The following paper was originally published in the
Proceedings of the 2nd USENIX Windows NT Symposium
Seattle, Washington, August 3–4, 1998

# SecureShare: Safe UNIX/Windows File Sharing through Multiprotocol Locking

Andrea J. Borr

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org/

# SecureShare: Safe UNIX/Windows File Sharing through Multiprotocol Locking

Andrea J. Borr
aborr@hummosa.com

## Abstract

As mixed UNIX/Windows environments become more common, safe file sharing among the NFS and CIFS clients becomes a significant problem. In this paper, we describe SecureShare, a multiprotocol file sharing technology that resolves the mismatch between NFS/NLM and CIFS file locking protocols, provides coherent caching and enables multiprotocol change notification. SecureShare achieves the goal through a uniform lock-mode model and multiprotocol oplock management. This paper presents the main features of SecureShare and discusses the rationale behind its design.

## 1. Introduction

Today, mixed UNIX/Windows environments are becoming increasingly common. Many commercial and academic institutes typically employ a mixed network of UNIX clients using the Network File System [3] (optionally with Network Lock Manager [5]) and Windows clients using either the Common Internet File System [6] or "(PC)NFS" [7]. Sharing files across the different systems is highly desirable and commonly done. However, correct concurrent read/write accesses from the different types of clients present a significant problem, because of the mismatch between NFS/NLM locking protocols and CIFS locking protocols.

Uncoordinated concurrent read/write accesses to files and directories can result in application failures or file data integrity problems. Examples of such problems are (1) readers receiving "stale" data (i.e. data currently in the process of being updated by another application), (2) writers overwriting each other's updates, and (3) applications having their in-use files deleted "out from under" them. Locking is typically used to coordinate concurrent accesses to files and directories, and prevents such problems from occurring.

Unfortunately, Windows and UNIX differ considerably in how they allow applications to access files and data while controlling the concurrency issues that arise in multi-user, multi-application environments. Windows implements a robust and complete set of file-open, data access and locking paradigms. UNIX, by contrast, typically implements only a basic data access facility, ignoring for the most part issues of concurrency and file sharing altogether. These differences in the handling of locking and file sharing issues carry over into the designs of the NFS and CIFS protocols used by UNIX and Windows, respectively, for remote data access. For example, CIFS transmits file open requests and byte-range lock requests to the file server, whereas NFS does not.

Correct interoperability of CIFS and NFS/NLM is impeded by the fact that CIFS and NFS/NLM have different and incompatible semantics for file-locking, file-open, and file sharing. The two principal interoperability problems are (1) CIFS mandatory locking vs. NFS/NLM advisory locking; (2) CIFS hierarchical locking vs. NFS/NLM non-hierarchical locking.

CIFS requires that the file server and all of its clients conform to the mandatory locking model. In the mandatory model, the file opener's (alternatively, the byte-range lock owner's) exclusivity of access to the opened file (locked byte-range) is enforced by the file server at the system level.

By contrast, NFS/NLM lacks file-open functionality, and provides only advisory locking. In the advisory model, system-level enforcement is replaced by application-level compliance. In this environment, each of a set of applications that read and write shared data depends for its correct functioning on global compliance with advisory locking rules.

In a mixed CIFS and NFS/NLM network, the incompatibility of the mandatory and advisory models poses a risk to data integrity. Windows-based applications depend on the stricter CIFS mandatory model for their correct functioning and for the integrity of their data. These applications may fail in the face of NFS accesses that write, remove, or otherwise corrupt in-use Windows files in a manner that violates the stricter CIFS mandatory model, but is permitted under the looser NFS/NLM advisory model.

The second problem impeding correct interoperability of CIFS and NFS/NLM is the fact CIFS expects all clients to conform to a hierarchical locking model. Correct application functioning and data integrity under CIFS rest on the assumption that a client first acquires a file-lock (i.e. by opening a file) before requesting a byte-range lock within the file. The NFS/NLM protocol, on the other hand, has no concept of a locking hier-

archy. NFS/NLM has provision for acquiring a byte-range lock on a file, yet lacks provision for pre-acquiring a file-lock on the file by opening the file.

Network Appliance's solution to these problems is SecureShare [1], a multiprotocol file sharing technology. SecureShare is integrated into the *Data ONTAP*™ microkernel, Network Appliance's operating system for its proprietary file server appliance, or *filer* [2]. SecureShare provides correct semantics for file sharing, opportunistic locking, byte-range locking, coherent caching, and change notification in a mixed network of UNIX clients using NFS, optionally with NLM, and Windows clients using either CIFS or (PC)NFS.

The key features of SecureShare are:

- uniform lock mode and multiprotocol lock enforcement
- multiprotocol oplock management
- multiprotocol change-notify.

In implementing multiprotocol data integrity, SecureShare reconciles the different and incompatible locking and file-open semantics utilized by CIFS and NFS/NLM clients. In implementing multiprotocol oplock management, SecureShare supports standard CIFS oplocks, while at the same time making oplocked data available to NFS-based clients through multiprotocol oplock break. In implementing multiprotocol change-notify, SecureShare supports standard CIFS change-notify, while extending the change-notification service such that it covers changes due to NFS in addition to covering changes due to CIFS.

