DESIGN AND IMPLEMENTATION OF iFCP

BY

CLAIRE KRAFT

Bachelor of Arts, University of Colorado, 1994

THESIS

Submitted to the University of New Hampshire
in Partial Fulfillment of
the Requirements for the Degree of

Master of Science

in

Computer Science

May, 2004

This thesis has been examined and approved

This thesis is dedicated to my parents and my brother.

ACKNOWLEDGEMENTS

I would like to thank Professor Russell for being my advisor, for taking the time to teach me so many valuable things, and for being patient with me as I processed them. I would also like to thank him for helping me learn from countless situations, both academically and otherwise. I would like to thank Professor Bartos and Scott Valcourt for providing me with feedback and support, and for being a part of my committee. I am very grateful to both of them for this. I would like to thank David Woolf for the time and technical expertise he has shared with me over the years as a colleague, and for his editorial contributions to this document, especially during its early stages. I would like to thank my boss, Stephen Schaeffer, for leading the UNH InterOperability Lab in its support of this project, in terms of both time and equipment. I would also like to thank the people of Medusa and Finisar for volunteering their time to provide us with a great deal of feedback regarding the traces which were taken over TCP/IP, so that we could clean them up as quickly and confidently as possible. Finally, I would like to thank my family for all of their support.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

DESIGN AND IMPLEMENTATION OF iFCP

by

Claire Kraft

University of New Hampshire, May, 2004

A SAN, or Storage Area Network, is a network established in a similar manner to a Local Area Network, or LAN. However, unlike the LAN, the SAN is designed for the sole purpose of establishing a direct connection between a host server and storage devices such as RAIDs, tapes or hard drives. The SAN is useful because it creates a means by which the storage bus can essentially be extended beyond the physical limitations of the host bus itself, and it allows data to be transferred in blocks. Fibre Channel is one of the fastest SAN technologies in existence, and SCSI is one of the most efficient protocols used for the storage of data. Together, Fibre Channel and SCSI provide a very effective storage vehicle.

One of the major drawbacks for those who rely solely on Fibre Channel is distance. Even when various methods of signal enhancement are in place, Fibre Channel signals can only be transmitted over distances of several hundred kilometers. Recently, however, disaster recovery concerns have arisen which have resulted in the promotion of storage which takes place over longer distances, sometimes crossing international boundaries. Additionally, Fibre Channel media and equipment can often be expensive and cumbersome, both to install and manage. Although backbone technologies do exist which can carry Fibre Channel data over longer distances, their installation, cost, and maintenance would present many difficulties that could be handled easily by a technology such as TCP/IP, which is already in place across the world. Thus, the idea of developing SAN technologies which run over TCP/IP has arisen, sometimes as a result of the desire to connect small-scale SANs over longer distances, and sometimes as a result of the desire to replace other SAN technologies altogether. These factors, combined with the fact that the Internet has become so widespread and convenient, are among the primary motivations for this thesis, and for the rising popularity of storage over TCP/IP.

iFCP is one form of storage over TCP/IP that allows hosts and Fibre Channel storage devices to communicate directly. It is an encapsulation protocol that dictates the means by which Fibre Channel frames become the payload in an iFCP message. In addition, iFCP introduces a few new types of messages for purposes of control. This thesis is comprised of the design and implementation of iFCP end devices. The initiator has been implemented as a software module that behaves like a Fibre Channel Host Bus Adapter with an attached encapsulator. The target has been implemented as a stand-alone software program that acts both as an encapsulator and as a Fibre Channel switch that is attached through a generator to a Fibre Channel disk.

I STORAGE AREA NETWORKING

1.1 Introduction

A Storage Area Network (SAN) is a network established in a similar manner to a Local Area Network (LAN). Like the LAN, the SAN is a group of computers in relatively close proximity sharing a common communication network, and one or more servers. However, unlike the LAN, the SAN is designed for the sole purpose of establishing a direct connection between a host server and storage devices, such as RAIDs, tapes or hard drives. The SAN is useful because it creates a means by which the storage bus can be extended beyond the physical limitations of the host bus itself, allowing data to be transferred in blocks. Additionally, SANs allow storage devices to be shared by multiple hosts, regardless of platform. Fibre Channel is one of the fastest SAN technologies in existence, and SCSI is one of the most efficient protocols used for the storage of data [14]. Together, Fibre Channel and SCSI provide a very effective storage vehicle.

Distance is one of the major drawbacks for those who rely solely on Fibre Channel. In order to promote disaster recovery, particularly in areas experiencing frequent earthquakes, Fibre Channel was originally designed to allow storage to take place over distances of up to approximately 10 km from hosts. Even by using various methods of signal enhancement, which might allow the distance to expand by several hundred kilometers, storing data over distances hundreds or thousands of miles is out of the question for devices connected merely by Fibre Channel cables. More recently, other disaster recovery concerns have arisen which have resulted in the promotion of storage which takes place over longer distances, sometimes crossing international boundaries [15]. Additionally, Fibre Channel media and equipment can often be expensive and cumbersome, both to install and manage. Although backbone technologies do exist which can carry Fibre Channel data over longer distances, their installation, cost, and maintenance would present many difficulties that could be handled easily by a technology such as TCP/IP, which is already in place across the world. Thus, the relationship between SAN technologies which run over TCP/IP and other SAN technologies can be considered similar to that between Wide Area Networking and the LAN. Additionally, some SANs which run over TCP/IP can sometimes replace other SAN technologies altogether. These factors, combined with the fact that the Internet has become so widespread and convenient, are among the primary motivations for this thesis, and for the rising popularity of storage over TCP/IP.

1.2 Storage Over Fibre Channel

Fibre Channel provides a means by which storage takes place across a serial network, usually over Fiber Optic cable, but copper is used as well. Currently, data is typically stored at rates of 1 to 2 Gbps over Fibre Channel, but speeds of 4 and 10 Gbps are also supported.

In order to facilitate the communication that takes place between Fibre Channel devices, some devices act as initiators, and some act as targets. Additionally, accessory devices, such as switches and bridges, may be present in any given Fibre Channel network. The initiators and targets are known as end devices, and they are usually either servers or storage devices such as disks, RAIDs, and tape drives. The servers are the initiators, since they contain the applications that originate service and task management requests to be processed by the targets. A target is a storage device that receives these requests from an initiator and guides them to the appropriate locations, where they are then executed. The servers must therefore initiate the actual processing mechanism whereby information is stored and retrieved from the storage devices. The storage devices must simply wait for this to take place, and respond appropriately to commands as the initiators issue them.

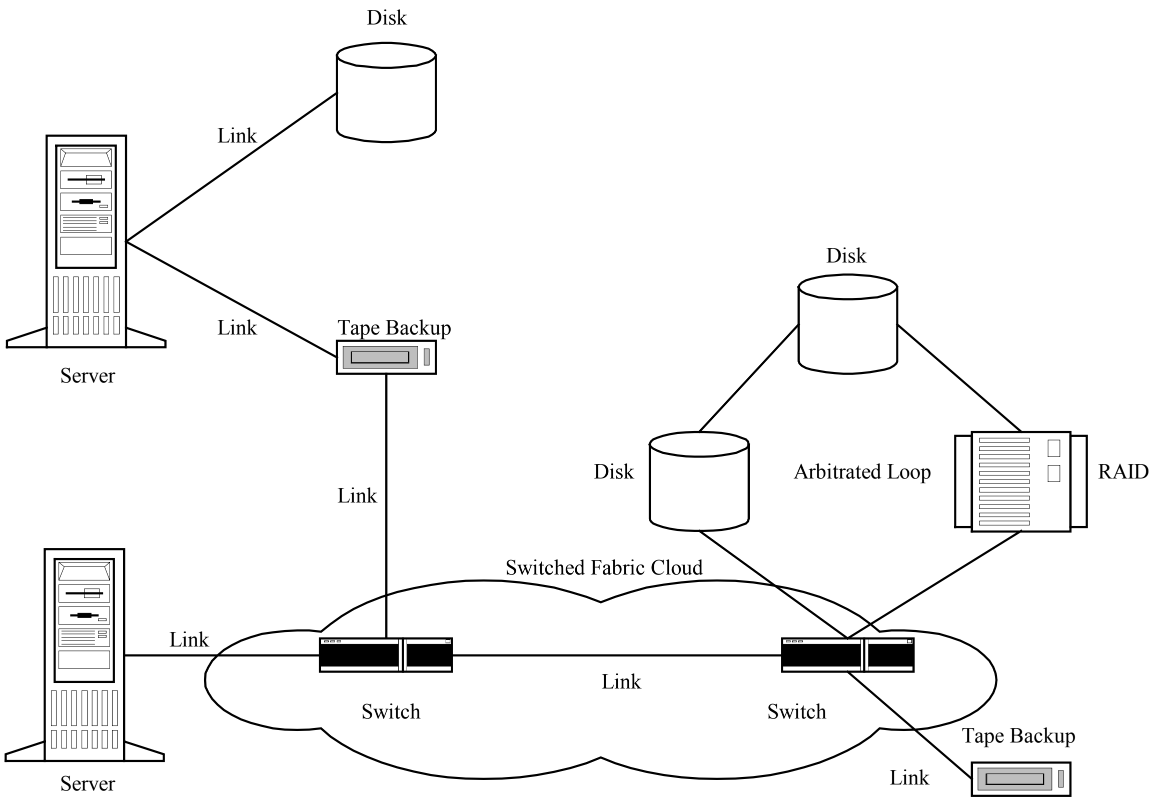


Figure 1: Fibre Channel Network

Once any combination of Fibre Channel devices are physically attached, they must initialize in order to gain access to one another. In this manner, the devices create either a link or an Arbitrated Loop (Figure 1). In any link, only two devices are present, and a generic addressing scheme is used. On the other hand, a loop among Fibre Channel devices may involve 2 to 126 end devices with or without the addition of one switch. The Arbitrated Loop has been defined in order to allow multiple end devices to exchange information without depending on a switch, thereby reducing costs and other maintenance issues introduced when a network is excessively reliant upon accessory devices. For this reason, a specific addressing scheme must be used on a loop in which each device is identified in a unique manner. Any number of loops and links may also be interconnected via switches, and a switch, or fabric, topology may consist of one or more links between switches and other switches or end devices.

After two Fibre Channel devices have initialized, data may be transferred between them in either direction via frames. This data transfer may or may not be connection-based, depending on the class of service. Analogous to a packet in IP, a Fibre Channel frame is the unit, ranging from 256 to 2112 bytes in length, which carries a segment of data over the Fibre Channel bus. Frames contain fields which provide the Fibre Channel protocol with its own addressing scheme, sequence identifiers, exchange identifiers, Cyclical Redundancy Checks for error detection purposes, and other elements. Additionally, the Fibre Channel protocol guarantees that these frames are delivered in order between initiators and targets, if they are directly connected. However, in-order delivery between an end device and a switch is negotiable for certain classes of service. A frame sequence consists of one or more Fibre Channel frames, and any given sequence may take place in only one direction. An exchange may take place either in one direction or both, and it is comprised of one or more consecutive frame sequences. The Login frames, which are exchanged between devices directly after initialization, allow the devices to convey various operational parameters to one another, such as protocol version, buffer space, and class or classes of service supported. This exchange eventually leads to those involving the actual I/O requests and data transmission.

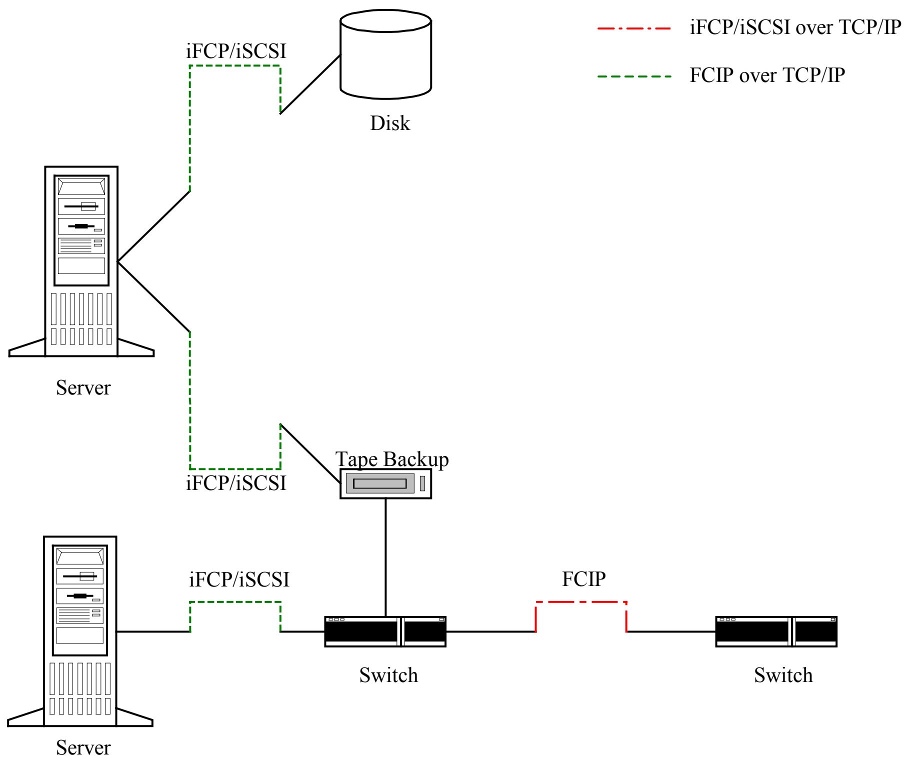


Figure 2: FCIP, iSCSI, iFCP

1.3 Storage Over TCP/IP

Three of the primary storage over TCP/IP protocols that have been defined by IETF are FCIP [7], iSCSI [8], and iFCP [9] (Figure 2).

1.3.1 FCIP

FCIP is a tunneling protocol used to connect two Fibre Channel switches via a TCP/IP connection which runs between their respective FCIP Link Endpoints. FCIP is known as a tunneling protocol as the FCIP link existence is not made known to any end devices connected to the switches. The Fibre Channel frames being utilized are a specific set of frames transmitted only between switches. FCIP is designed to merely provide an encapsulation mechanism that allows these point-to-point Fibre Channel frames to travel over the IP network, and it therefore does not provide any additional functionality or routing capabilities to the FCIP Entity.

1.3.2 iSCSI

iSCSI is designed to transport Protocol Data Units, or PDUs, over TCP/IP connections between the iSCSI Ports which reside on end devices. Unlike FCIP and iFCP, the iSCSI protocol does not incorporate Fibre Channel frames, and may thus be used by a device as a replacement for Fibre Channel altogether. In this protocol, multiple initiators may connect with a given target, and a single initiator may connect with multiple targets. Additionally, multiple connections can exist between any given pair of devices. Every connection, or set of connections, which exists between an initiator and a target is known as a session. In order to avoid confusion, initiators and targets, sessions and connections must all be uniquely identified. Furthermore, iSCSI initiators and targets may negotiate a large number of parameters, dictating which authentication method, if any, shall be used, PDU sizes, PDU burst sizes, whether CRCs are used, and many other features. Some of the negotiated features are applicable only to single connections, while others apply to entire sessions. After the parameter negotiation phase has taken place, the iSCSI nodes exchange encapsulated SCSI Commands in order to execute actual I/O operations.

1.3.3 iFCP

Aside from the fact that all three protocols utilize TCP/IP, the iFCP protocol shares some characteristics with FCIP, and some others with iSCSI. Like iSCSI, iFCP transmits packets between initiators and targets. The iFCP portion of an end device is known as an iFCP gateway. However, unlike the iSCSI node, the iFCP gateway does not negotiate many operational parameters. Additionally, like the FCIP link endpoint, the iFCP gateway transmits encapsulated Fibre Channel frames. However, unlike the FCIP link endpoint, the iFCP gateway may also employ its own addressing capabilities in order to route the frames which it sends and receives. Therefore, the iFCP gateway must perform two functions, aside from those associated with establishing the actual TCP/IP connection with another iFCP gateway: it must encapsulate Fibre Channel frames, and it must replace their destination address fields with others, if necessary.

1.4 Summary

The remainder of this thesis will concentrate upon the topics that have been introduced in this chapter which directly relate to the iFCP implementation discussed here.

Chapter 2 presents Fibre Channel, the primary building block of iFCP, in more detail. It provides an overview of Fibre Channel layering and framing, along with a basic depiction of the exchanges which take place at different layers in order to establish the Fibre Channel session.

Chapter 3 explores the methodology behind the implementation of iFCP. Here, TCP/IP essentially replaces the lower layers of Fibre Channel, and the manner in which the iFCP Session Control frames along with its encapsulation mechanism bring this protocol transition about is traced and depicted.

Chapter 4 focuses on the iFCP target and initiator implementations on which this thesis is based. Although the target and initiator share similar frame building and encapsulation methods, their mechanics are quite different. The initiator transmits and receives frames over TCP/IP only, and it interfaces with the SCSI Midlevel of the host. The target transmits and receives frames over both TCP/IP and Fibre Channel, and instead of interfacing with the SCSI layer of a host, it interfaces with a Fibre Channel disk through a Fibre Channel generator.

Chapter 5 discusses the performance tests that were run upon the completion of the iFCP target and initiator implementations. The strengths and weaknesses of the implementations are examined and discussed, and conclusions are drawn regarding what the implementation might be useful for, and the future work which may be done as a result of its creation.

II FIBRE CHANNEL

In order to clearly trace the processes that are integrated into iFCP, the Fibre Channel protocol must be examined. However, Fibre Channel is an extensive technology which incorporates multiple specifications, and it would be impossible to completely depict all of its details in this thesis. Additionally, a great deal of information would be extraneous with regard to iFCP. This supplementary chapter is therefore provided not as a complete Fibre Channel discussion, but as a mere introduction to some of its fundamental concepts.

2.1 Layering

Fibre Channel is a Storage Area Networking protocol comprised of five layers, FC-0 through FC-4, which correspond to layers one through five of the OSI model. The chart in Figure 3 [12] attempts to draw parallels between the Fibre Channel layering and the theoretical OSI model, for purposes of comparison:

OSI Model	Fibre Channel
L7 – Application	
L6 – Presentation	
L5 – Session	L5 – SCSI, IP (and others)
L4 – Transport	L4 – Common Services
L3 – Network	L3 – Framing
L2 – Data Link	L2 – Encode/Decode – 8bit/10bit
L1 – Physical	L1 – Physical Layer – Optical and Copper

Figure 3: OSI vs. Fibre Channel Comparison

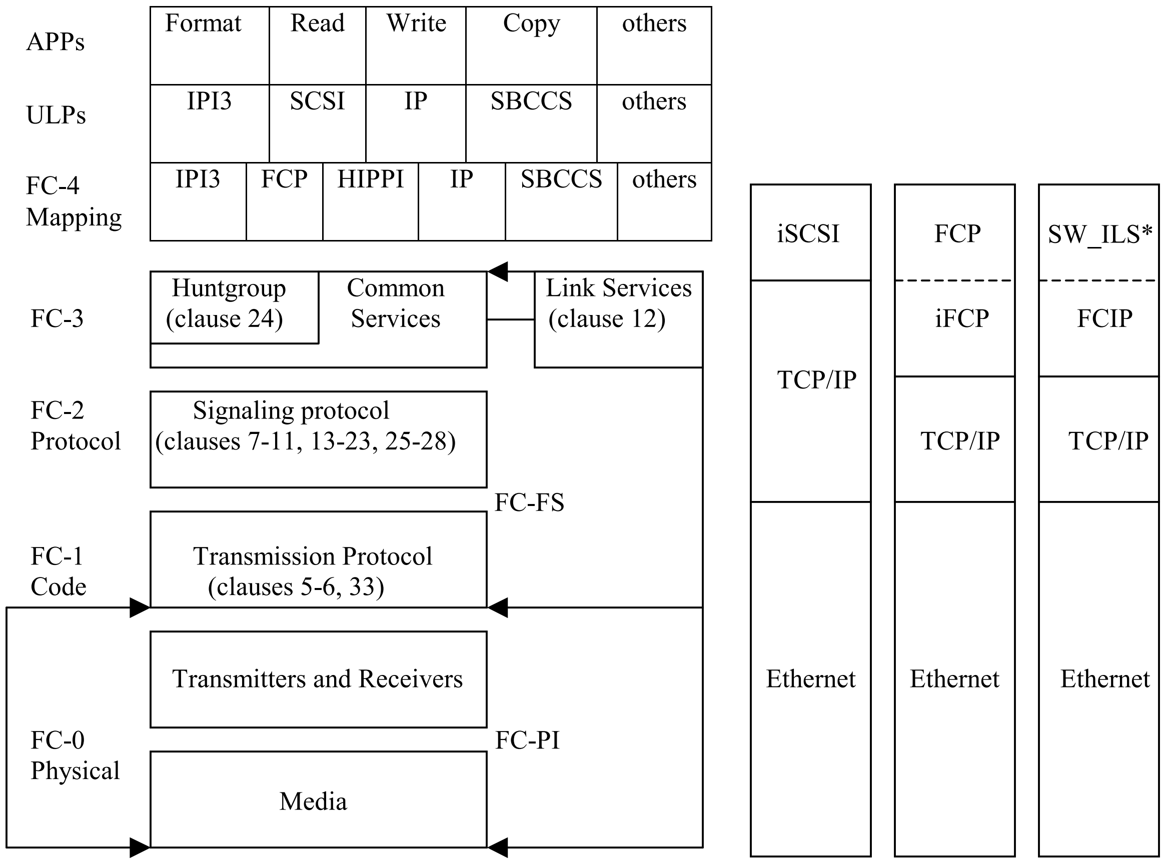


Figure 4: Fibre Channel Structure
* Switched Fabric Internal Link Services

The chart in Figure 4 [1] contains the actual Fibre Channel layers with their proper labels. An application layer has been added to the top which mentions some of the operations that might actually utilize these layers.

Layers FC-0 through FC-4 must be established prior to an action such as a read or write is performed, involving data stored on a Fibre Channel storage device. When the application layer of a given host issues an I/O request, an Upper Layer Protocol (ULP in Figure 4), such as SCSI, is called upon by the operating system to initiate the transaction.

A SCSI target must contain at least one Logical Unit. The Logical Unit is defined by SAM-3 as: a SCSI target device object, containing a device server and a task manager, that implements a device model and manages tasks to process SCSI commands sent by an application client [6]. Very often, an entire target will be treated as one Logical Unit. Each Logical Unit is addressed using a different Logical Unit Number, or LUN. The SCSI layer must open each LUN as a separate device [13], during which time it will build the initial SCSI Commands required to probe the LUNs. The SCSI layer stores the information necessary to fulfill the I/O request as dictated by the SCSI protocol, along with its type or Opcode, in a Command Descriptor Block, or CDB. The CDB, the buffer space necessary to hold the request, and the data conveyed by the request, are all queued by the SCSI layer which utilizes a Scsi_Cmnd data structure that is subsequently accessed by the FC-4 layer.

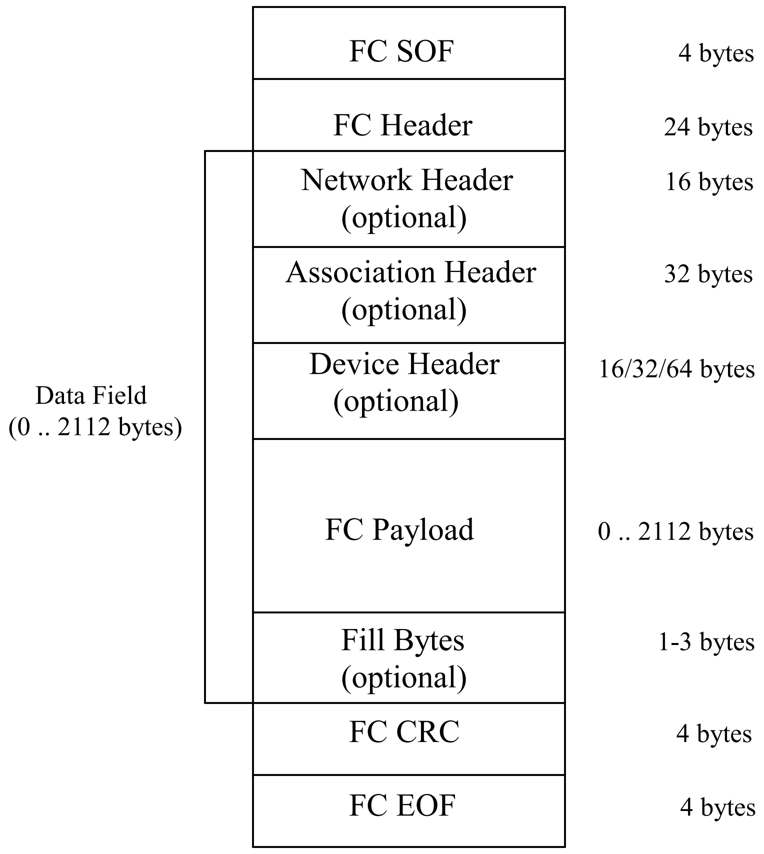


Figure 5: Fibre Channel Frame

FC-4, the mapping layer, is the highest layer in the Fibre Channel protocol, carrying various upper level protocols via Fibre Channel frames (Figure 5) [1]. FCP is one of the most common examples of a FC-4 level protocol. FCP provides a mechanism to map SCSI Commands onto frames. This mechanism consists of the Fibre Channel header, followed by three optional headers, followed by a Fibre Channel payload which contains a LUN field as well as other data such as the Command Reference Number and Task Management flags, in addition to the original SCSI CDB [3]. Another commonly utilized component of the FC-4 layer is the Generic Service protocol. Generic Services provide the means for end devices to register with switches as devices capable of performing I/O transfers, and for initiators to query a switch to discover which targets have registered as such.

