iSCSI Switch: an Implementation of a Software Switch for the iSCSI Storage Protocol

By

Cosmin Forian Dinu
Master of Science in Internetworking
University of Alberta 2013

PROJECT SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN INTERNETWORKING

In the
MINT Graduate Degree Program At The University of Alberta

University of Alberta
Fall 2013

Abstract

The IP storage technology becomes very popular nowadays, as a consequence of the rapid evolution in the IP networking infrastructure technology in terms of speed and reliability. The 10GBaseT standard 802.3an offers the speed of 10 GBps over copper wire with a Bit Error Rate of 10-12 , at the same level of speed and quality with optical fiber links. This is fact, IP networks becomes very reliable starting with the physical layer and taking into account the reliability mechanisms implemented at the upper layers as for example CRC32 error detection at the data link layer, internet checksum at the transport layer and also the header and data digest protection implemented at the application layer, we can conclude that the data error probability on a modern TCP/IP connection is at least comparable with the one of a physical SCSI bus. In addition, the IP technology has no distance limitations in compare to the physical SCSI bus, therefore the entire Internet can be used as a huge storage network with a minimal cost of an internet connection.
Putting storage traffic (I/O commands and data) over IP will also take advantage of the IP security enhancements: IPsec at the network layer and various layer 4 firewalls makes the storage networking very secure. Adding also the IP advanced load balancing and failover capabilities, we can conclude that iSCSI is a highly flexible, reliable and secure technology.

Despite all of this, the "iSCSI world" has a lack of diversity: beside iSCSI initiators and targets there are very few iSCSI-dedicated entities. In RFC3721 are described the concepts of iSCSI gateway and iSCSI proxy. In this project I will introduce a new iSCSI entity: the iSCSI switch. This new entity is basically an iSCSI proxy (as it is described in RFC3721): more than a socks proxy, it acts as a transparent proxy at the application layer. In addition, it will transparently interconnect initiators and targets based on a "switch table" that is described in the last section. As a protocol aware entity, the iSCSI switch has also the capability of tracing, filtering and logging the iSCSI traffic.

Acknowledgements

I would like to take this opportunity to thank Dr. Mike MacGregor who has offered me the opportunity to work on a capstone project that meets my area of interests: storage area networking and C Linux programming. I will remember the course MINT 704 "The Internet Protocol Suite" for its high level of knowledge, accuracy and information quality. I will also remember 704 course for the difficulty degree of the assignments, especially the last…

Table of Contents

List of figures

Introduction

This capstone project contains two sections.

1. The first sections consists in an overview of the storage technologies and data sharing protocols, presenting them in a structured manner with a focus on iSCSI protocol. I will describe my point of view regarding classification and I propose a layered model for data sharing hybrid protocols.

2. The second section of this project is a "proof of concept". I have implemented a software appliance named " iSCSI Switch" based on a concept described in RFC3721:

RFC 3721 section B.4: iSCSI Proxy

"An iSCSI proxy is a gateway that terminates the iSCSI protocol on both sides, rather than translate between iSCSI and some other transport. The proxy functionality is aware that both sides are iSCSI, and can take advantage of optimizations, such as the preservation of data integrity checks. Since an iSCSI initiator's discovery or configuration of a set of targets makes use of address-independent iSCSI names, iSCSI does not have the same proxy addressing problems as HTTP, which includes address information into its URLs. If a proxy is to provide services to an initiator on behalf of a target, the proxy allows the initiator to discover its address for the target, and the actual target device is discovered only by the proxy. Neither the initiator nor the iSCSI protocol needs to be aware of the existence of the proxy. Note that a SCSI gateway may also provide iSCSI proxy functionality when mapping targets between two iSCSI interfaces."

In the second section I will describe the general architecture, functional flow, purposes and benefits of the iSCSI Switch entity. I will also present the software architecture and development tools used in implementation.

Section 1 Technology background. Data sharing techniques.

Data sharing between computers (servers or end user desktops) is one of the main purposes of a TCP/IP network. There are many ways in such data can be shared between computers in a network but all of them falls in two main categories:

1. File level sharing.
2. Block level sharing.

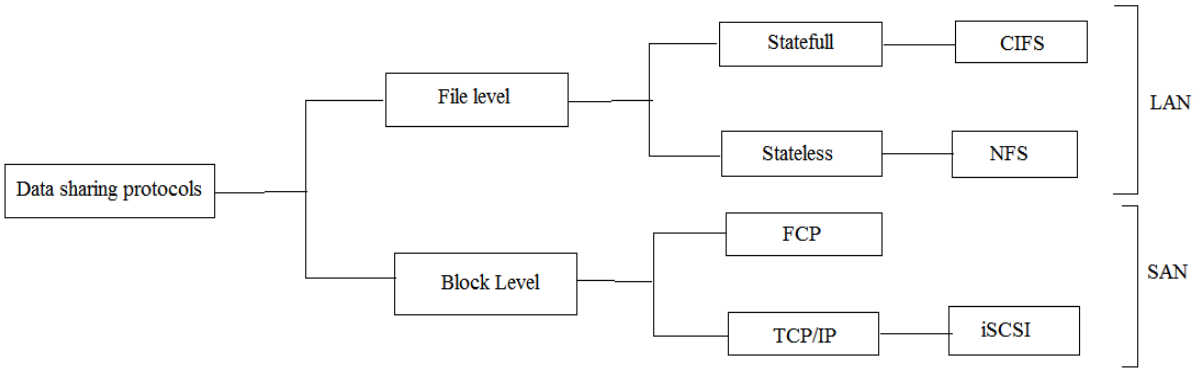

Fig 1 Classification of Data sharing protocols

This two main categories of accessing remote data, file level and block level, implies the use of two completely different abstractions. A file-access sharing protocol has to be aware of the filesystem as it will provide file level granularity in accessing remote data. In contrast, a block-level sharing protocol will not need any understanding of the remote filesystem, the format type of the storage will remain transparent and the protocol will not make any difference between let's say an EXT4 and a NTFS4 format.

Using NFS/CIFS provides a way of using remote storage as a file repository. Using block addressing, the remote storage can be used in the form of disks: shared storage logical unit can be formatted in any mode and can store files, operating systems, everything. But there is a drawback: although the IP infrastructure has the advantage of low costs, the IP protocol was designed for large chaotic networks compared to the SCSI protocol that transports SCSI commands and data over a simple (point-to-point) SCSI bus. Putting these completely different worlds, IP and SCSI, working together in a reliable manner is a challenging task and requires the usage of advanced testing and troubleshooting tools.