SecureShare implements a uniform set of file-locking semantics for the multiprotocol environment. To achieve consistent multiprotocol locking and file-open semantics, SecureShare manages all locks according to a uniform lock management model. In this model, an expedient described in Section 2.2 is used to overcome the interoperability problem posed by the hierarchical locking conformance mismatch between CIFS and NFS/NLM. Uniform lock management also solves the interoperability problem posed by the mandatory vs. advisory mismatch. Lock enforcement under SecureShare depends on the lock type, the protocol that set the lock, and the protocol performing a file access. Whole-file locks (representing file-opens and oplocks) are enforced uniformly on the mandatory model for both CIFS and NFS file accesses. Byte-range locks may be enforced either on the mandatory or the advisory model, depending on which protocol created the lock and which protocol is performing the read or write. In particular, NLM byte-range locks are treated as advisory with respect to NFS reads and writes, in accordance with NFS/NLM protocol standards.

The cornerstone of SecureShare is a multiprotocol lock manager. The lock manager coordinates and manages — in a unified set of kernel-space data structures — the lock types needed for multiprotocol file-open and locking support. Since it is integrated into the *Data ONTAP* kernel, the lock manager is able to validate that reads and writes of files and directories do not violate locks. SecureShare enforces CIFS locks and file-open semantics at the system level. By contrast, in a UNIX-based CIFS implementation such as Samba [9], Windows file-open information is maintained by a module that is disjoint from — and has no interaction with — the module that implements UNIX and NLM locking functionality. Moreover, because data access functions under UNIX do not normally check for lock conflicts, there is nothing to stop local or NFS-based UNIX users and applications from accessing, corrupting, or even removing "locked" Windows files and data. Thus, it is possible in such an environment for UNIX users or NFS clients to write, remove, or move files that CIFS-based Windows applications are holding open and actively accessing.

SecureShare implements a multiprotocol extrapolation of the Windows coherent caching and networking performance optimization known as "opportunistic locks" (*oplocks*) [6]. By extrapolating oplocks to the multiprotocol environment, SecureShare allows CIFS clients to safely reap oplocks' performance benefits of aggressive client-side caching without causing Windows applications to suffer the effects of potential data corruption due to uncoordinated NFS accesses. By allowing non-CIFS requests to *break* (i.e. revoke) CIFS oplocks, SecureShare ensures that oplocked file data remains available to NFS-based applications and users, while simultaneously protecting the integrity and cache coherence of that data.

The remainder of this paper is organized as follows. Section 2 discusses file and data locking paradigms in the CIFS, UNIX/NFS, and (PC)NFS environments, and presents the uniform lock-mode model in SecureShare. Section 3 discusses multiprotocol lock enforcement. Section 4 discusses the CIFS opportunistic locking model and SecureShare's multiprotocol implementation. Section 5 discusses the CIFS *change-notify* feature and SecureShare's multiprotocol implementation. Finally , Section 6 summarizes.

## 2. Data Locking Paradigms

A lock on a data element grants to the lock owner a degree of exclusivity of access represented by its *lock-*

*mode*. The lock *type* specifies the span of data covered by the lock. In the case of a file server, lock types are categorized as a two-level hierarchy:

- File-locks (pertain to whole files)
- Byte-range locks (also called "record" locks)

## 2.1. Lock Models in CIFS and NFS

In the Windows (and CIFS protocol) model, a file-lock is obtained as a side effect of a *file-open* operation. The file opener specifies the *access-mode* it desires and the *deny-mode* that it desires to impose on other openers of the same file. In granting a new file-open request, the CIFS file server validates that the access- and deny-mode requested by the new opener do not conflict with the access- and deny-mode granted to pre-existing file openers. It then creates a *file-lock* that represents the access- and deny-mode granted to the new file-open.

In Windows (and CIFS), the application (CIFS client) must perform a file-open operation before attempting to read or write data contained in the file. After the open request has been granted, the CIFS file server validates that reads or writes requested on the open file comply with the access-mode granted by the file-open. CIFS expects *all clients* to conform to a hierarchical locking model. Correct application functioning and data integrity under CIFS rest on the assumption that a client first acquires a file-lock (i.e. by opening a file) before requesting a byte-range lock within the file.

By contrast, the UNIX file-open API does not support the concept of a *deny-mode* that it desires to impose on other openers of the same file. Lacking the concept of a deny-mode, the UNIX opener lacks a means of specifying a file-lock, since a lock by its very nature embodies the specification of a degree of exclusivity of access. Furthermore — because it was designed to be stateless [4] and file-opens are inherently stateful — NFS does not transmit even the *access-mode* declared by the UNIX file opener to the file server. Thus, the UNIX/NFS client cannot perform an operation that results in the acquisition of a file-lock at the server. Nor can it pre-declare to the server its intent to issue file reads or writes. Since the UNIX/NFS client can issue an NLM byte-range lock request for a file (see below) without holding a file-lock for the file, NFS/NLM is seen to violate the CIFS hierarchical locking model. The implications of this violation for SecureShare's multiprotocol lock enforcement are seen in Section 3.

