:a/64 Scope:Link
          UP BROADCAST RUNNING NOARP MULTICAST  MTU:8192  Metric:1
          RX packets:11 errors:0 dropped:0 overruns:0 frame:0
          TX packets:25 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:924 (924.0 b)  TX bytes:2377 (2.3 Kb)
```

No changes and no special routing setup are required on the other servers.

8. Verify that the Red Hat Enterprise Linux (RHEL) system, LNXRH2, can ping the z/OS systems on the HiperSockets network now, as shown in Example 8-11.

Example 8-11 Network concentrator verification

```
[root@lnxrh1 ~]# ping 192.0.1.4
PING 192.0.1.4 (192.0.1.4) 56(84) bytes of data.
64 bytes from 192.0.1.4: icmp_seq=1 ttl=63 time=9.60 ms
64 bytes from 192.0.1.4: icmp_seq=2 ttl=63 time=0.253 ms
64 bytes from 192.0.1.4: icmp_seq=3 ttl=63 time=0.287 ms
^C
--- 192.0.1.4 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5079ms
rtt min/avg/max/mdev = 0.216/1.803/9.604/3.488 ms
[root@lnxrh1 ~]# ping 192.0.1.6
PING 192.0.1.6 (192.0.1.6) 56(84) bytes of data.
64 bytes from 192.0.1.6: icmp_seq=1 ttl=63 time=9.66 ms
64 bytes from 192.0.1.6: icmp_seq=2 ttl=63 time=0.238 ms
64 bytes from 192.0.1.6: icmp_seq=3 ttl=63 time=0.227 ms
^C
--- 192.0.1.6 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4089ms
rtt min/avg/max/mdev = 0.227/2.126/9.665/3.769 ms
```

8.3.2 References

For more information, see the Linux on System z document *Device Drivers, Features, and Commands*, SC33-8411. You can find the current version of this and other publications in the Linux on System z library on the developerWorks website:

http://www.ibm.com/developerworks/linux/linux390/documentation_dev.html

8.4 The z/VM Virtual Switch with HiperSockets bridge port

The z/VM Virtual Switch has been enhanced to transparently bridge guest VM network connections on a HiperSockets LAN segment. This bridge enables HiperSockets guest VMs with a HiperSockets connection to also communicate with the following components:

- Other servers on the HiperSockets network
- Other guest VMs on the z/VM Virtual Switch
- External network hosts through the virtual switch OSA Uplink port

8.4.1 The z/VM Virtual Switch

IBM z/VM has the capability to simulate a network segment and a virtual switch. The interfaces to this simulated network are represented to the z/VM guest systems as virtual OSA interfaces. The z/VM Virtual Switch can be connected to one or more real OSA trunk ports, and provides external LAN connectivity to the guest systems that are connected to the virtual switch.

Note that the virtual switch acts as a switch, not a router, so the OSA interfaces and the virtual interfaces are part of the same IP subnet. A z/VM Virtual Switch exists only inside a z/VM host, and only guests of this specific host can be directly connected to this virtual switch.

8.4.2 Bridging a HiperSockets LAN with a z/VM Virtual Switch

Ever since z/VM Version 6 Release 2, a HiperSockets Bridge Port can be defined on a z/VM Virtual Switch. A HiperSockets Bridge Port transparently bridges z/VM guests connected to the HiperSockets network with the z/VM Virtual Switch and the connected OSA network, as shown in Figure 8-6.

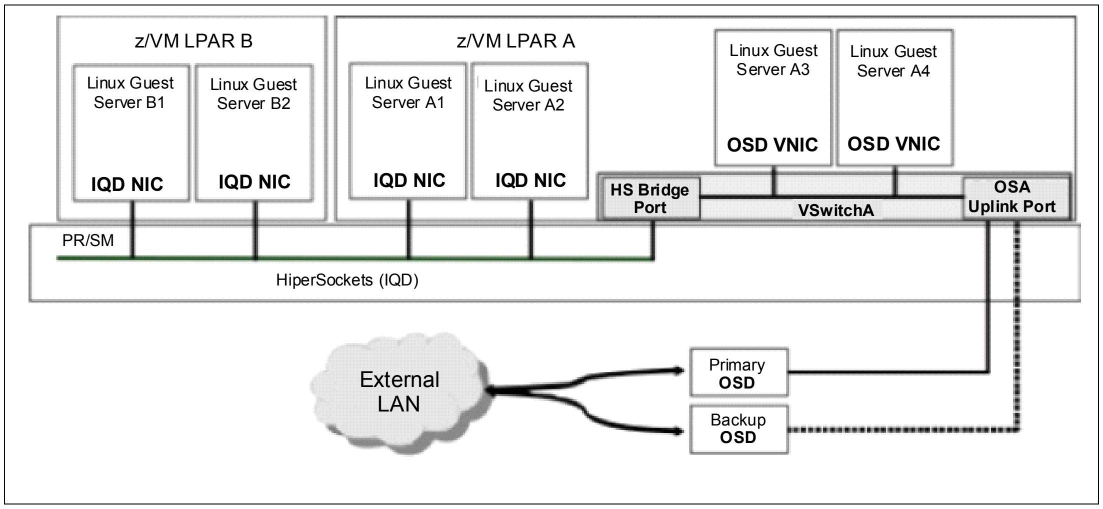


Figure 8-6 Bridged HiperSockets channel

The HiperSockets channel, the virtual switch, and the OSA network are all part of the same layer 2 broadcast domain and the same subnet. Only a single subnet is required, and the servers require only a single network interface and a single IP address. A single IP address and VM network connection can be used to communicate over the internal and external segments of the LAN.

The fact that a given destination address might be on the local HiperSockets channel or outside the CPC is totally indiscernible to the network interfaces of the servers. Incorporating the HiperSockets channel into the flat layer 2 broadcast domain through OSA adapters simplifies networking configuration and maintenance.

The virtual switch HiperSockets Bridge Port eliminates the need to configure a separate next hop router on the HiperSockets channel to provide connectivity to destinations outside of a HiperSockets channel. This avoids the need to create routes for this internal route in all hosted servers, and the extra hop of a router to provide the layer 3 routing functions.

8.4.3 Benefits of a bridged HiperSockets network

Connecting virtual servers directly to a bridged HiperSockets network (such as servers B1, B2, A1, and A2 in Figure 8-6 on page 112) provides maximum performance for communication with other servers inside the same CPC (same LPAR or other LPARs) and also provides connectivity to the external LAN. Administration is simple because the whole network is a single layer 2 broadcast domain, and each server only has a single network interface.

You can extend this network to connect to other virtual switches and other HiperSockets on other CPCs, and therefore create a single HiperSockets network across CPCs.

Compared to a HiperSockets network and an OSA network

When a bridged HiperSockets network is compared to a HiperSockets network and an OSA network connected to each server, you see the following benefits:

- There is no separate HiperSockets subnet required
- Only a single interface and IP address is needed per server

However, a directly connected OSA network can provide better performance if high throughput to the external network is required.

Compare a direct connection to the z/VM Virtual Switch

IBM z/VM guests that are directly connected to a HiperSockets network can use QDIO Enhanced Buffer-State Management (QEBSM). See 2.3.1, “Hardware assists” on page 25 for more information. Rather than having the z/VM host system simulate a network interface, HiperSockets firmware provides the network interface directly to the guest OS. This has the following benefits:

- Reduced amount of z/VM memory required
- Reduced processor use
- Reduced latency
- No VM host involvement for traffic inside the CPC
- Minimal z/VM resource usage to provide bridge to external network

The HiperSockets Bridge Port of a Virtual Switch in one z/VM LPAR can bridge the traffic from z/VM guests in other z/VM LPARs to other z/VM guests and to the external network. Consequently, this network is no longer restricted to a single z/VM host.

8.4.4 HiperSockets Bridge Port details

In a bridged HiperSockets network, one or more z/VM Virtual Switches provide connectivity to an external LAN for the z/VM guest systems connected to the HiperSockets network and create a single layer 2 broadcast domain. Therefore the following items apply:

- Only interfaces that use QEBSM are eligible to be bridged by HiperSockets firmware. Therefore only z/VM guest systems that use QEBSM will be bridged. Interfaces to native LPARs will not be bridged.
- All HiperSockets interfaces must be defined as layer 2 interfaces. To enforce this and to avoid misconfigurations, the HiperSockets channel must be defined as either one of the following options:
  - HiperSockets for z/VM External Bridge
  - HiperSockets for intraensemble data network (IEDN), or iQDIO extensions (IQDX)

  See 9.3, “HiperSockets for IEDN (IQDX)” on page 128 for more information about IEDN and IQDX. Also, 2.1.1, “Channel parameters for HiperSockets” on page 9 explains how these usage parameters are defined in the input/output configuration data set (IOCDS). You can only define layer 2 interfaces on these HiperSockets channels. You cannot define a bridge port on a normal HiperSockets channel.
- The HiperSockets Bridge Port for z/VM Virtual Switch is available with z/VM Version 6 Release 2 and later.
- The HiperSockets Bridge Port is available since IBM zEnterprise 196 (z196). Since then, HiperSockets firmware provides the following functionality required by a HiperSockets Bridge Port:
  - All frames with unknown destinations will be sent to the bridge port rather than being discarded.
  - HiperSockets firmware will notify the z/VM Virtual Switch of any changes to the addresses registered on the HiperSockets network.
  - The functionality provides support for multiple bridge ports per HiperSockets channel for redundancy and automatic failover.
- Multiple Virtual Switches with bridge ports to the same HiperSockets channel can be defined for redundancy. However, only one Virtual Switch will be actively bridging traffic at any time. IBM z/VM and HiperSockets firmware provide automatic failover mechanisms.
- Multiple OSA channels can be connected to a z/VM Virtual Switch for high availability. These OSA channels are configured as aggregated links in concert with a switch that also supports the IEEE 802.3ad specification. This configuration provides the benefits of increased bandwidth and near seamless failover of a failed link in the aggregated group.
- The z/VM HiperSockets Bridge Port uses the HiperSockets Completion Queue function, which is explained in 7.3, “Completion queue function” on page 92. Therefore, the bridge port will send traffic synchronously to the other participants on the HiperSockets channel, if possible.

  However, it can also send asynchronously in case a receiver does not provide enough empty inbound buffers. This prevents the bridge port from being held up by one slow receiver. It will continue to send traffic synchronously with the lowest possible latency to the other receivers.

8.4.5 Path MTU Discovery

Path MTU Discovery (PMTUD) is a standardized technique for determining an acceptable MTU size between two IP network connections. The goal of this technique is to discover the largest size datagram that does not require fragmentation anywhere along the path between the source and destination. This discovered datagram size is known as the *path maximum transmission unit* (PMTU) or the effective MTU for sending.

The virtual switch can be configured to provide MTU discovery responses for payloads that are determined by the virtual switch to be larger than the supported MTU of the OSA uplink port, or the system administrator-configured MTU of the external LAN. The PMTUD process will cause the virtual switch to respond to the sending guest with an Internet Control Message Protocol (ICMP) error response that contains the acceptable MTU.

The MTU value used to check payloads to be sent over the virtual switch uplink port is provided by the PATHMTUDISCOVERY operand of the SET VSWITCH command. By default, the virtual switch will use the largest allowable MTU supported by the OSA-Express feature currently deployed as the uplink port.

PMTUD is provided for guests connected directly to the virtual switch (simulated network interface cards, or NICs) and the HiperSockets bridge-capable ports. Connecting a virtual switch to a HiperSockets channel introduces the complexity of having a local broadcast domain with different MTU sizes, requiring infrastructure to orchestrate an acceptable MTU size between source and destinations that are connected through the bridged networks.

The virtual switch PMTUD provides the ability for the HiperSockets guest port-to-guest port communications to enjoy the performance benefits of using large MTUs of the HiperSockets channel, while also supporting external destinations for HiperSockets bridge-capable ports with lower MTUs. Bridge-capable guest ports, or simulated ports with a single network connection supporting PMTUD, can communicate optimally with all destinations over the entire broadcast domain.

For more information, see *z/VM V6R3 Connectivity*, SC24-6174.

8.4.6 References

For more information go to the z/VM V6R2.0 Information Center:

http://pic.dhe.ibm.com/infocenter/zvm/v6r2/index.jsp

More information about the z/VM HiperSockets Bridge Port, and additional details about how to set it up, can be found by expanding the following topics in the information center: **Planning and Administration → z/VM V6R2 Connectivity → Planning Virtual Networks → Bridging a HiperSockets LAN with a z/VM Virtual Switch**.

8.4.7 Example

Figure 8-7 represents the setup used for this example. Because no external workstations or a second CPC were available, z/OS LPAR SC30 plays the role of an external server, and is only connected to the OSA card. SC30 is not directly connected to the HiperSockets network. HiperSockets channel-path identifier (CHPID) F9 is defined as HiperSockets for z/VM External Bridge in the IOCDS.

LNXRH1 and LNXSU1 will be directly connected to the External Bridge Network F9 by layer 2 interfaces. They will only be connected to the HiperSockets network, and have no direct connection to the OSA network. Define a z/VM virtual switch (VSWALEX) to connect the OSA network to the HiperSockets network. SC30, LNXRH1, and LNXSU1 will all be connected to the same 192.168.6.0/25 IP network.

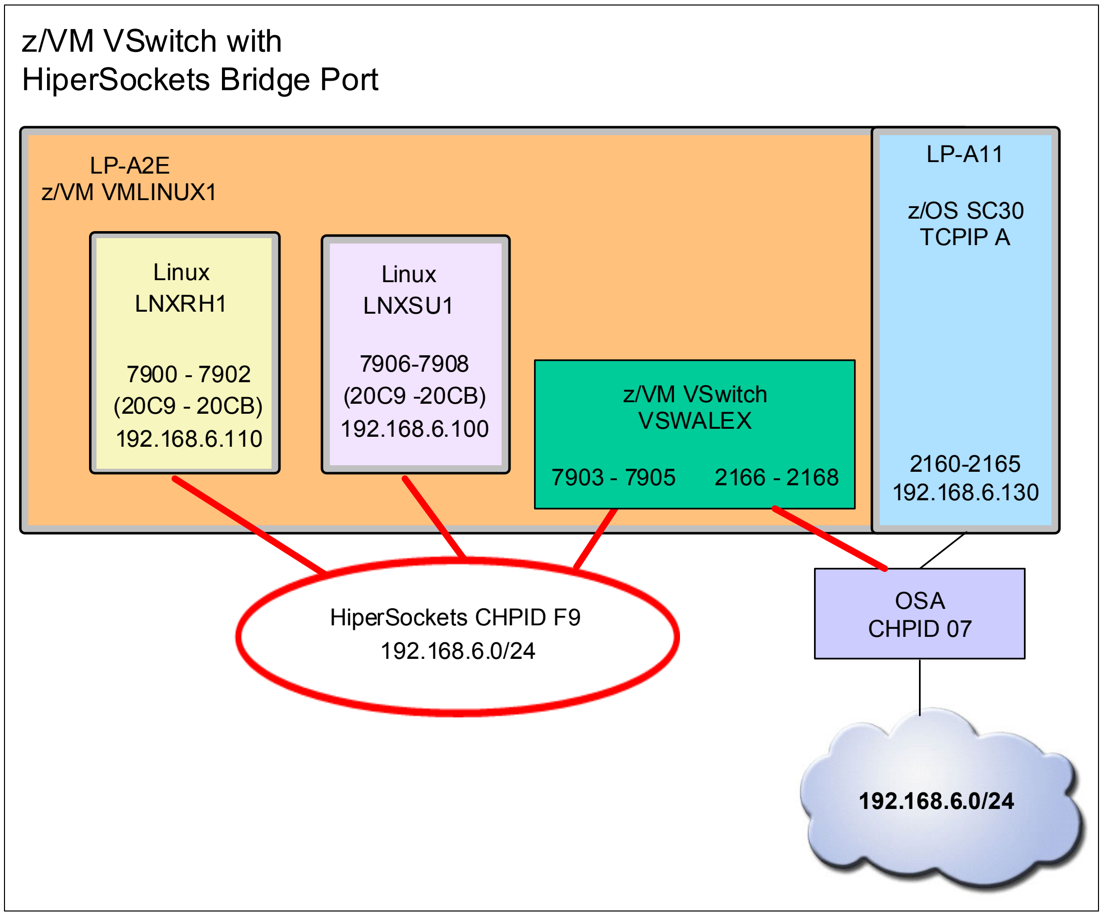


Figure 8-7 IBM z/VM VSwitch with HiperSockets Bridge example scenario

Note that you can also connect Linux guests from other z/VM LPARs to the HiperSockets CHPID F9 and bridge them. You could also have guests directly attached to the Virtual Switch, but directly attaching them to the HiperSockets channel has benefits, as described in 8.4.3, “Benefits of a bridged HiperSockets network” on page 113.

Connecting the Linux guests to HiperSockets CHPID F9

Follow these steps to connect Linux guests to HiperSockets:

1. In the IOCDS hardware definition, define CHPID F9 with channel parameter 04 (HiperSockets for External Bridge) and give the z/VM LPAR A2E access to devices 7900 - 790F of this channel, as shown in Example 8-12.

Example 8-12 HiperSockets devices available in z/VM

```
q chpid f9
Path F9 online to devices 7900 7901 7902 7903 7904 7905 7906 7907
Path F9 online to devices 7908 7909 790A 790B 790C 790D 790E 790F
Ready; T=0.01/0.01 09:37:57
q 7900-790f
OSA  7900 FREE    , OSA  7901 FREE    , OSA  7902 FREE    , OSA  7903 FREE
OSA  7904 FREE    , OSA  7905 FREE    , OSA  7906 FREE    , OSA  7907 FREE
OSA  7908 FREE    , OSA  7909 FREE    , OSA  790A FREE    , OSA  790B FREE
OSA  790C FREE    , OSA  790D FREE    , OSA  790E FREE    , OSA  790F FREE
Ready; T=0.01/0.01 09:38:20
```

2. Now connect some of these devices to the Linux guests, as shown in Example 8-13.

Example 8-13 Connect HiperSockets devices to Linux guests

```
attach 7900 lnxrh1 20c9
attach 7901 lnxrh1 20ca
attach 7902 lnxrh1 20cb
attach 7906 lnxsu1 20c9
attach 7907 lnxsu1 20ca
attach 7908 lnxsu1 20cb

q 7900-790f
OSA  7900 ATTACHED TO LNXRH1   20C9 DEVTYPE HIPER-BRDG  CHPID F9 IQD
OSA  7901 ATTACHED TO LNXRH1   20CA DEVTYPE HIPER-BRDG  CHPID F9 IQD
OSA  7902 ATTACHED TO LNXRH1   20CB DEVTYPE HIPER-BRDG  CHPID F9 IQD
OSA  7903 FREE    , OSA  7904 FREE    , OSA  7905 FREE
OSA  7906 ATTACHED TO LNXSU1   20C9 DEVTYPE HIPER-BRDG  CHPID F9 IQD
OSA  7907 ATTACHED TO LNXSU1   20CA DEVTYPE HIPER-BRDG  CHPID F9 IQD
OSA  7908 ATTACHED TO LNXSU1   20CB DEVTYPE HIPER-BRDG  CHPID F9 IQD
OSA  7909 FREE    , OSA  790A FREE    , OSA  790B FREE    , OSA  790C FREE
OSA  790D FREE    , OSA  790E FREE    , OSA  790F FREE
Ready; T=0.01/0.01 09:43:35
```

3. Next, configure the network interfaces to subnet 192.168.6.0/24 in RHEL, as shown in Example 8-14. Remember, interfaces to External Bridge channels need to be defined as layer 2 interfaces **1**!

Example 8-14 Layer 2 interface for RHEL

```
[root@lnxrh1 ~]# cio_ignore -r 0.0.20c9,0.0.20ca,0.0.20cb
[root@lnxrh1 ~]# znetconf -a 20c9 -o layer2="1" 1
Scanning for network devices...
Successfully configured device 0.0.20c9 (hsi1)
[root@lnxrh1 ~]# lsqeth hsi1
Device name                 : hsi1
---------------------------------------------
        card_type           : HiperSockets
        cdev0               : 0.0.20c9
        cdev1                : 0.0.20ca
        cdev2                : 0.0.20cb
        chpid                : F9
        online               : 1
        portname             : no portname required
        portno               : 0
        state                : SOFTSETUP
        priority_queueing    : always queue 2
        buffer_count         : 128
        layer2               : 1
        isolation            : none

[root@lnxrh1 ~]# ifconfig hsi1 192.168.6.110
```

Example 8-15 shows the interface configuration for the SUSE Linux guest.

Example 8-15 Layer 2 interface for SUSE

```
lnxsu1:~ # znetconf -a 20c9 -o layer2="1"
Scanning for network devices...
Successfully configured device 0.0.20c9 (hsi2)
lnxsu1:~ # ifconfig hsi2 192.168.6.100
```

Example 8-16 shows that now these two Linux guests can ping each other. (1 and 2). But they cannot communicate with SC30 yet (3 and 4).

Example 8-16 Ping without Virtual Switch

```
lnxsu1:~ # ping 192.168.6.110 1
PING 192.168.6.110 (192.168.6.110) 56(84) bytes of data.
64 bytes from 192.168.6.110: icmp_seq=1 ttl=64 time=7.98 ms
64 bytes from 192.168.6.110: icmp_seq=2 ttl=64 time=0.156 ms
64 bytes from 192.168.6.110: icmp_seq=3 ttl=64 time=0.168 ms
^C
--- 192.168.6.110 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.156/2.768/7.981/3.686 ms

[root@lnxrh1 ~]# ping  192.168.6.100 2
PING 192.168.6.100 (192.168.6.100) 56(84) bytes of data.
64 bytes from 192.168.6.100: icmp_seq=1 ttl=64 time=0.178 ms
64 bytes from 192.168.6.100: icmp_seq=2 ttl=64 time=0.136 ms
^C
--- 192.168.6.100 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1699ms
rtt min/avg/max/mdev = 0.136/0.157/0.178/0.021 ms

[root@lnxrh1 ~]# ping  192.168.6.130 3
PING 192.168.6.130 (192.168.6.130) 56(84) bytes of data.
^C
--- 192.168.6.130 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2479ms

lnxsu1:~ # ping 192.168.6.130 4

PING 192.168.6.130 (192.168.6.130) 56(84) bytes of data.
^C
```

```
--- 192.168.6.130 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1007ms
```

Defining the z/VM Virtual Switch with HiperSockets Bridge Port

To connect the Linux guests to the external OSA network, define a z/VM Virtual Switch (VSWALEX) with an OSA uplink port and a HiperSockets Bridge Port, as shown in Figure 8-7 on page 116. Note that there are no VM guests attached to this VSwitch.

Example 8-17 shows the necessary steps. The Virtual Switch needs to be defined in layer 2 mode, as indicated by the keyword `ETHERNET` 1. Then define an uplink port to OSA CHPID 07 2, and a bridge port to HiperSockets CHPID F9 3.

Example 8-17 Defining a Virtual Switch

```
define vswitch VSWALEX type QDIO ETHERNET 1
VSWITCH SYSTEM VSWALEX is created
Ready; T=0.01/0.01 10:05:15
set vswitch VSWALEX uplink rdev 2166 2
Command complete
Ready; T=0.01/0.01 10:07:16
HCPSWU2830I VSWITCH SYSTEM VSWALEX status is ready.
HCPSWU2830I DTCVSW1 is VSWITCH controller for device 2166.P00.
set vswitch VSWALEX bridgeport rdev 7903  3
Command complete
Ready; T=0.01/0.01 10:08:35
HCPSWU3048I DTCVSW2 is VSWITCH controller for HiperSockets bridge device 7903.
HCPSWU3048I VSWITCH SYSTEM VSWALEX secondary HiperSockets bridge status is active.
```

Now you can ping SC30 from the Linux guests, as shown in Example 8-18.

Example 8-18 Pinging z/OS from the Linux guests

