Redbooks Paper

Bill Ogden
Bill White

Getting Started with zSeries Fibre Channel Protocol

Introduction

The purpose of this IBM Redpaper is two-fold: to provide information to help you understand the concepts of zSeries Fibre Channel Protocol support, and to show you how various SCSI devices can be configured to build a zSeries FCP environment.

This document provides an overview of Fibre Channel (FC) topologies and terminology, a list of the hardware and software prerequisites, and a description of our configuration, with examples of the Linux definitions that we used when writing this paper.

This Redpaper is intended for Linux administrators, system programmers, hardware planners, and system engineers who will plan and install zSeries Fiber Channel Protocol support.

Fibre Channel Protocol (FCP) support

The Fibre Channel (FC) standard was developed by the National Committee of Information Technology Standards (NCITS). The zSeries FCP I/O architecture conforms to the FC standards specified by the NCITS. More detailed information about the FC standards can be obtained from the following Web sites:

```
www.t10.org
www.t11.org
```

zSeries FCP support allows Linux running on a z800 or z900 system to access industry-standard SCSI devices. For disk applications, these FCP storage devices utilize Fixed Block (512 byte) sectors rather than Extended Count Key Data (ECKD™) format.

A new Channel Path Identifier (CHPID) type has been defined for the FICON/FICON Express feature cards, called *FCP*. The FCP CHPID type is supported on the FICON Express features 2319 and 2320 on both the z/800 and z/900, and on existing FICON features 2315 and 2318 on the z/900.

zSeries FCP support is offered for channels with devices connected via a single switch or multiple switch fabric. Parallel SCSI devices can be attached with the use of a Fibre Channel-to-SCSI bridge.

Topologies

This section provides a high-level overview of the Fibre Channel (FC) topologies. This information will be useful for technical professionals not familiar with the FC architecture, helping them to understand our configuration and setup, which is detailed in subsequent sections.

The FC architecture defines three separate topologies to support connectivity between endpoints: point-to-point, arbitrated loops, and switched fabric. Let's examine each of these in more detail.

Point-to-point

This is the simplest topology to configure. A point-to-point configuration is a direct connection between two endpoints. Typically, it consists of a host, a device (such as a disk controller), and a dedicated fiber link (see Figure 1).

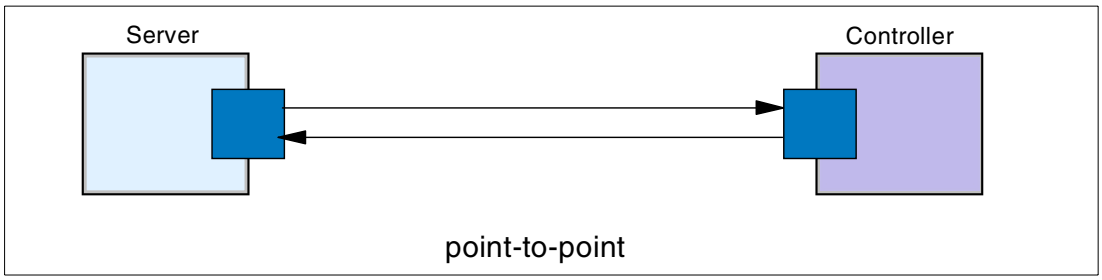

Figure 1 Point-to-point configuration

Arbitrated loop

This is a ring topology that shares the fibre channel bandwidth among multiple endpoints. The loop is implemented within a hub that interconnects the endpoints (see Figure 2 on page 3). An arbitrated scheme is used to determine which endpoint gets control of the loop. The maximum number of ports is 127.

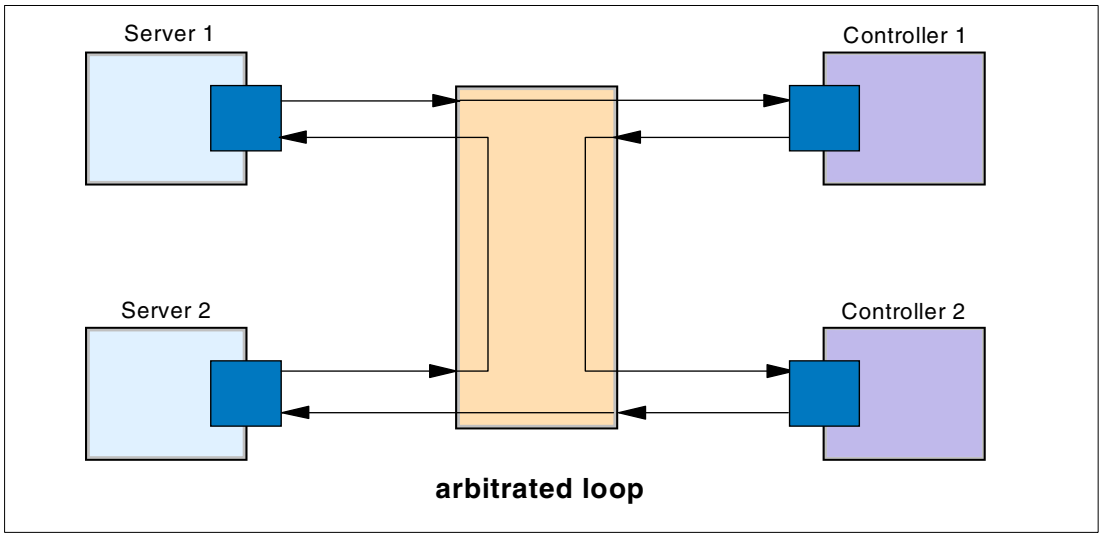

Figure 2 Arbitrated loop configuration

Switched fabric

This topology provides the most flexibility and makes the best use of the aggregated bandwidth by the use of switched connections between endpoints. One or more switches are interconnected to create a fabric, to which the endpoints are connected (see Figure 3).

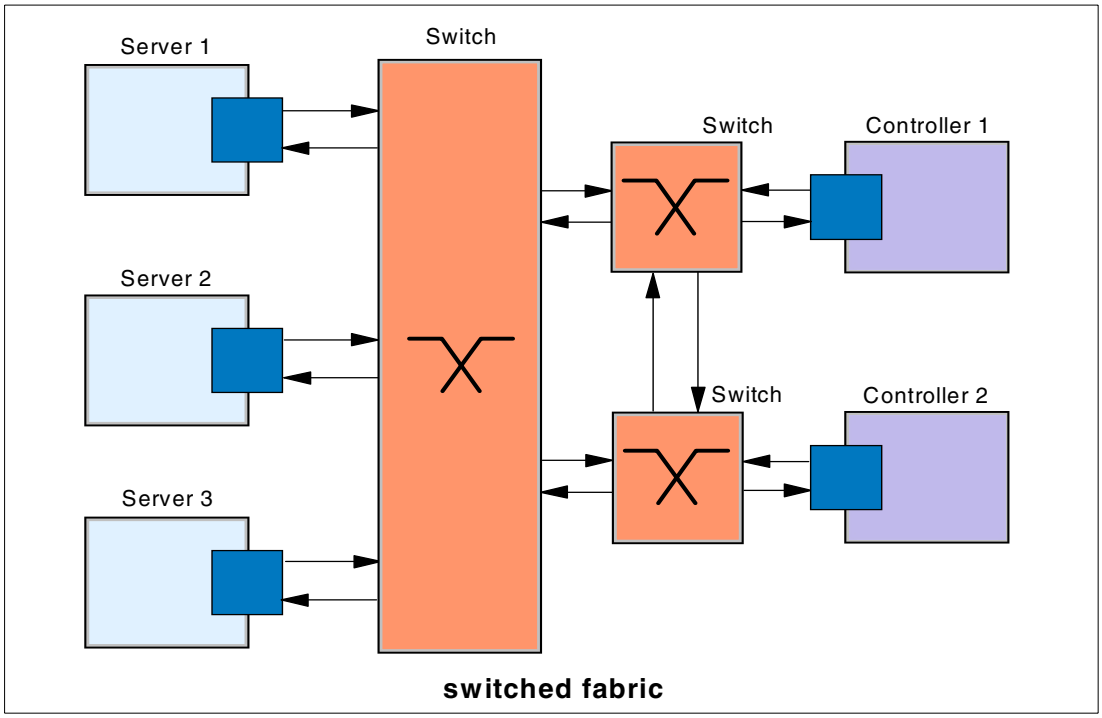

Figure 3 Switched fabric configuration

Supported topologies

The supported topologies for the zSeries (z800 and z900) FCP function include:

- A fibre channel through a single switch or multiple switches to an FCP device
- A fibre channel through a single switch or multiple switches to a Fibre Channel-to-SCSI bridge

Restriction: Point-to-point and arbitrated loop topologies are not supported as part of the FCP enablement.

Terminology

This section discusses some general terms used in the Fibre Channel (FC) environment when operating in a point-to-point, arbitrated loop, switched, or bridged configuration.

Node

A node is an endpoint that contains information. It can be a computer (host), a device controller, or a peripheral device (such as, disk or tape drives). A node has a unique 64-bit identifier known as the Node_Name. The Node_Name is used for system management purposes.

Port

Each node must have at least one port (hardware interface) to connect the node to the FC topology. This node port is referred to as an N_Port.

Each N_Port has a Port_Name, which is a unique 64-bit identifier that is assigned at the time it is manufactured. The N_Port is used to associate an access point to a node's resources.

Other port types include:

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

The port type is determined by the node's role in the topology, as shown in Figure 4 on page 5.

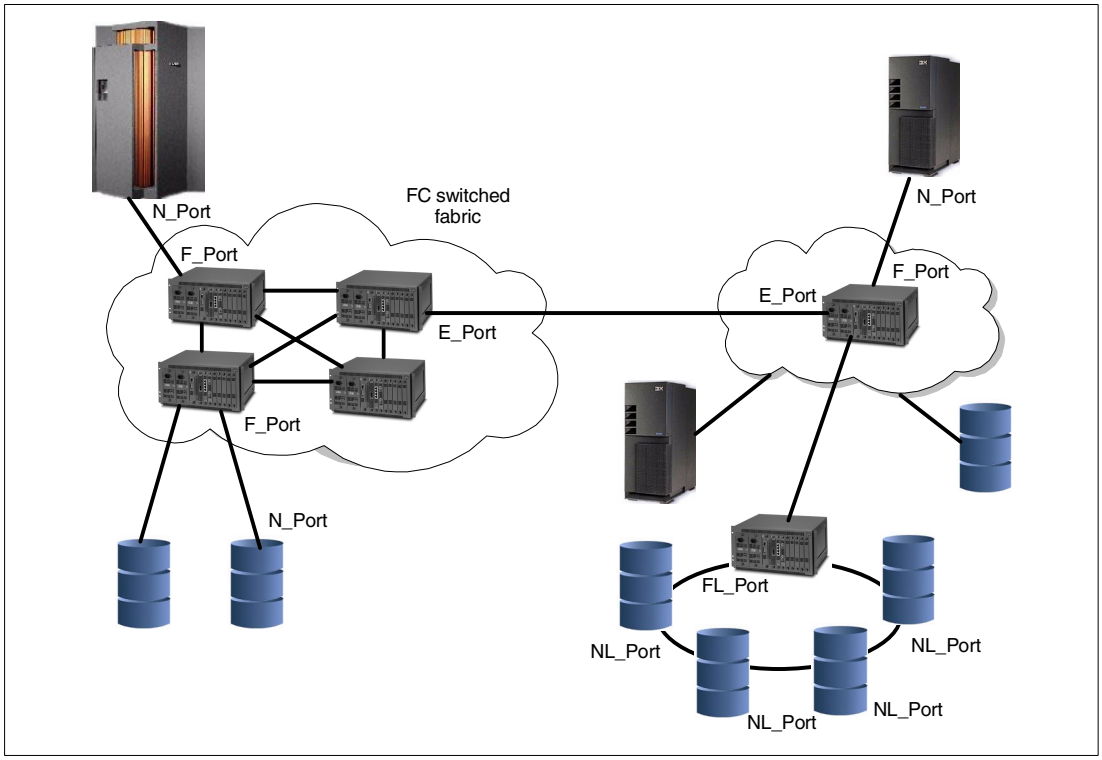

Figure 4 FC port types

Link

The port connects to the topology through a link. The link is a fiber optic cable that has two strands, one used to transmit a signal and the other to receive a signal (see Figure 5). A link is used to interconnect nodes and/or switches.