The FC-3 layer is comprised of Basic and Extended Link Service Fibre Channel frames [1]. Extended Link Service (ELS) frames are transmitted by their originators in order to cause specific functions to be performed by their recipients. For example, ELS frames exist which request port logins and logouts, along with more obscure events in Fibre Channel such as clock synchronization updates. Process Login frames are also ELS frames, and must be exchanged before proceeding with the Upper Level functions provided by FCP. Basic Link Service (BLS) frames are designed to perform a few relatively simplistic functions, and unlike ELS frames, BLS frames may be transmitted prior to a successful login. In addition, FC-3 provides services that are designed to reach multiple ports within a Fibre Channel device. The usage of the multicast mechanism of FC-3 is very specialized, and exists exclusively in relatively unique systems, such as real time systems.

The FC-2 layer is perhaps the most varied of the FC layers, in terms of format. Besides defining all of the data bytes in their decoded form, thus providing a transport layer for all frames, the FC-2 layer also defines all of the small Loop Initialization frames in their entirety, along with a few other small frames, such as the ACK frame used for end-to-end frame reception acknowledgement (see Appendix A) [1][2].

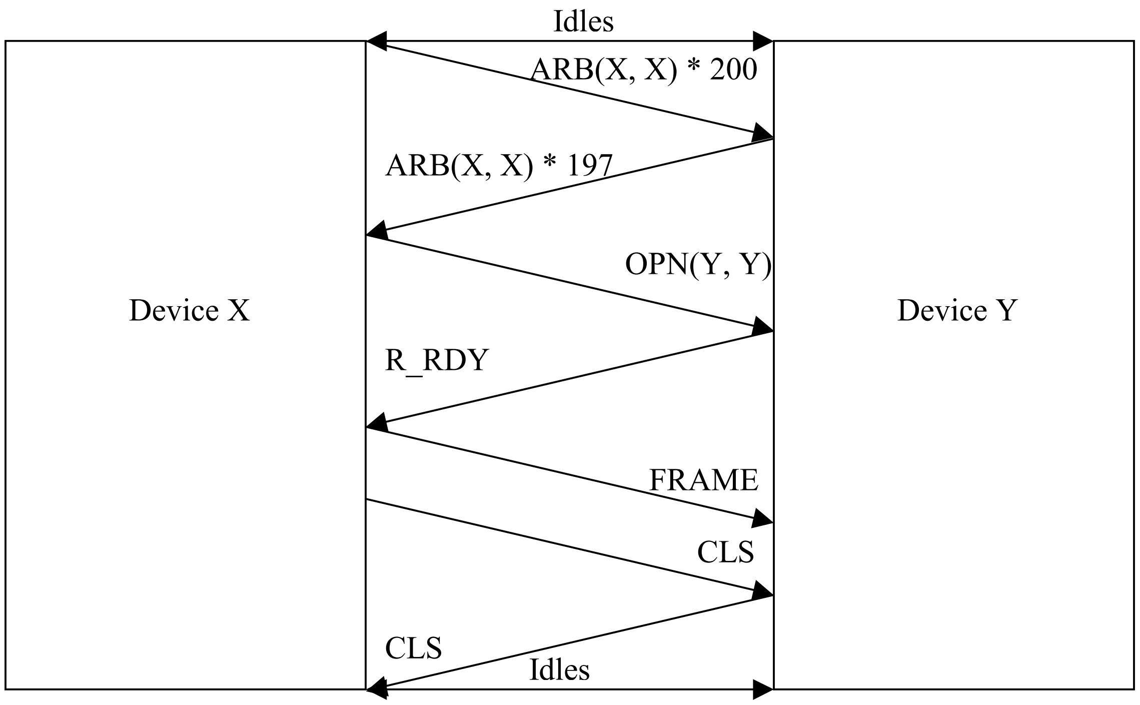


Figure 6: Primitives and Frame Transmission Example (Loop)

Please note: Figure 6 is intended to superficially depict one typical scenario that can take place on a given Fibre Channel Arbitrated Loop, and half-duplex operation is utilized. To further examine the intricacies of the Arbitrated Loop State Machine, please refer to the FC-AL-2 Standard [2].

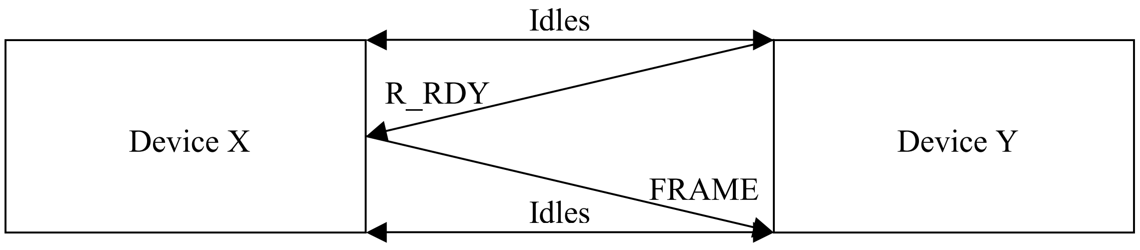


Figure 7: Primitives and Frame Transmission Example (Link)

In addition to the bytes incorporated into frames, the FC-2 level includes those bytes which reside within ordered sets [1][2]. Ordered sets are a specific group of transmission words, or four-byte quantities at the FC-2 level, with the comma character, 0xBC, in the leftmost position. Ordered sets exist either in the form of either Primitive Signals or Primitive Sequences, or as frame delimiters (see Appendix B, C, D). Both types of primitives are transmitted between frames (Figures 6 and 7), but the Primitive Signal must be detected upon the reception of only one ordered set. Primitive Sequences are expected to occur in spurts, and the detection of a Primitive Sequence is only required if at least three have been received at any given time. The Loop Initialization Primitive Sequence (LIP), originated by a port that needs to cause Loop Initialization to take place, would be an example of a Primitive Sequence. One example of a Primitive Signal would be the ARB Primitive Signal, transmitted by a Fibre Channel device in order to arbitrate for control of the Arbitrated Loop, prior to frame transmission. If, upon receiving its own ARB, this device discovers that it has won Arbitration, it will transmit the OPN Primitive Signal. If the recipient of the OPN contains enough buffer space to adequately process an incoming frame, the recipient indicates that this is the case by transmitting an R_RDY Primitive Signal to the potential frame sender (see Appendix L). Otherwise, the recipient should indicate the fact that it is unable to receive any frames by transmitting a CLS Primitive Signal. However, if the sender does receive an R_RDY, it transmits the frame, followed by CLS. The basic process of sending frames from one device to another over an Arbitrated Loop will be reviewed again in Chapter 4, with regard to both frames and frame sequences.

In addition to the primitives, the Start Of Frame (SOF) and End Of Frame (EOF) ordered sets must be sent at the beginning and end of each frame. Different types of frame delimiters are defined for different classes of service, allowing easy recognition in situations where some classes might need to be differentiated from, and even preempted by, others. Additionally, different types of frame delimiters are also used to denote frames that are being transmitted in the middle of a sequence.

The FC-1 layer provides the transmission protocol [1]. The transmission characters residing at the FC-1 level are encoded 10 bit entities, while the FC-2 layer consists of 8 bit unencoded versions of the same characters. The 8B/10B encoding scheme, which is also used in Gigabit Ethernet, provides a set of tables dictating the method by which each 8 bit byte is converted into a 10 bit transmission word prior to its transport over the wire, along with the reverse process for each word after reception from the wire. Many bit errors can, therefore, be easily detected during the decoding process.

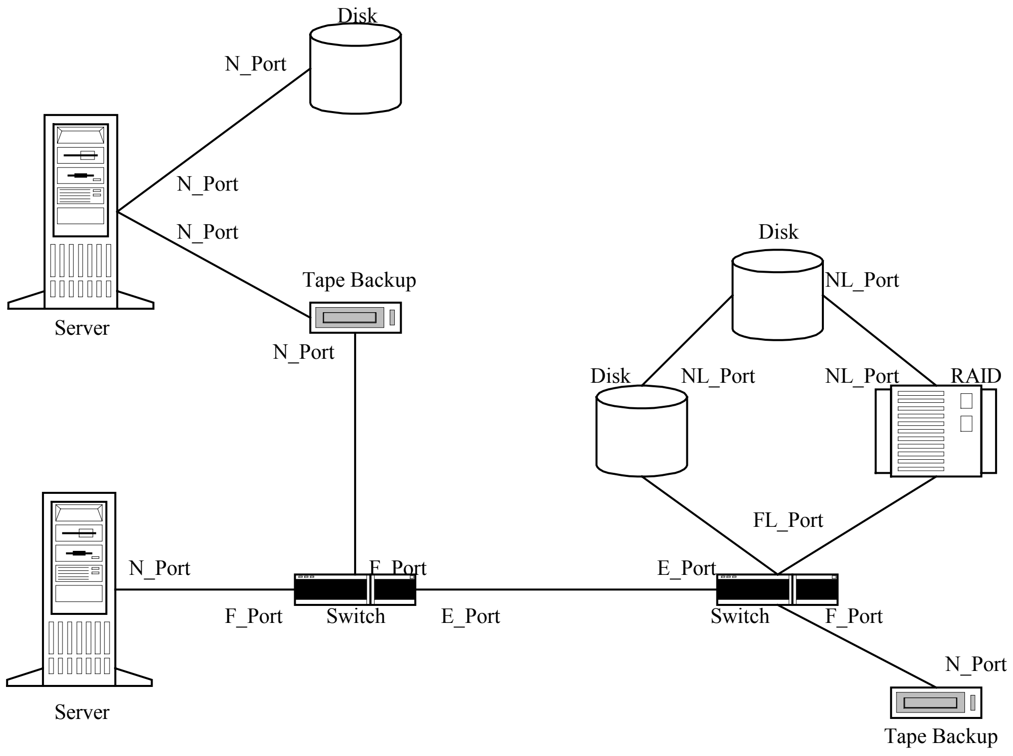
The Physical Interface of Fibre Channel (FC-0) defines different media, transmitters, connectors and receivers [1]. The media used generally consist either of copper, in forms such as coax and twisted pair, or fiber optics. Their present speeds of operation usually range from one to four Gigabits per second.

2.2 <u>Fibre Channel Session Execution</u>

Once they have been physically connected, Fibre Channel devices must proceed to synchronize and perform either Loop or Link Initialization. After initialization, login is performed, followed by the actual session operations. The session is comprised of operations such as those involving switch registration and device I/O. Finally, devices which explicitly log out with one another may gracefully terminate a session.

2.3 <u>Fibre Channel Topologies</u>

The mechanism used by Fibre Channel and other SANs in order to establish a relationship among servers and storage devices, or end devices, involves both initiators and targets. Initiators are generally Host Bus Adapter cards located in host machines, and targets are typically either block storage devices, such as disks or RAIDs, or sequential access storage devices, such as tape drives. The initiator is naturally the device which discovers the available targets, initiates the login procedure, and dictates where information should be stored and retrieved.



F_Port	The switch Link Control Facility that attaches to an N_Port through a link.
FL_Port	An F_Port which is capable of functioning on an Arbitrated Loop.
Fx_Port	Port capable of behaving as an F_Port or an FL_Port.
N_Port	The end device Link Control Facility that attaches to an F_Port or an N_Port through a link.
NL_Port	An N_Port which is capable of functioning on an Arbitrated Loop.
Nx_Port	Port capable of behaving as an N_Port or an NL_Port.

Figure 8: Fibre Channel Topologies

Fibre Channel end devices are generally arranged in one of the three previously mentioned topologies: fabric, loop, or point-to-point (Figure 8). Two end devices may communicate using a point-to-point link. Fibre Channel end ports which are configured to operate in link mode are known as N_Ports. Here, no real addressing scheme is needed, since the frames a given port receives must all be originated by the only other port to which it is attached. Two or more end devices may also be attached as N_Ports on links to a switch via the switch ports, or F_Ports, in what is known as a switched fabric topology. Here, the switch assigns the addresses used to direct the frames to their corresponding devices. Finally, up to 126 actively participating end devices, or NL_Ports, may be attached using the Arbitrated Loop topology. One switch, or FL_Port, may also be present on an Arbitrated Loop, but this is not required. In the Arbitrated Loop, each device assigns itself an address during the initialization process. These addresses are used not only in the frames, but also in many Primitive Signals and Primitive Sequences. These and additional port categories are listed in Appendix E [1].

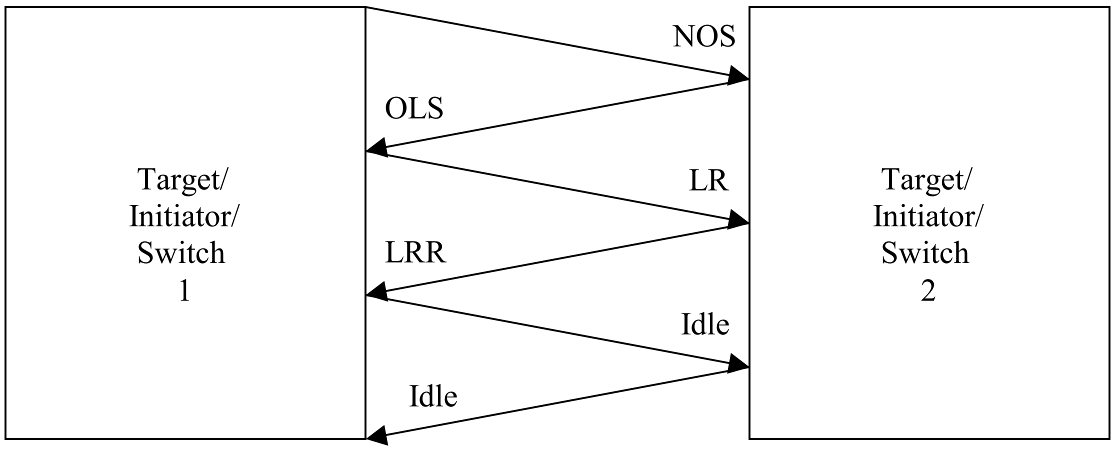


Figure 9: Link Initialization

2.4 Link Initialization

The topology which exists among Fibre Channel devices is determined during initialization. End devices and switches are designed to transmit certain signals over the

Fibre Channel bus immediately after they have been powered on in order to indicate the type of initialization they support. If more than a given device supports one type of initialization, the device will transmit the signals associated with one default initialization mode, and proceed to transmit those which are associated with another only if the first initialization attempt fails. The first type of initialization, Link Initialization (Figure 9), takes place between end devices in a point-to-point topology, and also by fabric ports which are attached to other ports not operating on a loop. During Link Initialization, one port transmits the NOS Primitive Sequence until it receives the OLS Primitive Sequence from the other port, at which time the first port transmits the LR Primitive Sequence until it receives the LRR Primitive Sequence from the other port. At this time, the first port transmits Idle, and upon recognizing this, the second port does so as well.

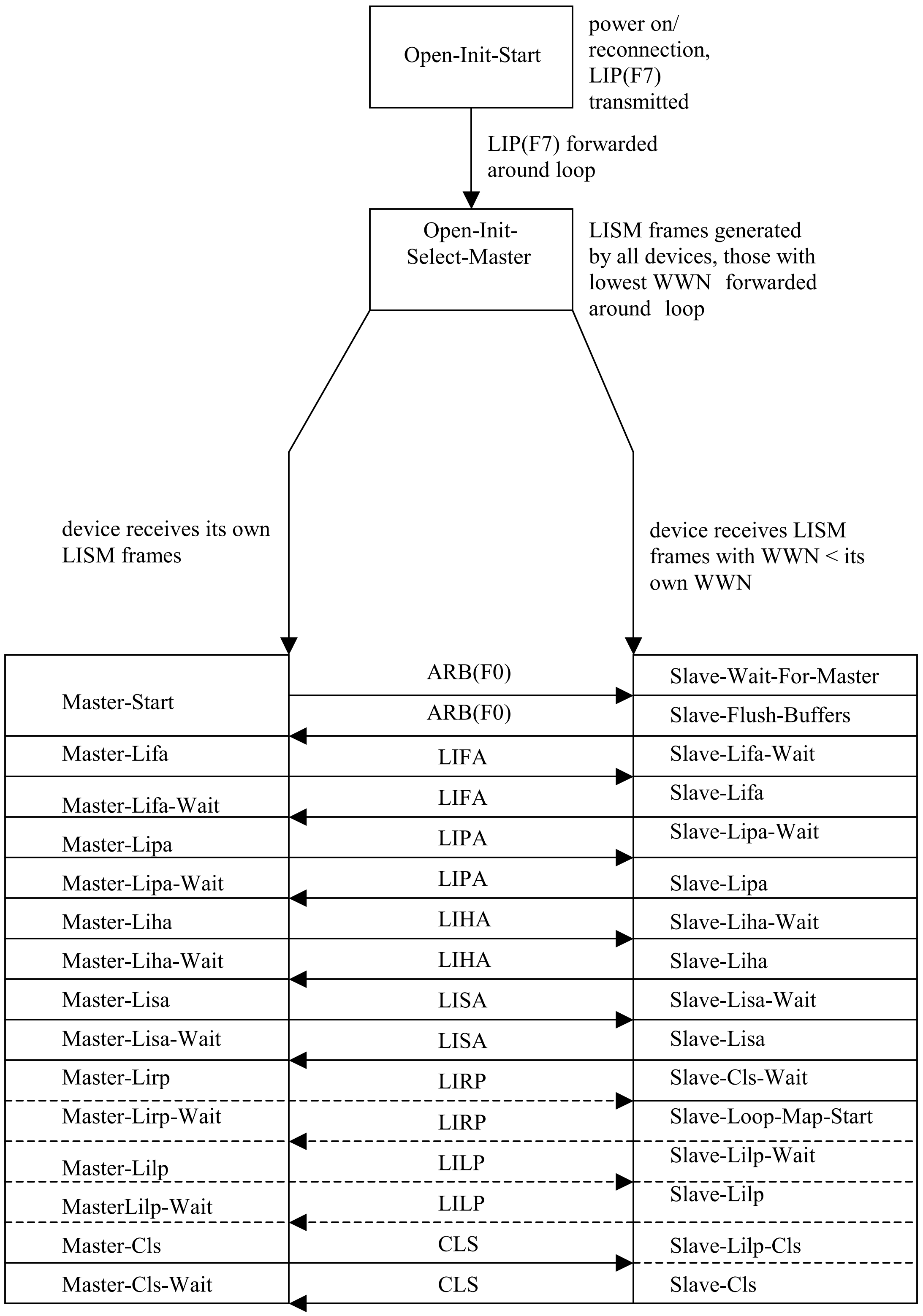


Figure 10: Loop Initialization

------ set of optional states

2.5 <u>Loop Initialization</u>

Unlike Link Initialization, Loop Initialization can take place among two or more devices which are physically connected in a ring-like manner. This process is governed by the device which becomes the Loop Master, or frame generator, until its completion (Figure 10) [2]. Loop Initialization takes place only as an alternative to Link Initialization; if Link Initialization completes, Loop Initialization does not take place, and vice versa. Additionally, the Master-Slave relationships which exist during initialization do not necessarily correlate with the initiator-target relationships which do not emerge until login. For this reason, it is possible for any type of Fibre Channel device to become Loop Master.

Loop Initialization begins when one device transmits the Loop Initialization Primitive Sequence, or LIP, across the Fibre Channel bus. A typical time for this to occur would be during Power-On or reconnection. Upon the reception and recognition of LIP, each device on the loop is expected to transmit at least twelve copies of the LIPs it has received, and proceed into the Open-Init-Select-Master state. In this manner, even a previously established loop will be completely reinitialized. Next, every device capable of becoming a Loop Master transmits a LISM (Loop Initialization Select Master) frame, each with its own World Wide Name. The World Wide Name is a unique, previously assigned 64-bit IEEE registered extended name identifier. Any device in this state receiving a LISM frame with a lower value in the World Wide Name (WWN) payload section than the value being transmitted is required to recognize this, and forward the lowest value possible. Furthermore, any device which cannot generate its own LISM frames must forward the LISM frames it receives in the same manner. In this regard, the LISM frame with the lowest WWN value must be progressively forwarded around the loop, until it reaches its originator.

The originator of the forwarded LISM frames detects that it is now the Loop Master, and transmits the ARB(F0) Primitive Signal in order to signify the fact that it is therefore entering the Master-Start state. Once this signal is forwarded around the loop, the master generates the LIFA, LIPA, LIHA, LISA and possibly the optional LIRP and LILP frames if supported, setting the appropriate bit in accordance with its Arbitrated Loop Physical Address (AL_PA), and copying the bitmap from the previous frames into those which follow, so that no AL_PA may be selected twice. Each frame is subsequently forwarded around the loop, and all of the slave devices select the bits which correspond to their AL_PAs as well. The CLS Primitive Signal denotes the end of Loop Initialization, at which time each device can be specifically identified by its AL_PA.

If a switch is performing Loop Initialization, it will fill the S_ID and D_ID fields of all initialization frames with 0x00, beginning with the LISM frame. In this regard, it will automatically become Loop Master, regardless of its World Wide Name. If another switch is on the loop, not only will the switch with the lowest value in the World Wide Name payload section become Loop Master, but the other switch will be required to discontinue its participation in this particular loop. Moreover, the switch that remains on the loop will also select the reserved value of 0x00 as its ALPA in all of the subsequent initialization frames, and the other devices will become aware of its presence.

Bits / Word	31 .. 24	23 .. 16	15 .. 08	07 .. 00	
	FC SOF				
0	R_CTL		D_ID		FC Header
1	CS_CTL		S_ID		FC Header
2	TYPE		F_CTL		FC Header
3	SEQ_ID	DF_CTL	SEQ_CNT		FC Header
4	OX_ID		RX_ID		FC Header
5	Parameter				FC Header
6	LS_Command code				FC Payload
7 8 9 10	Common Service Parameters (16 bytes)				FC Payload
11 12	Port_Name				FC Payload
13 14	Node_ or Fabric_Name				FC Payload
15 16 17 18	Class 1/6 Service Parameters (16 bytes)				FC Payload
19 20 21 22	Class 2 Service Parameters (16 bytes)				FC Payload
23 24 25 26	Class 3 Service Parameters (16 bytes)				FC Payload
27 28 29 30	Class 4 Service Parameters (16 bytes)				FC Payload
31 32 33 34	Vendor Version Level (16 bytes)				FC Payload
35 36	Services Availability (see note)				FC Payload
37	Login Extension Data Length (see note)				FC Payload
38 ..	Login Extension Data (see note)				FC Payload
70 71	Clock Synchronization QoS (see note)				FC Payload
72	FC CRC				
	FC EOF				

Note: These fields are only present when the Payload bit (word 8, bit 16) is set to one. When the Payload bit is set to zero, the Payload is 116 bytes long.

Figure 11: Port Login Frame (PLOGI)

2.6 Login

After initialization takes place, the login phase begins. However, devices will login in a slightly different manner, depending on their topologies. For instance, if Link Initialization has been performed between an initiator and a target, the initiator triggers the login process. Since the devices are residing on a link, a generic addressing scheme is used. Therefore, the initiator does not initially recognize whether it is attached to a target or a switch, and it transmits a Fabric Login frame. The Fabric Login Accept frame, which the target transmits in response to the Fabric Login frame, contains parameters which indicate the target is a N_Port. Subsequently, the initiator transmits a Port Login (Figure 11) [1], and the target replies with a Port Login Accept.

If the initiator-target pair has performed Loop Initialization, the initialization frames have indicated that the target is not a switch. Thus, the initiator is able to transmit a Port Login to the target without having transmitted a Fabric Login, and the target replies with a Port Login Accept.

If either a link or a loop topology is established involving an initiator and a switch, the initiator will transmit a Fabric Login frame, and the switch will reply with a Fabric Login Accept frame. Here, the Accept will contain information which tells the initiator that it is connected to a switched fabric. Additionally, the switch will assign a unique 3-byte N_Port_ID to the initiator and send it to the initiator in the D_ID field of the Accept. The N_Port_ID must be regarded by the initiator as its address from this point on. Nevertheless, the initiator must also transmit a Port Login frame. Likewise, the switch's F_Port or FL_Port must transmit a Port Login Accept to the initiator. This second login frame exchange must take place as different parameters are negotiated in each type of login frame. For instance, the Fabric Login frames indicate support for in-order delivery. Since in-order delivery is guaranteed between end devices, this field is not necessary in Port Login frames. Other fields such as N_Port end-to-end credit for each class of service being used, however, are clearly not necessary for Fabric Login, but are used during Port Login. If no initiator is present, and the link or loop exists between a target and a switch, the target initiates the login process in exactly the same manner as an initiator.

At this point, we have reviewed the basic building blocks which comprise the Fibre Channel layers residing beneath the FC-4 layer. We have defined these layers and their functions, as well as the processes which involve them, such as initialization and login. We are now ready to examine FCP, the ULP which is responsible for the transmission of SCSI over Fibre Channel, and iFCP, which allows the lower layers of Fibre Channel to either be replaced by or carried over TCP/IP. Once we have accomplished this, we will be prepared to see how iFCP transmits FCP, and therefore SCSI, over TCP/IP.

III iFCP

This chapter discusses the details of an iFCP session. First of all, an outline of the steps that should be taken in order to establish an iFCP session is discussed. Subsequently, each of these steps is examined more closely. The first step involves iFCP gateway discovery via iSNS. Then, the Session Control frames introduced by the iFCP specification are described, along with the iFCP encapsulation method. After the session execution has been depicted, the manner in which the iFCP session is expected to terminate is discussed. Finally, the designs of the iFCP initiator and target created as a result of this thesis are introduced.

