CIFS Explained

Software solutions for applications and appliances

www.codefx.com
ph: (619) 269-4274
fax: (619) 269-4274
4545 Campus Ave. Suite #9
San Diego, CA 92116

Table of Contents

CIFS Overview:

What is CIFS?

The Common Internet File System (CIFS), also known as Server Message Block (SMB), is a network protocol whose most common use is sharing files on a Local Area Network (LAN). The protocol allows a client to manipulate files just as if they were on the local computer. Operations such as read, write, create, delete, and rename are all supported – the only difference being that the files are *not* on the local computer and are actually on a remote server.

The CIFS protocol works by sending packets from the client to the server. Each packet is typically a basic request of some kind, such as open file, close file, or read file. The server then receives the packet, checks to see if the request is legal, verifies the client has the appropriate file permissions, and finally executes the request and returns a response packet to the client. The client then parses the response packet and can determine whether or not the initial request was successful.

CIFS is a fairly high-level network protocol. In the OSI model, it is probably best described at the Application/Presentation layer. This means CIFS relies on other protocols for transport. The most common protocol used for reliable transport is NetBIOS over TCP (NBT), which will be described in the NetBIOS chapter below. Other protocols have been used for the transport layer, however with the enormous popularity of the Internet, NBT has become the de-facto standard.

Although file sharing is CIFS's primary purpose, there are other functions that CIFS is commonly associated with. Most CIFS implementations are also capable of determining other CIFS servers on the network (browsing), printing, and even complicated authentication techniques. None of these subjects will be discussed in this document however, only the basics of CIFS file operations.

Where is CIFS?

The CIFS protocol is most commonly used with Microsoft operating systems. Windows For Workgroups was the first Microsoft operating system to use CIFS, and each Microsoft operating system since then has been able to function as both a CIFS server and client. Microsoft operating systems use CIFS for remote file operations (typically mapping network drives), browsing (via the Network Neighborhood icon), authentication (NT and Windows 2000), and remote printer services. It would be fair to say the core of native Microsoft networking is built around its CIFS services.

Because of Microsoft's large corporate and home user base, the CIFS protocol is found virtually everywhere. Flavors of the Unix operating system also implement a CIFS client/server via the Samba program. Apple computers also have CIFS clients and servers available, which might make CIFS the most common protocol for file sharing available.

CIFS History:

In 1984, IBM wrote an application programmer interface (API) that allowed basic network communication between hosts on a small subnet. The API, however, required a transport level protocol to actually send and receive data. The following year, IBM released a transport level protocol that was capable of making the NetBIOS API come to life. The API and the transport protocol were merged into one entity and called the NetBIOS Enhanced User Interface, or NetBEUI.

At the time, other transport protocols were in common use, and soon, the NetBIOS API was implemented using various other transport protocols such as DECnet, IPX/SPX, and TCP/IP. The API became quite popular.

Soon after, Microsoft and Intel created the first rendition of the SMB/CIFS file sharing protocol titled "Core Protocol". Microsoft and Intel chose to use the aforementioned NetBIOS API for the delivery of the upper layer CIFS packets. Hence, CIFS using NetBIOS over TCP became the standard network file sharing mechanism for Microsoft operating systems.

Many features have been added to the initial Core Protocol over time. Presently, most Windows clients support at least 6 different variations of the CIFS protocol, with each version typically containing a few more features than the last. There are at least 100 different CIFS operations to date, and the list keeps on growing. The modestly robust feature set includes:

- File access
- File and record locking
- Safe file caching
- File change notification
- Protocol negotiation
- Extended file attribute handling
- Batched requests
- Unicode support

The CIFS protocol, however, is definitely showing signs of age. The protocol's feature set has been extended several times over the last 13 years, and repercussions are becoming evident. There are multiple CIFS packets that accomplish the same task, and many of the CIFS packets have undocumented options. The Internet Engineering Task Force (IETF) and the Storage Networking Industry Association (SNIA) are trying hard to remedy this dilemma. They are both working towards creating the CIFS1.0 specification, which lists only a subset of the current CIFS operations that need to be supported into the future. The specification also tries to define more clearly the various packet options. There is much work to be done, but this is definitely a good start.

NetBIOS:

This section will focus on the NetBIOS functions that are commonly used by upper layer CIFS services. As mentioned above, NetBIOS runs over many transport level protocols, but for modern day implementations, the TCP/IP protocol suite is by the far the most common transport protocol used.

The techniques and concepts for running NetBIOS over TCP/UDP (aka NBT) were documented in RFC1001 and RFC1002 in 1987. These two documents are very complete; RFC1001 covers the concepts and methods, while RFC1002 covers the detailed specifications.

Within these documents three main services are described: name service, session service, and datagram service. Each of these services and their relation to typical CIFS implementations will be discussed below. Included at the end of this section is a brief description of potential future NetBIOS changes.

