Linux/iSCSI and a Generic Target Mode Storage Engine for Linux v2.6

Nicholas A. Bellinger
Linux-iSCSI.org

2005-02-25

Lecture Outline

- ISCSI Introduction
- ISCSI basics
- ISCSI advanced features
- Linux/iSCSI Target projects status
- LIO-SE and LIO-Target design
- The case for a generic target mode storage engine in Linux
- Kernel vs. User and the Target storage engine
- Other Linux-iSCSI.org Projects
- Questions and Thanks

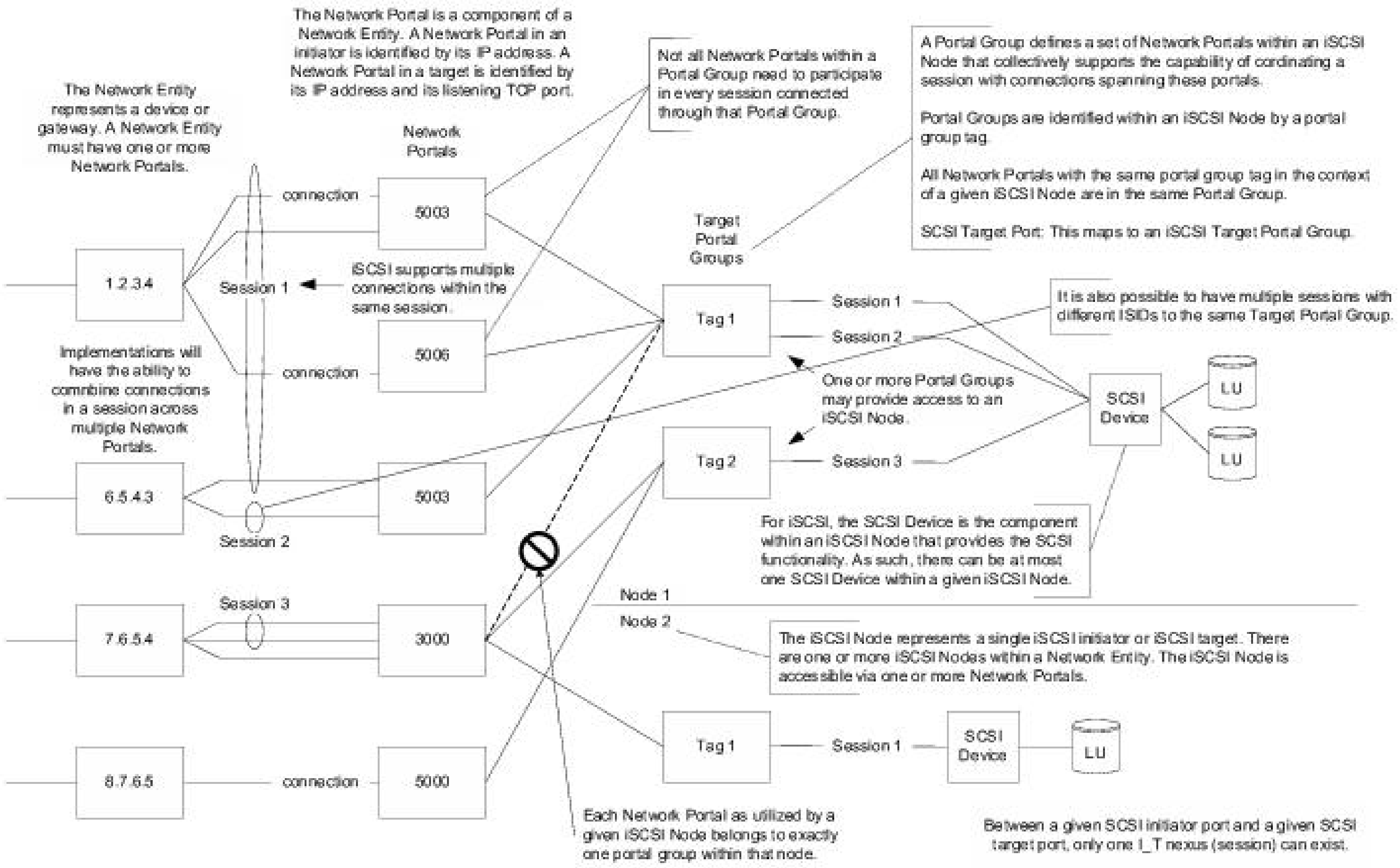
ISCSI Introduction

- ISCSI interest is growing..
  - Use within virtualization environments
  - Use within clusters that depend upon shared storage
  - Use within high end database deployments (will become real alternative with iSER and 10 Gb/sec)
  - Large and small customers are coming to Linux-ISCSI.org at an increasing rate for code and information.
- At the same time..
  - There are no less than four (4) iSCSI Target Projects on Linux.
  - All are out of tree, with the expection of STGT (userspace)
  - Some share iSCSI target code, while focusing development on target mode storage engine.