```
[root@lnxrh1 ~]# ping  192.168.6.130
PING 192.168.6.130 (192.168.6.130) 56(84) bytes of data.
64 bytes from 192.168.6.130: icmp_seq=1 ttl=64 time=0.767 ms
64 bytes from 192.168.6.130: icmp_seq=2 ttl=64 time=0.316 ms
64 bytes from 192.168.6.130: icmp_seq=3 ttl=64 time=0.317 ms
64 bytes from 192.168.6.130: icmp_seq=4 ttl=64 time=0.319 ms
64 bytes from 192.168.6.130: icmp_seq=5 ttl=64 time=0.326 ms
64 bytes from 192.168.6.130: icmp_seq=6 ttl=64 time=0.345 ms
64 bytes from 192.168.6.130: icmp_seq=7 ttl=64 time=0.409 ms
64 bytes from 192.168.6.130: icmp_seq=8 ttl=64 time=0.533 ms
64 bytes from 192.168.6.130: icmp_seq=9 ttl=64 time=0.308 ms
64 bytes from 192.168.6.130: icmp_seq=10 ttl=64 time=0.314 ms
^C
--- 192.168.6.130 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9409ms
rtt min/avg/max/mdev = 0.308/0.395/0.767/0.141 ms
```

```
lnxsu1:~ # ping 192.168.6.130
PING 192.168.6.130 (192.168.6.130) 56(84) bytes of data.
64 bytes from 192.168.6.130: icmp_seq=1 ttl=64 time=0.313 ms
64 bytes from 192.168.6.130: icmp_seq=2 ttl=64 time=0.329 ms
^C
--- 192.168.6.130 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.313/0.321/0.329/0.008 ms
```

Example 8-19 shows that SC30 can also ping the Linux guests.

Example 8-19 Ping from z/OS to Linux guests

```
                                ISPF Command Shell
 Enter TSO or Workstation commands below:

 ===> ping 192.168.6.110 (tcp tcpipe



 Place cursor on choice and press enter to Retrieve command

 => ping 192.168.6.110 (tcp tcpipe
 => ping 192.168.110 (tcp tcpipe
 => ping 192.168.110
 => netstat home (tcp tcpipf
 => netstat dev
 => netstat home
 =>
 =>
 =>
 =>
 CS V2R1: Pinging host 192.168.6.110
 Ping #1 response took 0.000 seconds.
 ***
```

Note that no routing setup is necessary at the endpoints, because they all belong to the same IP subnet. No changes to the Virtual Switch are necessary, if additional servers are added or removed to the HiperSockets network or the external network. You only need to define the Virtual Switch once, then the two networks are bridged.

Connecting a server directly to the Virtual Switch

In Figure 8-8 on page 121, you connect one of the Linux guests directly to the Virtual Switch. This is to show that such a server is also connected to the same bridged network formed by the HiperSockets network and the OSA network. This scenario uses more resources than the previous one, but it can be a solution for OSs that do not yet provide layer 2 support for HiperSockets, because the network interface of the guest is now a simulated OSA interface.

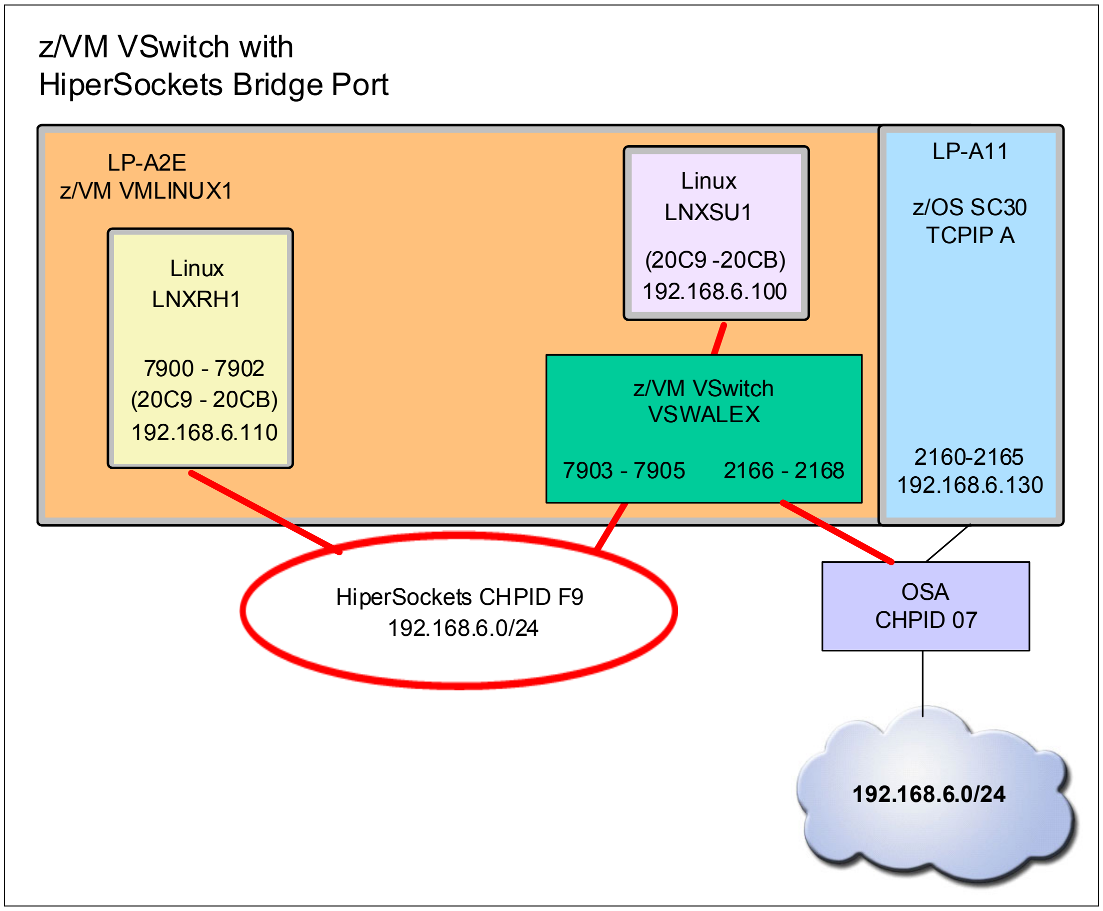


Figure 8-8 HiperSockets Bridge Scenario with z/VM guest attached to the Virtual Switch

Follow these steps to connect a server directly to the Virtual Switch:

1. First remove the HiperSockets interface from LNXSU1, as shown in Example 8-20.

Example 8-20 Remove HiperSockets interface

```
lnxsu1:~ # ifdown hsi2
lnxsu1:~ # znetconf -r 20c9
Remove network device 0.0.20c9 (0.0.20c9,0.0.20ca,0.0.20cb)?
Warning: this may affect network connectivity!
Do you want to continue (y/n)?y
Successfully removed device 0.0.20c9 (hsi2)

In VM MAINT:

detach 7906 lnxsu1 20c9
detach 7907 lnxsu1 20c9
detach 7908 lnxsu1 20c9
```

2. Connect LNXSU1 directly to a simulated OSA interface of the Virtual Switch, as shown in Example 8-21:
   a. First, you need to give LNXSU1 access to VSWALEX **1**.
   b. Then, define a virtual NIC for LNXSU1 **2**.
   c. Finally, configure the interface in Linux **3**. For details about how to configure a z/VM virtual switch, see the z/VM V6R2.0 Information Center:
   `http://pic.dhe.ibm.com/infocenter/zvm/v6r2/index.jsp`

   More information is also available in the *OSA-Express Implementation Guide*, SG24-5948.

Example 8-21 Connect LNXSU1 to VSwitch

```
In MAINT:

set vswitch VSWALEX grant LNXSU1 1
Command complete
Ready; T=0.01/0.01 11:00:17

In VMGuest LNXSU1:
define nic 20c9 type qdio 2
NIC 20C9 is created; devices 20C9-20CB defined
couple 20c9 system VSWALEX
NIC 20C9 is connected to VSWITCH SYSTEM VSWALEX

in lnxsu1:

lnxsu1:~ # znetconf -a 20c9 3
Scanning for network devices...
Successfully configured device 0.0.20c9 (eth2)

lnxsu1:~ # ifconfig eth2 192.168.6.100
```

Pings work! Example 8-22 shows that this Linux guest connected to the z/VM VSwitch can ping both LNXRH1 connected to HiperSockets **1** and SC30 connected to OSA **2**.

Example 8-22 ping from and to LNXSU1

```
lnxsu1:~ # ping 192.168.6.110 1
PING 192.168.6.110 (192.168.6.110) 56(84) bytes of data.
64 bytes from 192.168.6.110: icmp_seq=1 ttl=64 time=7.64 ms
64 bytes from 192.168.6.110: icmp_seq=2 ttl=64 time=0.233 ms
64 bytes from 192.168.6.110: icmp_seq=3 ttl=64 time=0.238 ms
64 bytes from 192.168.6.110: icmp_seq=4 ttl=64 time=0.186 ms
^C
--- 192.168.6.110 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.186/2.074/7.642/3.214 ms

lnxsu1:~ # ping 192.168.6.130 2
PING 192.168.6.130 (192.168.6.130) 56(84) bytes of data.
64 bytes from 192.168.6.130: icmp_seq=1 ttl=64 time=8.11 ms
64 bytes from 192.168.6.130: icmp_seq=2 ttl=64 time=0.333 ms
64 bytes from 192.168.6.130: icmp_seq=3 ttl=64 time=0.301 ms
^C
--- 192.168.6.130 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
```