Name Service:

NetBIOS names are human readable names that are assigned to computers on a network. These names are most commonly seen in Windows operating systems in the Network Neighborhood. NetBIOS naming serves the same purpose as the DNS system in the TCP/IP world; it allows humans to call computers by names, and has a system for mapping the readable name into a transport address, such as an IP. However, the methods for registering a name with an IP, and resolving a name into an IP are very different for NetBIOS.

To better understand the workings of NetBIOS name operations this section first covers major properties of NetBIOS naming conventions and then describes brief outlines of common NetBIOS naming procedures.

NetBIOS Naming Properties:

Broadcast and/or server based. NetBIOS name registrations and lookups can all be accomplished via broadcasting to the local area network and/or by using a central NetBIOS name server (NBNS or WINS[1]). Typically, with DNS, only a centralized name server can be used. Because of the "and/or" option, all NetBIOS computers on a network must be configured by the administrator to use either:

- Broadcast only (b-node)
- NBNS only (p-node)
- Broadcast first and NBNS only if no response to broadcast (m-node)
- NBNS, and then broadcast if server unresponsive (h-node)

[1] Microsoft refers to its NetBIOS name server implementation as WINS.

Because of these options, there are two descriptions for every naming activity, one describing the broadcast method, the other describing the server method.

Dynamic registration. DNS and NetBIOS naming also differ on how to create the association between a readable name and an IP address. With DNS, an administrator typically has to add the DNS entry to the server once, and the DNS name and the IP are statically held forever. With NetBIOS, when a computer boots, it registers its name/ip combination dynamically. The NetBIOS computer can also un-register its name whenever it no longer requires the name.

Name syntax. NetBIOS names must conform to a rigid set of rules. The list below describes the major properties that the names must adhere to.

1. A registered NetBIOS name can refer to either a single workstation (unique name), or can refer to multiple nodes that are all part of a workgroup (group name).
2. NetBIOS names exist in a flat name space with no hierarchical format. This means there are no “dots” (i.e. the ‘.’ character) in NetBIOS names as in DNS names[2]. Because of this flat name space, any unique name (as discussed above) can only be registered by one computer.
3. NetBIOS names must only consist of characters from the following ranges, a-z, A-Z, 0-9, or one of the following characters ! @ # $ % ^ & ( ) - ' { } . ~
4. NetBIOS names use a maximum of 15 characters to describe the resource, and the 16th byte refers to the resource type. The resource type byte indicates what properties the registering computer has[3].

NetBIOS Common Procedure Outlines:

The two most common NetBIOS name service procedures for a computer on a network are name registration, and name query. Name registration associates a NetBIOS name with an IP, while name query determines the IP address associated with a given name. Both procedures are described below for b-nodes and for p-nodes. For m-nodes and h-nodes, these procedures would be merged together, sometimes trying the broadcast method first, and sometimes trying the NBNS method first.

Name registration (b-node): The workstation wishing to register a name builds a NetBIOS name registration packet (as defined in RFC1002) and then broadcasts the packet to the subnet via the UDP protocol on port 137. This packet contains the name

[2] The exception to this rule is that NetBIOS names can contain a scope-id, which in essence tacks on a valid domain name to the end of the NetBIOS name. This option is so rarely used though, that it is almost irrelevant.

[3] The resource type byte is comparable to DNS names such as ftp.codefx.com, or www.codefx.com; they give insight into the machines’ capabilities. Example resource types: standard workstation service (0x00), winpopup capable (0x03), domain master browser (0x1B), master browser name (0x1D), and fileserver (0x20).

that the workstation wishes to register and the IP address of that workstation. The workstation then repeats this procedure three times, waiting 250 milliseconds between broadcasts. During this time, the workstation is waiting to see if any other workstation sends a defense packet back. This packet would be sent from any other workstation that has already registered this name as its own (a defense should not be sent on a group name as it is legal for multiple workstations to register the same group name). If no defense response is received, the workstation has successfully registered the name.

Name registration (p-node): The p-node also builds a NetBIOS name registration packet (as defined in RFC1002), but instead *unicasts* the packet directly to the NBNS server via the UDP protocol on port 137. The NBNS then searches its internal name database, checking for other workstations that have already registered an identical unique name. If the name is already registered, the NBNS sends back a negative name registration response[4]. If no workstation has already registered the name, the server sends a positive name registration response and the workstation has successfully registered the name.

Name query (b-node): The workstation wishing to obtain an IP address from a NetBIOS name first builds a name query request (as defined in RFC1002), and then broadcasts the packet to the subnet via the UDP protocol on port 137. The request packet contains the NetBIOS name that the workstation is wishing to query. The workstation then repeats this process 3 times, waiting 5 seconds between each retransmit. During this time, the workstation either receives a positive name query response from the computer that owns the name, or it receives nothing. If it receives a positive name query response, the packet will contain the IP address of the queried NetBIOS name.

Name query (p-node): The p-node query also builds a name query request, but again *unicasts* the packet directly to the NBNS server via the UDP protocol on port 137. The NBNS then searches its internal database for a matching NetBIOS name. If a match is found, the NBNS sends a positive name query response back to the workstation with the requested IP address. If no match is found, the NBNS sends a negative name query response back to the workstation.