(PC)NFS implementations for DOS and Windows emulate CIFS file-open functionality through use of NLM "share" locks, added to the NLM protocol for the specific purpose of supporting these clients. Called NLM "file-locks" in this paper, they are equivalent to CIFS file-locks. (This functionality is not normally exposed to the UNIX application environment, and so most UNIX users are unaware of its presence). The (PC)NFS client requesting an NLM file-lock specifies an *access-mode* and a *deny-mode*. In granting a new NLM file-lock, the server validates that the access- and deny-mode requested by the new request do not conflict with the access- and deny-mode granted to any pre-existing file-lock on the same file. The server then creates an NLM file-lock that emulates the presence of a new "file-open."

Byte-range locks are used to restrict other applications' access to sections of an open file, usually while the holder of the byte-range lock is intending to read or write the locked section. Byte-range locks can be either *read* locks or *write* locks. Depending on whether enforcement is advisory or mandatory, read locks either induce or enforce that other applications refrain from *writing* to the specified byte-range. Similarly, write locks either induce or enforce that other applications refrain from reading or writing the specified byte-range.

Both Windows and UNIX provide byte-range locking functionality. The Windows byte-range lock API is directly propagated via the CIFS protocol to the CIFS file server, where the resultant lock is enforced in accordance with the mandatory model. Since lock management is inherently stateful, NFS does not transmit UNIX byte-range lock requests to the server. Because the utility of locking in a file-sharing environment was recognized, however, the adjunct protocol NLM was defined to transmit UNIX byte-range lock requests to the server. NFS treats NLM locks as merely advisory, however.

## 2.2. The Uniform Lock-Mode Model

SecureShare implements a *uniform lock-mode* encompassing both file-locks and byte-range locks. The lock-mode combines a specification of the *access-mode* the lock requester desires (*Read*, *Write*, *Read-Write*), and the *deny-mode* it desires to impose on other clients concurrently attempting to access the same data (*Deny-None*, *Deny-Read*, *Deny-Write*, *Deny-All*). Some examples of uniform lock-modes are *Read/Deny-Write* and *Read-Write/Deny-All*.

When SecureShare creates a file-lock to represent a CIFS or (PC)NFS file-open, it derives a uniform lock-mode for the file-lock that combines the requested *access-mode* and *deny-mode*. In granting the new request, SecureShare validates that the derived lock-mode does not conflict with the lock-mode granted to any pre-existing file-lock on the same file.

CIFS or NLM byte-range locks are assigned uniform lock-modes as follows. In the case of a *write-lock* (also called an *exclusive* byte-range lock), the locked byte-range is *Read-Write* for the holder of the lock and *Deny-All* for others. Thus, the uniform lock-mode is *Read-Write/Deny-All* (RW/DA), and the lock is effectively "exclusive." A write-lock is grantable to only one "writer" of the byte-range at a time. In the case of a *read-lock* (also called a *non-exclusive* byte-range lock), the locked byte-range is *Read* for the holder of the lock and *Deny-Write* for others. Thus, the uniform lock-mode is *Read/Deny-Write* (R/DW), and the lock is "sharable." A read-lock is concurrently grantable to multiple "readers" of the byte-range, but is incompatible with a "writer" attempting to obtain a write-lock on the byte-range.

UNIX/NFS clients issue NLM byte-range locks without holding file-locks. This presents a problem for strict hierarchical lock enforcement, in which byte-range locks are not directly comparable to file-locks. SecureShare overcomes this problem through an expedient that allows a direct compatibility check of a file-lock against an NLM byte-range lock (see Table 2 at end of paper). Such comparisons, of course, violate the principles of hierarchical locking. The anomaly is circumvented by considering the NLM byte-range lock to have a deny-mode of *Deny-None* for purposes of comparison with a file-lock. After all, a write-lock's deny-mode — *Deny-All* — applies only to the byte range, not to the whole file. On the other hand, the write-lock's access-mode — *Read-Write* — signals the lock owner's intention to write to the file. In fact, since NFS provides no means of declaring a file-open time access-mode, the NFS/NLM client's only means of pre-declaring an intention to write the file is to acquire an NLM write-lock on one or more of the file's byte-ranges.

## 3. Multiprotocol Lock Enforcement

This section describes how SecureShare manages locks in a multiprotocol environment. It describes SecureShare's multiprotocol lock enforcement policies for CIFS file-locks, CIFS byte-range locks, and NLM byte-range locks. It highlights the risks to data integrity that could arise in the multiprotocol environment if locking were not managed in the described way. In describing various cases of multiprotocol lock enforcement, it points out scenarios in which oplock break protocol (discussed in Section 4) is triggered.