```
rtt min/avg/max/mdev = 0.301/2.915/8.111/3.674 ms

[root@lnxrh1 ~]# ping  192.168.6.100
PING 192.168.6.100 (192.168.6.100) 56(84) bytes of data.
64 bytes from 192.168.6.100: icmp_seq=1 ttl=64 time=0.210 ms
64 bytes from 192.168.6.100: icmp_seq=2 ttl=64 time=0.214 ms
64 bytes from 192.168.6.100: icmp_seq=3 ttl=64 time=0.229 ms
^C
--- 192.168.6.100 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2299ms
rtt min/avg/max/mdev = 0.210/0.217/0.229/0.018 ms
```

9

HiperSockets in an IBM zEnterprise ensemble

This chapter describes how to use and manage HiperSockets in the IBM zEnterprise and IBM zEnterprise BladeCenter® unified network.

The zEnterprise central processor complexes (CPCs) provide the capability to integrate HiperSockets connectivity into the intraensemble data network (IEDN). In each zEnterprise CPC that is a member of an ensemble, you can elect one HiperSockets (internal queued direct communication, or IQD) channel-path identifier (CHPID) to connect to the IEDN. This capability is enabled through the internal queued direct input/output (I/O) extensions (IQDX) function of HiperSockets.

This chapter describes how to configure IQDX in the zEnterprise Unified Resource Manager (URM). This chapter provides information about how to use IQDX, and how to connect it to the rest of the IEDN network.

9.1 The IBM zEnterprise System

The zEnterprise System, shown in Figure 9-1, is a heterogeneous hardware infrastructure that can consist of three components:

- IBM zEnterprise CPC

  Examples of IBM zEnterprise CPCs include zEnterprise EC12 (zEC12), zEnterprise BC12 (zBC12), zEnterprise 196 (z196), and zEnterprise 114 (z114).

- IBM zEnterprise BladeCenter Extension (zBX)

  The zBX provides the capability to run the wide variety of applications typically found in UNIX and x86 architectures. The zBX supports select IBM POWER7® blades running IBM AIX® and IBM System x® blades running Linux on System x and Microsoft Windows.

- IBM zEnterprise URM

  The zEnterprise URM is firmware that runs on the Hardware Management Console (HMC) and Support Element (SE).The URM consists of six management areas, which are identified in the pie chart in Figure 9-1:

  – Virtual servers
  – Performance
  – Energy
  – Hypervisors
  – Networks
  – Operations

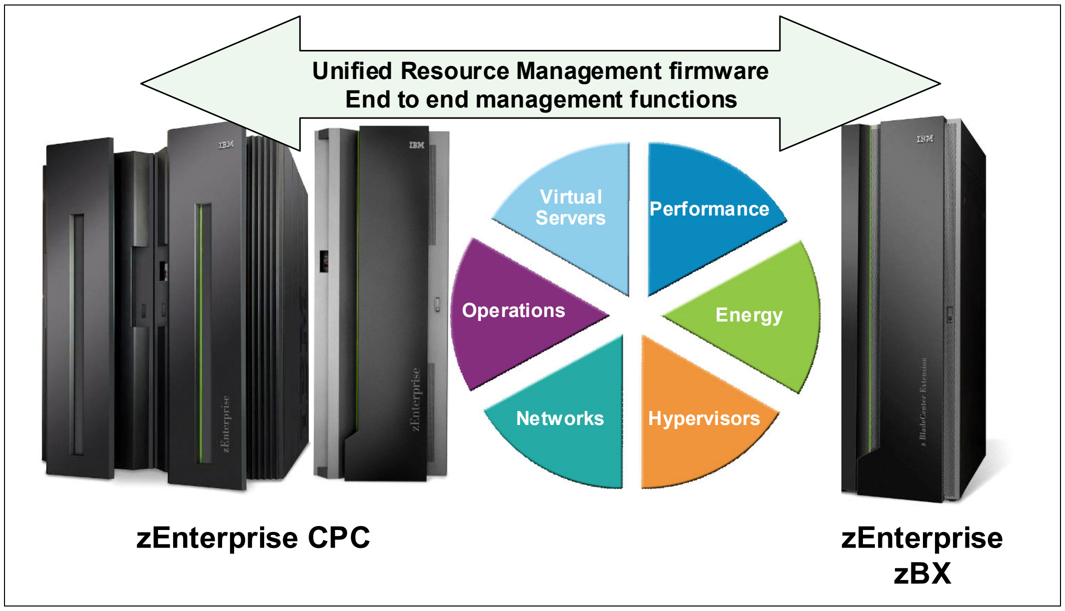


Figure 9-1 BM zEnterprise System components

One management area of the zEnterprise URM is network management (networks). This component creates and manages virtual networks, including access control, which enables virtual servers to be connected to each other.

For more information about the IBM zEnterprise System, see *Building an Ensemble Using IBM zEnterprise Unified Resource Manager*, SG24-7921.

9.2 The intraensemble data network

There are three types of LANs that attach to the zEnterprise system, each with redundant connections:

1. The IEDN
2. The intranode management network (INMN)
3. The client-managed data network

These three networks are depicted in Figure 9-2. The purple boxes and lines represent the URM communicating with agents through the INMN. The orange arrow at the bottom of the figure is the IEDN, which is used to transport data in the ensemble. The gray arrows represent the connections to the client-managed data network outside of the ensemble.

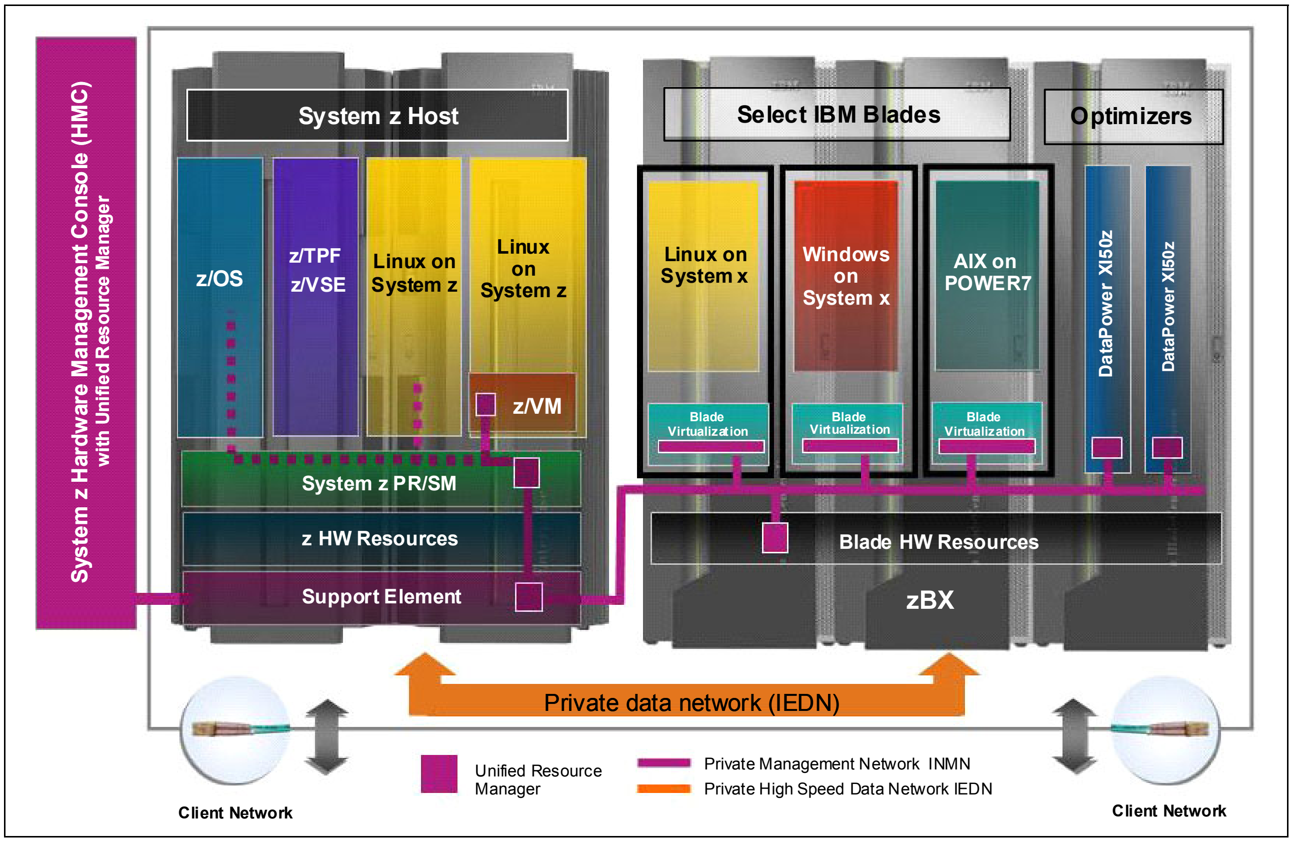


Figure 9-2 Network overview of a zEnterprise System

An IEDN enables communication to flow along the ensemble network components and between the nodes that comprise an ensemble:

- Supports zEnterprise applications communicating between operating system (OS) images to share data
- Supports zBX-to-zBX communication in an ensemble

Each IEDN is supported by a 10 gigabit Ethernet (GbE) Open Systems Adapter (OSA) data link, a single dedicated physical layer 2 network. An IEDN is composed of zEnterprise equipment, and is managed by the URM (on the HMC) as part of the ensemble. The HMC manages the ensemble through its user interface and includes networking tasks that are collectively known as *network virtualization tasks*. The IEDN supports running IPv4 or IPv6 protocols, and Internet Protocol (IP) addresses are client controlled.

Virtual servers must be isolated into groups on the physical network by defining virtual local area networks (VLANs).The IEDN default is one VLAN, and multiple other VLAN subnets can be defined.

Access to the IEDN is controlled by the URM (using the HMC and the SE), the hypervisor, and the physical switches. URM controls the configuration for all switches, and provides secure access to the IEDN. Communication through the IEDN has the benefit of being performed through a physical network in the machine, so it is more secure than external network communications.

For more information about the IEDN of the IBM zEnterprise System, see *Building an Ensemble Using IBM zEnterprise Unified Resource Manager*, SG24-7921.

9.3 HiperSockets for IEDN (IQDX)

The zEnterprise CPCs provide the capability to integrate HiperSockets connectivity to the IEDN. In each zEnterprise CPC that is a member of an ensemble, you can elect one HiperSockets (IQD) CHPID to connect to the IEDN. This capability is enabled through the IQDX function of HiperSockets.

The IQDX function is a channel function, and is not a new CHPID type. It is defined in the input/output configuration data set (IOCDS) by setting the channel parameter of this CHPID to 0x02, 0x42, 0x82, or 0xC2, depending on the maximum frame size (MFS). See 2.1.1, “Channel parameters for HiperSockets” on page 9 for more information about the HiperSockets channel parameters.

When the IQDX function is configured, this IQD CHPID is integrated with the IEDN. For communications in the CPC, the IQDX function enables secure network access using the designated IQD CHPID. Because it is part of the IEDN, the IQDX CHPID (channel parameter x2) shows some significant differences from normal HiperSockets channels (channel parameter x0):

- Because the IEDN is a flat layer 2 network, IQDX network interfaces can only be defined in layer 2 mode. Any attempt by the OS to activate a layer 3 interface will fail. See 5.3, “Layer 2 mode” on page 66 for more information about HiperSockets layer 2 interfaces.
- The URM orchestrates the generation of message authentication code (MAC) addresses for IQDX. See 9.4, “MAC management by the URM” on page 129 for more information.
- The whole IEDN network is VLAN-enforced and VLAN-controlled. See 9.5, “VLAN management by the URM” on page 130 for more information.
- The monitoring task of the URM can be used to display statistics about the usage of the IQDX channel. See 9.9, “Network monitoring with Unified Resource Manager” on page 143 for more information.

There are two ways for a virtual server to connect to the IEDN through IQDX:

- Using a converged interface, as described in 9.7, “The z/OS converged interface” on page 139.
- Using a single IQDX interface that is connected to the physical IEDN network by an IBM z/VM Virtual Switch that bridges IQDX and OSX channels, as described in 9.8, “IBM z/VM Virtual Switch for IEDN” on page 143.

Figure 9-3 illustrates these two types of interfaces to the IQDX channel.

- Server A is a native IBM z/OS logical partition (LPAR) that uses the z/OS converged interface with connect to IQDX and OSA for IEDN (OSX).
- VSwitch B is an IEDN Virtual Switch that bridges all of the queued direct input/output (QDIO) Enhanced Buffer-State Management (QEBSM) guests on the IQDX to the OSX network. In this scenario, Linux QA1-QA4 and Linux QB1-QB4 are QEBSM guests that are attached to the bridged IQDX network.
- The HiperSockets Bridge Port of Virtual Switch A is in standby state, because it is defined as a secondary bridge port. It serves as a backup for the bridge port of VSwitch B.
- Linux VA1 and VB1 and z/OS VA2 and VB2 have no direct interface with IQDX, but only an interface with VSwitch A or VSwitch B. The Virtual Switch bridges these interfaces to IQDX for communication with the QEBSM guests, and to OSX for all other traffic.
- Server B is a native Linux LPAR. Currently, Linux does not provide support for converged interfaces, and could not easily handle two interfaces to the same layer 2 subnet. Defining only an IQDX interface is not an option for server B, because it is not connected through QEBSM (because it is not a z/VM guest), so it cannot be bridged to OSX with z/VM VSwitch B. Therefore, defining an OSX interface and using it to communicate to the other servers in the IEDN is the only option for server B in the native Linux LPAR.

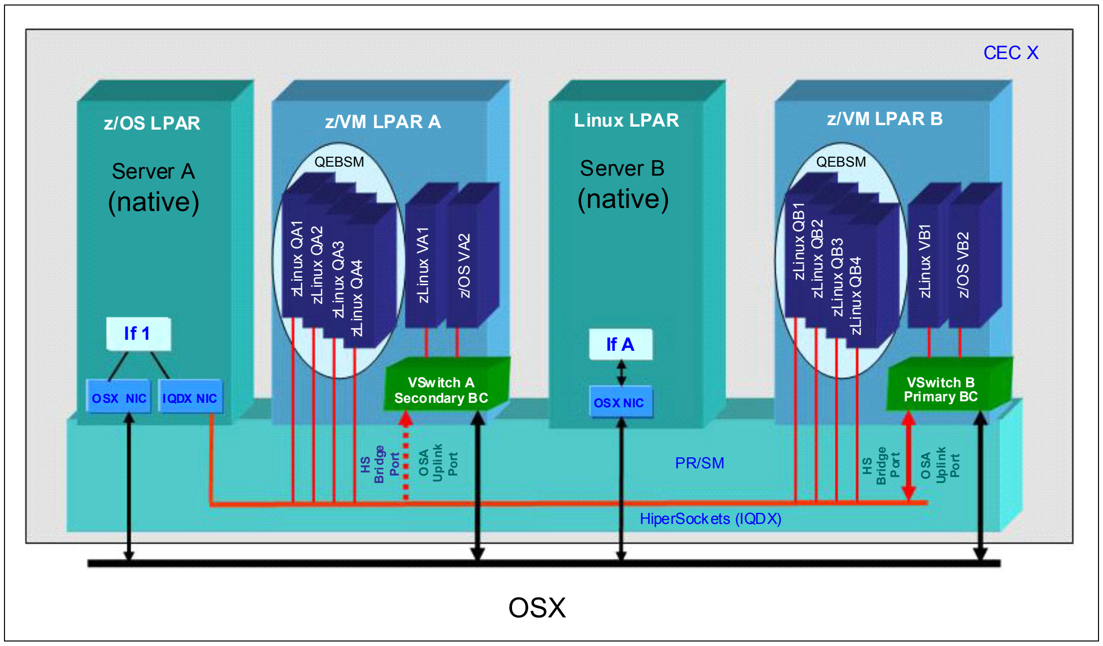


Figure 9-3 Virtual servers connected to the IQDX

9.4 MAC management by the URM

Although the IEDN consists of multiple physical and virtual components (Top-of-Rack switches, OSX cards, IQDX channels, and z/VM Virtual Switches), it is one flat layer 2 network. All of its virtual interfaces need to provide virtual MAC addresses for the virtual servers. The URM orchestrates this provision, so all MAC addresses in the IEDN are unique.

One part of this orchestration is to provide a prefix to the IQDX firmware that is used to generate virtual MAC addresses for the IQDX that do not collide with virtual MAC addresses on the OSX channels or other parts of the IEDN.

As opposed to normal HiperSockets channels or HiperSockets channels for external bridges, the device driver software cannot overwrite these computer-generated MAC addresses on IQDX channels. The computer-generated virtual IEDN MAC address of an IQDX HiperSockets device is predictable, which means that each data device will get the same firmware-generated MAC address again if the server is restarted, or the IQDX CHPID is configured off and on, or even if the CPC is restarted.

If the virtual server is restarted in a different LPAR, a different z/VM guest, or on a different CPC, it will get a different firmware-generated MAC address. The firmware-generated virtual MAC addresses of the IQDX devices are guaranteed to be unique in the IEDN.

The users cannot directly influence which virtual MAC address is assigned to which device of a virtual server in the IEDN, nor can they define the prefix that is assigned to be used by the IQDX. They can, however, reserve MAC address ranges, which must not be used as MAC addresses by the URM. See 9.6.1, “Reserve MAC address ranges” on page 131 for an example of how this can be done.

9.5 VLAN management by the URM

The whole IEDN network is VLAN-enforced and VLAN-controlled. Only VLAN traffic is allowed on the IQDX. A message without a VLAN tag will not be transferred. IQDX interfaces operate in a VLAN-enforced trunk mode. Therefore, the OSs need to be VLAN-aware, and define the VLAN IDs to be used, but they can only use VLAN IDs that were previously assigned by the URM to their virtual servers.

See Chapter 6, “Virtual local area network support” on page 69 for more information about VLAN support for HiperSockets, and see 9.6.3, “Add virtual servers to a VLAN” on page 135 for more information about how to assign VLAN IDs to virtual servers in the URM.

9.6 Using URM to manage IQDX

The test system had IQD CHPID F8 defined as IQDX in the IOCDS with access to the z/OS LPARs A11, A13, and A16. The z/VM system in LPAR A2E was running with z/VM V6.3, which is not managed by the URM. Therefore, it was not connected to the IQDX.

Only z/VM V6.2 and later, and its guests and virtual switches, can be managed by the URM. Other z/VM versions can run in ensemble LPARs in the same way as other OSs, but not as managed hypervisors.

9.6.1 Reserve MAC address ranges

Figure 9-4 shows the ensemble management view of the URM on your HMC. Follow these steps to reserve MAC address ranges:

1. Log on to your HMC with a user ID that has the correct authority to perform ensemble tasks.
2. Select an ensemble. This displays a menu item labeled **Ensemble Details**.

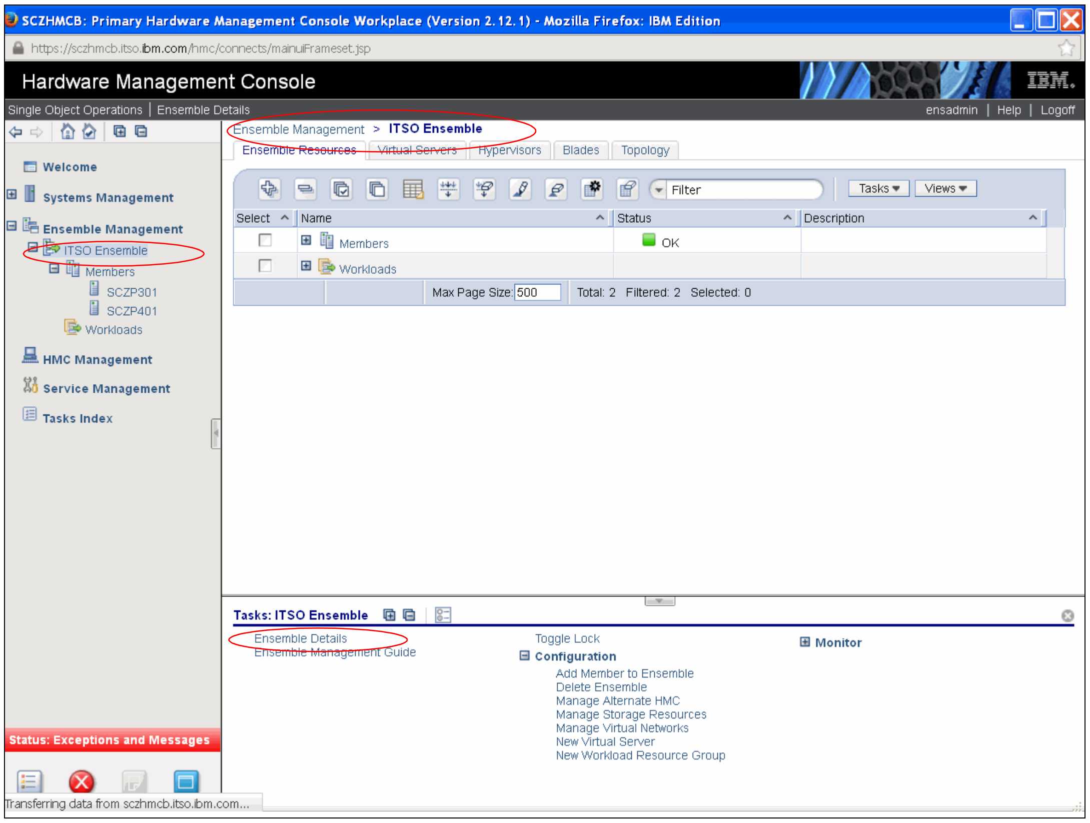
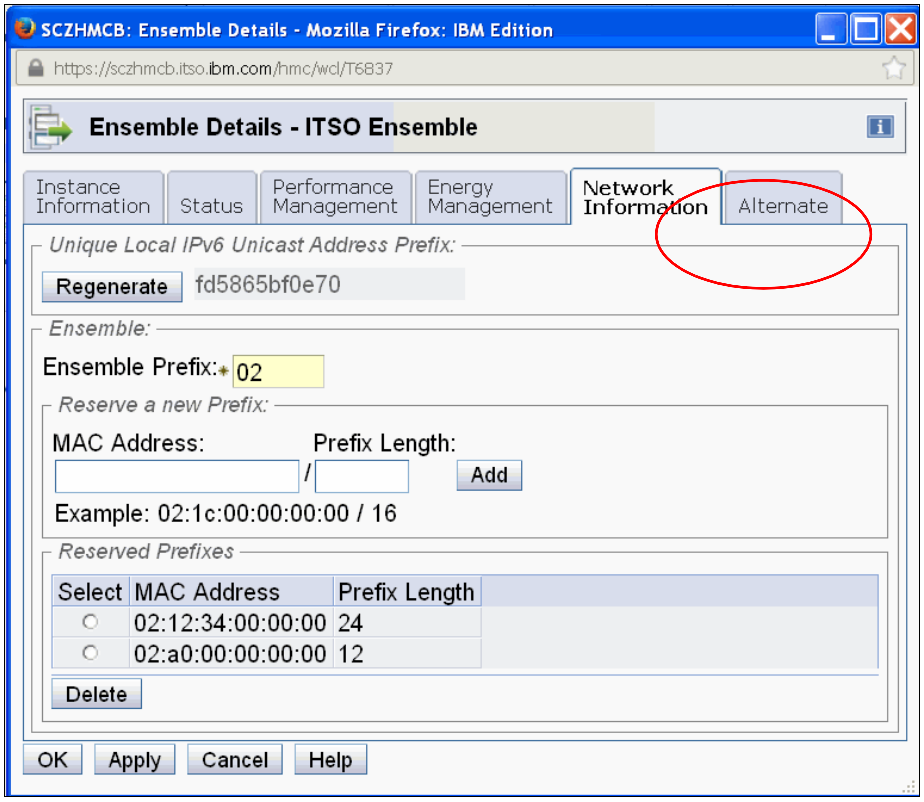


Figure 9-4 Unified Resource Manager - Ensemble view

3. Click **Ensemble Details**. You will get to the Network Information tab of the Ensemble Details window, as shown in Figure 9-5.
4. On this tab, you can reserve ranges of MAC addresses that the URM must *not* use for the IEDN network. In this example, MAC addresses 02:12:34:00:00:00 - 02:12:34:ff:ff:ff and 02:a0:00:00:00:00 - 02:a0:00:00:0f:ff will not be used for IEDN.


SCZHMCB: Ensemble Details - Mozilla Firefox: IBM Edition
https://sczhmcb.itso.ibm.com/hmc/wcl/T6837
Ensemble Details - ITSO Ensemble
Instance Information
Status
Performance Management
Energy Management
Network Information
Alternate
Regenerate
Add
Delete
OK
Apply
Cancel
Help

Unique Local IPv6 Unicast Address Prefix: fd5865bf0e70

Ensemble:

Ensemble Prefix: 02

Reserve a new Prefix:

MAC Address: / Prefix Length:

Example: 02:1c:00:00:00:00 / 16

Reserved Prefixes

Select	MAC Address	Prefix Length
○	02:12:34:00:00:00	24
○	02:a0:00:00:00:00	12

Figure 9-5 Ensemble details and reserving MAC address ranges

The ensemble prefix is chosen when an ensemble is created. It cannot be changed for a running ensemble.

9.6.2 Define VLANs

Figure 9-6 shows the ensemble management view of the URM on your HMC. Follow these steps to define VLANs:

1. Log on to your HMC with a user ID that has the correct authority to perform ensemble tasks.
2. Select an ensemble. This displays a menu item labeled **Manage Virtual Networks** in the **Configuration** menu of your ensemble.

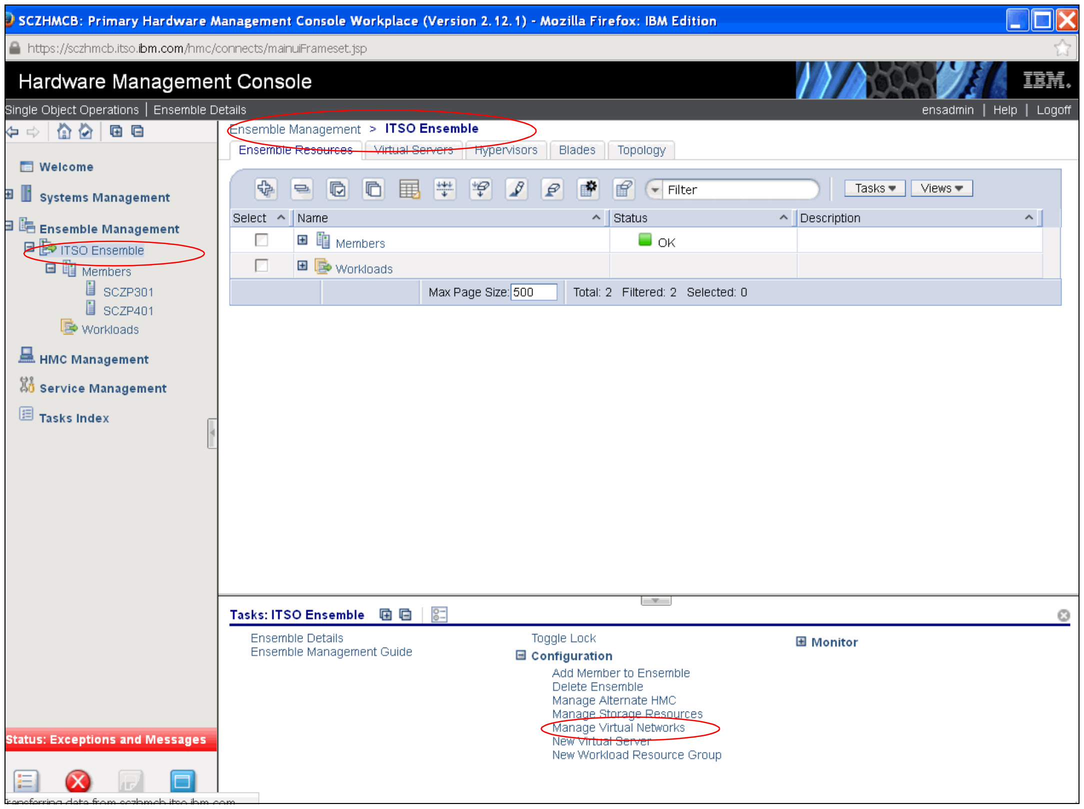
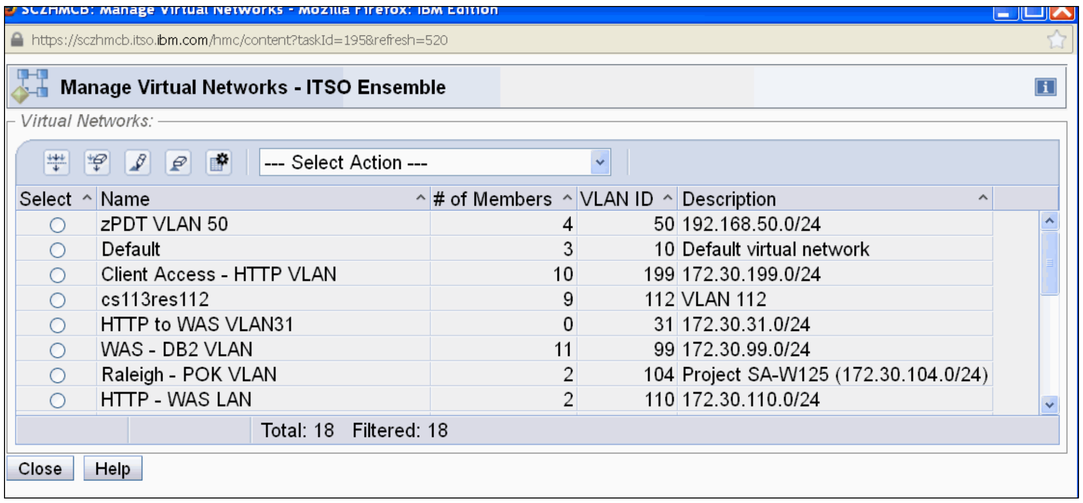


Figure 9-6 Unified Resource Manager - Ensemble view

3. Click **Manage Virtual Networks**. You will get to the Virtual Network pane, as shown in Figure 9-7.


SCZHMCB: Manage Virtual Networks - Mozilla Firefox: IBM Edition

https://sczhmcb.itso.ibm.com/hmc/content?taskId=195&refresh=520

Manage Virtual Networks - ITSO Ensemble

Virtual Networks:

--- Select Action ---