Session Service:

RFC1001 states, "A session is a reliable message exchange, conducted between a pair of NetBIOS applications. Sessions are full-duplex, sequenced, and reliable." The session service of NetBIOS is the generic means by which networked computers exchange messages. The messages can be of any type; the session service is only concerned that messages be transferred between two points reliably and sequentially.

[4] Depending on the NBNS, the server could also return an "end node challenge" response. This packet indicates that the NBNS found a conflicting entry for the unique name, but did not verify that the other workstation is still claiming the contested name. In this case, the end workstation is expected to find this information out on its own by querying the other workstation directly.

RFC1001 goes on to say that TCP on port 139 should be used to emulate the session service functionality. The session service is extremely similar to TCP with a few exceptions. The main difference is that TCP is stream oriented and does not preserve message boundaries. This is in contrast to the session service which sends one message at a time over the network and is expected to read one message at a time from the network. However, solving this problem is straightforward: each session service packet is simply encapsulated with a small header indicating what the message size is before it is sent.

CIFS uses the session service to send and to receive all upper layer commands, including all file and printer operations. Therefore, the first step in any CIFS network communication is usually to establish a NetBIOS session between the client and server.

The session service is defined by a small set of network primitives which are described below. The description also includes how the primitive is mapped into TCP.

- **Call**: Initiate a NetBIOS session. This is mapped into TCP as initiating and creating a full duplex TCP connection and then sending a NetBIOS call packet. The call packet contains the client's NetBIOS name and the server's NetBIOS name.
- **Listen**: Wait for a NetBIOS call command. This is mapped into TCP as a server waiting on port 139 for incoming session requests (a TCP initiation, followed by receiving the call packet described above).
- **Hang up**: End a NetBIOS session. This is mapped into TCP by simply initiating a normal TCP teardown sequence.
- **Send**: Send a message over a NetBIOS session. This is mapped into TCP by encapsulating the data with a small header that contains the message size and then sending the data over the TCP stream.
- **Receive**: Receive a message from a NetBIOS session. This is mapped into TCP as receiving from the TCP stream until the entire message (size dictated by the small header mentioned above) has arrived.
- **Session status**: From RFC1001 "Obtain information about all of the requestor's sessions, under the specified name." This is not used by any CIFS implementations the author is aware of.

The session service maps fairly easily into TCP, and therefore, the description and implementation are not overly complicated. Although every CIFS packet is wrapped with a 4-byte session header indicating the message size, it is very easy to forget that CIFS is not just running over TCP natively.

Datagram Service:

RFC1001 states, "The Datagram service is an unreliable, non-sequenced, connectionless service." The datagram service is used by NetBIOS applications as a fast, broadcast-capable, low-overhead method of transferring data. As the RFC description indicates, however, datagram delivery is not reliable and can also arrive out of order.

The RFC continues on to state that the UDP protocol on port 138 should be used to implement the NetBIOS datagram service within the TCP/IP suite of protocols. UDP is very similar to the NetBIOS datagram service, but still must be slightly modified to emulate all of the datagram service functionality.

All NetBIOS datagram packets that are to be sent over UDP have a header pre-pended to them. This header contains two important pieces of information: the NetBIOS name of the packet sender, and whether or not the NetBIOS datagram was fragmented to be sent via UDP. With this information, the NetBIOS datagram service can be completely emulated over the UDP protocol.

CIFS implementations typically use the NetBIOS datagram service for browsing. Browsing is the process of discovering the NetBIOS names of CIFS servers that are on the network (Windows then displays this list in the Network Neighborhood). Browsing is not part of the standard CIFS protocol, however. Although most CIFS implementations also implement browsing, they are separate entities. Therefore, a pure CIFS implementation would not need to implement the NetBIOS datagram service, only the session and name service described above.

Future Changes:

Many vendors are currently looking to completely phase out NetBIOS and simply run CIFS directly over TCP and UDP. The draft specifications of CIFS1.0 explicitly states that CIFS does not depend on any specific transport protocol and has a brief appendix which indicates how CIFS would run natively over TCP.

NetBIOS still exists mainly for backwards compatibility. If NetBIOS is removed, the changes will not be very drastic. DNS and domain names will replace the NetBIOS naming service. All packets that required the session service would run directly over TCP, and there would be no NetBIOS pre-pended header. All packets that required the datagram service would run directly over UDP, and again, the pre-pended header would become unnecessary.

It is unknown if these changes will actually be made by vendors. Microsoft appears to be very interested in providing CIFS without NetBIOS. If Microsoft does eventually release an OS with CIFS decoupled from NetBIOS, other vendors are likely to make the switch also.