There is still a debate regarding which type of granularity (file or block) is the best suitable for an IP network. Benchmarks described in [1] proved that iSCSI (block-access) and NFS (file-access) are comparable for data intensive applications, while in meta-data intensive application iSCSI performs better due to its asynchronous meta-data update mechanism which appears to be faster than NFS synchronous meta-data update. However the CPU load of the NFS server during benchmarks has proved to be significantly higher than the CPU load of the iSCSI server so the benchmarks favors the block access over the file access.

1.1 File level sharing

In this category, we can find network file sharing protocols:

1. SAMBA/CIFS (stateful)
2. NFS (stateless).

In a stateful file sharing protocol (like SAMBA or CIFS), once a client will open a file on the file server, the server will "memorize" that the file was opened by that particular client. It will act by denying other clients to access that file until the client closes it. In fact, server creates a "connection object" that will be alive from the connection phase to the release phase. It will always keep track of the "memorized" current state: server will remember client state from one request to the next and future requests will change the current state accordingly.

In contrast, in a stateless file sharing protocol server will not create any object that will track informations regarding client requests: it can be imagined scenarios when a file on a remote share is opened by more than one client at the same time. That's why in a "stateless" NFS setup is not recommended that two or more clients to mount the same share in the same time.

File sharing protocols are file system aware: they are high level protocols which act as a proxy by redirecting kernel syscalls trough the network. A client request will be something like: "open the file with the path /root/dir/file.txt and return the content to me". This scenario can be very inefficient when clients does not need the entire file, but only part of it, operating on specific fragments at a time.

NFS and SAMBA/CIFS are considered classical data sharing protocols at the file level. Beside these "standard" protocols, there are some others let's say "exotic" protocols that I want to remind:

1. FUSE user space filesystem: FUSE allows to implement a fully functional filesystem in an user space program like ssh or ftp client. FUSE works by presenting to the local operating system a virtual filesystem containing remote directories that can be accessed via ftp or ssh. On top of FUSE resides:

- CurlFtpFS: is a filesystem for accessing FTP hosts. Using this, directories resides on remote FTP servers can be "mounted" on a host in a transparent manner for the operating system. User space applications will be able to access remote FTP files and directories like there are ordinary local files. It uses Curl library for accessing remote FTP accounts.
- SshFS: similar to CurlFtpFS but remote directories are accessed by ssh client.

2. WebDAV filesystem: allows mounting remote WebDAV resources into the local filesystem. WebDAV (Web Distributed Authoring and Versioning) described in RFC 4918 is an extension of HTTP protocol that allows (besides reading) also writing remote files stored on HTTP servers. This way, Web becomes also a writable environment. WebDAV allows maintaining some extra file properties like for example various informations about file author, creation/modification timestamps.

3. “NoFS” category of distributed filesystems: a distributed file system works by using a central master server to manage metadata of distributed files. In contrast, NoFS uses a central master server for managing the file volumes: files and metadata will be managed by the distributed volume servers. In this way, files metadata are distributed across volume servers memory allowing things that are considered impossible like for example file access with O(1) disk read operation. An example of NoFS is Weed-FS, an opensource project hosted by Google.

1.2 Block level sharing

In contrast with file level sharing, .block level sharing doesn’t work by addressing files by their logical name. They address data by blocks. Block level sharing doesn’t have any understanding of file systems. It address only raw data in the form of blocks as required by higher level protocols and, the main advantage for this is speed: it works faster, furthermore it reduces load from both remote physical disk and network.

I propose the following layered model for protocols used in block level sharing:

Layer 1: I/O protocols for accessing block devices:
- ATA
- SCSI

Layer 2: Transport protocols: used to carry I/O protocols from Layer 1
- Ethernet protocol
- TCP/IP
- FCP (fiber channel protocol)

Layer 3: Resulting hybrid protocols
- ATA over Ethernet (AoE)
- FC over Ethernet
- FC over IP (FCIP)
- SCSI over IP (iSCSI)
-

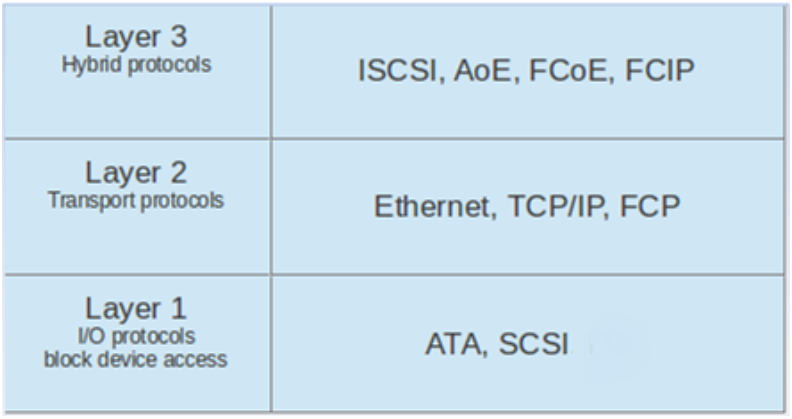

Fig 2. The layered model for block data sharing: I/O protocols from Layer 1 are mapped into one of the transport protocols: the result is a hybrid protocol.

In this layered model, each low level I/O protocol can be mapped into a specific transport protocol. The result is what I mean by "hybrid protocol": a set of specifications that maps an I/O protocol into a transport protocol.

1.2.1 Layer 1 protocols: SCSI and ATA

First of all I want to clarify from the beginning that SCSI and ATA are more than I/O protocols for accessing block devices. Along with upper-level command set and message formats, both defines physical cable and connectors, bus signal characteristics, device addressing and data routing, mapping between physical (hardware) and upper-level (software).
In this paper SCSI and ATA are referred as upper-level (software) I/O protocols only.