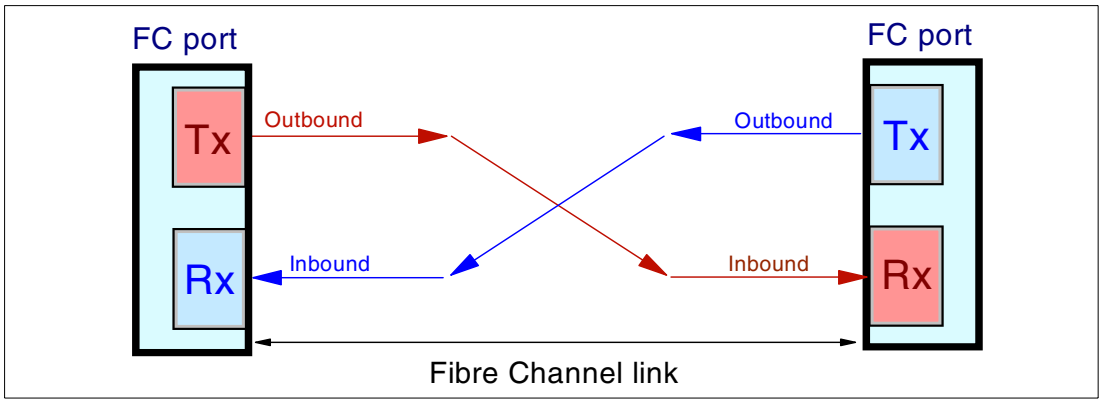

Figure 5 Fibre Channel link

For example, a Fiber Channel link (port-to-port connection) can be:

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

World-Wide Names

As mentioned, nodes and ports have unique 64-bit address that are used to identify them in an FC topology. These addresses are assigned by the manufacturer, with a vendor-specific portion defined by the IEEE standards committee. These addresses (in the FC standard) are called Node_Names and Port_Names, and when they are world-wide unique, they are referred to as:

- World-Wide Node_Name (WWNN)
- World-Wide Port_Name (WWPN)

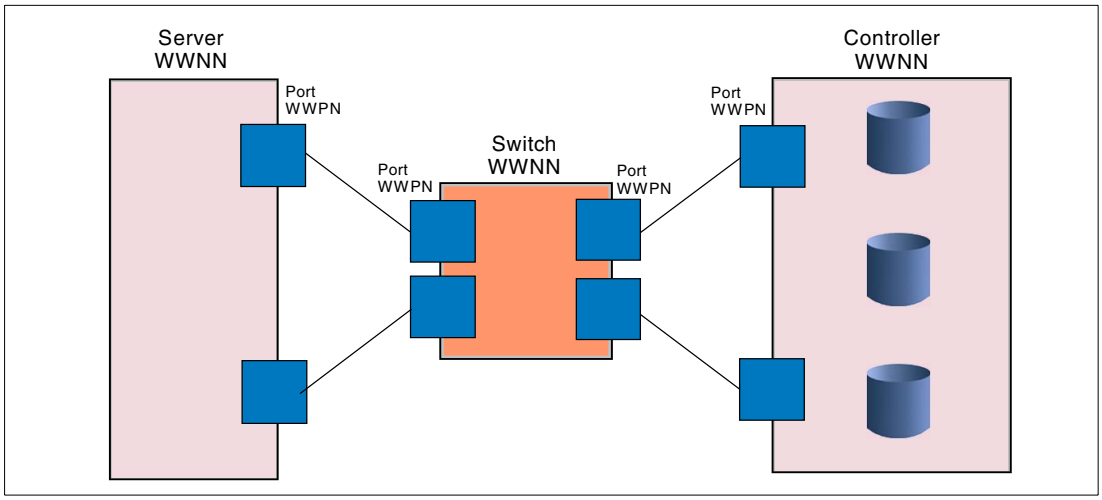

Figure 6 Example of World-Wide Names

Controlling access

The ability to control access to nodes and/or devices is provided as a function in switches and controllers, and is called *LUN masking* and *zoning*. LUN masking and zoning can be used, for example, to prevent servers from accessing storage that they are not permitted to access. In the following sections we look at these functions in more detail.

Logical Unit Number (LUN) masking

A LUN represents a portion of a controller, such as a disk device. With the use of LUNs, a controller can be logically divided into independent partitions. Access to these LUNs can be restricted to distinctive WWPNs as part of the controller configuration. This method is known as LUN masking.

Figure 7 on page 7 depicts an example of LUN masking. In this example, only Server 1 (WWPN1) is permitted to access LUN1. LUN2 can be accessed by Server 1 and Server 2 (WWPN2), and LUN3 can only be accessed by Server 3 (WWPN3).

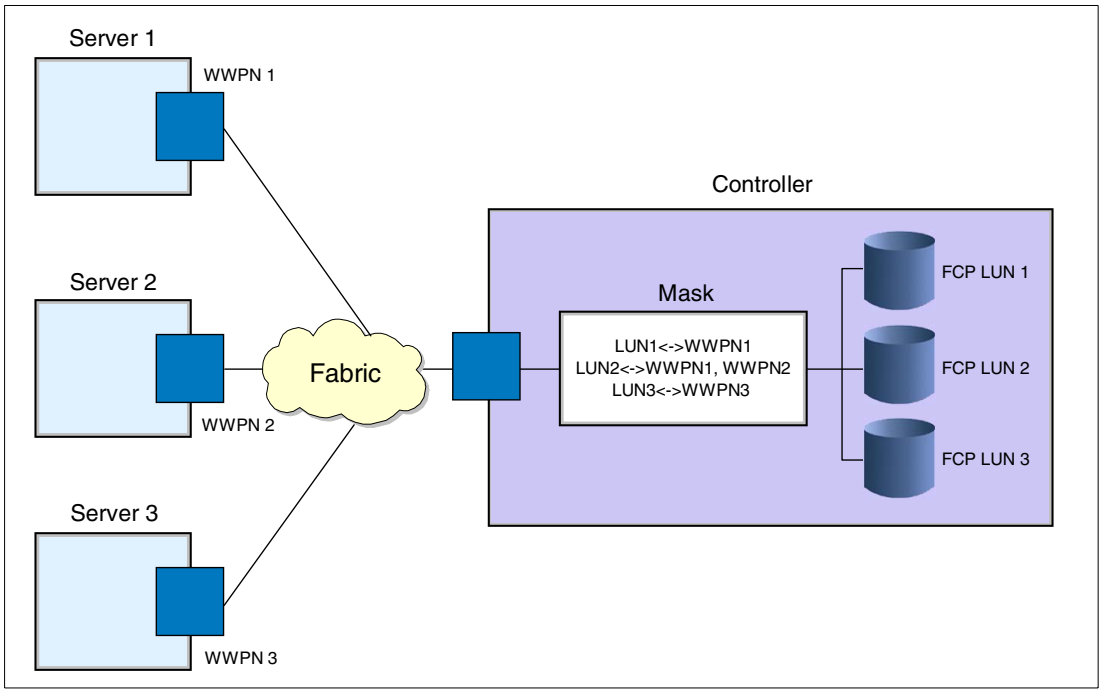

Figure 7 LUN masking example

Zoning

Segmentation of a switched fabric is achieved through zoning. It can be used to partition off certain portions of the switched fabric, allowing only members of a zone to communicate within that zone. All others attempting to access from outside that zone are rejected, hence zoning provides a security function.

Zoning can be implemented in two ways: hard zoning or soft zoning.

1. Hard zoning is based on the physical port number. A port number can belong to multiple zones.
2. Soft zoning is defined in the name server of a switch, using WWNs. The WWN can belong to multiple zones.

Figure 8 on page 8 illustrates an example of zoning. In this example, the group1 members are Server 1 and Target 4, Target 5, and Target 6. The group2 members are Server 2 and Target 5, therefore, Server 2 is not permitted to communicate with Target 4 and Target 6. The group3 members are Server 3 and Target 6, therefore, Server 3 is not permitted to communicate with Target 4 and Target 5.

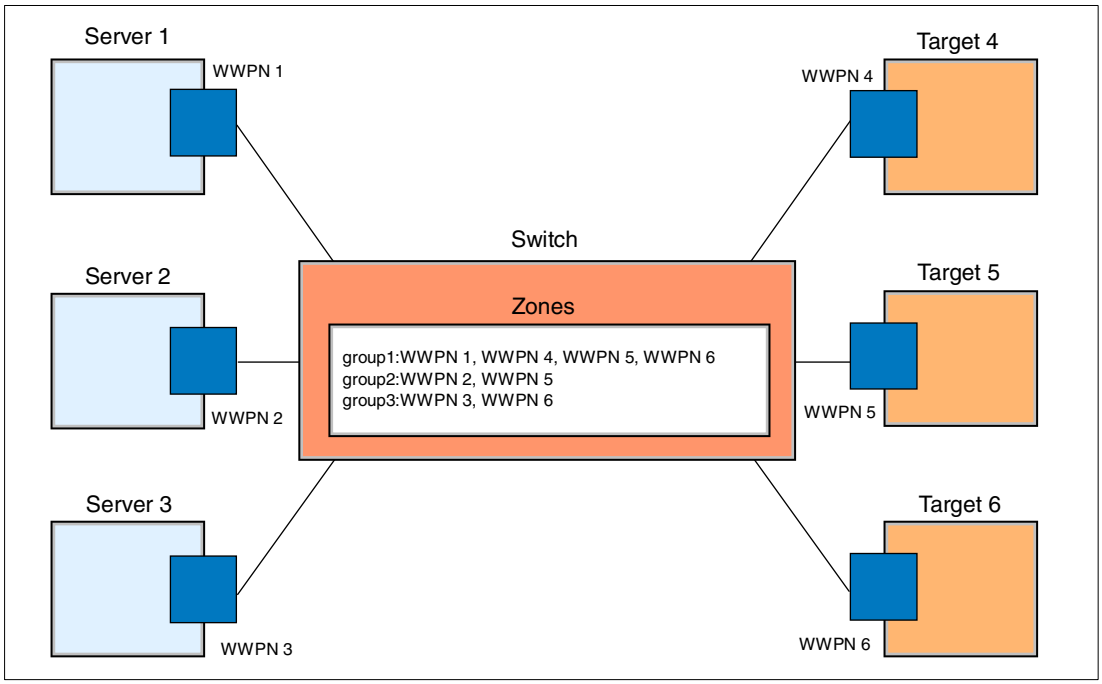

Figure 8 Zoning example

For detailed information about FC topologies and terminology, as well as LUN masking and zoning, refer to *IBM SAN Survival Guide*, SG24-6143.

Prerequisites

The zSeries (z800 and z900) FCP function for Linux is implemented by hardware features and software support described in this section.

Hardware requirements

zSeries FCP channels require a FICON card (feature 2315 or 2318), or a FICON Express card (feature 2319 or 2320). This is the same channel hardware used for FICON channels, however, a different firmware load is required.

The type of firmware to be loaded into the FICON/FICON Express card, turning it into either an FCP channel or into one of the FICON type channels (FC or FCV), is controlled by the definition of the channel type for that particular channel in the IOCP (CHPID statement). Thus, by defining FCP type channels in the IOCP, the total number of FICON type channels that can be configured is reduced accordingly.

Check with your local service representative to insure your zSeries server has the required firmware installed.

The two channels residing on a single FICON/FICON Express card can be configured individually, and each can be a different channel type.

Fiber implementation options

There are two laser types available for both FICON channels and FICON Express channels:

- Long wavelength laser (LX) at 1300nm
- Short wavelength laser (SX) at 850nm

These two laser types, combined with the different fiber cable modes, support five implementation options, as listed in Table 1.

Table 1 FICON and FICON Express cabling options

Laser type	Single-mode (9/125)	Multimode (62.5/125)	Multimode (50/125)
Long wavelength	X	X[a]	X[a]
Short wavelength		X	X

a. Mode Conditioning Patch (MCP) cables are required

The recommended (and most flexible) fiber cable option is based on 9-micron single-mode fiber cables with long wavelength lasers. However, the optimum cabling implementation option for any given installation depends on the environment and configuration requirements.

From a cabling standpoint, the most important factor of an FC link is the selection of the laser type. This is based on requirements such as distance, attenuation, and fiber cable type.

Keep in mind: The laser type (LX or SX) at the ends of each FC link must match.

Connector types

The FICON features 2315 (LX) and 2318 (SX) have SC Duplex port types, and the FICON Express features 2319 (LX) and 2320 (SX) have LC Duplex port types. Figure 9 shows the fiber cable connector types needed for these features.

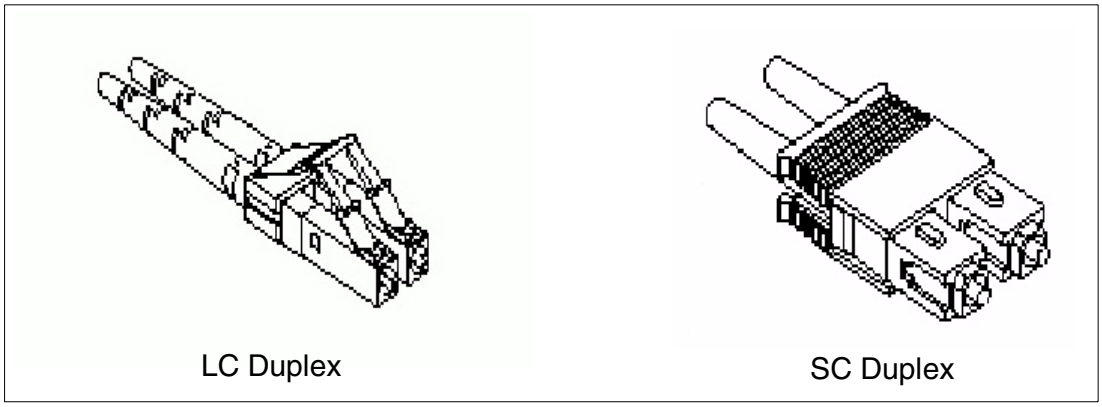

Figure 9 Fiber cable connector types

Note: Conversion Kit cables are required to connect SC Duplex connectors to FICON Express ports (LC Duplex).

Distances

Table 2 lists the supported fiber cable types and distances allowed for the FICON and FICON Express cards.

Table 2 zSeries supported distances

Fiber cable type	Laser type	Distance
Multimode (50/125)	Short wavelength	Up to 500 meters
Multimode (62.5/125)	Short wavelength	Up to 300 meters

Fiber cable type	Laser type	Distance
Multimode (50/125)[a]	Long wavelength	Up to 550 meters
Multimode (62.5/125)[a]	Long wavelength	Up to 550 meters
Single-mode (9/125)	Long wavelength	Up to 10 km

a. Mode Conditioning Patch (MCP) cables are required

Software requirements

FCP and SCSI controllers and devices can be accessed by Linux for zSeries (64-bit mode) or Linux for S/390 (31-bit mode) with the appropriate I/O driver support. Linux may run either natively in a logical partition, or as a guest operating system under z/VM Version 4 Release 3.