CIFS Internals:

In a nutshell, the Common Internet File System (CIFS) is a network protocol that allows file sharing between network nodes. The protocol is based around a client server design where the client sends request packets to the server, and the server responds back to the client with response packets. Each packet that is sent contains a standard header, plus two variable length fields that are used for packet specific information. Each packet also contains a command field that indicates the general purpose the packet is trying to accomplish. Common command fields indicate that the packet's purpose is to login, open a file, read from a file, or write to a file.

To gain a more in-depth understanding of the protocol, there are three detailed sections on CIFS below. The first section covers major protocol properties. The second section introduces the CIFS standard packet header by diagramming the various fields and defining their purpose. The final section has two typical packet sequence walkthroughs: logging into a server and a file open/read.

Protocol Properties:

Client/server + request/response: As mentioned above, the CIFS architecture is based upon a client sending requests and a server replying to each request sent[5]. The protocol is capable of having multiple simultaneous requests outstanding. This is accomplished through the use of a multiplex id (MID). The client insures that every request that is sent to the server has a unique MID. When the server replies to a given request, the response contains the same MID. In this way, multiple requests can be sent to the server, and the client can simply match the response MID with the MID it generated to know which request has just been replied to.

Command based: Each CIFS packet contains a 1-byte command field. There are currently 100+ commands available and the core functionality of each packet revolves around the specified command. Responses to the client always have the same command code as the request. A list of common command codes is available within the CIFS1.0 draft specification.

Protocol dialects/negotiation: There have been many versions of the CIFS protocol since its inception in the 1980's. Each protocol version is referred to as a dialect and is assigned a unique string to identify the dialect such as "PC NETWORK PROGRAM 1.0" or "NT LM 0.12". When a client wishes to access files on a remote server, the first CIFS packet that is sent is a negotiate protocol packet. In this CIFS packet, the client lists all of the dialect strings that it is capable of understanding. In the response packet, the server indicates which dialect it wishes to communicate in, or

[5] There is one case in which the server will send an unsolicited request to the client. This occurs when the server must break an oplock it has established with the client.

indicates that the server understood no dialects. In this way, the client and server can negotiate which dialect to use for a particular CIFS session.

User/share level security: A share is a server entity (typically a file folder or printer) that is tagged as available to CIFS clients for network sharing. Restricted access to the share is brought about in one of two ways:

1. *User level security*: Indicates that a client wishing to access the share must provide a username and a password for access. This provides the server administrator fine grain control over who has access to the share. This type of security is used in Windows NT and Windows 2000.
2. *Share level security*: Indicates that the share itself requires a password to access, but no username is required and no user identity is established. For example, a password X could be assigned to a certain share. Any user knowing password X can then gain access to the share. There is no fine grain control because there is no concept of individual users and their rights. This type of security is used in Windows 95 and 98.

Encryption: For either of the two security methods above, the actual password is sent in an encrypted format to the server[6]. There are two encryption methods that are commonly used: the newer NT style and the older LAN Manager style. Both encryption methods use challenge-response authentication, where the server sends the client a random string and expects a response that proves the client knows both the random string and the user password.

Command batching: Many CIFS packets are capable of piggybacking other CIFS packets in order to reduce response latency and better utilize network bandwidth. This technique is referred to as ANDX batching.

Opportunistic locking: When a CIFS packet specifies to open a file, an opportunistic lock (oplock) can be requested. If granted by the server, the oplock indicates to the client that no other entities are accessing the file. This allows the client to make any modifications to the file that it wants and not have to write them all to the server immediately. There are multiple types of oplocks and many nuances to them. See the CIFS1.0 draft specification for more information.

[6] Older dialects and bit tweaking in the setup packets can cause passwords to be sent and accepted in plaintext (Hobbit 12).

Packet Header:

Every CIFS request and response uses the packet header below as a template. The various fields are described after the template.

0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7
0xFF	'S'	'M'	'B'
Command	Error Class	Must be zero	Error Code
Error Code (continued)	Flags	Flags2	
Pad or security signature – typically pad and therefore must be zero			
Tree ID (TID)		Process ID (PID)	
User ID (UID)		Multiplex ID (MID)	
WordCount	ParameterWords[WordCount] – the number of words in this variable size section is specified by the WordCount variable.		
ByteCount			
Buffer[ByteCount] – the number of bytes in this variable size section is specified by the ByteCount variable.			

Header: The beginning of every CIFS packet contains a 4-byte header. The first byte is 0xFF, followed by the ASCII representation of the letters 'S', 'M', and 'B'.

Command: The command field contains a one-byte code indicating the CIFS packet type. Examples from the CIFS1.0 draft for this field are SMB_COM_READ_ANDX[7] (0x2E), SMB_COM_TREE_CONNECT (0x70), and SMB_COM_NEGOTIATE (0x72).