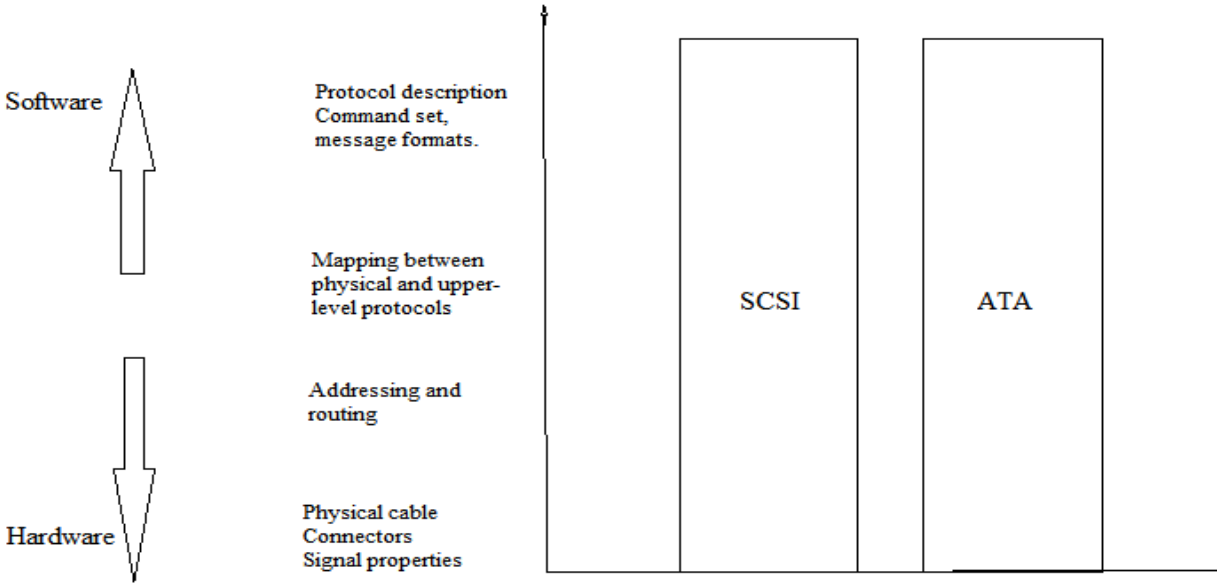

Fig.3 Layers of SCSI and ATA

1. SCSI (Small Computer System interface) is a client-server protocol for transferring block I/O data between block I/O devices. The SCSI Architecture Model [SAM-2] contains architecture description. Clients are called initiators and acts by issuing SCSI commands to the server known as SCSI target. A very common example of a SCSI conversation between initiator and target is when initiator tells target to read or write N number of bytes starting with a particular offset and target replies to initiator by indicating whether data was successfully committed or not to the attached disk. To ensure reliability, SCSI protocol is transaction oriented: if a command in a transaction fails, then the entire transaction is discarded and data is not committed to the disk.

According to [2] there are near 60 different SCSI commands grouped in 4 categories:

1. N (non-data)
2. W (writing data from initiator to target)
3. R (reading data)
4. B (bidirectional).

Along with protocol specifications and command set description, SCSI standard includes addressing definition: how data is addressed through the SCSI bus. A target is defined by a target ID and can contain one or more Logical Units each one identified by its own Logical Unit Number (LUN). Initiator addresses the Target in the form of **Bus:Target ID: LUN**

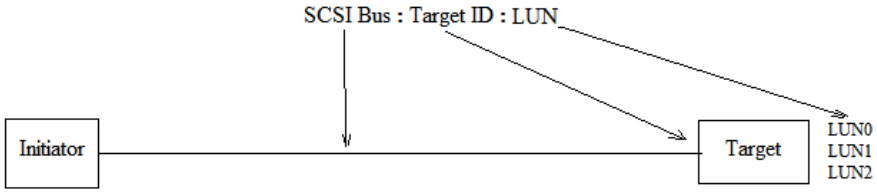

Fig.4: SCSI Address format

2. ATA (AT Attachment) protocol is similar with SCSI. Although ATA command set differs from SCSI command set, there is the same logic: for accessing data on a block device, initiators are sending protocol commands to the target, and target will reply with responses. In addition, the ATA command set includes commands for device management at the hardware level like for example PU (power-up), PUIS (power-up in standby), DR(device reset) etc. which are not related to data access.

1.2.2 Layer 2: transport protocols for storage traffic

1. Ethernet
2. TCP/IP
3. FCP

Description of Ethernet and TCP/IP is beyond the scope of this paper. I will limit to make some observations regarding how these protocols are suitable for transporting storage traffic i.e. I/O commands and data. I will start by observing that Ethernet as protocol is not routable compared to IP. Frames are living in the same Ethernet segment (LAN) thus if the transport protocol for storage traffic is Ethernet, devices sharing data i.e. servers, workstations and storage arrays should reside in the same LAN. Moreover, it is well known that Ethernet addresses (MACs) can be easily spoofed and this should be a major security concern. Choosing IP as transport could also raise issues: IP was designed as a best effort transport protocol trough chaotic networks, IP packets can follow different paths in their travel to destination, and can cross third party routers and networks that are almost impossible to control. Moreover, IP traffic is prone to packet loss due to congestions. Putting storage traffic to be carried by IP raises serious security and reliability concerns. On the other hand, an IP link has no distance limitations compare to a SCSI bus which is limited to 25m (Fast Wide SCSI) so putting SCSI storage traffic to be carried by IP will totally eliminate the bus length limitations. Initiators and targets can reside in any place on the internet. Of course this can raise latency issues but the iSCSI link will be functional.

FCP. Despite of its name, the fiber channel protocol is not bounded to the optical fiber. It can run through any physical medium but optical fiber is preferred because it operates at longer distances. It was designed as a serial data transmission protocol for transporting any type of data between two nodes that can be for example two computers, a server and a disk array or a library tape and a storage. FCP is concerned with data delivery only, it doesn't care about the content that can be storage traffic (I/O commands and data) or IP, ATM and other upper layer protocols. Current implementations operates at very high speed 1, 2, 4, 8 Gbps at long distances: multimode up to 500m, single mode up to 10Km (120Km with extended-distance transceivers). FC frames are very similar to data link frames (layer 2 in the OSI model): a FC frame starts with SOF, followed by a header, payload, CRC and EOF. The difference from a data link frame is that header contains (along with the FC address) the FC port destination and from this perspective it's similar to a TCP segment at layer 4 in the OSI model. FCP has its own layered model which is in many ways similar to the OSI model but I find it more complex:

FCP Layer 0 ("FC-0" or "Physical interface")
FCP Layer 1 ("FC-1" or "Encoding")
FCP Layer 2 ("FC-2" or "Framing and Flow Control")
FCP Layer 3 ("FC-3" or "Common Services")
FCP Layer 4 ("FC-4" or "Upper-Layer Interfaces")

FC-0 is similar to physical layer in the OSI model. It deals with the physics of the signal at different transfer rates. It specify the format (physical signaling protocol) of the signal (electrical or optical) and how this signal will be transmitted/received at the physical layer. Also there is necessary to convert the parallel signal originated from specific initiators to the serial signal used by FCP: this is called SERDES (serializer-deserializer).

FC-1 is responsible with encoding (FCP use 8B/10B encoding scheme scheme), bit error detection and synchronization of data streams.

FC-2 is responsible with framing (construct frames or parse frames from the stream, frame error detection using CRC32), flow control, link initialization and recovery.

FC-3 was designed to support provide various services like: authentication, encryption, compression, link multiplexing and also some services related strictly to storage traffic: disk mirroring, virtualization.

FC-4 is responsible for mapping upper-layer protocols to FCP allowing multiple protocols to be transported over the same interface. This layer provides the mechanism in how the storage traffic is packed and transported with FCP frames.