3.1 iFCP Session

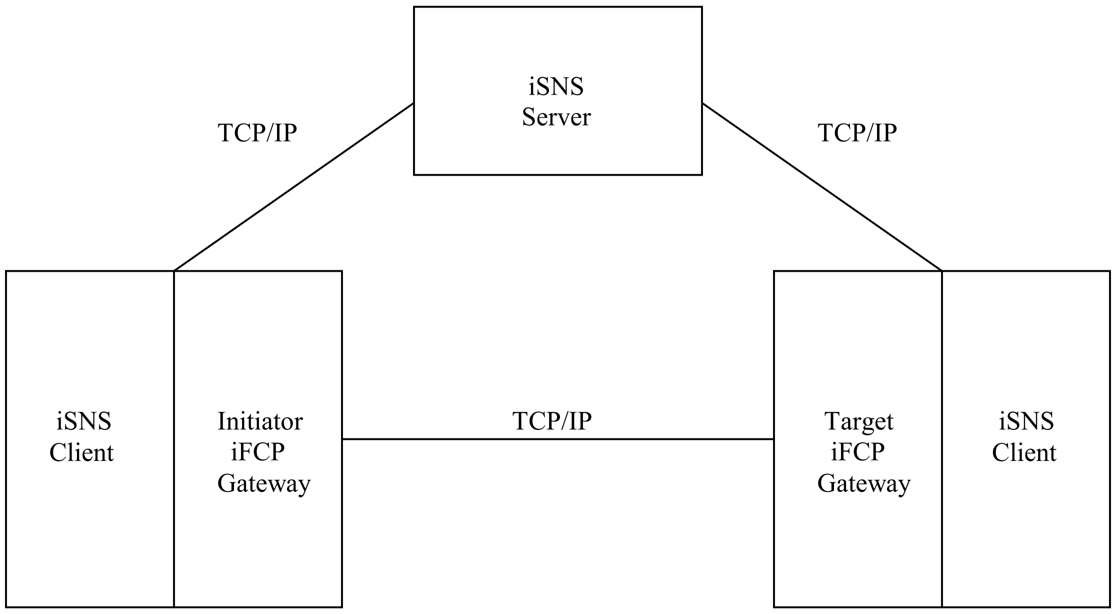


Figure 12: iFCP Topology Example

The iFCP protocol was designed in order to facilitate the transfer of I/O data between end devices, or gateways, over an IP network (Figure 12) [9]. The iFCP specification defines the basic means by which iFCP target and initiator gateways should:

1. register their identifiers and addresses with the discovery (iSNS) servers on the TCP/IP network, and query them for the identifiers and locations of other gateways
2. respond to iFCP frame exchanges when they are initiated by iFCP initiator gateways
3. negotiate the iFCP addressing mode and maintain addressing information which corresponds with local and remote (iFCP) devices
4. terminate any iFCP frame exchanges which must be terminated as defined by iFCP

Before an iFCP initiator establishes a connection with an iFCP target, the iFCP initiator must generally utilize the iSNS discovery protocol in order to discover which iFCP targets are located in its domain [10]. It must then connect to an iFCP target, and initiate the iFCP session. During the course of the session, both iFCP frames and encapsulated Fibre Channel frames are exchanged, and a common iFCP addressing mechanism must be utilized as negotiated during the initial iFCP handshake. The iFCP session is terminated in a predefined manner when the initiator has completed its task, or when an error has taken place as defined by the iFCP specification.

This protocol essentially expects iFCP end devices to behave in much the same manner as the Fibre Channel protocol, with one fundamental difference, since anything residing below the FC-3 layer is replaced by the combination of the iFCP encapsulation structure and the transport mechanisms of TCP/IP. In order to accommodate this change and spawn the iFCP session without the usual lower-level signaling protocol, a few new iFCP-specific frames have been introduced. Similarly, since the usual lower-level Fibre Channel addressing mechanisms are simply bypassed, the iFCP standard has created two slightly different addressing schemes in order to make sure that all devices are identified uniquely within any given domain.

3.1.1 iSNS

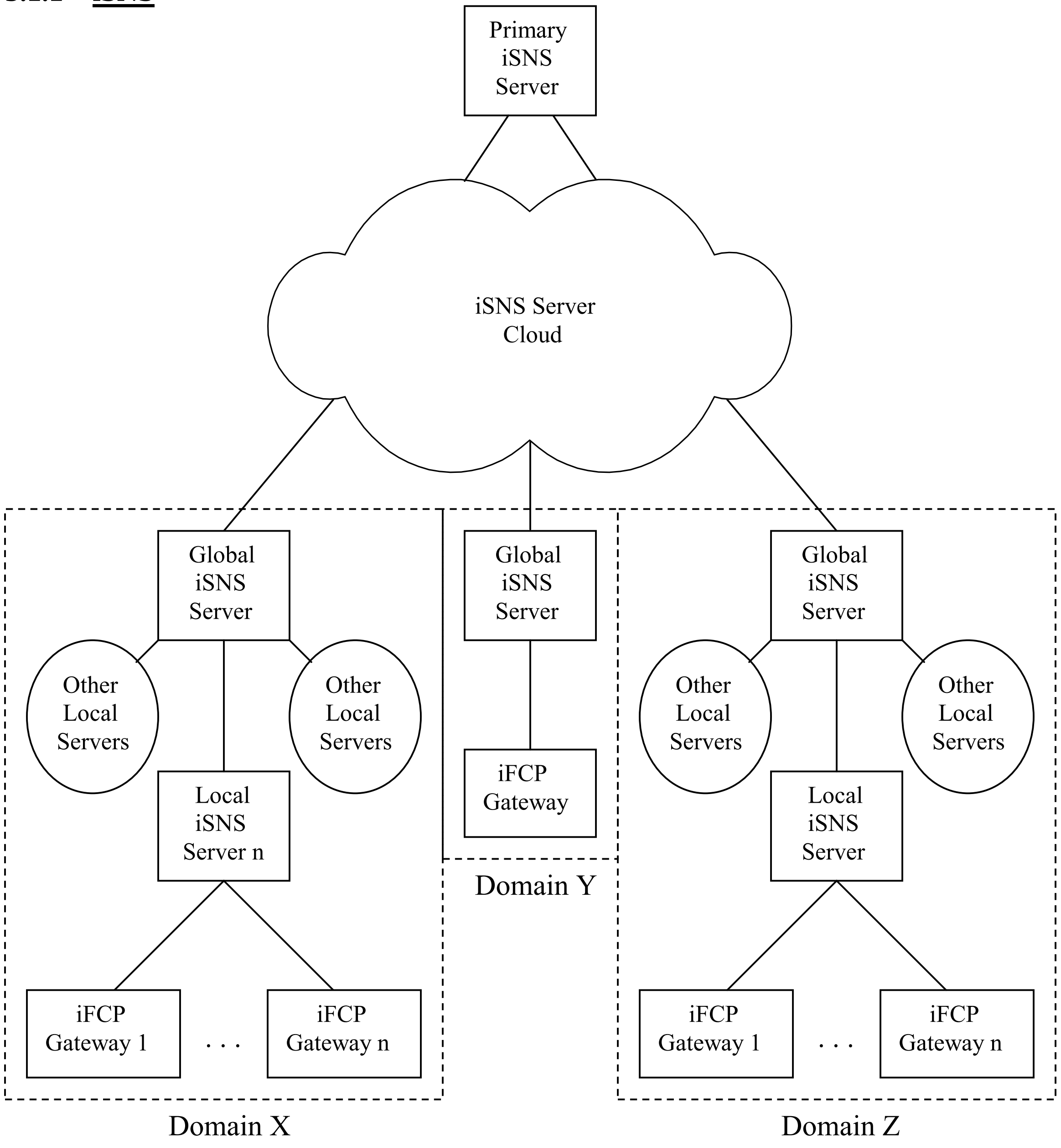


Figure 13: iSNS Topology Example

iSNS [10] is a protocol used to facilitate the discovery of other devices on the network, and it is a mandatory component of iFCP (Figure 13) [9]. In a given iSNS server hierarchy, a primary iSNS server must be implemented which retains a database containing addressing and domain assignment information for all other servers. A local or global iSNS server is required to be accessed by every iFCP gateway, and to register with the primary iSNS server as such. In order to do so, the local and global servers may have to implement the broadcast mechanism specified by iSNS in order to discover the IP address of the primary iSNS server. The servers may also utilize SLP, another discovery protocol, in order to find one another. The iSNS servers are contained within their respective domains, and one server is predefined as the global server for each discovery domain, while all others remain local. In this regard, the iSNS servers establish a hierarchy, which ensures that no domain assignments will overlap. Additionally, the iSNS servers are responsible for SNTP server location discovery and for the domain identifier assignment service provided by any iFCP gateway that simulates the behavior of an F_Port which operates in Address Transparent Mode (see Appendix F). Clearly, it is therefore crucial for the iSNS and iFCP components of a given implementation to exchange information whenever necessary.

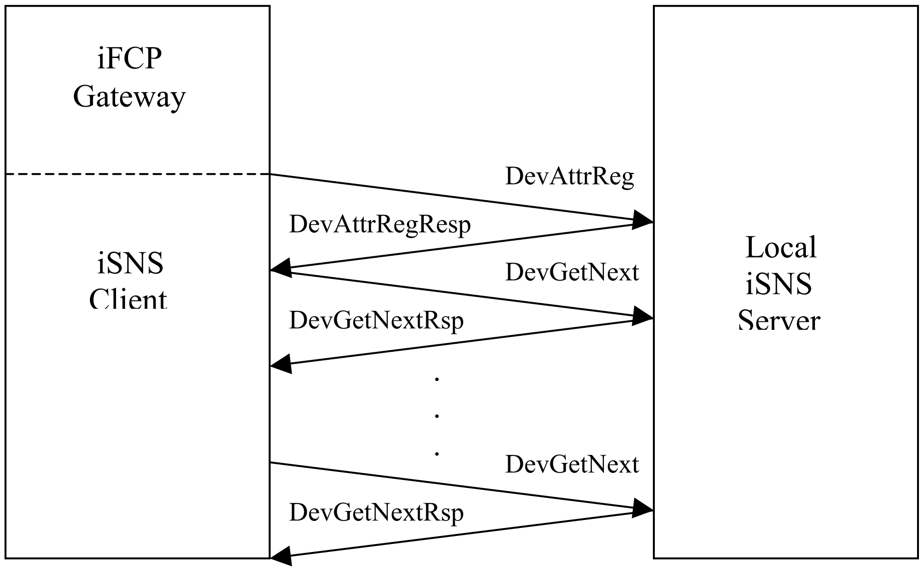


Figure 14: iSNS Exchange Example

iSNS is a naming protocol which requires iSNS clients to register with their respective iSNS servers as a certain type of node, and in this case the iFCP node type is used. One typical register and query exchange is exemplified in Figure 14. In the iSNS Device Attribute Register Request frame (DevAttrReg), the iSNS client provides the iSNS server with different keys that specify the iFCP attributes of the client, such as the World Wide Name, the Fibre Channel identifier, the port type (in accordance with Appendix E), and the IP address. Upon receiving a Device Attribute Register Response frame (DevAttrRegResp) from the server indicating success, the client may then query the server in order to retrieve information about other nodes in the domain to which it is assigned. One query method involves the Device Get Next Request frame (DevGetNext), modeled after a similar Fibre Channel frame. The client transmits the first DevGetNext frame with identification keys attached that have no values. The server, in turn, transmits a Device Get Next Response frame (DevGetNextRsp), with the desired values that pertain to a given iFCP device. In the next DevGetNext, the client retransmits the values it received in the last DevGetNextRsp, and the server retrieves the corresponding information for another device in the database of the server, and so on, until the end of the database is reached. The server indicates that all devices have been reported by transmitting a final DevGetNextRsp with the value "No Such Entry".

3.1.2 Addressing Modes

One of the addressing modes introduced by the iFCP protocol relies upon the iSNS server to assign globally unique identifiers to each device. This addressing mode is termed Address Transparent mode, and its usage is not mandatory (see Appendix F). Although the reason for this is not listed explicitly in the standard, one possible explanation for this is that an iFCP gateway which does not interface with Fibre Channel devices as an Fx_Port will not be able to convey to these devices the addressing assignments provided by the iSNS server.

The second, mandatory addressing mode is Address Translation mode. In this mode, each gateway assigns its own locally unique identifiers to devices that are attached both locally and remotely. In this regard, there is a risk that the identifiers assigned by different gateways could overlap. Therefore, every gateway engaging in an iFCP session with a remotely attached initiator or target maintains a database containing not only its own locally assigned identifier for the device, but also the remotely assigned identifier, identified by the context of the given TCP/IP connection. The specific identifier used for this purpose is deciphered based upon the value of the Fibre Channel S_ID or D_ID fields of the iFCP frame as it arrives via TCP/IP. There are three types of address translation which may occur. First, if the S_ID field is set to 0x000001, the iFCP gateway must replace it with the N_Port alias of the remote N_Port. Second, if the D_ID field is set to 0x000002, the iFCP gateway must replace it with the N_Port ID of the locally attached N_Port. Finally, if the D_ID field is set to 0x000003, the iFCP gateway must perform a search for the N_Port ID in accordance with the World Wide Name provided later in the same frame. If the Fibre Channel payload does not contain this data, the remote iFCP gateway must append it as supplemental information. If the N_Port ID fails for any reason, the frame is rejected. Due to the fact that in all three cases the recipient must not only replace the appropriate identifier, but also recalculate the Fibre Channel CRC, this addressing mode is less efficient than the first. Examples of both iFCP addressing modes are provided in Appendix G.

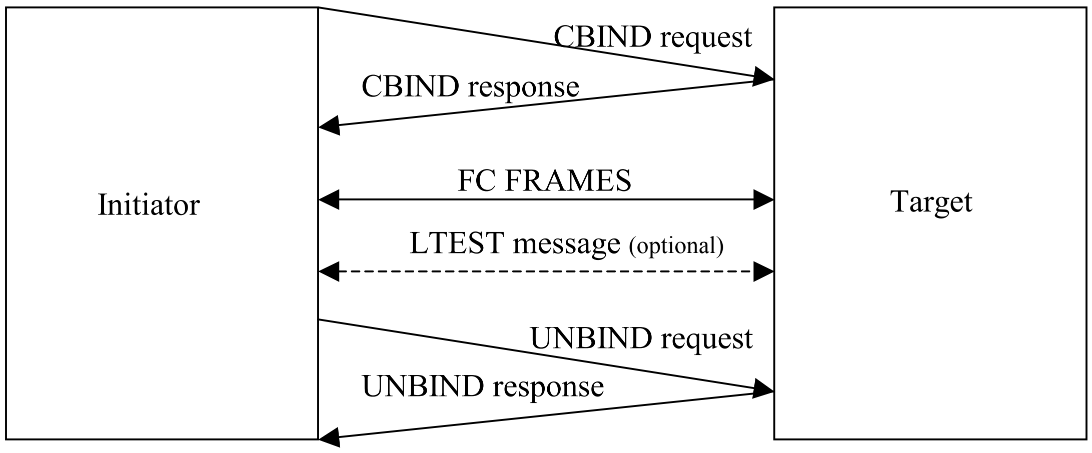


Figure 15: iFCP Session Example

3.1.3 <u>New iFCP Frames</u>

In order to establish and maintain an iFCP session, a few iFCP-specific frames are used, known as Session Control frames (Figure 15). These consist of: 1) an initial frame called the CBIND request which is transmitted in order to establish the iFCP connection, 2) the CBIND response, 3) the UNBIND request frame, which terminates the connection, 4) the response to the UNBIND request, called the UNBIND response, 5) the LTEST message, an optional frame which may be transmitted at various intervals in either direction, in order to verify the fact that the connection remains alive.

The establishment of an iFCP session could take place directly after a remote iFCP target reports the N_Port ID of a locally attached Fibre Channel target to an iSNS server within the discovery domain. The initiator could subsequently perform an iSNS query either on its own behalf, or as a result of receiving an iSNS State Change Notification frame (SCN) from the iSNS server, which is very similar to the Fibre Channel Registered State Change Notification frame (RSCN) discussed in Appendix I [1]. If the iFCP target behaves as an F_Port, the iFCP initiator can also discover the fact that a locally attached Fibre Channel target has registered via the Fibre Channel mechanisms discussed further in section 3.5.

The iFCP initiator attempts to establish a session by generating a CBIND request frame. If the iFCP target transmits a CBIND response with a successful status code, the session is established. During the iFCP session, the iFCP target may periodically generate LTEST messages, if this has been requested in the CBIND request, or by the iFCP initiator, if this has been requested in the CBIND response. Finally, the UNBIND request may be transmitted by the iFCP initiator or target, in order to terminate the session gracefully.

Some features of the iFCP protocol are as simple as possible. For instance, only one iFCP session may exist on any port at any given time, and multiple sessions and connections are not utilized. Also, since loops do not exist within this protocol, and some port types may not be present, many Fibre Channel specifications become irrelevant. In fact, only a small number of ELS frames are mandatory within the iFCP protocol, although many of the other common Fibre Channel frames remain important.

3.2 iFCP Session Establishment

When its global iSNS server notifies an iFCP initiator of the presence of any iFCP targets, it creates an internal Remote N_Port Descriptor for each of them, if the gateway is operating in Address Translation mode. This data structure contains the remote N_Port World Wide Name, the remote IP Address, and the remote N_Port ID. The descriptor will also contain the remote N_Port Alias, which is the Fibre Channel address assigned to the remote device by the target gateway. This data structure is not necessary in Address Transparent mode, since here the iSNS server has already assigned a unique identifier to all of the Fibre Channel devices in the global domain. The initiator gateway then proceeds to utilize the information contained in each descriptor in order to establish connections with the targets and to transmit CBIND requests to them. Likewise, when an iFCP target gateway receives an incoming TCP/IP connection from an iFCP initiator gateway followed by a CBIND request, that target also creates a Remote N_Port Descriptor.

An iFCP session has two states: the OPEN state, and the OPEN PENDING state. In the OPEN PENDING state, the iFCP initiator has issued a CBIND request, but no response has been received. The initiator may not transmit LTEST Messages or encapsulated Fibre Channel frames at this time. In the OPEN state, encapsulated Fibre Channel frames may be both sent and received. This state is entered by an iFCP target when it issues a CBIND response with a positive status, and by the initiator upon receiving the positive response.

A positive CBIND response contains a status of either “Success” or “iFCP Session Already Exists”. Other CBIND status values exist which indicate that a failure has occurred. Failures will occur for various reasons, some of which have specific values. For instance, specific values are defined for fields in the CBIND request which are formatted incorrectly. A lack of synchronization will also produce a negative response, as well as a CBIND request which carries a value in the addressing mode field that is not supported by the target. At this point, this would have to be the value which corresponds with Address Transparent mode. If a CBIND response is received with a status not mentioned in the previous sentence, the initiator terminates the session.

Once the session is in the OPEN state, both the initiator and target iFCP gateways create a Session Descriptor, if a duplicate Session Descriptor does not already exist. The

Session Descriptor is comprised of the TCP connection context (the information necessary to identify this TCP connection), the N_Port ID of the local N_Port, and the N_Port ID assigned to the remote device by the remote iFCP gateway. The remote N_Port Alias is copied to the Session Descriptor from the Remote Descriptor, if it exists.

During the iFCP session, encapsulated Fibre Channel frames are transmitted, in either Address Translation mode or Address Transparent mode, along with any optionally requested LTEST messages. At the end of the iFCP session, either the initiator or the target transmits an UNBIND request, and the other replies with an UNBIND response.

Word	Byte 0	Byte 1	Byte 2	Byte 3
0	Protocol#	Version	-Protocol#	-Version
1	Reserved			
2	LS_ACC	iFCP flags	SOF	EOF
3	Flags	Length	-Flags	-Length
4	Time Stamp [integer]			
5	Time Stamp [fraction]			
6	Header CRC			

Figure 16: iFCP Encapsulation Header

3.3 iFCP frames

As previously stated, five new iFCP frames were introduced in order to provide iFCP gateways with a handshake mechanism to establish, maintain, and close connections. An encapsulation specification is used both for the iFCP-specific frames and the frames used by iFCP specified by the Fibre Channel protocol (Figure 16). The encapsulation headers add various fields to the Fibre Channel frames which allow them to be identified appropriately by the iFCP gateway [9][11].

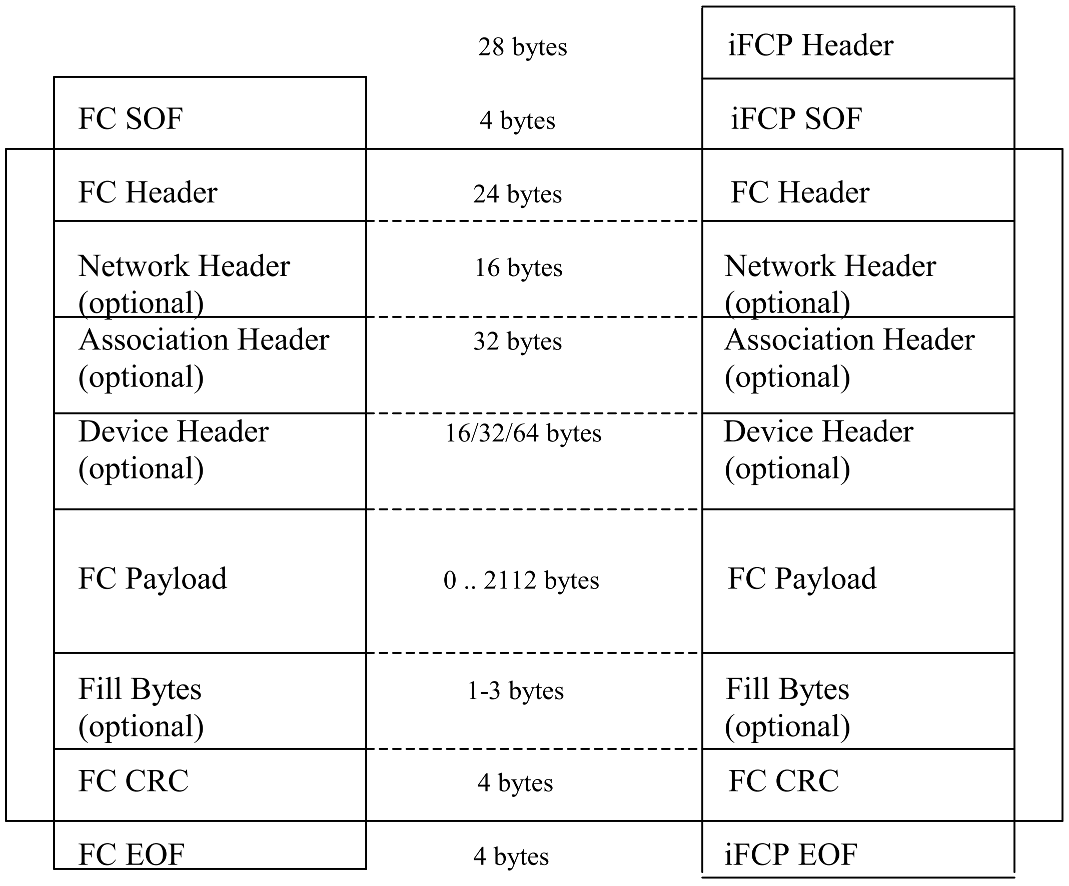


Figure 17: FC Frame vs. iFCP Frame

All iFCP frames begin with an iFCP encapsulation header (Figure 17). The header consists of: the iFCP protocol number and the encapsulation version number, along with their respective one's complements, the LS_COMMAND_ACC field, an iFCP flags field, an SOF and an EOF field, a flags field and a frames length field, followed by the respective ones complements, an integer time stamp, a fraction time stamp, and a CRC.

If the frame being encapsulated by iFCP is a special link service ACC to be processed by iFCP (Appendix H), specific fields must be processed as dictated by the iFCP specification. For example, if a Logout Accept (LOGO_ACC) frame is transmitted, the N_Port ID of the N_Port which is logging out is contained within the payload of the Fibre Channel frame. It is mandatory for the iFCP recipient to process this information, since it may be necessary for the recipient to transmit an UNBIND frame to this N_Port. In order to notify the iFCP recipient that it must process the payload of such a frame, the iFCP sender marks the LS_COMMAND_ACC field of the iFCP header with bits 0 through 7 of the Command Code for the frame to which this ACC frame is responding, in this case the Logout (LOGO). Otherwise, the LS_COMMAND_ACC field is 0.

<u>iFCP Flags</u>: Bit 23 22 21 20 19 18 17 16

23 22 21 20 19	18	17	16
Reserved	SES	TRP	SPC

SES	1	Session control frame (TRP and SPC MUST be 0)
TRP	1	Address transparent mode enabled
	0	Address translation mode enabled
SPC	1	Frame is part of a link service message requiring special processing by iFCP prior to forwarding to the destination N_PORT

<u>Figure 18: iFCP Flags Field</u>

The iFCP flags field (Figure 18) [9] indicates whether or not one of the previously mentioned Session Control frames is being transmitted, and which addressing mode is being used. The SOF and EOF fields contain 0x2E and 0x42 for Session Control frames, which correspond to the SOF and EOF codes for SOFi3 and EOFt, respectively. Similarly, other values may be present which are assigned to these fields in accordance with many of the other SOF and EOF ordered sets which are part of the Fibre Channel frames. The flags field only contains the CRCV flag, which must always be set to one to indicate that the header CRC is valid, since the header CRC is always required in iFCP.

The frame length field contains the length in 32 bit words of the entire encapsulated frame, including the SOF and EOF words. The integer time stamp contains either 0 or the number of seconds since the 0 hour on January 1, 1900 at the time the frame is placed in the stream of outgoing data. The fraction time stamp contains the fraction of the second at the time the frame is placed in the outgoing data stream. The CRC field contains the mandatory 32 bit Cyclical Redundancy Check for the header. This iFCP header CRC is calculated solely with the iFCP header, and is in addition to the FC CRC, which is calculated with the remaining portion of the frame. The CRCs are separate, since the FC CRC will be the only one remaining in place after deencapsulation. Both CRCs use the CRC-32 algorithm and the generator polynomial 0x104C11DB7. Although it is a positive factor in the maintenance of data integrity, the software calculations which the iFCP header CRC necessitates may decrease the efficiency of the implementation.