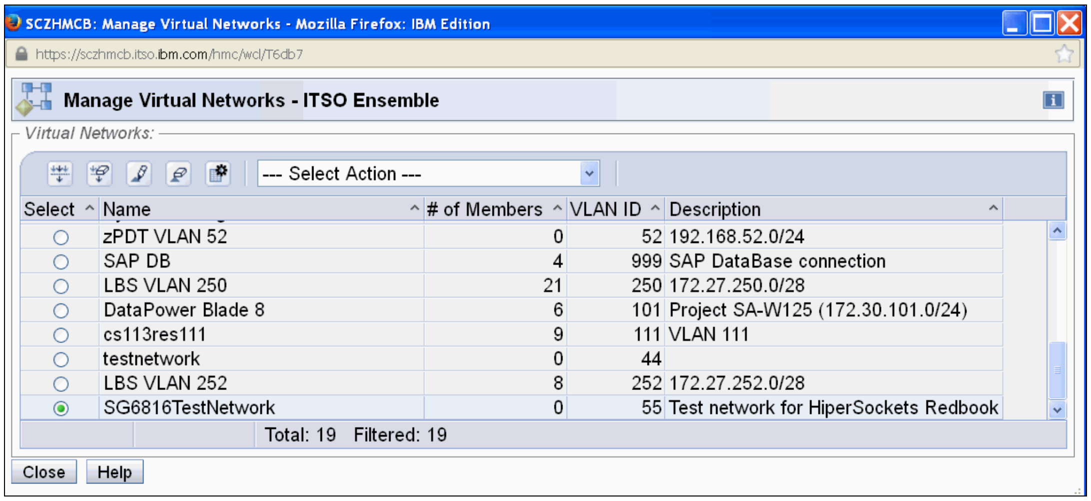
Select	Name	# of Members	VLAN ID	Description
○	zPDT VLAN 50	4	50	192.168.50.0/24
○	Default	3	10	Default virtual network
○	Client Access - HTTP VLAN	10	199	172.30.199.0/24
○	cs113res112	9	112	VLAN 112
○	HTTP to WAS VLAN31	0	31	172.30.31.0/24
○	WAS - DB2 VLAN	11	99	172.30.99.0/24
○	Raleigh - POK VLAN	2	104	Project SA-W125 (172.30.104.0/24)
○	HTTP - WAS LAN	2	110	172.30.110.0/24

Total: 18 Filtered: 18

Close Help

Figure 9-7 Manage Virtual Networks pane

4. Select the action **New Virtual Network** to define a SG6816TestNetwork with the VLAN ID 55. The result is shown in Figure 9-8.


SCZHMCB: Manage Virtual Networks - Mozilla Firefox: IBM Edition

https://sczhmcb.itso.ibm.com/hmc/wcl/T6db7

Manage Virtual Networks - ITSO Ensemble

Virtual Networks:

--- Select Action ---

Select	Name	# of Members	VLAN ID	Description
○	zPDT VLAN 52	0	52	192.168.52.0/24
○	SAP DB	4	999	SAP DataBase connection
○	LBS VLAN 250	21	250	172.27.250.0/28
○	DataPower Blade 8	6	101	Project SA-W125 (172.30.101.0/24)
○	cs113res111	9	111	VLAN 111
○	testnetwork	0	44	
○	LBS VLAN 252	8	252	172.27.252.0/28
●	SG6816TestNetwork	0	55	Test network for HiperSockets Redbook

Total: 19 Filtered: 19

Close Help

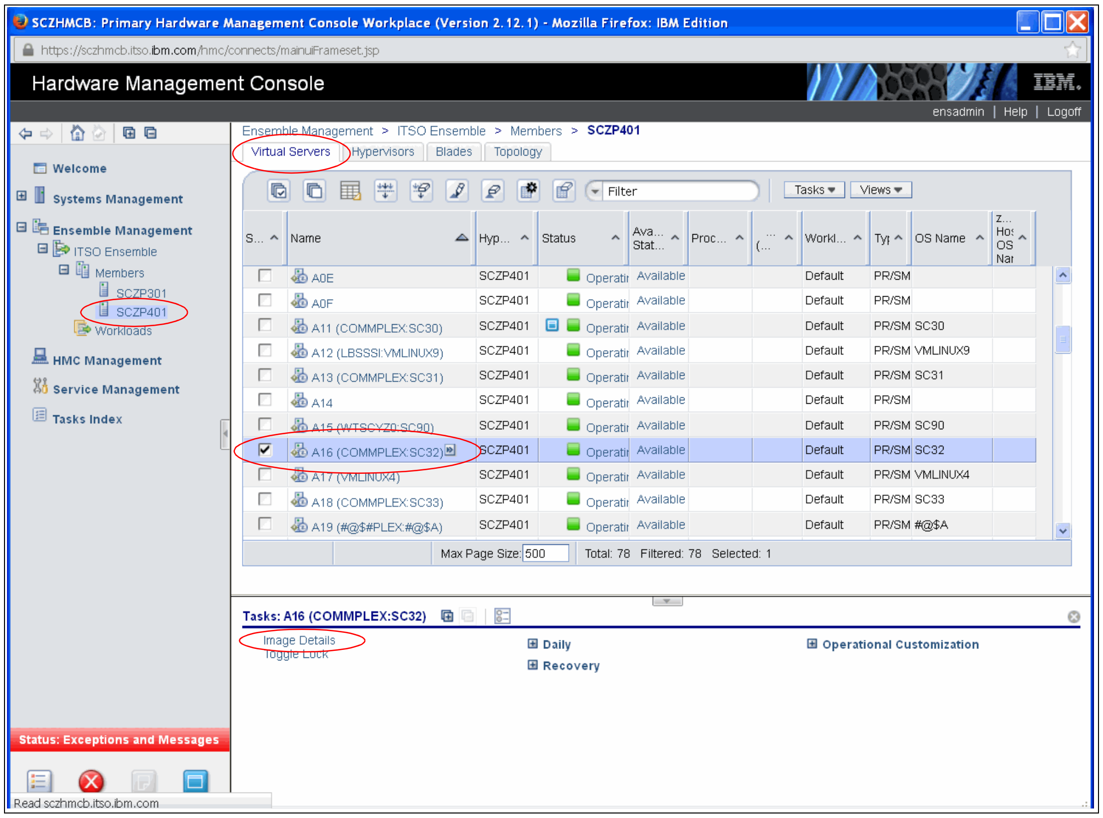
Figure 9-8 Manage Virtual Networks pane with SG6816TestNetwork

9.6.3 Add virtual servers to a VLAN

Figure 9-9 shows the Virtual Server view of the URM on your HMC.

To add virtual servers to the VLAN, follow these steps:

1. Log on to your HMC with a user ID that has the correct authority to perform ensemble tasks and select a virtual server. This opens a menu item **Image Details**.



SCZHMCB: Primary Hardware Management Console Workplace (Version 2.12.1) - Mozilla Firefox: IBM Edition
https://sczhmcb.itso.ibm.com/hmc/connects/mainuiFrameset.jsp
Hardware Management Console
ensadmin | Help | Logoff
Welcome
Systems Management
Ensemble Management
ITSO Ensemble
Members
SCZP301
SCZP401
Workloads
HMC Management
Service Management
Tasks Index
Status: Exceptions and Messages
Ensemble Management > ITSO Ensemble > Members > SCZP401
Virtual Servers
Hypervisors
Blades
Topology
Filter
Tasks
Views

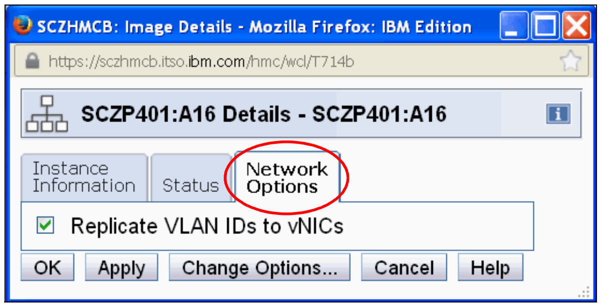
S...	Name	Hyp...	Status	Ava... Stat...	Proc...	(...	Workl...	Typ	OS Name	z... Hos OS Nar
☐	A0E	SCZP401	🟩 Operati	Available			Default	PR/SM		
☐	A0F	SCZP401	🟩 Operati	Available			Default	PR/SM		
☐	A11 (COMMPLEX:SC30)	SCZP401	🟩 Operati	Available			Default	PR/SM	SC30	
☐	A12 (LBSSSI:VMLINUX9)	SCZP401	🟩 Operati	Available			Default	PR/SM	VMLINUX9	
☐	A13 (COMMPLEX:SC31)	SCZP401	🟩 Operati	Available			Default	PR/SM	SC31	
☐	A14	SCZP401	🟩 Operati	Available			Default	PR/SM		
☐	A15 (WTSCYZ0:SC90)	SCZP401	🟩 Operati	Available			Default	PR/SM	SC90	
✓	A16 (COMMPLEX:SC32)	SCZP401	🟩 Operati	Available			Default	PR/SM	SC32	
☐	A17 (VMLINUX4)	SCZP401	🟩 Operati	Available			Default	PR/SM	VMLINUX4	
☐	A18 (COMMPLEX:SC33)	SCZP401	🟩 Operati	Available			Default	PR/SM	SC33	
☐	A19 (#@$#PLEX:#@$A)	SCZP401	🟩 Operati	Available			Default	PR/SM	#@$A	

Max Page Size: 500 Total: 78 Filtered: 78 Selected: 1

Tasks: A16 (COMMPLEX:SC32)

Figure 9-9 Unified Resource Manager—Virtual Server view

2. If you click **Image Details** you will get to the Network Options tab of the Image Details window as shown in Figure 9-10 on page 135.



SCZHMCB: Image Details - Mozilla Firefox: IBM Edition
https://sczhmcb.itso.ibm.com/hmc/wcl/T714b
SCZP401:A16 Details - SCZP401:A16
Instance Information
Status
Network Options
OK
Apply
Change Options...
Cancel
Help

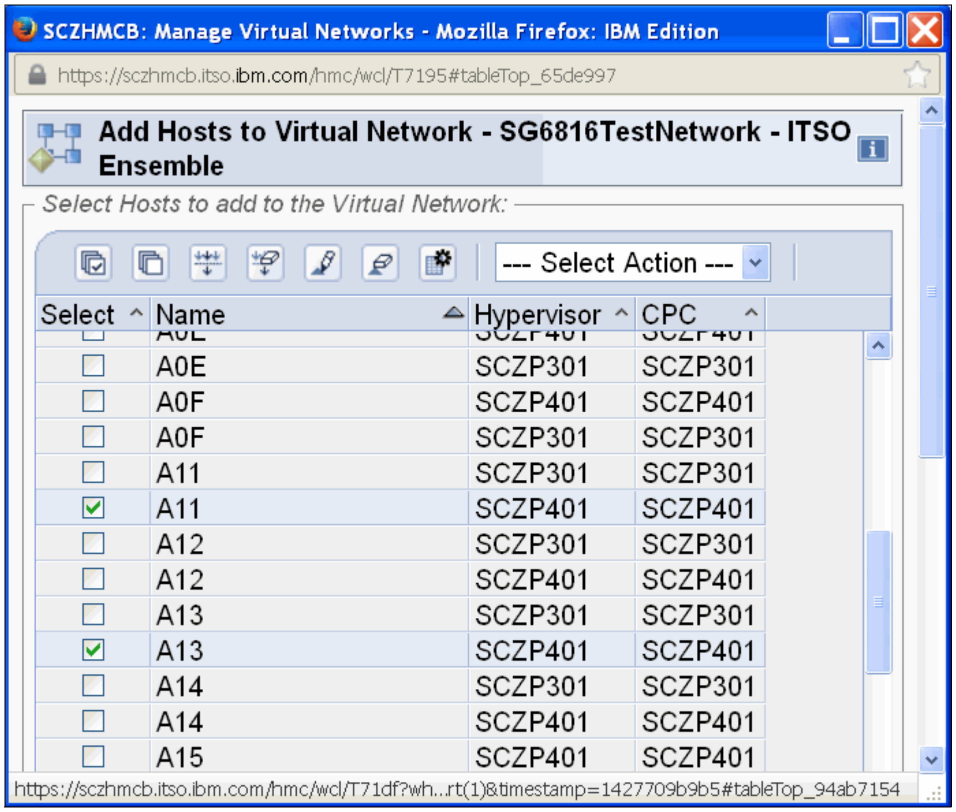
☑ Replicate VLAN IDs to vNICs

Figure 9-10 Virtual Server Image Details window

3. There is a check box for **Replicate VLAN IDs to vNICs**, which is selected by default. If it is selected and you add this server to a VLAN, then all of the IEDN (virtual) network interfaces (vNICs) that belong to this server will be authorized to use this VLAN.

In most scenarios this will be what you want to accomplish. It saves you the chore of enabling each vNIC separately.

4. To add a virtual server to a VLAN, go back to the Ensemble view shown in Figure 9-6 on page 133 and click **Manage Virtual Networks** to get to the Virtual Network window, as shown in Figure 9-7 on page 134.
5. Now, select the SG6816TestNetwork and select the action **Add Hosts to Virtual Network** to open the Add Hosts to Virtual network window as shown in Figure 9-11.


SCZHMCB: Manage Virtual Networks - Mozilla Firefox: IBM Edition

https://sczhmcb.itso.ibm.com/hmc/wcl/T7195#tableTop_65de997

Add Hosts to Virtual Network - SG6816TestNetwork - ITSO Ensemble

Select Hosts to add to the Virtual Network:

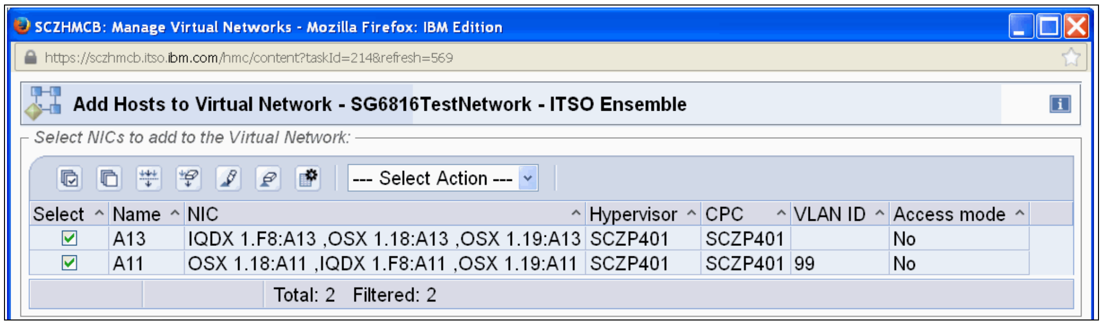
--- Select Action ---

Select	Name	Hypervisor	CPC
☐	A0E	SCZP401	SCZP401
☐	A0E	SCZP301	SCZP301
☐	A0F	SCZP401	SCZP401
☐	A0F	SCZP301	SCZP301
☐	A11	SCZP301	SCZP301
✓	A11	SCZP401	SCZP401
☐	A12	SCZP301	SCZP301
☐	A12	SCZP401	SCZP401
☐	A13	SCZP301	SCZP301
✓	A13	SCZP401	SCZP401
☐	A14	SCZP301	SCZP301
☐	A14	SCZP401	SCZP401
☐	A15	SCZP401	SCZP401

https://sczhmcb.itso.ibm.com/hmc/wcl/T71df?wh...rt(1)&timestamp=1427709b9b5#tableTop_94ab7154

Figure 9-11 Add Hosts to Virtual Network window

6. Select A11 and A13, two virtual servers that have an active **Replicate VLAN IDs to vNICs** option, as shown in Figure 9-10 on page 135.
7. Click **Next** and a confirmation window, shown in Figure 9-12, opens.


SCZHMCB: Manage Virtual Networks - Mozilla Firefox: IBM Edition

https://sczhmcb.itso.ibm.com/hmc/content?taskId=214&refresh=569

Add Hosts to Virtual Network - SG6816TestNetwork - ITSO Ensemble

Select NICs to add to the Virtual Network:

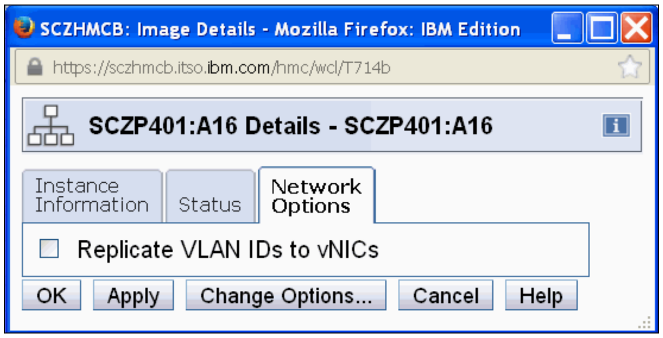
--- Select Action ---

Select	Name	NIC	Hypervisor	CPC	VLAN ID	Access mode
✓	A13	IQDX 1.F8:A13 ,OSX 1.18:A13 ,OSX 1.19:A13	SCZP401	SCZP401		No
✓	A11	OSX 1.18:A11 ,IQDX 1.F8:A11 ,OSX 1.19:A11	SCZP401	SCZP401	99	No

Total: 2 Filtered: 2

Figure 9-12 Add Hosts to Virtual Network confirmation pane

8. You can only add a virtual server with all of its vNICs or with none of them. In this example, both LPARs A11 and A13 have interfaces to OSX CHPIDs 1.18 and 1.19 and to IQDX CHPID 1.F8. LPAR A11 is already authorized to use VLAN ID 99, but LPAR A13 is not yet authorized for a VLAN ID. Click **OK** to add the two LPARs to the SG6816TestNetwork with VLAN ID 55.
9. For the third z/OS LPAR, LPAR A16, go back to the Image Details window, as shown in Figure 9-13, and clear the **Replicate VLAN IDs to vNICs** check box.


SCZHMCB: Image Details - Mozilla Firefox: IBM Edition

https://sczhmcb.itso.ibm.com/hmc/wcl/T714b

SCZP401:A16 Details - SCZP401:A16

Instance Information | Status | Network Options

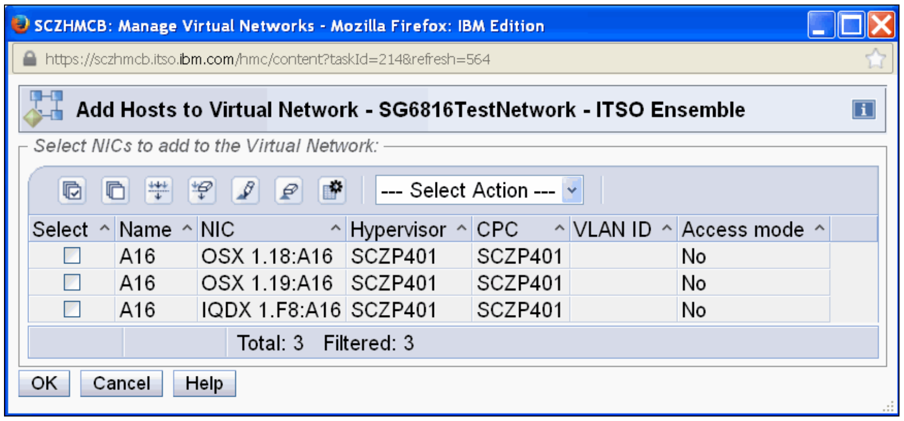
☐ Replicate VLAN IDs to vNICs

OK | Apply | Change Options... | Cancel | Help

Figure 9-13 Virtual Server Image Details window with cleared check box

10. Use the **Add Hosts to Virtual Network** window as shown in Figure 9-11 on page 136 to add A16 to the SG6816TestNetwork.
11. Click **Next** to get a different confirmation window, as shown in Figure 9-14. You notice that you now can select which network interfaces ought to be authorized for the VLAN. In this example, you select all interfaces and click **OK.**

    In most cases, it will not be necessary to authorize single vNICs for a VLAN, but this option is available for special cases. In-depth knowledge about how the virtual server will use the interfaces is required when using this option.


SCZHMCB: Manage Virtual Networks - Mozilla Firefox: IBM Edition

https://sczhmcb.itso.ibm.com/hmc/content?taskId=214&refresh=564

Add Hosts to Virtual Network - SG6816TestNetwork - ITSO Ensemble

Select NICs to add to the Virtual Network:

--- Select Action ---

Select	Name	NIC	Hypervisor	CPC	VLAN ID	Access mode
☐	A16	OSX 1.18:A16	SCZP401	SCZP401		No
☐	A16	OSX 1.19:A16	SCZP401	SCZP401		No
☐	A16	IQDX 1.F8:A16	SCZP401	SCZP401		No

Total: 3 Filtered: 3

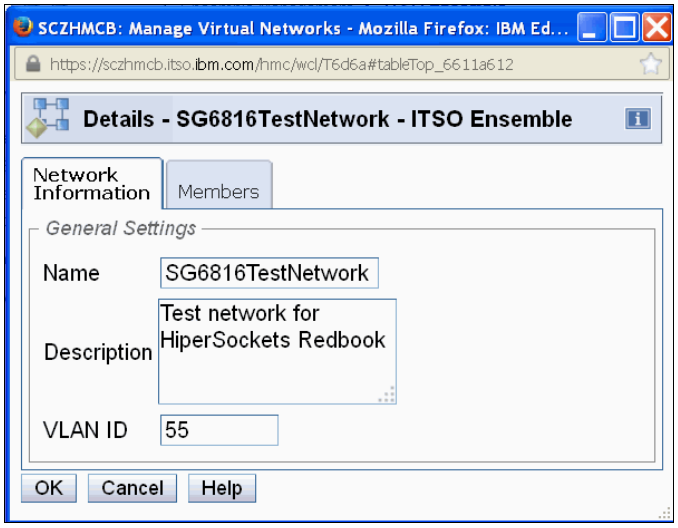
OK | Cancel | Help

Figure 9-14 Add Hosts to Virtual Network confirmation window with separate vNICs

9.6.4 Verify details of a VLAN

To verify the details, follow these steps:

1. To examine the virtual network settings, go to the Ensemble view (Figure 9-6 on page 133) and click **Manage Virtual Networks** to get to the Virtual Network window. Now select the SG6816TestNetwork and select the action **Details** to open the Details window, as shown in Figure 9-15.


SCZHMCB: Manage Virtual Networks - Mozilla Firefox: IBM Ed...

https://sczhmcb.itso.ibm.com/hmc/wcl/T6d6a#tableTop_6611a612

Details - SG6816TestNetwork - ITSO Ensemble

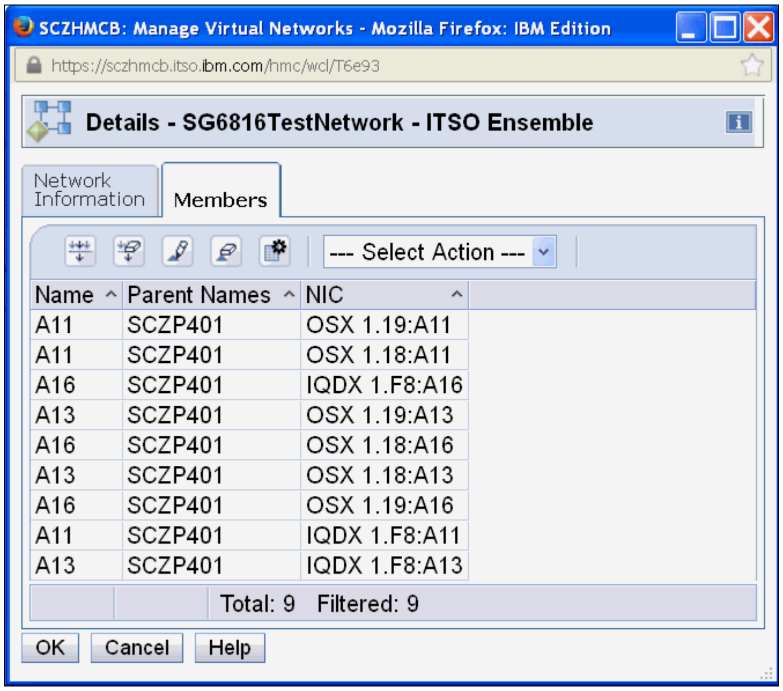
Network Information | Members

General Settings

Name	SG6816TestNetwork
Description	Test network for HiperSockets Redbook
VLAN ID	55

Figure 9-15 Virtual Network Details window

2. Go to the Members tab to see a list of all of the virtual servers and vNICs that are authorized to access this virtual network, as shown in Figure 9-16.


SCZHMCB: Manage Virtual Networks - Mozilla Firefox: IBM Edition

https://sczhmcb.itso.ibm.com/hmc/wcl/T6e93

Details - SG6816TestNetwork - ITSO Ensemble

Network Information | Members

Name	Parent Names	NIC
A11	SCZP401	OSX 1.19:A11
A11	SCZP401	OSX 1.18:A11
A16	SCZP401	IQDX 1.F8:A16
A13	SCZP401	OSX 1.19:A13
A16	SCZP401	OSX 1.18:A16
A13	SCZP401	OSX 1.18:A13
A16	SCZP401	OSX 1.19:A16
A11	SCZP401	IQDX 1.F8:A11
A13	SCZP401	IQDX 1.F8:A13

Total: 9 Filtered: 9

Figure 9-16 Members of a virtual network

9.7 The z/OS converged interface

HiperSockets connectivity to the IEDN is referred to as the *z/OS Communications Server IEDN-enabled HiperSockets function*. Figure 9-17 shows the key concepts of this function. HiperSockets connectivity is transparently converged with OSA-Express connectivity.

Communications Server provides transparent access to the IQD CHPID that is configured for the IQDX function. For each configured OSX CHPID and for each IP version, Communications Server dynamically creates and associates an IQDX interface and transport resource list element (TRLE) to the IQD CHPID. The dynamically created IQDX interfaces inherit the VLAN IDs and IP addresses of their associated OSX interfaces, therefore eliminating the need for IP topology, routing, or security changes in the network.

Communications Server dynamically determines whether traffic routed to the IEDN over a given OSX CHPID can also use the IQDX interface. If the traffic can use the IQDX interface, Communications Server routes the traffic over the high-performance HiperSockets IQD CHPID.

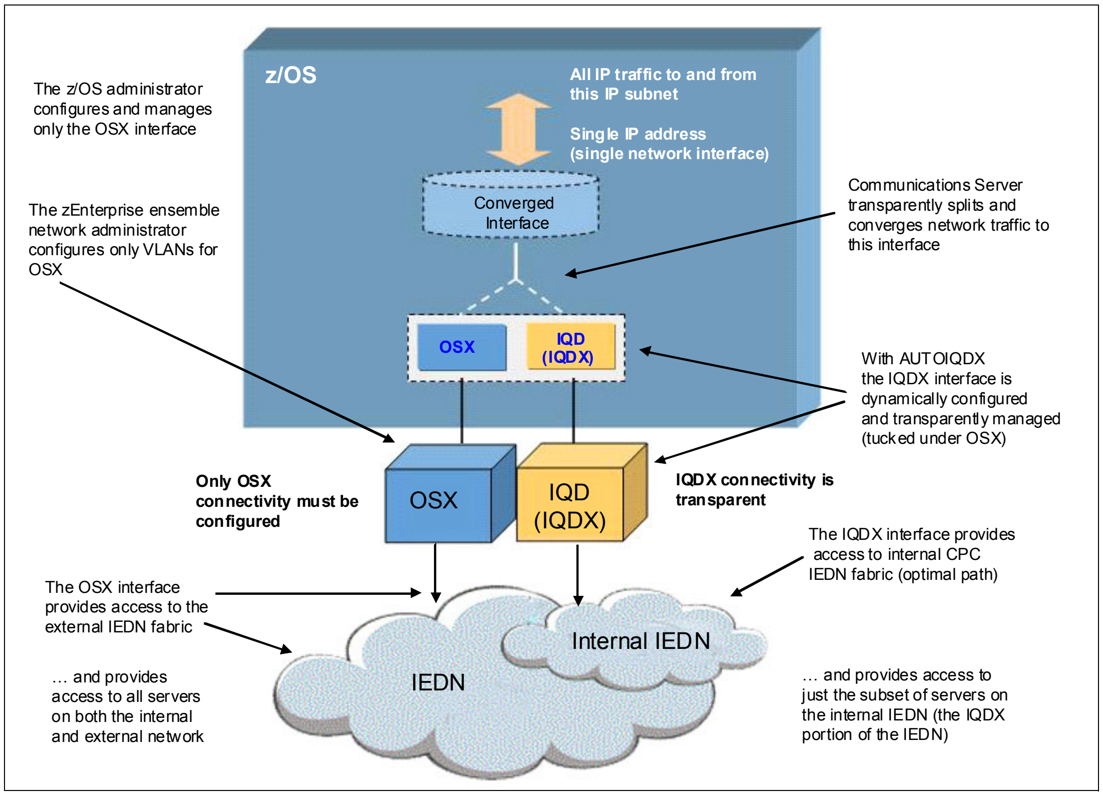

The z/OS administrator configures and manages only the OSX interface
z/OS
All IP traffic to and from this IP subnet
Single IP address (single network interface)
Converged Interface
The zEnterprise ensemble network administrator configures only VLANs for OSX
Communications Server transparently splits and converges network traffic to this interface
OSX
IQD (IQDX)
With AUTOIQDX the IQDX interface is dynamically configured and transparently managed (tucked under OSX)
Only OSX connectivity must be configured
OSX
IQD (IQDX)
IQDX connectivity is transparent
The IQDX interface provides access to internal CPC IEDN fabric (optimal path)
The OSX interface provides access to the external IEDN fabric
Internal IEDN
IEDN
… and provides access to all servers on both the internal and external network
… and provides access to just the subset of servers on the internal IEDN (the IQDX portion of the IEDN)

Figure 9-17 The z/OS Communications Server IEDN-enabled HiperSockets overview

OSX and IQDX interfaces of the same virtual server need to be authorized for the same VLAN IDs in the URM. This is best achieved by the Replicate VLAN IDs to vNICs option (which is enabled by default) as described in 9.6.3, “Add virtual servers to a VLAN” on page 135.

AUTOIQDX and ALLTRAFFIC are default values for GLOBALCONFIG. So there is basically no need to define anything else than the OSX interface. Then z/OS will automatically try to generate converged interfaces and use them.

For more information, see the z/OS Communications Server Information Center:

http://pic.dhe.ibm.com/infocenter/zos/v2r1/index.jsp

There, expand the following topic: **z/OS Communications Server → z/OS Communications Server: IP Configuration Guide → Base TCP/IP system → TCP/IP in an ensemble → HiperSockets connectivity to the intraensemble data network**.

9.7.1 How to define the converged interface

As explained in 9.7, "The z/OS converged interface" on page 139, most of the definitions are done by the system. This is the sequence of tasks that are required to define a converged interface for SC30 TCPIPE:

1. Update VTAM ATCSTRxx. Set the **ENSEMBLE** parameter to YES:

   ```
   F NET,VTAMOPTS,ENSEMBLE=YES
   ```

2. Add OSX definitions to the TCP PROFILE, as shown in Example 9-1.

   Example 9-1 OSX definition in TCP/IP profile

   ```
   ; ------------- IEDN INTERFACES FOR ENSEMBLE ATTACHMENTS ---------
   ; -------------          VLAN 55 for TEST                ---------
   INTERFACE OSX1111
   DEFINE IPAQENET CHPIDTYPE OSX        1
   CHPID 18 VLANID 55                 2
   MTU 8992 IPADDR 192.168.55.1/24
   ;
   BEGINROUTES
    ROUTE 192.168.55.0 255.255.255.0 =          OSX1111     MTU 8992
   ENDROUTES OSX1111
   ```

   The following elements are defined:

   – The CHPID type is an OSX 1.
   – The VLAN ID is mandatory for defining the converged interface 2.

   This set of definitions has to be defined for every LPAR that uses a converged interface.

3. Example 9-2 shows the output of TCPIPE. Note the IQDX interface EZAIQX18 that was automatically generated. Also note that EZAIQX18 does not have its own IP address, but is only used together with OSX1111. This function also applies to IPv6. If you start an OSX IPV6 interface for this CHPID, the stack generates IQDX interface EZ6IOX18.

   Example 9-2 TCPIP E output

   ```
   Display  Filter  View  Print  Options  Search  Help