Error class: A server indicates whether or not a given request was successful with the error class field. Typically, the field is zero, indicating success. If non-zero, the field indicates what class the error code is from. When set, the error class field takes one of the following values:

- ERRDOS (0x01) – Error is from the core DOS operating system set
- ERRSRV (0x02) – Error is generated by the server network file manager
- ERRHRD (0x03) – Hardware error
- ERRCMD (0xFF) – Command was not in the "SMB" format

Error code: This 16-bit field indicates the type of error that has occurred. It is typically set to zero, indicating no error. If set, this number in conjunction with the error class above can be looked up in the CIFS1.0 draft to give full error descriptions,

[7] The _ANDX following the command name indicates that this command is capable of carrying batched commands as described in the properties section.

such as “bad password” or “file does not exist”. As with the error class, this field is set only by servers in response to a previous request.

Flags: Most of the 8 bits in the flags field specify particular options. The only one of note in this field is bit 3. When bit 3 is set to ‘1’, all pathnames in this particular packet must be treated as caseless. When bit 3 is set to ‘0’, all pathnames are case sensitive.

Flags2: This 16-bit field specifies more options. Bits that are useful:

- Bit 0, if set, indicates that the server may return long file names in the response.
- Bit 6, if set, indicates that any pathname in the request is a long file name.
- Bit 16, if set, indicates strings in the packet are encoded as UNICODE.

Pad/security signature: This field is typically set to zero.

Tree ID (TID): The TID is a 16-bit number that identifies which resource (disk share or printer, typically) this particular CIFS packet is referring to. When packets are exchanged which do not have anything to do with a resource, this number is meaningless and ignored.

If a client wishes to gain access to a resource, the client sends a CIFS packet with the command field set to SMB_COM_TREE_CONNECT_ANDX. In this packet, the share or printer name is specified (i.e. \\SERVER\DIR). The server then verifies that the resource exists and the client has access, then sends back a response indicating success. In this response packet, the server will set the TID to any number that it pleases. From then on, if the client wishes to make requests specific to that resource, it will set the TID to the number it was given.

Process ID (PID): The PID is a 16-bit number that identifies which process is issuing the CIFS request on the client. The server uses this number to check for concurrency issues (typically to guarantee that files will not be corrupted by competing client processes).

User ID (UID): The UID is 16-bit number that identifies the user who is issuing CIFS requests on the client side. The client must obtain the UID from the server by sending a CIFS session setup request containing a username and a password. Upon verifying the username/password, the server responds to the session setup and includes a generated UID. The client then uses the assigned UID in all future CIFS requests. If any of these client requests require file/printer permissions to be checked, the server will verify that the UID in the request has the necessary permissions to perform the operation.

A UID is valid only for the given NetBIOS session. Other sessions could potentially be using an identical UID that the server correlates with a different user. Note: if a server is operating in share level security mode (see above), the UID is meaningless and ignored.

Multiplex ID (MID): The MID is a 16-bit value that is used to allow multiple outstanding client requests to exist without confusion. Whenever a client sends a CIFS packet, it checks to see if it has any other unanswered requests pending. If it does, it insures that the new request will have a different MID then the previously outstanding requests. Whenever a server replies to a CIFS request, it insures that the response it sends matches the request MID that it received. In following this procedure, the client can always know exactly which outstanding request an incoming reply is correlated to.

WordCount and parameter words: CIFS packets use these two fields to hold command-specific data. The CIFS packet header template above cannot hold every possible data type for every possible CIFS packet. To remedy this, the parameter words field was created with a variable length. The wordcount specifies how many 16-bit words the parameter words field will actually contain. In this way, each CIFS packet can adjust to the size needed to carry its own command-specific data.

The wordcount for each packet type is typically constant and defined in the CIFS1.0 draft. There are two wordcounts defined for every single command; one wordcount for the client request and another for the server response. Two counts are needed because the amount of data necessary to make a request is not necessarily the same amount needed to issue a reply.

ByteCount and buffer: These fields are very similar to the wordcount and parameter words fields above; they hold a variable amount of data that is specified on a per packet basis. The bytecount indicates how many bytes of data will exist in the buffer field that follows.

The major difference between the parameter data section above and the buffer is what type of data they store. The parameter words data section typically holds a small number of packet options, while the buffer data section typically holds large amounts of raw data (e.g. file data).

Packet Sequence Walkthroughs:

Two common CIFS client/server packet exchanges are presented below. The first exchange shows what packets are sent when a client first initiates contact with a server: a NetBIOS session is established, a CIFS dialect is negotiated, a username and password are sent, and finally, a resource (such as a printer or directory share) is connected to. The second exchange examines the packets required to open a file and read from it.

For each exchange, an initial summary gives an overview of the entire process. In addition, each packet that is sent and received is detailed (by listing out the various packet header values) and also summarized.

Note that in all examples, the client always sends to TCP port 139, and the server always replies to the ephemeral port number that is chosen by the client. In addition, the following CIFS packet fields will not be described for each individual packet:

- *Error class/code* – always zeros when sent from client and assumed to be zeros when returned from server (the examples do not cover error scenarios).
- *Flags* – set to 0x00 for all packets (case sensitive pathnames)
- *Flags2* – set to 0x0001 for all packets (long file name support)
- *Pad/security signature* – Set to zero in all packets

Initial contact, login, and tree connect:

When a CIFS client determines that it wishes to access resources on a CIFS server, the following packet sequence is exchanged. First, a NetBIOS session is established in order to provide a reliable message sequence transport service. Then, the client and server negotiate the CIFS dialect in which they will be communicating. The client then logs into the server, sending a username and password (for this example, the server will be operating in user level security). Finally, the client connects to the desired resource.

