NFS & iSCSI: Linux Network Storage

Designing, Deploying, and Securing Network-Based Storage Infrastructure on Linux

Preface

When I first began working with iSCSI in production Linux environments, I was struck by how few resources existed that treated it with the depth and seriousness it deserved. iSCSI—Internet Small Computer Systems Interface—has quietly become one of the most powerful and cost-effective storage technologies available to Linux administrators, yet it often lives in the shadow of more glamorous solutions. This book was written to change that.

Why This Book Exists

iSCSI enables block-level storage access over standard TCP/IP networks, eliminating the need for expensive Fibre Channel infrastructure while delivering the performance and flexibility that modern data centers demand. Whether you're building a virtualization cluster, architecting shared storage for a database tier, or simply looking to consolidate storage across your Linux environment, iSCSI is likely at the center of your solution—or it should be.

NFS & iSCSI: Linux Network Storage was conceived as a comprehensive, practical guide to designing, deploying, and securing network-based storage infrastructure on Linux, with **iSCSI as its central focus**. While NFS is covered thoroughly as an essential companion technology for file-level access, the heart of this book beats around iSCSI: understanding its architecture, mastering its configuration, hardening its security, and troubleshooting its failures in real-world environments.

What You Will Learn

This book is structured to take you on a deliberate journey. We begin with foundational concepts—understanding network storage models and how iSCSI fits within the broader landscape of block and file storage protocols. From there, we move into hands-on territory: configuring iSCSI targets and initiators, connecting clients to shared block storage, and integrating iSCSI volumes with LVM for flexible volume management.

Security receives dedicated attention because iSCSI, by its nature, transmits storage traffic over networks that may be shared with other services. Chapters on **securing iSCSI storage** cover CHAP authentication, network segmentation, IPsec, and the critical design decisions that separate a lab experiment from a production-ready deployment.

The book also addresses the environments where iSCSI truly shines: **virtualized infrastructures**. Whether you're using KVM, Proxmox, or other hypervisors, you'll learn how iSCSI serves as the backbone for shared storage in clustered and high-availability configurations.

How This Book Is Organized

The sixteen chapters and five appendices are arranged in a logical progression:

- **Chapters 1–2** establish the conceptual foundation for network storage, with emphasis on where iSCSI fits in.
- **Chapters 3–6** cover NFS server and client configuration, security, and performance tuning.
- **Chapters 7–10** form the core of the book, delivering deep, hands-on coverage of **iSCSI targets, initiators, connectivity, and security**.

- **Chapters 11–12** explore integration topics—LVM and virtualization—where iSCSI's block-level capabilities are most impactful.
- **Chapters 13–14** provide systematic troubleshooting methodologies for both NFS and iSCSI.
- **Chapters 15–16** elevate your perspective from tactical configuration to strategic design and career growth.
- **Appendices A–E** offer quick-reference materials, including an **iSCSI command reference**, security checklists, troubleshooting flowcharts, and a guide to building a career in Linux storage engineering.

Who This Book Is For

If you are a Linux system administrator, DevOps engineer, or infrastructure architect who needs to implement reliable, secure, network-attached storage—and particularly if iSCSI is part of your current or future architecture—this book was written for you. No prior storage specialization is required, only a working familiarity with Linux administration.

Acknowledgments

This book would not exist without the open-source communities that have built and maintained the Linux iSCSI target frameworks, `open-iscsi`, LVM, and the countless tools that make Linux the premier platform for network storage. I am also deeply grateful to the system administrators and storage engineers whose real-world questions, war stories, and hard-won lessons shaped every chapter.

Storage is infrastructure. Infrastructure is trust. Let's build something worth trusting.

Bas van den Berg

Table of Contents

Chapter 1: Understanding Network Storage Models

Network storage has fundamentally transformed the way organizations manage, access, and protect their data. In the early days of computing, storage was a simple affair: a hard disk physically attached to a server, accessible only to that single machine. As networks grew in complexity and businesses demanded more flexibility, the limitations of this approach became painfully apparent. The need to share data across multiple servers, to centralize management, and to scale storage independently from compute resources gave rise to an entirely new discipline of network-based storage. Among the most important technologies in this space is iSCSI, the Internet Small Computer Systems Interface, which has democratized access to enterprise-grade storage by leveraging the ubiquitous TCP/IP network infrastructure that already exists in virtually every organization.