   -------------------------------------------------------------------------------
    SDSF OUTPUT DISPLAY TCPIPE   STC04854  DSID     2 LINE 236     COLUMNS 16- 95
    COMMAND INPUT ===>                                           SCROLL ===> CSR
        INTFNAME: OSX1111          INTFTYPE: IPAQENET   INTFSTATUS: READY
            PORTNAME: IUTXP018  DATAPATH: 2303     DATAPATHSTATUS: READY
            CHPIDTYPE: OSX      CHPID: 18
            PNETID: IEDN
            SPEED: 0000010000
            IPBROADCASTCAPABILITY: NO
            VMACADDR: 02279200002D   VMACORIGIN: OSA    VMACROUTER: ALL
            ARPOFFLOAD: YES                 ARPOFFLOADINFO: YES
   ```

```
    CFGMTU: 8992                    ACTMTU: 8992
    **IPADDR: 192.168.55.1/24**
    **VLANID: 55**                      VLANPRIORITY: DISABLED
    DYNVLANREGCFG: NO               DYNVLANREGCAP: YES
    READSTORAGE: GLOBAL (4096K)
    INBPERF: DYNAMIC
      WORKLOADQUEUEING: NO
    CHECKSUMOFFLOAD: YES            SEGMENTATIONOFFLOAD: NO
    SECCLASS: 255                   MONSYSPLEX: NO
    ISOLATE: NO                     OPTLATENCYMODE: NO
  MULTICAST SPECIFIC:
    MULTICAST CAPABILITY: YES
    GROUP             REFCNT        SRCFLTMD
    -----             ------        --------
    224.0.0.1         0000000001    EXCLUDE
      SRCADDR: NONE
  INTERFACE STATISTICS:
    BYTESIN                           = 0
    INBOUND PACKETS                   = 0
    INBOUND PACKETS IN ERROR          = 0
    INBOUND PACKETS DISCARDED         = 0
    INBOUND PACKETS WITH NO PROTOCOL  = 0
    BYTESOUT                          = 0
    OUTBOUND PACKETS                  = 0
    OUTBOUND PACKETS IN ERROR         = 0
    OUTBOUND PACKETS DISCARDED        = 0
  **ASSOCIATED IQDX INTERFACE: EZAIQX18  IQDX STATUS: READY**
    BYTESIN                           = 0
    INBOUND PACKETS                   = 0
    BYTESOUT                          = 0
    OUTBOUND PACKETS                  = 0
INTFNAME: EZAIQX18          INTFTYPE: IPAQIQDX   INTFSTATUS: READY
    DATAPATH: 7803          DATAPATHSTATUS: READY
    VMACADDR: 027482110003
    READSTORAGE: MAX (2048K)
    IQDMULTIWRITE: DISABLED
  MULTICAST SPECIFIC:
    MULTICAST CAPABILITY: NO
  INTERFACE STATISTICS:
    BYTESIN                           = 0
    INBOUND PACKETS                   = 0
    INBOUND PACKETS IN ERROR          = 0
    INBOUND PACKETS DISCARDED         = 0
    INBOUND PACKETS WITH NO PROTOCOL  = 0
    BYTESOUT                          = 96
    OUTBOUND PACKETS                  = 1
    OUTBOUND PACKETS IN ERROR         = 0
    OUTBOUND PACKETS DISCARDED        = 0
F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=IFIND     F6=BOOK
F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
 . . . . . . . . . . . . . . . . . . . . . . . . . . .
.
```

4. Example 9-3 shows the output of the **netstat** command after several pings from SC31 to SC30 over IEDN (**ping 192.168.55.1**).

Example 9-3 The tcpe netstat dev command output

```
IntfName: OSX1111          IntfType: IPAQENET   IntfStatus: Ready
     PortName: IUTXP018  Datapath: 2303     DatapathStatus: Ready
     CHPIDType: OSX      CHPID: 18
     PNetID: IEDN
     Speed: 0000010000
     IpBroadcastCapability: No
     VMACAddr: 02279200002D   VMACOrigin: OSA    VMACRouter: All
     ArpOffload: Yes                 ArpOffloadInfo: Yes
     CfgMtu: 8992                    ActMtu: 8992
     IpAddr: 192.168.55.1/24
     VLANid: 55                      VLANpriority: Disabled
     DynVLANRegCfg: No               DynVLANRegCap: Yes
     ReadStorage: GLOBAL (4096K)
     InbPerf: Dynamic
       WorkloadQueueing: No
     ChecksumOffload: Yes            SegmentationOffload: No
     SecClass: 255                   MonSysplex: No
     Isolate: No                     OptLatencyMode: No
   Multicast Specific:
     Multicast Capability: Yes
     Group            RefCnt        SrcFltMd
     -----            ------        --------
     224.0.0.1        0000000001    Exclude
 ***
       SrcAddr: None
   Interface Statistics:
     BytesIn                          = 316
     Inbound Packets                  = 1
     Inbound Packets In Error         = 0
     Inbound Packets Discarded        = 0
     Inbound Packets With No Protocol  = 0
     BytesOut                         = 0
     Outbound Packets                 = 0
     Outbound Packets In Error        = 0
     Outbound Packets Discarded       = 0
   Associated IQDX interface: EZAIQX18  IQDX Status: Ready
     BytesIn                          = 0
     Inbound Packets                  = 0
     BytesOut                         = 334
     Outbound Packets                 = 1

 IntfName: EZAIQX18          IntfType: IPAQIQDX   IntfStatus: Ready
     Datapath: 7803          DatapathStatus: Ready
     VMACAddr: 027482110003
     ReadStorage: MAX (2048K)
     IQDMultiWrite: Disabled
   Multicast Specific:
     Multicast Capability: No
   Interface Statistics:
     BytesIn                          = 384
     Inbound Packets                  = 4
```

```
    Inbound Packets In Error          = 0
    Inbound Packets Discarded         = 0
    Inbound Packets With No Protocol  = 0
    BytesOut                          = 718
    Outbound Packets                  = 5
    Outbound Packets In Error         = 0
    Outbound Packets Discarded        = 0
```

9.8 IBM z/VM Virtual Switch for IEDN

IBM z/VM V6.2 is a hypervisor that is managed by the URM. Apart from defining z/VM guests as virtual servers, URM can also define z/VM virtual switches as part of the IEDN. The purpose of defining the switches is either to connect z/VM guests that are directly attached to the Virtual Switch and OSX channels as uplink ports, or to bridge OSX and IQDX and therefore connect virtual servers that have an IQDX QEBSM interface the IEDN, or both.

As of the writing of this book, this type of IEDN connection only applies to Linux guests, because z/OS has no QEBSM support, and z/VSE has no support for HiperSockets layer 2 interfaces.

For this book, we did not have a z/VM 6.2 available, so no example of an IEDN Virtual Switch is included. See *Building an Ensemble Using IBM zEnterprise Unified Resource Manager*, SG24-7921 for an example of how to set up an IEDN z/VM Virtual Switch. See 8.4, “The z/VM Virtual Switch with HiperSockets bridge port” on page 112 for an example of how to configure a HiperSockets Bridge Port for a z/VM Virtual Switch.

9.9 Network monitoring with Unified Resource Manager

The IEDN in the ensemble is a layer 2 data network and is managed by URM. It includes VLANs, virtual switches, and vNICs. With the URM network monitoring function, you can get IEDN network performance data from a layer 2 network perspective. The URM monitors the IEDN network resources, collects metrics, and presents them through the following external interfaces:

- Network Monitors Dashboard
- URM external APIs

URM monitors and collects network resource metrics at the following network interface levels:

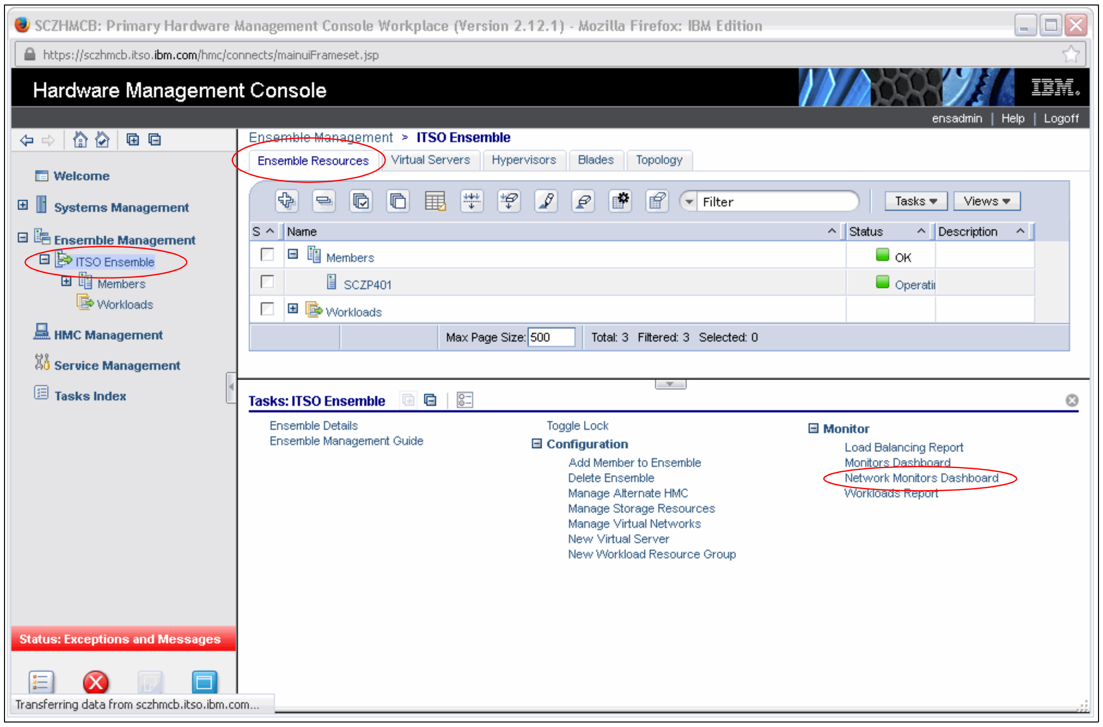
- Virtual network interfaces (by VLAN) between virtual servers and virtual switches
- Physical network interfaces between virtual switches and physical NICs

The network monitoring is performed on all of the hypervisors in the ensemble, and collected at fixed intervals. The following metrics are collected:

- Bytes sent/received, packets sent/received
- Packets dropped, packets discarded

Figure 9-18 shows the ensemble management view of the URM on your HMC. To set up monitoring, follow these steps:

1. Log on to your HMC with a user ID that has the correct authority to perform ensemble tasks, and select an ensemble. This opens a menu item **Manage Monitors Dashboard** in the **Monitor** menu of your ensemble.



SCZHMCB: Primary Hardware Management Console Workplace (Version 2.12.1) - Mozilla Firefox: IBM Edition

https://sczhmcb.itso.ibm.com/hmc/connects/mainuiFrameset.jsp

Hardware Management Console

ensadmin | Help | Logoff

Ensemble Management > ITSO Ensemble

S	Name	Status	Description
☐	Members	🟩 OK	
☐	SCZP401	🟩 Operati	
☐	Workloads		

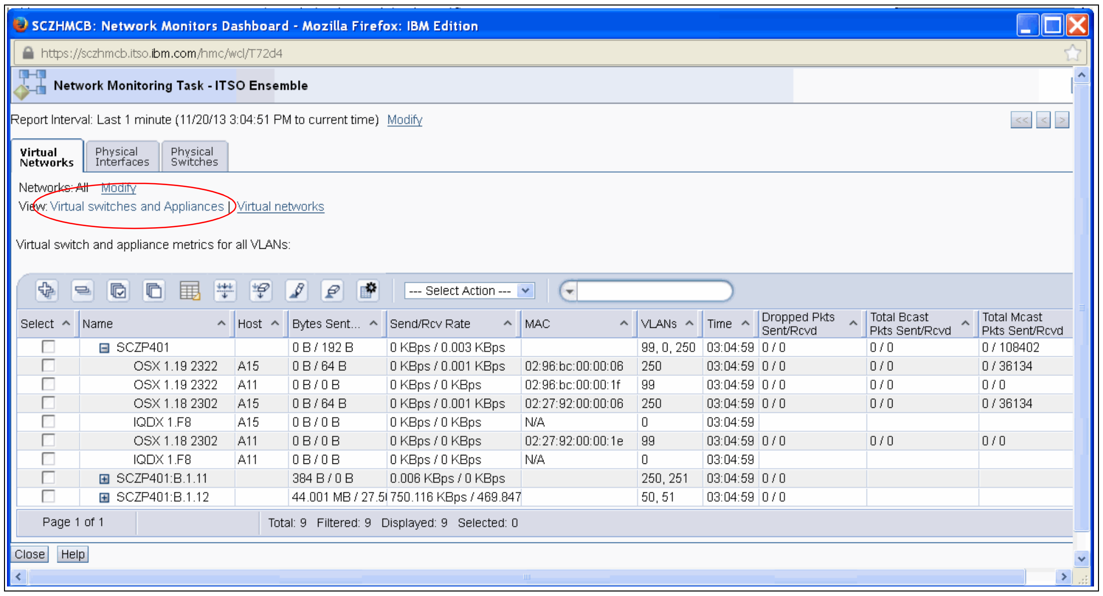
Max Page Size: 500   Total: 3   Filtered: 3   Selected: 0

Tasks: ITSO Ensemble

Transferring data from sczhmcb.itso.ibm.com...

Figure 9-18   Unified Resource Manager—Ensemble view

2. Click **Network Monitors Dashboard** to get to the Network Monitoring window shown in Figure 9-19. To view the metrics by vNIC, select **Virtual switches and Appliances.**



SCZHMCB: Network Monitors Dashboard - Mozilla Firefox: IBM Edition

https://sczhmcb.itso.ibm.com/hmc/wcl/T72d4

Network Monitoring Task - ITSO Ensemble

Report Interval: Last 1 minute (11/20/13 3:04:51 PM to current time) Modify

Networks: All   Modify

View: Virtual switches and Appliances | Virtual networks

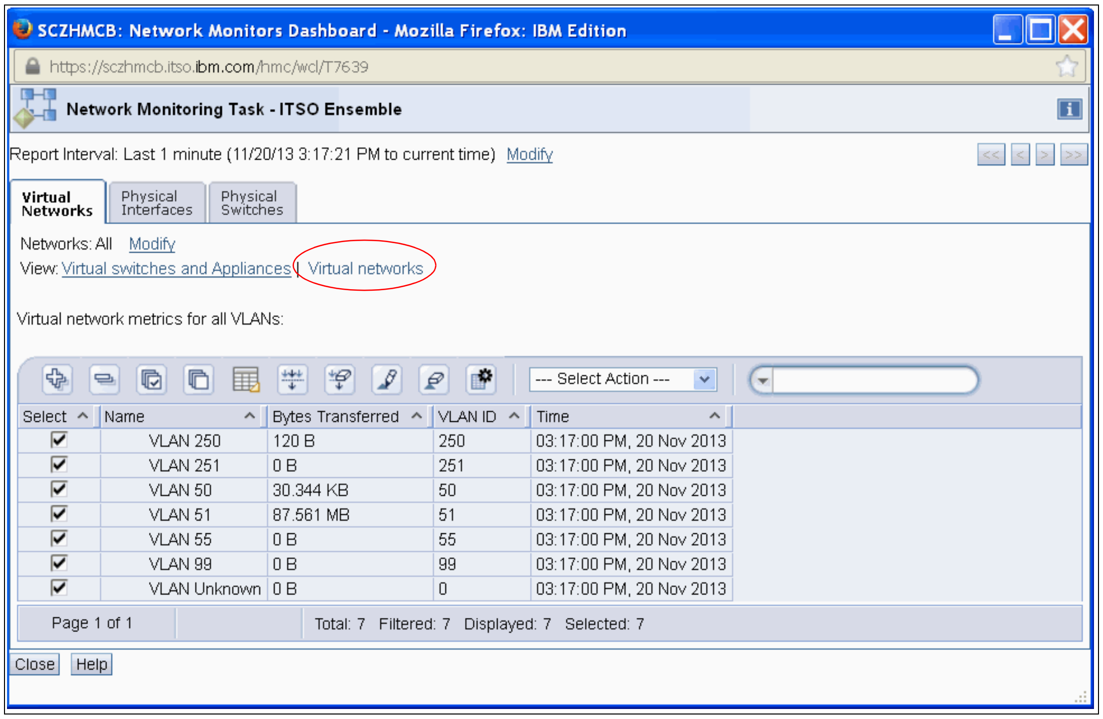
Virtual switch and appliance metrics for all VLANs:

Select	Name	Host	Bytes Sent...	Send/Rcv Rate	MAC	VLANs	Time	Dropped Pkts Sent/Rcvd	Total Bcast Pkts Sent/Rcvd	Total Mcast Pkts Sent/Rcvd
☐	SCZP401		0 B / 192 B	0 KBps / 0.003 KBps		99, 0, 250	03:04:59	0 / 0	0 / 0	0 / 108402
☐	OSX 1.19 2322	A15	0 B / 64 B	0 KBps / 0.001 KBps	02:96:bc:00:00:06	250	03:04:59	0 / 0	0 / 0	0 / 36134
☐	OSX 1.19 2322	A11	0 B / 0 B	0 KBps / 0 KBps	02:96:bc:00:00:1f	99	03:04:59	0 / 0	0 / 0	0 / 0
☐	OSX 1.18 2302	A15	0 B / 64 B	0 KBps / 0.001 KBps	02:27:92:00:00:06	250	03:04:59	0 / 0	0 / 0	0 / 36134
☐	IQDX 1.F8	A15	0 B / 0 B	0 KBps / 0 KBps	N/A	0	03:04:59			
☐	OSX 1.18 2302	A11	0 B / 0 B	0 KBps / 0 KBps	02:27:92:00:00:1e	99	03:04:59	0 / 0	0 / 0	0 / 0
☐	IQDX 1.F8	A11	0 B / 0 B	0 KBps / 0 KBps	N/A	0	03:04:59			
☐	SCZP401:B.1.11		384 B / 0 B	0.006 KBps / 0 KBps		250, 251	03:04:59	0 / 0		
☐	SCZP401:B.1.12		44.001 MB / 27.5	750.116 KBps / 469.847		50, 51	03:04:59	0 / 0		

Page 1 of 1   Total: 9   Filtered: 9   Displayed: 9   Selected: 0

Figure 9-19   Network Monitoring — Virtual Switches

3. To view the metrics by VLAN ID, select **Virtual networks** as shown in Figure 9-20.


SCZHMCB: Network Monitors Dashboard - Mozilla Firefox: IBM Edition
https://sczhmcb.itso.ibm.com/hmc/wcl/T7639
Network Monitoring Task - ITSO Ensemble
Virtual Networks
Physical Interfaces
Physical Switches
--- Select Action ---
Close
Help

Report Interval: Last 1 minute (11/20/13 3:17:21 PM to current time) Modify

Networks: All Modify

View: Virtual switches and Appliances | Virtual networks

Virtual network metrics for all VLANs:

Select	Name	Bytes Transferred	VLAN ID	Time
✓	VLAN 250	120 B	250	03:17:00 PM, 20 Nov 2013
✓	VLAN 251	0 B	251	03:17:00 PM, 20 Nov 2013
✓	VLAN 50	30.344 KB	50	03:17:00 PM, 20 Nov 2013
✓	VLAN 51	87.561 MB	51	03:17:00 PM, 20 Nov 2013
✓	VLAN 55	0 B	55	03:17:00 PM, 20 Nov 2013
✓	VLAN 99	0 B	99	03:17:00 PM, 20 Nov 2013
✓	VLAN Unknown	0 B	0	03:17:00 PM, 20 Nov 2013

Page 1 of 1 Total: 7 Filtered: 7 Displayed: 7 Selected: 7

Figure 9-20 Network Monitoring - Virtual networks

A

Gathering statistics in a HiperSockets environment

IBM z/OS Resource Management Facility (IBM RMF™) provides a significant amount of information for central processor (CP) usage purposes, including on the processor, input/output (I/O), and memory. It correlates resources with products such as IBM Lotus® Domino®, IBM WebSphere®, and so on.

Questions about Virtual Storage Access Method record-level sharing (VSAM RLS), Lotus Domino, Hypertext Transfer Protocol (HTTP), UNIX System Services, Fibre Channel (FC) activity, IBM System z HiperSockets, and coupling facility (CF) activity are frequently addressed by using RMF data.

RMF has been integrated with several products and z/OS features. The traditional RMF reporting is complemented by powerful workstation features that provide a state-of-the-art graphical user interface (GUI). IBM Tivoli® integration is important, because it enables clients to build an enterprise-class performance monitoring infrastructure.

This appendix describes the purpose of the report, the various fields of the report, and how to use RMF for generating a report.

The Resource Measurement facility (RMF)

RMF gathers data using three monitors:

- Short-term data collection with Monitor III
- Snapshot monitoring with Monitor II
- Long-term data gathering with Monitor I and Monitor III

The system operator starts all monitors as non-interactive (background) sessions with a variety of options that determine what type of data is collected and where it is stored.

Short-term data collection with Monitor III

A typical Monitor III gatherer session has a gathering cycle of one second, and consolidated records are written for a range that is typically set to 100 seconds. You can collect short-term data and continuously monitor the system status to solve performance problems using Monitor III reports. You get actual performance data, such as response times and execution velocity, on a detailed level for comparison with goals defined in your service policy.

The Monitor III Channel Path Activity Report

The Channel Path Activity report (CHANNEL) gives you information about channel path activity for all channel paths in the system. The report contains data for every channel path that is online during data gathering. The report identifies each channel path by identifier and channel path type. It reports both the total channel usage by the central processing complex (CPC) and the channel usage of the individual system image or logical partition (LPAR).

How to generate the report

The report displayed in Example A-1 was created using the RMF panels in z/OS Interactive System Productivity Facility (ISPF). To reach those panels, follow these steps:

Note: Whenever three asterisks (***) are displayed, press **Enter**.

1. From the Master Application menu, select **PDF**.
2. From the ISPF Primary Option menu, select **12** (z/OS System).
3. From the z/OS System Programmer Primary Option menu, select **9** (RMF).
4. From RMF - Performance Management, select **3 (**Monitor III).
5. From the RMF Monitor III Primary menu, select **3 (**RESOURCE).
6. From the RMF Resource Report Selection menu, select I/O Subsystem option **12** (Channel path activity).

Example A-1 shows the output of a Channel Path Activity report.

Example A-1 Channel Path Activity report

```
MF V2R1   Channel Path Activity          Line 1 of 19
Command ===>                                                   Scroll ===> CSR

Samples: 100     System: SC30  Date: 11/18/13  Time: 11.01.40  Range: 100   Sec

 Channel Path       Utilization(%)   Read(B/s) Write(B/s)  FICON OPS  zHPF OPS
ID No  G  Type  S   Part  Tot  Bus   Part  Tot  Part  Tot  Rate Actv  Rate Actv

