A Portable iSCSI Initiator

Alistair Crooks
The NetBSD Foundation
agc@NetBSD.org

Abstract

iSCSI is a method of accessing block storage across a network; it enables the SCSI protocol to work remotely, using a TCP/IP transport mechanism. The iSCSI infrastructure has two components - the target, which is the device; and the initiator, which is the end which communicates with the target. The initiator is normally situated in the operating system, sitting underneath the file system code, and making the remote storage appear as a normal, local SCSI device.

The NetBSD iSCSI target is implemented as a userspace iSCSI target. This paper covers the other half of the iSCSI world; the initiator, its interaction with the iSCSI target, and the desirable attributes for an ideal initiator. The portable iSCSI initiator is then described, and in particular describes the unique approach that was taken in the design and implementation of the portable iSCSI initiator. There follows some analysis and discussion of the overall iSCSI performance in real world benchmarks. The initiator's portability between operating systems is also discussed, along with experience gained from its deployment on other BSD systems, and further afield on Linux and Solaris. Some interesting uses of the initiator are described. The paper continues by examining remote storage as a whole, and outlines some places where remote storage could be used to considerable effect. The paper concludes by examining some areas in which iSCSI can bring benefits, and also some other areas in which iSCSI could be extended.

1. Introduction

Until recently, storage access across a network has been accomplished at two levels - at the block level via Storage Area Network (SAN) implementations, or at the file level using file systems such as NFS and Samba, often referred to as Network Attached Storage (NAS). More recently, the emphasis has been placed on block-storage SAN networks. Traditionally these networks have been implemented over custom networks of dedicated fibre, and using Fibre Channel (FC) to communicate. Recently, IP SANs have been deployed more commonly, taking advantage of already existing and cheaper TCP/IP networks, administration and management. Although the price of FC devices and networks have recently dropped, the perception still remains that they are the higher-priced option.

The virtualized world, with mobile and agile virtual machines which can be migrated from physical server to physical server, demands block storage which is geographical location independent. Although the virtual machines currently need to stay on the same VLAN when migrating, the storage may be completely remote. The provision of mobile IPv6 may address this restriction.

The market and opportunities for iSCSI storage exist, and until now, the BSD projects have been able to take advantage of this storage protocol, but only in part, by acting as a server, using the NetBSD iSCSI target. The NetBSD iSCSI target [Crooks2006] was added to the NetBSD src

repository in February 2006, and was subsequently made available for use on other POSIX-based operating systems, by packaging it up, complete with GNU autoconf scripts and templates, and added to pkgsrc. It was quickly added to the FreeBSD ports collection. It is portable to any POSIX operating system, and has even been compiled on Windows XP. Initiators, or client side of the protocol, despite a few examples, have not been forthcoming. There are a number of reasons for this, foremost amongst them being the necessity to embed the initiator within the kernel, which therefore limits their portability. iSCSI initiators thus are usually made available for only one operating system.

However, the iSCSI target is only half of the equation - the initiator is the harder part. It listens for operating system requests from a file system to manipulate storage (read, write, mkdir, mknod etc), formats the request and sends them to the target, and waits for the response, in turn waiting for the response to appear from the target, and sending the response back to the file system which requested the operation. The similarity of this operation to that of a userspace filesystem, as commonly found in the FUSE and ReFUSE examples, is striking.

2. Background

2.1 iSCSI Overview

The IETF proposed RFC 3720 in 2003, specifying the iSCSI protocol. RFC 3720 identifies the iSCSI protocol - in brief, two SCSI devices will now communicate by using a TCP connection as a transport mechanism, rather than a dedicated cable. The geographical independence that this provides gives rise to a number of dramatic benefits, but also raises a number of issues.

An iSCSI target can be thought of as the SCSI server - it sits waiting for requests to arrive from an initiator, performs work according to the nature of the request, and returns results to the initiator.

Immediately, a number of conclusions can be seen:

- geographical independence places a much greater load on TCP/IP communications networks
- geographical independence makes iSCSI a much more attractive solution for Disaster Recovery and Business Continuity
- existing SANs, typically high-end, expensive, Fibre Channel-based have a low-cost, pervasive, easily-managed competitor
- the security aspects are much more prevalent - MiTM attacks, spoofing, data traffic analysis, commercial espionage are now a reality (typically, existing SANs will be built on a custom Fibre network, separate from all other networks)

2.2 iSCSI Targets

iSCSI targets are the block servers for the iSCSI protocol. They are available from many vendors, although, typically, vendors have chosen to embed these in the operating system kernel. This is presumably to take advantage of speed benefits - there is no need for a network packet, destined for the iSCSI target, to move to userspace; instead, the SCSI request can be resolved within the kernel, with existing SCSI driver functionality if necessary, and the results communicated back to the initiator immediately. Whilst this is certainly true, the advent of zero-copy TCP functionality does nullify the performance benefits (to a certain extent). In addition, some things are better performed

in userspace - authorisation, and perhaps link aggregation, for example. Userspace programs are also much easier to develop and maintain, and have less impact in the unlikely situation that bugs exist.