ISCSI: Basics

- Mapping of SCSI Architecture Model to IP
- Basic In-band discovery for small deployments
- Authentication via CHAP for small deployments
- Single communicaton path between Initiator and Target (SC/S)
- ISCSI provides Command Sequence Number (CmdSN) ordering to ensure delivery of tasks from Initiator to Target Port.
- Targets provide Network Portals, Target Portal Groups, and SCSI Target Port Endpoints..

Network Entity



The Network Entity represents a device or gateway. A Network Entity must have one or more Network Portals.

The Network Portal is a component of a Network Entity. A Network Portal in an initiator is identified by its IP address. A Network Portal in a target is identified by its IP address and its listening TCP port.

Not all Network Portals within a Portal Group need to participate in every session connected through that Portal Group.

A Portal Group defines a set of Network Portals within an iSCSI Node that collectively supports the capability of cordinating a session with connections spanning these portals.

Portal Groups are identified within an iSCSI Node by a portal group tag.

All Network Portals with the same portal group tag in the context of a given iSCSI Node are in the same Portal Group.

SCSI Target Port: This maps to an iSCSI Target Portal Group.

iSCSI supports multiple connections within the same session.

It is also possible to have multiple sessions with different ISIDs to the same Target Portal Group.

Implementations will have the ability to combine connections in a session across multiple Network Portals.

One or more Portal Groups may provide access to an iSCSI Node.

For iSCSI, the SCSI Device is the component within an iSCSI Node that provides the SCSI functionality. As such, there can be at most one SCSI Device within a given iSCSI Node.

The iSCSI Node represents a single iSCSI initiator or iSCSI target. There are one or more iSCSI Nodes within a Network Entity. The iSCSI Node is accessible via one or more Network Portals.

Each Network Portal as utilized by a given iSCSI Node belongs to exactly one portal group within that node.

Between a given SCSI initiator port and a given SCSI target port, only one I_T nexus (session) can exist.

ISCSI: Advanced features

- ErrorRecoveryLevel=2 for active-active fabric level recovery
  - Completely OS independent! (same benefits as multipath, but WITHOUT OS dependent software requirements on initiator side).
  - This makes fabric redundancy easier for admins.
  - ERL=2 and OS dependent multipath work great together!
- Multiple Connections per Session (MC/S)
  - Multiple communication paths between Initiator and Target can be added/removed on the fly across subnets.
  - Faster across multiple 1 Gb/sec ports than ethernet bonding

ISCSI: Advanced features cont.

- ISER (RFC-5045) for scaling to 10 Gb/sec and beyond
  - Direct Data Placement (RFC-504[0,4]) on Internet Protocol
  - Infiniband
- ISNS (RFC-4171) for discovery for large deployments
  - Allows for much cleaner handling of typical fabric and node changes
  - Exentisble for other storage fabrics

Linux/iSCSI Target Projects Status

- STGT
  - New userspace design for Linux v2.6.
- SCST
  - Older design, original focus on fabrics other than iSCSI.
- LIO-SE and LIO-Target
  - Code released in Fall of 2007, in development since Fall of 2001.
  - Runs Linux-iSCSI.org cluster and a bunch of sexy embedded Linux hardware..
- IET
  - Included in some distributions, but little development.
  - A lot of users end up on the IET mailing list asking questions

Linux/iSCSI Project Status: STGT

- Support for traditional iSCSI (from IET).
- Support for hardware accelerated traditional iSCSI (Qlogic)
- Initial support for FcoE from Intel code
- Support for iSCSI Extentions for RDMA using IB hardware with OpenFabrics stack.
- Userspace design
- Merged User <-> Kernel SCSI task submission API
- Small development community
- Included in CentOS 5u1

Linux/iSCSI Project Status: SCST

- Supports traditional iSCSI (from IET) and SRP.
- Only project to support PSCSI, FC and SAS Target mode (with out of tree hardware drivers)
- Initial FcoE code from Intel is based on SCST
- Hardware drivers for target mode operation not supported by vendors (assuming because they are out of tree)
- Extensive emulation of very SCSI specific control CDBs
- Kernelspace design
- Small development community