Regarding the network topology, FCP starts with basic p2p links (full duplex transmitter-receiver pairs) and, using these primitives, a complex topology is developed: switched fabric. Although there are similarities between FC switched fabric and a network switch fabric, the FC fabric has its own architecture for Address Space: instead of MAC addresses it uses worldwide port names (WWPN) on 128 bits, and FCIDs (fiber channel IDs on 24 bits) in the role of IP addresses.

Conclusions: Although FCP can transport any type of upper layer protocols, we can observe that its design reveals a special attention for SCSI traffic. FCP was mainly developed in response to the increasing demand for high speed storage links. The costs for FC fabric switches and optical fiber links is very high, but FCP remains the first option for transporting storage traffic.

1.2.3 Layer 3: hybrid protocols

1. FCoE and FCIP.
Storage traffic can be carried by FC protocol. Putting FC protocol to be carried by IP or Ethernet makes sense for the reasons of costs: there are situations in which a FC link using fiber optic as physical medium could be expensive and instead can be used the existing IP infrastructure.
The FCIP hybrid protocol carries FC frames using TCP/IP as transport protocol. In contrast, FCoE uses Ethernet data link protocol to carry FC frames. The difference is that IP is a routable protocol thus, using FCIP, the FC payload can be routed between different LANs using the existing L3 devices (routers or L3 switches). FCoE also makes use of the existing network infrastructure but it runs at the Ethernet layer therefore is not routable, it keeps data in the same Ethernet segment (LAN). FCoE requires the creation of point to point links in the switches using the MAC addresses of hosts and storages. FCoE has the advantage of speed: using a L2 protocol like Ethernet, FCoE is faster than FCIP.

2. SCSI over Ethernet.

Although there is no any implementation for this, it can be imagined commands and data encapsulated directly in Ethernet 802.11 frames. In this way, a 10Gigabit Ethernet segment will become a SCSI bus. This appears to be faster than iSCSI because Ethernet is situated at a lower layer in the OSI model compared to TCP/IP.
There are some major disadvantages for this architecture. First, Ethernet is not routable thus the initiators and the target needs to be in the same LAN. Second (and the biggest) is security: spoofing Ethernet MAC addresses is very easy. It's impossible to secure SCSI packets flowing through a flat Ethernet segment and this will be equivalent to run a SAN without any zoning and LUN masking on the storages.

3. FCIP and iFCP

There is a particular hybrid protocol in storage networking that doesn't map into the layered model described above. It's an exception. This is the iFCP protocol. iFCP allows hosts and SAN devices (storage arrays, tape libraries) to communicate over a VAN. Fiber channel port addresses (the |WW|PNs) are mapped to IP addresses retained in an iSNS (Internet Storage Name Server) table. In contrast with FCIP, iFCP connections are not permanent and iFCP works in a many to many architecture: any device can connect to any device in the same zone. While FCIP uses IP for transporting FC frames trough permanent fixed point to point links (similar to permanent virtual circuits in packet switched networks), iFCP uses TCP for flow control and segment reordering in "many to many" adhoc links. It becomes clear that FCIP is similar to a gateway-to-gateway protocol while iFCP is similar to device-to-device NAT protocol. While FCIP gateways will not touch the FC_IDs (the equivalent of IP in SAN) in the FC frame headers. iFCP switches

have to modify each FC frame header in order to translate the FC_IDs. This is the main reason for the considerable higher latency of iFCP compared to FCIP.

1.2.4 iSCSI

The iSCSI protocol

One of the widely used technique of sharing remote disks trough a TCP/IP network is iSCSI. iSCSI is not a "standalone" protocol but a set of specifications for transporting SCSI commands over TCP/IP, taking advantage of the existing Internet infrastructure. In fact, an IP link can be thought as the "SCSI bus" for iSCSI: SCSI commands are encapsulated in TCP segments carried by IP packets.

In this section I will describe how SCSI protocol I/O commands and data are packed into protocol data units (iSCSI PDUs). Next, the PDUs will be exchanged trough a TCP socket between iSCSI server (called iSCSI Target) and iSCSI client (called iSCSI initiator).
The iSCSI PDU should contain, along with data (payload) the following informations:

1. The code of the specific iSCSI operation (opcode).
2. A flag that indicates immediate delivery (Imm flag).
3. A flag that indicates if the PDU is the last in sequence.
4. The LUN on the target which is addressed by the PDU.
5. The task (transaction) ID for which PDU belongs.

Beside these, the PDU should contain information regarding the length of the payload and also the length of additional headers (optional header and data digest). All these informations are embedded in a 48 bytes long header structure named Basic Header Segment (the BHS).

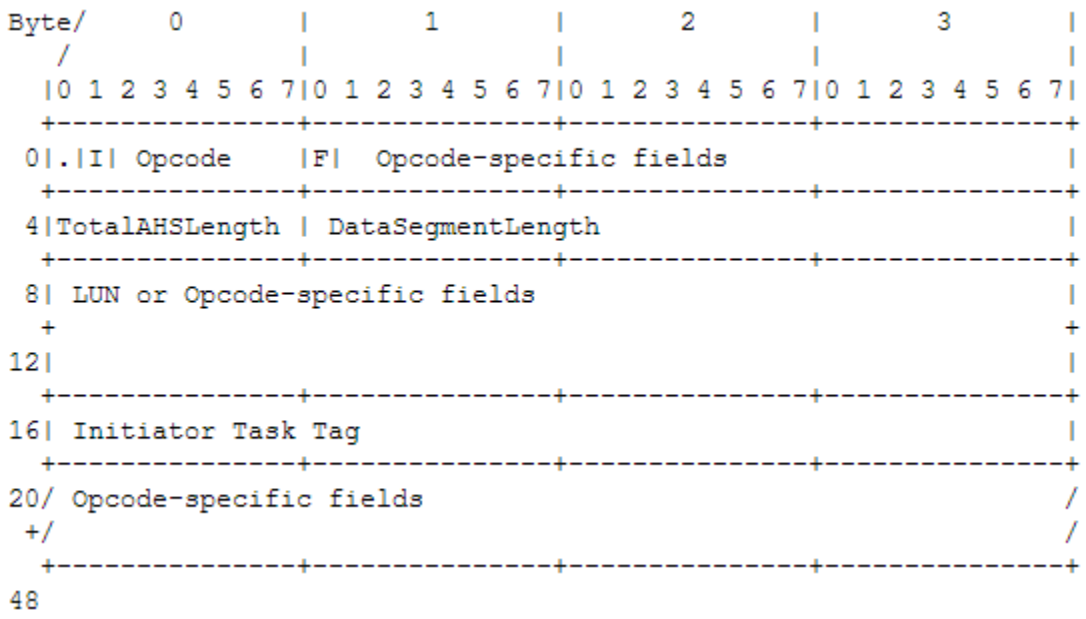

Fig 5. The format of a BHS

Every PDU will start with this 48 bytes long, followed by additional headers (that are optional) and payload. Using this format, the PDUs are exchanged between iSCSI initiators and targets trough a TCP/IP link in a client server architecture.