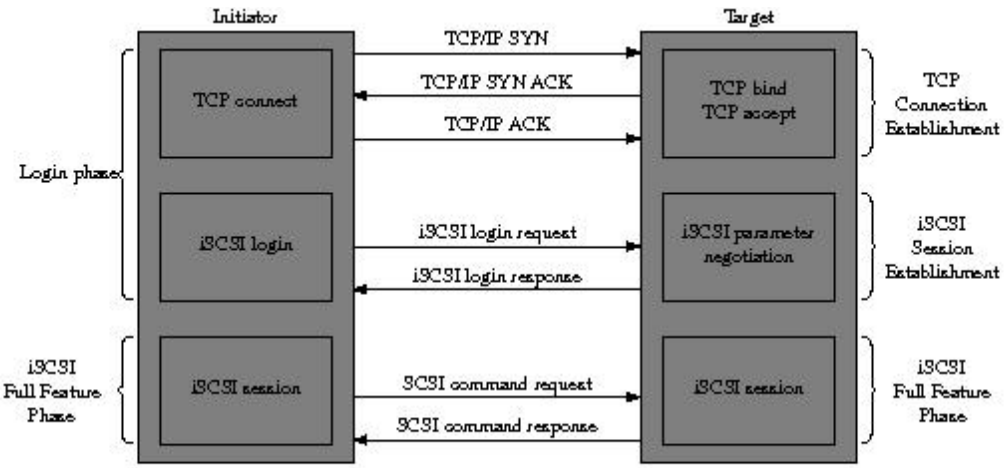
The NetBSD iSCSI target presents LUNs to initiators which connect to it. It can do a primitive form of LUN masking, by specifying a CIDR for requesting initiators, and not responding to requests from initiators which do not conform to this CIDR. The initiator also needs to "log in", although, since CHAP is used to do this process, this primitive form of authentication should not be considered to be any form of security.

2.3 iSCSI Initiators

To communicate with an iSCSI target, an initiator is used. Because of the nature of the initiator (by issuing SCSI requests, and interpreting responses, it mirrors a disk driver) initiators have typically been implemented as part of the operating system . This is analogous to traditional SCSI drivers within the operating system - issuing SCSI requests to block devices, and sending responses back to file system components, also within the operating system.

iSCSI initiators can take various forms: hardware initiators are available in the form of Host Bus Adapters (HBAs), and software initiators come with many operating systems. It is expected that some ethernet NICs will get iSCSI offload capabilities over the next few years. The rest of this paper will deal with software iSCSI initiators.

To illustrate the information flow in an iSCSI session, the establishment and operation of an iSCSI session is shown in the iSCSI Session Establishment Diagram below:



iSCSI Session Establishment Diagram

There are a number of software initiators already available for FreeBSD - the Lucent initiator from 2003, which is for an older version of FreeBSD and does not appear to have been actively maintained, and the kernel-based initiator which again seems to have less maintenance applied than

is desirable. For Linux, the most popular iSCSI solution is the UNH initiator and target, which is actively maintained, and regularly updated. For Mac OS X, Studio Networks provide an initiator, and the Microsoft Initiator is freely available for Windows XP. Solaris 10 has had an iSCSI initiator since Solaris 10 Update 2, and an iSCSI target since Solaris 10 update 5.

Despite the fact that iSCSI has been around as a protocol for over five years now, it has yet to make its mark in the BSD world. The NetBSD iSCSI target has been available for two years, and it has been incoporated into NetBSD's pkgsrc, the FreeBSD ports system, and is part of the FreeNAS distribution of storage-related products. Most of the use of the iSCSI target has been in presenting LUNs to initiators running on different operating systems – the target gets most use, measured through questions received, and “thank you” emails, from Microsoft and Linux initiators. The unenthusiastic take-up of iSCSI in the BSD community must be laid at the door of the initiator – it is difficult to use a product if the means of conversing and interacting with that product is not there.

For the BSD operating systems to take advantage of all the benefits of iSCSI storage, a useful, robust iSCSI initiator was needed. It must be portable across all the variants of the BSD world, though, since they use different SCSI and IDE subsystems, to say nothing of SATA and SAS devices. To achieve these differing goals, some lateral thinking would be necessary.