FC SOF	iFCP Code	Class	FC SOF	iFCP Code	Class
SOFf	0x28	F	SOFi4	0x29	4
SOFi2	0x2D	2	SOFn4	0x31	4
SOFn2	0x35	2	SOFc4	0x39	4
SOFi3	0x2E	3	SOFn3	0x36	3

Figure 19: SOF Encapsulation Codes

FC EOF	iFCP Code	Class	FC EOF	iFCP Code	Class
EOFn	0x41	2,3,4,F	EOFdt	0x46	4
EOFt	0x42	2,3,4,F	EOFdti	0x4E	4
EOFni	0x49	2,3,4,F	EOFrt	0x44	4
EOFa	0x50	2,3,4,F	EOFrti	0x4F	4

Figure 20: EOF Encapsulation Codes

The payload of the iFCP frame contains the frame being encapsulated, with an SOF word at the beginning, and an EOF word at the end. The S_ID and D_ID of the frame are referenced prior to encapsulation. If no iFCP session descriptor is found to verify their integrity, the frame is discarded. The SOF word contains two bytes with the byte code for the appropriate SOF version as defined by iFCP (Figure 19) [9], followed by two bytes with the one's complement version of those bytes. The EOF word contains two bytes with the byte code for the appropriate EOF version as defined by iFCP (Figure 20) [9], followed by two bytes with the one's complement version of these bytes. If the frame being encapsulated is an iFCP frame, its payload is laid out using a Fibre Channel frame template with a specific SOF and EOF byte code. It is then encapsulated in the same manner as a Fibre Channel frame.

If the gateway is operating in Address Translation mode, it must replace either the S_ID or the D_ID of the Fibre Channel frame as discussed in section 3.1.2. Also, if the frame contains a special link service message payload, it may be adjusted by a gateway operating in Address Translation mode. The N_Port address field may be altered, and/or supplemental data may be added to the end of the frame. In either case, the gateway must recalculate the FC CRC prior to encapsulation.

By the same token, the recipient gateway must check the Header CRC and the iFCP flags field. If either field is incorrect, the frame is discarded. The gateway must then check the FC CRC as well, and discard the frame if it is invalid.

Word	Byte 0	Byte 1	Byte 2	Byte 3
0	0xE0	0x00	0x00	0x00
1	LTI (Seconds)		Addr Mode	iFCP Ver
2	USER INFO			
3 4	SOURCE N_PORT NAME			
5 6	DESTINATION N_PORT NAME			

Figure 21: CBIND Request

The CBIND request (Figure 21) [9], which is transmitted to the iFCP target by the iFCP initiator in order to establish an iFCP session, associates a TCP connection with an N_Port. This frame contains several pieces of information: the source and destination N_Port names, an optional four byte user information field, the addressing mode which is desired, the iFCP version number, and the Liveness Test Interval for the iFCP connection. The N_Port Names are provided in the N_Port Descriptors. These are actually the 64-bit World Wide Names of the source and destination N_Ports. The user information field contains any data which should be echoed by the recipient in the CBIND response. The address mode field is set to 0 for Address Translation mode, and 1 for Address Transparent mode. If the Liveness Test Interval field is set to a positive value, the recipient will transmit the optional LTEST frame repeatedly across the iFCP connection at this interval, in seconds. If this field is 0, the LTEST frame must not be transmitted.

Word	Byte 0	Byte 1	Byte 2	Byte 3
0	0xE0	0x00	0x00	0x00
1	LTI (Seconds)		Addr Mode	iFCP Ver
2	USER INFO			
3 4	SOURCE N_PORT NAME			
5 6	DESTINATION N_PORT NAME			
7	Reserved		CBIND Status	
8	Reserved		CONN HANDLE	

Figure 22: CBIND Response

The CBIND response (Figure 22) [9], sent from the iFCP target to the iFCP initiator, is very similar to the CBIND request, with the addition of two non-reserved fields: the CBIND status field, and the connection handle. The status code is defined as 0 for "Success", and a number of nonzero status codes are defined for unsuccessful conditions. The connection handle is assigned by the target in order to identify the connection, and it is conveyed to the initiator so that it can be used to close the connection gracefully in the UNBIND frame. Additionally, the user information field must echo that which was received in the CBIND request.

Word	Byte 0	Byte 1	Byte 2	Byte 3
0	0xE5	0x00	0x00	0x00
1	LTI (Seconds)		Reserved	
2	COUNT			
3 4	SOURCE N_PORT NAME			
5 6	DESTINATION N_PORT NAME			

Figure 23: LTEST Message

The LTEST message (Figure 23) [9] consists of the Liveness Test Interval field, the count field, the Source N_Port Name, and the Destination N_Port Name. The Liveness Test Interval field contains a copy of the Liveness Test Interval which was transmitted in the CBIND request or CBIND response, accordingly, and specifies the interval at which the LTEST is transmitted. The count field contains 0 in the first LTEST message, and increments for each successive LTEST message thereafter. The source and destination N_Port IDs are the same as the ones utilized in the CBIND request. However, the order in which they are listed will be reversed if the LTEST message is transmitted from the target to the initiator.

Word	Byte 0	Byte 1	Byte 2	Byte 3
0	0xE4	0x00	0x00	0x00
1	USER INFO			
2	Reserved		CONN HANDLE	
3	Reserved			
4	Reserved			

Figure 24: UNBIND Request

The UNBIND request (Figure 24) [9] is transmitted in order to unbind a connection. It consists of a User Info field and a Connection Handle. The User Info contains optional data which must be echoed in the UNBIND response. The Connection Handle contains the gateway-assigned value from the CBIND request, which is used in order to identify the connection. Like the LTEST, the UNBIND request may be sent in either direction, if necessary.

Word	Byte 0	Byte 1	Byte 2	Byte 3
0	0xE4	0x00	0x00	0x00
1	USER INFO			
2	Reserved		CONN HANDLE	
3	Reserved			
4	Reserved			
5	Reserved		UNBIND Status	

Figure 25: UNBIND Response

The UNBIND response contains the echoed User Info and Connection Handle fields, as well as an Unbind Status field (Figure 25) [9]. An Unbind Status field of 0 indicates success, and 2 failure codes are also defined. The UNBIND response is generated by the recipient of a prior UNBIND request.

Event	IFCP Sessions to Terminate
PLOGI terminated with LS_RJT response	Peer N_PORT
State change notification indicating N_PORT removal or reconfiguration	All iFCP Sessions from the reconfigured N_PORT
LOGO ACC response from peer N_PORT	Peer N_PORT
ACC response to LOGO ELS sent to F_PORT server (D_ID = 0xFF-FF-FE) (fabric logout)	All iFCP sessions from the originating N_PORT
Implicit N_PORT LOGO as defined in [FC-FS]	All iFCP sessions from the N_PORT logged out
LTEST Message Error (see section 5.2.2.4)	Peer N_PORT
Non fatal encapsulation error as specified in section 5.3.3	Peer N_PORT
Failure of the TCP connection associated with the iFCP session	Peer N_PORT
Receipt of an UNBIND session control message	Peer N_PORT
Gateway enters the Unsynchronized state (see section 8.2.1)	All iFCP sessions
Gateway detects incorrect address mode to peer gateway(see section 4.6.2)	All iFCP sessions with peer gateway

Figure 26: iFCP Session Termination Events

3.4 iFCP Session termination methods

Figure 26, borrowed from the iFCP specification, lists the events which must cause the iFCP session to be terminated [9]. If an iFCP session is being terminated due to incorrect address mode, the TCP connection is aborted without an UNBIND. If an iFCP session is being terminated for any other reason, encapsulated Fibre Channel frames must no longer be sent over the TCP connection, and all incoming frames besides UNBIND messages are discarded. If an UNBIND message is received at any time, an UNBIND response is returned. If an UNBIND which is transmitted contains a status which does not indicate success, the TCP connection should be aborted. Otherwise, the TCP connection can remain open, and kept in a pool of unbound TCP connections. Since only one connection is allowed per session, this will not enable multiple connections to be established, but it should speed up the creation of a new iFCP session.

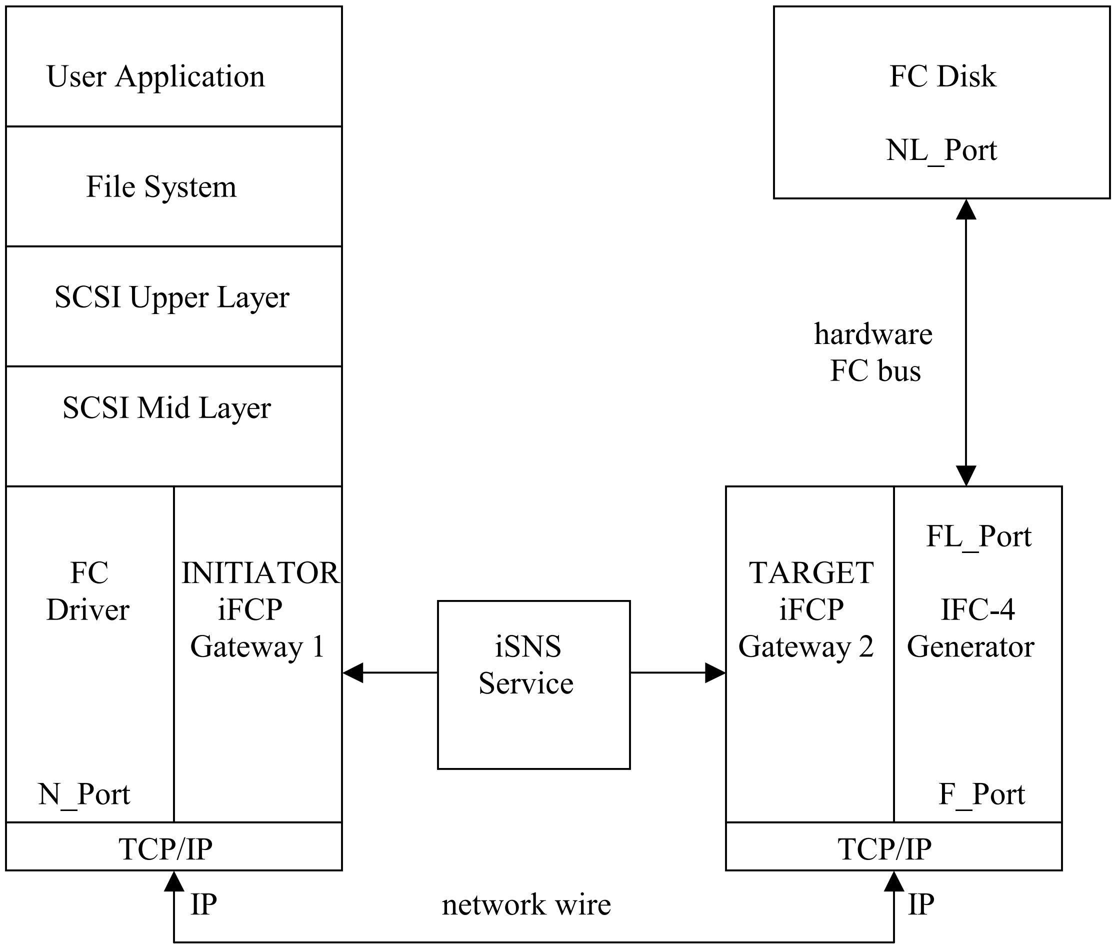


Figure 27: iFCP Initiator and Target Implementations

3.5 iFCP Initiator and Target Design

Figure 27 depicts the iFCP target and initiator implementation design. The iFCP target is designed to interface with the iFCP initiator as an F_Port, and to interface with its locally attached Fibre Channel device as an FL_Port.

There are several reasons for this design decision. First of all, the fact that the iFCP target presents itself to both the locally attached Fibre Channel device and the remotely attached iFCP implementation as an Fx_Port causes both Nx_Ports to transmit registration and/or query frames to it. As a result, when it delivers any query response frames to the remotely attached initiator, the iFCP target presents the locally attached Fibre Channel disk to the initiator as a logical Fibre Channel device. If a locally attached Fibre Channel device were to query the iFCP target, the iFCP target would report the presence of the remotely attached initiator as a logical Fibre Channel device, as well. The query responses which are issued from the iFCP target to the remotely attached initiator cause the initiator be informed of the Fibre Channel disk's presence, and to establish a session with the disk.

In order to comply with this specification on the target side, the iFCP target implementation must enable exchanges to take place between a storage device and the iFCP initiator that is connected to it via TCP/IP. However, it is undesirable for the target implementation to allow its storage device, a Fibre Channel disk, to timeout during any delays which might take place between the iFCP gateways. Implementing the iFCP target as a if it were a switch allows it to present itself to the FC disk as a separate entity from the iFCP initiator, while also allowing it to virtually connect the initiator with the target. Due to the fact that both the Fibre Channel disk and the iFCP initiator register with the iFCP target independently of their communication with one another, any failure or congestion which takes place either on the Fibre Channel link or the TCP/IP link does not cause the other link to go down, and link recovery is that much more efficient.

Bit: 23 16	15 8	7 0
Domain ID	Area ID	Port ID

Figure 28: N_Port Identifier (N_Port ID) Format

Finally, the addressing mechanism which can be deployed by the iFCP target is much more powerful, due to the FL_Port interface. The iFCP target can assign the domain identifier provided by the iSNS server directly to the Fibre Channel disk, along with the area and port identifier portions of the N_Port ID (Figure 28) [1][9]. Although a slightly higher initial overhead is required in order to perform login and registration with the switch, this allows it to easily utilize the address transparent mode of iFCP, and to thus avoid recalculating the Fibre Channel CRC of incoming frames, increasing the efficiency of the implementation whenever possible.

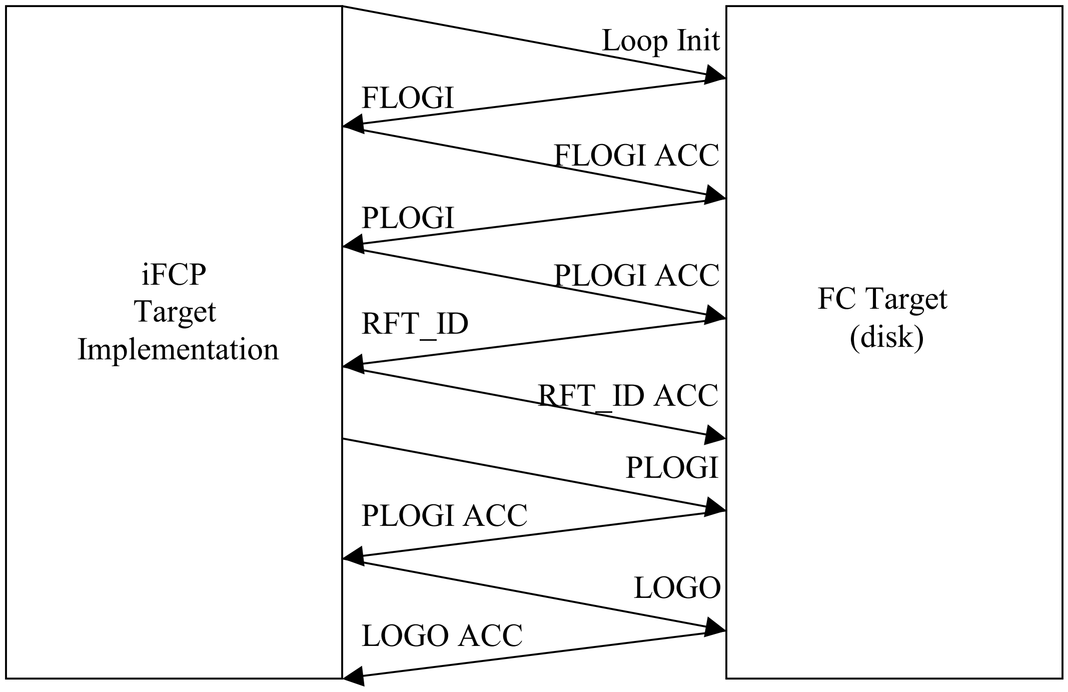


Figure 29: iFCP Target Implementation (switch) and FC Target Exchanges

Due to the fact that most Fibre Channel disks operate in loop mode, the iFCP target implementation interfaces with the Fibre Channel disk as the FL_Port of a switch, and performs Loop Initialization prior to logging in with the disk. The full sequence of exchanges that takes place during this login process is shown in Figure 29. It is important to note that this is only one example, and that this frame exchange may differ slightly when different Fibre Channel disks are used. During the login process, the target's FL_Port port receives a Fabric Login frame (FLOGI) from the Fibre Channel disk, and subsequently assigns the disk a domain identifier, which it sends in the Fabric Login Accept (FLOGI ACC) frame. The Port Login exchange then takes place, whereby the disk transmits a Port Login (PLOGI) frame, and the FL_Port of the target transmits a Port Login Accept (PLOGI ACC) frame.

The FL_Port of the iFCP target also provides Fibre Channel registration and query capabilities to the disk [4]. Thus, in this particular frame exchange, the disk in Figure 29 transmits the Register FC-4 Descriptor Request (RFT_ID), in order to register its port name with the switch as an FC-4 layer device, and indicate the ULP that it supports (a code is used to indicate SCSI FCP). The FL_Port recognizes this frame, parses it, and transmits a Register FC-4 Descriptor Accept (RFT_ID_ACC). This response indicates that any initiators which query the iFCP target implementation will be provided with the address of the disk. This particular disk does not transmit a State Change Registration frame, or any other generic service frames, so the FL_Port proceeds to transmit a PLOGI frame, to which it receives a response. Since no I/O will take place directly between the FL_Port and the NL_Port, the FL_Port immediately terminates this particular frame exchange with a Logout (LOGO) frame, and the disk completes the logout procedure by transmitting a Logout Accept (LOGO_ACC) in return.

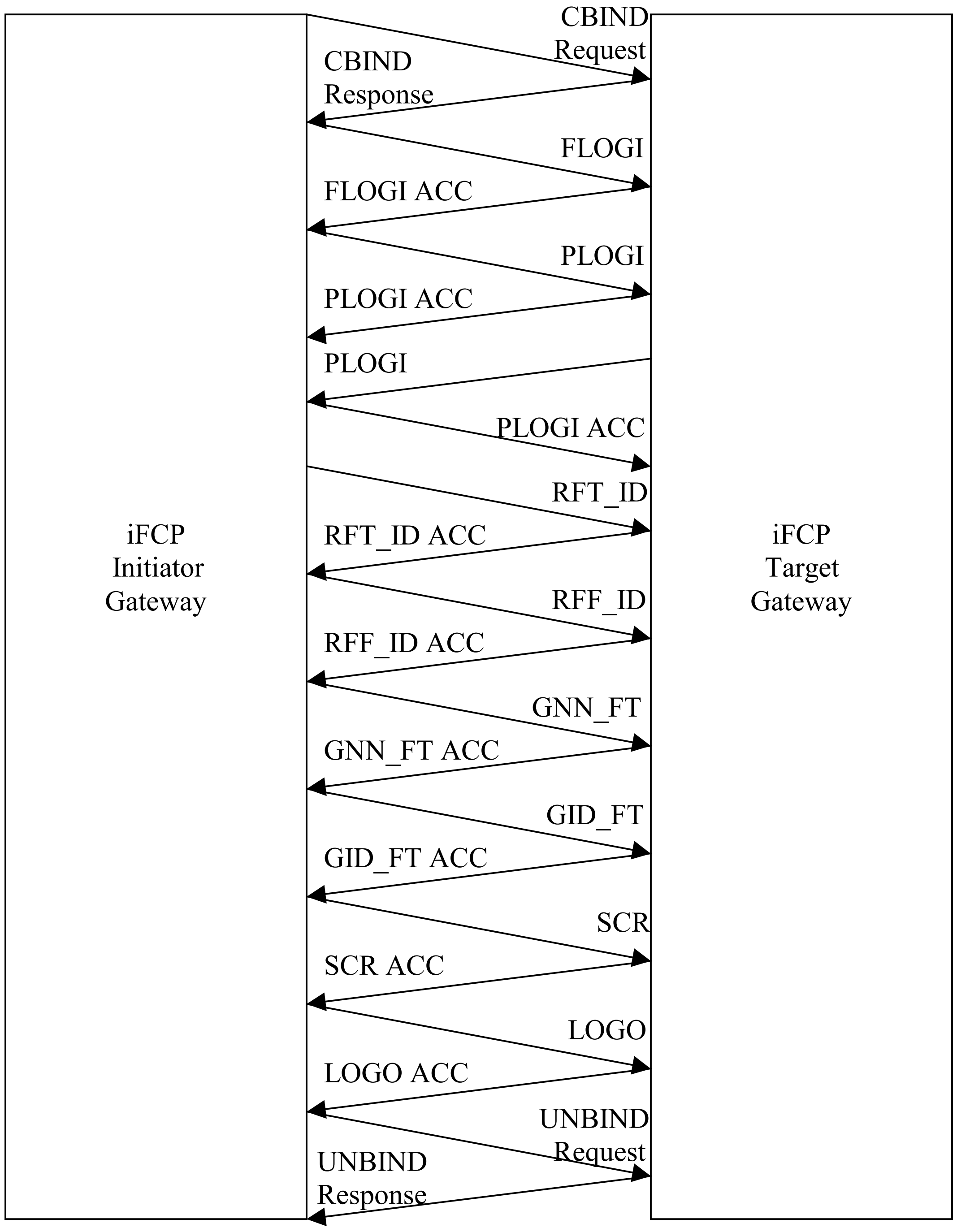


Figure 30: Target and Initiator iFCP Gateway Exchanges

In a similar manner to the Fibre Channel iFCP target interface, the iFCP target interface on the TCP/IP side transmits frames resembling those of an F_Port, and the iFCP initiator communicating with it has the option of transmitting registration frames as well (Figure 30). Thus, the interaction is very much the same as that exemplified in Figure 29, except for the fact that it takes place over TCP/IP instead of Fibre Channel, the Loop Initialization is replaced by a CBIND Request from the iFCP initiator to the iFCP target, followed by a CBIND Response from the iFCP target to the iFCP initiator, and all of the FC frames are encapsulated. Additionally, the initiator transmits a Register FC-4 Features frame (RFF_ID), a Get Node Names query frame (GNN_FT), a Get Port Identifiers query frame (GID_FT), and a State Change Registration (SCR) frame. These are all replied to with the appropriate Accept frames. Furthermore, the World Wide Name of the Fibre Channel disk is reported in the Get Node Names Accept (GNN_FT ACC) frame, and the N_Port ID of the disk is reported in the Get Port Identifiers Accept (GID_FT ACC) frame. In this manner, the iFCP target reports any Fibre Channel devices or implementations which have registered with it to any others which transmit queries for information concerning their particular type, just as a Fibre Channel switch would. Typically, a SCSI initiator will perform a query for devices that have registered with a given switch as SCSI targets. Additionally, since the initiator has performed State Change Registration, any devices that either register or cancel their registrations with the switch implementation, both implicitly or explicitly, must be reported to the initiator by the switch via a Registered State Change Notification (RSCN), so that the initiator can login with the device (see Appendix I) [1].

The iFCP initiator acts as an initiator end device, or N_Port. To accomplish this goal from a lower layer perspective, the scheme is quite simple compared with that of the target, since the initiator is self-contained, and its only external interface communicates with the target via TCP/IP. Therefore, the initiator is not implemented as a switch. Instead, the initiator implementation simply assigns itself a domain identifier and inserts this into the headers of its outgoing Fibre Channel frames before they are encapsulated. Likewise, the initiator must recognize the identifier portions of incoming encapsulated Fibre Channel frames as it receives them from the target, so that their originators may be recognized, and their destination verified. The manner in which these identifiers are assigned depends on whether the device is operating in Address Transparent mode or Address Translation mode. The implementation discussed here will utilize both addressing modes.

We have now reviewed the parameters which dictate the execution of an iFCP session. We have looked at the manner in which iFCP devices initially discover one another, and the frame structure which they utilize in order to communicate. Finally, we have discussed the iFCP initiator and target implementations from a somewhat abstract perspective. This brings us to the next chapter, in which we will look more closely at both the initiator and the target.

IV iFCP INITIATOR AND TARGET DESIGN AND IMPLEMENTATION

4.1 Overview

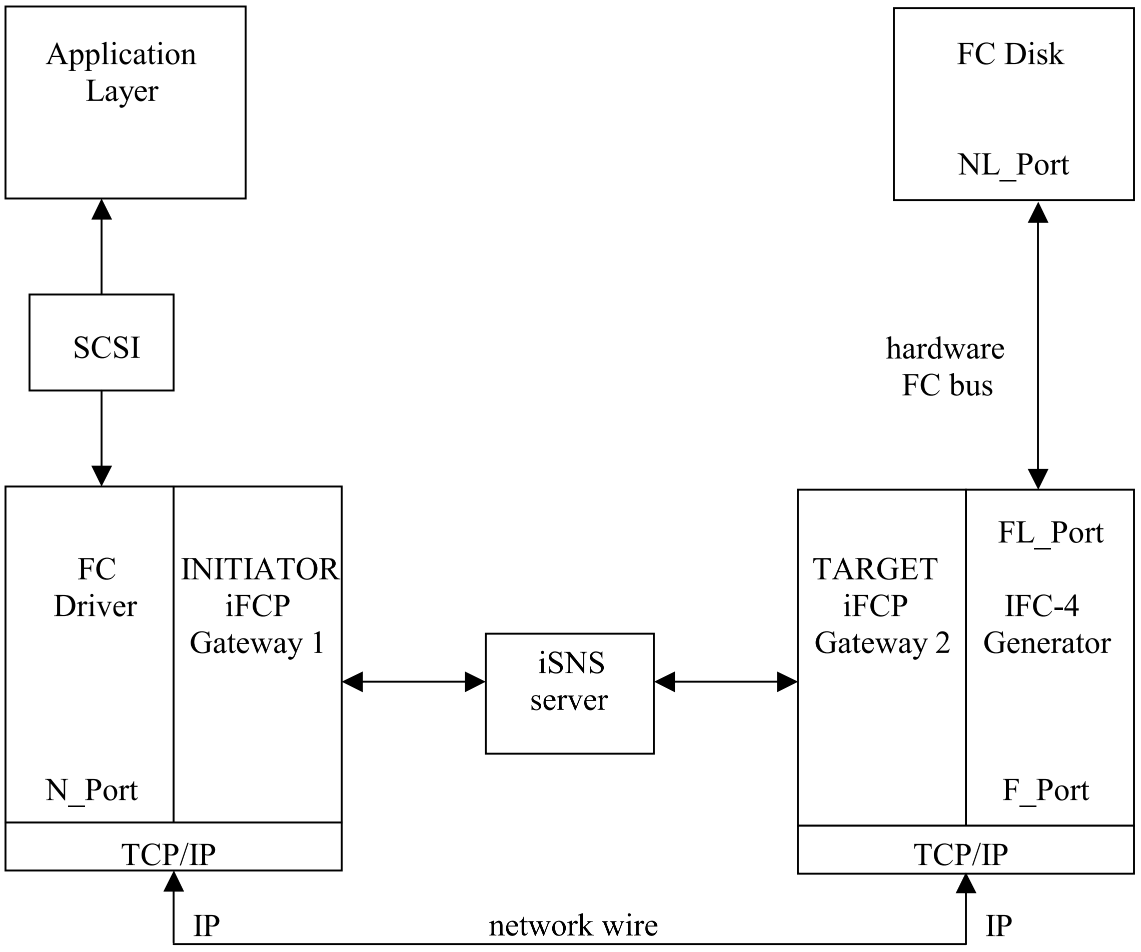


Figure 31: iFCP Initiator and Target Implementations

The iFCP target is a stand-alone user program written in C++ for a Windows platform. In order to communicate with the iFCP initiator over TCP/IP, the F_Port of the target uses Win sockets. In order to communicate with the Fibre Channel disk, the FL_Port of the target utilizes the I-TECH IFC-4 Fibre Channel generator card. The target interfaces with the generator using a Visual C++ workspace into which specific library files are linked. All of the code used to generate frames and retrieve information from incoming frames was created for the purposes of this project. Therefore, the Fibre Channel frames are all passed to the generator from user space, and vice versa, which means that the generator is used for Fibre Channel encoding and decoding purposes only, and none of its upper level functionality is utilized.