iSCSI transactions: Every SCSI command (for example read or write request) will be sent to the target in a structured form named the Command Descriptor Block (CDB). After the Target will finish processing the CDB, it will send a PDU to Initiator to indicate the completion of the request. iSCSI protocol is "transaction oriented" meaning that more than one CDB can be grouped in a transaction and in this case Initiator will indicate the end of the transaction by sending a PDU with the final flag set. Target will send one final PDU indicating the completion of the transaction. If transaction fails, data will not be committed to the LUN.

iSCSI layered model

Next is presented the big picture of the iSCSI layered model, similar with OSI layered model, each layer providing services (acting as a provider) to the upper layer (which acts as a client). SCSI I/O requests issued by an application to the remote LUN will be packed in TCP segments, IP packets and finally in OSI layer 2 frames that can be Ethernet, PPP, or any type of data link frames.

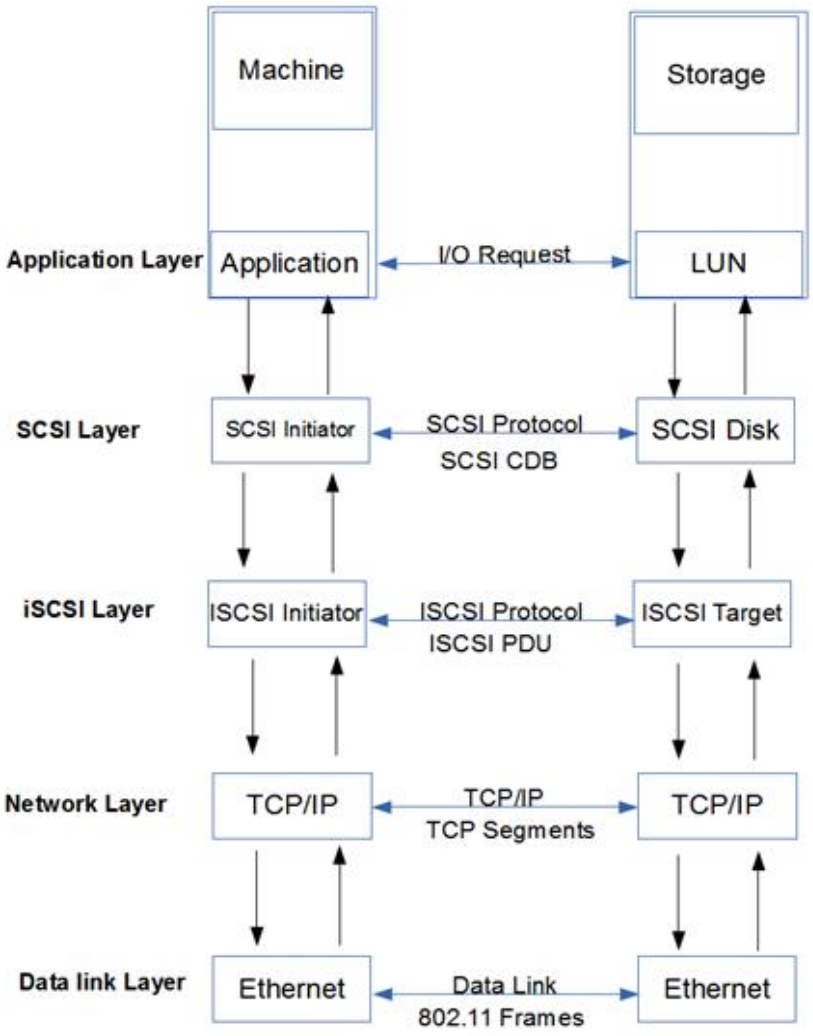

Fig 6. The iSCSI layered model

iSCSI error handling and recovery

One of the main iSCSI goals is to perform over unreliable, non-predictable, chaotic networks like the Internet. That's why iSCSI protocol was designed with a special attention to error handling and recovery. For this to work, initiators and targets were designed to buffer commands/responses until acknowledge is received. As an example to illustrate how buffering mechanism works, let's consider a SCSI write transaction: initiator will keep data in its buffer after transmission until target will send a R2T (ready to transfer command) indicating that the data was successfully received and target is ready to receive next amount. This can also be considered as a flow control mechanism at the application layer.

Error handling and recovery starts at the very low level of an individual PDU: each PDU format is checked for errors (missing or corrupted fields revealed by digest mismatch) and if it will not pass the format check, the other endpoint will be asked for retransmission. The retransmission mechanism can be implicit (using timeouts, known as "silence of the receiver" technique) or explicit by sending an iSCSI Reject PDU as response. Usually the PDUs of type Reject is sent when a data digest error or header digest error is detected and will trigger the retransmission of that PDU. In the case of a header corruption (header digest mismatch), the Reject type PDU will indicate the offset of the first erroneous byte in the corrupted header. In the case of a corrupted payload (data digest mismatch) the request of retransmission is accomplished by modifying the offset field in the R2T. Beside of corrupted PDUs, retransmission can also occur in the case of a missing PDU. In this case, iSCSi protocol uses a special type of PDU named SNACK PDU. The SNACK will contain the sequence number of the missing PDU and will ask for retransmission of that PDU.

Beside the error recovery at the PDU level, iSCSI protocol also contains recovery mechanisms at the connection and session level, which are beyond the scope of this paper. In addition of the error handling and recovery techniques implemented at the upper-level protocol, an iSCSI link relies also on the CRC based error detection mechanisms at the OSI datalink layer and also on TCP internet checksum error detection and TCP retransmission mechanisms contained in the sliding window algorithm at OSI layer 4. Based on this, I conclude that iSCSI is a high reliable storage protocol.

iSCSI security

The iSCSI permits various security mechanisms to be negotiated during the login phase. PDUs exchanged between initiator and target during the login phase are of type text: payload of text PDUs provides a way to negotiate link parameters that are packed in the form of "key=value". The protocol login phase specifies a category of text fields used in negotiating the type of security supported by both initiator and target. Type of security can be based on CHAP (for secure exchange of login credentials), Public/Private keys, Kerberos. In addition, an iSCSI link can benefit from various security mechanisms offered by the layers below, like for example IPsec. Also various IP based firewall rules can be added on target side to restrict initiators access. Based on all this, I conclude that iSCSI can be configured to meet the most demanding security standards and can run safety on insecure, non-reliable internet connections.