2.4 iSCSI Test Harness

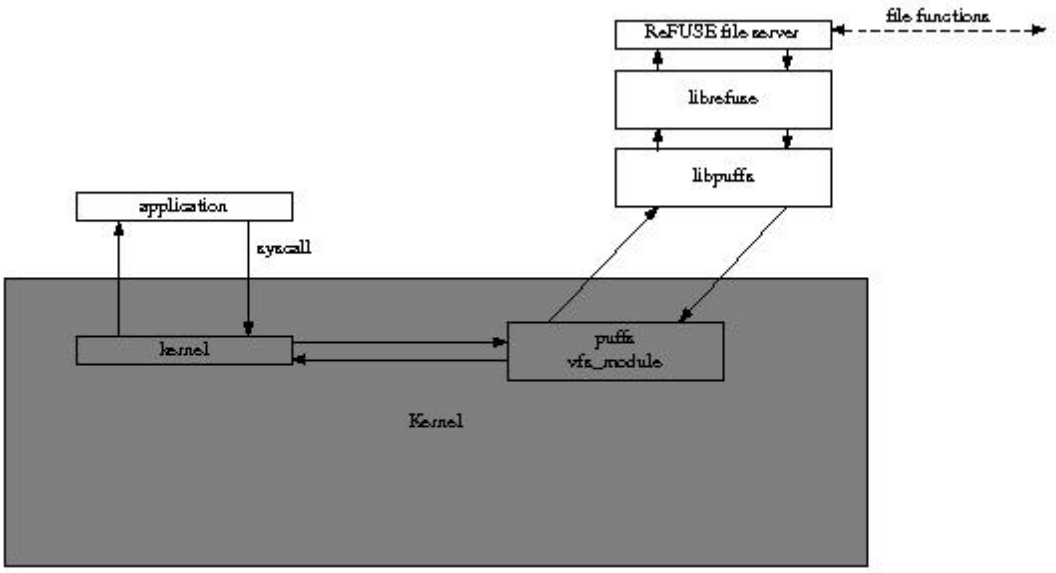
The iSCSI code in the NetBSD repository has initiator functionality, mainly for use in its test harness. All of the protocol encapsulation and decapsulation is available, working and tested. The test harness for the iSCSI target is a userspace program which issues iSCSI requests to the target, and verifies the responses it receives back.

The iSCSI target in the NetBSD src repository is split into a *libiscsi* library component, and a driver program for the target itself. The library will do the encapsulation and decapsulation of the iSCSI protocol itself, along with dealing with setting up the iSCSI connection, and tearing it down, and dealing with all the parameter negotiation and authentication that is necessary for a full-phase iSCSI login to take place.

2.5 puffs and ReFUSE

Recent work by Antti Kantee [Kantee2007] to provide "pass to userspace file system" framework, and by the author to re-implement the FUSE interface on top of puffs [KanteeCrooks2007], has resulted in a framework, within NetBSD, whereby user-implemented file systems can be written to take file system input requests from the kernel, perform work, and to return the reply to the kernel, thereby emulating a kernel-based file system.

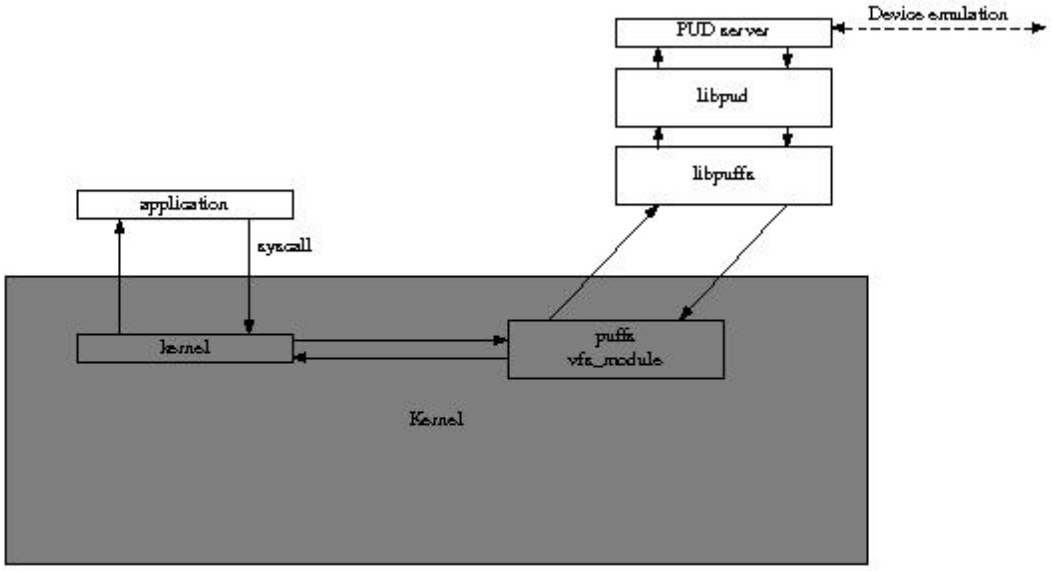
The FUSE interface is used within NetBSD to provide a standardised means of writing userspace-based file systems, the few puffs example file systems notwithstanding. This allows the puffs interface itself to mutate and change. Since FreeBSD presents a FUSE interface as its userspace-based file system, any userspace-based file system written to run via FUSE will run on NetBSD, FreeBSD, Mac OS X, Linux and Solaris.