Linux/iSCSI Project Status: LIO-SE and LIO-Target

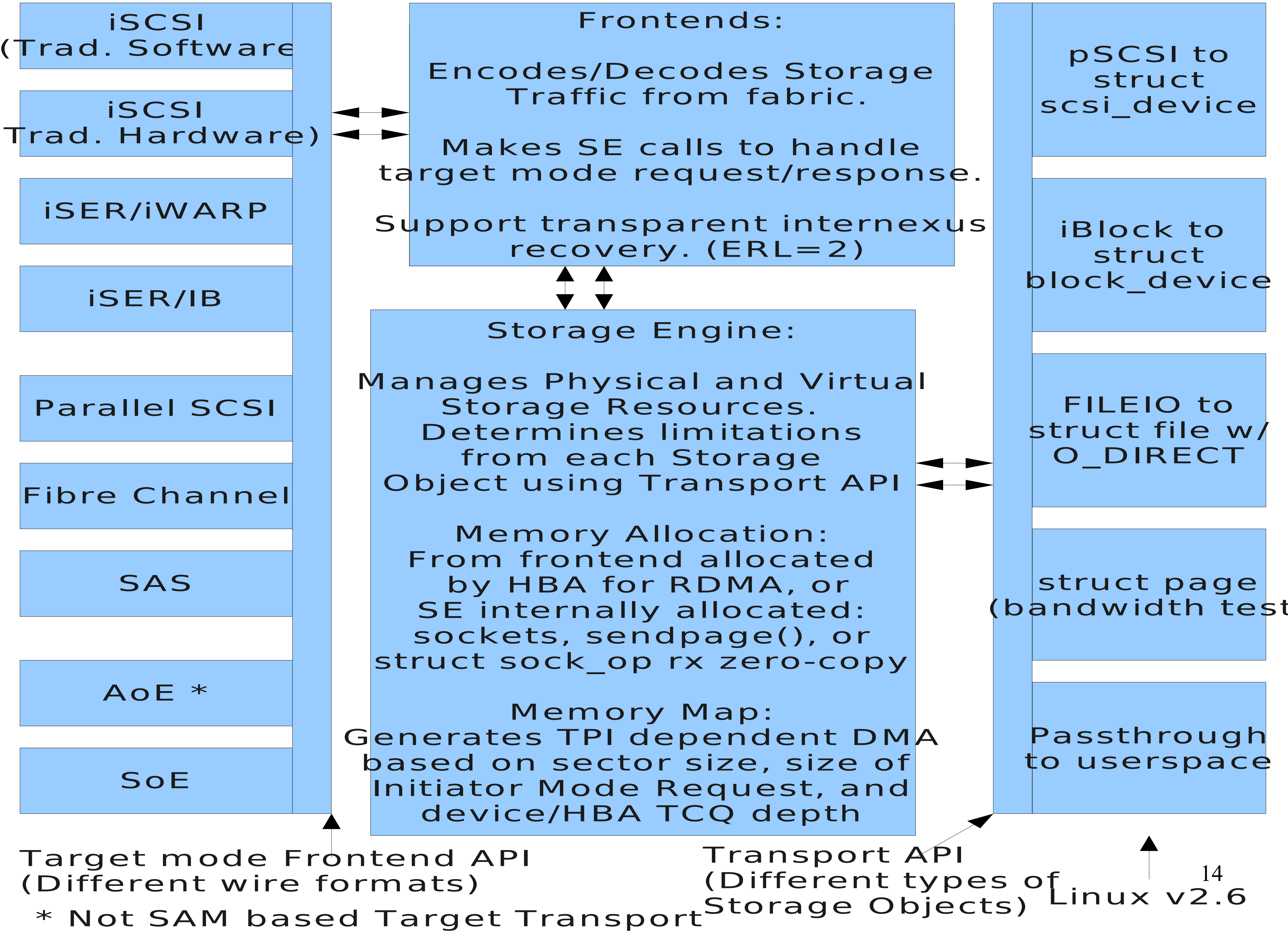
- Most complete support for traditional iSCSI (MC/S and ERL=2)
- Mature API for interaction with Linux storage subsystems (SCSI, BLOCK, FILE)
- Kernel level design with authentication in userspace.
- Very small development community
- Interest of merging LIO-Target and pieces of LIO-SE is starting to increase
- Merge of iSCSI Target code will most likely wait until a proper generic target mode is merged..