This chapter lays the groundwork for understanding where iSCSI fits within the broader landscape of network storage. Before diving into configuration files and command-line tools, it is essential to develop a clear mental model of the different storage architectures, understand the terminology, and appreciate the design decisions that make iSCSI such a compelling choice for Linux-based storage infrastructure.

The Evolution of Storage: From Local Disks to Network Storage

To appreciate the significance of iSCSI, one must first understand the problem it solves. In a traditional computing environment, each server has its own set of locally attached disks. The operating system communicates with these disks through a bus interface, most commonly SCSI (Small Computer Systems Interface) or its successors like SAS (Serial Attached SCSI). This communication happens at the block level, meaning the operating system sends raw read and write commands to specific locations on the disk surface. The operating system's file system layer then organizes these blocks into files and directories.

This model works perfectly well for a single server, but it introduces serious challenges at scale. Consider a data center with fifty servers, each with its own local storage. If one server runs out of disk space while another has terabytes sitting idle, there is no straightforward way to redistribute that capacity. If a server fails, the data on its local disks may become inaccessible until the hardware is repaired. Backup processes must run independently on each machine. Provisioning a new server requires physically installing disks, a process that can take hours or days.

Network storage emerged as the answer to these challenges. By separating storage from the server and connecting them over a network, organizations gained the ability to pool storage resources, share them dynamically, manage them centrally, and protect them more effectively. Two primary models of network storage emerged: file-level storage and block-level storage. Understanding the distinction between these two models is absolutely critical to understanding why iSCSI exists and what makes it different from technologies like NFS.

File-Level Storage versus Block-Level Storage

File-level storage, exemplified by NFS (Network File System) and CIFS/SMB (Common Internet File System/Server Message Block), operates at the file system layer. When a client accesses a file over NFS, it sends requests like "open this file," "read 4096 bytes starting at offset 8192," or "create a new directory." The storage server receives these requests, translates them into local file system operations, and returns the results. The client never directly interacts with the underlying disk blocks. The server's file system handles all the details of where data physically resides on disk.

Block-level storage operates at a fundamentally lower layer. Instead of requesting files by name, the client sends raw SCSI commands to read and write specific blocks of data on a storage device. The client's operating system treats the remote storage device as if it were a locally attached disk. The client runs its own file system on top of this block device, making all decisions about how to organize data into files and directories. The storage server, often called a storage target, simply receives block-level I/O commands and executes them against its physical disks.

This distinction has profound implications. With block-level storage, the client has complete control over the file system. It can format the device with any file system it chooses, whether ext4, XFS, Btrfs, or even a raw database tablespace. It can use the device as a swap partition, as a physical volume for LVM (Logical Volume Manager), or as a member of a software RAID array. None of this is possible with file-level storage, where the server dictates the file system and the client must work within its constraints.

iSCSI is a block-level storage protocol. It takes the venerable SCSI command set, the same set of commands that operating systems have used to communicate with local disks for decades, and encapsulates those commands inside TCP/IP

packets for transmission over a standard Ethernet network. This is the core insight of iSCSI: rather than requiring specialized hardware and cabling, it reuses the existing network infrastructure to deliver block-level storage access.

The following table summarizes the key differences between file-level and block-level network storage:

Characteristic	**File-Level Storage (NFS)**	**Block-Level Storage (iSCSI)**
Protocol Layer	Application layer (file operations)	Block layer (SCSI commands over TCP/IP)
Client Perspective	Sees shared directories and files	Sees raw block devices (like local disks)
File System Control	Server controls the file system	Client controls the file system
Typical Use Cases	Shared home directories, web content, general file sharing	Databases, virtual machine disk images, boot devices, applications requiring raw block access
Multi-Client Access	Natively supports multiple concurrent clients	Typically single-client access per LUN unless clustered file system is used
Network Requirements	Standard TCP/IP Ethernet	Standard TCP/IP Ethernet
Performance Sensitivity	Moderate latency tolerance	More sensitive to latency due to block-level operations
Flexibility	Limited to file operations	Full control over file system, partitioning, LVM, and RAID

The Three Storage Architectures: DAS, NAS, and SAN

Network storage is commonly categorized into three architectural models: Direct Attached Storage (DAS), Network Attached Storage (NAS), and Storage Area Network (SAN). Understanding where iSCSI fits among these architectures is essential for making informed design decisions.