ReFUSE infrastructure

2.6 Pass-to-Userspace Device Functionality

The same functionality, when used to satisfy block-device requests, would be tremendously beneficial to implement a userspace version of the iSCSI initiator.



Proposed PUD Infrastructure

A system call would be made from the application to the kernel, from where it would be routed through the kernel to the PUD server in userspace, which would issue the requests, gather the responses, and return that response through the kernel to the calling application.

However, puffs is perfectly capable of routing a request to a virtual device through its own protocol to a userspace file system, which can then route this request further. This means that there is no

reason to create new "pass to userspace device" functionality - the existing puffs userspace library and kernel functionality can be reused to satisfy device requests.

3. The Portable iSCSI initiator

3.1 Initiator Infrastructure

At a conceptual level, the iSCSI initiator itself takes requests for blocks (read, write, get size) etc from a higher level, encapsulates these requests in the iSCSI porotocol, sends them to the iSCSI target, and receives the response, which is decapsulated, and sent back to the caller.

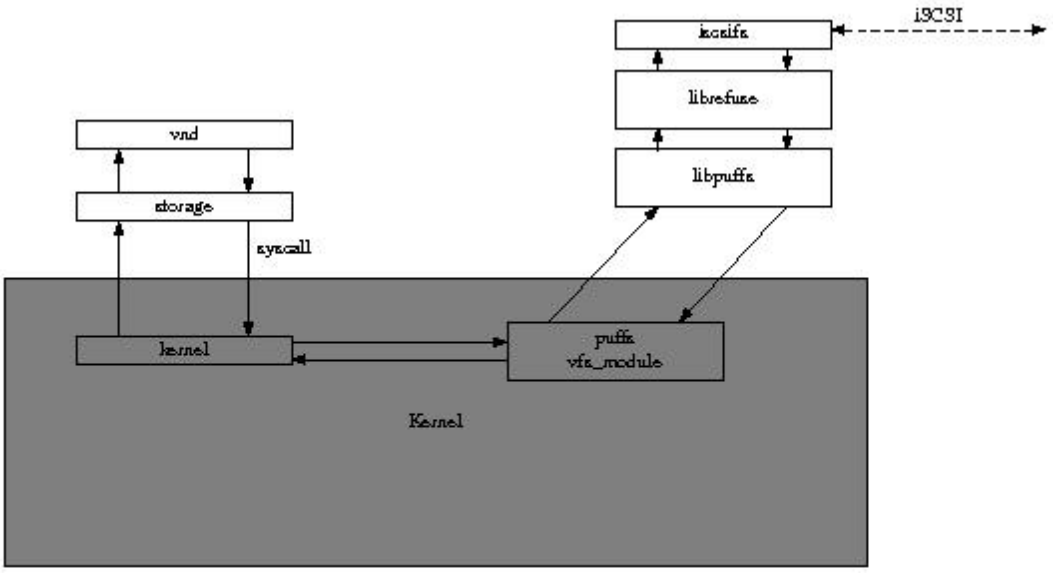
The initial development plan for the portable initiator - given the presence and availability of a number of building blocks such as the iSCSI test harness, and the NetBSD iSCSI target as a userland program, with a POSIX interface, and quite portable to other POSIX systems - was to build a device analogue of FUSE/ReFUSE/puffs functionality; where FUSE works on file system requests, this layer would work on device driver requests. The BSD-licensed FUSD [Nelson2003], a Linux Framework for User-Space Devices, was investigated - it is a kernel facility for Linux which provides user-space device support. Our new device functionality would interpret read/write/ioctl requests in the same manner, sending them to the userland for the component to re-route the response back through the kernel to the caller. This involved a significant amount of code duplication between the device level and puffs/ReFUSE.

During the implementation of the PUD functionality, a number of features were observed:

- substantial re-implementation of puffs functionality was being attempted
- the puffs transport was a multiplexing transport layer, and to re-implement that would not be a good use of resources
- a huge amount of code re-use should therefore be possible betwen puffs and pud
- the ability to use the userspace iSCSI test harness as a pud server became very attractive

After much discussion, during which one observation was made - that devices could work perfectly well on top of puffs, since a /dev instance on top of puffs as part of Antti Kantee's initial puffs testing, the author arrived at the current design - to use the current ReFUSE functionality to receive the requests via the standard FUSE or puffs/ReFUSE capabilities, rather than to implement pud functionality.

The portable iSCSI initiator was then built on top of the userspace puffs and ReFUSE; it uses initiator functionality available in the existing libiscsi to encapsulate requests and decapsulate responses from the target. An immediate advantage of the user-space implementation was the speed of development.