LIO-SE and LIO-Target design

- ISCSI Target logic is independent of LIO Storage Engine
- LIO Storage Engine is capable of export of any storage object from any storage subsystem.
- LIO Storage Engine allows for memory allocation from storage transport (RDMA hardware) or internally (traditional iSCSI w/ Linux IP stack)
- Memory is allocated into linked list scatterlists, then mapped to a subsystem dependent method, usually a contigious array of scatterlists for SCSI or BLOCK.
- LIO Storage Engine algorithms handle every possible combination of MaxSector + SectorSize requests.
- Storage Engine controls TCQ depth based on values from hardware and Linux storage subsystems.

The case for a Generic Target Mode SE: Why?

- Provide a single target mode engine design for:
  - All current and future Linux Storage Subsystems
    - drivers/scsi (real SCSI hardware, LibATA, USB, FW)
    - Block via BIO (MD and LVM)
    - FileIO (same as Block, but with buffered IO or O_DIRECT)
    - New request API for >= 2.6.26..?
  - All current and future Storage Fabrics
    - Parallel SCSI, Fibre Channel, SAS, SRP, FCoE
    - Traditional ISCSI (hardware and software)
    - Non SCSI (AoE and NBD)
  - Increase participation amongst the different protects into a single codebase
  - Reduce confusion amongst users and potential developers



Frontends:

Encodes/Decodes Storage Traffic from fabric.

Makes SE calls to handle target mode request/response.

Support transparent internexus recovery. (ERL=2)

Storage Engine:

Manages Physical and Virtual Storage Resources. Determines limitations from each Storage Object using Transport API

Memory Allocation: From frontend allocated by HBA for RDMA, or SE internally allocated: sockets, sendpage(), or struct sock_op rx zero-copy

Memory Map: Generates TPI dependent DMA based on sector size, size of Initiator Mode Request, and device/HBA TCQ depth

The case for a Generic Target Mode SE: How?

- Determine the strengths of each project is the challenge..
- LIO-SE for memory mapping model and subsystem interaction API
- Create common code for SCSI control path emulation
  - Some of this logic will be VERY SCSI-centric
  - Reduce duplicated CDB emulation code when talking to not “genuine” SCSI devices. (BLOCK, FILEIO, LIBATA, USB)
- Use STGT / SCST Transport <-> Storage Engine API as base
- The kernel-level requirement will probably NEVER go away, especially in the vendor community.

Kernelspace vs. Userspace for Generic Target Mode SE:

- Performance Case
  - Still undecided how much additional overhead will be added in various use cases.
  - Keeping kernel component small at risk of sacrificing performance and latency..?
- Complexity Case
  - Difficulty of pushing PSCSI, FC, SAS and RNIC Target mode drivers to userspace
  - Debugging is easier is userspace (with VM and KDB these days, not really true anymore IMHO)
- Historical Cases of Kernel vs. User (from Linus)
  - The only split that has worked pretty well is “connection initiaton/setup in user space, actual data tranfers in kernel space”

Kernelspace vs. Userspace (from Linus) cont.

- Pure user space solutions work, but tend to eventually be turned into kernel space if they are simple enough and really do have throughput and latency considerations (nfsd) and aren't quite complex and crazy enough to have a large impedance-matching problem for basic IO stuff (samba).
- Totally pure kernel solutions work only if there are very stable standards and no major authentication or connection setup issues.
- So just by going what has happened in the past, I'd assume that iSCSI would eventually turn into “connecting/authentication in userspace” with “data transfers in kernel space”. But only if it really does end up mattering enough.

Other Linux-iSCSI.org Projects

- LIO-VM
  - Self configuring iSCSI Target VM that allows export of storage on both Linux and non Linux hosts!
  - Virtual Machines available for Vmware, KVM and Qemu
- iSCSI/HD
  - Allows export of commercial HD media to any environment capable of running an iSCSI Initiator and HD playback.
  - Linux/iSCSI on the Playstation 3.
- ISCSI on Handheld and Mobile Devices
  - Releases for Nokia and OpenMoko Devices
- ISCSI on NeurosOSD
  - Use of iSCSI with an OSS Personal Video Recorder!

Discussion and Thanks

- Linux-iSCSI.org
  - SBEi and Onestop Systems
  - Mike Mazarick, Bryan Black
- Linux/iSCSI Development Community
  - Fujita Tomonori, Mike Christie, Ming Zhang
- Kernel Development Community
  - H. Peter Anvin, Christoph Hellwig, James Bottomley, and many, many more
- Vendor Community
  - Neterion and Leonid Grossman