However, a generator-specific function call must be used in order to pass the frame buffer through the generator to the Fibre Channel bus, and another function call must be used in order to receive incoming Fibre Channel frames from the Fibre Channel bus (see Appendix J). Additionally, the generator produces the Fibre Channel Primitive Signals and Sequences, either automatically or through another generator-specific function call.

As discussed later in this chapter, the target code is arranged into various classes, in order to handle various tasks such as frame building, frame transmission and reception over TCP/IP, and frame transmission and reception over Fibre Channel. The functions in these classes mostly consist of both receiving and responding to various frames accordingly, while storing and retrieving data from the appropriate data structures whenever necessary. Additionally, the iFCP encapsulator must be capable of encapsulating and sending all frames over TCP/IP, as well as both receiving and de-encapsulating frames, while verifying their integrity by checking CRCs, timestamps, and the validity of other fields.

The initiator is implemented as a kernel module written in C, and it is Linux-based. The initiator code is arranged in a similar manner to that of the target, except the functions and data structures are not explicitly restricted to certain classes. The initiator establishes a socket connection with the target and both transmits and receives encapsulated iFCP and Fibre Channel frames over TCP/IP. Since it does not need to interface separately with a Fibre Channel device, the initiator does not contain any code to interface with a Fibre Channel generator. Instead, the initiator handles all of its own addressing capabilities, and additional initiator code exists which establishes the mechanisms necessary to interface with the SCSI Midlevel of the initiator host.

4.2 Initiator Data Structures

The primary data structure in the initiator design is the nport structure (see Appendix M). This structure contains the information necessary to carry the Fibre Channel driver portion of the initiator through all of the procedures it executes during a given iFCP session. The Fibre Channel identifiers for both the initiator and the target are stored here, along with other data, such as the exchange and sequence identifiers for both the last frame sent and the last frame received are stored here as well.

The nport_maker structure was designed as a container for the nport structure. It also carries a few pieces of information as they are supposed to be stored for various Fibre Channel payloads, as well as the outgoing frame buffer, and lists for target names and World Wide Names as they are reported in incoming generic queries. The nport_maker also contains fields which hold iFCP-specific data, such as an iFCP descriptors structure which holds both the Remote Descriptor and the Session Descriptor. Originally, this structure was also developed in order to become the component of a linked list if multiple sessions were created. However, when the Fibre Channel directory was merged with the SCSI directory, the session structure performed this task, and the nport_maker address is now simply referenced as part of the session structure.

The connection structure is basically a simplified version of the UNH IOL iSCSI initiator connection structure, containing data members which are necessary for the TCP/IP socket connection to be established and maintained. Additionally, this structure contains the address of the receive thread, along with its semaphore and frame buffer, to ensure the fact that FCP SCSI Data and Response frames are received and queued properly.

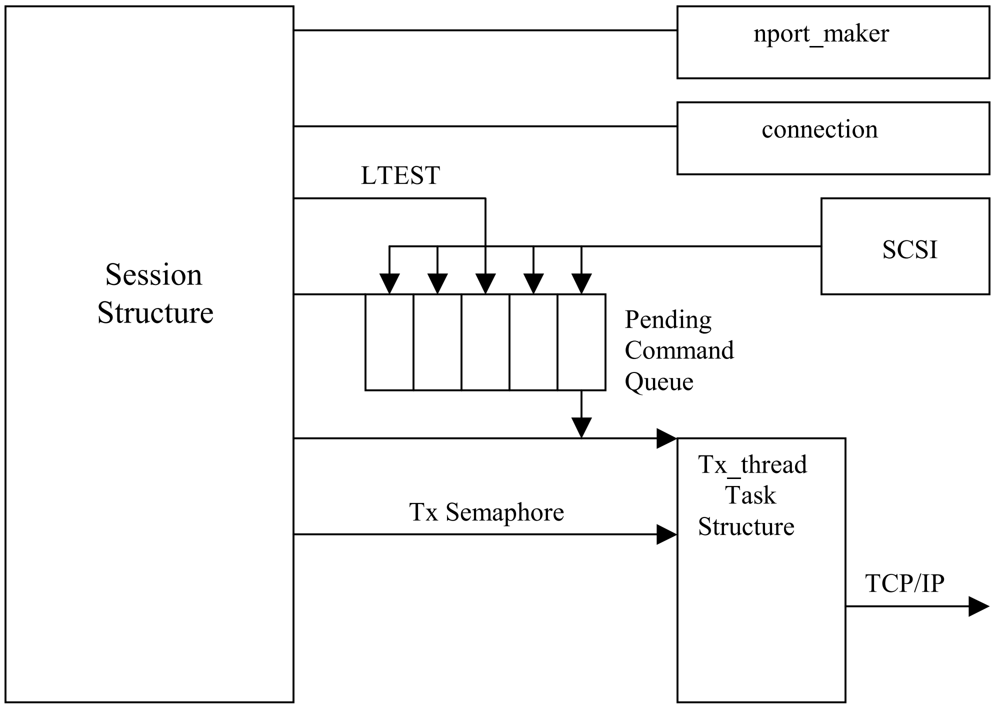


Figure 32: Initiator Session Structure

The session structure is a simplified and adapted version of a similar structure originated by the iSCSI consortium (Figure 32). This structure maintains the SCSI Command transmission queue, as well as the LTEST timer mechanism, and it contains the addresses of the nport_maker and connection structures. This structure also contains the address of the transmit thread, along with its semaphores. Also, as mentioned previously, if multiple sessions are initiated, this structure will be linked to additional session structures.

4.3 Initiator Design

The Fibre Channel driver portion of the initiator is extensive, yet relatively straightforward. A great deal of time was spent on this section of the code, but this was not due to the fact that the code itself needed to be produced in an overly complicated manner. Rather, the frames which should generally be available for a Fibre Channel driver are plentiful, and generating them properly and learning when to use them involved studying quite a few different specifications and testing them against many different Fibre Channel devices. Therefore, the most time-consuming section of the Fibre Channel driver is the frame generation portion.

Although the Fibre Channel driver is unique, along with the iFCP encapsulator, the SCSI Command queuing mechanism has been based upon a version of the UNH iSCSI initiator SCSI Midlevel interface that has been both stripped-down and modified to accomplish the actions performed by the iFCP initiator. The multiple connections that can exist in iSCSI do not exist in iFCP, so the multiple connection mechanism has been removed, and only one connection structure exists per session structure. Additionally, iFCP does not contain the relatively complicated set of negotiable parameters that iSCSI possesses, so these variables have been removed as well, and the login functions run using the procedures defined by iFCP and Fibre Channel, which are almost entirely different from those of iSCSI. Other mechanisms, such as error recovery, are completely erased and replaced, as they are also defined much differently in their respective specifications. Additionally, aside from the SCSI CDB and data portions of the frame payloads, the frame formats differ, to which great deal of frames have been added in the iFCP and Fibre Channel portion of the directory tree. As a result, some checks and searches are either completely eliminated or replaced by those which involve other types of identifiers. However, the basic SCSI queuing mechanism remains virtually the same, with very few changes.

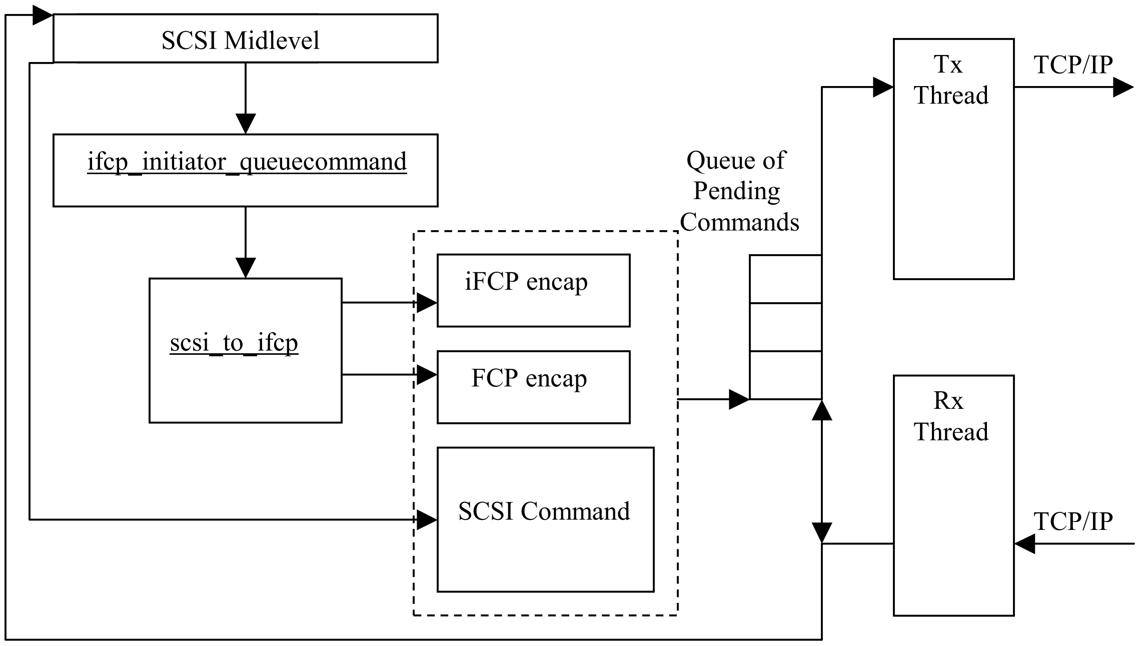


Figure 33: Initiator Command Queue

The initiator queuing mechanism depicted in Figure 33 works as follows. The initiator module creates two threads, one of which transmits frames, the other of which receives them. After the devices have logged in, the threads are started, and as long as a given I/O process is not interrupted, the transmit thread operates only with the queue of SCSI Commands originated by the SCSI Midlevel, which operates via a third thread of execution. When the SCSI Midlevel has produced the Scsi_Cmnd structure that corresponds to a given command, it calls the ifcp_initiator_queuecommand function. This function calls the scsi_to_ifcp function, which identifies the SCSI CDB by its opcode, encapsulates the frame in accordance with both the FCP and the iFCP specifications, attaches the frame to the queue of pending commands, and uses a semaphore to wake up the transmit thread. Thus, when the transmit thread resumes its execution, it must merely send the frame over the TCP/IP socket. Once the frame is transmitted, its structure is not released right away. Instead, it remains on the pending commands list with a flag set to indicate the fact that it has been sent. If the frame is a Write Command, the payload of each successive outgoing FCP_DATA frame will actually replace that of the original Command, so that space is conserved. The receive thread is blocked until an incoming frame arrives via TCP/IP, and it resumes operation upon completion of a successful read operation on the TCP/IP socket. The command code of the incoming frame is then identified, and if the frame is a SCSI Response, the command which possesses the same exchange identifier on the pending command queue is identified and freed.

4.4 iFCP Initiator Operation

4.4.1 Overview

Steps which must be performed for initiator operation:

1) load the module
2) pass identifiers to the iSNS client, which registers and queries the iSNS server
3) select the desired target
4) pass target and initiator identifiers to the module via the proc file system, so that the iFCP session is established
5) perform I/O operations
6) unload the module

4.4.2 Description

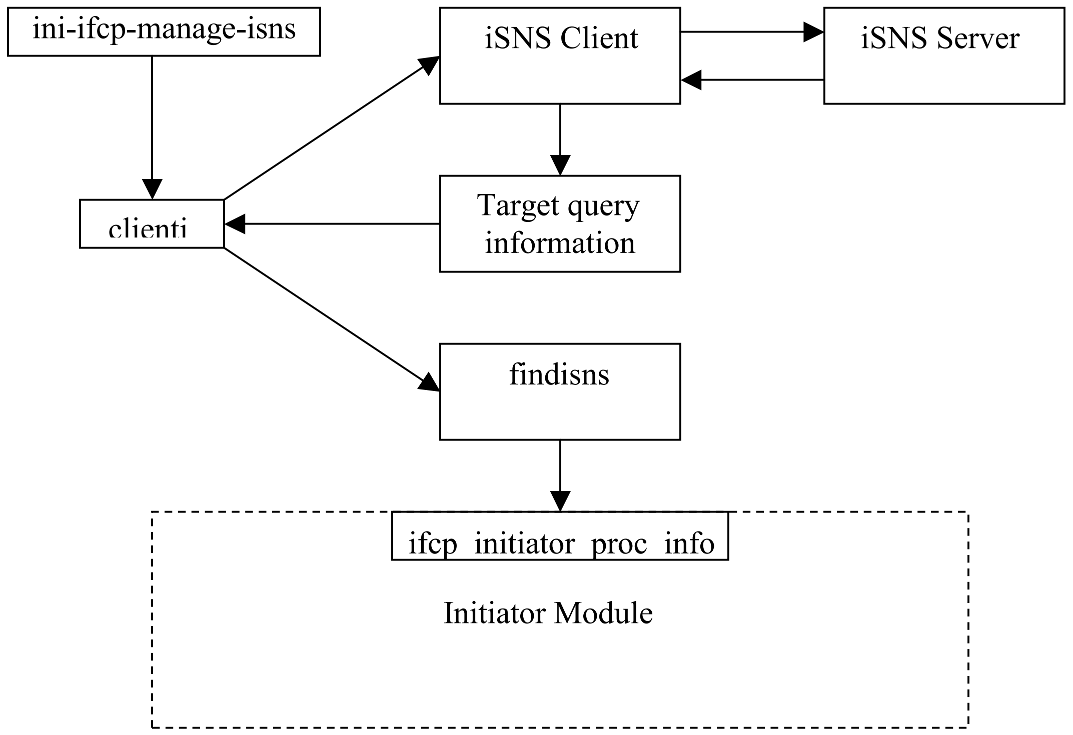


Figure 34: Initiator Operation

Prior to session establishment, the iFCP initiator module must be loaded, and it must also discover any iFCP targets in its discovery domain via the iSNS protocol. At compile time, the initiator module has already initialized the Scsi_Host_Template data structure with the necessary function pointers and values associated with its operation, including the scatter-gather list limit, and the maximum number of sectors which can be requested in a given SCSI READ or WRITE Command (see Appendix K). One of the fields of this structure contains the address for the ifcp_initiator_detect function, which is called by the SCSI subsystem when the iFCP initiator module is loaded, in order to register the iFCP initiator as a SCSI driver. The user runs the ini_install script to load the unh_ifcp_intiator module, or types "/sbin/insmod <path>/unh_ifcp_initiator.o". When it is loaded, the initiator module passes the Scsi_Host_Template structure as a parameter to the scsi_register function, in order to register with the SCSI Midlevel as a SCSI host adapter. After the module has been installed, the user runs the ini-ifcp-manage-isns shell script, in order to operate the initiator using the iSNS discovery protocol. In the ini-ifcp-manage-isns script, the user may set variables for the iSNS server IP address, the Fibre Channel address of the initiator, as well as its unique port identifiers.

The ini-ifcp-manage-isns script calls the clienti executable, which is responsible for running the iSNS client and causing it to communicate with the iFCP management tool. The clienti executable forks the iSNS client process and feeds the previously mentioned user assigned variables into it through an input pipe. These are used to register the initiator as an iFCP device with the iSNS server. After it receives a successful registration response, clienti causes the iSNS client process to query the server, and receive back a list of addresses for iFCP target devices. The user is prompted to select the desired iFCP device from a list that clienti prints to standard output.

The values corresponding to the selected target's Fibre Channel identifiers and IP address, along with the initiator variables that were previously fed in via the ini-ifcp-manage-isns script, are then passed to the findisns script as command line parameters by clienti. The findisns script then passes the variables which contain the initiator and target identifiers to the ifcp_initiator_proc_info function, which stores them in the appropriate locations within iFCP initiator.

At this point, the findisns script calls the ifcp_config executable, which reads the IP address of the iFCP target from the proc file system entry, and creates the necessary entry in the /dev directory which will be used as an access point for the device. In turn, the ifcp_initiator_proc_info function of the loaded module reads in the information from /proc/scsi/ifcp_initiator/<host number> file, and creates a new session to which all pertinent information is passed to SCSI in the host structure. When the new session is created, the initiator module starts the transmit and receive threads, creates the TCP/IP connection, and calls the ifcp_initiator_login function, causing the iFCP initiator to initiate a session with the iFCP target.

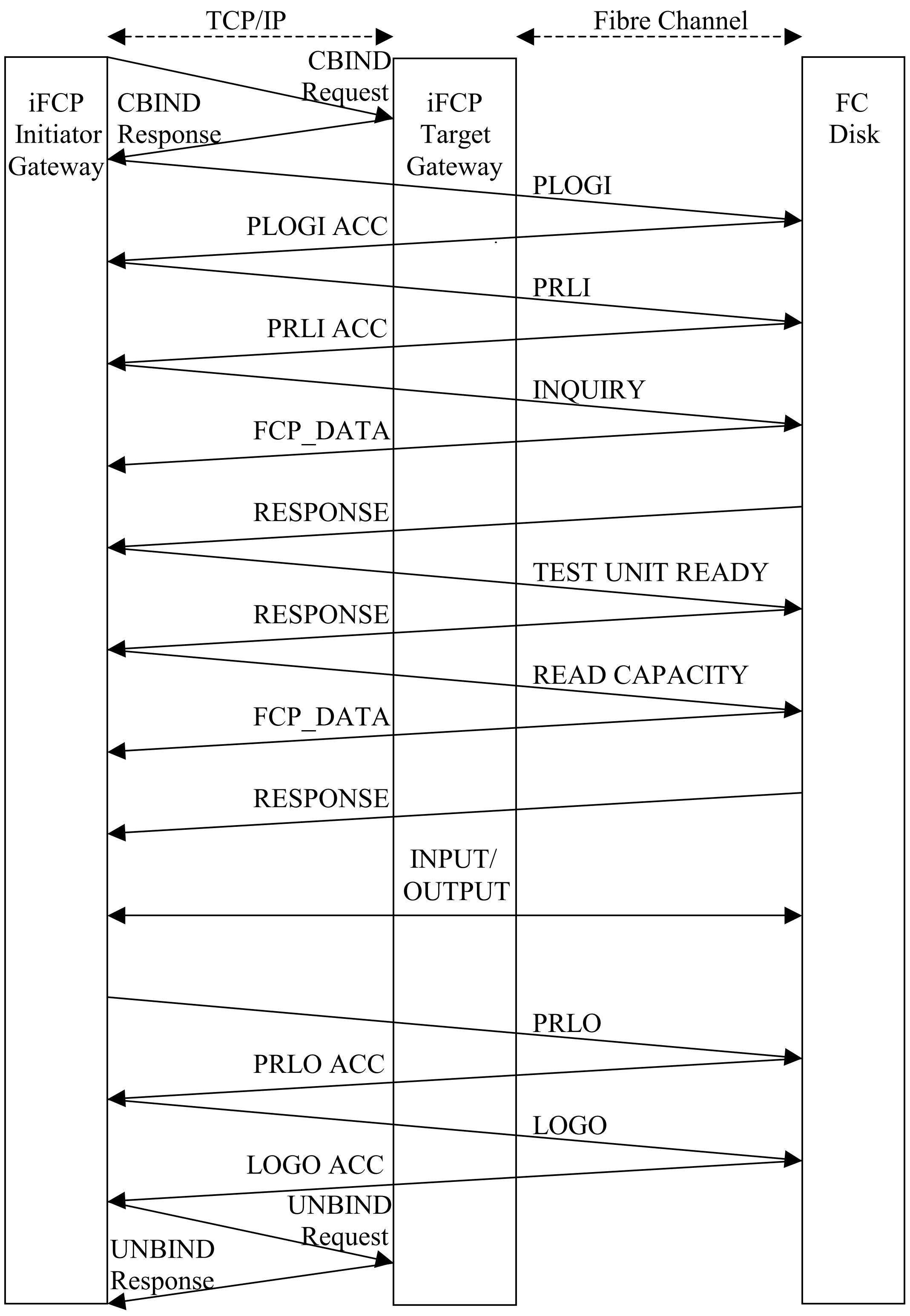


Figure 35: iFCP Gateways and Fibre Channel Disk

In order to establish the iFCP session, the initiator transmits a CBIND Request to the target (Figure 30). If it receives a positive CBIND Response in return, it performs login. During login, the FLOGI_ACC transmitted by the target contains a bit which indicates whether or not it is indeed an F_Port, and the initiator checks this bit before it transmits any generic registration frames. If this bit is set, the initiator checks the D_ID field to see if it has received a different N_Port ID assignment, completes the login process, and proceeds to then transmit generic and state change registration frames in order to register itself with the iFCP target. Additionally, the initiator transmits two generic query frames to the iFCP target, one of which is designed to provide the initiator with the World Wide Name of the target, the other of which is designed to provide the initiator with the corresponding N_Port ID of the target.

If the initiator has received notification through its generic query frame response that a Fibre Channel disk is attached to the iFCP target, the initiator transmits a second CBIND Request, receives the CBIND Response, and proceeds to transmit frames which are directed to the address of the Fibre Channel disk (Figure 35). First, the initiator transmits a PLOGI frame, this time to exchange login parameters directly with the Fibre Channel disk. Upon receiving its PLOGI_ACC, the initiator proceeds to transmit a Process Login (PRLI). This is an Extended Link Service frame defined by the FCP specification in order to prompt the exchange of preliminary SCSI parameters, such as whether the originator is operating as a SCSI target or initiator, and whether the originator supports functions such as the retransmission of data and the transmission of unsolicited data. In response, the disk transmits a Process Login Accept (PRLI_ACC), which contains the same parameters and indicates whether or not the parameters indicated in the PRLI are acceptable. From this point on, the majority of frames generated by the initiator are originated by the SCSI layer of the host, and passed to the initiator module through the ifcp_initiator_queuecommand function described in section 4.3.

If an iFCP target has indicated that it is an N_Port, the initiator performs as depicted in Figure 35. The implication here is that the initiator will not be able to rely on the iFCP target for any part of its N_Port ID, as well as any discovery information which might otherwise be received via queries and state change notifications, and that the iSNS server must be solely relied upon for any such services.

In order to remove the iFCP initiator module, the user runs the ini_uninstall script, or types "/sbin/rmmod unh_ifcp_initiator". The SCSI Midlevel calls the ifcp_initiator_release function when the module is removed. This function causes the initiator to transmit a Process Logout frame (PRLO) to the disk, in order to logout with the FCP layer, and to wait for a Process Logout Accept frame (PRLO_ACC) to be transmitted in return. Next, the initiator transmits a Logout frame (LOGO), to the disk, in order to terminate the Fibre Channel exchange, and waits for a Logout Accept (LOGO_ACC). The initiator has now logged out with the disk. Next, it transmits an UNBIND Request frame to the iFCP target gateway, and upon receiving the UNBIND Response, halts the execution of the transmit and receive threads, and calls the scsi_unregister function, so that the initiator device will no longer be recognized by the

SCSI Midlevel. The ifcp_initiator_release function also calls the initiator_unregister function, so that any existing partitions on targets that had been connected to the host through the initiator module are no longer recognized as an extension of the host file system.

4.4.3 **Input/Output**

During the preliminary phase of a SCSI session, the SCSI Midlevel issues a SCSI Inquiry Command, the initial SCSI frame which is used to determine the configuration of the SCSI target's logical unit(s). The disk replies with a FCP_DATA frame, containing Inquiry response data which provides information regarding factors such as whether or not the device is sequential, whether multiple ports are present, and whether certain optional commands are supported. The initiator module must then process the FCP_DATA frame, by checking certain fields, removing the iFCP and FCP portions of it, and storing the SCSI data portion of the frame into the appropriate address in the Scsi_Cmnd structure which was previously provided by the SCSI Midlevel. This is followed by a SCSI Response frame from the disk, which contains the status of the Inquiry (i.e. whether or not the command has been accepted). If a negative status of "Check Condition" is provided in the SCSI Response frame, the initiator must pass a failure code to the SCSI Midlevel, along with the corresponding SCSI sense data contained in the frame, which may provide the SCSI layer of the host with supplementary information concerning the failure of the command.

The next command issued by the SCSI Midlevel is the Test Unit Ready Command, which is used to discover whether or not the target controller is prepared to provide access to its media. The host transmits these commands periodically if necessary, until it receives a SCSI Response with good status.

Due to the fact that the Fibre Channel disk has already indicated the fact that it is a block device in the Inquiry response data, the next command issued by the SCSI Midlevel is a Read Capacity Command, in order to determine the block capacity of the disk. Again, the disk responds with an FCP_DATA frame, followed by a SCSI Response.