Note: z/VM Version 4 Release 3 is required to support FCP for Linux guests. However, z/VM itself does not support FCP devices.

For current information on FCP channel support for Linux for zSeries or Linux for S/390, and for appropriate support for z/VM, see:

http://oss.software.ibm.com/developerworks/opensource/linux390/technical-2.14.19-may2002.shtml

Considerations and limitations

FCP usage from a zSeries system is new, and will certainly evolve over time. Likewise, the switched fabric capabilities[1] will continue to evolve. You should be aware of the following current limitations:

- **IPL**

  An operating system cannot be loaded (“IPLed”) from FCP devices. This means that CKD or FBA disks (with appropriate channels and control units) are required. From a Linux perspective, the system must be booted from a CKD or FBA disk. Once booted (and after the necessary drivers have been installed), all subsequent disk accesses can be from the FCP devices.

- **Multipathing**

  The term *multipathing* can imply a variety of functions. In traditional zSeries terminology, it usually involves multiple (possibly simultaneous) channels to access a disk. No multipathing of this nature exists for FCP devices under Linux for zSeries (or Linux for S/390). (Multiple paths may exist through the FCP fabric, but this is a different topic.)

- **Security and integrity**

  We briefly discuss these topics in “Security controls” on page 38. Considerable planning is needed in this area.

- **Supported devices**

  A large number of SCSI devices exist in the marketplace, especially when older devices are included. IBM cannot fully test all of these for use in zSeries systems. There will be a distinction between *IBM supported devices* (such as the Enterprise Storage Server, Enterprise Tape System 3590, and Enterprise Tape Library 3494), devices that have been used and appear to work[2] but are not formally supported for zSeries FCP by IBM, and

[1] For practical purposes, this means the functions available through the FC switches.
[2] Most of the devices we used while writing this paper are in this category.

devices that have never been tried with zSeries FCP. A fourth category might be devices that are known not to work with zSeries FCP. For a complete list of supported devices, consult:

http://http://www-1.ibm.com/servers/eserver/zseries/connectivity/#fcp

- **z/VM support**

  At the time of writing, z/VM supports FCP operation to the extent of passing it through to guest systems. A more complete virtualization of FCP operation is not available at this time.

- **FC topologies**

  Only an FC switched fabric environment is supported. Point-to-point connections (an FCP channel connected directly to a FAStT200, for example) and arbitrated loops are not supported.

- **FCP channel capabilities**

  Because of the inherent store-and-forward architecture, the total amount of data that can be transferred with a single FCP operation is limited. However, the maximum size is large compared to traditional zSeries channel operations. Linked SCSI commands (a method used to move large amounts of data in a single I/O operation) are not supported by the current implementation of the FCP channel.

Addressing concepts

Implicit in zSeries FCP usage are a number of different *addressing* styles and concepts. To make it all work, there must be ways to *map* one type of address to another type of address. This FCP *address mapping* will be a new element to experienced zSeries users, and a short review of some of the concepts involved may be helpful.

Traditional SCSI

Traditional SCSI devices have simple addresses. Older devices use a number in the range 0-7, with address 7 usually reserved for the SCSI controller. These are known as *target* addresses. Newer SCSI architecture extends this range to 0-15 (or 0-F in hexadecimal), with 7 still usually reserved for the controller.

The target address of the SCSI device is usually set by a hardware function, such as a thumb switch or jumper pins. The user must ensure that two devices are not set to the same target address. More advanced implementations, often used with hot-pluggable drives, set the target address automatically by using backplane hardware logic.

Each of the devices (targets) can have subsidiary units, known as LUNs (logical units). These are provided in the SCSI architecture. Older systems seldom used anything other than LUN0, and they used this by default; the system administrator often did not “see” the LUN. The older SCSI architecture had a maximum of 7 LUNs (3 bits for addressing), while newer definitions provide for much larger numbers.

More advanced SCSI adapters provide several *bus* interfaces, and each bus can have a full compliment of target devices (and each target can have its LUNs). Several SCSI adapters (each with several bus interfaces) can be used, although this is usually seen only in larger systems.

RAID adapters often map their logical drives into separate SCSI target addresses. Some RAID adapters have an option to map their logical drives into different LUN numbers at a single target address, but this setup is seldom encountered in typical small SCSI systems.

zSeries device numbers

In S/360 days, a S/360 “address” had a specific meaning. For example, a unit at address 327 was on channel 3, control unit 2, unit 7. This was later expanded to allow two hex digits for the channel address, so address B237 meant channel (or CHPID) number B2, control unit 3, device 7.

Later system evolution changed this “address” to a “device number” (which is still informally known as an “address” or a “software address”). The device number is set by the IOCDS and is an arbitrary 16-bit number expressed as four hexadecimal digits. It is no longer related to actual channel, control unit, or device addresses unless the person constructing the IOCDS happens to assign device numbers to match some elements of the hardware addresses involved.

zSeries hardware addresses

At the zSeries hardware level, a number of addresses are involved in using an I/O device. These include:

- One or more channel identifiers (CHPIDs)
- Optionally, a channel switch (such as an ESCON director) for which the appropriate port addresses are needed (the LINK addresses)
- One or more [logical] control unit addresses (CUADDs)
- A unit address (UNITADD)

The key addressing elements (CHPID, CUADD, UNITADD) are involved for all zSeries devices, although their exact meaning can vary, depending on the device type. For example, the meaning of UNITADD is quite different for a 3390 disk and an OSA Express (LAN) adapter using QDIO interfaces. (For QDIO devices the IOCDS actually defines a queue, but we can ignore this detail in this general description.)

FCP addresses

A new addressing scheme was developed for Fibre Channel Protocol (FCP) usage. It is built around World Wide Names (WWN) that are eight bytes long. Part of the name represents an address type, part is a number that identifies the manufacturer, and part is a unique number assigned (by the manufacturer) for each *port* or *node*.

A node is typically a box that contains information. Nodes have one or more *ports* (and only FC ports are relevant here). A given box may have several addresses; one for the node itself, and one for each FC port contained in the node. Common abbreviations are:

- WWNN is a World Wide Node Name
- WWPN is a World Wide Port Name
- WWN is any World Wide Name (WWNN or WWPN)

A WWN is usually written in sets of two hex digits, separated by colons (for example, 08:45:12:56:43:00:D5:A0).

In addition to the WWNN and WWPN names, addressable units (such as a disk drive, or logical drive on a RAID controller) are assigned a unit name that is also an 8-byte number. These assigned names are created by the FCP node owning the device. The devices are known as LUNs (for logical units) and the names are LUN names. For these network devices, it’s important to remember that a *physical drive* (such as our old SCSI drives, described later) can be a LUN. Think of a LUN as an addressable object within a node.

Linux device names

Linux works with device names instead of device addresses. For example, /dev/hda might be a disk drive. The exact scheme and name format varies between Linux platforms and, to some extent, is a function of the device drivers being used.

Basic Linux for zSeries uses names such as this:

- /dev/dasda (name of whole drive, such as a 3390-3)
- /dev/dasda1 (name of a partition on drive dasda)

A more recent version of Linux for zSeries (using the optional devfs module) provides a different mapping:

- /dev/dasd/0A80/disc (name of whole drive, such as a 3390-3, at device number A80)
- /dev/dasd/0A80/part1 (partition on the drive at device number A80)

Another addressing/mapping extension is provided by devfs. This maps the volume serial number of a disk to the name provided by devfs:

- /dev/labels/WORK01 --> /dev/DASD/0302 (where WORK01 is the volser)

Linux SCSI support

Linux provides extensive support for SCSI devices. From a naming/addressing viewpoint, it is based on the more advanced forms of traditional SCSI addressing. That is, a SCSI device is expected to have a number (target address, typically a single digit) and this target may have multiple LUNs. The full SCSI addressing scheme can be used:

- Device number (from your IOCDS) (coded as a “host number”)
- A bus number on this adapter (always zero in current zSeries implementations)
- A SCSI target number on this bus
- A LUN number on this SCSI target
- A partition or device within this LUN

If devfs is used, the name is written in this style:

```
/dev/scsi/host3/bus0/target2/lun4/part2
```

devfs

This is an optional function that can be used to replace the normal /dev entries in Linux. The option is selected when the Linux kernel is built. This paper assumes that Linux uses devfs, and all the device names discussed are in the format created by devfs.

If you have Linux for zSeries (or Linux for S/390) running, you can easily determine if your system was built with devfs.

```
# ls /dev/dasd                          (dasd is the device type for CKD disks)
0A90 0A91 0A92 ....
```

If the /dev/dasd directory contains a number of names consisting of four hexadecimal digits, you are using devfs. These “names” are the device numbers (“software addresses”) of all the CKD disks available to your system, including the disk you used to boot Linux.

Mapping points

For someone familiar with zSeries, the best known mapping point is the IOCDS. This is, in effect, a table (used by the zSeries internal firmware) that translates between device numbers (“software addresses”) and hardware addresses (CHPID, LINK address, CUADD, UNITADD).

A support team for a zSeries installation creates the IOCDS. This can be a complex undertaking for a larger system, because it involves the real hardware addresses (CHPID, CUADD, and so forth), descriptions of multiple paths to devices, and segmentation of the devices by LPARs (if needed). For this reason, a z/OS tool called HCD is often used to create an IOCDS (although it can also be created with a simple editor on a PC). The IOCDS contains all the required mapping for using traditional zSeries I/O with traditional zSeries operating systems.

(To be more precise, a file known as an IOCP source file must be compiled to create an IOCDS. An appropriate compiler is available as a standalone zSeries function or within z/OS. The term “IOCDS” is often used for both the source and the compiled results; this usage is technically incorrect, but is commonly used and understood.)

Linux requires an IOCDS to access zSeries I/O. Earlier Linux systems used parameters to the DASD I/O driver to map zSeries disk device numbers to Linux names. For example, these parameters could cause disk A80 to be named /dev/dasdc, and so forth. Other Linux files might map addresses E20 and E21 to a LCS driver as /dev/eth0. The exact nature of this mapping is still changing as the zSeries I/O drivers for Linux evolve. A Linux administrator can manipulate this device mapping by editing various files in the /etc file system, and by changing the boot parameter list. Various Linux distributions have higher-level administrative interfaces that help hide some of the details involved.

FCP usage involves another mapping element, which must be understood in order to successfully implement FCP. The goal of this mapping is to make logical drives in an FCP network (either simple SCSI drives or, more likely, logical drives in a RAID array) appear as “normal” SCSI drives to Linux. Typically, they are made to appear as a number of LUNs on a few SCSI targets. I/O access is still through standard zSeries architecture, and this requires the use of the IOCDS. The mapping table used for FCP involves device numbers (to match the IOCDS), WWN addresses (to specify FCP-connected drives), and local SCSI target/LUN numbers (to fit with the standard Linux SCSI naming patterns).

Linux FCP mapping

The required FCP mapping is unfamiliar to experienced zSeries users. Some of the mapping elements (device number, Linux SCSI target IDs and LUNs) are assigned on the zSeries side (by appropriate IOCDS parameters or Linux file contents), and other mapping elements are created within the FC fabric (WWN and LUNs). LUN addresses appear on both the zSeries side and the FCP side, and it is important not to confuse the two uses of LUNs.

Remember that a LUN can be a disk drive (a physical drive or a logical drive within a RAID array), a tape drive, a CD-ROM, and so forth. Usually it is a disk drive and we assume this for the following discussion.

The format of an FCP map entry is:

```
     1    2        3               4       5               <---Element number in map
   0x600 0x01:0xpppppppppppppppp  0x00:0xnnnnnnnnnnnnnnnn
     |  |        |               |       |
     |  |        |               |       +--LUN name within SAN node
     |  |        |               +----------LUN number to be used by Linux
     |  |        +--------------------------WWPN of SAN node
     |  +-----------------------------------SCSI target number to be used by Linux
     +--------------------------------------Device number for FCP channel (from IOCDS)
```

Each LUN to be accessed in the FC network requires a map entry like this. Lines of mapping data can be provided to Linux in a number of different ways, and these ways are discussed

later. You must construct this mapping information from the information provided by your SAN hardware. In our case, this was the FC switch, the Data Gateway and the FAStT200.

There are five elements in a map entry, and each is discussed in the following sections. You will normally have many map entries. The entries can be in any order, but there is a minor performance benefit in ordering by device number, target, and then LUN. We suggest that a map is easier to maintain if all the entries for a particular WWPN (element three in the format listing above) are grouped together, and, within this group, entries are ordered by their LUN number.

FCP map and devfs names

You must deal with both FCP maps and devfs names. Figure 10 illustrates how some of the values are related.

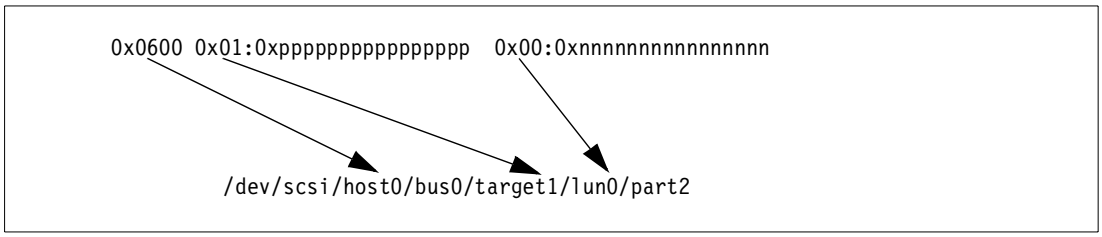

Figure 10 FCP to devfs mapping

The host number starts with zero and is incremented for each *new* device number used in the FCP map. The Linux target and LUN numbers in the FCP map are the same numbers used in the devfs name. The *part2* in the example refers to a partition on the target disk, and was produced by **fdisk** when the disk was partitioned.