Direct Attached Storage (DAS) is the simplest model. Storage devices are physically connected to a single server through a local interface such as SATA, SAS, or USB. There is no network involved. DAS is inexpensive and offers low latency, but it lacks the flexibility and shareability of network storage. Every server manages its own storage independently, leading to the capacity imbalance and management overhead problems described earlier.

Network Attached Storage (NAS) is a file-level storage architecture. A NAS device is a specialized server that connects to the data network and presents shared file systems to clients using protocols like NFS or SMB. Clients mount these shared file systems and access files using standard file operations. NAS is excellent for scenarios where multiple clients need to access the same set of files simultaneously, such as shared home directories, media libraries, or collaborative workspaces.

Storage Area Network (SAN) is a block-level storage architecture. A SAN creates a dedicated network specifically for storage traffic, separate from the regular data network. Storage devices on the SAN present raw block devices (called LUNs, or Logical Unit Numbers) to servers, which then format and use these devices as if they were local disks. Traditionally, SANs were built using Fibre Channel, a high-speed, low-latency networking technology purpose-built for storage. Fibre Channel SANs deliver exceptional performance but require specialized switches, host bus adapters (HBAs), and cabling, all of which carry significant cost.

This is precisely where iSCSI enters the picture and changes the economics of block-level storage dramatically. iSCSI enables organizations to build a SAN using their existing Ethernet network infrastructure. Instead of Fibre Channel switches costing tens of thousands of dollars, organizations can use standard Ethernet switches. Instead of expensive Fibre Channel HBAs, servers use their existing network interface cards (NICs). Instead of specialized Fibre Channel cabling, standard Cat6 or Cat6a Ethernet cables carry the storage traffic. The result is a SAN that delivers block-level storage access at a fraction of the cost of a traditional Fibre Channel SAN.

The following table compares these three architectures with a focus on how iSCSI relates to each:

Architecture	**Storage Type**	**Network**	**Protocol**	**Cost**	**iSCSI Relevance**
DAS	Block	None (local bus)	SATA, SAS, NVMe	Low	iSCSI can replace DAS by providing centralized block storage over the network
NAS	File	Standard Ethernet (TCP/IP)	NFS, SMB/CIFS	Moderate	iSCSI operates at a different layer; NAS and iSCSI can coexist on the same network

SAN (Fibre Channel)	Block	Dedicated Fibre Channel	FCP (Fibre Channel Protocol)	High	iSCSI provides equivalent block-level access at significantly lower cost
SAN (iSCSI)	Block	Standard Ethernet (TCP/IP)	iSCSI (SCSI over TCP/IP)	Moderate	This is iSCSI's primary role: an IP-based SAN

How iSCSI Works: A Conceptual Overview

At its heart, iSCSI is an elegantly simple idea: take SCSI commands and transport them over TCP/IP networks. However, the implementation involves several important concepts that must be understood before any practical deployment.

In the iSCSI architecture, there are two primary roles: the **initiator** and the **target**. The initiator is the client, the server that needs access to remote storage. It generates SCSI commands and sends them over the network. The target is the storage server that receives these commands, executes them against its physical storage, and returns the results. A single target can present multiple storage devices, each identified by a Logical Unit Number (LUN). A single initiator can connect to multiple targets, and a single target can serve multiple initiators.

Every iSCSI target and initiator is identified by a globally unique name called an iSCSI Qualified Name (IQN). The IQN follows a specific naming convention that includes the date of registration, the naming authority (typically a reversed domain name), and a unique identifier. For example:

```
iqn.2024-01.com.example.storage:target01
```

This naming convention ensures that every iSCSI device on any network anywhere in the world can be uniquely identified, which is essential for security and management purposes.