At this point, the initiator and target are ready to transfer data to and from the disk. If the disk is being introduced to the operating system of the host for the first time, it may need to be formatted and partitioned. In order to perform this task, preparing the disk for user access, the System Administrator may utilize the fdisk command. The user may then use some version of the mkfs command on a given partition in order to establish the desired file system. In order to mount the disk onto the host file system, the user must then add the appropriate entry for the partition and its corresponding directory into the /etc/fstab file, and then issue the mount command. Once mounted, the disk is available to all users as if it were a “local” disk on the initiator’s host platform.

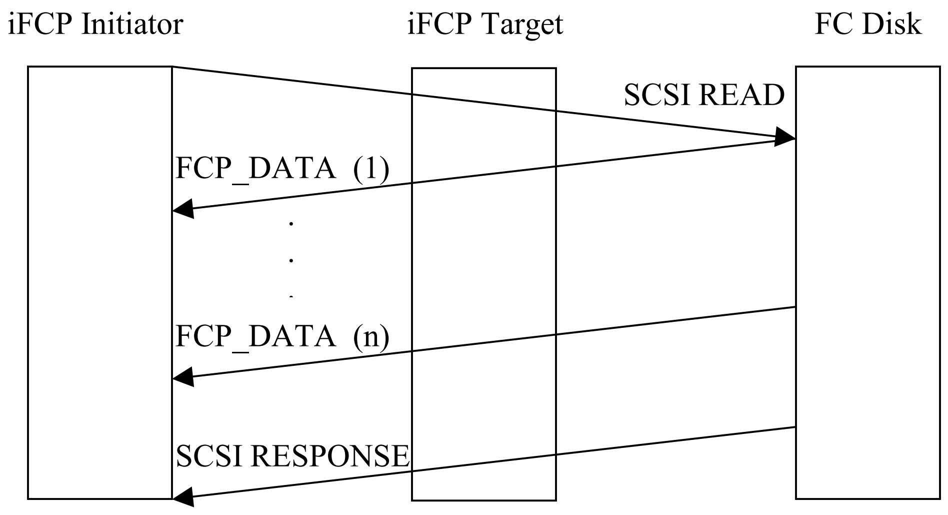


Figure 36: Read Operation

To read data from the disk, the frame exchange illustrated in Figure 36 is used. During this exchange, the SCSI Midlevel transmits a SCSI READ CDB to the iFCP initiator, the initiator encapsulates the command using the FCP and iFCP protocols, places it on the transmission queue, wakes the transmit thread with the transmit semaphore, and the transmit thread sends the frame to the Fibre Channel disk through the iFCP target. In response, the Fibre Channel disk transmits a sequence of FCP_DATA frames to the FL_Port of the iFCP target, addressed to the initiator. These frames have already been encapsulated via FCP, so the iFCP target must only add the iFCP headers to them, along with their respective SOF and EOF words. After the disk has transmitted the number of bytes requested by the initiator, the disk also transmits a SCSI Response frame to the initiator through the iFCP target.

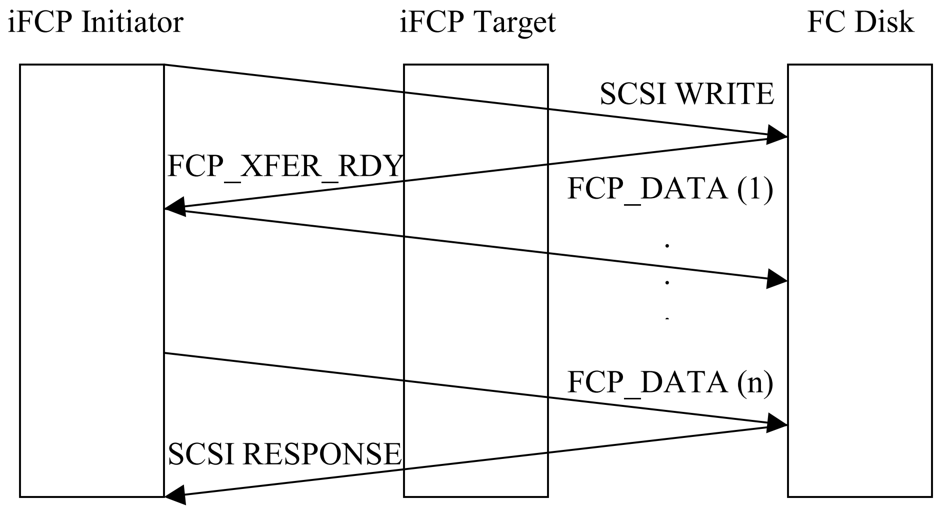


Figure 37: Write Operation

To write data to the disk, the frame exchange illustrated in Figure 37 is used. The SCSI Midlevel transmits a SCSI WRITE CDB to the iFCP initiator, and the initiator transmit thread sends the frame to the Fibre Channel disk through the iFCP target. At this point, the initiator waits for a Transfer Ready frame (FCP_XFER_RDY), which is originated by the disk in order to indicate the amount of buffer space it has allocated for incoming write data. When the Transfer Ready frame has been received by the receive thread, the initiator transmits the number of bytes indicated in the data length field of the received frame, in the form of multiple FCP_DATA frames. It then proceeds to wait for a SCSI Response to be transmitted from the disk.

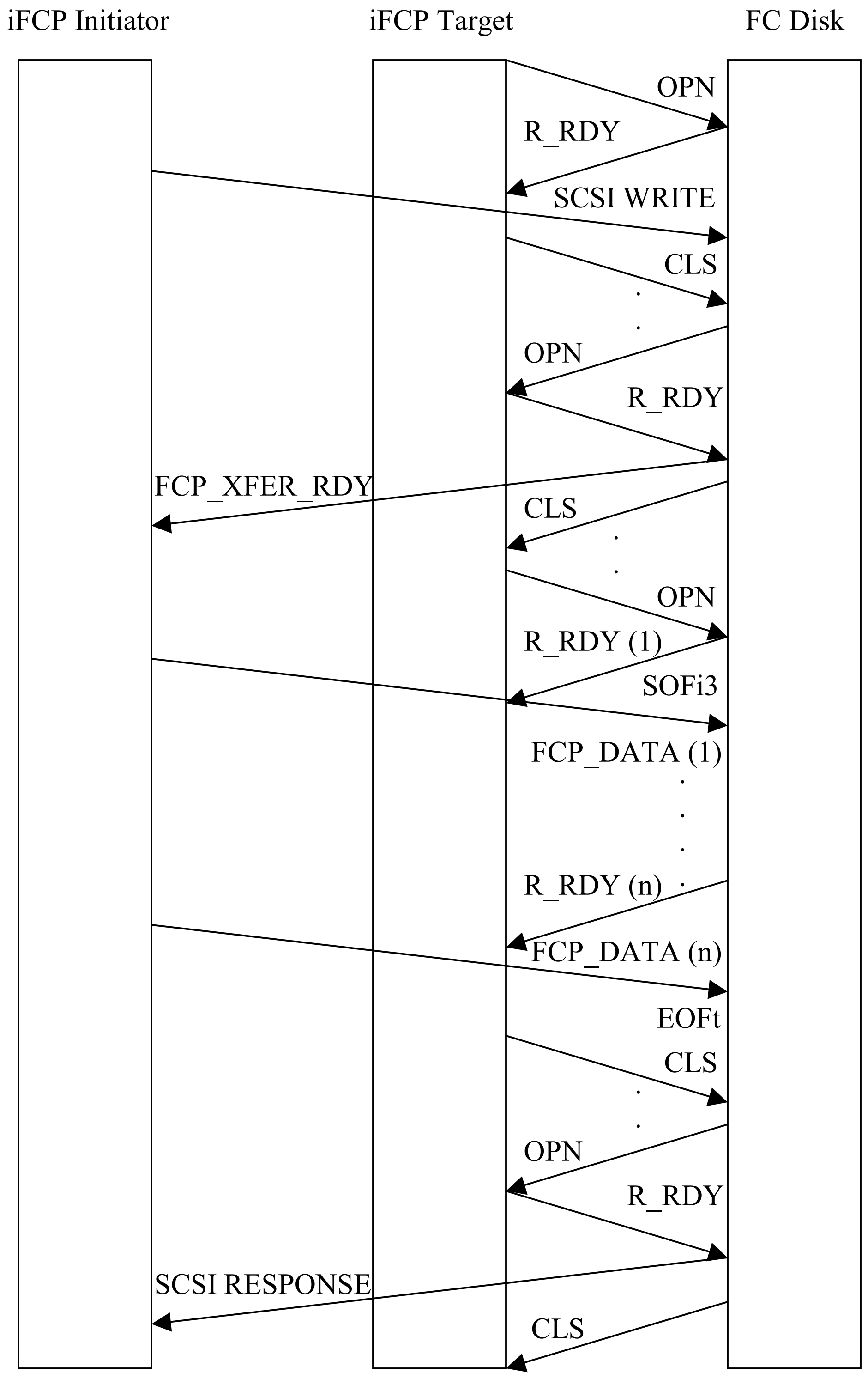


Figure 38: Fibre Channel Frame Sequence Transmission

In order to transmit a single Fibre Channel frame to the disk, as discussed in section 2.1, the FL_Port of the iFCP target must transmit an OPN Primitive Signal to the Fibre Channel disk, wait for an R_RDY to be sent from the disk, transmit the frame, and transmit CLS to the target. However, when multiple frames are being transmitted over Fibre Channel within a sequence, the sender only transmits an OPN before the first frame of the sequence, and a CLS after the last frame of the sequence. In this particular situation, the beginning of the first frame is marked with an SOFi3, and the final one is terminated with an EOFt, while all of the other Start of Frame and End of Frame Ordered Sets are SOFn3 and EOFn, respectively. Therefore, as Figure 38 demonstrates, when the iFCP target receives a burst of data from the iFCP initiator, it detects the fact that the iFCP initiator gateway has put an iFCP SOFi3 at the beginning of the first frame to signify the beginning of a sequence, and sends OPN to the Fibre Channel disk before it waits for an R_RDY to be transmitted by the disk to indicate buffer-to-buffer credit (see Appendix L)[1][2]. Here, the iFCP target detects the fact that the first frame contains an EOFn, and as a result it does not transmit a CLS after it transmits the first data frame, but instead it simply transmits the next FCP_DATA frame sent by the initiator upon receiving a second R_RDY. This time, the FCP_DATA frame will most likely have an SOFn3 at the front and an EOFn at the end, to signify the fact that it is located somewhere in the middle of a sequence. Finally, when the last FCP_DATA frame is received, it will start with an SOFn3 and end with an EOFt, indicating the fact that it is located at the end of a sequence. The target iFCP gateway notes this, as well as the total data byte count of the sequence, and sends CLS to the Fibre Channel disk immediately following the last FCP_DATA frame.

4.5 Target Data Structures

The iFCP target will use the iSNS protocol in a similar manner to that of the initiator in order to register with the server. Although it is required in the specification, it is not as crucial for the target to query the iSNS server as it is for the initiator, since it does not initiate the TCP/IP connection. Additionally, Fibre Channel does not require a discovery protocol, since a direct connection is established among Fibre Channel devices, and they discover the necessary information regarding one another during the initialization process. Therefore, no additional discovery process is required to be present between the iFCP target gateway and the Fibre Channel disk.

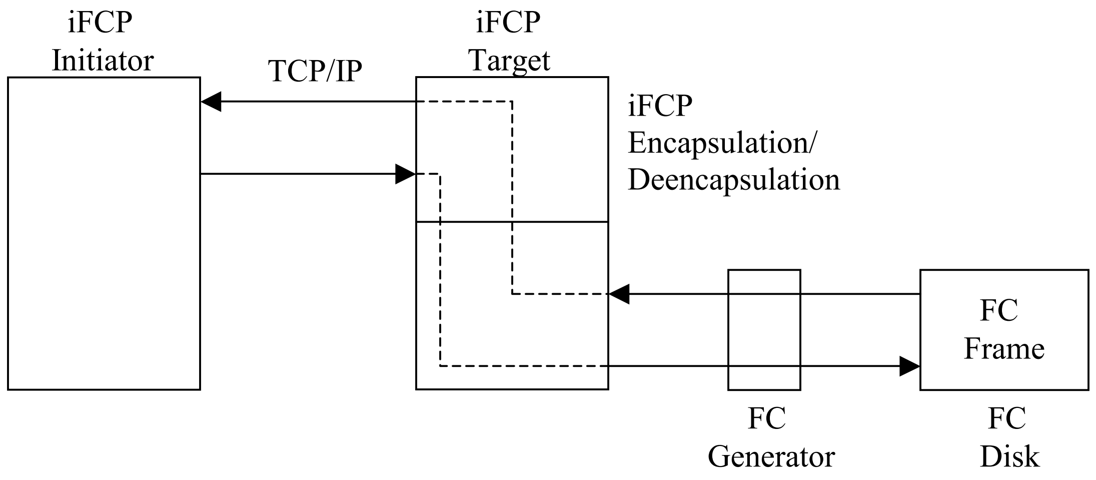


Figure 39: Path of Fibre Channel Frame Through iFCP Target

The format of the data structures and functions contained within the target are similar to that of the initiator. However, the functions that are associated with specific data structures in the target are organized into classes, and while the initiator must interface with the SCSI Midlevel, the target must interface with the Fibre Channel target via the generator (Figure 39). Therefore, new functions and classes are introduced within the target which handle the lower layer functionality of the Fibre Channel protocol as well as the generator-specific function calls, while the calls which communicate with the SCSI Midlevel and queue the SCSI commands are eliminated.

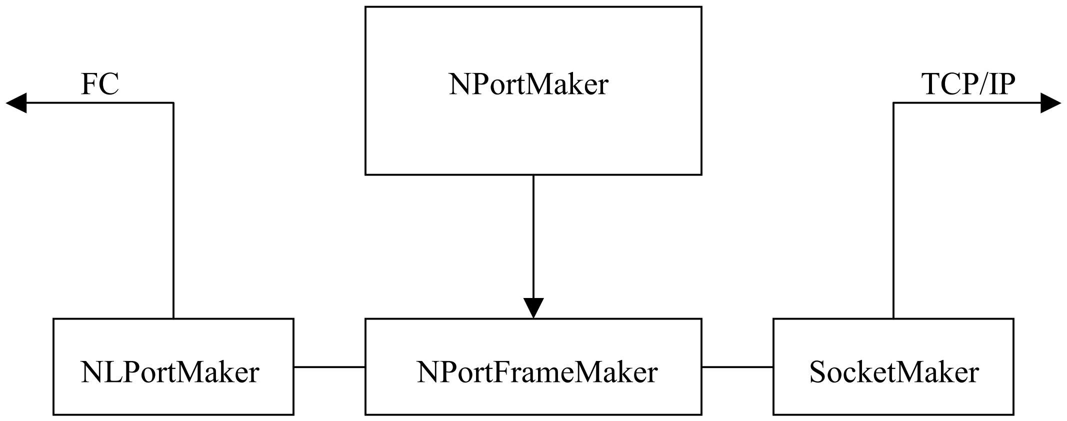


Figure 40: iFCP Target Class Structure

The target NPortFrameMaker class creates the actual iFCP and Fibre Channel frames that are used by all other classes (see Appendix M). All Fibre Channel frames which are used after initialization, during and after login, including FCP SCSI frames, in addition to the CBIND, UNBIND and LTEST iFCP frames are built by this class (Figure 40).

The SocketMaker class contains no data structures of its own. This class contains functions that are related to the establishment and usage of the Windows socket mechanisms. These functions encapsulate and deencapsulate frames using the iFCP protocol, while utilizing the information provided by the data structures in the NPortMaker class in order to transmit and receive frames over TCP/IP.

The primary data component in the design of the target is the NPortMaker class. In a similar manner to the nport_maker structure of the initiator, this class contains the structures necessary to implement the Fibre Channel portion of the target iFCP gateway. The np data structure contained within the NPortMaker class is an exact replica of the nport structure which resides within the nport_maker structure of the initiator. Additionally, the NPortMaker class contains the functions which are specifically used by the NPortMaker data structure in order to identify and process incoming frames, send frames, and so forth. Due to the fact that the target does not contain connection or session structures, the NPortMaker also possesses the capability to be linked to other NPortMaker structures, in case multiple sessions are initiated. The SocketMaker and NPortFrameMaker classes are friend classes of the NPortMaker class, so that information may be shared among them.

The LoopInitMaker class is a minimal component. It simply allows the iFCP target to perform as a switch during loop initialization with any Fibre Channel target, and to record the Arbitrated Loop Physical Address selected by the target, before the NLPortMaker class transmits the higher level frames over the Fibre Channel bus. The NLPortMaker class contains a pointer to the current NPortMaker class, and it utilizes the data structures of the NPortMaker class in order to transmit and receive frames to the Fibre Channel target over the Fibre Channel Arbitrated Loop, after the LoopInitMaker class has initialized the Loop.

4.6 Target Operation

A script runs the iFCP executable. Once it is running, it creates an instance of the NPortMaker and performs the initial login and registration exchanges with the Fibre Channel disk displayed in Figure 29. It then creates another instance of the NPortMaker, assigns it a target number index, and binds it to a socket, listening for and accepting an incoming connection on its iFCP port. When it begins to receive incoming frames from the initiator via TCP/IP, the iFCP target gateway identifies them, and subsequently processes them if they are iFCP-specific, or if they are Fibre Channel frames that are addressed directly to it. In this case, the iFCP target stores any information necessary, and it then generates the appropriate response frame, encapsulates it, and transmits it over TCP/IP. The frames which are not addressed directly to the iFCP target are only processed enough so that they can be deencapsulated properly and their types identified, before they are passed to the Fibre Channel target through the generator. Additionally, the Fibre Channel addresses of the frames are replaced with the address in the session descriptor component of the NPortMaker structure, if the session is taking place in address translation mode. When the transaction requires a response, the iFCP target waits for a response to come back directly from the Fibre Channel disk through the generator, encapsulates it and transmits the response to the iFCP initiator via TCP/IP. This process takes place synchronously, since the target implementation does not contain multiple threads. Due to the fact that the FL_Port of the iFCP target is designed to act like a Fibre Channel switch, the target continues to execute until it is stopped by the user, or until the Fibre Channel connection between the FL_Port and the Fibre Channel target is broken.

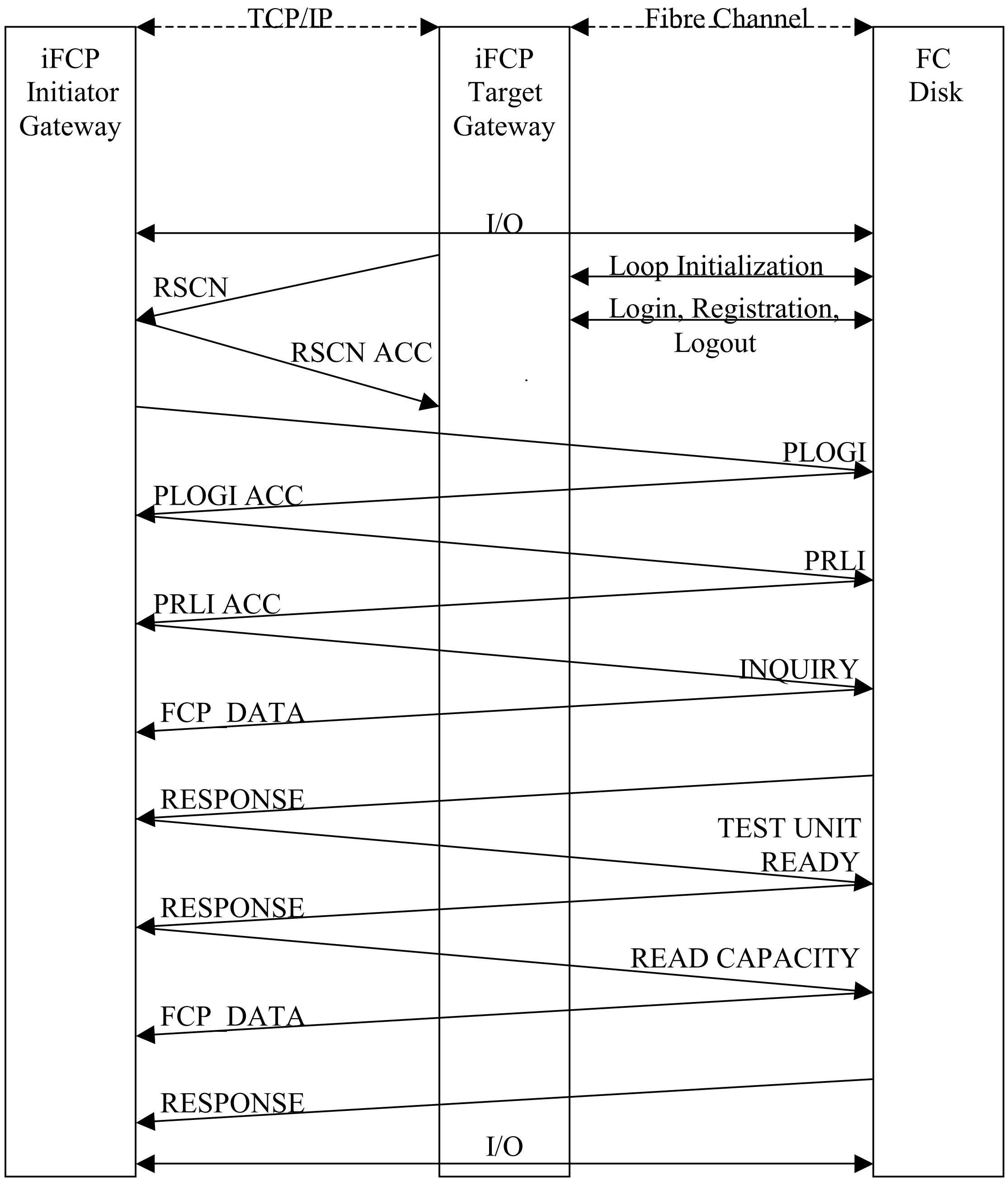


Figure 41: iFCP Error Recovery Procedure

4.7 Error Recovery

Unlike the TCP/IP port of the iFCP target gateway, the iFCP target FL_Port must constantly be ready to reinitialize with the Fibre Channel disk and to transmit a Registered State Change Notification frame (RSCN) to the iFCP initiator if it receives an indication that the Fibre Channel disk has detected an error on the loop, and therefore that the session must be recovered (Figure 41). Three of the most reliable indicators of the fact that the loop must be recovered are: Loop Initialization Primitives (LIP), Loop Initialization Select Master (LISM) frames, and timeouts. Although the second of these may appear redundant, as the LISM is often transmitted not long after a LIP is transmitted on any given loop, it is still checked in the event that the LIP is retransmitted for a very short duration and is therefore difficult to detect. The type of error that stimulates this type of response from the disk is generally triggered when a piece of hardware that sits between the Fibre Channel disk and the Fibre Channel generator, such as an old cable or gbic, malfunctions and transmits incorrect Ordered Sets as a result.

When the Fibre Channel disk reads an Ordered Set that it does not recognize, the Arbitrated Loop State Machine inside the disk may fail in quite a few different ways, and it will therefore sometimes fail to recognize the frames that have been addressed to it afterwards. This has often been observed to take place during one of three scenarios. In the first scenario, the OPN Primitive Signal transmitted before the frame is not recognizable, causing the device not to process any subsequent frames which arrive. In the second scenario, the Start of Frame Ordered set is malformed, causing the frame to appear as if it were a large sequence of unidentified data words. In the third scenario, the

CLS Primitive Signal is not recognizable, and the disk continues to wait for frames that should not be delivered. In all three situations, the disk will either timeout and reinitialize the loop, or the switch will timeout upon waiting for a response frame, attempt to resend the frame, and reinitialize the loop if it experiences a second timeout. If initialization proves to be necessary, the initiator will perform login with the Fibre Channel disk upon receiving the Registered State Change Notification (RSCN) frame, after it transmits a Registered State Change Notification Accept (RSCN ACC), as depicted in Figure 41 (see Appendix I) [1].

The initiator must also notify the SCSI Midlevel that an error has taken place, and that any pending transactions must be aborted. Therefore, it returns an error code to the SCSI Midlevel in response to the current transaction, and as a result, the Midlevel calls the ifcp_initiator_abort function. The initiator does not need to explicitly logout with the target, as it does when the initiator module is removed, since the target has implicitly logged out already. Instead, the initiator performs login with the target, followed by process login, and a line is again written to the /proc/scsi/scsi file which causes the SCSI Midlevel to transmit another set of preliminary SCSI Commands, starting with the SCSI INQUIRY, and to continue the exchange from the point where the last Command was responded to with a positive status.

In this chapter, we have examined the data structures which provide the internal framework for both the iFCP initiator and the iFCP target. We have traced the execution of the initiator, and how its operation is triggered by the application layer and the SCSI layer of the initiator host. We have also traced the execution of the target and its primary functions, as it both responds to and forwards the frames which arrive on both of its interfaces. Additionally, we have analyzed the frame exchanges that take place between the iFCP target and the Fibre Channel target, the iFCP initiator and the iFCP target, and finally between the iFCP initiator and the Fibre Channel target. We have also looked at what happens when errors take place during a session, and the actions which are taken in order to preserve the session.

V PERFORMANCE TESTING

5.1 Procedure

The performance measurements taken with the iFCP target and initiator are not extensive, since the goal of this project is primarily based upon its functional aspects. However, since the project is functional, it is interesting for us to discover where any performance or efficiency bottlenecks have occurred, so as to educate ourselves regarding different aspects of the protocol in general, and to provide awareness for any future users. In order to accomplish this goal, measurements have been taken of the traffic flow rates between the iFCP initiator and the Fibre Channel disk, and between the iFCP target and the Fibre Channel disk, for purposes of comparison.