### 3.1. CIFS File-Lock Enforcement

SecureShare fully enforces Windows mandatory file locking across the CIFS, (PC)NFS and UNIX/NFS data access environments. When a Windows client attempts to open a file residing on a Network Appliance filer, SecureShare tests whether the open request can be granted based on the following criteria:

If other CIFS or (PC)NFS clients already have the file open, a "file-lock compatibility check" is performed to determine whether the access- and deny-mode of the new open are in conflict with access- and deny-mode(s) already granted to pre-existing opens of the same file. Computation of the compatibility check utilizes Table 1 (at end of paper), known in database management locking terminology as a *lock compatibility matrix* [8]. First, the access- and deny-mode requested by the new open are combined into a *uniform lock-mode*. Next, if there are multiple pre-existing opens, a *lock conversion matrix* (as defined in [8]) is used iteratively to combine the lock-modes implied by the access- and deny-mode(s) of pre-existing opens into a *cumulative lock-mode*. Then, Table 1 is used to test the compatibility of the lock-mode requested by the new open with the cumulative lock-mode of pre-existing opens. If the new lock-mode is not compatible, then the file-open request fails with a "sharing violation" error.

Suppose that — prior to the filer's receipt of the CIFS file-open request — a UNIX/NFS client has obtained an NLM byte-range lock on the file being opened. In this case, SecureShare uses Table 2 (at end of paper) to check whether the *deny-mode* component of the file lock-mode of the new open is compatible with the access-mode component (*Read* or *Read-Write*) of the lock-mode of the NLM byte-range lock. Note that this check — which violates the locking hierarchy — compares a file-lock with a byte-range lock. The rationale for this check is that — in the absence of NFS file-open functionality — the only way for a UNIX client to pre-declare its intention to make read (write) accesses to a file is by acquiring a read-lock (write-lock) on one or more byte-ranges.

Once one or more Windows clients have successfully opened a file on a Network Appliance filer, SecureShare coordinates subsequent NFS access to the open file in accordance with the deny-mode(s) granted to the openers.

SecureShare will reject any attempt by an NFS client (UNIX or Windows) to delete or rename a file that is being held open (after a possible oplock break sequence) by a Windows client. Note, however, that CIFS provides protocol for specifying a file open operation (i.e. *NTCreateX* with *share delete* mode) that permits *another CIFS client* to delete or rename a file (or directory) that is being held open.

SecureShare will reject any attempt by an NFS client to write to a file that is being held open (after a possible

oplock break sequence) by a Windows client with a deny-mode of *Deny-Write* or *Deny-All*.

SecureShare is able to manage NFS file access and file system manipulation functions at the system level. Thus, it can deny NFS operations — such as writes, removes, renames, or moves — that could overwrite and/or corrupt a Windows-open file, violate a CIFS byte-range lock, or delete the file "out from under" a Windows application.

By contrast, because data access functions under UNIX do not normally check for lock conflicts, Windows mandatory locks are typically not enforced in UNIX-based CIFS implementations such as Samba. In such an environment, there is nothing to stop local or NFS-based UNIX users and applications from accessing, corrupting, or even removing "locked" Windows files and data.

On the other hand, SecureShare *does* allow UNIX/NFS clients to obtain read-only access to files that are actively being held open by Windows applications, even if those files were opened with deny-mode *Deny-Read* or *Deny-All*. Note that we are referring here to *actively open* files, as distinguished from files that temporarily *appear* to be open due to breakable stale batch oplocks (discussed in Section 4). This design decision was consciously made with consideration to practicality over technical "purity." The ability of a UNIX/NFS client to read files that are actively open by Windows applications cannot corrupt the file data from the CIFS point of view, and the NFS client's perceived cache coherence is no worse than is normal under NFS. However, if SecureShare were to deny all UNIX/NFS clients' attempts to read files that are open under Windows, this could generate an unacceptably large number of seemingly "inexplicable" errors for UNIX/NFS clients in the multiprotocol environment. This design decision is compatible with SecureShare's goal of protecting Windows data from corruption by NFS write accesses. It is arguably preferable to allow an NFS read request to return "dirty" data than to fail the request with a seemingly inexplicable "access violation" error.

## 3.2. CIFS Byte-Range Lock Enforcement

Since a file must be opened via CIFS before a byte-range lock request can occur, a CIFS byte-range lock request never occasions an oplock break (it would have occurred at file open time). Enforcement of CIFS byte-range locks in a multiprotocol environment occurs as follows:

- If another CIFS client already holds a byte-range lock that conflicts with the new CIFS byte-range lock being requested, then the new lock request will be denied.
- If a UNIX/NFS client or a (PC)NFS client already holds an NLM byte-range lock that conflicts with the new CIFS byte-range lock, then the new lock request will be denied.
- If neither of the above violations is detected, then the new CIFS lock request will be granted.

SecureShare treats CIFS byte-range locks as mandatory with respect to NFS file access functions, assuring that NFS reads and writes do not violate CIFS byte-range locks. By contrast, a UNIX-based multiprotocol file server such as Samba has no easy way to prevent violation of CIFS locks by NFS file access functions. Thus, it would be possible — due to the advisory nature of UNIX/NFS locking — for UNIX users and applications to write data to the CIFS "locked" byte-ranges of files.