00        OSD   Y    0.0  0.0  0.0     60   2K     0   20
```

```
0B        OSM   Y    0.0  0.0  0.0      0  138     0    0
19        OSX   Y    0.0  0.0  0.0      0    0     0    0
40    13  FC_S  Y    0.0  0.7  0.1    33K 743K    4K 116K    53    1     1    2
41    13  FC_S  Y    0.0  0.6  0.0    26K 703K    4K  90K    54    1     1    2
42    13  FC_S  Y    0.0  0.7  0.1    33K 731K    3K 116K    52    1     1    2
43    13  FC_S  Y    0.0  0.7  0.1    29K 728K    3K  91K    54    1     1    2
44    13  FC_S  Y    0.0  0.3  0.0    24K 273K    5K 117K    40    1     1    2
C8        ICP   Y   ---- ----
C9        ICP   Y   ---- ----
CA        ICP   Y   ---- ----
CB        ICP   Y   ---- ----
F0        IQD   Y                                  0    0
F1        IQD   Y                                  0    0
F2        IQD   Y                                  0    0
F3        IQD   Y                                 40    0
F4        IQD   Y                                  0    0
F8        IQD   Y                                  0    0
F9        IQD   Y                                  0    0
```

Monitor III utility fields

You can use the Monitor III utility to customize the CHANNEL report in a way that the following additional values are shown.

The following fields are used for HiperSockets:

WRITE (B/SEC)	PART. This is the data transfer rates from the channel to the control unit for this partition.
	TOTAL. This is the data transfer rates from the channel to the control unit for the CPC.
MESSAGE RATE	PART. This is the rate of messages sent by this partition.
	TOTAL. This is the rate of messages sent by the CPC.
MESSAGE SIZE	PART. This is the average size of messages sent by this partition.
	TOTAL. This is the average size of messages sent by the CPC.
SENT FAIL	PART. This is the rate of messages sent by this partition that failed.
RECEIVE FAIL	PART. This is the rate of messages (received by this partition) that failed.
	TOTAL. This is the rate of messages (received by the CPC) that failed.

References

For more information, see *z/OS Resource Measurement Facility Report Analysis,* SC33-7991.

B

IBM z/OS Sysplex subplexing and HiperSockets

IBM z/OS Communications Server enables you to subdivide a sysplex network into multiple *subplex* scopes from a sysplex networking function perspective. With subplexing, you are able to build security zones. Therefore, only members in the same security zone can communicate with each other.

This appendix describes how DYNAMICXCF, IBM System z HiperSockets, and virtual local area network (VLAN) technology can be used to provide such a subplex communication environment.

Sysplex subplexing

HiperSockets can also improve TCP/IP communications in a sysplex environment when the DYNAMICXCF facility is used. When a DYNAMICXCF HiperSockets device and link are activated, a subnetwork route is created across the HiperSockets link. The subnetwork is created by using the DYNAMICXCF IP address and mask. See 3.3, “DYNAMICXCF HiperSockets implementation” on page 36 for more information about how to set up DYNAMICXCF for HiperSockets.

IBM z/OS Communications Server enables you to subdivide a sysplex network into multiple subplex scopes from a sysplex networking function perspective. For example, some Virtual Telecommunications Access Method (VTAM) and Transmission Control Protocol/Internet Protocol (TCP/IP) instances in a sysplex might belong to one subplex, and other VTAM or TCP/IP instances in the same sysplex might belong to different subplexes.

Before subplexing, VTAM and TCP/IP Sysplex functions were deployed Sysplex-wide, and users had to implement complex resource controls and disable many of the dynamic XCF and routing functions to support multiple security zones. For example, as shown in Figure B-1 on page 153, TCP/IP stacks access different networks with diverse security requirements in the same Sysplex:

- In the top configuration, two TCP/IP stacks in the left LPARs access an internal network, and the TCP/IP stacks in the right two LPARs access the external network. Presumably, the security requirements include isolating external traffic from the internal network. However, all of the TCP/IP stacks in the Sysplex can dynamically establish connectivity with all of the other TCP/IP stacks in the Sysplex.
- In the bottom configuration, TCP/IP stacks in the same LPAR have different security requirements. The first stack in each LPAR connects to the internal network, and the second stack connects to the external network. Through the IUTSAMEH connection, the two stacks in each LPAR can dynamically establish connectivity with each other, possibly violating security policies.

Figure B-1 shows examples of Sysplex connectivity with different security requirements.

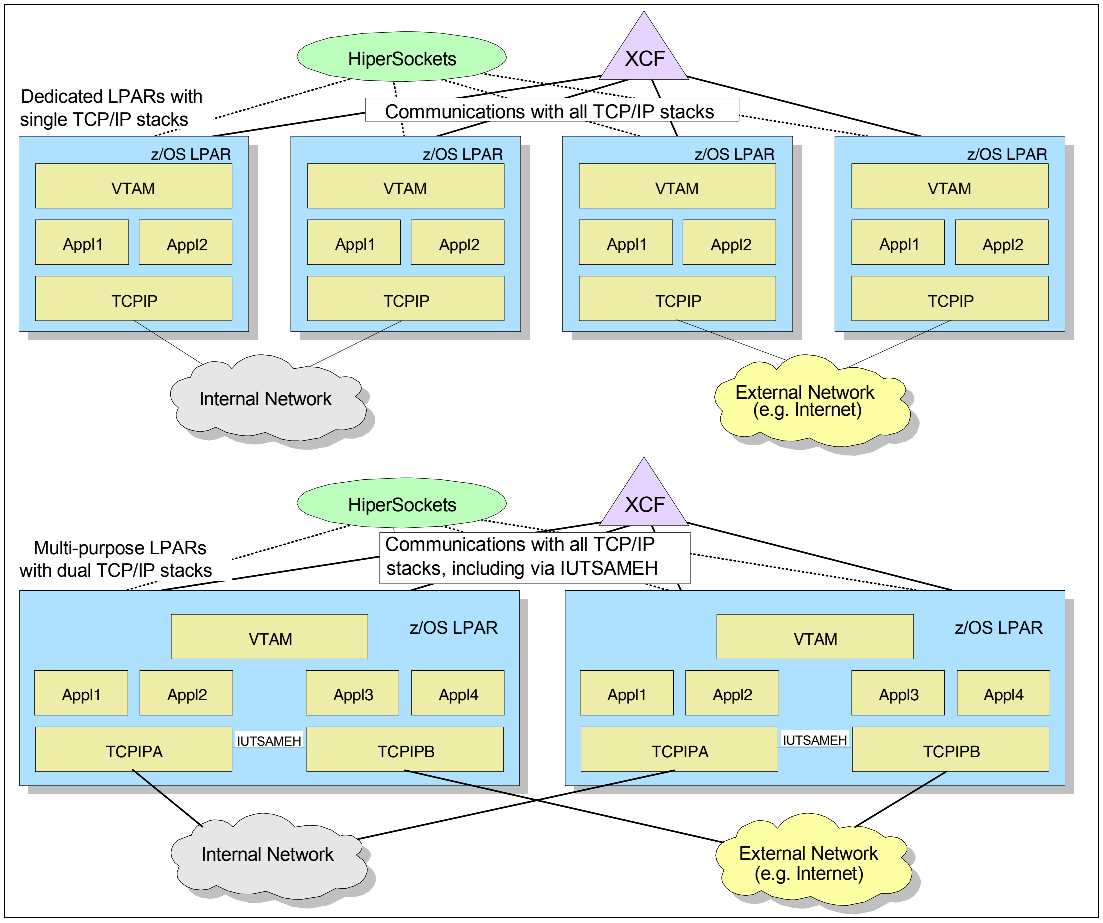

HiperSockets
XCF
Dedicated LPARs with single TCP/IP stacks
Communications with all TCP/IP stacks
z/OS LPAR
VTAM
Appl1
Appl2
TCPIP
Internal Network
External Network (e.g. Internet)
HiperSockets
XCF
Multi-purpose LPARs with dual TCP/IP stacks
Communications with all TCP/IP stacks, including via IUTSAMEH
z/OS LPAR
VTAM
Appl1
Appl2
Appl3
Appl4
TCPIPA
IUTSAMEH
TCPIPB
Internal Network
External Network (e.g. Internet)

Figure B-1 Sysplex connectivity - examples

With subplexing, you are able to build security zones. Only members in the same security zone can communicate with each other. Subplex members are VTAM nodes and TCP/IP stacks that are grouped in security zones to isolate communication.

Concept of subplexing

As cited, a subplex is a subset of a Sysplex that consists of selected members. Those members are connected and communicate through dynamic XCF groups to each other, using the following methods:

- XCF links (for cross-system IP and VTAM connections)
- IUTSAMEH (for IP connections in an LPAR)
- HiperSockets (IP connections cross-LPAR in the same server)

Subplexes do not communicate with members outside the subset of the Sysplex. For example, in Figure B-2, TCP/IP stacks with connectivity to the internal network can be isolated from TCP/IP stacks connected to external network using subplexing.

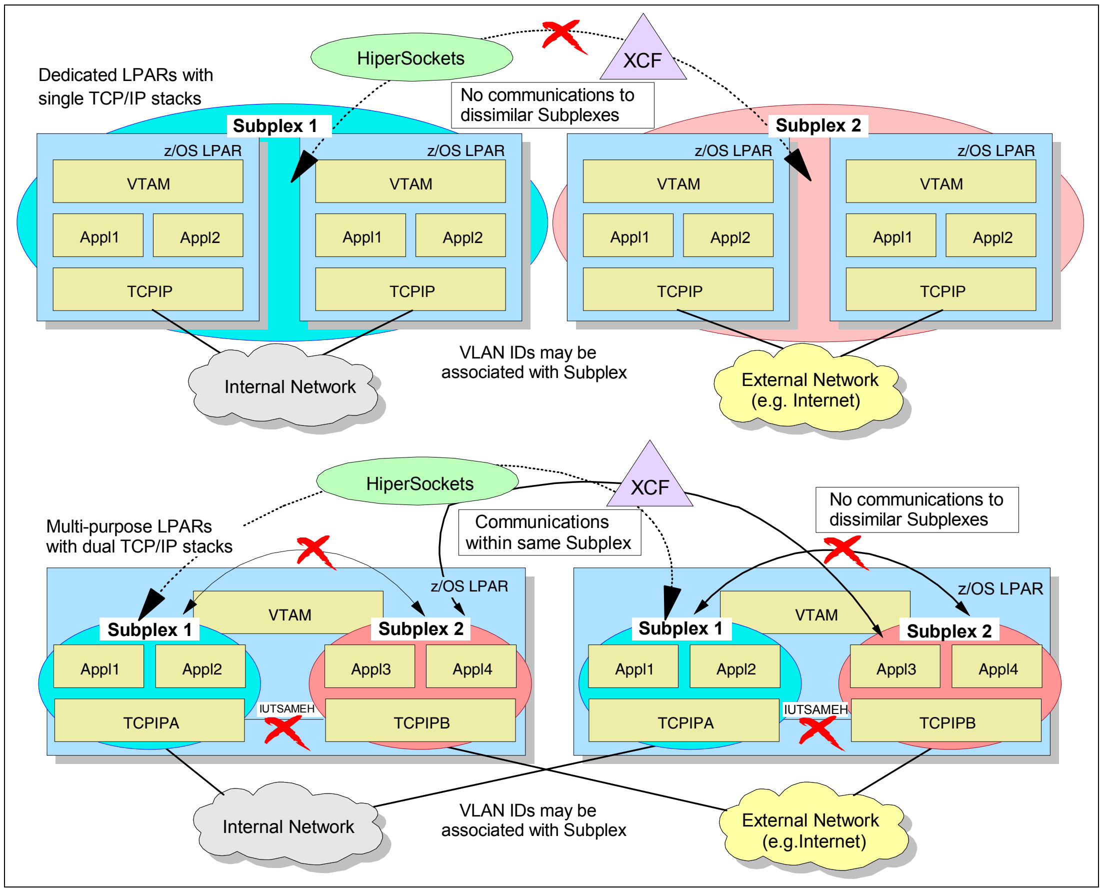

HiperSockets
XCF
Dedicated LPARs with single TCP/IP stacks
No communications to dissimilar Subplexes
Subplex 1
Subplex 2
z/OS LPAR
VTAM
Appl1
Appl2
TCPIP
VLAN IDs may be associated with Subplex
Internal Network
External Network (e.g. Internet)
HiperSockets
XCF
Multi-purpose LPARs with dual TCP/IP stacks
Communications within same Subplex
No communications to dissimilar Subplexes
z/OS LPAR
VTAM
Subplex 1
Subplex 2
Appl1
Appl2
Appl3
Appl4
TCPIPA
IUTSAMEH
TCPIPB
Internal Network
VLAN IDs may be associated with Subplex
External Network (e.g.Internet)

Figure B-2 Subplexing multiple security zones

TCP/IP stacks are defined as members of a subplex group with a defined group ID. For example, in Figure B-2, TCP/IP stacks in subplex 1 are able to communicate only with stacks in the same subplex group. They are not able to communicate with stacks in subplex 2.

In an environment where a single LPAR has access to internal and external networks over two TCP/IP stacks, those stacks are assigned to two different subplex group IDs. Even though IUTSAMEH is the communication method, it is controlled automatically through the association of subplex group IDs, therefore creating two separate security zones in the LPAR.

Suggestion: Network connectivity provided through an OSA port in a multiple security zone environment must not be shared across different subplex groups. The OSA ports ought to be physically isolated or logically separated using firewall and VLAN technologies.

Subplex implementation environment

Follow these steps to configure the test environment to enable IP subplexing using DYNAMICXCF and HiperSockets, as shown in Figure B-3:

1. Configure stacks TCPIPD on LPAR A23 and TCPIPD LPAR A24 to be part of subplex 11.
2. Configure stacks TCPIPE on LPAR A24 and LPAR A25 to be part of subplex 22.

To implement subplexing, Configure the environment according to the following guidelines:

- Because a subplex uses DYNAMICXCF, the VTAM **`IQDCHPID`** and **`XCFINIT`** parameters must be specified.
- The **`XCFGRPID`** and **`IQDVLANID`** parameters of the GLOBALCONFIG statements must be specified for each TCP/IP stack participating in the subplex using the HiperSockets connection.
- All TCP/IP stacks in the same subplex must have the same TCP/IP **`XCFGRPID`**.
- TCP/IP stacks in the same subplex using the same HiperSockets connection must have the same VLANID to establish connectivity.
- When defining only a TCP/IP subplex, a default VTAM subplex is defined automatically.
- A TCP/IP subplex uses VTAM XCF support for DYANMICXCF connectivity, therefore, a TCP/IP stack cannot span different VTAM subplexes. In this environment, the automatically created VTAM subplex consists of all of the nodes in this sysplex, so the TCP/IP subplexes do not span different VTAM subplexes.

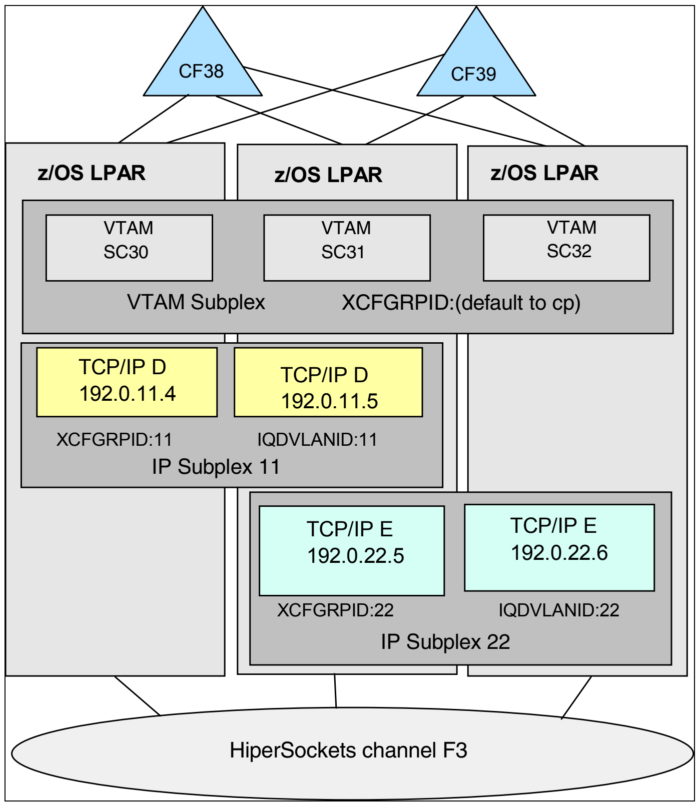


Figure B-3 Sysplex subplex

XCF group names

Basically, XCF group names for subplexes are created through the **XCFGRPID** parameter for the VTAM and TCP/IP environment:

- To define a VTAM subplex, use the **XCFGRPID** parameter in the VTAM start option. For detailed information about group and structure names, see *SNA Network Implementation Guide,* SC31-8777.
- To define a TCP/IP subplex, use the **XCFGRPID** parameter on the GLOBALCONFIG statement in the TCP/IP profile.

For TCP/IP, both the VTAM group ID suffix and the TCP group ID suffix will be used to build the TCP/IP group name. This group name is also used to join the Sysplex. Remember, when starting TCP/IP under Sysplex Autonomics control in previous z/OS releases, the stack joined the Sysplex group with the name EZBTCPCS. You can verify this using the **D XCF,GROUP** command.

EZBTCPCS is the default TCP/IP group name. Actually, the format of this group name is EZBT*vvtt*. The last four characters have the following meanings:

- In this example, *vv* is a 2-digit VTAM group ID suffix, specified on the VTAM **XCFGRPID** start option. The default is CP if not specified.
- Likewise, *tt* is a 2-digit TCP group ID suffix, specified on the **XCFGRPID** parameter of the GLOBALCONFIG statement. The default is CS if not specified.

In this test case, **XCFGRPID 11** was defined for TCP/IP and an **XCFGRPID** was defined for VTAM. The result was an XCF group name of EZBTCP11.

You might recognize that both **XCFGRPID**s are important in creating the subplex group name. Be aware that changing the VTAM **XCFGRPID** will change the XCF group name for the TCP/IP stack. Therefore, the stack will no longer be a member of the previous TCP/IP subplex group.

For example, in this environment, no VTAM **XCFGRPID** was defined and **XCFGRPID 11** was specified for TCP/IP. Therefore, the XCF group name was dynamically built as EZBTCP11. If you add **XCFGRPID=02** to the VTAM start option, then the new XCF group name will be EZBT0211.

Although nothing has been changed in the TCP/IP profile definitions in this example, the TCP/IP stack with the new subplex group name is no longer a member of the previous subplex (EZBTCP11). Therefore, the TCP/IP stack will lose the connectivity to the subplex.

Implementation steps

Perform the following steps to implement Sysplex subplex over HiperSockets:

1. Define the HiperSockets channel, control unit, and device.
2. Configure VTAM to specify **XCFINIT=YES** and to specify the **IQDCHPID** parameter.
3. Add the DYNAMICXCF statement to the TCP/IP profile.
4. Add **XCFGRPID** and **IQDVLANID** to the GLOBALCONFIG statement in the tcp profile.
5. Start the TCP/IP stacks.

This section uses the same definitions shown in 3.3, “DYNAMICXCF HiperSockets implementation” on page 36. No additional changes were required. We verified that channel-path identifier (CHPID) F7 and addresses in the range EB00-EB1F were online.

VTAM configuration setup for Sysplex subplex

Subplexing requires VTAM to be configured to use XCF. **XCFINIT=YES** must be coded as a VTAM parameter 1. The HiperSockets channel used for the TCP/IP DYNAMICXCF connection must be specified in the **IQDCHPID** parameter. This example uses CHPID F7 2. Specify the following values in the VTAM start options member (ATCSTRxx) for each of the VTAM nodes, as shown in Example B-1.

Example B-1 VTAM configuration changes

```
IQDCHPID=F7,     2
XCFINIT=YES     1
```

The **IQDCHPID** value can be dynamically changed using the following command, where *xx* specifies the HiperSockets channel:

```
V NET,VTAMOPTS,IQDCHPID=xx
```

Enable the VTAM **XCFGRPID** to default for all VTAM nodes in the sysplex. The **XCFINIT=YES** statement causes the creation of a VTAM XCF group. Because you did not specify a VTAM **XCFGRPID**, the default name for the VTAM XCF group is ISTXCF. This XCF group is the default VTAM subplex. In this test environment, this default VTAM XCF group includes all VTAM nodes in the sysplex.

TCP/IP configuration setup for Sysplex subplex

Configure two IP subplexes in a single VTAM subplex:

- TCPIPD on LPAR A23 and TCPIPD on LPAR A24 are defined to subplex 11.
- TCPIPE on LPAR A24 and TCPIPE on LPAR A25 are defined to subplex 22.

For IP stacks to communicate, they must be in the same subplex, so the **XCFGRPID** must be the same. Additionally, because you are using HiperSockets as the link, the **IQDVLANID** must also be the same. To configure the subplex, follow these steps:

1. Update the tcp profile for TCPIPD on LPAR A23, as shown in Example B-2.
2. Add the **XCFGRPID** with a value of 11 and an **IQDVLANID** of 11 1. These two values do not have to match. **XCFGRPID** supports values from 2 to 31, and **IQDVLANID** supports values from 1 to 4096.
3. Because Sysplex subplexing requires that the TCP/IP stack be configured for DYNAMICXCF, add the **DYNAMICXCF** parameter to the IPCONFIG statement with a host address, subnet mask, and cost metric 2.

Example B-2 LPAR A23 TCPIPD tcp profile configuration

```
GLOBALCONFIG NOTCPIPSTATISTICS XCFGRPID 11 IQDVLANID 11 1
IPCONFIG DYNAMICXCF 192.0.11.4 255.255.255.0   1 2
```

4. Update TCPIPD on LPAR A24, as shown in Example B-3. Note that TCPIPD on LPAR A23 and TCPIPD on LPAR A24 have matching values for **XCFGRPID**, so these two stacks will be part of the same XCF group. The XCF created EZBTCP11. Because these two stacks also have matching **IQDVLANID** parameters, they will be able to establish communication over the HiperSockets channel defined by the VTAM IQDCHPID statement F7.

Example B-3 LPAR A24 TCPIPD tcp profile configuration

```
GLOBALCONFIG NOTCPIPSTATISTICS XCFGRPID 11 IQDVLANID 11
IPCONFIG DYNAMICXCF 192.0.11.5 255.255.255.0   1
```

5. TCPIPE on LPAR A24 and LPAR A25 will be defined as subplex 22. Update the tcp profile on LPAR A24 shown in Example B-4. Add the **XCFGRPID** and **IQDVLANID** parameters to the GLOBALCONFIG statement:
   a. Assign a value of 22 for XCFGRPID and a value of 22 for IQDVLANID 1.
   b. Also add the required **DYNAMICXCF** parameter to the IPCONFIG statement along with the host IP address, subnet mask, and cost metric 2.

Example B-4 LPAR A24 TCPIPE tcp profile configuration

```
GLOBALCONFIG NOTCPIPSTATISTICS XCFGRPID 22 IQDVLANID 22
IPCONFIG DYNAMICXCF 192.0.22.5 255.255.255.0   1 2
```

6. Update TCPIPE’s tcp profile on LPAR A25, as shown in Example B-5. Because TCPIPE on LPAR A24 and LPAR A25 have matching **XCFGRPID**s, they will join the same sysplex group. The XCF group created is EZBTCP22. Because these two stacks also have matching **IQDVLANID** parameters, they will be able to establish communication over the HiperSockets channel defined by the VTAM IQDCHPID statement, F7.

Example B-5 LPAR A25 TCPIPE tcp profile configuration

```
GLOBALCONFIG NOTCPIPSTATISTICS XCFGRPID 22 IQDVLANID 22
IPCONFIG DYNAMICXCF 192.0.22.6 255.255.255.0   1
```

Verification of the IP subplex over HiperSockets

This section provides information about the verification of the IP subplex over HiperSockets.

Verify the VTAM setup before starting the IP stack

Before starting a TCP/IP stack to use the DYNAMICXCF connection, verify that the VTAM configuration is active:

1. Use the display VTAM options command to make sure that IQDCHPID=F7 1 and XCFINIT=YES 2 were defined.
2. Issue the command to each VTAM node in the sysplex to verify that the required parameters were correct. Note that you did not specify a value for XCFGRPID 3. See Example B-6.

Example B-6 Vtamopts display (partial display)

```
-D NET,VTAMOPTS
 IST097I DISPLAY ACCEPTED

 IST1189I IQDCHPID = F7      1         IQDIOSTG = 7.8M(126 SBALS)
 IST1189I XCFGRPID = ***NA***  3        XCFINIT  = YES   2
```

TCP/IP startup

After the configuration changes are made and the VTAM configuration has been verified, the IP stacks can be started. The first IP stack in a subplex that starts will cause the creation of an XCF group.

From the SYSLOG message generated at the IP stacks initialization, you can verify that DYNAMICXCF is enabled 1, that IP stack has joined its sysplex group 2, and that the DYNAMICXCF HiperSockets IUTIQDIO device has successfully started 3. The SYSLOG messages when TCPIPD is started on LPAR A23 are shown in Example B-7.

Example B-7 TCP syslog messages

```
...
EZZ0624I DYNAMIC XCF DEFINITIONS ARE ENABLED 1
...
EZD1176I TCPIPD HAS SUCCESSFULLY JOINED THE TCP/IP SYSPLEX GROUP
EZBTCP11 2
...
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTIQDIO 3
...
```

Verify the DYNAMICXCF device and link

To verify the status of devices and links defined to the TCP/IP stack, follow these steps:

1. Use the **D TCPIP,procname,NETSTAT,DEVLINKS** command to request NETSTAT information. The Netstat device display shows the HiperSockets connection with VLANID 11 1, which was what was specified for TCPIPD's **IQDVLANID** parameter, as shown in Example B-8.

Example B-8 Netstats device display showing IUTIQDIO (HiperSockets) VLANID

```
-D TCPIP,TCPIPD,NETSTAT,DEV
 EZD0101I NETSTAT CS V1R8 TCPIPD 952
DEVNAME: IUTIQDIO          DEVTYPE: MPCIPA
   DEVSTATUS: READY
   LNKNAME: IQDIOLNKC0000B04  LNKTYPE: IPAQIDIO   LNKSTATUS: READY
     NETNUM: N/A  QUESIZE: N/A
     IPBROADCASTCAPABILITY: NO
     CFGROUTER: NON                  ACTROUTER: NON
     ARPOFFLOAD: YES                 ARPOFFLOADINFO: YES
     ACTMTU: 8192
     VLANID: 11     1                VLANPRIORITY: DISABLED
     DYNVLANREGCFG: NO               DYNVLANREGCAP: NO
     READSTORAGE: GLOBAL (2048K)
     SECCLASS: 255
   BSD ROUTING PARAMETERS:
     MTU SIZE: 8192             METRIC: 01
     DESTADDR: 0.0.0.0          SUBNETMASK: 255.255.255.0
   MULTICAST SPECIFIC:
     MULTICAST CAPABILITY: YES
     GROUP            REFCNT
     -----            ------
     224.0.0.1        0000000001
   LINK STATISTICS:
     BYTESIN                          = 316
     INBOUND PACKETS                  = 1
     INBOUND PACKETS IN ERROR         = 0
```