Packet #1 request, client –► server
Purpose: Establish NetBIOS session
Summary:

The client, wishing to exchange CIFS packets with the server, initiates a NetBIOS session between itself and the server (referred to as "calling the server" in the previous NetBIOS section). This provides for sequenced, reliable message delivery between the two endpoints. Note that the client must know the server's NetBIOS name in order to call it and also must indicate its own NetBIOS name.

The events to establish the NetBIOS session are as follows. First, the client establishes a full duplex TCP connection with the server on port 139. Once this is accomplished, the client builds and sends a NetBIOS session request packet (not diagrammed in the NetBIOS section above, but described in RFC1002) over the TCP connection. In summary, the session request packet contains the client's NetBIOS name, the server's NetBIOS name, and an integer constant which indicates the packet's purpose is to establish a NetBIOS session. Please see RFC1002 for more details.

Packet #2 response, server –► client
Purpose: Positive NetBIOS session acknowledgement
Summary:

If the above session request packet contained the server's NetBIOS name, and the packet was formatted correctly, the server replies with a simple session established acknowledgement. This 4-byte packet is also described in RFC1002. In summary, it indicates either a successful session establishment or an error code.

Packet #3 request, client –► server
Purpose: Negotiate CIFS dialect

Summary:

Now that the NetBIOS session is established, the client is ready to send the first real CIFS request. The client sends the SMB_COM_NEGOTIATE command and includes a list of CIFS dialects that it understands.

Packet:

Command: SMB_COM_NEGOTIATE (0x72)
TID: Ignored in this packet.
PID: Set to process ID of client process.
UID: Ignored in this packet.
MID: Any unique number.
WordCount: 0
ParameterWords: There are none because wordcount is 0.
Bytecount: Set to 119 (variable depending on how many CIFS dialects the client understands).
Buffer: Contains 119 bytes worth of dialect descriptions, examples would be as follows: “PC NETWORK PROGRAM 1.0”, “MICROSOFT NETWORKS 3.0”, “DOS LM1.2X002”, “DOS LANMAN2.1”, “Windows for Workgroups 3.1a”, “NT LM 0.12”.

Packet #4 response, server –► client
Purpose: Choose CIFS dialect from request list
Summary:

The server is now responding to the negotiate protocol request by selecting the dialect that it wishes to communicate in.

Packet:

Command: SMB_COM_NEGOTIATE (0x72)
TID: Ignored in this packet.
PID: Ignored when packet is from server.
UID: Ignored in this packet.
MID: matches unique number chose above.
WordCount: This number depends on the dialect that is chosen. For this example, we will assume that the server chose “NT LM 0.12”[8]. In this case, the wordcount is 17.
ParameterWords: The 17 words contained here indicate the chosen dialect and many server properties. Of note is the MaxMpxCount (which states the max number of pending requests the client can initiate) and the 32-bit capabilities flags (which indicate if UNICODE is supported, if large files are supported, if NT commands are supported, and more).
Bytecount: Variable, usually greater than 8.
Buffer: Typically contains an 8-byte random string that the client uses in the next packet for encryption purposes.

Packet #5 request, client –► server
Purpose: User login
Summary:

[8] The CIFS dialect “NT LM 0.12” is the most modern dialect available as of this writing.

Now that the CIFS dialect has been agreed upon, the client sends a packet containing a username and password to gain a user ID (UID). This packet also relays client capabilities to the server, so the packet must be sent even if the server is using share level security.

Packet:

Command: SMB_COM_SESSION_SETUP_ANDX (0x73)
TID: Ignored in this packet.
PID: Set to process ID of client process.
UID: Ignored in this packet.
MID: Any unique number.
WordCount: 12
ParameterWords: This section is very similar to the server's negotiate protocol parameter words response. However, instead of listing the server's capabilities, it lists the client's. It also contains the size of the passwords to be supplied in the buffer section below.
Bytecount: Variable, the buffer below contains the encrypted password, the username, the name of the operating system and the native LAN manger. Therefore, the size listed here depends on the string sizes of all these entities.
Buffer: As mentioned above, this field actually contains the password, username, and other strings that identify the operating system involved.

Packet #6 response, server –► client
Purpose: Indicates User ID (UID) or returns error if bad password
Summary:

Once the server receives the encrypted password and username, it checks if the combination is valid. If the password is invalid, this response packet will be returned with the error class and code set to the appropriate error value. If the username/password is correct, then this packet contains the UID that the client will begin to send with every packet from here on.