## 3.3. UNIX Byte-Range Lock Enforcement

UNIX provides an API only for byte-range locking. When NFS-mounted data is locked, UNIX uses NLM to transmit the byte-range lock request to the file server. (The filer's NLM implementation supports all of the UNIX byte-range locking semantics, such as region lock merging and sub-region unlocking). SecureShare handles a new NLM byte-range lock request as follows:

- SecureShare first breaks any pre-existing oplock that may be held on the file by a CIFS client. This ensures that SecureShare has all of the information on pre-existing CIFS byte-range locks for the file. (Note: As discussed in Section 4, CIFS clients that hold exclusive or batch oplocks do not propagate lock information back to the server).
- SecureShare next uses Table 2 to check whether the *access-mode* component (*Read* or *Read-Write*) of the lock-mode of the new NLM byte-range lock request conflicts with the *deny-mode* component of any pre-existing file lock-mode. Note that this check is anomalous with respect to the locking hierarchy.
  - If the NLM byte-range lock request is for an exclusive lock (write-lock), it is incompatible with a pre-existing CIFS open having a deny-mode other than *Deny-None.*
  - If the NLM byte-range lock request is for a non-exclusive lock (read-lock), it is incompatible with a pre-existing CIFS open having a deny-mode of *Deny-Read* or *Deny-All*
- SecureShare then checks whether the new NLM byte-range lock request conflicts with any pre-

existing NLM or CIFS byte-range lock that has already been granted on the same file. If a conflict exists, the lock request is rejected.

- If none of the above violations is detected, then the NLM byte-range lock request is granted.

In accordance with NFS/NLM protocol standards, SecureShare treats NLM byte-range locks as advisory with respect to NFS file access functions. However, in accordance with the design goals for its multiprotocol support, SecureShare treats NLM byte-range locks as mandatory with respect to CIFS file access functions.

## 4. Multiprotocol Oplock Management

CIFS oplocks provide dramatic performance benefits to Windows-only networks. The challenge of a multiprotocol implementation is to provide oplock functionality that compromises neither on data integrity guarantees nor on maximizing accessibility via NFS to oplocked data.

### 4.1. CIFS Opportunistic Locks

Although the sharing of files and data between multiple Windows clients is fully supported by the CIFS protocol, file sharing between multiple clients is quite rare on most networks. The CIFS protocol leverages the rarity of file sharing in its implementation of the Windows networking performance optimization known as "opportunistic locks" (*oplocks).* An oplock is an exclusive (i.e. *Read-Write/Deny-All*) file-lock that the CIFS client system obtains from the CIFS file server "opportunistically" at application file open time *if* the file being opened is not currently being accessed by any other application. By obtaining an oplock, the client gets a *temporary* exclusive lock on the file being opened, despite the fact that the application did not request an exclusive open. The oplock is *temporary* in the sense that it can be *broken* (i.e. revoked) in case another application tries to gain access to the file. While holding the oplock, the client system takes advantage of the fact that the file is not currently being accessed by any other application. The client's operating system can then — without compromising data integrity — optimize the client/server network traffic needed to satisfy the file accesses by the application. The client holding an oplock minimizes network traffic to the file server by:

1. Performing aggressive read-ahead on open files. Because the client knows that it is the only system accessing the file, it can be certain that the read-ahead data that it obtains is not being changed by other clients back on the server.
2. Aggressively caching write operations to files. There is no need to propagate file changes back to a server synchronously if there are no other clients accessing the file concurrently. Write operations can be delayed and aggregated to optimize I/O performance.
3. Aggressively caching lock requests. The client has no need to inform the server of the various locks an application may have acquired on a file if it can be certain that no other clients are accessing the file concurrently. The locking semantics can be managed entirely on the client side of the connection.

When an application on a CIFS client opens a server-resident file, the client's operating system typically requests an oplock (almost invariably, a type of oplock known as a *batch oplock* that outlives the application's open, as described below) when it transmits a CIFS open request to the file server. If the open was non-exclusive, and the reply to the open did *not* grant an oplock, then the client must synchronously propagate all of the application's interactions with the file back to the file server. In particular, all of the application's write and byte-range lock operations against the file must be synchronously transmitted to the file server. By contrast, if the reply to the open *did* grant an oplock, then the client's operating system can locally-cache write and byte-range lock operations against the file, and can perform aggressive read-ahead on the file.

Oplocks reduce the network traffic transmitted between a CIFS client and the file server, reducing the burden on both network and file server. Note that oplocks are not exposed to Windows applications through the Windows API. They are internal to the CIFS protocol, and are requested automatically from the CIFS file server by the client's operating system at file-open time.

CIFS oplocks are roughly comparable in functionality to the token-based cache synchronization mechanisms of the OSF DCE/DFS distributed file system [10]. On the other hand, the "callback" mechanism of Transarc's AFS is less functional, being equivalent only to DFS "status read" tokens. There is no mechanism for the file server to tell the AFS client to store back to the server any data that the client has modified locally[10].