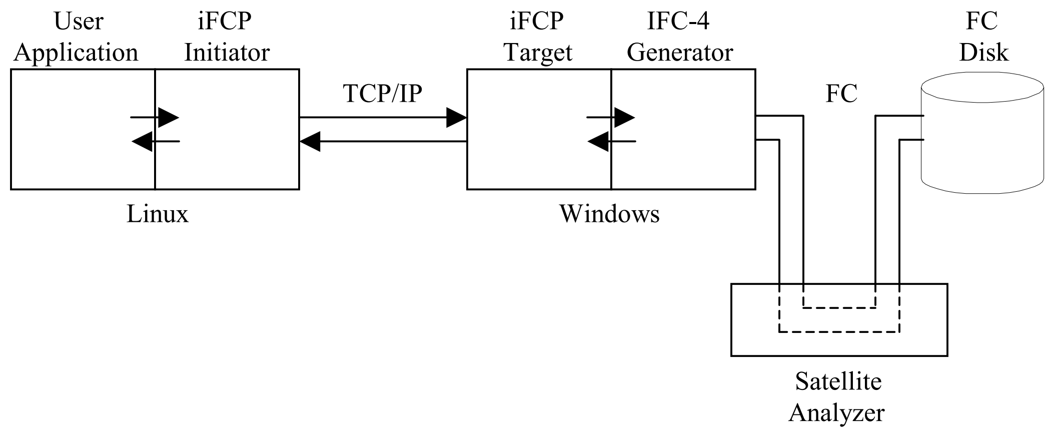


Figure 42: Test Setup 1

5.2 <u>Test Setup 1</u>

During the first round of testing, a user-level application program on the iFCP initiator host triggered Reads and Writes of one Megabyte, over the topology shown in Figure 42. The burst sizes and individual data frame sizes for these commands were set in the code of the iFCP initiator module, and the fact that the received data frame size should not exceed 2048 bytes was also conveyed by both the iFCP initiator and the Fibre Channel target during login.

In this manner, one Megabyte writes were transmitted from the user application to the Fibre Channel disk via iFCP. These writes took place in 128 KB bursts, with 2 KB of data in each FCP_DATA frame sent from the iFCP initiator to the Fibre Channel target. The results were quite extreme. The transfers consistently occurred at 0.23 MBps, which was very slow.

The I-TECH IFC-4016 Satellite analyzer was used during this process to record the time which was taken for the iFCP bursts to travel over Fibre Channel. The analyzer displayed the fact that it was taking at least 4.44 seconds for the 1 MB of data to travel over Fibre Channel.

Another surprise occurred when one Megabyte reads took place over the iFCP link, again triggered by the application layer in the iFCP initiator host. These operations were performed in a similar manner to the writes, causing 128 KB bursts of traffic with 2 KB of data in each FCP_DATA frame to travel from the Fibre Channel target to the iFCP initiator. When they were measured from the iFCP initiator host, the reads performed at 0.12 MBps. According to the I-TECH IFC-4016 Satellite analyzer, one such transaction took 9.80 seconds to travel across Fibre Channel alone.

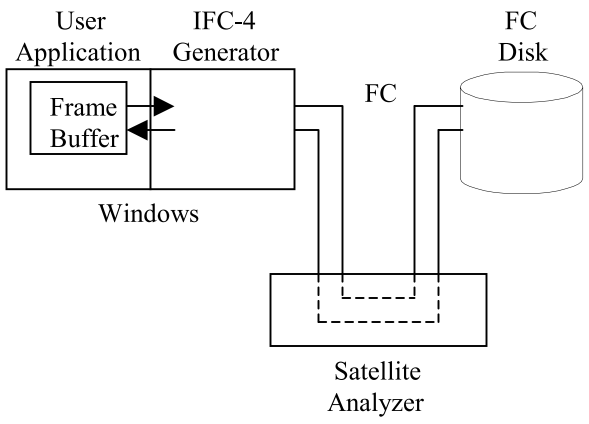


Figure 43: Test Setup 2

5.3 Test Setup 2

In order to verify that the iFCP initiator was not causing these performance issues, a second set of tests was performed using the IFC-4 generator and the Fibre Channel disk. As depicted in Figure 43, the frames which were both transmitted and received by the generator in this topology were built by an application in user space. The frame bursts and frame sizes were defined as before. In order to ensure that any software processing that took place would be relatively minimal, the data buffers created in user space were merely initialized to zero. These empty buffers were passed directly from the generator to the disk, using the I-TECH generator's frm_user function, and from the disk to the generator, using the get_frame and get_frame_info functions, all three of which must be used to pass the buffers to and from user space (see Appendix J). Although one Megabyte of data had taken 4.44 seconds to travel over Fibre Channel in test setup 1 (0.23 MBps), the same amount of data took 4.0 seconds to travel over Fibre Channel in test setup 2 (0.25 MBps). Therefore, the encapsulation and deencapsulation overhead of iFCP, along with the TCP/IP transmission, took only 0.44 seconds, or about 10% of the total cost of transmission in setup 1.

Similarly, when the data bursts in test setup 2 were measured individually, each 128 KB burst took an average of 498.2 milliseconds from the generator to the disk, while in test setup 1 each 128 KB burst took an average of 548.7 milliseconds over the Fibre Channel loop. In this case, eliminating the iFCP initiator and target and the TCP/IP link saved only 50.5 milliseconds, or about 9% of the total cost.

Again, the handicap imposed upon the generator by user-space frame buffers was even more apparent when reads were performed than writes, which appears to demonstrate the fact that buffer transfer from the Fibre Channel wire might be more difficult than buffer transfer to the wire. The one Megabyte read, which had taken 9.80 seconds in test setup 1 (0.10 MBps), took 8.48 seconds in test setup 2 (0.12 MBps), demonstrating that the bandwidth utilization percentage measured with iFCP in test setup 1 was roughly 87% of what it was in test setup 2.

As expected, the rate that was recorded for the read operations increased slightly when it was measured in bursts. Test setup 1 averaged approximately 1.08 seconds per

128 KB burst, while test setup 2 performed at an average speed of 1.05 seconds per burst. Here, test setup 1 with iFCP ran at about 97% of test setup 2 without iFCP, indicating that an extremely small amount of overhead was caused by iFCP. These results are displayed in Figure 44.

	Total	Per Burst
Write	10%	9%
Read	13%	3%

Figure 44: Approximated Data Transfer Costs Associated with iFCP

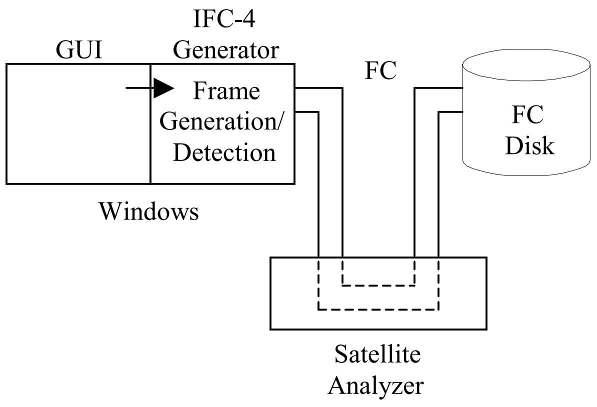


Figure 45: Test Setup 3

5.4 <u>Test Setup 3</u>

The third set of tests was performed with a different user application, as shown in Figure 45. This time, the generator functions that were called by the user were comprised only of black box functions operating from a generator-defined address space, perhaps involving direct access to registers located on the card itself. This type of function comprises the vast majority of library functions that have been created for the generator. When this type of function is called, the user is provided with access to a Graphical User Interface, and only a few parameters. Naturally, this is done to increase the speed of the card, and to simplify the role of the user. In this case, although the user defined the burst sizes, the user did not control the individual frame sizes. The frames were 2KB in length, as they were in test setup 1 and test setup 2.

When the generator was instructed to create Write Commands and transmit its own bursts of simulated data to the disk, the difference was dramatic. One Megabyte of data was transferred by the generator in 31.45 milliseconds, for a rate of 31.80 MBps, compared with 0.25 MBps in test setup 2. The observed data transfer rate in test setup 3 exceeds the data transfer rate in test setup 2 by a factor of approximately 127. Likewise, when the generator was programmed to generate its own Read Commands, one Megabyte of data was transferred to the generator from the disk in 31.81 milliseconds, for a rate of 31.44 MBps, exceeding the transfer rate of 0.12 MBps in test setup 2 by a factor of 262. The large differences between the rates observed in test setup 2 and the rates observed in test setup 3 are due entirely to their respective interfaces with the generator.

5.5 <u>Testing Conclusions</u>

One major point to consider in evaluating the performance of the I-TECH generator that has been incorporated into the iFCP target implementation, at least for the time being, is that this Fibre Channel generator was developed for frame generation purposes, not for frame tunneling purposes. As a consequence, the generator functions are composed mostly of upper level automated I/O simulations that can make the job of the Fibre Channel test script writer relatively simple.

However, in this particular situation, these features are not useful, since the iFCP target does not generate test frames, but rather obtains its frames from the iFCP initiator and Fibre Channel disk. Therefore, the iFCP target requires only a byte translator and detector at the Fibre Channel level, or a mechanism whereby ordered sets and frames can be encoded and decoded, transmitted and received via the 8-bit/10-bit scheme of the FC-1 and FC-2 layers, in accordance with the Fibre Channel Arbitrated Loop State Machine.

For our purposes, the generator provides a great deal of unnecessary overhead. The generator operation that is required during user-space buffer transfers is expensive, and there appears to be no solution with this particular version of this generator. When the manufacturers were consulted, they provided us with one suggestion involving some file transfer functions, but this method caused the target portion of the iFCP implementation to become even slower.

Additional problems were encountered at various stages during the testing which raise some interesting questions. At one point, one millisecond delays were inserted between each Read Command in test setup 3 to ensure that all data would be received properly between commands, since synchronization was lost when the get_frame or get_frame_info functions were inserted at these points, with or without blocking. According to the analyzer, the programmed 1 ms delays translated into 15 ms delays each on the Fibre Channel loop. If each programmed operation on the generator were to introduce a delay of 15 ms, this would account for 960 ms in each transfer involving 128 KB of data for the data frame portion of the transfer alone, or 91% of the 1.05 second average for each 128 Kilobyte burst to travel over Fibre Channel during the read portion of test setup 2.

Although using the I-TECH generator as the Fibre Channel interface introduces a major inefficiency to the iFCP target, the implementation retains the majority of its value due to several factors. The simplicity of the code structure allows virtually any kind of testing, since the frames can be manipulated directly, errors can be introduced, along with additional checks, messages, and interfaces. The same is true of the iFCP initiator code.

The code is as portable as possible. Although many generator-specific commands exist which would automatically handle the Arbitrated Loop State Machine and the Fibre Channel Extended Link Service frames that are exchanged, they have not been used. Instead, generator-specific commands have been kept to a minimum, which ensures that if frames must be altered, or if a generator upgrade or replacement takes place, the code will need minimal changes to run in any Windows environment which supports C++. Again, the initiator code can also be easily ported to any Linux system with accessible source code which supports C.

Third, if a function is added to the IFC-4 generator library that efficiently handles the passing of buffers between user-space and generator-space, it can be incorporated into the implementation very quickly and easily.

VI Conclusions

In this thesis, we have reviewed the basic concepts which define the establishment of a Storage Area Network. We have discussed the reasons for which the SAN was invented as a platform independent vehicle for data storage among multiple hosts and storage units. The fact that the Internet has become so widespread and robust, and that the demands of the data storage community have grown has led us into our discussion regarding IP Storage protocols. Some of the recent forms IP Storage has taken have been summarized: iSCSI, FCIP and iFCP. IP Storage, in general, provides us with a deeper, more flexible and extensive concept of what a Storage Area Network can become.

Some of the basic fundamentals of the Fibre Channel technology have been provided in this thesis, which allow us to understand the origins of many Storage Area Networking mechanisms. We can understand on a basic level how Fibre Channel devices discover one another during initialization, which enables them to establish the lower layers necessary for the transport of subsequent Fibre Channel frames.

From there, we have transitioned into the focal point of the thesis, iFCP, a protocol which defines the means by which Fibre Channel end devices can communicate over the Internet. We have reviewed iSNS, the means by which iFCP gateways are intended to discover the presence of one another. We know that four new frames have been introduced with which iFCP gateways establish and terminate iFCP sessions, and that one additional frame may be periodically transmitted during the iFCP session. The other frames which are exchanged during a given iFCP session consist of Fibre Channel frames that are addressed and encapsulated in a specific manner for transport over TCP/IP. We also know that although the iFCP specification does not provide us with many strict guidelines regarding the usage of these Fibre Channel frames, we can draw upon our Fibre Channel specifications and end devices for details concerning these exchanges.

After our discussion of the iFCP protocol, we have examined our choices for the target and initiator implementation designs. We have then taken a closer look at each implementation and its data structures separately, since each was created for a different purpose. In addition to the Fibre Channel and iFCP functions which it is capable of performing, the initiator has been provided with a mechanism which interfaces directly with the SCSI layer of its host. In this manner, any I/O operations which the initiator module performs transpire as a result of the demands which are placed by the user on the host. The target, on the other hand, does not require a SCSI interface. Its task is to bridge the gap between two separate interfaces and implementations. It must behave not only as an iFCP gateway, but also as a Fibre Channel switch. It successfully notifies the initiator not only of its own presence, but also of the presence of the Fibre Channel target to which it is connected, making sure that all incoming frames reach their appropriate destinations.

Finally, we have measured the throughput of the resulting system. We have seen that the weakness of the design resides within the Fibre Channel generator mechanism, and that this part of the design could be improved.

VII Future Work

Clearly, the largest hindrance introduced with regard to the transmission of Fibre Channel over TCP/IP is the interface for the Fibre Channel hardware itself. We would like to look into whether or not a driver for the IFC-4 exists for the Linux platform, so that we may be able to compare these results with those obtained on the Windows platform. If we can dissect the source code for the generator on either platform to a large enough extent, perhaps we can improve the performance of the iFCP target. Additionally, we might be able to either use or develop a different Fibre Channel generator.

In the future, we would like to develop a test suite for iFCP that can be run with the iFCP implementations discussed in this thesis. Additionally, the socket and encapsulation code for this iFCP implementation might be usable for FCIP, as well, since these mechanisms are almost identical.

We would also like to interoperate with other iFCP implementations, if possible. Additionally, the people of Medusa and Finisar have been kind enough to provided us with a great deal of feedback concerning the appearance of our iFCP and Fibre Channel frames as they traversed TCP/IP, helping us clean up the aesthetics of both the initiator and target implementations quite a bit.

LIST OF REFERENCES

[1] NCITS 1331-D, Fibre Channel Framing and Signaling (FC-FS), American National Standards Institute, 2002.

[2] BSR NCITS 332, Fibre Channel Arbitrated Loop (FC-AL-2), American National Standards Institute, 1999.

[3] NCITS 1144-D, Information Systems - Fibre Channel Protocol for SCSI, Second Version (FCP-2), American National Standards Institute, 2001.

[4] INCITS 1505-D, Fibre Channel Generic Services - 4 (FC-GS-4), American National Standards Institute, 2002.

[5] NCITS 1236-D, Information Technology - SCSI Primary Commands - 2 (SPC-2), American National Standards Institute, 2001.

[6] INCITS 1561-D, SCSI Architectural Model - 3 (SAM-3), Draft, American National Standards Institute, 2003.

[7] http://www.ietf.org/internet-drafts/draft-ietf-ips-fcovertcpip-12.txt, "Fibre Channel Over TCP/IP (FCIP)", 2002.

[8] http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-20.txt, "iSCSI", 2003.

[9] http://www.ietf.org/internet-drafts/draft-ietf-ips-ifcp-14.txt, "iFCP - A Protocol for Internet Fibre Channel Storage Networking", 2002.

[10] http://www.ietf.org/internet-drafts/draft-ietf-ips-isns-21.txt, "Internet Storage Name Service (iSNS)", 2003.

[11] http://www.ietf.org/internet-drafts/draft-ietf-ips-fcencapsulation-08.txt, "FC Frame Encapsulation", 2002.

[12] Larsen, Brian D., http://cnscenter.future.co.kr/resource/networking/san-nas/BL091200.pdf, "Storage Over IP (Internet Protocol), Fibre Channel versus SCSI/IP", 2000.

[13] Eiβfeldt, Heiko, http://www.tldp.org/HOWTO/SCSI-Programming-HOWTO.html, "The Linux SCSI programming HOWTO", 1996.

[14] Tate, et al., http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg245470.pdf, "Introduction to Storage Area Networks", 2003.

[15] http://www.sec.gov/news/studies/34-47638.htm, "Interagency Paper on Sound Practices to Strengthen the Resilience of the U.S. Financial System", U.S. Securities and Exchange Commission, 2003.

[16] Clark, Tom, "IP SANs: A Guide to iSCSI, iFCP, and FCIP Protocols for Storage Area Networks", Addison Wesley, 2002.

Appendix A

Fibre Channel Loop Initialization Frames:

This appendix provides us with an introduction to the manner in which the Fibre Channel Arbitrated Loop initialization frames are structured. It is intended not only to provide us with an idea of what these frames contain, but also to present an example of a typical FC-2 layer frame.

Start_of_Frame delimiter - 4 bytes

```
SOFiL
```

Frame_Header - 24 bytes

```
22XXXXXX | 00XXXXXX | 01380000 | 00000000 | FFFFFFFF | 00000000
```

where 'XXXXXX' is hex '000000' for an FL_Port and hex '0000EF' for an NL_Port or F/NL_Port, or some other value specified by a future standard.

Payload - 12, 20, or 132 bytes

```
LI_ID   8-byte Port_Name

 and    16-byte AL_PA bit map

LI_FL   128-byte AL_PA position map (1-byte offset followed by up to 127 AL_PAs)
```

where LI_ID and LI_FL contain the following:

LI_ID (Identifiers) (16 bits)
Value (hex) Name Description Payload size)
'1101' LISM Select Master based on 8-byte Port_Name (12-byte)
'1102' LIFA Fabric Assign AL_PA bit map (20-byte)
'1103' LIPA Previously Acquired AL_PA bit map (20-byte)
'1104' LIHA Hard Assigned AL_PA bit map (20-byte)
'1105' LISA Soft Assigned AL_PA bit map (20-byte)
'1106' LIRP Report AL_PA position map (132-byte)
'1107' LILP Loop AL_PA position map (132-byte)

LI_FL (Flag) (16 bits; all 'r's are reserved—not checked, but originated as zero)
LI_ID Flag Mask (binary) Meaning
LISM - rrrr rrrr rrrr rrrr reserved
LIFA - rrrr rrrr rrrr rrrr reserved
LIPA - rrrr rrrr rrrr rrrr reserved
LIHA - rrrr rrrr rrrr rrrr reserved
LISA - rrrr rrr1 rrrr rrrr LIRP and LILP supported
LIRP - rrrr rrrr rrrr rrrr reserved
LILP - rrrr rrrr rrrr rrrr reserved

Cyclic Redundancy Check - 4 bytes

```
CRC
```

End_of_Frame delimiter - 4 bytes

```
EOFt
```

Appendix B

Fibre Channel Primitive Signals:

This chart illustrates the composition of many Primitive Signals in both 32 bit (FC-2) and 40 bit (FC-1) formats. It is important to note that they all begin with either the 8 bit (FC-2) or 10 bit (FC-1) version of the comma character.

Abbr.	Link/Loop	Primitive Signal	FC-2 Ordered Set / FC-1 Ordered Set
			FC-2 Ordered Set
			FC-1 Ordered Set
Idle	both	Idle	0xBC-0x95-0xB5-0xB5
			K28.5-D21.4-D21.5-D21.5
R_RDY	both	Receiver_Ready	0xBC-0x95-0x4A-0x4A
			K28.5-D21.4-D10.2-D10.2
VC_RDY	both	Virtual Circuit Ready	0xBC-0xF5-VC_ID-VC_ID
			K28.5-D21.7-VC_ID-VC_ID
BB_SCs	both	Buffer-to-buffer state change (SOF)	0xBC-0x95-0x96-0x96
			K28.5-D21.4-D22.4-D22.4
BB_SCr	both	Buffer-to-buffer state change (R_RDY)	0xBC-0x95-0xD6-0xD6
			K28.5-D21.4-D22.6-D22.6
SYNx	both	Clock Synchronization Word X	0xBC-0x7F-CS_X-CS_X
			K28.5-D31.3-CS_X-CS_X
SYNy	both	Clock Synchronization Word Y	0xBC-0x7F-CS_Y-CS_Y
			K28.5-D31.3-CS_Y-CS_Y
SYNz	both	Clock Synchronization Word Z	0xBC-0x7F-CS_Z-CS_Z
			K28.5-D31.3-CS_Z-CS_Z
ARByx	Loop	Arbitrate	0xBC-0x94-y-x
			K28.5-D20.4-y-x
ARB(val)	Loop	Arbitrate	0xBC-0x94-val-val
			K28.5-D20.4-val-val
CLS	Loop	Close	0xBC-0x85-0xB5-0xB5
			K28.5-D5.4-D21.5-D21.5
DHD	Loop	Dynamic Half-Duplex	0xBC-0x8A-0xB5-0xB5
			K28.5-D10.4-D21.5-D21.5
MRKtx	Loop	Mark	0xBC-0x5F-MK_TP-AL_PS
			K28.5-D31.2-MK_TP-AL_PS
OPNyx	Loop	Open full-duplex	0xBC-0x91-AL_PD-AL_PS
			K28.5-D17.4-AL_PD-AL_PS
OPNyy	Loop	Open half-duplex	0xBC-0x91-AL_PD-AL_PD
			K28.5-D17.4-AL_PD-AL_PD
OPNyr	Loop	Open selective replicate	0xBC-0x91-AL_PD-0xFF
			K28.5-D17.4-AL_PD-D31.7
OPNfr	Loop	Open broadcast replicate	0xBC-0x91-0xFF-0xFF
			K28.5-D17.4-D31.7-D31.7

Appendix C

Fibre Channel Primitive Sequences:

This chart illustrates the composition of many Primitive Sequences in both 32 bit (FC-2) and 40 bit (FC-1) formats. It is important to note that they all begin with either the 8 bit (FC-2) or 10 bit (FC-1) version of the comma character.

Abbr.	Link/Loop	Primitive Sequence	FC-2 Ordered Set
			FC-1 Ordered Set
NOS	both	Not_Operational	0xBC-0x55-0xBF-0x45
			K28.5-D21.2-D31.5-D5.2
OLS	both	Offline	0xBC-0x35-0x8A-0x55
			K28.5-D21.1-D10.4-D21.2
LR	both	Link Reset	0xBC-0x49-0xBF-0x49
			K28.5-D9.2-D31.5-D9.2
LRR	both	Link_Reset_Response	0xBC-0x35-0xBF-0x49
			K28.5-D21.1-D31.5-D9.2
LIP(F7, F7)	Loop	Loop Initialization – F7, F7	0xBC-0x15-0xF7-0xF7
			K28.5-D21.0-D23.7-D23.7
LIP(F8, F7)	Loop	Loop Initialization – F8, F7	0xBC-0x15-0xF8-0xF7
			K28.5-D21.0-D24.7-D23.7
LIP(F7, x)	Loop	Loop Initialization – F7, x	0xBC-0x15-0xF7-AL_PS
			K28.5-D21.0-D23.7-AL_PS
LIP(F8, x)	Loop	Loop Initialization – F8, x	0xBC-0x15-0xF8-AL_PS
			K28.5-D21.0-D24.7-AL_PS
LIPyx	Loop	Loop Initialization – reset	0xBC-0x15-AL_PD-AL_PS
			K28.5-D21.0-AL_PD-AL_PS
LIPfx	Loop	Loop Initialization – reset all	0xBC-0x15-0xFF-AL_PS
			K28.5-D21.0-D31.7-AL_PS
LIPba	Loop	Loop Initialization – reserved	0xBC-0x15-b-a
			K28.5-D21.0-b-a
LPByx	Loop	Loop Port Bypass	0xBC-0x09-AL_PD-AL_PS
			K28.5-D9.0-AL_PD-AL_PS
LPBfx	Loop	Loop Port Bypass all	0xBC-0x09-0xFF-AL_PS
			K28.5-D9.0-D31.7-AL_PS
LPEyx	Loop	Loop Port Enable	0xBC-0x05-AL_PD-AL_PS
			K28.5-D5.0-AL_PD-AL_PS
LPEfx	Loop	Loop Port Enable all	0xBC-0x05-0xFF-AL_PS
			K28.5-D5.0-D31.7-AL_PS

Appendix D

Fibre Channel Frame Delimiters:

This chart illustrates the composition of Frame Delimiters in both 32 bit (FC-2) and 40 bit (FC-1) formats. It is important to note that they all begin with either the 8 bit (FC-2) or 10 bit (FC-1) version of the comma character. Additionally, it is good to consider the differences between those that are transmitted in the middle of a sequence, and those that are not.

