SMB v2.1

David Kruse

SMB 2 Goals

- Simplified Command Set
- Extensible Compounding
- Improved Bandwidth Scaling
- Better Recovery from Disconnects

SMB 2.1 Design Goals

- Continue to improve WAN & Branch Experience
  - Chattiness reduction (leasing, larger dir cache, improved CopyFile, large MTU, distributed cache support)
- Better application support
  - Make durable handles more deterministic (resilient handles)
  - Write-Thru on per write basis with unbuffered handle
  - Modify client timeout for interim (async) operations
- Better performance for 10 gigabit Ethernet
  - Larger MTU

Oplocks

Existing SMB Oplock Model

- Oplocks enable client caching of handle state and file data on a per file basis
  - Oplock breaks are used to flush cached state to server
- Existing Oplock models:
  - Exclusive → caching of reads & writes.
  - Batch → caching of reads, writes & handles.
  - Level II → caching of reads.
- Oplock breaks occur when:
  - Batch & Exclusive oplocks are broken when a 2nd handle is opened to the file.
  - Level II oplocks are broken when the file is modified

Critique of Existing Oplocks

- Multiple open handles on a single client is becoming common. Optimal case less common

```
H1 = open(RW, shared_RWD); // batch oplock
Read(H1, …); // read data and cache on client.
Close(H1);  // delay-closed.
H2 = open(RW, shared_RWD); // collapses onto H1
H3 = open(RW, shared_RWD); // collapses onto H1
Read(H2, …); // data read from cache.
Write(H3, …); // data written to cache.
Close(H2);  // close not sent to server.
Close(H3); // delay-closed.
```

Non-Optimal Oplock Cases

- Non-Identical Sequential Opens

```
H1 = open(R, shared_RWD); // batch oplock
Read(H1, …); // read data and cache on client.
Close(H1);  // close not sent to server.
H2 = open(RW, shared_RD); // doesn't collapse.
// Client must close H1 before opening H2.
// Cache is also lost.
Write(H2, …); // data read from server 2nd time* & written to
   cache.
Close(h2);  // close not sent to server.
```

- Multiple Parallel Non-Identical Opens

```
H1 = open(R, shared_RWD); // batch oplock
Read(h1, …); // read data and cache on client.
H2 = open(RW, shared_RD); // doesn't collapse.
// open sent to server. oplock break for H1, both H1/H2 at level 2.
Write(H2, …); // write not cached, oplock break to none.
Close(h2);  // close sent to server.
Close(h1);  // close sent to server.
```

*Windows implementation issue due to buffer cache

Oplock Break Effects

- Oplock breaks can have a variety of effects on clients:
  - Loss of read/write caching which prevents operation aggregation, leading to poor performance
  - Loss of durability in SMB2 resulting in application errors if a disconnect occurs that could have been avoided
  - Purging of data from the Offline Files cache if read caching is lost, which requires the data be synced later
  - Prevent a client from retrieving data from its peers when using Peer Caching in WAN scenario
  - Transitioning the file offline if there is dirty data in the Offline Files cache, so user changes must be synced later
  - Loss of handle caching results in more creates/closes sent to the server, reducing performance
  - Require application lock traffic to be sent to the server, which slows down applications on WANs
  - …