#### 4.1.1. Oplock Break Protocol in a CIFS-Only Environment

When a second CIFS client attempts to open a file for which there is an outstanding oplock held by an existing CIFS client, the new opener is "held off" while the file server sends the oplock holder an *oplock-break* message. An oplock is actually held by a client operating

system, not by an application. Consequently, a client system may choose to hold an oplock for a file even after the application that caused that oplock to be obtained has closed the file and exited. Upon receiving the oplock-break message from the server, the CIFS client operating system can respond in one of two ways:

1. Close the file (after possibly flushing any locally cached write operations on the file back to the server). This would be the client's response in the case of a *batch oplock* when no local opens of the file remain.
2. Flush all outstanding CIFS write and lock operations on the file back to the server, and discard any read-ahead data that it may have obtained for the file. The read-ahead data must be discarded because the second client may subsequently write to the file, invalidating data that the first client originally obtained via read-ahead operations. After it flushes all the cached operations to the server, the client sends the server an *oplock-break-acknowledgement* message, completing the oplock break sequence.

In Case 2, the file remains open. However, the lock-mode of the file-lock representing the open is *down-converted* [8] from lock-mode *Read-Write/Deny-All* to a lock-mode that corresponds to the access- and deny-mode of the original open request. After acknowledging the oplock break, the client must revert to a mode of operation in which it synchronously transmits writes and byte-range lock operations on the file to the server, and in which it refrains from performing file read-ahead.

### 4.1.2. Exclusive, Batch, and Level II Oplocks

The CIFS protocol defines three types of oplocks: (1) exclusive, (2) batch, and (3) level II.

If a client operating system requests an *exclusive* or *batch* oplock when an application opens a file — and the client is the first and only opener — then the server may grant the client an exclusive or batch oplock. The oplock is represented as a temporary (i.e. *breakable*) file-lock with *Read-Write/Deny-All* lock-mode.

In the case of an *exclusive* oplock, the server then suspends subsequent attempts to open the same file while it breaks the oplock owner's oplock. The oplock break protocol is complete — and the suspended open can proceed (given that its lock-mode is compatible with any file-lock remaining after the oplock break) — once the oplock owner has flushed any outstanding writes to the file server, and has either acknowledged the oplock-break, or has closed the file. Only older CIFS clients appear to use exclusive oplocks. Most CIFS clients use only batch oplocks, a functional superset of exclusive oplocks.

In the case of a *batch* oplock, the server suspends not only subsequent attempts to open the file, but also attempts to remove, rename, or otherwise modify file system metadata associated with the file, while it breaks the oplock owner's oplock. A significant difference between exclusive and batch oplocks is that the latter can outlive file open and close API calls by applications running on the client system. That is, applications on the client system can go through multiple file-open file-close cycles without the client's having to transmit the CIFS open and close requests to the file server. It has been observed that a Windows client will typically request a batch oplock in its CIFS file open request, regardless of the application's requested access- and deny-mode. Acquisition of a batch oplock obviates the client's need to transmit CIFS open and close messages to the server as local applications open and close the file. Unfortunately, some clients will keep a batch oplock *indefinitely* — pending receipt of an oplock break — long after all local openers of the file have exited. This paper terms such a batch oplock "stale."

A level II oplock is — in SecureShare uniform lock-mode terms — a file-lock with lock-mode *Read/Deny-Write*. Holders of level II oplocks on a file — potentially multiple Windows NT clients that may have opened the file for write access, but none of which has been issuing writes — can aggressively cache file read-ahead data. However, they cannot cache file byte-range locks. A client cannot explicitly request a level II oplock at file-open time. However, a Windows NT client opening a file, and requesting an exclusive or batch oplock, can instead be granted a level II oplock. This might occur, for example, if there are multiple concurrent openers, none of which has been issuing any writes. When the first write occurs, all level II oplock openers are *broken to none*; that is, each is sent an oplock break message that revokes its level II oplock, leaving it with no oplock. Alternatively, a former exclusive or batch oplock owner that had originally requested read-only access can be *broken to level II*; that is, it can be sent an oplock break message that revokes its exclusive or batch oplock, leaving it with a level II oplock.

## 4.2. Extrapolating Oplocks to Multiprotocol

SecureShare extrapolates oplock management to the multiprotocol environment, where UNIX/NFS and/or (PC)NFS clients concurrently access files oplocked by CIFS clients. SecureShare's innovation is to allow non-CIFS accesses to oplocked files — both NFS and NLM

requests — to initiate oplock break protocol. The rationale for enabling NLM byte-range lock requests to break oplocks is as follows. Lock-compliant applications issue NLM requests prior to issuing NFS reads or writes. If the NLM request encountered an oplock that it was unable to break, then it would fail, making the file unnecessarily unavailable to the NFS/NLM client. By allowing NFS and NLM accesses to break oplocks, SecureShare ensures that file data remains available to non-CIFS clients while protecting its integrity. A multiprotocol file server that did not enable non-CIFS accesses to break oplocks would have a choice between (1) unnecessarily enforcing a potentially breakable oplock, thereby diminishing file *availability*, or (2) ignoring the oplock, thereby imperiling file data *integrity*. Unnecessary enforcement would cause unreasonable unavailability of the file to NFS/NLM applications in cases where (1) there are no more active openers on the client (i.e. the common case of a stale batch oplock), or (2) there is still an active opener, but its open was non-exclusive. In the case where the oplock is erroneously ignored, the non-CIFS access could lead to data corruption.