When an initiator wants to access storage on a target, it first performs a process called **discovery**. During discovery, the initiator contacts the target and requests a list of available targets and their addresses. Once the initiator knows what targets are available, it establishes a **session** with the desired target. A session is a TCP connection (or multiple TCP connections for performance) over which SCSI commands and data flow. Within a session, the initiator can access the LUNs presented by the target, and the operating system treats these LUNs as local block devices.

The following table outlines the key iSCSI terminology:

Term	Definition
Initiator	The client that sends SCSI commands over the network to access remote storage
Target	The server that receives SCSI commands and provides access to storage devices
LUN (Logical Unit Number)	A unique identifier for a specific storage device presented by a target
IQN (iSCSI Qualified Name)	A globally unique name assigned to each initiator and target
Portal	A combination of IP address and TCP port on which a target listens for connections (default port is 3260)
Discovery	The process by which an initiator locates available targets on the network
Session	An active connection between an initiator and a target over which SCSI commands are exchanged

CHAP (Challenge Handshake Authentication Protocol)	An authentication mechanism used to verify the identity of initiators and targets
TPG (Target Portal Group)	A set of network portals through which a target can be accessed

Why iSCSI on Linux

Linux has become the dominant operating system for server infrastructure, and its iSCSI support is both mature and comprehensive. On the initiator side, the **open-iscsi** package provides a robust, well-tested implementation that integrates seamlessly with the Linux kernel's SCSI subsystem. On the target side, the **LIO (Linux-IO)** target framework, which is part of the mainline Linux kernel, provides a feature-rich, high-performance iSCSI target implementation that rivals commercial alternatives.

The combination of Linux and iSCSI is particularly powerful for several reasons. First, Linux provides excellent tools for managing block devices, including LVM for flexible volume management, dm-crypt for encryption, and mdadm for software RAID. All of these tools work transparently with iSCSI-attached block devices, because the operating system treats them identically to local disks. Second, Linux's networking stack is highly optimized and supports features like jumbo frames, network bonding, and multiple network namespaces, all of which can be leveraged to optimize iSCSI performance. Third, the open-source nature of both Linux and its iSCSI implementations means there are no licensing costs, making it an extremely cost-effective solution.

Consider a practical scenario. A small company has ten Linux servers running various applications, including a database server, a mail server, and several web servers. Instead of managing local disks on each server independently, the compa-

ny deploys a single Linux server configured as an iSCSI target with a large RAID array. Each application server is configured as an iSCSI initiator and receives one or more LUNs from the target. The database server formats its LUN with XFS and uses it for database tablespaces. The mail server formats its LUN with ext4 and stores mailboxes on it. The web servers each receive their own LUNs for application data.

This centralized approach simplifies backup (only the target server needs to be backed up), improves utilization (storage capacity can be reallocated as needs change), and enhances reliability (the target server can use RAID for redundancy). All of this is achieved using standard Ethernet networking and open-source Linux software, at a fraction of the cost of a proprietary SAN solution.

Planning Considerations for iSCSI Deployments

Before deploying iSCSI in any environment, several important planning considerations must be addressed. Network design is paramount. Because iSCSI traffic shares the same Ethernet infrastructure as regular data traffic, it is essential to ensure that storage traffic does not compete with application traffic for bandwidth. Best practices include dedicating separate VLANs or even separate physical networks for iSCSI traffic, using jumbo frames (MTU 9000) to reduce CPU overhead and improve throughput, and configuring network bonding or multipathing for redundancy and performance.

Security is another critical consideration. By default, iSCSI traffic is not encrypted, and any device on the network could potentially discover and connect to iSCSI targets. CHAP authentication should always be configured to ensure that only authorized initiators can access storage. For environments with strict security require-

ments, IPsec can be used to encrypt iSCSI traffic, although this comes with a performance penalty.