1) Device number

The first element in the map is a zSeries device number. This device number must be defined in the IOCP/IOCDS and be assigned to the FCP channel attached to the FCP switch. You can use the same device number for all your FCP connections (by a given Linux image), or you can elect to use multiple device numbers (all assigned to the FCP channel, of course). “IOCP/IOCDS definitions” on page 27 shows an FCP channel definition with 64 device numbers assigned to it.

The following rules apply to the use of device numbers for FCP channels:

- In a simple case (ignoring LPAR effects), a *device number* corresponds to a *unit* on the channel. In more complex cases, we need to consider both *device numbers* and *units*.[3]
- The maximum number of FCP channels possible depends on your specific z800 or z900 configuration. There are two channels per adapter card, and each of these channels can be used as a FICON channel or an FCP channel.
- A maximum of 240 valid subchannels (or unit device numbers, in the simplest case) may be specified for a single FCP channel.
- You can use the same unit (device number) for all your FCP maps within a given Linux image. In the unlikely event you exceed the maximum number of target addresses and LUN addresses possible on a single unit address, you might need additional units (additional device numbers).

[3] It is difficult to avoid reusing certain terms, such as “unit” and “address”, for multiple purposes in this discussion. When discussing device numbers and IOCDS assignments, a “unit” on an FCP channel is an abstraction. It does not correspond to anything in the network and, in particular, is not a LUN. As used here, a unit corresponds to a device address used by *one* LPAR. The *unit* is a communication path (QDIO queue pair) between the FCP channel and an operating system.

- However, if an error occurs with a SCSI device, operation of *all devices connected to the device number* may be stopped while recovery takes place. If you have known devices that tend to be unreliable, you should consider using a different device number for those devices. It is recommended to use separate device numbers for each Tape device.
- Each Linux image under z/VM must use a different device number to access FCP devices. This implies that a maximum of 240 Linux images can be used under z/VM if only one FCP channel is available. If two FCP channels are available, a maximum of 480 Linux images under z/VM can use them, and so forth.
- Your IOCDS can define the same device numbers for multiple LPARs. However, the zSeries channel firmware internally creates separate "units" for each LPAR. The total of these devices cannot exceed 240 per FCP channel. The IOCDS definitions in Example 1 on page 28 defines 64 device numbers, starting with 0600. These are defined for *two* LPARs (LINUX1 and LINUX2), and use 2 x 64 = 128 of the maximum possible 240 units in the FCP channel.
- All the above rules imply that a maximum of 240 Linux images can use a single FCP channel, regardless of whether the images are run natively in an LPAR, or under z/VM, or under multiple z/VMs in several LPARs, or some combination of these modes.
- Device numbers are expressed as four-digit hexadecimal numbers, in the normal zSeries style. High-order zeros may be omitted.
- Unit addresses (in your IOCDS definitions for device numbers) must *not* include FC, FD, FE, and FF. They are reserved for future functions.

2) Target number

The standard Linux SCSI support understands traditional SCSI addressing, with adapter numbers, bus numbers, target ("SCSI address") numbers, and LUNs. The second element of the map provide a SCSI target number for Linux to use. You assign this number. The basic rules are these:

- Target address 0 cannot be used. Address 0 is automatically used, internally, to address the FCP adapter itself and cannot be used for other purposes.
- Usable target addresses range from 1 to any positive 31-bit number. (Traditional SCSI target addresses ranged from 0-7 or 0-15. More recent SCSI architecture allows a much larger number.)
- Normal usage starts with target address 1, and increments by 1 for each new WWPN used. We did not experiment with nonsequential target numbers. That is, we started with number 1 and incremented this by 1 for each new WWPN we used.

  However, we understand that in later releases, any positive 31-bit number can be used for a target number, and that the target numbers need not be sequential. We also understand that there is a minor performance benefit to starting with 1 and incrementing by 1.
- Target addresses *can* be reused with different device numbers.
- A different target address must be used for each different WWPN address you use in your FCP map.
- All LUNs on a single WWPN address (using the same device number) must use the same target number.
- The number can be expressed in decimal, hexadecimal, or (we assume) octal. We suggest you select a uniform method of expressing these numbers.

3) WWPN (port number)

The third map element is the World Wide Port Name (WWPN) of the device containing the LUN. This is the WWPN *as seen by the FC switches*. WWPNs reported by devices may be slightly different. Basic rules are:

- In practice, this number is always written in hexadecimal.
- It must be exactly 16 hexadecimal digits.
- It must be the WWPN as used by whatever switches are installed in your FC fabric.

  You do not assign this number; it is built into the FCP devices and you must determine the proper number by querying your FCP elements.

4) Linux LUN number

The fourth element is LUN number *to be used by Linux*. You assign this number. The basic rules are:

- Normal usage is to start with 0 and to increment by one for each LUN. (We attempted to use nonsequential numbers and were unable to access some of the devices. That is, we attempted to use LUN numbers 0, 1, and 8. We were unable to access the device (tape drive) at LUN 8. We changed the LUN numbers to 0, 1, and 2, and were able to access the device.)

5) SAN device LUN number

The fifth element is the LUN number used by the remote device. You do not assign this number. You must determine the number assigned by the node controller. For our ITSO devices and setup, the practical aspects were these:

- In practice, this number is always written in hexadecimal.
- It must be *exactly* 16 hexadecimal digits.
- All of the devices we worked with, used only the high-order four digits of the 16 hexadecimal digit LUN address. Some devices only check the first four hexadecimal digits; therefore, different LUN addresses that have the same high-order four hexadecimal digits will lead to the access of the same device. This can cause errors or erratic operation. Thus, we strongly recommend that the low-order twelve hexadecimal digits be zero.

  The format of the high-order four hexadecimal digits is not well defined and may vary with different products. The products we used for this paper all assigned simple LUN addresses starting with 0000, 0001, 0002, and so forth. However, not all controllers start their LUN addresses at 0000, so you should not assume these addresses.

Syntax rules

You should be aware of a few basic syntax rules for FCP maps:

- By default, numbers in the FCP maps are “C language” decimal. If you want a hexadecimal number, you must indicate this with a “0X” prefix. The “C language” condition means that a leading zero (when no 0X prefix is present) indicates an octal number; for example, 010 (which is an octal number) equals 8 (decimal).
- Comments may appear in maps that exist as separate lines in a file. They are indicated by the # character at the beginning of the comment. The comment can be a whole line, or the right-hand portion of a line.
- Where multiple FCP map entries may be supplied as parameters to a module, the entries are separated with semicolons. The back slash (\) escape character should be used to make all entries appear as the same input line. Some command formats require the whole map to be enclosed in double quotes. See the following example:

  ```
  insmod zfcp map="\
  0x0600 0x1:0x1234567887654321 0x0:0x0000000000000000;\
  0x0600 0x1:0x1234567887654321 0X1:0x0001000000000000"
  ```

- The requirement for starting at a particular number (such as LUN0) and assigning sequential numbers may be removed in later releases.

Device descriptions

This section provides a short description of the devices we used to demonstrate the necessary steps and definitions to set up our zSeries FCP environment, shown in Figure 11 on page 20.

FAStT

The FAStT200 Storage Server is a 3U rack-mountable device containing a single RAID controller and space for up to 10 Fibre Channel (FC) hard disk drives. It contains hot-swappable and redundant power supplies and fans; however, there is no RAID controller redundancy.

The FAStT200 HA Storage Server is identical to the FAStT200 Storage Server, except that it contains two hot-pluggable RAID controllers and can therefore provide a higher level of availability. This is the model we installed.

You can upgrade the FAStT200 Storage Server by adding the hot-pluggable FAStT200 Redundant RAID Controller, effectively turning it into the FAStT200 HA Storage Server.

The RAID controller unit has the following fiber connectors:

- Host port GBIC slot

  It is used to attach the FAStT200 to node or a switch. Either a short or long wavelength GBIC can be installed in this slot.

- Expansion port GBIC slot

  This connector provides storage expandability. Up to ten additional EXP500 storage expansion enclosures can be physically connected, allowing a total of up to 110 disk drives. Either short or long wavelength GBICs can be install.

For more information on the FAStT, refer to *Fibre Array Storage Technology Introduction,* SG24-6246.

Data Gateway

The IBM SAN Data Gateway (SDG) is basically a protocol converter between FCP and SCSI, and acts as a FCP-to-SCSI bridge. It is a fully scalable product with up to three Fibre Channel and four Ultra SCSI Differential interfaces for disk and tape storage attachment.

Each Fibre Channel interface could support dual or single short wavelength ports (for a maximum of six ports), and single long wavelength ports (for a maximum of two ports). The details of the interfaces on a SDG are as follows:

- Fibre Channel
  - Supports both loop (private and public) and point-to-point topologies
  - Supports connections to nodes or switches
- SCSI
  - SCSI channels have automatic speed negotiation capability for wide or narrow bus widths and Standard, Fast, or Ultra speeds.
  - Each channel supports up to 15 target IDs and up to 32 LUNs per ID. However, there is a 255 LUN maximum for the SDG.

Note: Although the SDG supports a maximum of 255 LUNs, the limitations of the host operating system and storage subsystem must be considered. Whichever is the lowest, dictates the maximum value that can be used.

The SDG also supports zoning and LUN masking between the FC ports and SCSI ports. This enables you to specify which Fibre Channel hosts can connect to LUNs defined on a given SCSI ports. This is essential to avoid data integrity problems (by not allowing multiple hosts to access the same LUNs).

For information on the IBM SAN Data Gateway, refer to *Installation and User 's Guide 2108 Model G07,* SC26-7304.

FC switch

The IBM 2109 S-16 SAN Fibre Channel Switch is used to interconnect host systems with other devices. This switch is available with 8 ports (model S-8) and 16 ports (model S-16). Each switch port provides bandwidth of up to 100MB/s with maximum latency of 2 microseconds, and the non-blocking architecture allows multiple simultaneous connections. The switch ports can operate in F_Port, FL_Port or E_Port modes.

The IBM 2109 comes with four short wavelength GBICs: additional switch ports (either short or long wavelength GBICs) can be installed.

The 2109 supports cascading of several switches, to achieve a complex Fabric with a large number of switch ports. You can also increase the distance by connecting the switches in series. Up to seven hops are allowed; this means up to 70 km if you use long wavelength GBICs and single-mode fiber cables.

This switch is self-learning, allowing the Fabric to automatically discover and register host and devices. Another important capability is self-healing, which enables the Fabric to isolate a problem port and reroute traffic through an alternate path.

For performance and flexibility, an internal processor provides services such as name serving, zoning and routing. For availability, you can add an optional second hot-plug power supply to provide redundant power. Dynamic microcode upgrades are also supported.

For more information on the IBM 2109, refer to *IBM SAN Survival Guide: Featuring the IBM 2109,* SG24-6127.

ITSO configuration

Figure 11 on page 20 provides an overview of the hardware configuration we used while writing this paper. It is not intended to represent physical sizes, but rather to delineate a typical small FCP environment. The FC Switch, Data Gateway, and FAStT200 are all small, desktop units.

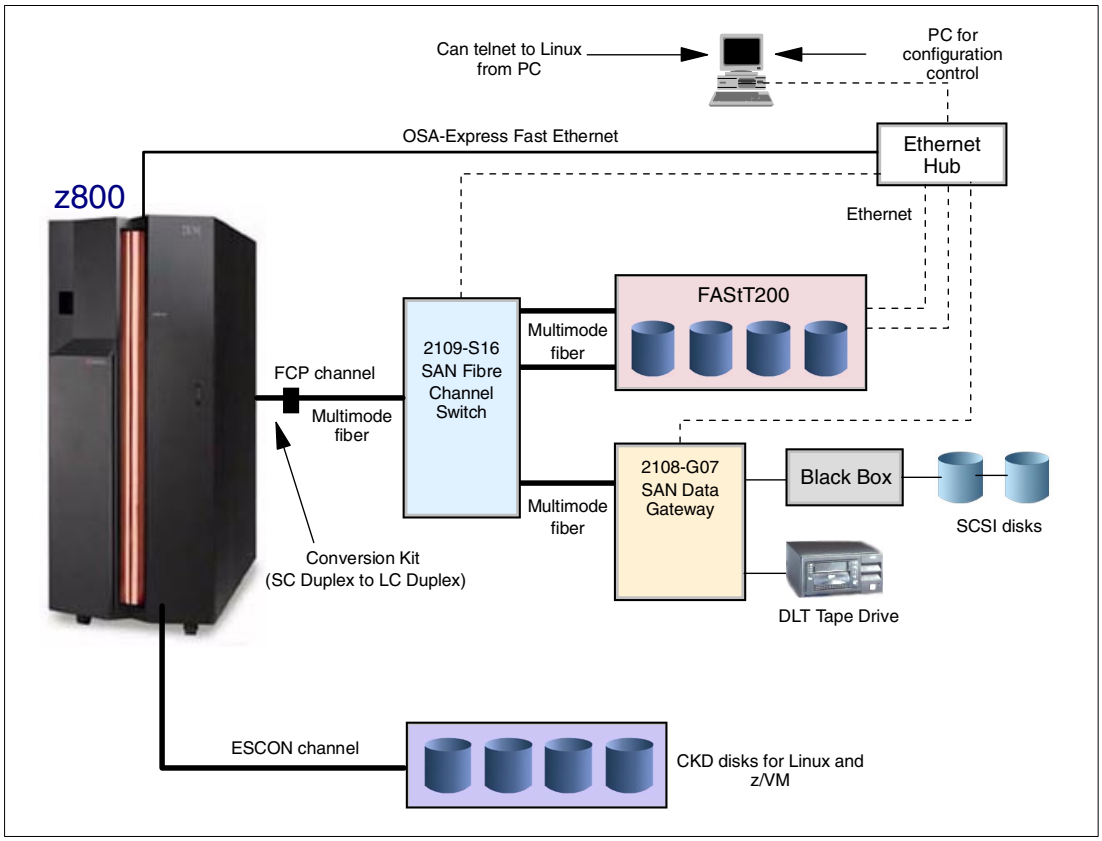

Figure 11 Hardware for initial ITSO operation

Briefly, the units involved were these:

- A z/800 processor, with the FICON Express feature 2320 (SX) and the appropriate FCP channel enablement firmware. A conversion cable was needed to connect the FCP port (with an LC duplex connector) to our existing fiber cables (with SC duplex connectors). All our fiber interfaces had short wavelength ports, and we used multimode fiber cables.
  - We used two LPARs on the z800: one for running Linux native, the other for running Linux as a guest system under z/VM.
  - A string of existing 3390-type drives was used for Linux and z/VM residence. Our Linux resided on a single 3390-3, and z/VM used two 3390-3 drives. A clone of the Linux system used another 3390-3 drive. We used one ESCON channel to access these drives.
  - One OSA Express Fast Ethernet channel was available and we dedicated it for our use.
- A PC, running a version of Microsoft Windows, was used to customize the SAN units. We also used it for convenient Telnet sessions to Linux. A CD-ROM drive on the PC was needed to load configuration software for the FAStT200 unit.
- An Ethernet hub (IBM 8242-016) permitted us to establish an isolated LAN to configure and use this system. This is a 10 Mbps unmanaged hub with 16 ports. There was nothing unique about it.
- The Ethernet cables were simple 4-wire RJ45 cables.
- The IBM 2109-S16 Storage Area Network Switch provided the FC fabric function. This is the smallest FC switch available from IBM.

- The IBM 2108-G07 Storage Area Network Data Gateway provided an interface between older 68-pin SCSI devices and a fiber interface. The SCSI interfaces are differential SCSI.
- The IBM 7205-311 DLT tape drive is a differential SCSI unit and is similar to tape drives found with many UNIX systems. An appropriate SCSI terminator was installed.
- The two SCSI disk drives (IBM 7204-139 and IBM 7204-114) are small, single-ended SCSI drives (in external cabinets, each with a power supply). These were previously used with older RS/6000 systems. The second drive contained an appropriate terminator.
- The Black Box converter (Black Box part number IC152A) converts between differential and single-ended SCSI.
- The SCSI cables and connectors were the standard 68-pin version.
- The IBM 3542-2BU FAStT200 unit provided high-volume disk storage. It contains its own RAID adapter. We installed four 36 GB disk drives, in a RAID 5 configuration with a single array. We partitioned the array into four logical drives, with 25 GB per drive. (With only four physical drives, we elected not to use one as a *hot spare*.) We used the high-availability version of the FAStT200, with two controllers. This provides two fiber interfaces and two RAID controllers.
- Not shown in Figure 11 on page 20 are two null-modem serial port cables: one cable with an RJ45 connector on one end and 9-pin D-shell on the other end, and the other cable with 9-pin D-shell connectors on both ends. These cables are used with the FAStT200 and Gateway for initial setup (primarily to assign IP addresses to the units). One cable was supplied with the FAStT200 unit.

Installing the devices

We set up our devices, as shown in Figure 11 on page 20, in an orderly way and gathered the mapping information we would need later for Linux. We initially installed the devices using default settings whenever possible. For this setup we used no LUN masking or zoning—meaning that any Linux system with access to the devices could access *all* devices. (Expressed another way, we took the simplest setup we could devise.)

Important: Sharing of CHPIDs is allowed between multiple Linux images. However, this initial FCP offering is limited to serial Logical Unit Number (LUN) access, that is, multiple Linux operating systems cannot share the same LUN over the same physical CHPID.

Private LAN

You need to connect **`telnet`** sessions and other applications between a PC (running a current release of Microsoft Windows and using an Ethernet adapter) and several boxes, as shown by the dotted lines in Figure 11 on page 20. This can be done using a public LAN, provided you have assigned IP addresses you can use for the boxes.

However, we recommend *against* using a public LAN for this purpose. One consideration is security (or the lack of security). The security controls consist of simple userid/password combinations, with well-known defaults.

A private Ethernet LAN can be as simple as a single unmanaged 10 Mpbs Ethernet hub. A suitable 8-port unit can cost less than US$100.

Given a private LAN, you can assign any IP addresses you wish. They should all be on the same subnet. The most common approach is to use the private Class A addresses, with no subnets defined. That is, use addresses such as 10.xxx.xxx.xxx, with net mask 255.0.0.0.

Our Data Gateway already had IP address 10.21.30.111 when we received it. (This is not a default address; it was set by a previous user.) We continued to use this address, and planned other addresses in the same general range:

```
Data Gateway:      10.21.30.111
FCP Switch:        10.21.30.119
FAStT200:          10.21.30.112 and 10.21.30.113
PC:                10.21.30.1
z800 (Linux):      10.21.30.100
```

There is no requirement to connect the Linux running on the z800 to same PC used for device customization; we did it simply for convenience. Again, we stress that these IP addresses were quite arbitrary.

SCSI devices

Our older SCSI devices (two disk drives and a tape drive) had thumb switches for setting a SCSI target address. We arbitrarily set the addresses for the two disk drives as 4 and 5. We connected the tape drive to a different SCSI bus in the Data Gateway and set its address to 4. (We did this to demonstrate that the same address on different SCSI buses would not be a problem.)

The Black Box converter, shown in Figure 11 on page 20, is transparent to the rest of the hardware and had no address or other setup associated with it. It was required because the Data Gateway has only differential SCSI connections, and our old SCSI disk drives are single-ended SCSI devices. Our DLT tape drive is differential SCSI and connected directly to one of the SCSI adapters in the Gateway. All these connections use 68-pin cables.

Data Gateway

There are two basic levels of customization and query required for the Data Gateway:

- IP address assignment
- Unit customization and query

Initial IP address assignment is done using a serial connection (with a null-modem cable) between the Data Gateway unit and a “dumb ASCII terminal.” A userid and password can also be set this way. In practice, this “dumb ASCII terminal” is probably the HyperTerminal application provided with Microsoft Windows. Our Data Gateway already had an assigned IP address. We did not use this serial connection procedure and cannot describe it in detail.

A set of GUI applications is available to use for configuring and querying the Data Gateway. These applications include the *SAN Data Gateway Explorer* and the *IBM StorWatch SAN Data Gateway Specialist*. These are valuable when a large or complex SAN network is involved—especially if masking will be used to isolate LUNs within the Data Gateway. For our simple, small network (with no initial masking), we elected to use only the line commands available through **`telnet`** and did not install either of these GUI applications.

In your case, once the IP address is set, you can **`telnet`** from the PC to the Data Gateway. A previous user had established userid *Admin* with password *password*. Once logged into the Data Gateway, use the **`help`** command to review the available functions. We suggest you take time to explore most of the commands.

We discovered two key facts:

- The Data Gateway automatically configures itself in the case of a simple environment like ours.
- Only a few line commands were needed for our environment.

The commands we used are these:

```
initializeBox          erases existing device maps.  Need to log in again.
scsiRescan             examines all the connected scsi devices
scsiShow               lists all the scsi devices it has detected
targets                lists devices in a different way
fcShow                 lists FCP connections
fcShowDevs             lists the scsi devices ready for FCP connections
mapShowDatabase        lists the scsi device with FCP information
```

The Data Gateway remembers the identity of SCSI devices, even after they are disconnected. (We discovered this when we disconnected the tape drive and found that **scsiShow** still listed it.) The **initializeBox** command clears this memory. A **scsiRescan** function is performed automatically when the Data Gateway is booted, or you can enter the command manually; it simply checks the SCSI interfaces and adds any new devices to its internal lists.

The **scsiShow** command lists attached devices, and includes the SCSI address (ID) and any LUNs on that device (typically there is only LUN 0, which is the basic device):

```
    SCSI Initiator  Channel 1: 0xC19C9460
ID  LUN  Vendor     Product      Rev     |   sync/offset   width
----------------------------------------|---------------------
4    0   IBMRISC    DFHSS4W     LYED     |   12/15         16 sw..      (old scsi disk)
5    0   IBM        DGHSO9U     03E0     |   12/15          8 s..       (other old disk)
    SCSI Initiator  Channel 2: 0xC1990B60
No devices
    SCSI Initiator  Channel 3: 0xC19E6DE0
4    0   Quantum    IBM-7205    2560     |   25/15         16 sw..      (tape drive)
    SCSI Initiator  Channel4: 0xC19AD6E0
No devices
```

The **targets** command lists similar information in a different way:

```
Idx    Tdev       Vendor   Product       Rev   |  Type Specific
-----------------------------------------------|---------------
 0  0xC197C900  PATHLIGHT  SAN Gateway  0016   | cmd/cntrl status 0h....
 1  0xC1FFC590  IBMRISC    DFHSS4W      LYED   | Disk 8,813,869 blks of 512 bytes...
 2  0xC1FFBF10  IBM        DGHSO9U      03E0   | Disk 17,774,159 blks of 512 bytes...
 4  0xC15E6610  Quantum    IBM-7205     2560   | Tape: blksize 0, flags 4000000...
```

The **fcShow** command lists FC interface information:

```
                        Fibre Channel Controllers
-------------------------------------------------------------------------------
ctlr | PCI Addr | ISP  |Firmware   | Firmware | Loop | Fabric   | Prot
 ID  | Bs Dv Fn | Type | Stack     | Version  |  ID  | Attached | Mode
-------------------------------------------------------------------------------
 1     00 06 00   2200   Ready       2.01.12     0       yes      targ
 4     00 18 00   2200   sync lost   2.01.12    none     no       targ
```

The Data Gateway has two FC interfaces. We used only the first one and, in this sample output, it was successfully attached to the FC switch (*Fabric Attached = yes*).

The **fcShowDevs** command lists the devices visible to the FC network:

```
FC1:
LUN  chan  Id  LUN  Vendor      Product      Rev    SN
-----------------------------------------------------
 0    0    0    0   PATHLIGHT   SAN Gateway  0016   21081302087
 1    1    4    0   IBMRISC     DFHSS4W      LYED   PCB=20-113000-02 ...
 2    1    5    0   IBM         DGHS09U      03E0   6827C524GA
 4    3    4    0   QUANTUM     IBM-7205     2560   PXA4591590
FC4:
(same list)
```

The first LUN column lists the FCP LUN identity assigned to the devices. This number is used later to construct an FCP map for Linux. The chan/ID/LUN columns list the local scsi ID and LUN. Notice that these local addresses have been mapped to SAN network LUN numbers. Both disks appear as LUNs on a single channel.

The **mapShowDatabase** command provides more information:

```
devID  Type   Chan   tID  tLUN  UID
-------------------------------------------------
 000   SNA    127    127  127   00000060:45161247
 001   SCSI   001    004  000   20100060:45161247
 002   SCSI   001    005  000   20200060:45161247
 004   SCSI   003    004  000   20400060:45161247
 005   SCSI   003    004  001   20500060:45161247
```

In this listing, SNA refers to the Data Gateway (and has no connection with VTAM terminology). The *chan* is the SCSI channel, corresponding to the SCSI connector on the back of the box. The *tID* and *tLUN* are the local SCSI target and LUN addresses. Notice that the gateway automatically converted the DLT tape drive into two entries in this list.

Several of the output examples contain four-byte hexadecimal numbers. These refer to internal addresses within the Data Gateway, and are only for diagnostic purposes.

FAStT200 setup

There are two basic levels of customization and query required for the FAStT200:

- IP address assignment for the Ethernet ports
- Logical customization, using connections to the Ethernet ports

Initial IP address assignment is done using a serial connection (with a null-modem cable) between the FAStT200 unit and a “dumb ASCII terminal.” In practice, this terminal is probably the HyperTerminal application provided with Microsoft Windows. Our FAStT200 had two controllers; each must be assigned an IP address.

Serial connection

We connected the null model cable provided with the FAStT200 to controller A and to serial port 1 of our PC. We started a HyperTerminal session using 9600 bps, 8 data bits, no parity, and 1 stop bit. This configuration is sometimes written as 9600-8-N-1. We also used the autodetect mode, although the ANSI mode can also be used.

Start the terminal (with the Connect option, through the toolbar) and press Ctrl-Break (where the Break key is same as the Pause key on many PC keyboards). This should produce a logo screen and a request for a password. The default password is *infiniti*.

Once logged into the FAStT200, use **help** to obtain a partial list of line commands available. The resulting list is helpful, but does not contain one of the key commands (**netCfgSet)** that is

required at this stage (the **netHelp** and **ifShow** commands are also useful). Remember to use Enter to scroll through all the help commands before attempting to enter another command.