### 4.2.1 Oplock Break due to (PC)NFS

If, after an oplock has been granted to a CIFS client, a (PC)NFS client sends an NLM file-lock request in an attempt to open the file, SecureShare breaks the oplock held by the CIFS client. To complete the oplock break, the first client either closes the file; or else it keeps the file-open, but flushes all its outstanding write and lock operations on the file back to the filer. At this point, the (PC)NFS client's NLM file-lock request can be granted — if it compatible with any file-lock remaining after the oplock break. The (PC)NFS client may then access the file.

### 4.2.2 Oplock Break due to UNIX/NFS

If, after an oplock has been granted to a CIFS client, a UNIX/NFS client attempts to read or write the open file, SecureShare breaks the oplock held by the CIFS client. If the CIFS client completes the oplock break by closing the file, or else if the file-lock remaining after oplock break completion without a close is compatible with the NFS operation, the NFS request will complete without error. If the NFS request was a read, the read will now access the latest version of the data, as flushed to the filer as a result of the oplock break. If, in the case of an NFS write, the oplock break did not result in the file's being closed, and if the file was opened with *Deny-Write* or *Deny-All*, then the NFS request will fail.

If, after an oplock has been granted to a CIFS client, a UNIX/NFS client attempts to remove or rename the open file, SecureShare breaks the oplock held by the CIFS client. If the CIFS client completes the oplock break by closing the file (i.e. because it was a "stale" batch oplock), then the NFS request is allowed to proceed. If the oplock break did not result in the file's being closed (i.e. because a running Windows application is holding the file open), then the NFS request will fail.

### 4.2.3 Multiprotocol Oplock Issues

It is worth noting the difference in how SecureShare breaks CIFS oplocks in the contexts of UNIX/NFS data access and (PC)NFS data access to a file. When a UNIX/NFS client attempts to access a file for which an oplock is held by a CIFS client, SecureShare performs the oplock break at the time of the first NFS read or write to the file. However, when a (PC)NFS client attempts to access such a file, SecureShare performs the oplock break at the time the client opens the file. This difference in behavior is due to the fact that, unlike UNIX/NFS clients, (PC)NFS clients request an NLM file-lock at the time a file is opened by a Windows application. Thus SecureShare can detect a (PC)NFS client's intent to access a file at an earlier stage than is possible with a UNIX/NFS client. As the NFS protocol contains no "file-open" operation, SecureShare can only detect a NFS/NLM client's intent to read or write a file at the time that the application actually requests an I/O, or — in lock-compliant applications — requests a byte-range lock.

Without multiprotocol oplock break support in the file server, it would be possible for an NFS client – either UNIX or (PC)NFS based — to corrupt the contents of oplocked files. Such a client could also receive stale, out-of-date data when accessing an oplocked file, even if the Windows application that modified the file exited hours ago (i.e. case of a stale batch oplock). Furthermore, an NFS client could potentially corrupt an oplocked file by NFS-writing into "stale" areas of the file that will later be overwritten when the oplock holder flushes its locally cached file data back to the file. Correct oplock management in a multiprotocol environment is thus seen to be critical to the maintenance of data integrity and cache-coherence.

## 5. Multiprotocol *Change-Notify*

The CIFS protocol contains a feature, *change-notify*, that enables an application running on a CIFS client to request that the server notify the client whenever a change occurs in one or more directories that the client wishes to monitor. The application can monitor either a

single directory or the entire file system subtree under a directory. The application supplies a parameter specifying the kinds of changes to be monitored. Examples of directory-relative events that can be monitored are file/directory creation, deletion, rename/move, attribute change, modification time change, etc. The notification takes the form of a server-to-client message containing potentially multiple entries, each of which specifies the name of a changed file or sub-directory within the monitored directory, together with the type of change. When one client changes a directory that is being monitored by other clients, the monitoring clients are notified of the change so that they can take appropriate action, such as updating a GUI display.

SecureShare extrapolates *change-notify* to the cross-platform, multiprotocol environment. With multiprotocol *change-notify*, modification to a monitored directory by *either CIFS or NFS* triggers the asynchronous transmission of change notification messages to the *change-notify* clients.

SecureShare's cross-platform *change-notify* facility enables independent software vendors to utilize Network Appliance filers in developing cooperating, cross-platform applications that communicate through the filer's file system. UNIX applications can process files, and then "hand them off" to Windows NT applications for further processing. Cross-platform *change-notify* allows these applications to accomplish the "hand-off" simply by copying the files into monitored directories.

The *change-notify* mechanism is implemented using a SecureShare-proprietary type of file-lock, a *change monitoring file-lock*. When a Windows NT application uses the *change-notify* API, CIFS sends a directory file-open, followed by a CIFS *change-notify* request. These CIFS requests cause SecureShare to generate a change monitoring file-lock on the monitored directory. The change monitoring file-lock allows Data ONTAP to efficiently test for the potential need to send change notifications whenever the kernel executes a CIFS or NFS request that makes changes within a directory.