Performance tuning is also important. iSCSI adds network latency to every storage I/O operation, which can impact performance-sensitive applications like databases. Techniques for optimizing iSCSI performance include using dedicated gigabit or 10-gigabit Ethernet links, enabling TCP offload features on network adapters, tuning TCP buffer sizes, and using multipath I/O (MPIO) to distribute traffic across multiple network paths.

The following table summarizes key planning considerations:

Planning Area	Consideration	Recommendation
Network Isolation	iSCSI traffic competing with data traffic	Use dedicated VLANs or separate physical networks for iSCSI
MTU Size	Standard 1500-byte frames increase CPU overhead	Configure jumbo frames (MTU 9000) on all iSCSI network components
Redundancy	Single point of failure in network path	Implement network bonding and multipath I/O (dm-multipath)
Authentication	Unauthorized access to storage targets	Configure CHAP authentication on all targets and initiators
Encryption	Data in transit is unencrypted by default	Use IPsec for sensitive environments
Bandwidth	Insufficient throughput for storage workloads	Use 10GbE or higher for production iSCSI deployments
Latency	Network latency impacts I/O performance	Minimize network hops and use low-latency switches

Exercises

The following exercises are designed to reinforce the concepts covered in this chapter and prepare you for the hands-on work in subsequent chapters.

Exercise 1: Identifying Storage Models

On a Linux system, use the `lsblk` command to list all block devices. Identify which devices are locally attached (DAS) and consider how these would appear differently if they were iSCSI-attached devices.

```
lsblk -o NAME,TYPE,SIZE,TRAN,MOUNTPOINT
```

Note: The `TRAN` column shows the transport type. Local SATA disks will show `sata`, while iSCSI devices will show `iscsi` once configured in later chapters.

Exercise 2: Examining SCSI Subsystem

Use the following command to examine the SCSI devices currently recognized by the Linux kernel:

```
cat /proc/scsi/scsi
```

Alternatively, use:

```
lsscsi
```

Note: If `lsscsi` is not installed, install it with `yum install lsscsi` on RHEL-based systems or `apt install lsscsi` on Debian-based systems. This command will become invaluable when verifying that iSCSI LUNs have been successfully discovered and attached.

Exercise 3: Network Readiness Assessment

Evaluate your network's readiness for iSCSI by checking the current MTU settings and network interface capabilities:

```
ip link show
cat /sys/class/net/eth0/mtu
```

```
ethtool eth0 | grep -i "speed\|link detected\|offload"
```

Note: The output of these commands will tell you the current link speed, whether jumbo frames are configured, and what offload capabilities your network adapter supports. All of these factors directly impact iSCSI performance.

Exercise 4: Conceptual Design

On paper or in a text document, design a simple iSCSI storage infrastructure for a small office with five servers. Define the following elements:

1. The IQN for the iSCSI target server
2. The IQNs for each of the five initiator servers
3. The number and size of LUNs to be created
4. The network configuration, including VLAN assignments and IP addressing
5. The authentication method to be used

This exercise forces you to think through the planning considerations discussed in this chapter before any actual configuration takes place.

Exercise 5: Comparing Protocols

Research and document the differences between iSCSI, Fibre Channel, and FCoE (Fibre Channel over Ethernet). Create a comparison table that includes the following attributes: transport medium, typical bandwidth, latency characteristics, cost, complexity, and Linux support. This exercise will deepen your understanding of where iSCSI fits in the broader storage ecosystem.

Summary

This chapter has established the foundational understanding necessary for working with iSCSI on Linux. We explored the evolution from locally attached storage to

network storage, examined the critical distinction between file-level and block-level storage protocols, and positioned iSCSI within the three major storage architectures of DAS, NAS, and SAN. The key takeaway is that iSCSI brings the power of block-level SAN storage to standard Ethernet networks, eliminating the need for expensive specialized hardware while providing the same fundamental capabilities. With Linux's mature and comprehensive iSCSI support through open-iscsi and LIO, organizations of any size can deploy enterprise-grade storage infrastructure using open-source tools and commodity hardware. The concepts, terminology, and planning considerations covered in this chapter will serve as the foundation for every subsequent chapter, where we will move from theory to practice, configuring real iSCSI targets and initiators on Linux systems.