As mentioned, the key command you need at this point is **netCfgSet**. It will step through a list of network parameters, one per line, allowing you to enter a new value for that line. The parameters (and our responses to the right of the colons) are:

```
--> netCfgSet
My Host Name        : FAStA                     (We picked the name FAStA)
My IP Address       : 10.21.30.112              (The address we assigned)
Server Hostname     : tot86                     (Our PC name; probably not needed)
Server IP Address   : 10.21.30.1                (IP address of our PC)
Gateway IP Address  : 10.21.30.1
Subnet Mask         : 255.0.0.0
Network Int Flags   : 0x00                      (Never knew what these were for)
Network Mgmt Timeout: 30
Network Route #1    : dest= 10.21.30.1
Network Route #2    :
RAIDMGR Server #1   : 10.21.30.1
RAIDMGR Server #2   :
Network Manager #1  : 10.21.30.1
Network Manager #2  :
Shell password      :
Username            : guest
User password       :
NFS                 :
```

We then connected the serial cable to the second controller in the FAStT200 and did exactly the same setup. We assigned this controller the IP address of: 10.21.30.113.

We went through this initial setup procedure a number of times before we were able to **telnet** to the Ethernet ports of the controllers. For the first few attempts, we supplied only the IP address and subnet mask in the list shown above. We found that we needed to supply *all* the parameters shown here (for both FAStT200 controllers) before an Ethernet connection would respond.

It is necessary to cycle the power off and back on for both FAStT200 controllers after entering parameters through the serial port. After entering the addressing parameters shown here, and cycling power, we could **ping** the FAStT200 from our PC.

GUI application

A CD-ROM is provided with the FAStT200, and this contains a number of programs and documentation. The only program we used is the FAStT Storage Manager 7 Client. (We installed all the programs we could find on the CD, but found we needed only one of them.)

We discovered we were unable to connect the GUI client (over the Ethernet connection) to the FAStT200 until we had completely set the IP information through the serial ports to *both* FAStT200 controllers. We also needed an Ethernet connection to *both* controllers.

After starting the FAStT Storage Manager 7 Client, we used the **Add Device** function (available through the Edit function in the toolbar) to add the Ethernet addresses of both FAStT200 controllers. This produced a screen similar to Figure 12 on page 26.

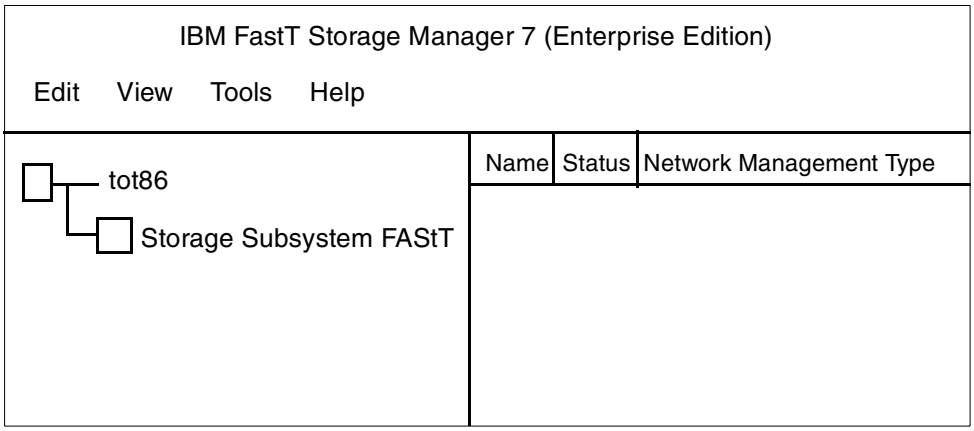

Figure 12 Initial display of FastT Storage Manager 7 Client

Double-clicking the Subsystem name (in the left portion of the screen) produces a second screen with a toolbar containing the following:

- Storage Subsystem
- Configuration
- Array
- Logical Drive
- Controller
- Drive
- Help

We used these functions to create a single RAID 5 array and define four logical drives of 25 GB each.

Note: We first tried the automatic configuration function, but found that it configured a hot spare—leaving us with the equivalent of only two drives of space. We deleted this configuration and manually configured a RAID 5 array with no hot spare.

Once we had this Client application started, we explored the toolbar options and found it easy and intuitive to use for normal RAID and logical drive setup. Our choice of four logical drives was completely arbitrary and was not related to the fact that we had four physical drives in the FAStT200.

After defining our four logical drives, we found we had 12 FC WWNs in our FAStT200. (There are several ways to display these, using various toolbar options.) The names were:

```
Controller A WWPN         200400A0B80BA687
Controller A WWNN         200400A0B80BA686
Controller B WWPN         200500A0B80BA687
Controller B WWNN         200400A0B80BA686       (same as controller A WWNN)
Physical drive slot 1     040000206E437FCF
Physical drive slot 2     0400002052417FCF
Physical drive slot 3     04000020EB437FCF
Physical drive slot 4     040000205F437FCF
Logical drive 1           600A0B80000BA68E000008F13CDA47D3
Logical drive 2           600A0B80000BA68E000008F23CDA4863
Logical drive 3           600A0B80000BA68E000008F33CDA487F
Logical drive 4           600A0B80000BA68E000008F43CDA489A
```

WWNs are 8 bytes. Names with 16 bytes, such as the logical drive names, are two WWNs concatenated together.

Switch setup

The FC switch we used has a configuration panel on the front of the box. It is a small two-line LCD with four buttons for input. We used this panel to set a SCSI address. We used the default userid/password for **telnet** connections; this was userid *admin* and password *password*. Once logged into the box, a **help** command can be used to list the available functions.

We found the switch automatically configured itself appropriately for our purposes. Manual configuration is needed for more sophisticated requirements, such as zoning. A GUI application is available to assist with these more complex setups, but we did not use the GUI application for our simple environment.

We found two line commands (**nsShow** and **fabricShow**) especially useful, and these provided information we needed to construct FCP maps for Linux access. The **nsShow** command produced the following:

```
Type Pid     COS   Portname               Nodename                 TTL
 NL  041200   3;  20:01:00:60:45:16:12:47;10:00:00:60:45:16:12:47; na FC4s: FCP IBM...
 N   041500   3;  20:05:00:A0:B8:0B:A6:87;20:04:00:A0;B8:0B:A6:86; na FC4s: FCP IBM...
 N   041500   3;  20:04:00:A0:B8:0B:A6:87;20:04:00:A8;B8:0B:A6:86; na FC4s: FCP IBM...
 N   041F00 3,2;  50:05:07:64:01:00:01:7D;50:05:07:64:00:C1:1C:E3; na
```

This listing reflects the fiber connections to the switch. The first line is the Data Gateway. The second and third lines are the FAStT200. The last line is the FCP channel to the zSeries host. We deduced these connections by comparing the WWNs shown here with the WWNs displayed by our telnet sessions to the Data Gateway and the FAStT200. These WWNs do not exactly match (due to internal structured fields within the WWNs), but the match is close enough to determine which line in **shShow** output corresponds to which device.

The **fabricShow** command produced:

```
Switch  ID   WorldWide Name             Enet IP Addr  FC IP Addr   Name
  4   fffc04 10:00:00:60:69:10:65:03    10.21.30.119   0.0.0.0     IBM 2109
```

This lists all the FC switches in the network. We had only one. As shown, we did not set an IP address to access the switch through a fiber connection.

IOCP/IOCDS definitions

zSeries FCP support is provided via the FICON or FICON Express features of the zSeries processors. As with all channel-attached devices, these features must be defined by a channel path, a control unit, and I/O devices in the IOCP/IOCDS.

A CHPID Report that specifies where the FICON or FICON Express feature is plugged into your z800 or z900 server can be supplied by your IBM representative, or obtained through panels in the Support Element. The CHPID number and type (FCP) will be required for your FCP configuration setup.

These are the hardware configuration limits:

- 240 valid subchannels (which implies a maximum of 240 Linux instances per CHPID)
- 512 physical destination ports per CHPID
- 2048 logical destination ports per CHPID
- 4096 LUNs per CHPID

Briefly, these numbers mean the following:

- A maximum of 240 Linux images (or other operating systems using FCP) can use a single FCP channel. This number is reduced if any images use more than one device number to access the FC network. This total is across all LPARs and within all z/VM systems running Linux guests. The 240 value is the maximum number of valid subchannels that can use the FCP CHPID.
- A maximum of 512 remote WWPNs can be accessed through a single FCP channel. In our case, our FAStT200 had two destination ports and the Data Gateway had a single port. FC switches do not have *destination* ports, in the context used here.
- As implemented within the FCP CHPID firmware, operating systems access *logical ports*. As a trivial example, four operating systems (in four LPARs) could each access the same 512 physical ports, thus exhausting the maximum of 512 physical ports and 2048 logical ports.
- A maximum of 4096 LUNs (on remote nodes accessed through the WWPNs) can be used from a single FCP channel.

The IOCP/IOCDS does not define the actual Fibre Channel storage controllers and devices, nor the Fibre Channel interconnect units like switches, directors, bridges, and so forth. The Fibre Channel devices (nodes) in a Fibre Channel network are addressed using World Wide Names (WWNs), Fibre Channel Identifiers (IDs), and Logical Unit Numbers (LUNs). These addresses are configured on an operating system level, and passed to the FCP channel together with the corresponding Fibre Channel I/O or service request via a logical QDIO device (queue).

Example 1 shows the IOCP definitions we used in our environment.

Example 1 IOCP definitions

```
RESOURCE PARTITION=((A1,1),(A2,2),(A3,3),(A4,9),(A5,A),(A6,B),          *
(C1,4),(C2,5),(C3,C),(C4,D),(LINUX1,6),(LINUX2,7),(ZOSE1,8))
         CHPID PATH=(14),SHARED,                                        *
               PARTITION=((LINUX1,LINUX2),(LINUX1,LINUX2)),TYPE=FCP
         CNTLUNIT CUNUMBR=0600,PATH=(14),UNIT=FCP
         IODEVICE ADDRESS=(600,064),CUNUMBR=(0600),UNIT=FCP
```

This definition provides 64 addresses (600, 601, 602, and so on) for the FCP channel. We used only two addresses (600 and 601); the extra defined addresses do no harm in our simple situation. However, these extra addresses do consume storage both in Linux and on the FCP channel adapter. A better plan is to define only the device addresses you actually require.

z/VM guest system access

z/VM allows for permanent allocation of I/O devices to guest systems, via the DEDICATE statement. This is also the case for FCP I/O devices. Following is an example of the z/VM authorized user ID statement we used:

```
DEDICATE 600-601 LINUX2
```

Our addresses

We obtained the following WWPN addresses and LUN names during the installation of our hardware:

```
From the FC switch (nsShow command)
Type Pid     COS   Portname               Nodename                TTL
 NL  041200   3;  20:01:00:60:45:16:12:47;10:00:00:60:45:16:12:47; na FC4s: FCP IBM...
 N   041500   3;  20:05:00:A0:B8:0B:A6:87;20:04:00:A0;B8:0B:A6:86; na FC4s: FCP IBM...
 N   041500   3;  20:04:00:A0:B8:0B:A6:87;20:04:00:A8;B8:0B:A6:86; na FC4s: FCP IBM...
 N   041F00 3,2;  50:05:07:64:01:00:01:7D;50:05:07:64:00:C1:1C:E3; na

From the FAStT200 (status display)
Logical drive 1          600A0B80000BA68E000008F13CDA47D3
Logical drive 2          600A0B80000BA68E000008F23CDA4863
Logical drive 3          600A0B80000BA68E000008F33CDA487F
Logical drive 4          600A0B80000BA68E000008F43CDA489A

From the Data Gateway (fcShowDevs command)
FC1:
LUN  chan  Id  LUN  Vendor      Product      Rev    SN
-----------------------------------------------------------
 0    0    0    0   PATHLIGHT   SAN Gateway  0016   21081302087
 1    1    4    0   IBMRISC     DFHSS4W      LYED   PCB=20-11300-02 ...
 2    1    5    0   IBM         DGHS09U      03E0   6827C524GA
 4    3    4    0   QUANTUM     IBM-7205     2560   PXA4591590
```

For our installation, we decided to map these as:

```
SCSI target 1 LUN 0  First old SCSI drive (at actual SCSI thumb switch address 4)
SCSI target 1 LUN 1  Second old SCSI drive (thumb switch address 5)
  (leave a gap in LUN numbers for more old disks)
SCSI target 1 LUN 2  DLT tape drive
SCSI target 2 LUN 0  First logical drive on FAStT200
SCSI target 2 LUN 1  Second logical drive on FAStT200
SCSI target 2 LUN 2  Third logical drive on FAStT200
SCSI target 2 LUN 3  Fourth logical drive on FAStT200
```

Our map looked like this:

```
0x600 0x01:0x2001006045161247 0x00:0x0001000000000000     (first old scsi drive)
0x600 0x01:0x2001006045161247 0x01:0x0002000000000000     (second old scsi drive)
0x600 0x01:0x2001006045161247 0x02:0x0004000000000000     (tape drive)
0x600 0x02:0x200500a0b80ba687 0x00:0x0000000000000000     (first logical drive FAStT)
0x600 0x02:0x200500a0b80ba687 0x01:0x0001000000000000     (second logical drive FAStT)
0x600 0x02:0x200500a0b80ba687 0x02:0x0002000000000000     (third logical drive FAStT)
0x600 0x02:0x200500a0b80ba687 0x03:0x0003000000000000     (fourth logical drive FAStT)
```

How did we determine these numbers?

The device number, 600, was the first available device number in our IOCDS (we could have used any of the 64 device numbers defined in our IOCDS, and arbitrarily selected the first one). We could use more than one device number, but this would reduce the number of other Linux images that could potentially use these FCP devices.

The target numbers (0x01 and 0x02) are sequential, but different target numbers must be assigned to different WWPN addresses. We had two different WWPNs in this map.