Packet:

Command: SMB_COM_SESSION_SETUP_ANDX (0x73)
TID: Ignored in this packet.
PID: Ignored when packet is from server.
UID: The 16-bit number that the server has assigned to represent client user identity.
MID: Matches unique number chose above.
WordCount: 3
ParameterWords: Nothing relevant to normal operation.
Bytecount: Variable, the buffer below contains strings stating the server OS and native LAN manager type.
Buffer: Contains strings indicating the server OS and LAN manager type.

Packet #7 request, client –► server
Purpose: Connect to particular resource
Summary:

At this point, the client has authenticated itself to the server and may proceed to connect to the actual share. In this packet, the client specifies the share that it wishes to access. Share names are specified in UNC format (i.e. \\SERVER\SHARE).

Packet:

Command: SMB_COM_TREE_CONNECT_ANDX (0x75)
TID: Ignored in this packet.
PID: Set to process ID of client process.
UID: Set to the server returned UID from the above session setup response.
MID: Any unique number.
WordCount: 4
ParameterWords: Nothing relevant to normal operation.
Bytecount: Variable, depends on the size of the UNC string that is requested below.
Buffer: Contains the share name that the client wishes to access.

Packet #8 response, server –► client
Purpose: Indicates Tree ID (TID) or error if share name does not exist
Summary:

If the share specified above exists and the user has access permission, then the server returns a successful response with the TID set to the number it wishes to refer to the resource as. If the share does not exist or the user does not have access permission, the server will return the appropriate error class and error code here. Assuming that this packet indicates success, the client now has everything it needs to access files from the specified share. This is the final packet in this client/server exchange.

Packet:

Command: SMB_COM_SESSION_SETUP_ANDX (0x73)
TID: 16-bit number which server has assigned to represent the requested resource.
PID: Ignored when packet is from server.
UID: 16-bit number representing the user.
MID: Matches unique number chosen above.
WordCount: 3
ParameterWords: Nothing relevant to normal operation.
Bytecount: Variable, the buffer below contains strings stating the native file system and device type of the requested resource.
Buffer: Contains strings that state the native file system and device type.

File open and read:

Once a client has completed the initial packet exchange sequence described above, it may open and read files from the share that was requested. The file open consists of one CIFS request and one CIFS response. The read request also consists of one request and one response packet.

Packet #1 request, client –► server
Purpose: Open a file
Summary:

In order to read or write to a file, it first must be opened. This CIFS packet does exactly that.

Packet:

Command: SMB_COM_OPEN_ANDX (0x2D)
TID: Set to the server returned TID from the tree connect response above.
PID: Set to process ID of client process.
UID: Set to the server returned UID from the session setup response above.
MID: Any unique number.
WordCount: 15
ParameterWords: Specifies many open options such as mode (read, write, or readwrite) and sharing mode (none, read, write).
Bytecount: Variable, depends on the size of the string that contains the filename.
Buffer: Contains the name of the file to be opened.

Packet #2 response, server –► client
Purpose: Indicate File ID, or error code if problem
Summary:

The server checks to see if the filename above exists and if the user specified in the UID has permission to access the file. If these conditions are not met, the server will return the appropriate error class and error code indicating what that problem is. If there are no errors, the server returns a response packet that includes a File ID (FID) that can be used in subsequent packets for accessing the file. Note that the FID is returned to the client in the parameter words field of the response. There is no FID field in the standard CIFS header.

Packet:

Command: SMB_COM_OPEN_ANDX (0x2D)
TID: 16-bit number which the server assigned to represent the requested resource.
PID: Ignored when packet is from the server.
UID: 16-bit number representing the user.
MID: Matches unique number chosen above.
WordCount: 15
ParameterWords: Many flags indicating what type of actions occurred and the very important 16-bit FID.
Bytecount: 0
Buffer: No data in buffer.

Packet #3 request, client –► server
Purpose: Read from a file
Summary:

Assuming that the above response indicated a FID for the client to use, an actual read request for file data can now be issued.

Packet:

Command: SMB_COM_READ_ANDX (0x2E)
TID: Set to the server-returned TID from the tree connect response above.
PID: Set to process ID of client process.

UID: Set to the server-returned UID from the session setup response above.
MID: Any unique number.
WordCount: 10
ParameterWords: Here, the FID is stated so the server knows which opened file the client is referring to. Also indicated here are a 32-bit file offset and a 16-bit count value. These two numbers dictate where and how much data to return from the file.
Bytecount: 0
Buffer: No data in buffer.

Packet #4 response, server –► client
Purpose: Returns file data requested
Summary:

This packet contains the requested file data. Assuming the UID, TID, and FID were all valid numbers in the request, an error here should be unlikely.

Packet:

Command: SMB_COM_READ_ANDX (0x2E)
TID: 16-bit number which server has assigned to represent the requested resource.
PID: Ignored when packet is from the server.
UID: 16-bit number representing the user.
MID: Matches unique number chosen above.
WordCount: 12
ParameterWords: Here, the number of bytes that were actually read is indicated. This does not necessarily match the number requested (in case the request exceeded the file boundary).
Bytecount: Variable, the buffer holds the file data, so this number is also the number of bytes that were actually read.
Buffer: The file data requested.