## 6. Summary

SecureShare's approach to multiprotocol network storage fully effectively addresses the data integrity, cache-coherence, and file-sharing issues inherent in a cross-platform file-sharing environment. SecureShare protects shared data from the various challenges to data integrity and cache-coherence that arise when the same files and directories are being read and written concurrently by CIFS and NFS/NLM clients. It does this by reconciling the different and incompatible semantics utilized by CIFS and NFS/NLM clients for file locking, file-open, and file sharing. By allowing NFS and NLM requests to break CIFS oplocks, it ensures that file data remains available to NFS clients while simultaneously protecting the integrity and cache-coherence of that data. SecureShare accomplishes these goals by means of a uniform approach to managing all the locking paradigms in the UNIX and Windows environments.

## 7. Acknowledgement

I am deeply indebted to Pei Cao, Assistant Professor of Computer Science, University of Wisconsin, Madison, who served as my conference paper "shepherd." She reorganized this paper, greatly improving the presentation of the material.

## 8. References

[1] Brown, K., A. Borr: "SecureShare: Guaranteed Multiprotocol File Locking," Technical Report 3024, Network Appliance, Inc., November 1997.

[2] Hitz, D.: "An NFS File Server Appliance," Technical Report 3001, Network Appliance, Inc., 1995.

[3] Sun Microsystems, Inc: "NFS: Network File System Protocol Specification," RFC-1094, DDN Network Information Center, SRI International, March 1989.

[4] Sandberg, R., D. Goldberg, S. Kleiman, D. Walsh, B. Lyon: "Design and Implementation of the Sun Network Filesystem," USENIX Conference Proceedings, USENIX Association. Summer 1985.

[5] X/Open Company, Ltd.: "X/Open CAE Specification: Protocols for X/Open Interworking: XNFS," X/Open Company, Ltd., 1991.

[6] Internet Engineering Task Force Network Working Group: "A Common Internet File System (CIFS/1.0) Protocol", ftp://ds.internic.net/internet-drafts/draft-leach-cifs-v1-spec-01.txt, December 1997.

[7] X/Open Company, Ltd.: "X/Open CAE Specification: Protocols for X/Open Interworking: (PC)NFS," X/Open Company, Ltd., 1991.

[8] Gray, J: "Notes on Data Base Operating Systems," IBM Research Report RJ2188, IBM San Jose Research Laboratory, February 1978.

[9] The Samba Team: "Samba: A LanManager like SMB fileserver for UNIX," home page: http://samba.anu.edu.au/samba.

[10] Kazar, M., et. al.: "DEcorum File System Architectural Overview," USENIX Conference Proceedings, USENIX Association, June 1990.

| | Existing file lock-mode | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **New mode being requested** | | NULL | A: R D: DN | A: R D: DR | A: R D: DW | A: W D: DN | A: W D: DR | A: W D: DW | A: RW D: DN | A: RW D: DR | A: RW D: DW | A: Any D: DA |
| | A: R D: DN | ✓ | ✓ | X | ✓ | ✓ | X | ✓ | ✓ | X | ✓ | X |
| | A: R D: DR | ✓ | X | X | X | ✓ | X | ✓ | X | X | X | X |
| | A: R D: DW | ✓ | ✓ | X | ✓ | X | X | X | X | X | X | X |
| | A: W D: DN | ✓ | ✓ | ✓ | X | ✓ | ✓ | X | ✓ | ✓ | X | X |
| | A: W D: DR | ✓ | X | X | X | ✓ | ✓ | X | X | X | X | X |
| | A: W D: DW | ✓ | ✓ | ✓ | X | X | X | X | X | X | X | X |
| | A: RW D: DN | ✓ | ✓ | X | X | ✓ | X | X | ✓ | X | X | X |
| | A: RW D: DR | ✓ | X | X | X | ✓ | X | X | X | X | X | X |
| | A: RW D: DW | ✓ | ✓ | X | X | X | X | X | X | X | X | X |
| | A: Any D: DA | ✓ | X | X | X | X | X | X | X | X | X | X |

**Table 1**

**Lock Compatibility Matrix**

Used to check whether a requested lock-mode is compatible with a pre-existing file-lock.

A = Access-Mode (R = Read, W = Write, RW = Read-Write, Any = any one of R or W or RW)

D = Deny-Mode (DN = Deny-None, DR = Deny-Read, DW = Deny-Write, DA = Deny-All)

✓ = New open request will be **granted.** **X** = New open request will be **denied.**

| | Existing file lock-mode | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | None | A: R D: DN | A: R D: DR | A: R D: DW | A: W D: DN | A: W D: DR | A: W D: DW | A: RW D: DN | A: RW D: DR | A: RW D: DW | A: Any D: DA |
| NLM Write Lock | ✓ | ✓ | X | X | ✓ | X | X | ✓ | X | X | X |
| NLM Read Lock | ✓ | ✓ | X | ✓ | ✓ | X | ✓ | ✓ | X | ✓ | X |

**Table 2**

**Compatibility of NLM Byte-Range Locks with CIFS File-Locks**

✓ = New request will be **granted.** **X** = New request will be **denied.**