```
        INBOUND PACKETS DISCARDED        = 0
        INBOUND PACKETS WITH NO PROTOCOL  = 0
        BYTESOUT                          = 316
        OUTBOUND PACKETS                  = 1
        OUTBOUND PACKETS IN ERROR         = 0
        OUTBOUND PACKETS DISCARDED        = 0
```

Verify transport resource list element definitions

To verify the transport resource list element (TRLE) definitions, follow these steps:

1. Use the **D NET,TRL,TRLE=*trle_name*** command to verify that the dynamically created TRLE, IUTIQDIO, for the HiperSockets connection, was active 1, as shown in Example B-9.

Example B-9 IUTQDIO TRLE display

```
D NET,TRL,TRLE=IUTIQDIO
IST097I DISPLAY ACCEPTED
IST075I NAME = IUTIQDIO, TYPE = TRLE
IST1954I TRL MAJOR NODE = ISTTRL
IST486I STATUS= ACTIV, DESIRED STATE= ACTIV      1
IST087I TYPE = LEASED            , CONTROL = MPC , HPDT = YES
IST1715I MPCLEVEL = QDIO       MPCUSAGE = SHARE
IST1716I PORTNAME = IUTIQDF7   LINKNUM =   0   OSA CODE LEVEL = ...(
IST1577I HEADER SIZE = 4096 DATA SIZE = 16384 STORAGE = ***NA***
IST1221I WRITE DEV = EB01 STATUS = ACTIVE     STATE = ONLINE
IST1577I HEADER SIZE = 4092 DATA SIZE = 0 STORAGE = ***NA***
IST1221I READ  DEV = EB00 STATUS = ACTIVE     STATE = ONLINE
IST1221I DATA  DEV = EB02 STATUS = ACTIVE     STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1717I ULPID = TCPIPD
IST1815I IQDIO ROUTING DISABLED
IST1918I READ STORAGE = 2.0M(126 SBALS)
IST1757I PRIORITY1: UNCONGESTED PRIORITY2: UNCONGESTED
IST1757I PRIORITY3: UNCONGESTED PRIORITY4: UNCONGESTED
IST1801I UNITS OF WORK FOR NCB AT ADDRESS X'0EAEC010'
IST1802I P1 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST1802I P2 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST1802I P3 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST1802I P4 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST1221I DATA  DEV = EB03 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = EB04 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = EB05 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = EB06 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = EB07 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = EB08 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = EB09 STATUS = RESET      STATE = N/A
```

```
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST314I END
```

Verify XCF groups

To verify the XCF groups, follow these steps:

1. Verify that the expected XCF groups have been created by issuing the **D XCF,GROUP** command, as shown in Example B-10. Subplex 11 (TCPIPD on LPAR A23 and LPAR A24) specified **XCFGRPID** on the tcp profile GLOBALCONFIG statement. So you expected an XCF group name of EZBTCP11 (CP is the default if no VTAM **XCFGRPID** value is specified) 1. For subplex 22 (TCPIPE on LPAR A24 and LPAR A25), you expected a group name of EZBTCP22, because 22 was specified on the GLOBALCONFIG **XCFGRPID** parameter 2.

Example B-10 Display of XCF groups

```
D XCF,GROUP
IXC331I  16.40.41  DISPLAY XCF 917
   GROUPS(SIZE):   COFVLFNO(3)        DBCDU(3)           EZBTCPCS(6)
                   EZBTCP11(2)  1     EZBTCP22(2)   2    IDAVQUIO(3)
                   IGWXSGIS(6)        IOEZFS(3)          IRRXCF00(3)
                   ISTCFS01(3)        ISTXCF(3)          IXCLO00A(3)
                   IXCLO00B(3)        IXCLO006(3)        SYSATB01(2)
                   SYSATB02(2)        SYSATB03(2)        SYSBPX(3)
                   SYSCNZMG(3)        SYSDAE(4)          SYSENF(3)
                   SYSGRS(3)          SYSGRS2(1)         SYSIEFTS(3)
                   SYSIGW00(3)        SYSIGW01(3)        SYSIGW02(3)
                   SYSIGW03(3)        SYSIKJBC(3)        SYSIOS01(3)
                   SYSJES(3)          SYSMCS(8)          SYSMCS2(8)
                   SYSRMF(3)          SYSTTRC(3)         SYSWLM(3)
                SYSXCF(3)          WTSCPLX5(3)        WTSC77(1)
```

2. The **D XCF,GROUP,GROUPNAME** command will list the members of the XCF group, as shown for EZBTCP11 in Example B-11.

Example B-11 Display of XCF group EZBTCP11

```
D XCF,GROUP,EZBTCP11
IXC332I  16.44.32  DISPLAY XCF 930
 GROUP EZBTCP11:    SC30TCPIPD         SC31TCPIPD
```

3. Issue the **D XCF,GROUP,GROUPNAME** command to verify the member list of EZBTCP22, as shown in Example B-12.

Example B-12 Display of XCF group EZBTCP22

```
D XCF,GROUP,EZBTCP22
IXC332I  16.44.32  DISPLAY XCF 930
 GROUP EZBTCP22:    SC31TCPIPE            SC32TCPIPE
```

4. The **D TCPIP** command can also be issued to display the sysplex group that a specific IP stack belongs to, as shown in Example B-13.

Example B-13 Display of Sysplex group to which an IP stack belongs

```
-D TCPIP,TCPIPD,SYSPLEX,GROUP
 EZZ8270I SYSPLEX GROUP FOR TCPIPD   AT SC30     IS EZBTCP11
...
-RO SC31,D TCPIP,TCPIPD,SYSPLEX,GROUP
```

```
 EZZ8270I SYSPLEX GROUP FOR TCPIPD   AT SC31     IS EZBTCP11
...
-RO SC31,D TCPIP,TCPIPE,SYSPLEX,GROUP
 EZZ8270I SYSPLEX GROUP FOR TCPIPE   AT SC31     IS EZBTCP22
...
-RO SC32,D TCPIP,TCPIPE,SYSPLEX,GROUP
 EZZ8270I SYSPLEX GROUP FOR TCPIPE   AT SC32     IS EZBTCP22
```

Verify XCFGRPID and IQDVLANID

To verify the parameters, follow these steps:

1. The Netstat config display shows the **XCFGRPID** and **IQDVLANID** for stack D, as shown in Example B-14. Verify that the **IQDVLANID** and **XCFGRPID** values displayed match what was specified on the GLOBALCONFIG statement.

Example B-14 Netstat config display with XCFGRPID and IQDVLANID for TCPIPD

```
-D TCPIP,TCPIPD,NETSTAT,CONFIG
 EZD0101I NETSTAT CS V1R8 TCPIPD 946
GLOBAL CONFIGURATION INFORMATION:
 TCPIPSTATS: NO   ECSALIMIT: 0000000K  POOLLIMIT: 0000000K
 MLSCHKTERM: NO   XCFGRPID:  11  IQDVLANID:  11
 SEGOFFLOAD: YES  SYSPLEXWLMPOLL: 060
 SYSPLEX MONITOR:
```

2. The Netstat config display shows the **XCFGRPID** and **IQDVLANID** for stack E, as shown in Example B-15.

Example B-15 Netstat config display with XCFGRPID and IQDVLANID for TCPIPE

```
-RO SC31,D TCPIP,TCPIPE,NETSTAT,CONFIG
 EZD0101I NETSTAT CS V1R8 TCPIPE 940
GLOBAL CONFIGURATION INFORMATION:
 TCPIPSTATS: NO   ECSALIMIT: 0000000K  POOLLIMIT: 0000000K
 MLSCHKTERM: NO   XCFGRPID:  22  IQDVLANID:  22
 SEGOFFLOAD: YES  SYSPLEXWLMPOLL: 060
```

IP subplex connectivity test

Test connectivity between the IP stacks to verify that the stacks cannot communicate across subplex boundaries:

1. Attempt a **ping** command using TCPIPD on LPAR A23 to LPAR A24 (same subplex 11). The results are successful, as shown in Example B-16.

Example B-16 Ping attempt to same subplex (subplex 11)

```
===> ping 192.0.11.5 (tcp tcpipd)
CS V1R8: Pinging host 192.0.11.5
Ping #1 response took 0.000 seconds.
***
```

2. Attempt a **ping** command using TCPIPD on LPAR A23 to LPAR A24 (subplex 11 to subplex 22). This fails, as shown in Example B-17.

Example B-17 Ping attempt to another subplex (subplex 11 to subplex 22)

```
===> ping 192.0.22.5 (tcp tcpipd)
CS V1R8: Pinging host 192.0.22.5
```

```
   sendto(): EDC8130I Host cannot be reached.
   ***
```

Also verify that multiple stacks in the same z/OS LPAR but in different subplexes cannot use an IUTSAMEH device to communicate. In this example, A24 has two IP stacks, TCPIPD in subplex 11 and TCPIPE in subplex 22.

3. First, display the status of the IUTSAMEH TRLE, as shown in Example B-18. You see that stacks TCPIPE and TCPIPE are defined to this link.

Example B-18 Display of IUTSAMEH dynamic TRLE

```
RO SC31,D NET,TRL,TRLE=IUTSAMEH
IST097I DISPLAY ACCEPTED
IST075I NAME = IUTSAMEH, TYPE = TRLE
IST1954I TRL MAJOR NODE = ISTTRL
IST486I STATUS= ACTIV, DESIRED STATE= ACTIV
IST087I TYPE = LEASED             , CONTROL = MPC , HPDT = YES
IST1715I MPCLEVEL = HPDT       MPCUSAGE = SHARE
IST1717I ULPID = TCPIPE
IST1717I ULPID = TCPIPD
IST314I END
```

4. Attempt to ping across the stacks, which will use the IUTSAMEH rather than the IUTIQDIO (HiperSockets) link and fail, as shown in Example B-19.

Example B-19 Ping attempts between TCPIPD and TCPIPE on LPAR A24

```
   PING 192.0.22.5 (TCP TCPIPD)
   CS V1R8: Pinging host 192.0.22.5
   sendto(): EDC8130I Host cannot be reached.
   READY
   PING 192.0.11.5 (TCP TCPIPE)
   CS V1R8: Pinging host 192.0.11.5
   sendto(): EDC8130I Host cannot be reached.
   READY
```

References

- *IBM z/OS V2R1 Communications Server TCP/IP Implementation Volume 1: Base Functions, Connectivity, and Routing*, SG24-8096
- *IBM System z Connectivity Handbook,* SG24-5444
- *z/OS Communications Server, IP Configuration Guide,* SC31-8775
- *z/OS Communications Server, IP Configuration Reference,* SC31-8776
- *z/OS Communications Server, SNA Resource Definition Reference*, SC31-8778

These publications are available online from IBM:

http://www.ibm.com/servers/eserver/zseries/zos/bkserv

C

Useful commands

This appendix lists the commands that were used to set up and verify the various local area network (LAN) environments described in this IBM Redbooks publication.

IBM z/OS commands

The following z/OS commands were used in this publication. For a complete list and description of Transmission Control Protocol/Internet Protocol (TCP/IP) Console and Time Sharing Option (TSO) commands, see *IP Systems Administrator's Commands*, SC31-8781.

To enter the commands in Table C-1, follow these steps:

1. In the Master Application Menu (panel you see after logon) enter `SD` to go to the System Display and Search Facility.
2. Enter `ULOG` in the System Display and Search Facility.
3. Type a backslash (/) followed by one of the listed commands.

Table C-1 IBM z/OS TCP/IP operator commands

Command	Description
D U,,,*dddd*[a]**,***nnn*	Gives the status of the device or devices
D U,,ALLOC,*dddd***,***nnn*[b]	Shows to whom the device or devices are allocated
D M=DEV(*dddd***)**	Gives the status of the paths defined to a device
D M=CHP	Gives the status and type of all CHPIDs defined to the z/OS
D M=CHP(*cc*[c]**)**	Gives the status of the path to the defined devices
D A,L	List of the jobs running in the system
V *dddd-dddd***,ONLINE**	Varies a device or range of devices online
V *dddd-dddd***,OFFLINE**	Varies a device or range of devices offline
CF CHP(*cc***),ONLINE**	Configures a channel-path identifier (CHPID) online
CF CHP(*cc***),OFFLINE**	Configures a CHPID offline
D TCPIP	Lists TCP/IP stacks that have started since the last initial program load (IPL) and stack status
D TCPIP,*tcpproc***,NETSTAT,ARP**	Displays contents of Address Resolution Protocol (ARP) cache for the TCP/IP stack
D TCPIP,*tcpproc***,NETSTAT,DEV**	Status of a device or devices, or interface or interfaces, defined in TCP/IP stack profile
D TCPIP,*tcpproc***,NETSTAT,HOME**	Displays the home IP address or addresses defined in the TCP/IP stack profile
D TCPIP,*tcpproc***,NETSTAT,ND**	Displays the contents of the IPv6 neighbor cache
D TCPIP,*tcpproc***,NETSTAT,ROUTE**	Displays the routing information for the TCP/IP stack
D TCPIP,*tcpproc*,OSAINFO,INTFN=	Displays information for an active IPAQENET/IPAQENET6 interface.
V TCPIP,*tcpproc***,PURGECACHE,***linkname*	Purges ARP cache for the specified adapter (linkname or intfname from NETSTAT,DEV)
V TCPIP,*tcpproc***,START,***tcpipdev*	Starts a device or interface defined in a TCP/IP stack
V TCPIP,*tcpproc***,STOP,***tcpipdev*	Stops a device or interface defined in a TCP/IP stack

a. dddd indicates the device number.

b. nnn indicates the number of devices to be displayed.
c. cc indicates the CHPID number.

Table C-2 lists additional TCP/IP TSO commands.

Table C-2 TCP/IP TSO commands

Command	Description
`NETSTAT ?`	Displays Netstat options
`NETSTAT ARP ALL`	Displays ARP cache
`NETSTAT DEV`	Displays the TCP/IP devices and links
`NETSTAT HOME`	Displays the TCP/IP Home IP addresses
`NETSTAT GATE`	Displays the TCP/IP Gateway addresses
`PING` *ipaddress*	Performs one PING to specified address
TRACERTE *ipaddress*	Traces router hops to a specified address
OBEYFILE	Executes selected TCP/IP profile statements

For a complete list and description of Virtual Telecommunications Access Method (VTAM) commands, in addition to those shown in Table C-3, see *SNA Operation*, SC31-8779.

Table C-3 VTAM commands

Command	Description
`D NET,VTAMOPTS`	Displays the current VTAM start options
`F` *vtamname*`,VTAMOPTS,` *optionname*=*value*	Modifies the current VTAM options (vtamname is the STC name; optionname is from VTAMOPTS.)
`D NET,MAJNODES`	Displays the VTAM major nodes
`D NET,ID=`*mnodename*`,E`	Displays information about a specified ID (for example, a Line, PU, or LU)
`D NET,TRL`	Displays the list of TRLEs
`D NET,TRL,TRLE=`*trlename*	Displays the status of a TRLE
`V NET,ID=ISTTRL,ACT,UPDATE=ALL`	Deletes inactive TRLEs from the TRL list
`V NET,ID=`*mnodename*`,ACT`	Activates a major node
`V NET,ID=`*mnodename*`,INACT`	Deactivates a major node

Important: If your static transport resource list element (TRLE) definition is incorrect, remember that an active TRLE entry *can not* be deleted. In such cases, you can use these steps:

1. Vary activate the TRL node with a blank TRLE to cause the deletion of previous entries.
2. Code the correct TRL with correct TRLE entries and definition.
3. Vary activate this corrected TRL/TRLE node.

Editing network profiles in z/OS

For somebody not familiar with z/OS, its concepts, panels, and tools might be quite confusing. The following section describes step-by-step how to edit a network profile. If you are familiar with z/0S already, this information might be obvious to you.

Multiple TCPI/IP stacks

In z/OS you can define multiple TCP/IP processes that operate independently. These processes are called TCPIP stacks. You first have to determine which TCPIP stack you want to work with. In this example you choose TCPIPA.

Find the active profile

To find the active profile, follow these steps:

1. Log on. After logon you will see the Master Application Menu, as shown in Example C-1.

Example C-1 IBM z/OS Master Application Menu

```
Master Application Menu - SC30
 Opt => sd 1                                                        Sc => HALF

                                                             USERID - AWINTER
 Enter SESSION MANAGER Mode ===> NO    (YES or NO)           TIME   - 07:27

   AO  AOC      - Automated Operations Control/MVS Dialogs
   CN  CONS     - Console Display and Search Facility
   CP  CPSM     - CICSPlex SM
   EJ  EJES     - (E)JES from Phoenix Software
   HC  HCD      - Hardware Configuration Definition
   IH  IHV      - ESCON Manager
   IP  IPCS     - Interactive Problem Control Facility
   IS  ISMF     - Interactive Storage Management Facility
   LR  LOGREC   - Interactive LOGR (CF) LOGREC Viewer
   OL  OPERLOG  - Interactive OPERLOG (Syslog) Browser
   WLM   WLM    - WLM Administrative Dialog
   TWS  TWS     - Tivoli Workload Scheduler
   TWSA  TWS    - Tivoli Workload Scheduler (TWSA)
   P   PDF      - ISPF/Program Development Facility
   R   RACF     - Resource Access Control Facility
   SD  SDSF     - System Display and Search Facility
   SJ  J3SD     - System Display and Search Facility (JES3)
   SM  SMP/E    - SMP/E Dialogs
   PA  DB2PA    - DB2 Performance Analyzer 1.1.0
   PM  DB2PM    - DB2PM 6.1
   P7  DB2PM    - DB2PM 7.1
   PE  DB2PE    - DB2PE 1.1
   PES  DB2PES  - DB2PE 1.1  PERFORMANCE EXPERT SERVER ADMINISTRATOR
   Q7  QMF      - QMF 7.1 SSN DB2G
   8A  DB8A     - DB2 V8  SSN -DB8A beta !!
   DBCD  DBCD   - DB2 V8  GROUP DCD1, DCD2, DCD3
   QMF8A QMF V8 - QMF for DB8A
   8B  DB8B     - DB2 V8  SSN -DB8B QCC
   8E  DB8E     - DB2 V8  SSN -DB8E
```

```
  8F  D8FG     - DB2 V8  SSN -D8FG (D8F1, D8F2 data sharing)
  8Q  DB8Q     - DB2 V8  SSN -DB8Q
  QMF8Q QMF V8 - QMF for DB8Q
  8X  DB8X     - DB2 V8  SSN -DB8X
  9K  D9KG     - DB2 V9  SSN -D9KG (D9K1, D9K2 data sharing)

 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

2. 1 Enter option **sd** to go to the System Display and Search Facility shown in Example C-2. Note that upper or lower case does not matter in z/OS.

Example C-2 System Display and Search Facility

```
 Display  Filter  View  Print  Options  Search  Help

--------------------------------------------------------------------------------
 HQX7790 -----------------  SDSF PRIMARY OPTION MENU
--------------------------
 COMMAND INPUT ===> da 1                                       SCROLL ===> CSR

 DA    Active users                       INIT  Initiators
 I     Input queue                        PR    Printers
 O     Output queue                       PUN   Punches
 H     Held output queue                  RDR   Readers
 ST    Status of jobs                     LINE  Lines
                                          NODE  Nodes
 LOG   System log                         SO    Spool offload
 SR    System requests                    SP    Spool volumes
 MAS   Members in the MAS                 NS    Network servers
 JC    Job classes                        NC    Network connections
 SE    Scheduling environments
 RES   WLM resources                      RM    Resource monitor
 ENC   Enclaves                           CK    Health checker
 PS    Processes
                                          ULOG  User session log
 END   Exit SDSF

 Licensed Materials - Property of IBM

 5650-ZOS Copyright IBM Corp. 1981, 2013.
 US Government Users Restricted Rights - Use, duplication or
 disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
```

3. 1 Enter command **da** to display a list of all processes (the active users), as shown in Example C-3.

Example C-3 Display active users

```
Display  Filter  View  Print  Options  Search  Help
 -------------------------------------------------------------------------------
 SDSF DA SC30     SC30     PAG  0  CPU/L/Z   1/  1/  0  LINE 1-5 (5)
 COMMAND INPUT ===>                                           SCROLL ===> CSR
 NP   JOBNAME  StepName ProcStep JobID    Owner    C Pos DP Real Paging    SIO
      TCPIP    TCPIP    TCPIP    STC04733 TCPIP      NS  FE 7306   0.00   0.00
      TCPIPF   TCPIPF   TCPIPF   STC04697 TCPIP      NS  FE 7152   0.00   0.00
      TCPIPE   TCPIPE   TCPIPE   STC04854 TCPIP      NS  FE 6922   0.00   0.00
  s 1 TCPIPA   TCPIPA   TCPIPA   STC05215 TCPIP      NS  FE 6921   0.00   0.00
      TCPIPD   TCPIPD   TCPIPD   STC05202 TCPIP      NS  FE 6810   0.00   0.00
```

4. 1 Enter **s** in front of TCPIPA to select TCPI/IP stack A, as shown in Example C-4.