Section 2: iSCSI Switch

2.1 Architecture

An iSCSI switch is a software appliance that connects initiators and targets from different networks and handles the iSCSI traffic at the protocol level. The appliance has its own forwarding table, each row containing layer 4 (TCP) address in the form of IP:port. The 3 fields of the forwarding table will be: 1. Initiator address., 2. Target address, 3. Next hop. The next hop represents the local IP:port address used to bind the socket from the switch to the next target. The switch stays between initiators and targets, acting like a proxy for iSCSI protocol.
It will act as a server for initiators and as a client for the targets. The iSCSI behaves as a "man in the middle", presenting itself to the initiators as an iSCSI target: it will listen on standard iSCSI port 3260 accepting TCP connections from initiators. Once an initiator is connected, switch will act like an iSCSI initiator for the real target founded in the switch table: it will initiate a connection to the target, presenting itself as an initiator (client).

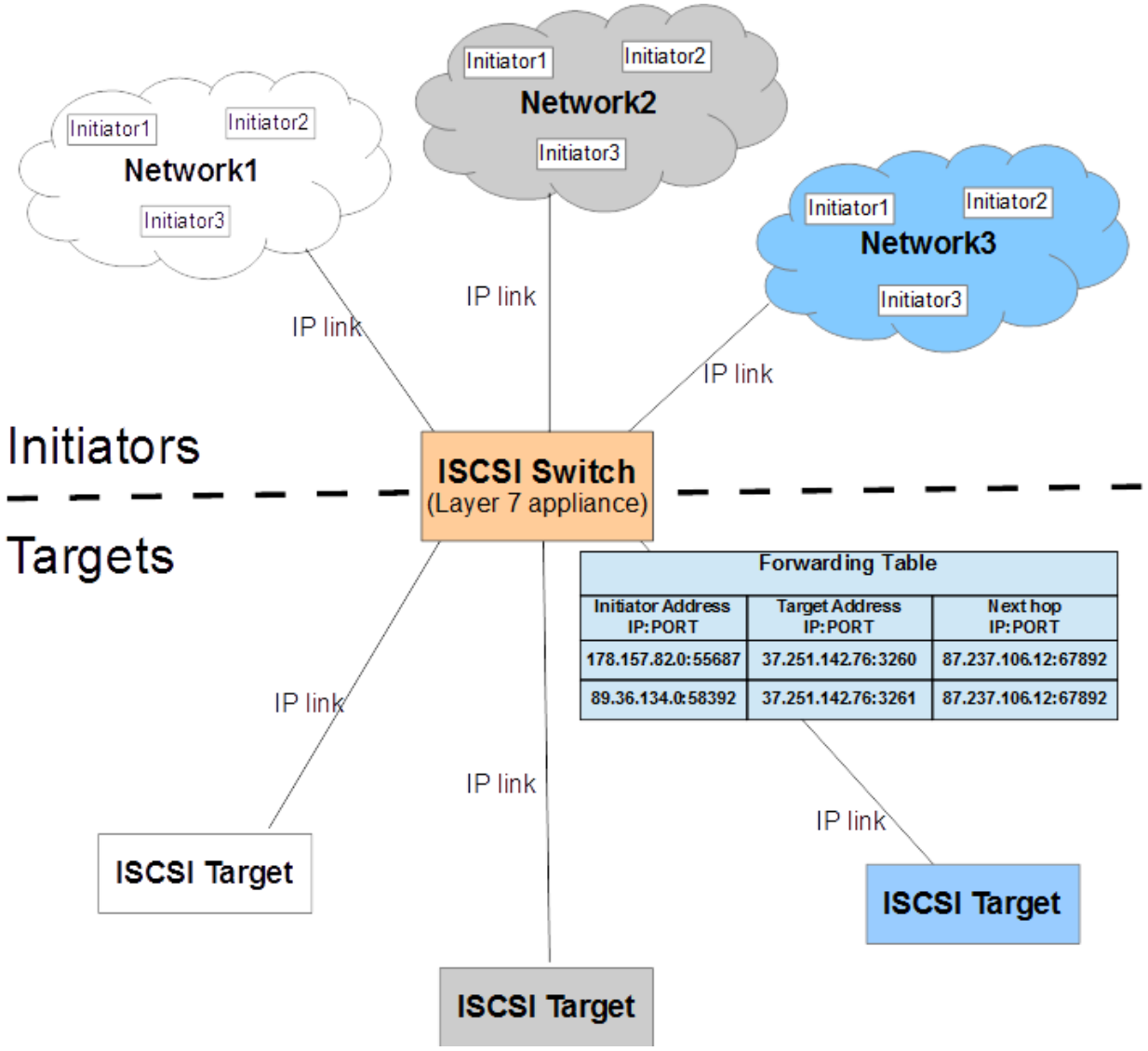

Fig 7. iSCSI Switch architecture

2.2 Functional flow diagram

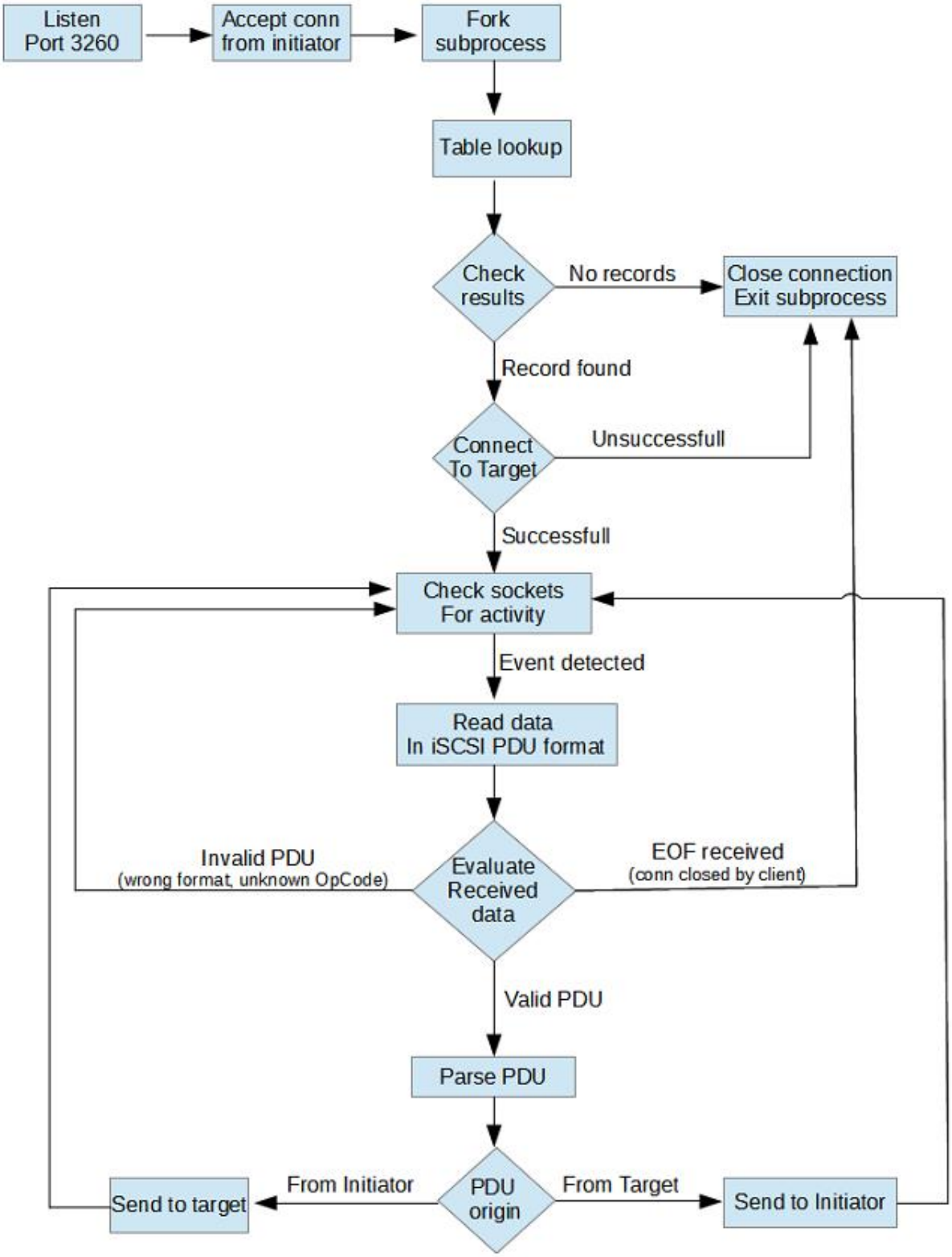

Fig 8. iSCSI Switch functional flow diagram.

As of RFC3720, iSCSI server (target) and initiator (client) talk each other by exchanging iSCSI protocol data units (PDUs) on an asynchronous TCP socket. iSCSI PDU is in the form of header+payload, header containing a field with the payload length. Each iSCSI PDU always starts with a 48 bytes fixed length header (the basic header segment or BHS) followed by payload of variable length. The length of the payload is contained in the BHS: 3 bytes of data segment length + 1 byte of additional headers length. Server and client reads the 48 bytes BHS, calculates the payload length and then read the remaining bytes of payload.
The Switch will act like a "man in the middle", presenting itself to the initiators as an iSCSI target: it will listen on standard iSCSI port 3260 accepting TCP connections from initiators. Once an initiator is connected, Switch will act like an iSCSI initiator for the real target: it will initiate a connection to the target, presenting itself as an initiator (client). Then will handle two TCP sockets: the socket from initiator and the socket to target. The target address (IP:port) is retrieved by a lookup in the switching table. Note that the target address is a layer 4 address, it composed by the layer 3 IP address and the port. This allows multiple instances of an iSCSI target to run on the same host.
I will use the following terminology:

1. Initiator socket: the socket between iSCSI initiator and Switch.
2. Target socket: the socket between Switch and iSCSI target.

The Switch software will use the (asynchronous) poll mechanism to check activity on the two sockets: It will poll the initiator and the target socket to check for activity. There are the following possibilities:
1. If the initiator socket is found readable, then Switch will read an iSCSI PDU from the socket: first it will read the 48 bytes BHS structure, parse it, determine the payload length, read the payload. At this point, the PDU is read from the initiator socket, and will be written to the target socket.
2. If the target socket is found readable, Switch will do the same as above but in reverse order: it will read an iSCSI PDU from target and will write it to the initiator socket.
In this way, Switch will be able to intercept PDUs exchanged between initiators and target. PDUs will be dissected, all headers will be parsed, and the output will be displayed or logged for analysis purposes.