Portable iSCSI Initiator

In brief, the following example shows the portable iSCSI initiator in operation.

```
rxvt
[11:11:12] agc@inspiron1300 ~ 12 > priv iscsifs -u agc -h inspiron1300-wired /mnt &
[1] 17928
[11:11:19] agc@inspiron1300 ~ 13 > ls -al /mnt/inspiron1300-wired/target0/
total 2304
drwxr-xr-x  2 agc  agc       6656 Feb 10 10:55 .
drwxr-xr-x  2 agc  agc       6656 Feb 10 10:55 ..
lrw-r--r--  1 agc  agc         18 Feb 10 10:55 hostname -> inspiron1300-wired
lrw-r--r--  1 agc  agc          9 Feb 10 10:55 ip -> 10.4.0.42
lrw-r--r--  1 agc  agc         16 Feb 10 10:55 product -> NetBSD iSCSI
-rw-r--r--  1 agc  agc  104857600 Feb 10 10:55 storage
lrw-r--r--  1 agc  agc         43 Feb 10 10:55 targetname -> iqn.1994-04.org.netbsd.iscsi-target:target0
lrw-r--r--  1 agc  agc          8 Feb 10 10:55 vendor -> NetBSD
lrw-r--r--  1 agc  agc          4 Feb 10 10:55 version -> 0
[11:11:26] agc@inspiron1300 ~ 14 >
```

3.2 Benefits

A number of benefits are gained through the use of userspace components

- The first, and main, benefit of using a userspace initiator, is ease of development. Using the existing puffs functionality allows us to route file system requests through the initiator functionality in libiscsi with no new kernel components.
- The size of the iSCSI initiator, in terms of source code, is 17 kilobytes of commented C. Although this does not include the virtual directory routines which are used to keep track of directory entries in virtual file systems, these are common to all of the NetBSD-initiated

ReFUSE-based file systems in the *src/share/examples/refuse* directory.

- Because it is implemented in userspace, the iSCSI initiator is better suited to handling authentication, and to using existing userspace tools to take advantage of VPN, IPsec and other means of securing the communications mechanism between initiator and target.
- Portability is obviously a keen benefit of a userspace implementation, since the same initiator can be used, without modification, on a variety of different operating systems. Primarily, any POSIX operating system which supports the FUSE interface, is the only pre-requisite of the iSCSI initiator. Since not even the different BSD variants use the same kernel code for disk abstractions and drivers, this is a substantial win. This also increases hugely the number of testers, and the amount of testing, which the initiator receives. This is not constrained to BSD operating systems - Linux can use the portable initiator as well.
- Simplicity is an underrated benefit. If there is a lot to go wrong, Murphys law posits that it will go wrong. Conversely, if there is little which is capable of malfunction, then little or nothing will malfunction.
- Security is greatly increased with the above benefit, that of simplicity. Especially for such a piece of code, which is used to manipulate basic data, any improvement in the security is to be welcomed.
- A userspace implementation, especially on NetBSD, means that it is easily possible to stack other devices on top of the iSCSI initiator, such as encrypting block devices, or to place the initiator in a RAID implementation, to mirror simultaneously updates to the local file system on a remote site. This will be expanded in more detail later in this paper.

3.3 Drawbacks

There is a feeling in some BSD circles that using a userspace component for iSCSI initiation is somehow "cheating". Whilst it is certainly much easier to implement an iSCSI initiator in user-space, the ease of development, debugging and improvement in development time should not be underestimated.

Concerns have also been expressed about the performance of an iSCSI initiator with a sizeable userspace component. This will be addressed in Section 3.5 of this paper, Benchmarks and Real-World Experience; altogether, it is not considered an issue for the portable initiator.

By presenting the remote storage as a regular file in the local file system, there is a need for a mapping block device (vnd) to be used to give the regular file block-device characteristics, so that device management and manipulation will function properly. This device may not be available to all users of the computer. To avoid use of the vnd device, a mknod could be attempted by the initiator, or the equivalent of the umass USB attachment for mass storage devices could be made. This is an area of future work, although the use of devfs instances does mean that we lose some of the inherent portability by presenting the storage via a block device directly.

Security could be another issue related to performing iSCSI initiation in userspace. Since all data is travelling over communications lines, and block level devices have none of the file and directory permissions that pertain to networked file systems, this is always going to be a problem, and is not limited to iSCSI alone. There are a number of areas which have to be protected from a number of attacks, including replay, MITM, snooping and spoofing. Once again, this is the case for all SAN and IP SAN instances.

3.4 Availability

The portable iSCSI initiator is available under a 3-clause BSD licence, and should be portable to any system with a FUSE implementation, including FreeBSD, Mac OS X, Solaris and Linux. The NetBSD iSCSI target is available under a 3-clause BSD licence, and should be portable to any POSIX-like system.

3.5 Benchmarks and Real-world Experience

The first test that was run on the portable initiator (communicating with the iSCSI target on the same machine) was to do a build.sh run, which builds the complete NetBSD userspace and kernels. The run completed within 0.5% of the time taken to complete the build.sh run using standard disk-based storage. The machine used was an amd64 running NetBSD 4.99.34, with 8 GiB ram. Because of the memory size on that machine, memory caching will be a factor in the figures.