Example C-4 Output display for TCPIPA

```
Display  Filter  View  Print  Options  Search  Help

-------------------------------------------------------------------------------
 SDSF OUTPUT DISPLAY TCPIPA   STC05215  DSID     2 LINE 0      COLUMNS 02- 81
 COMMAND INPUT ===> f profile= 1                                SCROLL ===> CSR
********************************* TOP OF DATA
**********************************
                   J E S 2  J O B  L O G  --  S Y S T E M  S C 3 0  --  N O D E

07.16.46 STC05215 ---- TUESDAY,   26 NOV 2013 ----
07.16.46 STC05215  IEF695I START TCPIPA   WITH JOBNAME TCPIPA   IS ASSIGNED TO
U
07.16.46 STC05215  $HASP373 TCPIPA   STARTED
07.16.47 STC05215  IEE252I MEMBER CTIEZB00 FOUND IN SYS1.IBM.PARMLIB
07.16.47 STC05215  IEE252I MEMBER CTIIDS00 FOUND IN SYS1.IBM.PARMLIB
07.16.47 STC05215  IEE252I MEMBER CTINTA00 FOUND IN SYS1.PARMLIB
07.16.47 STC05215  EZZ7450I FFST SUBSYSTEM IS NOT INSTALLED
07.16.49 STC05215  EZZ0162I HOST NAME FOR TCPIPA IS WTSC30A
07.16.49 STC05215  EZZ0300I OPENED PROFILE FILE DD:PROFILE
07.16.49 STC05215  EZZ0309I PROFILE PROCESSING BEGINNING FOR DD:PROFILE
07.16.49 STC05215  EZZ0318I ZE WAS FOUND ON LINE 111 AND ROUTE OR ENDROUTES WAS
07.16.49 STC05215  EZZ0318I ZE WAS FOUND ON LINE 113 AND ROUTE OR ENDROUTES WAS
07.16.49 STC05215  EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE DD:PROFILE
07.16.49 STC05215  EZZ0641I IP FORWARDING NOFWDMULTIPATH SUPPORT IS ENABLED
07.16.49 STC05215  EZZ0623I PATH MTU DISCOVERY SUPPORT IS ENABLED
07.16.49 STC05215  EZZ0688I IQDIO ROUTING IS ENABLED
07.16.49 STC05215  EZZ0338I TCP PORTS 1 THRU 1023 ARE RESERVED
07.16.49 STC05215  EZZ0338I UDP PORTS 1 THRU 1023 ARE RESERVED
07.16.49 STC05215  EZZ0613I TCPIPSTATISTICS IS DISABLED
07.16.49 STC05215  EZZ4202I Z/OS UNIX - TCP/IP CONNECTION ESTABLISHED FOR
TCPIPA
07.16.49 STC05215  EZZ4340I INITIALIZATION COMPLETE FOR INTERFACE HIPERLFO
07.16.49 STC05215  EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA2160
07.16.49 STC05215  EZB6473I TCP/IP STACK FUNCTIONS INITIALIZATION COMPLETE.
07.16.49 STC05215  EZAIN11I ALL TCPIP SERVICES FOR PROC TCPIPA ARE AVAILABLE.
07.16.50 STC05215  EZZ4340I INITIALIZATION COMPLETE FOR INTERFACE EZ6OSM02
07.16.51 STC05215  EZD1176I TCPIPA HAS SUCCESSFULLY JOINED THE TCP/IP SYSPLEX
GR
07.16.51 STC05215  EZZ4340I INITIALIZATION COMPLETE FOR INTERFACE EZ6OSM01
        1 //TCPIPA   JOB MSGLEVEL=1
        2 //STARTING EXEC TCPIPA,PROFILE=PROFA30D 2
          XX****************************************
          XX* SYS1.PROCLIB(TCPIPA)
          XX****************************************
        3 XXTCPIPA    PROC PARMS='CTRACE(CTIEZB00),IDS=00',
          XX*            PROFILE=PROFA&SYSCLONE.,TCPDATA=DATAA&SYSCLONE
 F1=HELP      F2=SPLIT     F3=END        F4=RETURN    F5=IFIND      F6=BOOK
 F7=UP        F8=DOWN      F9=SWAP      F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

5. 1 Enter **`f profile=`** to find 2 in the output of TCPIPA. This tells you that the name of the currently active profile for TCPI PA is `PROFA30D`.

Edit a profile

To edit a profile, follow these steps:

1. In the Master Application Menu, as shown in Example C-1 on page 168, enter option **`p`** to go to the Interactive System Productivity Facility/Program Development Facility (ISPF/PDF) shown in Example C-5.

Example C-5 The ISPF/Program Development Facility

```
Menu  Utilities  Compilers  Options  Status  Help
 ..............................................................................
                            ISPF Primary Option Menu
 Option ===>3 1

 0  Settings      Terminal and user parameters           User ID . : AWINTER
 1  View          Display source data or listings        Time. . . : 08:16
 2  Edit          Create or change source data           Terminal. : 3278
 3  Utilities     Perform utility functions              Screen. . : 1
 4  Foreground    Interactive language processing        Language. : ENGLISH
 5  Batch         Submit job for language processing     Appl ID . : PDF
 6  Command       Enter TSO or Workstation commands      TSO logon : IKJACCT
 7  Dialog Test   Perform dialog testing                 TSO prefix: AWINTER
 9  IBM Products  IBM program development products       System ID : SC30
 10 SCLM          SW Configuration Library Manager       MVS acct. : ACCNT#
 11 Workplace     ISPF Object/Action Workplace           Release . : ISPF 7.1
 12 z/OS System   z/OS system programmer applications
 13 z/OS User     z/OS user applications

      Enter X to Terminate using log/list defaults
```

2. 1 Enter option **3** to go to the Utilities panel shown in Example C-6.

Example C-6 Utility selection panel

```
Menu  Help
 ..............................................................................
                            Utility Selection Panel
 Option ===> 4 1

 1  Library     Compress or print data set.  Print index listing.  Print,
                  rename, delete, browse, edit or view members
 2  Data Set    Allocate, rename, delete, catalog, uncatalog, or display
                  information of an entire data set
 3  Move/Copy   Move, or copy members or data sets
 4  Dslist      Print or display (to process) list of data set names.
                  Print or display VTOC information
 5  Reset       Reset statistics for members of ISPF library
 6  Hardcopy    Initiate hardcopy output
 7  Transfer    Download ISPF Client/Server or Transfer data set
 8  Outlist     Display, delete, or print held job output
 9  Commands    Create/change an application command table
 11 Format      Format definition for formatted data Edit/Browse
 12 SuperC      Compare data sets                          (Standard Dialog)
 13 SuperCE     Compare data sets Extended                 (Extended Dialog)
 14 Search-For  Search data sets for strings of data       (Standard Dialog)
 15 Search-ForE Search data sets for strings of data Extended (Extended Dialog)
 16 Tables      ISPF Table Utility
 17 Udlist      Print or display (to process) z/OS UNIX directory list
```

3. 1 Enter option **4** to display the data sets shown in Example C-7.

Example C-7 Data Set List Utility

```
Menu  RefList  RefMode  Utilities  Help
 ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
                            Data Set List Utility
 Option ===>

    blank Display data set list              P Print data set list
        V Display VTOC information          PV Print VTOC information

 Enter one or both of the parameters below:
    Dsname Level . . . TCPIPA.TCPPARMS 1
    Volume serial  . .

 Data set list options
    Initial View                 Enter "/" to select option
    1  1. Volume                 /  Confirm Data Set Delete
       2. Space                  /  Confirm Member Delete
       3. Attrib                 /  Include Additional Qualifiers
       4. Total                  /  Display Catalog Name
                                    Display Total Tracks
                                    Prefix Dsname Level

 When the data set list is displayed, enter either:
   "/" on the data set list command field for the command prompt pop-up,
   an ISPF line command, the name of a TSO command, CLIST, or REXX exec, or
   "=" to execute the previous command.
```

4. 1 Enter TCPIPA.TCPPARMS to search for the TCPIPA data set, as shown in Example C-8.

Example C-8 Search results for TCPIPA data set

```
Menu  Options  View  Utilities  Compilers  Help
 ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
 DSLIST - Data Sets Matching TCPIPA.TCPPARMS                         Row 1 of 3
 Command ===>                                                   Scroll ===> PAGE

 Command - Enter "/" to select action                  Message          Volume

-------------------------------------------------------------------------------
      e 1 TCPIPA.TCPPARMS                                                COMCAT
          TCPIPA.TCPPARMS.GESESC30                                       COMCAT
          TCPIPA.TCPPARMS.OLD                                            COMCAT
 ***************************** End of Data Set list
****************************
```

5. 1 Enter **e** in front of the TCPIPA.TCPPARMS data set to edit it, as shown in Example C-9.

Example C-9 Content of TCPIPA.TCPPARMS data set

```
Menu  Functions  Confirm  Utilities  Help
 ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
 EDIT            TCPIPA.TCPPARMS                          Row 0000084 of 0000161
 Command ===> l profa 1                                           Scroll ===>
PAGE
            Name     Prompt       Size   Created          Changed          ID
 _________ PAGENV30                 5  2007/09/12  2007/09/14 11:16:54  CS02
 _________ PROFAAAA
 _________ PROFAD                 270  2007/09/05  2007/09/05 14:03:33  CS02
 _________ PROFAGS                473  2013/07/18  2013/08/09 10:58:24  CS05
 _________ PROFAGS1                 2  2013/07/24  2013/07/24 15:28:46  CS05
 _________ PROFA030               270  2007/09/07  2007/09/07 11:44:03  CS06
 _________ PROFAS                 274  2007/09/04  2007/09/04 18:01:25  CS03
 _________ PROFASNS               273  2007/09/04  2007/09/05 13:30:20  CS04
 _________ PROFATST               280  2007/09/11  2007/09/11 10:00:04  CS03
 _________ PROFA30
 _________ PROFA30B               251  2006/08/02  2006/08/04 07:13:31  CS02
 _________ PROFA30C               254  2006/08/08  2006/08/21 14:24:19  CS01
 ___e 2___ PROFA30D                127  2013/11/21  2013/11/26 07:14:48
AWINTER
 _________ PROFA31                363  2008/10/28  2011/11/02 09:49:46  CS01
 _________ PROFA31B               257  2006/07/27  2006/08/04 07:11:12  CS02
 _________ PROFA31C               259  2003/04/03  2006/08/21 14:25:51  CS01
 _________ PROFA31D                125  2013/11/20  2013/11/27 09:02:20
AWINTER
 _________ PROFA31K                484  2013/11/21  2013/11/21 22:25:06
KURIADI
 _________ PROFA310                62  2006/10/05  2006/10/30 07:52:04  CS03
 _________ PROFA319               266  2009/08/26  2009/08/26 10:59:02  CS08
 _________ PROFA32                217  2007/08/31  2011/11/01 16:49:00  CS01
  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

6. 1 Enter the **`l profa`** command to locate files beginning with `profa`.
7. 2 Enter **e** in front of `PROFA30D` (the active profile) to edit it.

A few tips for using the editor

You can put the following letters into the line number field:

i	Insert a line.
r	Repeat a line.
c	Copy a line, then:
a	Insert the copied line after this one.
d	Delete a line.

Files often have a fixed line length. Therefore, if it is not possible to insert characters, delete some of the spaces at the end of the line.

Terminate and restart a TCP/IP stack

In order for a profile to become active, you need to terminate and restart the TCP/IP stack:

1. Go to the Display Active Users panel, as shown in Example C-2 on page 169. Example C-10 shows the list of active users. Note that TCPIPA is active.

Example C-10 Terminate a TCPIP stack

```
. . . . . . . . . . . . . . . . . . . . . . . . . . .
   Display  Filter  View  Print  Options  Search  Help

-------------------------------------------------------------------------------
 SDSF DA SC30     SC30     PAG  0  CPU/L/Z   2/  1/  0  LINE 1-5 (5)
 COMMAND INPUT ===> /p tcpipa 1                              SCROLL ===> CSR
 NP   JOBNAME  StepName ProcStep JobID    Owner    C Pos DP Real Paging    SIO
      TCPIP    TCPIP    TCPIP    STC04733 TCPIP      NS  FE 7306   0.00   0.00
      TCPIPF   TCPIPF   TCPIPF   STC04697 TCPIP      NS  FE 7154   0.00   0.00
      TCPIPE   TCPIPE   TCPIPE   STC04854 TCPIP      NS  FE 6922   0.00   0.00
      TCPIPA   TCPIPA   TCPIPA   STC05215 TCPIP      NS  FE 6921   0.00   0.00
      TCPIPD   TCPIPD   TCPIPD   STC05202 TCPIP      NS  FE 6814   0.00   0.00
```

2. 1 Enter the **`/p tcpipa`** command to terminate TCPIP stack A. Press Enter until it is gone from the list of active users, as shown in Example C-11.

Example C-11 Start a TCPIP stack

```
. . . . . . . . . . . . . . . . . . . . . . . . . . .
   Display  Filter  View  Print  Options  Search  Help

-------------------------------------------------------------------------------
 SDSF DA SC30     SC30     PAG  0  CPU/L/Z   2/  1/  0  LINE 1-4 (4)
 COMMAND INPUT ===> /s tcpipa,profile=profa30d 1             SCROLL ===> CSR
 NP   JOBNAME  StepName ProcStep JobID    Owner    C Pos DP Real Paging    SIO
      TCPIP    TCPIP    TCPIP    STC04733 TCPIP      NS  FE 7306   0.00   0.00
      TCPIPF   TCPIPF   TCPIPF   STC04697 TCPIP      NS  FE 7154   0.00   0.00
      TCPIPE   TCPIPE   TCPIPE   STC04854 TCPIP      NS  FE 6922   0.00   0.00
      TCPIPD   TCPIPD   TCPIPD   STC05202 TCPIP      NS  FE 6814   0.00   0.00
```

3. 1 Enter the **`/s tcpipa, profile=profa30d`** command to restart TCPIP stack A with profile `profa30d`. Press Enter until TCPIPA displays again in the list of active users.

Work with a TCP/IP stack

To work with a TCP/IP stack, follow these steps:

1. Go to the Program Development Facility, as shown in Example C-5 on page 171. Enter option **`6`** to go to the command panel, as shown in Example C-12.

Example C-12 Targeting TCPIPA

```
Menu  List  Mode  Functions  Utilities  Help
 ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
                              ISPF Command Shell
 Enter TSO or Workstation commands below:

 ===> tcpa 1
```

```
Place cursor on choice and press enter to Retrieve command

=> netstat route
=> ping 192.168.6.44
=> netstat home
=> ping 192.168.6.1
=> netstat dev
=> tcpa
```

2. **1** Enter `tcpa` first to indicate that the following commands target TCPIPA. Use the commands displayed in the list of commands in the previous example to display status or ping an IP address.

   Alternatively, you can use the **`SD > ULOG`** command line, as described in "IBM z/OS commands" on page 166.

Linux on System z commands

Table C-4 shows some useful commands for HiperSockets for Linux support.

Table C-4 Useful commands

Command	Description
`cat /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/if_name`	Displays the network interface name, for example, hsi1 for the device_bus_id 0.0.7000.
`echo <text> > <file>`	Writes text into a file.

`ifconfig`
`ifconfig <nw_if_name>`
`ifconfig <nw_if_name> down`
`ifconfig <nw_if_name> up`
`ifconfig <nw_if_name> <ip_adrs> netmask <subnet_mask>`
	Displays all network interfaces defined.
	Displays the given network interface.
	Brings down a network interface.
	Starts a network interface.
	Defines the network interface with the given IP address and subnet mask with default values.

`lscss`	Displays the channel subsystem.

`lsqeth`
 	
`lsqeth <nw_if_name>`
	Displays all qeth information.
	Displays the specific qeth information for given network interface name.

`readlink /sys/class/net/<nw_if_name>/device`	Displays the device_bus_id, for example, 0.0.7000, for the network interface name, hsi1.
`route`	Displays the current network routes.
`znetconf -u`	Displays a list of unconfigured network devices.
`znetconf -c`	Displays a list of configured network devices.
`znetconf -a <device_number>`	Adds a network device.
`znetconf -r <device_number>`	Removes a network device.

IBM z/VM commands

For information about z/VM TCP/IP operations commands, see *z/VM CP Command and Utility Reference*, SC24-6175. Also see Table C-5.

Table C-5 IBM z/VM TCP/IP operations commands

Command	Description
`Q OSA ACTIVE|ALL`	Displays the status of network devices, Open Systems Adapter (OSA) and internal queued direct communication (IQD)
`Q` *rdev|rdev-rdev*	Displays the status of real devices
`Q PATHS` *rdev|rdev-rdev*	Displays the path status to real devices (PIM, PAM, LPM)
`Q CHPID` *cc*	Displays the real CHPID status
`VARY OFF|ON` *rdev|rdev-rdev*	Varies devices off or online
`VARY OFF|ON CHPID` *cc*	Configures a CHPID off or on to both hardware and software

These commands can be used to create and verify virtual LAN (VLAN), network interface card (NIC), and COUPLED definitions for the HiperSockets emulation called z/VM guest LAN.

Table C-6 z/VM guest LAN commands

Command	Description
ATTACH <rdev> <vmid> <vdev>	Attaches real device address to the virtual machine (VM) ID at its virtual address
DEFINE LAN	Creates a VM LAN segment managed by the CP system
DEFINE NIC	Installs a virtual network adapter (a NIC) in the invoker's virtual machine configuration
DETACH LAN	Eliminates a VM LAN segment from the CP system
DETACH NIC	Removes a virtual network adapter (a Network Interface Card) in the invoker's virtual machine configuration
QUERY <dev_adrs>	Displays information about the device
QUERY CHPID <xx>	Displays information about CHPID *xx*
QUERY LAN	Displays information about the designated VM LAN (or every LAN in the System LAN Table)
QUERY NIC	Displays information about a virtual NIC (vNIC) in your virtual machine configuration
QUERY OSA	Displays information about the OSA defined to z/VM
QUERY VMLAN	Determines the status of VM LAN activity on the CP host
SET LAN	Modifies the attributes of a VM LAN
UNCOUPLE	Disconnects a virtual channel-to-channel adapter (CTCA) from a coupled CTCA device, or disconnects a virtual network adapter from a VM LAN segment

Table C-7 and Table C-8 on page 177 list z/VM commands for TCP/IP and Virtual Switches.

Table C-7 IBM z/VM TCP/IP commands

Command	Description
`NETSTAT ?`	Displays Netstat options
`NETSTAT ARP`	Displays the ARP cache
`NETSTAT DEV`	Displays the TCP/IP devices and links
`NETSTAT HOME`	Displays the TCP/IP Home IP addresses
`NETSTAT GATE`	Displays the TCP/IP Gateway addresses
`NETSTAT OBEY START|STOP DEV`	Starts or stops the device name identified in NETSTAT DEV output
`IFCONFIG (z/VM4.3)`	Displays the TCP/IP devices and links (similar to the NETSTAT DEV command, but has other uses; see note)
`PING` *ipaddress*	Performs one PING to a specified address
`TRACERTE` *ipaddress*	Traces router hops to a specified address
`OBEYFILE`	Executes selected TCP/IP profile statements
Note: IFCONFIG can help to temporarily modify network interfaces in the current TCP/IP stack. See *z/VM TCP/IP Planning and Customization*, SC24-6238, for detailed uses. For more information about the other z/VM TCP/IP commands, see *z/VM TCP/IP User's Guide*, SC24-6240.	

Table C-8 IBM z/VM Virtual Switch commands

Command	Description
`DEFINE VSWITCH`	Defines the virtual switch and attributes
`DEFINE NIC`	Defines the simulated NIC
`COUPLE`	Helps to connect the NIC to the virtual switch
`SET VSWITCH`	Controls the attributes of an existing virtual switch
`QUERY CONTROLLER`	Displays the controller service machines
`QUERY VSWITCH`	Displays information about the virtual switch
`QUERY VSWITCH DETAILS`	Displays detail information about the virtual switch
`QUERY VSWITCH` *name* `ACCESS`	Displays authorized user IDs
`QUERY VMLAN`	Displays system-wide MAC addresses

Defining and coupling a NIC using CP commands

Tip: You might choose to use the **`DEFINE NIC`** and **`COUPLE`** approach rather than the NICDEF z/VM user directory statement. In that case, consider adding those two commands into your guest's `PROFILE EXEC` file so that they are automatically executed at IPL, or whenever the guest logs on.

To create a virtual NIC, use the following command syntax:

```
DEFINE NIC vdev [ operands ]
```

In this syntax, **vdev** specifies the base virtual device address for the adapter, and **operands** defines the characteristics of the vNIC. Operands accepted by the **DEFINE NIC** command are listed in Table C-9.

Table C-9 Operands for the DEFINE NIC command

Operands	Description
TYPE	This operand specifies the type of NIC adapter to be created, specifically the hardware and protocol that the adapter is to emulate. This is an optional keyword that you can specify with HIPERsockets or queued direct input/output (QDIO).
HIPERsockets	This operand defines this adapter as a simulated HiperSockets NIC. This adapter functions similar to the HiperSockets internal adapter. A HiperSockets NIC can function without a z/VM Guest LAN connection or can be coupled to a HiperSockets Guest LAN.
QDIO	This operand defines this adapter as a simulated QDIO NIC. This adapter functions similar to the OSA-Express (QDIO) adapter. A QDIO NIC is only functional when it is coupled to a QDIO Guest LAN.
IEDN	This operand defines this adapter as a simulated intraensemble data network NIC. This adapter will function like an OSA-Express CHPID type OSX adapter (device model 1732-02) that is connected to an IEDN internal network that is managed by the Unified Resource Manager. An IEDN NIC is functional only when it is coupled to an IEDN virtual switch.
INMN	This operand defines this adapter as a simulated intranode management network NIC. This adapter will function like an OSA-Express CHPID type OSM adapter (device model 1732-03) that is connected an INMN internal network that is managed by the Unified Resource Manager. An INMN NIC is only functional when it is coupled to an INMN virtual switch.
DEVices *devs*	Determines the number of virtual devices associated with this adapter. For a simulated HiperSockets adapter, *devs* must be a decimal value between 3 and 3072 (inclusive). For a simulated QDIO, IEDN, or INMN adapter, *devs* must be a decimal value between 3 and 240 (inclusive). The DEFINE NIC command creates a range of virtual devices from *vdev* to *vdev + devs -1* to represent this adapter in your virtual machine. The default value is 3.
CHPID *nn*	A two-digit hexadecimal number that represents the CHPID number that the invoker wants to allocate for this simulated adapter. If the requested CHPID number is available, all of the virtual devices belonging to this adapter share the same CHPID number. This option is useful only if you need to configure a virtual environment with predictable CHPID numbers for your simulated devices.

After the NIC is installed, use the **COUPLE CP** command to connect the adapter to a guest LAN or virtual switch. The following syntax is used for the **COUPLE** command in this scenario:

```
COUPLE vdev TO [ operands ]
```

In this syntax, *vdev* specifies the base virtual device address for the adapter, and *operands* defines where to connect the NIC. Table C-10 lists the operands that are accepted by the **COUPLE** command for the purpose of connecting a vNIC to a Guest LAN.

Table C-10 Operands for the Couple command

Operands	Description
vdev	This is the base address (hex) of the network adapter.
ownerid lanname	The *ownerid* is the name of the owner of the Guest LAN (such as SYSTEM). The *lanname* is the name of the Guest LAN.

Remember that a virtual NIC can only be coupled to a *compatible* guest LAN. For example, a QDIO NIC cannot be coupled to a guest LAN of type “HiperSockets”.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- *IBM z/OS V2R1 Communications Server TCP/IP Implementation Volume 1: Base Functions, Connectivity, and Routing*, SG24-8096
- *IBM System z Connectivity Handbook*, SG24-5444
- *OSA-Express Implementation Guide*, SG24-5948
- *Enhanced Networking on IBM z/VSE*, SG24-8091
- *Introduction to the New Mainframe: z/VSE Basics*, SG24-7436

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Other publications

These publications are also relevant as further information sources:

- *Linux on System z, Device Drivers, Features, and Commands*, SC33-8281
- *Stand-Alone Input/Output Configuration Program User's Guide,* SB10-7152
- *SNA Operation*, SC31-8779
- *IP Systems Administrator's Commands*, SC31-8781
- *Device Drivers, Features, and Commands*, SC33-8281
- *z/OS Communications Server, IP Configuration Guide,* SC31-8775
- *z/OS Communications Server, IP Configuration Reference,* SC31-8776
- *z/OS Communications Server: SNA Network Implementation*, SC31-8777
- *z/OS Communications Server, SNA Resource Definition Reference*, SC31-8778
- *z/OS Hardware Configuration Definition (HCD) Planning*, GA22-7525
- *z/OS Hardware Configuration Definition (HCD) User's Guide,* SC33-7988
- *z/OS Resource Measurement Facility™ (RMF) Performance Management Guide*, SC33-7992
- *z/OS Resource Measurement Facility (RMF) Report Analysis,* SC33-7991
- *z/OS Resource Measurement Facility (RMF) User's Guide,* SC33-7990
- *z/VM V6R3 Connectivity*, SC24-6174

- *z/VM V6R3 CP Commands and Utilities Reference,* SC24-6175
- *z/VM V6R3 CP Planning and Administration,* SC24-6178
- *z/VM V6R3 TCP/IP Planning and Customization,* SC24-6238
- *z/VM V6R3 TCP/IP User's Guide,* SC24-6240
- *z/VM V6R3 TCP/IP Messages and Codes,* GC24-6237
- *z/VM V6R3 TCP/IP Diagnosis Guide,* GC24-6235
- *z/VSE Planning*, SC33-8301
- *z/VSE Administration*, SC33-8304
- *z/VSE Operation*, SC33-8309
- *z/VSE V5R1 e-business Connectors User's Guide*, SC34-2629
- *z/VSE V5R1 TCP/IP Support*, SC34-2640
- *IPv6/VSE IPv6 V1R1.0 Installation Guide*, SC34-2616

Online resources

These websites are also relevant as further information sources:

- z/OS V2R1 information center

  http://pic.dhe.ibm.com/infocenter/zos/v2r1/index.jsp
- z/VM information and documentation

  http://www.vm.ibm.com
- z/OS Internet Library

  http://www-03.ibm.com/servers/eserver/zseries/zos/bkserv/
- Linux on IBM System z

  http://www-03.ibm.com/systems/z/os/linux/
  http://www.vm.ibm.com/linux
  http://www.ibm.com/servers/eserver/zseries/os/linux/
- IBM developerWorks

  http://www-128.ibm.com/developerworks
  http://www-128.ibm.com/developerworks/opensource/
- IBM performance benchmarks in developerWorks

  http://www.ibm.com/developerworks/linux/linux390/perf/index.html/
- HiperSockets NTA FAQ document and link

  http://ibm.biz/BdRFgz

  http://www.ibm.com/systems/z/hardware/networking/products.html#hipersockets
- CSI international for z/VSE TCP/IP

  http://www.csi-international.com
- Barnard Software Inc. for z/VSE

  http://www.bsiopti.com
- z/VSE documentation

  http://www.ibm.com/systems/z/os/zvse/documentation/#tcpip

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

IBM Redbooks

IBM HiperSockets Implementation Guide

(0.2”spine)
0.17”<->0.473”
90<->249 pages

IBM HiperSockets Implementation Guide

Understand HiperSockets architecture, functions, and operating systems support

Learn tips for planning and implementing HiperSockets

See examples for IBM z/OS, IBM z/VM, and Linux on System z

This IBM Redbooks publication provides information about the IBM System z HiperSockets function. It offers a broad description of the architecture, functions, and operating systems support. This publication will help you plan and implement HiperSockets. It provides information about the definitions needed to configure HiperSockets for the supported operating systems.

This book is intended for system programmers, network planners, and systems engineers who want to plan and install HiperSockets. A solid background in network and Transmission Control Protocol/Internet Protocol (TCP/IP) is assumed.

SG24-6816-02 ISBN 0738439444