Abbr.	Delimiter Function	FC-2 Ordered Set	FC-1 Ordered Set
SOFc1	SOF Connect Class 1	0xBC-0xB5-0x17-0x17	K28.5-D21.5-D23.0-D23.0
SOFi1	SOF Initiate Class 1	0xBC-0xB5-0x57-0x57	K28.5-D21.5-D23.2-D23.2
SOFn1	SOF Normal Class 1	0xBC-0xB5-0x37-0x37	K28.5-D21.5-D23.1-D23.1
SOFi2	SOF Initiate Class 2	0xBC-0xB5-0x55-0x55	K28.5-D21.5-D21.2-D21.2
SOFn2	SOF Normal Class 2	0xBC-0xB5-0x35-0x35	K28.5-D21.5-D21.1-D21.1
SOFi3	SOF Initiate Class 3	0xBC-0xB5-0x56-0x56	K28.5-D21.5-D22.2-D22.2
SOFn3	SOF Normal Class 3	0xBC-0xB5-0x36-0x36	K28.5-D21.5-D22.1-D22.1
SOFc4	SOF Connect Class 4	0xBC-0xB5-0x19-0x19	K28.5-D21.5-D25.0-D25.0
SOFi4	SOF Initiate Class 4	0xBC-0xB5-0x59-0x59	K28.5-D21.5-D25.2-D25.2
SOFn4	SOF Normal Class 4	0xBC-0xB5-0x39-0x39	K28.5-D21.5-D25.1-D25.1
SOFf	SOF Fabric	0xBC-0xB5-0x58-0x58	K28.5-D21.5-D24.2-D24.2
EOFt	EOF Terminate	0xBC-0x95-0x75-0x75/ 0xBC-0xB5-0x75-0x75	K28.5-D21.4-D21.3-D21.3/ K28.5-D21.5-D21.3-D21.3
EOFdt	EOF Disconnect-Terminate Class 1 or EOF Deactivate-Terminate Class 4	0xBC-0x95-0x95-0x95/ 0xBC-0xB5-0x95-0x95	K28.5-D21.4-D21.4-D21.4/ K28.5-D21.5-D21.4-D21.4
EOFa	EOF Abort	0xBC-0x95-0xF5-0xF5/ 0xBC-0xB5-0xF5-0xF5	K28.5-D21.4-D21.7-D21.7/ K28.5-D21.5-D21.7-D21.7

EOFn	EOF Normal	0xBC-0x95-0xD5-0xD5/ 0xBC-0xB5-0xD5-0xD5
		K28.5-D21.4-D21.6-D21.6/ K28.5-D21.5-D21.6-D21.6
EOFni	EOF Normal-Invalid	0xBC-0x8A-0xD5-0xD5/ 0xBC-0xAA-0xD5-0xD5
		K28.5-D10.4-D21.6-D21.6/ K28.5-D10.5-D21.6-D21.6
EOFdti	EOF Disconnect-Terminate-Invalid Class 1 or EOF Deactivate-Terminate-Invalid Class 4	0xBC-0x8A-0x95-0x95/ 0xBC-0xAA-0x95-0x95
		K28.5-D10.4-D21.4-D21.4/ K28.5-D10.5-D21.4-D21.4
EOFrt	EOF Remove-Terminate Class 4	K28.5-D10.4-D21.4-D21.4/ K28.5-D10.5-D21.4-D21.4
EOFrti	EOF Remove-Terminate-Invalid Class 4	K28.5-D21.4-D25.4-D25.4/ K28.5-D21.5-D25.4-D25.4
		K28.5-D10.4-D25.4-D25.4/ K28.5-D10.5-D25.4-D25.4

Appendix E

Fibre Channel Port Types:

This chart defines some typical port categories, in order to provide the reader with a basic introduction to the types of Fibre Channel devices that exist. It is important to note that although generic frames are used by all implementations discussed in this document, they are still referred to with regard to their primary roles as Fx_Ports and Nx_Ports.

PORT (FC_Port)	FUNCTION
B_Port	A Fabric inter-element port used to connect bridge devices with the E_Ports on a switch.
E_Port	A Fabric extension port which connects to either an E_Port or a B_Port to create an Inter-Switch link.
F_Port	The switch Link Control Facility that attaches to an N_Port through a link.
FL_Port	An F_Port which is capable of functioning on an Arbitrated Loop.
Fx_Port	Port capable of behaving as an F_Port or an FL_Port.
G_Port	Generic Fabric Port which can function either as an E_Port or an F_Port.
GL_Port	A G_Port which is capable of functioning on an Arbitrated Loop.
L_Port	A port which is capable of functioning on an Arbitrated Loop.
N_Port	The end device Link Control Facility that attaches to an F_Port or an N_Port through a link.
NL_Port	An N_Port which is capable of functioning on an Arbitrated Loop.
Nx_Port	Port capable of behaving as an N_Port or an NL_Port.

Appendix F

iFCP Definitions:

Address Transparent Mode: An iFCP gateway is operating in this mode when the scope of a Fibre Channel address is fabric-wide, and derived from the Domain Id issued by the iSNS server. In this case, it is not necessary for the gateway to replace the address with the N_Port alias from the session descriptor, since it is already unique. Support for this is optional.

Bounded iFCP fabric: Collection of iFCP gateways that operate in address transparent mode.

Address Translation Mode: An iFCP gateway is operating in this mode when the scope of a Fibre Channel address is only local to its gateway region. Every gateway must replace the destination address of any outgoing iFCP frame with the N_Port alias in the session descriptor for the remote N_Port. The N_Port alias has been assigned to the remote N_Port by each gateway in its CBIND request or response. Support for this is mandatory.

Unbounded iFCP Fabric: Collection of iFCP gateways that operate in address translation mode.

Global Server (in this implementation, also the initiator module): During power up, the global server is required to immediately register with the iSNS service as the global server for its discovery domain, as long as another global server has not already registered as the global server for that particular domain. The global server then initiates iFCP sessions with each local server in this domain. Whenever a new local server (in this implementation, the target module) registers, the global server is notified of this change through the iSNS server, and it must then initiate an iFCP session with the local server.

Local Server (in this implementation, also the target module): During power up, the local server is required to invoke the iSNS service to register in the broadcast discovery domain. The local server must then wait for the global server (in this implementation, the initiator module) to establish an iFCP session with the new local server.

Appendix G

iFCP Addressing Mode Examples:

Note: all identifiers are completely arbitrary

A. Address Transparent Mode (optional):

1. Initiator and Target register with iSNS server
2. iSNS server assigns N_Port ID to Initiator and Target
3. Initiator queries iSNS server for Target
4. Initiator transmits CBIND request
5. Target queries iSNS server for Initiator
6. Target transmits CBIND response with status of "Success"
7. Target creates Session Descriptor for Initiator
   Initiator Session Descriptor:
   TCP Connection Context:
      IFCP Portal Addr: 192.1.1.101/3420
      Interface: eth1
      Connection ID: 0
   Local N_Port ID: 0x456789
   Remote N_Port ID: 0x123456
   Remote N_Port alias: (null)

8. Initiator receives CBIND response from Target
9. Initiator creates session descriptor for Target
   Target Session Descriptor:
   TCP Connection context:
      IFCP Portal Addr: 192.1.1.100/3420
      Interface: eth1
      Connection ID: 0
   Local N_Port ID: 0x123456
   Remote N_Port ID: 0x456789
   Remote N_Port alias: (null)

10. All frames transmitted and received retain the same S_ID and D_ID

B. Address Translation Mode (mandatory):

1. Initiator and Target register with iSNS server
2. Initiator queries iSNS server for Target
3. Initiator creates Remote N_Port Descriptor for Target
   Target Remote N_Port Descriptor:
   WWN: 0x7DDDDDDD 0xDDDDDDDD; provided by iSNS server
   IFCP Portal Addr: 192.1.1.100/3420; provided by iSNS server
   N_Port ID: 0x456789; provided by iSNS server

N_Port Alias: 0xFEDCBA; assigned by Initiator

4. Initiator transmits CBIND request
5. Target queries iSNS server for Initiator
6. Target creates Remote N_Port Descriptor for Initiator
   Initiator Remote N_Port Descriptor:
   WWN: 0x10000000 0x00000001; provided by CBIND request
   IFCP Portal Addr: 192.1.1.101/3420; provided by iSNS server
   N_Port ID: 0x123456; provided by iSNS server
   N_Port Alias: 0xABCDEF; assigned by Target

7. Target transmits CBIND response with status of "Success"
8. Target creates Session Descriptor for Initiator
   Initiator Session Descriptor:
   TCP Connection Context:
      IFCP Portal Addr: 192.1.1.101/3420
      Interface: eth1
      Connection ID: 0
   Local N_Port ID: 0x456789
   Remote N_Port ID: 0x123456
   Remote N_Port Alias: 0xABCDEF

9. Initiator receives CBIND response from Target
10. Initiator creates Session Descriptor for Target
   Target Session Descriptor:
   TCP Connection context:
      IFCP Portal Addr: 192.1.1.100/3420
      Interface: eth1
      Connection ID: 0
   Local N_Port ID: 0x123456
   Remote N_Port ID: 0x456789
   Remote N_Port alias: 0xFEDCBA

11.
   a) Mode 1:

      i. iFCP initiator transmits an encapsulated FC frame to iFCP target, with:
         D_ID: 0x456789
         S_ID: 0x000001
      ii. iFCP target replaces S_ID with Initiator alias:
         D_ID: 0x456789
         S_ID: 0xABCDEF
      iii. iFCP target recalculates FC CRC and transmits frame to locally attached Fibre Channel device

b) Mode 2:

i. Fibre Channel device transmits response to iFCP target, with:
D_ID: 0xABCDEF
S_ID: 0x456789

ii. iFCP Target replaces D_ID field, recalculates FC CRC, and transmits encapsulated frame to iFCP Initiator, with:
D_ID: 0x000002
S_ID: 0x456789

iii. iFCP Initiator replaces D_ID field with local N_Port ID:
D_ID: 0x123456
S_ID: 0x456789

iv. iFCP initiator recalculates FC CRC and transmits frame to locally attached Fibre Channel device

c) Mode 3:

Scenario I:

i. iFCP initiator transmits an encapsulated Third Party Process Logout (TPRLO) frame to iFCP target, with:
D_ID: 0x000003
S_ID: 0x123456
and in the iFCP extension at the end of the Fibre Channel payload:
WWN: 0x22222222 0x33333333

ii. iFCP target queries iSNS server, discovering that the WWN belongs to a locally attached Fibre Channel device with N_Port ID 0x456712

iii. iFCP target replaces D_ID with N_Port ID:
D_ID: 0x456712
S_ID: 0x123456

iv. iFCP target recalculates CRC and transmits frame to locally attached Fibre Channel device

Scenario II:

i. iFCP initiator transmits an encapsulated Third Party Process Logout (TPRLO) frame to iFCP target, with:
D_ID: 0x000003
S_ID: 0x123456
and in the iFCP extension at the end of the Fibre Channel payload:
WWN: 0x44444444 0x55555555

ii. iFCP target queries iSNS server, discovering that the WWN belongs to a remote iFCP gateway which is no longer a member of this discovery domain (or is otherwise unreachable)

iii. iFCP target transmits an encapsulated Fibre Channel Link Service Reject (LS_RJT) frame with reason code 0x07 (Protocol Error) and Reason Explanation 1F (Invalid N_Port Identifier) to iFCP initiator

Appendix H

<u>Special Link Service Fibre Channel frames defined by iFCP</u>:

ABTX	Abort Exchange
ADISC	Discover Address
ADISC ACC	Discover Address Accept
FARP-REPLY	Fibre Channel Address Resolution Protocol Reply
FARP- REQ	Fibre Channel Address Resolution Protocol Request
LOGO	N_PORT Logout
PLOGI	Port Login
REC	Read Exchange Concise
REC ACC	Read Exchange Concise Accept
FCP REC	FCP Read Exchange Concise
FCP REC ACC	FCP Read Exchange Concise Accept
RES	Read Exchange Status Block
RES ACC	Read Exchange Status Block Accept
RLS	Read Link Error Status Block
RRQ	Reinstate Recovery Qualifier
RSI	Request Sequence Initiative
RSS	Read Sequence Status Block
SRL	Scan Remote Loop
TPRLO	Third Party Process Logout
TPRLO ACC	Third Party Process Logout Accept

Appendix I

RSCN (as defined by FC-FS):

12.3.2.22.2 RSCNs issued by the Fabric Controller

The Fabric Controller shall issue an RSCN Request to all registered Nx_Ports for an affected Nx_Port when the fabric detects an event. The Fabric Controller shall ensure that any Fabric-provided resources (e.g., the Name Service) have received updates to reflect changes caused by the event, prior to issuing the RSCN for the event. An event may include any of the following:

a) an implicit fabric Logout of the affected Nx_Port, including Loss-of-Signal, NOS, and OLS, or when the fabric receives a FLOGI from a port that had already completed FLOGI;
b) a loop initialization of the affected L_Port, and the L_bit was set in the LISA Sequence;
c) a fabric Login from an affected Nx_Port not previously logged in;
d) the fabric path between the affected Nx_Port and any other Nx_Port has changed (e.g., a change to the fabric routing tables that affects the ability of the fabric to deliver frames in order, or an E_Port initialization or failure);
e) any other fabric-detected state change of the affected Nx_Port;
f) the affected Nx_Port issues an RSCN Request to the Fabric Controller. A registered Nx_Port that receives an RSCN Request may perform any operation to determine the nature of the state change. These operations include the PDISC ELS, the ADISC ELS, a query to the Name Service, or a ULP query. The fabric may accumulate affected Nx_Port addresses for subsequent delivery to reduce the volume of RSCN traffic.

Appendix J

Relevant IFC-4 I-TECH/Eagle software library functions:

1. frm_user(ui32 * buffer, ui32 length, ui32 payloadtype, ui32 offset);
   Function: Transmits frame in buffer of size length + 3 in words (the length parameter does not include SOF, EOF or CRC) over Fibre Channel.

2. get_frame(ui8 wait_flag);
   Function: Polls for the presence of an incoming Fibre Channel frame if the wait_flag is set to 0, or to both wait for and process the incoming frame if the wait_flag is set to 1.

3. ioto(int value);
   Function: Sets the timeout value in seconds for the get_frame function, if the wait_flag has been set to 1.

4. get_frame_info(char * info_category, ui16 first_word, ui16 word_count, ui32 * array_address);
   Function: Retrieves additional information about an incoming frame not provided by the get_frame function, such as the SOF or EOF value.

5. prm_open(ui16 id, OPENTYPE type);
   Function: Opens a Fibre Channel device on the Arbitrated Loop with identifier id.

6. prm_user(ui16 char0, ui16 char1, ui16 char2, ui16 char3, ui32 count);
   Function: Transmits the first four parameters as a user-defined ordered set that is repeated count times.

7. prm_rready();
   prm_close();
   Function: Transmits the R_RDY and CLS Ordered Sets on an Arbitrated Loop, respectively.

8. set_xmit_mode(int value);
   set_R_RDY_mode(int value);
   Function: Establishes and detects buffer credit settings for the generator and its locally attached Fibre Channel device.

9. set_state(ui8 stage, ui16 value);
   Function: Sets the initialization state of the loop to one of thee stages. In this implementation, two stages were used. The first stage initiates the transmission of LIP(F7) to transition the loop into OPEN-INIT-START. The second initiates the transmission of ARB(F0) to transition the loop into MASTER-START. The

value parameter indicates which type of LIP is being used in the first scenario, and it is not used in the second.

10. set_topology(int value);
Function: Establishes the Fibre Channel topology as either a loop or a link.

11. get_AL_PA();
xlate_to_id(ui8 value);
Function: Retrieves the target AL_PA from the initialization, and translates the AL_PA into the N_Port ID format recognized by the generator, respectively.

12. get_loop_state();
Function: Retrieves a value that corresponds to the current state of the arbitrated loop.

Appendix K

ifcp initiator host template structure:

```
#define IFCP_INITIATOR {\
            proc_info:                            ifcp_initiator_proc_info,\
            name:                                 OUR_NAME,\
            detect:          ifcp_initiator_detect,\
            release:         ifcp_initiator_release,\
            info:            NULL,\
            ioctl:                                NULL,\
            queuecommand:        ifcp_initiator_queuecommand,\
            eh_strategy_handler:      NULL,\
            eh_abort_handler:    ifcp_initiator_abort,\
            eh_device_reset_handler:ifcp_initiator_reset,\
            eh_bus_reset_handler:     NULL,\
            eh_host_reset_handler:  NULL,\
            bios_param:          NULL, \
            can_queue:           1,\
            this_id:         -1, \
            sg_tablesize:        MAX_IOV_SLOTS-4,\
            max_sectors:                   4*(MAX_IOV_SLOTS-4),\
            cmd_per_lun:          MAX_COMMANDS,\
            present:          0,\
            unchecked_isa_dma:    0,\
            use_clustering:       ENABLE_CLUSTERING,\
            use_new_eh_code:               1,\
            emulated:                             0,\
            proc_name:                            "ifcp_initiator"\
            }

static Scsi_Host_Template driver_template = IFCP_INITIATOR;
```

Appendix L

Buffer-to-buffer credit definitions:

As defined by FC-FS:

18.5.5 BB_Credit management
BB_Credit management involves an FC_Port receiving the BB_Credit value from the FC_Port to which it is directly attached. Fabric Login is used to accomplish this. The common Service Parameters interchanged during Fabric Login provide these values (see 15.5.2). The transmitting FC_Port is responsible to manage BB_Credit_CNT with BB_Credit as its upper bound.

18.5.6 Buffer-to-buffer flow control model
The buffer-to-buffer flow control model is illustrated in figure 25. The model includes flow control parameters, control variables for a Class 2, Class 3, or Class 1 and 6/SOFc1 frame and R_RDY as its response, and the resources for Connectionless service. All possible responses to a Class 2 or Class 3 Data frame are illustrated. Each FC_Port provides a number of receive buffers for Connectionless service. Each Nx_Port obtains the allocation of receive buffers from the Fx_Port (or Nx_Port in case of point-to-point topology) to which it is attached, as BB_Credit. Each Fx_Port obtains the allocation of receive buffers from the Nx_Port to which it is attached, as total B_Credit for Connectionless service.

As defined by FC-AL-2:

The L_Port which receives OPNy shall obey the following rules for transmitting R_RDYs:
NOTE — The number of R_RDYs which the L_Port transmits before the first frame is a balance between delaying the transmission of the first frame and delaying receiving frames.
— if the opened BB_Credit equals zero (0), the L_Port shall transmit one R_RDY for each currently available receive buffer.
— if the opened BB_Credit equals zero (0), the L_Port may transmit CLS if there are no available receive buffers.
— if the opened BB_Credit is greater than zero (0), the L_Port shall transmit one R_RDY for each BB_Credit which this L_Port advertised plus one R_RDY for each additional available receive buffer.
— If CLS is received before all R_RDYs have been transmitted, the remaining R_RDYs are not required to be transmitted in the Loop circuit. The L_Port shall initialize the open BB_Credit to zero (0). If the L_Port can determine the open BB_Credit, it may transmit the number of frames specified by the open BB_Credit. If the L_Port transmitted frames based on the open BB_Credit, it shall discard one received R_RDY for each of these frames sent. When the number of discarded R_RDYs equals the open BB_Credit, the L_Port shall use Available_BB_Credit management.

Appendix M

Some important data structures:

```
struct nport
    {
    /* our header */
    ui32 my_did;
    ui32 my_sid;
    ui32 my_f_ctl;
    ui8 my_seq_id;
    ui8 my_df_ctl;
    ui16 my_seq_cnt;
    ui16 my_ox_id;
    ui16 my_rx_id;
    /* their header */
    ui32 their_did;
    ui32 their_sid;
    ui32 their_f_ctl;
    ui8 their_seq_id;
    ui8 their_df_ctl;
    ui16 their_seq_cnt;
    ui32 their_ox_id;
    ui32 their_rx_id;
    /* other addresses */
    ui32 their_port[PORT_NAME];
    ui32 their_node[PORT_NAME];
    ui32 their_ip[PORT_NAME];
    ui32 our_port[PORT_NAME];
    ui32 our_node[PORT_NAME];
    ui32 our_ip[PORT_NAME];
    /* address translation names */
    ui32 my_port_id;
    ui32 their_port_id;
    /* other information */
    ui32 block_size;
    ui32 last_sent;
    ui8 initiator;
    ui8 frameno;
    ui16 flow_control;
    };
```

```
struct nport_maker
      {
            ui8  target_num;
            ui32 user_info;
            ui32 prli_type;
            ui32 snd_frm_len;
            ui32 * target_names;
            ui32 * target_wwns;
            ui32 * snd_frm;
            ui32 last_sent;
            ui32 domain_id;
            struct scn_list scn;
            struct nport np;
            struct prlogin prli;
            struct prlogout prlo;
            struct lestatus les;
            struct ssblock ssb;
            struct descriptors des;
      };

class NPortFrameMaker {
  public:
      NPortFrameMaker(void* s, ui8 initiator, ui8 flow_control);
      void assign_rxid();
      void reset_rxid();
      void make_fctl(ui8 einitiator, ui8 sinitiator);
      void increment_fields(void* port);
      void make_header(ui32* buffer, void* port, ui8 acc, ui8 fcp, ui32 word_three, ui8 loop);
      void make_rb_header(ui32* buffer, void* port, ui8 rctl, ui8 loop);
      void print_frame(ui32* buffer, ui16 length, char name[]);
      ui8  make_cbind_request(ui32 * wwn, ui32 name);
      ui8 make_cbind_response(ui32* wwn1, ui32* wwn2, ui32 name);
      ui8 make_unbind_request(ui32 * wwn);
      ui8 make_unbind_response(ui32 * wwn);
      ui8 make_ltest(ui32 * wwn, ui32 * wwn_hold);
      ui8  make_rjt(ui8 reason, ui8 loop);
      ui8  make_fbsy(ui8 reason, ui8 loop);
      ui8  make_pbsy(ui8 reason, ui8 loop);
      ui8  make_fan(ui8 loop);
      ui8  make_ack1(ui8 loop);
      ui8  make_flogi_acc(ui8 loop);
      ui8  make_prli_acc(ui8 loop);
      ui8  make_prlo(ui8 loop);
      ui8  make_prlo_acc(ui8 loop);
      ui8  make_abtx(ui8 loop);
      ui8  make_abtx_acc(ui8 loop);
      ui8  make_adisc(ui8 loop);
      ui8  make_adisc_acc(ui8 loop);
      ui8  make_fdisc(ui8 loop);
      ui8  make_fdisc_acc(ui8 loop);
      ui8  make_farp_reply(ui8 loop);
      ui8  make_farp_request(ui8 loop);
      ui8  make_plogi(ui8 loop);
      ui8  make_plogi_acc(ui8 loop);
      ui8  make_logo(ui8 loop);
```

```
        ui8  make_logo_acc(ui8 loop);
        ui8  make_rec(ui8 originator, ui8 loop);
        ui8  make_rec_acc(ui8 ended, ui8 normal, ui8 loop);
        ui8  make_rtv(ui8 loop);
        ui8  make_rtv_acc(ui8 loop);
        ui8  make_res(ui8 loop) ;
        ui8  make_res_acc(ui8 requested_action, ui8 ended, ui8 normal, ui8 loop);
        ui8  make_rls(ui8 loop);
        ui8  make_rls_acc(ui8 loop);
        ui8  make_rss(ui8 loop);
        ui8  make_rss_acc(ui8 loop);
        ui8  make_rrq(ui8 loop);
        ui8  make_rrq_acc(ui8 loop);
        ui8  make_rsi(ui8 loop);
        ui8  make_rsi_acc(ui8 loop);
        ui8  make_srl(ui8 flag, ui32 fl_port, ui8 loop);
        ui8  make_srl_acc(ui8 flag, ui32 fl_port, ui8 loop);
        ui8  make_tprlo(ui8 loop);
        ui8  make_tprlo_acc(ui8 loop);
        ui8  make_rft_id_acc(ui8 loop);
        ui8  make_rpn_id_acc(ui8 loop);
        ui8  make_rnn_id_acc(ui8 loop);
        ui8  make_rff_id_acc(ui8 loop);
        ui8  make_ga_nxt_acc(ui8 loop);
        ui8  make_gid_ft_acc(ui8 loop);
        ui8  make_gnn_ft_acc(ui8 loop);
        ui8  make_gpn_id_acc(ui8 loop);
        ui8  make_scr_acc(ui8 loop);
        ui8  make_rscn(ui8 loop);
    private:
        void* nport;
        };

class SocketMaker {
    public:
        SocketMaker();
        SOCKET return_fd();
        SOCKET initiate_connection(const char * address);
        ui8 receive_connection();
        ui32 send_socket(NPortMaker * npm, Descriptors * des, int * type);
        ui32 recv_nonblock(ui8 fd, char * buf);
        void close_connection(ui8 fd);
        ui8 send_frame(ui32* buffer, ui8 length);
        void print_frame(ui32* buffer, ui8 length, char name[]);
        int encap_frame(NPortMaker * npm, Descriptors * des, ui8 type);
        int verify_crc(ui32 * buf, ui8 length);
        ui32 * unpack_frame(NPortMaker * npm, ui32 * buf, ui32 size);
        };
```

```
class LoopInitMaker {
  public:
    LoopInitMaker();
    ui8 loopinit();
    ui8 loopinit_one();
    ui8 loopinit_two();
    ui8 loopinit(ui32 userwwn1, ui32 userwwn2);
    void send_lism();
    ui8 receive_lism();
    ui8 init_frames_slave(ui8 us_two);
    ui8 init_frames_master(ui8 which, ui8 us_two);
    void late_frames_slave(ui8 more_than_one, ui8 us_two);
    void late_frames_master(ui8 more_than_one, ui8 us_two);
  private:
    bool lirp_sent;
    ui32 wwn1;
    ui32 wwn2;
    ui32 my_alpa;
    ui32 their_alpa;
    };

class NLPortMaker {
  public:
    NPortMaker * nport;
    NLPortMaker();
    NLPortMaker(NPortMaker * nlp);
    void copy_port(NPortMaker * npm);
    ui32 get_alpa();
    void loop_initialize();
    void loop_plogi();
    void loop_prli();
    int receive_frame(ui8 get_oxid, ui8 check_last, ui8 fibre_channel, ui32 * buf);
    int receive_frame(ui8 get_oxid, ui8 check_last);
    int send_frame(ui32 * buffer, ui32 size, ui8 no_opn, ui8 no_close);
    void resend_frame(ui32 * buffer, ui32 size, ui8 no_opn, ui8 no_close);
    void send_frame(ui8 type);
    void send_frame(ui8 type, ui32 mode);
    void send_frame(ui8 type, ui32 mode1, ui32 mode2);
  private:
    friend class NPortFrameMaker;
};
```

Appendix N

Acronyms:

EOF: End of Frame frame delimiter
LIFA: Loop Initialization Fabric Assigned loop initialization sequence
LIHA: Loop Initialization Hard Assigned loop initialization sequence
LIP: Loop Initialization Primitive Sequence
LIPA: Loop Initialization Previously Assigned loop initialization sequence
LISA: Loop Initialization Soft Assigned loop initialization sequence
LISM: Loop Initialization Select Master loop initialization sequence
RAID: Redundant Array of Independent Disks
SCSI: Small Computer Systems Interface
SOF: Start of Frame frame delimiter