A New Oplock Model

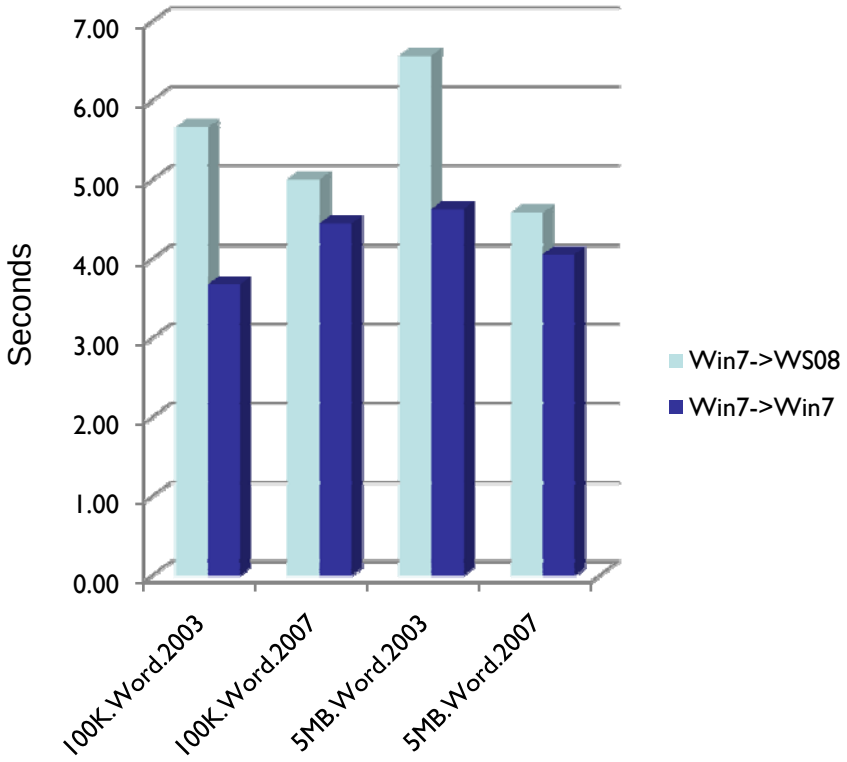
- Parameterized "oplocks" which describe specific combinations of (R)ead-caching, (W)rite-caching and (H)andle-caching intentions.
  - RWH maps to "batch oplock"
  - RW maps to "exclusive oplock"
  - R → maps to "Level_II oplock"
  - RH → new!!
- 'Key' based rather than 'handle' based.
  - A lease key generated by the client is used to link together multiple handles to the same file from the same client.
  - Server tracks state using ClientId (ClientGUID:LeaseKey)
  - Operations on handles with the same lease key do not affect the oplock state.
- Lease state can be upgraded by client (but not downgraded).
- Lease state can be downgraded by server

Contrasting Office 2007 Open

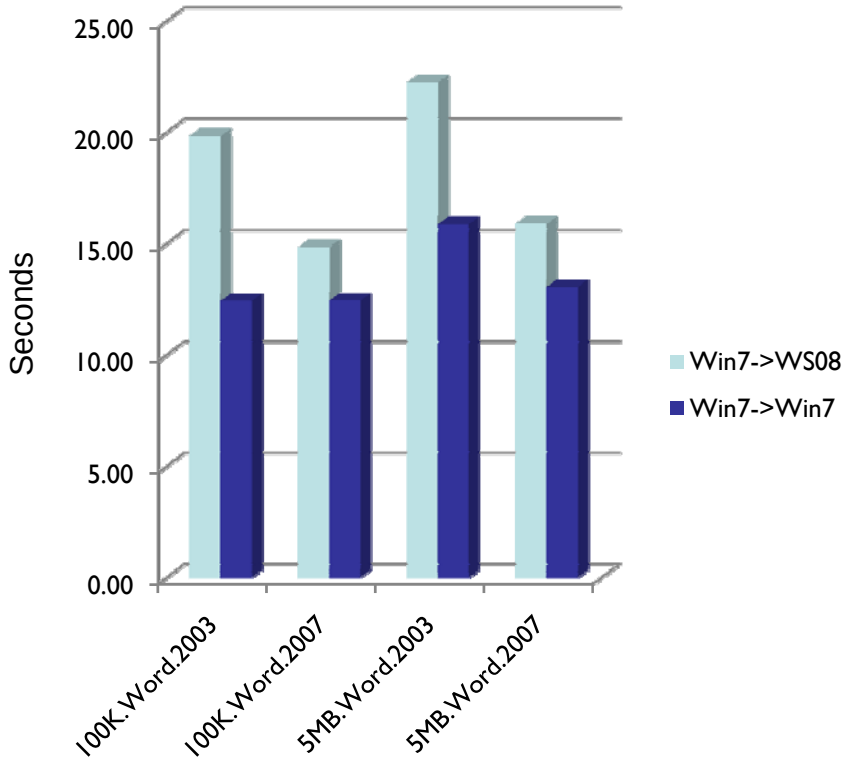
Scenario	Office 2007 open (.docx file - 900k)			Office 2007 open (.doc file - 1MB)		
Build	Win7-Win7	VistaSP1-WS08	Diff	Win7-Win7	VistaSP1-WS08	Diff
CREATE	26	31	-5	26	31	-5
CLOSE	22	30	-8	21	30	-9
READ	20	20	0	29	40	-11
WRITE	5	5	0	5	6	-1
LOCK	0	0	0	0	19	-19
OPLOCK_BREAK	0	1	-1	0	2	-2
IOCTL	12	16	-4	12	13	-1
QUERY_DIRECTORY	6	6	0	6	8	-2
QUERY_INFO	17	21	-4	17	19	-2
SET_INFO	0	0	0	0	0	0
TOTAL PDUs (REQUESTS):	108	130	-16.9%	116	168	-31.0%
Total Bytes Sent to server	15,966	17,954	-11.1%	16,839	21,264	-20.8%
Total Bytes Received from server	935,406	939,876	-0.5%	1,099,363	1,111,498	-1.1%