Design option: for each initiator accepted by the Switch as client, a connection to the iSCSI target will be made and Switch will fork a child process that will handle the initiator and target sockets using poll mechanism. I prefer fork instead of threads for robustness reasons: treads resides in the main process address space and if a thread crashes unexpectedly than the main process and all other threads can be affected.

Along with the two sockets (initiator and target described above) the switch will have a third socket opened to a log server (included in the distribution), and will send PDU informations for tracing and logging purposes. As an additional utility, I have developed a graphical console written in Java (a client for the log server), for real-time displaying the iSCSI traffic flowing through the proxy.

2.3 Benefits of an iSCSI switch. A real world scenario.

First, the iSCSI switch provides a method for interconnecting initiators and targets situated in different networks. The iSCSI switch is a layer 7 device in the OSI model: it understands the iSCSI protocol and it can be used also as a protocol analysis tool. It will provide the possibility to take a deeper look into iSCSI transactions, helping storage and network administrators to troubleshoot and identify protocol errors or deviations from standards and avoiding potential data loss.

Identifying protocol-level problems such as an interoperability issue that can cause further problems is crucial in providing reliable storage services to industry. At this moment there are many protocol analysers/debuggers on the market, and some of them, for example Wireshark and EtherReal, are opensource. However, these tools are generic, none of them are dedicated to the iSCSI protocol. Moreover, these tools are basically packet sniffers built using the libpcap library and they are not able to health checking/modifying iSCSI protocol data units.

I consider the iSCSI switch useful for storage networking administrators for the following main reasons:

1. Short time: It provides an interface to real-time debug an iSCSI link. Within this interface, administrators can see in details what's happening in the iSCSI link during different phases of the protocol (for example in the negotiation phase when initiator and target exchanges link parameters) and detect/fix misconfigurations.

2. Long time: It provides an interface to standard log servers of any type (for example syslogd). Many commercial log servers comes with features like: database backend, report builder, alert builder, real time filtering. Using "tracer to log server" feature, the iSCSI link events are logged in a standard way and, and log messages will provide material for building various types of log reports, filtering and alerts that will help administrators to detect and prevent undesirable situations.

Beside these benefits, an iSCSI switch offers the possibility of filtering storage traffic in a transparent manner (initiators and targets will not be aware about the presence of the switch). Along with PDU health check, various filtering policies can be imagined that will fill the security gaps of the iSCSI protocol.

I will present a real world scenario involving the iSCSI switch in building a topology for switching the iSCSI traffic. To illustrate the iSCSI switch role I will use the STAR model (Situation, Task, Action, Result).

1. Situation: A company has different geographically distributed branches, each branch having its own IT infrastructure (LAN, network equipment, servers and workstations, NAS storages) called “site”. Each site contains an “iSCSI island” consisting in iSCSI initiators and targets situated in one or more VLANs.

2. Task: The Company wants to interconnect the iSCSI islands from all the sites. There are many motivations for this: remote backups, redundancy for iSCSI targets, backup paths between several iSCSI initiators and targets, disaster-recovery compliance and so on.