It is interesting to note that this was the first test carried out on the initiator. Usually, with a kernel component, there would be a small “Hello World” type of testing carried out, and gradually ramp up the resources to provide more and more functionality. In this case, since the *libiscsi* library had already been used and found to work reliably during the development of the iSCSI target with the iSCSI test harness, the code was known to work well.

Subsequent runs have shown that it is the usage pattern of data on the iSCSI volume which is the main factor in determining the speed of operations.

3.6 Portability

The portable iSCSI initiator, due to its implementation on top of ReFUSE, should work on any implementation of FUSE. This includes FreeBSD, Mac OS X, and extends to Linux and Solaris. If a traditional Berkeley Fast File System is used, there is a requirement for a special device to be able to be crafted on top of the storage virtual file described earlier, since FFS uses the vnode internally to detect repeated attempts to mount a file system on the same underlying storage. FreeBSD and Mac OS X have the *vnd* device, which can accomplish this, Linux has the *loopback* device, and Solaris has *lofi*, the loopback file device.

3.7 Features

Some of the features of the Portable iSCSI initiator are discussed below.

3.7.1 LUN Masking

In the same way that zones need to be able to specified in a better manner, we also need to be able to mask individual LUNs from discovery by certain hosts. To a certain extent, this is mandated by the desire of one commercial operating system to acquire whichever LUNs that it discovers. Accordingly, the LUNs presented by the target can be dependent upon a CIDR which is given in the /etc/iscsi/targets file. Although LUN masking can be perfomed at the present time, its ability is limited, and this part of the iSCSI solution will be enhanced in future.

3.7.2 LUN RAIDing

To a certain extent, the NetBSD iSCSI target allows for LUN RAID already - the */etc/iscsi/targets* configuration file is made up of two types of definition - the basic unit of storage is the extent which specifies the file or device, the starting address and its length. LUNs sit on top of extents or other LUNs, and can be arranged in a concatenated manner (raid0), or a mirrored manner (raid1). The areas for future enhancement here are

- the provision of dynamic configuration of additional RAID1 LUNs, to allow online backup to be made seamlessly
- the dynamic growing and shrinking of LUNs
- the possible addition of RAID5 (which may add too much complexity for too little gain in today's era of disk sizes of 1 Terabyte and more). File systems which are able to take advantage of these enhancements are necessary, too. NetBSD has long used RAIDframe as its component for providing resilience within the disk storage subsystem. RAIDframe can use an iSCSI "disk" as one of its components.

3.7.3 Block Device Encryption and Security

The iSCSI protocol does not mean to define or include any means of security; rather, it allows other forms of security to be used to authenticate and control access to the storage. It allows CHAP, SRP, Kerberos (not GSSAPI) or none authentication mechanisms to be used. In addition, to maintain data integrity, different digest algorithms can be used to verify that the storage has reached its destination without being corrupted (undetected, despite TCP's checksumming ability, which is considered weak by some).

Security of communication should be done by using some form of VPN to prevent the communications being understood, even if they are captured.

Security of the data at rest can be assured by using the appropriate block-level encrypted device, such as NetBSD's *cgd*, FreeBSD's *GBDE*, or OpenBSD's *svnd*. Once remote storage becomes a possibility, security aspects start to take on a larger role. Encrypted block devices have already been mentioned, but encrypting the storage is no good if all communications to and from the storage take place in the clear. For that reason, communication of iSCSI traffic should take place over IPsec wherever possible, or some other form of VPN. However, for domains which exist in a co-located facility, or which are not under one's complete physical control at all times (such as on a laptop or ultra-portable machine), some form of encrypted block device can be used, stacked on top of the iSCSI volume, to provide protection from people who can gain access to the physical disk storage backing the iSCSI volume. Instructions for using an encrypted block device in conjunction with the portable iSCSI initiator are provided elsewhere [Crooks2008] .

4. Further Work

This section discusses some possible enhancements that could be made to the NetBSD iSCSI system, and any further work that is envisaged.

4.1 Kernel-based initiators

It is quite possible to "drop" the small bits of functionality implemented for the iSCSI initiator into the kernel. In doing that, we would lose the extreme portability that we have now, but performance and efficiency may be gained.

4.2 iSCSI Booting

To be able to boot from an iSCSI device, network drivers would have to be modified. This is similar to modifying the libsa drivers to be able to etherboot. Ideally, to enable operating systems to boot from an iSCSI volume, iSCSI boot functionality needs to be available. This means that either the operating system, or some specialised hardware like an offload engine, or iSCSI Host Bus Adaptor, will issue the request to the target to read the bootblock from the iSCSI volume. This, however, is not a necessity - typically even the smallest device today has its own memory in the form of FLASH or ROM or memory card. This means that it can be used when a full network interface is unavailable. There is another way that iSCSI booting could be achieved without using custom hardware. A NetBSD kernel, with a ramdisk attached, could be downloaded from a tftp server, or other network-boot enabled appliance. As part of the boot sequence, the ramdisk will be set up, including the puffs device and refuse library, and then a userlevel initiator can be used to connect to a given iSCSI target, and to mount a file system on top of the iSCSI volumes presented by the target.