Final Comments:

This concludes the discussion of the CIFS protocol. For more information, please reference the bibliography section or www.codefx.com. Comments or error submissions? Please email details@codefx.com.

Bibliography:

Auerbach, Karl. Protocol Standard For A Netbios Service On A Tcp/Udp Transport: Concepts And Methods. RFC1001, March 1987.
[Presents concepts and methods on how to implement the various NetBIOS services on a TCP/UDP network. No CIFS information, however, CIFS uses NetBIOS for the transport protocol so the information is still relevant. Freely available all over the web.]

Auerbach, Karl. Protocol Standard For A Netbios Service On A Tcp/Udp Transport: Detailed Specifications. RFC1002, March 1987.
[Same description as above, except this RFC gives actual packet formats and psuedo-code implementations. Also available freely on the web.]

Eckstein, Collier-Brown, and Peter Kelly. Using Samba. O'Reilly. Nov. 1999
[Although this book is mainly about using and learning Samba, it provides quality information on NetBIOS and good descriptions of common CIFS issues. Chapter 1 is most relevant. The book is available freely online at http://www.oreilly.com/catalog/samba/chapter/book/index.html.]

Hertel, Chris. "Samba: An Introduction". Samba Webpages. 10 June 1999. Samba Webpages. 2000. http://www.samba.org/samba/docs/SambaIntro.html.
[A decent overview of CIFS. Contains good info on potential future CIFS. Available freely online at link above.]

Hobbit. "CIFS: Common Insecurities Fail Scrutiny". Avian Research. Jan. 1997. http://web.textfiles.com/hacking/cifs.txt.
[The title says it all. This lengthy paper gives great overviews on security flaws in CIFS. Very good information, available freely online at link above.]

Internet Engineering Task Force (IETF), Storage Networking Industry Association (SNIA). The Common Internet File System (CIFS) Protocol. INTERNET-DRAFT version, May 2000.
[This is the CIFS1.0 draft specification that is referenced heavily in this document. This is the number one place to look for detailed information. The best place to find the latest version is probably the SNIA website (www.snia.org). Follow the links from "products and services" to "technical activities" to "workgroups" to "nas" to "cifs". Once there, check the "work completed" section.]

The Open Group. Protocols for X/Open PC Internetworking: SMB, Version 2. Berkshire, UK: X/Open Company Limited, September 1992.
[This document is the X/Open's 1992 attempt at standardizing the SMB protocol. All in all, very complete and thorough, however, all material is somewhat dated at this point. Not available freely, must buy directly from X/Open.]

Sharpe, Richard. "What is SMB". Samba Webpages. 27 Sept. 1999. Samba Webpages 2000. http://www.samba.org/cifs/docs/what-is-smb.html.
[Probably the best SMB/CIFS introduction available. Also a number of interesting links.]

Shearer, Dan. "History of SMB". Samba Webpages. 16 Nov. 1996. Samba Webpages 2000. http://www.samba.org/cifs/docs/smb-history.html.
[Contains a history of the SMB protocol. Basically a list of the various documents that have contributed to SMB development, who wrote them, and when.]

Tridgell, Andrew. "Inside Microsoft Networking". Samba Ftp Site. ftp.samba.org/pub/samba/slides/inside_SMB.tar.gz. 31 July 1999.
[Another very good introduction to CIFS. This also includes some browsing and NetBIOS name service information. Written by the original and core author of Samba.]

Additional Information:

For additional information, obtain access to all or part of the bibliography listed above. If more information is still required, please reference the links below.

- The Samba website contains a wealth of information, from presentation slides to web links. This can be found at www.samba.org by clicking on the documentation link. Please note that the bibliography above links directly to a few outstanding articles on this site.

- A very detailed explanation of NetBIOS and a brief overview of CIFS can be found at http://www.ubiqx.org/cifs/index.html.

- Microsoft has a number of CIFS documents on their FTP site. This can be located at ftp.microsoft.com/developr/drg/cifs. Many of the documents describe older dialects of the CIFS protocol. Read the "index.txt" file on the ftp site for more information on the files available.

- The following website contains many useful and interesting CIFS links. http://ourworld.compuserve.com/homepages/TimothyDEvans/nbf.htm

- There are two mailing lists that the CIFS community uses. The more active of the two is the Samba list, which can be searched and subscribed to by following the link below. http://samba.org/samba/archives.html. Microsoft sponsors the other list; it can be viewed or subscribed to at http://discuss.microsoft.com/archives/cifs.html.

- One of the best methods of understanding the CIFS protocol is to watch it in action. Download a copy of the Ethereal packet sniffer from www.ethereal.com. Ethereal is available for Win32 and Unix machines if you have the GTK+ GUI libraries. It is totally free and includes a CIFS parser that works great.

- Finally, the newsgroup comp.protocols.smb is also available for searching and posting.