- Leasing with Office 2007 provides a substantial reduction in
  - Create/close traffic
  - Read traffic (and some write traffic) for .doc files
  - Lock traffic for .doc files
  - Oplock break traffic
- Office 2007 open of .docx file is effectively CopyFile traffic . Open of .doc file is much less optimal
- IOCTL, QUERY_DIRECTORY, QUERY_INFO reductions due to improved client metadata caching (i.e. unrelated)

Application Perf Comparison



65 ms latency



200 ms latency

Large MTU

64k Should Be Enough for Anyone?

- NetBIOS API limited chained sends to 2x (64k-1)
- SMB1 requests initially defined to remain within transport limits
- Later requests (REQ_NT_READ_ANDX) add "ByteCountHigh", but transport implementations are often unchanged
- Move off of NetBIOS to TCP-445 laid the groundwork for larger messages

Moving Large I/O Beyond Copy

- CopyFile API's allows full control of buffering
- Pushing Large I/O mainstream requires:
  - App changes
  - Client Caching changes
    - Evaluate algorithms on caching/paging may be based on page size or other artificial limitations
    - Design in custom scaling based on transport
- Large I/O may require changes to other components
  - Speculative reads may have higher cost
  - Cache management may require refining as larger chunks of data claim cache space
  - Synchronous blocking on I/O affects performance