4.3 Multipathing

Being able to use resiliency features such as multipath is extremely attractive, especially in the use of iSCSI for Disaster Recovery and Business Continuity Planning.

4.4 Zoning

SAN zoning can be implemented at two levels: in hardware, and in software. Because disks are no longer attached to the buses on the computers, there needs to be a way to discover, from the initiator, which targets are available. Traditionally, FC SANs have used a Simple Name Server, or SNS, to provide this facility. iSCSI has a similar facility, called iSNS. iSNS can be thought of as similar to DNS - iSNS will return the TargetName of the target in response to a query from the initiator. As part of this facility, certain targets can be configured to be available to certain hosts. This creates zones of storage which are available to different hosts. For example, production zone storage could be presented to production zone servers. At the present time, no iSNS has been provided, although this may change in the future.

4.5 VM integration and Application Migration

iSCSI boot is essential for Virtual Machine mobility and migration. Only when the storage is accessible on the complete network can a virtual machine be moved between physical hosts. It is envisaged that Virtual Machine Service Providers could use this functionality to manage their client virtual machines much more effectively.

4.6 Clustered File systems

Remote storage adds some complexity to file systems, since it is now possible to have mltiple file systems attempting to use the same storage at the same time. Clustered file systems are used for this purpose, allowing multiple access simultaneously to different hosts, and propagating changes to other users of the same storage.

Linux has long been using the GFS file system, from Red Hat; a BSD port of this will be forthcoming. This is a standard clustered file system.

High availability and resilience is another area which can be used to go efct with remote block-level storage. Chironfs is a high availability file system, meant to provide resilience in the case where one piece of storage is unavailable. RAIF is another file system which is used for high availability.

4.7 Dynamic management of iSCSI Target LUNs

At the present time, the NetBSD iSCSI target will read a configuration file, and present LUNs depending on the contents of the configuration file. There is no means of modifying the configuration without stopping the iSCSI target, and re-starting it, this time using the modified configuration from the configuration file.

4.8 iSCSI Bridging

There is the possibility to bridge the iSCSI protocol - to have an appliance which takes the iSCSI protocol, and decapsulates requests, and changes them into requests for a different protocol. This could be anything within reason.

4.9 iFCP and FCIP

In the Fibre Channel SAN world, two RFCs have been provided by the IETF, which provide remote block Fibre Channel storage via ethernet. These are RFC 4172, which defines "A Protocol for Internet Fibre Channel Storage Networking; iFCP", and RFC 3821 on "Fibre Channel Over TCP/IP (FCIP)". These two RFCs provide a means of using Fibre Channel protocols by communicating over TCP/IP. At the present time, there is some interest in these two RFCs, although it is uncertain as yet whether either is useful enough to overcome the huge long-term existing investments in Metropolitan-Area dark fibre, system administration, and hardware for existing Fibre-Channel implementations. There could be a market for extending the range of existing SANs by using some form of iSCSI or FCIP.

5 Benefits and Opportunities

There are a number of areas where the BSD operating systems can take advantage of a full iSCSI solution – both initiator and target:

- virtualization support - with virtualization becoming such a big topic in computing these days, remote block-level storage is an increasingly important part of the virtualization roadmap. In order to migrate guest or domU domains between physical hosts (in order to maintain service levels, and accomplish maintenance with high availability uptimes), the guest domains can be migrated to different physical hosts. This can only accomplished when storage is held remotely from the host or dom0 domain. iSCSI is a natural fit for this.
- General Remote Block-level Storage. Remote storage provision may be useful for computing equipment which is in places which are difficult to access - for example, in hazardous places, or for appliances like IP security cameras, whereby there is enough power to grab images, but not enough to power storage. NetBSD has already been deployed in the International Space Station - iSCSI can also be used to great effect in space, by using storage over communications lines to upgrade and downgrade software easily. Bearing in mind the increased capacities of portable storage (the standard storage on a USB key is currently 8 or 16 GiB), and with the increased bandwidth available from telcos, and despite the labelling of any transfer speed greater than 64 Kb/s as "broadband", remote storage is becoming a much more interesting area, especially when the storage appears to be directly attached:

    - PDAs - the ability to have a central block of storage which can be accessed for all things, such as photographs, calendars, contact books and general storage is not to be underestimated. Using iSCSI for this storage is the obvious choice, since the provision of TCP has already been done. The benefits also include never having to "sync" a portable device ever again.
    - Mobile phones - today, the lines between the typical PDA and the mobile phone are getting blurred. The standard phone nowadays will include a 4+ Megapixel camera, video cameras, and the ability to listen to MP3 files. Most people are standardizing on one device and carrying it - the advantage of this to them is the ease of use. iSCSI can benefit this hugely, since MP3 files can take up a large amount of space.
    - PVR (Personal Video Recorders) - this set of appliances need even more space than the previous set, and this is certainly an area where iSCSI can be of benefit. The relative lack of screen size on these devices means that viewing on a conventional-sized television of the same video streams, using the same source, would be a benefit. That can be achieved by a single central store, accessed by both home network and PVR.
    - Security cameras are often placed in areas which are difficult, or hazardous, to maintain. Having to access the camera to change a tape has not been done for a number of years. However, CCTV can produce a large amount of video imagery, which needs to be stored somewhere, and using iSCSI storage for this would be a plus point
    - Media distribution - for some new films and movies, the standard way of making the movie available is to copy it, and to transport these copies to the individual cinemas, where they are displayed. Rather than using this intermediate copy, providing an encrypted iSCSI device, and letting initiators attach to it, with zoning and LUN masking in place, together with strict IP filtering, would appear to be a safer and less wasteful method
    - Software distribution - instead of providing ISOs which are downloaded, software

distributors can present the image to the net as an iSCSI volume. The initiator can then mount the image as a cd9660, in read-only mode, and the image will appear as if it is a local CD or DVD. No media needs to be used in this process, it is a simple network transfer. It is hoped that NetBSD 5.0 will be made available in this manner (as well as existing methods such as bittorrent, ISO image and ftp), and it will be interesting to see how much download traffic is used in this way. It is interesting to note that the Microsoft Initiator expects any LUN presented to it to be a writable piece of storage, and so it cannot mount ISO images across the internet, for example. Windows XP needs to initialize the LUN presented to it in the Disk Administration menu.

- There has been an upsurge in the number of companies offering remote storage provision. This can be used as off-site backups for Business Continuity Planning and Disaster Recovery purposes. The problem with using these facilities is usually that the interface demands a Windows installation. There are some concerns about these sites, especially in the security arena - who is looking at my backups, and how can I protect them without inconveniencing them myself? - but the Windows interface is the reason that the whole debate is rendered moot for BSD users. If remote storage providers were to provide an iSCSI interface, then the use of encrypted block devices on top of the iSCSI initiator would answer the previous criticism.

- Other opportunities - the iSCSI target and initiator together have a complete SCSI subsystem in software. There are opportunities to make appliances based on this software. Still other opportunities present themselves:

    - an appliance can be made which keeps a record, with timestamps, of all the modifications to a piece of storage, in precise time order. Using this with existing snapshots of devices, a complete copy of changes ensures that the precise state of any storage device can be determined at any given time. This can be useful for statutory or audit requirements.
    - an appliance can be made which will do precise online backups, using a copy-on-write model, with no snapshot mechanism necessary.
    - instead of making snapshots of production systems, online and continuous backups can be done to a secure remote site by the methods outlined previously. In the event of recovery being needed from remote data, read-only copies are available with no re-configuration necessary (this avoids logistical problems of tapes being copied and transported back to the original site)
    - storage service providers could use iSCSI rather than proprietary protocols
    - distribution of software by exporting data from read-only iSCSI targets could become a mechanism of choice. Not only can packet filters be used to govern access to the iSCSI host and port, but encryption can also be used to make sure that, even if unauthorised access is gained to the image being distributed, this access is rendered useless, since the means to decrypt the image is not known.
    - "anonymous" iSCSI could be used to allow read-only access to an image. This would mean, for example, that ISO images could be made available on the Internet, people could mount the images remotely, and upgrades and software installation could be accomplished without having to transfer the whole ISO image (with iSCSI, only the blocks which were required would be read, and would be saved locally in the page cache). This could also remove the intermediate step of writing a CD or DVD containing the ISO image.
    - the provision of remote storage for PDAs and personal electronic equipment has already been mooted. With iSCSI booting, it is possible to enable remote devices attached to the Internet to boot from a remote iSCSI target.

6. Conclusions

This paper has talked about the portable iSCSI target, and the unique design of the portable initiator. Both the target and initiator are designed to be portable, so that the benefits of iSCSI may be obtained on all of the BSD operating systems, and also further afield. The iSCSI protocol was specified a number of years ago, and the initial promise of iSCSI has yet to materialise - in some ways it has been held back in the BSD world due to the lack of a viable, robust iSCSI initiator. This paper has described such a portable, yet performant and robust initiator, and has suggested some opportunities for the deployment of iSCSI, and also some extensions to the iSCSI model, applications and implementation.

References

[Kantee2007] http://2007.asiabsdcon.org/papers/P04-slides.pdf

[KanteeCrooks2007] http://2007.eurobsdcon.org/presentations/Antti_Kantee/refuse.pdf

[Crooks2006] http://www.alistaircrooks.co.uk/agc/papers/2006-11-eurobsdcon/iSCSI_beyond_the_hype.pdf

[Crooks2008] http://mail-index.netbsd.org/netbsd-users/2008/01/05/0001.html

[RFC3720] http://www.ietf.org/rfc3720.txt

[Nelson2003] http://www.circlemud.org/~jnelson/software/fusd/