3. Action: The task can be accomplished by doing several setups at Layer 3 or 4 on the existing network equipment. The initiators and targets IP addresses have to be translated using NAT configurations on the ISP network equipment and also there must be some configurations at each

initiator and target (default gateways or at least a static route). There are several drawbacks in choosing this solution like for example many equipment from different vendors must be configured one by one in all the sites, and when all configurations are done it comes the problem of maintenance. But the major disadvantage is that we don't have access to configure the ISP network equipment (routers, gateways or NAT devices). So we decide to build a topology containing iSCSI switches to interconnect the iSCSI island from the remote sites. In each site an iSCSI switch will be installed and all iSCSI switches will be interconnected. Within a site, all iSCSI initiators will be configured to use the local iSCSI switch as target. On each iSCSI switch we will configure the forwarding tables using the web based interface. This is very similar to configuring permanent virtual circuits (PVCs) in a frame relay setup. Now it becomes clear that a topology formed by iSCSI switches is very similar to a circuit switched network, as opposed to a packet switched network. When a connection comes, the iSCSI switch will get the layer 4 address (IP:port) of the client socket and will do a lookup in the forwarding table to determine the layer 4 address (IP:port) of the next switch and also the next hop which is the local layer 4 address used to bind the new socket. This "next hop" layer 4 address will be used by the next iSCSI switch for the lookup in its own forwarding table and finally the circuit is established and the iSCSI traffic will flow.

The figure below illustrates an example of how initiators and targets from multiple sites can be interconnected using iSCSI routers. For example, an initiator form Site1 can connect to a target in the Site4 using a route through Site2 and a backup route through Site3 in a mesh topology.

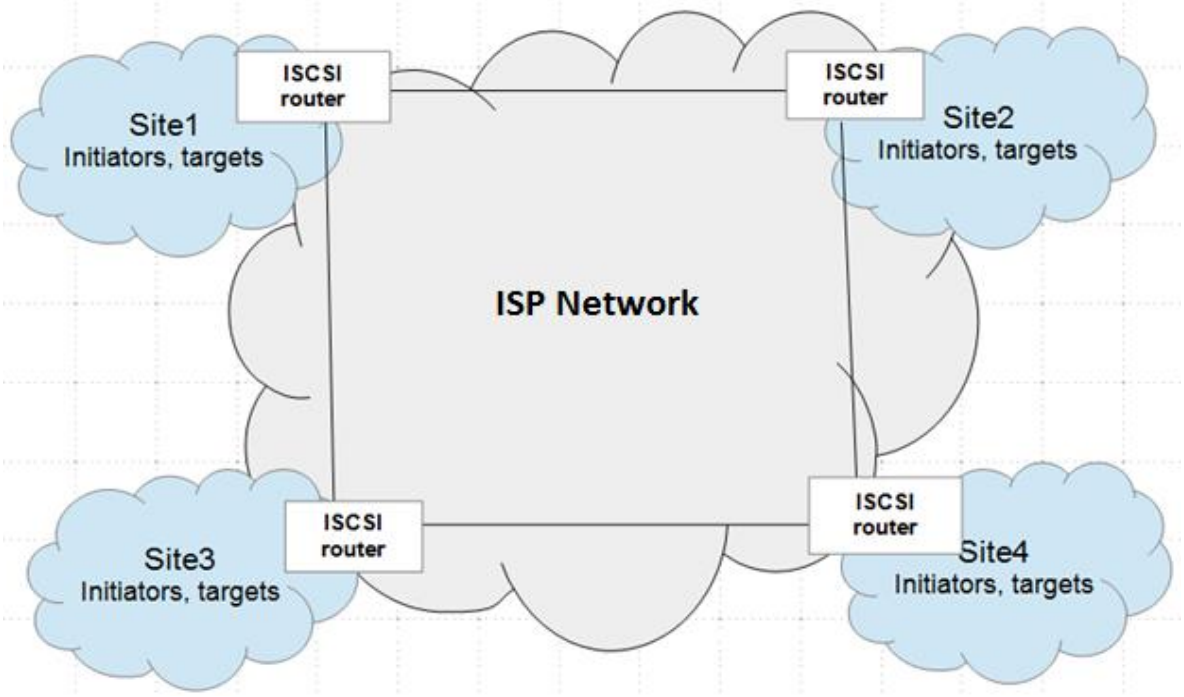

Fig 9: Mesh topology using iSCSI switches

4. Result: The iSCSI islands from all the customer sites are now interconnected and the iSCSI traffic flows. In addition, he have provided a web based management for configuration, monitoring and debugging. Can be imagined a central management station and a unified configuration interface for all the switches: each switch in the topology is configured from the central web interface and then the configuration is applied to the remote switch. Using the java interface (that can run in the browser as an applet) the administrator can debug and monitor the iSCSI traffic in real-time for each switch in the topology, and this will provide centralized control and overview of the entire setup. The iSCSI switch is a protocol aware device enhanced with logging/debugging capabilities related to the iSCSI protocol thus can be considered an iSCSI device with practical applicability in real world scenarios.

2.4 Software architecture and development tools:

iSCSI switch will be developed in POSIX C, using POSIX standard libraries and system calls, and the GNU gcc compiler on x86_64. Network interaction will use asynchronous poll/select socket calls. As OS will be used community enterprise Linux (CentOS) installed on virtual machines using VMWare as hypervisor. The release will be tested using standard iSCSI Linux packages: iscsi-initiator, iscsi-target utils, Linux SCSI target framework (tgt).

Because of its compiled nature, C is faster than scripting languages (like python) and provides better control in memory management than Java. On the other hand, C is lower-level language and development is slower than in high-level programming languages like python or Java, but in this application runtime time is more important than programming time.

For the "switch table" I have used MySQL as database backend. The table can be edited from the MySQL console and also within a web interface using for example the opensource phpMyEdit which is a simple script for editing MySQL tables.

The Java client for the log server can run on any desktop computer (OS independent) and was developed using Oracle jdk1.7 and Java Swing Application Framework for graphical interface. Beside this graphical interface, I have built also a commandline utility.

Bibliography:

[1] Richard Stevens: "Unix Network Programming vol 3"
[2] Michael Kerrisk, "The Linux Programming Interface"
[3] Jan Bodnar: "Java Swing Layout management"
[4] www.zetcode.com Tutorials for programmers
[5] RFC 3720, 3721, 3260
[6] Peter Radkov, Li Yin Pawan Goyal, Prasenjit Sarkar and Prashant Shenoy "A Performance Comparison of NFS and iSCSI for IP-Networked Storage"
[7] James Long, "Storage Networking Protocol Fundamentals"
[8] Tom Weimer , "Fibre Channel Fundamentals"
[9] SNIA e-course: "Storage Networking management and administration"