The WWPNs (the third element in the map lines) are taken from the Portname section in the listing produced by the **nsShow** command on the FC switch. We know the first line (in our **nsShow** output) is for the Data Gateway. The next two lines are for the FAStT200. However, we remember that when we configured this unit, we assigned all the drives to the first controller. Therefore, we used the WWPN address for the FAStT200. (If this had not worked, we would have tried the other WWPN.)

For the fourth elements (the LUN number on the target address, as used by Linux) we simply started with 0 for each target.

The fifth elements (the LUN numbers assigned on the network nodes) can be confusing. For the Data Gateway devices, we used the LUN number in the first column of the output of the **fcShowDevs** command. For the FAStT200, we assumed that logical drive 1 would be LUN 0, logical drive 2 would be LUN 1, and so forth. We found this was correct once we had the FCP devices operational. In each case, the device LUN number is four hexadecimal digits (right justified), followed by 12 hexadecimal zeros.

Linux usage

Once you have your hardware installed and the FCP mapping parameters ready, you can begin working with Linux. You need to follow these steps:

1. Ensure that the required modules are loaded.
2. Make the FCP map available to the zfcp module.
3. Notify the SCSI driver about the SCSI drives defined for FCP access (if they were not accessible at boot time).
4. Before you use the new disks the first time, partition them and create file systems.
5. Mount the new disks.

Required modules

You must have the following modules installed to use FCP devices:

- qdio - the same module is used for other qdio devices
- scsi_mod - general SCSI support
- zfcp - provides FCP support for zSeries Linux
- sd_mod - SCSI disk support
- st - SCSI tape support (if needed)
- sr_mod - SCSI CD/DVD (read-only) support (if needed)
- sg - other SCSI devices (if needed)

The modules should be loaded in the order listed. All except zfcp can be loaded with **modprobe**. The zfcp module is best loaded with **insmod**.

You can use Linux with or without devfs, but the use of devfs makes the naming of FCP disks (and the naming of CKD disks) more orderly. The remainder of this paper assumes you are using devfs.

Disk names with devfs

If you use devfs, then CKD disks have names such as:

```
/dev/dasd/0300/device       whole drive, before formatting
/dev/dasd/0300/disc         whole drive, after formatting
/dev/dasd/0300/part1        first partition
```

SCSI disks have names such as:

```
/dev/scsi/host0/bus0/target1/lun3/disc       whole drive
/dev/scsi/host0/bus0/target1/lun3/part1      first partition
```

The last part of the name is device, disc, part1, part2, part3, and so forth. The device name is not used for SCSI. Only four partitions are allowed for DASD (CKD) disks, but up to 16 are allowed for SCSI devices. Only four partitions can be *primary* partitions, but one primary partition can be made into several *extended* partitions. (This design is obviously derived from PC disks, but it applies to any disks seen as SCSI drives.)

The SCSI disk names used by devfs have these characteristics:

- The /dev/scsi portion of the name is fixed.
- The host*n* portion of the name is related to the device number in the FCP map (which is 0600, in our examples). The first device number used in the FCP map is host0. If a second device number is used, it is host1, and so forth.
- The host*n* numbers are assigned in the order in which different device numbers are found in the fcp map. Many systems will use only one device number for FCP access, and host0 is the keyword for this.
- The bus*n* number is always bus0 for zSeries machines.
- The target and lun numbers correspond to the second and fourth elements in the FCP map.
- The numeric portions of these names are normally written as decimal numbers.

Note that the **format** step required for CKD disks has no equivalent for SCSI disks.

Shell script

We constructed the following shell script in /etc/fcpgo (an arbitrary name):

```
#!/usr/sh
rmmod zfcp
modprobe qdio
modprobe scsi_mod
insmod zfcp map="\
0x600 0x01:0x2001006045161247 0x00:0x0001000000000000;\
0x600 0x01:0x2001006045161247 0x01:0x0002000000000000;\
0x600 0x01:0x2001006045161247 0x02:0x0004000000000000;\
0x600 0x02:0x200500a0b80ba687 0x00:0x0000000000000000;\
0x600 0x02:0x200500a0b80ba687 0x01:0x0001000000000000;\
0x600 0x02:0x200500a0b80ba687 0x02:0x0002000000000000;\
0x600 0x02:0x200500a0b80ba687 0x03:0x0003000000000000"
modprobe sd_mod
modprobe st
```

We ran this shell script after booting Linux, and after making any changes to the FCP map in the shell script. Basic considerations for this shell script include:

- Executing **modprobe** for a module that is already loaded does no harm.
- We used **insmod** to load zfcp (the FCP driver).

- A module loaded with **insmod** must be unloaded with **rmmod** before it can again be loaded with **insmod**. We placed an appropriate **rmmod** at the beginning of the shell script.

  The first time the script is run, there is no zfcp to remove, but this does no harm. In subsequent use of the script, zfcp is removed so that it can be loaded again (with, presumably, altered parameters).
- The script assumes the use of SCSI disks (sd_mod) and SCSItape drives (st).

Partitioning, file systems, mounting

Once you can access your FCP devices, you may partition the disks (if desired) and create file systems. It is not necessary to partition a disk if you want to use the whole disk as a single “partition”; however, partitioning with **fdisk** (even for a single partition using the whole disk) might be recommended. Typical devfs names, for example, assume that at least one partition is defined. Sample commands are:

```
# fdisk /dev/scsi/host0/bus0/target1/lun0/disc
   (use the normal fdisk commands to partition it, just like a PC disk)
   (assume you create two partitions)
# mke2fs /dev/scsi/host0/bus0/target1/lun0/part1
# mke2fs /dev/scsi/host0/bus0/target1/lun0/part2
# mkdir /mydisk1
# mkdir /mydisk2
# mount /dev/scsi/host0/bus0/target1/lun0/part1 /mydisk1
# mount /dev/scsi/host0/bus0/target1/lun0/part1 /mydisk2
# ls -al /mydisk1
    (you should see the dot and dot-dot entries)
```

You are not required to create a file system in a partition. It is possible to use *raw* access, but this tends to be dependent on particular applications.

We found that **fdisk** would not overwrite a partition map created by AIX. We removed any existing partition information with the command:

```
# dd if=/dev/zero of=/dev/scsi/host0/bus0/target1/lun0 bs=512 count=10
```

This effectively destroys the contents of the first few sectors of the physical drive, and **fdisk** then functioned correctly to partition the disk.

fdisk details

The user interface for **fdisk** is rather primitive, but it does the necessary job. The program is started with the command:

```
# fdisk /dev/scsi/host0/bus0/target1/lun0/disc
```

Make certain you have the correct target and LUN number, corresponding to your FCP map. The low-order name will always be disc, and this means the whole LUN is being accessed.

An **m** command, within **fdisk**, lists a menu of possible commands:

```
a  toggle a bootable flag
b  edit bsd disklabel
c  toggle the DOS compatibility flag
d  delete a partition
l  list known partition types
m  print this menu
n  add a new partition
o  create a new empty DOS partition table   (i.e., remove all existing partitions)
p  print the current (in-memory) partition table
q  quit without saving changes
s  create a new empty Sun disklabel
t  change a partition's system id
u  change diskpay/entry units
v  verify the partition table (and list amount of free space)
w  write the table to disk and exit
x  extra functionality (for experts)
```

You will typically use only the **m**, **n**, **p**, and **w** functions.

You can create a maximum of four primary partitions. Instead of one of these, you can create an extended partition. Within the extended partition you can create logical partitions. You can create a maximum of 16 partitions.

For practical purposes, there is no difference between primary and logical partitions. If you need only one or two partitions, we suggest using primary partitions. If you need more than this, we suggest making the whole disk into an extended partition and then creating logical partitions (up to 16) within this extended partition. The **n** function is used to create primary, extended, and logical partitions.

Once you complete the **fdisk** functions, you can use the following command to list the partition names on the LUN:

```
# ls /dev/scsi/host0/bus0/target1/lun1
```

You need to specify the correct host, target, and lun numbers in the command, of course.

Our disk partitions

We partitioned our disks as follows:

```
First SCSI drive (4GB):                  one partition for the whole drive
Second SCSI drive (9GB):                 two partitions, each 4.5 GB
First logical drive on FAStT200 (25GB):  one partition for the whole drive
Second logical drive on FAStT200 (25GB): one partition for the whole drive
Third logical drive on FAStT200 (25GB):  one partition for the whole drive
Fourth logical drive on FAStT200 (25GB): one partition for the whole drive
```

These choices were appropriate for the few programs we wanted to use. Your partitioning requirements are likely to be quite different. Notice that only one drive has more than a single partition.

mke2fs details

The **mke2fs** command creates a normal Linux file system in a partition; see the following example:

```
# mke2fs -m0 /dev/scsi/host0/bus0/target1/lun1/part1
```

The -m option may not be familiar. By default, **mke2fs** reserves 5% of the file system disk space such that only *root* (UID=0) can write in it. The -m option is used to specify a different

percentage of reserved space. The -m0 that we used prevents the creation of any reserved space.

Mount points

Trivial mount points, such as are shown in the examples above (/mydisk1 and /mydisk2), are not appropriate for anything more than initial testing. Even a small SAN network may have dozens of disk partitions that need mount points. You can devise any naming scheme you like, but it should be orderly. We created a number of mount points like this:

```
/mnt/lunxxyyzz
        | | +---the partition number
        | +-----the lun number
        +-------the target number

For example:
    /mnt/lun010001     first partition on first old scsi disk
    /mnt/lun010101     first partition on second old scsi drive
```

These mount point names are not intuitively readable, but they are orderly. If you can devise an orderly naming scheme that is also intuitive, that would be better. The following example is more intuitive, but assumes a well-planned usage pattern:

```
/mnt/Bill
/mnt/Patti
```

There is no requirement to place the mount points in /mnt.

More shell scripts

We created two more shell scripts in /etc: **fcpmount** and **fcpumount**:

```
/etc/fcpmount:
   #!/usr/sh
   mount /dev/scsi/host0/bus0/target1/lun0/part1  /mnt/lun010001
   mount /dev/scsi/host0/bus0/target1/lun1/part1  /mnt/lun010101
   mount /dev/scsi/host0/bus0/target1/lun1/part2  /mnt/lun010101
   mount /dev/scsi/host0/bus0/target2/lun0/part1  /mnt/lun020001
   mount /dev/scsi/host0/bus0/target2/lun1/part1  /mnt/lun020101
   mount /dev/scsi/host0/bus0/target2/lun2/part1  /mnt/lun020201
   mount /dev/scsi/host0/bus0/target2/lun3/part1  /mnt/lun020301

/etc/fcpumount:
   #!/usr/sh
   umount /dev/scsi/host0/target1/lun0/part1
   umount /dev/scsi/host0/target1/lun1/part1
   umount /dev/scsi/host0/target1/lun1/part2
   umount /dev/scsi/host0/target2/lun0/part1
   umount /dev/scsi/host0/target2/lun1/part1
   umount /dev/scsi/host0/target2/lun2/part1
   umount /dev/scsi/host0/target2/lun3/part1
```

These **mount** and **umount** commands match the partitions we created on our drives. The **umount** commands are needed because the zfcp driver cannot be uninstalled while any fcp device is mounted. We could make the **mount** commands part of the **fcpgo** script, but we found that a separate script was more convenient in an experimental environment.

Using the shell scripts

We used our shell scripts in this order:

```
boot the Linux system
run the fcpgo script to make the fcp devices available
run the fcpmount script to mount the fcp partitions for use
--- use the system -----
if we need to change the fcp configuration, run the fcpumount script
edit the fcpgo script, making whatever changes are needed
run the fcpgo script to install our new fcp map
run the fcpmount script to mount the fcp partitions
-----use the system-----
before shutting down Linux, run the fcpumount script
```

The **umount** commands may fail if any Linux processes are using partitions that are targets of **umount**. You may need to find and stop these processes. The **fcpumount** script can be run as many times as necessary.

Stable environment

Once our FCP environment was stable, we wanted to run our shell scripts automatically during boot and shutdown. We did this by moving the scripts to /etc/init.d and adding pointers to them in the /etc/rc3.d directory, using names such as S86fcpgo, S87fcpmount, and K86fcpumount. (Linux administrators use various methods for customizing their startup functions, and we intentionally omit cookbook-style instructions for doing this.

Our changes cause our **fcpgo** and **fcpmount** scripts to be run whenever the system enters runlevel 3.[4] The **fcpumount** script is run whenever the system leaves runlevel 3. Our implementation is rather primitive, but it illustrates a simplistic method of automating the connection to FCP devices.

Make a chart

We found the device names, maps, and correspondence with “real” devices could quickly become confusing, even in the very small environment we built. Using a chart such as Table 3 helped deal with the confusion. We kept several copies of this chart posted in convenient places.

Table 3 Chart for mapping devices

Source	host	target	lun	part	GB	fs	Mountpoint	Usage
1st SCSI drive	0	1	0	1	4.5	Y	/mnt/lun010001	scratch
2nd SCSI drive p1	0	1	1	1	4.5	Y	/mnt/lun010101	
2nd SCSI drive p2	0	1	1	2	4.5	Y	/mnt/lun010102	Bill’s area
1st logical FAStT drive	0	2	0	1	25	Y	/mnt/lun020001	
2nd logical FAStT drive	0	2	1	1	25	Y	/mnt/lun020102	
3rd logical FAStT drive	0	2	2	1	25	Y	/mnt/lun020201	
4th logical FAStT drive	0	2	3	1	25	Y	/mnt/lun020301	SAMBA

[4] Runlevel 3 is the “normal” operational level of Linux.

FCP map alternations

In the following sections, we discuss various FCP map alterations, including changing the map and listing definitions.

Map input

The FCP maps can be passed to Linux in a number of ways, and these are documented in the formal driver documentation. We elected to simply change the FCP map in our shell script and then reload the whole map. This disrupts any Linux processes using the driver and the FCP devices.

In a limited environment, this is easy and acceptable. In a larger environment, this disruption might not be acceptable. A less disruptive method is to add or delete individual FCP map entries.

Changing the map

You can install a new map entry to a running system by writing it to /proc/scsi/zfcp/add_map. This can be done with an **echo** command that is piped to /proc/scsi/zfcp/add_map. We suggest it would be better to enter the new map line(s) in a file, and then **cat** the file to add_map. See the following example:

```
    create file "changescsi" (in the current directory) with your new map

# cat changescsi > /proc/scsi/zfcp/add_map
```

With the current release, you cannot add (or delete) a single line in the existing map—you can only replace the whole map. We did not try this interface, since our **fcpgo** script accomplishes the same thing. We do not know whether the add_map interface requires **umount**ing file systems resident on the FCP devices before invoking the interface.

Listing map and SCSI device definitions

The command **cat /proc/scsi/scsi** will list the identity of all your attached SCSI devices. FCP devices are seen by Linux to be SCSI devices. The command **cat /proc/scsi/zfcp/map** will list the currently active FCP map.

SCSI definitions

In addition to the FCP maps for accessing devices, you must also define these devices to the SCSI driver. This is done automatically by the zfcp driver when it is started. If you add FCP drives after zfcp is started, you must manually define them to the SCSI driver.This is done by sending data to /proc/scsi/scsi. See the following example:

```
# echo "scsi add-single-devices 0 0 1 1" > /proc/scsi/scsi
```

Alternatively, you can build a file of these data lines and **cat** it to /proc/scsi/scsi:

```
    create file "definescsi" in the current directory, with lines such as:
       scsi add-single-device 0 0 1 0
       scsi add-single-device 0 0 1 1
       scsi add-single-device 0 0 1 2

# cat definescsi > /proc/scsi/scsi
```

The four numeric parameters, in order, are: the host adapter number, the bus number (always zero), the target number, and the LUN number. A remove-single-device function also exists, with the same arguments.

Special handling

Open and reopen

Once Linux recognizes a SCSI logical unit, that unit is *open* at the device level. The open state is independent of any Linux programs that may be used. If you make hardware or FCP network changes, you may want to force Linux to close and reopen the logical unit. The following command can be used, where `reset erp` is a fixed keyword:

```
# echo "reset erp" > /proc/scsi/zfcp/devno0x0600/id0x1/lun0x0/status
```

In this example, 0600 is the device number used in the FCP map, the SCSI target id is 1, and the LUN number is 0.

If you remove the fcp driver and reload it (as we do in our shell script), this automatically closes and reopens the LUNs.

Device offline

To take a logical unit offline, you remove the corresponding FCP map entry as described in “Changing the map” on page 36, or by using the shell scripts we presented. This will close the unit properly and make it inaccessible.

General usage

Once an FCP-attached LUN is mapped, partitioned, formatted, and mounted, it is used the same way as any Linux disk. A Linux application will detect no difference between these drives and “normal” drives.

Shared disk access

Traditional zSeries users are accustomed to *shared DASD* operation, usually under z/OS. It may appear that similar shared disk operation is possible with FCP-attached units. Many hosts can be attached to the FCP network, and all the hosts can potentially access all the devices in the network. (Various zoning and LUN masking controls can be used as security mechanisms, but security controls and shared disk access are different topics.) Linux, like UNIX, does not generally support shared disk access by multiple systems.

Using the same FCP CHPID across multiple Linux images for shared access to a LUN is currently not supported. However, shared access to a LUN using different CHPIDs (on the same or different hosts) is possible.

This same restriction applies to VM guests or LPARs that share a single FCP channel. The Fibre Channel standard (FCP and FCP2) has no provisions for supporting host systems where multiple operating systems are executed concurrently, and where two or more of these operating systems share access to a single Fibre Channel adapter at the same time.

In the case of zSeries machines, a single FCP *channel* can be shared by multiple LPARs or by multiple VM guests, but these operating systems must not attempt to share a single *device* (as specified by a WWPN and LUN pair). Doing so could lead to timeouts and other problems. Note that a device can be shared (if this makes sense for the particular device) if it can be accessed by different channel paths (different CHPIDs) for each user.

Note: For FCP devices to be accessed by an operating system, they are typically specified by an 8-byte (16 hexadecimal digits) LUN in a specific configuration file, such as the Linux FCP map file. It must be noted, however, that some devices only regard a subset of the LUN as the device address. For example, only the four high-order hexadecimal digits are recognized, and the remaining digits are ignored. Therefore, different 8-byte LUN addresses which have the same 4 high-order hexadecimal digits would lead to access of the same device.

We strongly recommend to never use different LUN addresses for the same device in such a case. Otherwise, shared access to a device may happen without being detected, and unpredictable results can occur.

Tape drive usage

We used our SCSI-attached DLT tape drive without problems. We suggest you see the **man** pages for **mt** before using a SCSI tape drive. An example of drive use follows:

```
# man mt                                          (read about a useful command)
   (read it)
   (place a tape in the drive)
# mt -f /dev/tapes/tape0/mt status                (try the command)
SCSI 2 tape drive
File number=0, block number=0, partition=0
Tape block size=0 bytes,  Density code 0x1b (unknown to this mt)
Soft error count since last status = 0
General status bits on (41010000)
  BOT ONLINE IM_REP_EN
# export TAPE=/dev/tapes/tape0/mt                 (make it easier to use mt)
# mt status                                       (issue the same status request)
   (same report as shown above)
# mt offline                                      (this unloads the tape drive)
  (reload the tape manually)
# dd if=/etc/fcpgo of=/dev/tapes/tape0/mt bs=1024      (write a file to tape)
0+1 records in
0+1 records out
# mt rewind
# dd if=/dev/tapes/tape0/mt                       (read the file back from tape)
  (our /etc/fcpgo file is listed on stdout)
0+1 records in
0+1 records out
# mt rewind
```

We could use /dev/scsi/host0/bus0/target1/lun2/mt instead of /dev/tapes/tape0/mt, but the second name is easier to type. The specific address of your tape drive (target and lun, or tape) may differ from our example, of course.

Security controls

Security is a large topic and generally beyond the scope of this paper. However, we can make a key point relative to FCP devices:

> *Controls are available within the FC network to limit access to specific network-attached devices. The FC switches have* zone *controls, and the Data Gateways and controllers have a masking (partition) function. A critical concept is that these controls affect only which zSeries FCP channel (host machine) can connect to which network devices. These controls cannot detect which user (or Linux image) within a host machine is using the device.*

Security and access elements include the following:

- FC node *masking parameters* (if implemented) can control which host port (FCP channel) can access which LUNs on that node.
  - The control is by *FCP channel*. A single host may have several FCP channels attached to the network.
  - Multiple hosts may be attached to the network, each with one or more FCP channels.
  - The masking controls cannot distinguish between multiple channels to a single host, or channels to multiple hosts.
  - The masking controls cannot distinguish between different users within the host.
  - This *masking* is sometimes known as LUN *partitioning*. Do not confuse this partitioning with the partitioning of a disk.
- SAN *zone* parameters (in the FC switches) control which network devices can be accessed from a zSeries FCP channel. (We ignore the Data Gateway partitioning function for the remainder of this discussion.) If zone controls are used, we must assume they are trusted.
  - The control is by *FCP channel*. A single host may have several FCP channels attached to the network.
  - Multiple hosts may be attached to the FC network, each with one or more FCP channels.
  - The zone controls cannot distinguish between multiple channels to a single host, or channels to multiple hosts.
  - The zone controls cannot distinguish between different users within the host.
- The zSeries IOCDS controls which logical partitions can use which device numbers. The IOCDS assigns each FCP channel a range of device numbers.
- Any of the device numbers assigned (through IOCDS) to an FCP channel can be used (via an FCP map within Linux) to access any network device accessible to that channel.
  - This is limited by the restriction that any of the 240 paths through the FCP CHPID cannot be shared by multiple Linux images.
- Within a logical partition (“LPAR”), the machine-level operating system determines which users can access which device numbers. For this discussion, the operating system is either Linux or z/VM.
- Device numbers for FCP devices are assigned by the FCP maps within a Linux image. (A Linux image requires access to a real device number through z/VM and the IOCDS in order to use the device number.)
- FCP maps are in Linux, not in z/VM.
- If multiple Linux guests are running under z/VM, then z/VM can further restrict which Linux guest can access which device numbers. Remember, however, that any device number associated with a given FCP channel can access any network device that can be accessed by that channel.
- Linux controls which users can access logical devices through privileged users (*root* or *superuser*) and permission bits.

The highest level of security control might be seen as host control. If more than one host system (such as a zSeries machine) can be connected to the network, then either (1) all hosts must be trusted, or (2) FC zone controls must be used to limit host access. Remember that a “host” can be a simple PC with an FC adapter. If an untrusted host can connect to the FC network, other host owners must deal with security issues at the FC fabric level (probably by using zone controls).

IOCDS control for a zSeries machine is usually considered a trusted element. If separate FCP channels are used to access separate network devices, then IOCDS parameters can

control which logical partitions are allowed to use which FCP channels. Different FCP channels might be connected to different FC networks (unlikely), or managed as different zones by parameters within the FC switches (more likely). If only a few FCP channels are available, then controls at this level are very coarse.

Another key concept is whether an operating system is *trusted* by the installation owner. Within a trusted Linux system, restricted root access and appropriate permission bits can be relied upon to control file and device access by users. Using a trusted Linux system, an installation can control access to FCP-attached devices just as it controls access to other files and devices.

z/VM is usually managed as a trusted system, and it can be used to control which z/VM guests (Linux images) can access which FCP device numbers. This equates (via IOCDS definitions) to controlling which guests can access which FCP channels. If you have only a small number of FCP channels and a large number of Linux images, the control available at the z/VM level is a very coarse control.

An *untrusted* operating system can run natively in an LPAR or as a guest under z/VM. It is unlikely that a completely untrusted operating system would be allowed to run natively in an LPAR, because the owner must be permitted access to a Support Element (SE) or a Hardware Management Console (HMC) to load the operating system.

Containing the discussion to Linux (and Linux under z/VM), an untrusted system is one where root access (or the equivalent) is available to untrusted users. A user with root access can change the FCP device map at any time. Remember that the FCP map contains the device numbers to be used for network access, the WWPN of a network device, and a WWN-like number for LUNs within that device.

Stated a different way, the only controls between an untrusted Linux (or many untrusted Linux images) under z/VM and network device access are these:

- z/VM can control which real FCP device numbers are available to the images. However, any device number associated with an FCP channel can potentially access any network device on that FCP channel. In practice, this level of control is useful only if several FCP channels are available.
- FC fabric controls (zone parameters in the FC switches, masking parameters in FC nodes) control which FCP channels (which may be on one, or several, hosts) can connect to which devices (ports) in the network.
- A given device number can be used by only one Linux image. Later attempts by other images to use this device number will fail.

A parallel situation exists with an untrusted Linux running natively in an LPAR. In this case, IOCDS parameters can control which FCP channel(s) the LPAR can access. In turn, the FCP channels can access only devices permitted by the zone controls in the FC fabric.

Integrity issue

Security and integrity overlap in many areas. Inadvertent sharing of read-write network disks (LUNs) by multiple Linux images can lead to corrupted data. The multiple Linux images can be on the same host (LPARs or z/VM), or on a mixture of hosts. There is no automatic mechanism in the FC fabric to detect this situation.

Summary

In typical environments, with a few zSeries FCP channels connected to a SAN fabric, potential security controls are very limited. This is not a problem if only trusted hosts and trusted operating systems are used, with appropriate planning for potentially shared network LUNs. Any other environment requires careful planning.

Planning for larger environments

This Redpaper discusses a small FCP environment, and is intended for initial users of zSeries FCP facilities. The environment discussed is practical for many purposes, but it ignores issues that must be considered for larger environments. These issues include the following:

- Consistent Linux addressing (“naming”) of LUNs when using large numbers of Linux images. This can be a complex problem and *must* be considered before starting implementation of a larger environment.
- How to add and delete LUNs without disturbing other Linux images and without altering the addressing of existing LUNs.
- Addressing practical integrity and security concerns, given the basic controls provided by an FC network; for example:
  - How to create zone controls for our FC switch.
  - How to create LUN masking for our Data Gateway and FAStT200.
- Managing shared read-only LUNs.
- Using NFS to implement limited read-write sharing of LUNs.
- Setup of other switches and gateway devices.
- Use of more complex devices, such as IBM’s Enterprise Storage Server (“Shark”).
- Managing device numbers to avoid confusion and errors.
  - Concerns when multiple LPARs are used.
  - Concerns when many Linux images are used under z/VM.
- Usage of different type drives, such as SCSI-connected 3480/90-compatible units.

Acknowledgements

We thank the following people for their contributions to this project:

Roy Costa
International Technical Support Organization, Poughkeepsie Center

Bob Haimowitz
International Technical Support Organization, Raleigh Center

Stefan Mueller, Raimund Schroeder, Aron Zeh
IBM Germany

Christoph Arenz, George Kuch, Carlos Ordonez, Paul Wojciak
IBM Poughkeepsie

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Send us your comments in one of the following ways:

- Use the online **Contact us** review redbook form found at:
  ibm.com/redbooks
- Send your comments in an Internet note to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

CUA®	FICON™	StorWatch™	Redbooks(logo)™ 
ECKD™	IBM®	VTAM®	
Enterprise Storage	Redbooks™	z/OS™	
Server™	S/390®	z/VM™	
ESCON®	SP™	zSeries™	

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.