Protocol Changes

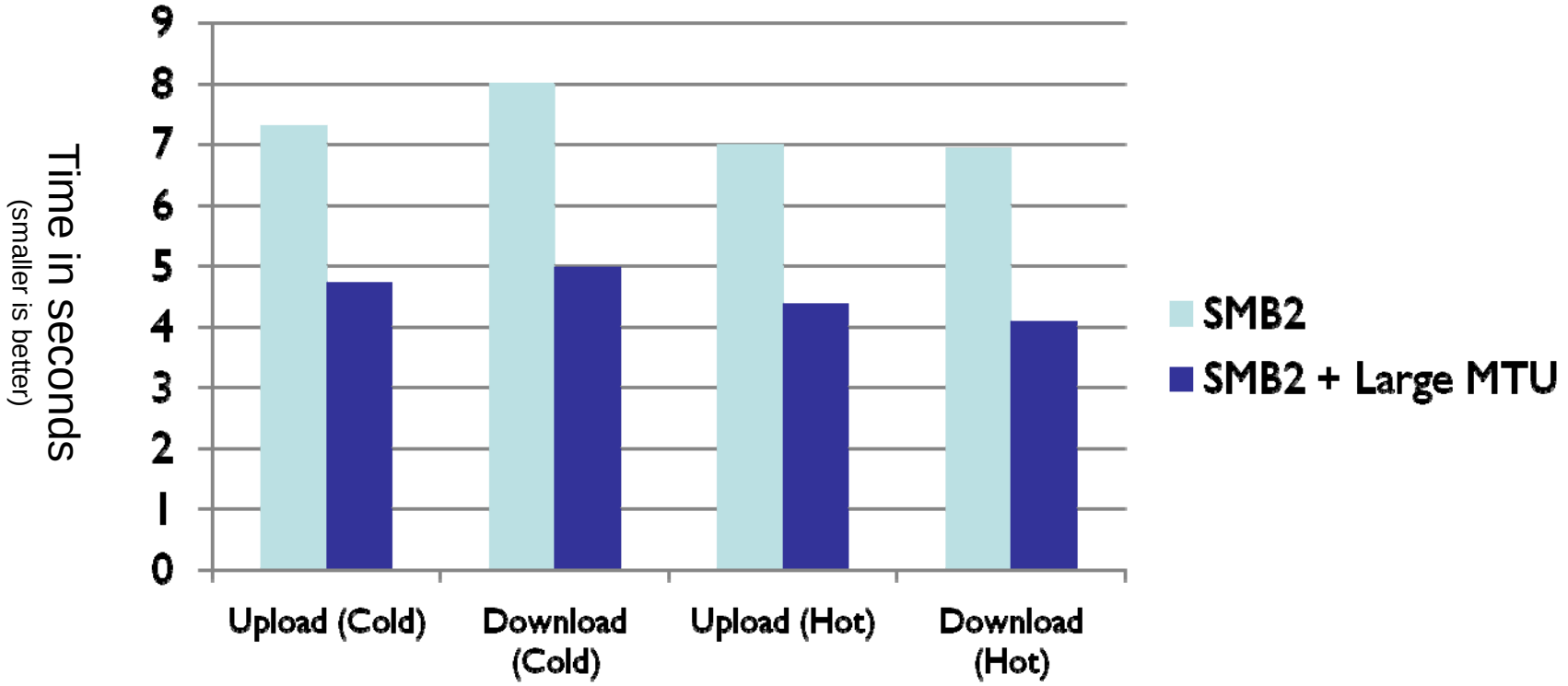
- Smb2: R NEGOTIATE (0x0), GUID={C57C03AC-5B20-F194-4EF8-441FEEA0C794}, Mid = 1
  SMBIdentifier: SMB
  + SMB2Header: R NEGOTIATE (0x0)
  - RNegotiate:
    Size: 65 (0x41)
    SecurityMode: Signing Enabled (0x1)
    DialectRevision: 528 (0x210)
- Capabilities: 0x7
    DFS: (.....1) DFS available
    Leasing: (.....1.) Leasing available
    LargeMTU: (.....1..) Large Messages available
    MaxTransactSize: 1048576 (0x100000)
    MaxReadSize: 1048576 (0x100000)
    MaxWriteSize: 1048576 (0x100000)
    SystemTime: 05/29/2009, 08:16:22 PM
    SystemStartTime: 05/09/2009, 06:17:33 AM
    SecurityBufferOffset: 128 (0x80)
    SecurityBufferLength: 120 (0x78)
    Reserved2: 0 (0x0)
  + securityBlob:

-SMB2: C WRITE (0x9), FID=0xFFFFFFFF00000005, 0x100000 bytes at offset 36700160 (0x2300000), Mid = 570
  Epoch: 16 (0x10)
  Credits: 15 (0xF)
  MessageId: 570 (0x23A)
  DataOffset: 112 (0x70)
  DataLength: 1048576 (0x100000)
  Offset: 36700160 (0x2300000)

Smb2: R WRITE (0x9), 0x100000 bytes written, Mid = 570
  SMB2Header: R WRITE (0x9)
  Size: 64 (0x40)
  Epoch: 16 (0x10)
  Command: WRITE (0x9)
  Credits: 16 (0x10)
  DataLength: 1048576 (0x100000)

I/O Performance Improvements

X-Copy, 2 GB file, 10 gigE, TOE enabled, credit tuned,
4 Gb Ram, dual-proc, **Preliminary**



Metadata Caching with Large MTU

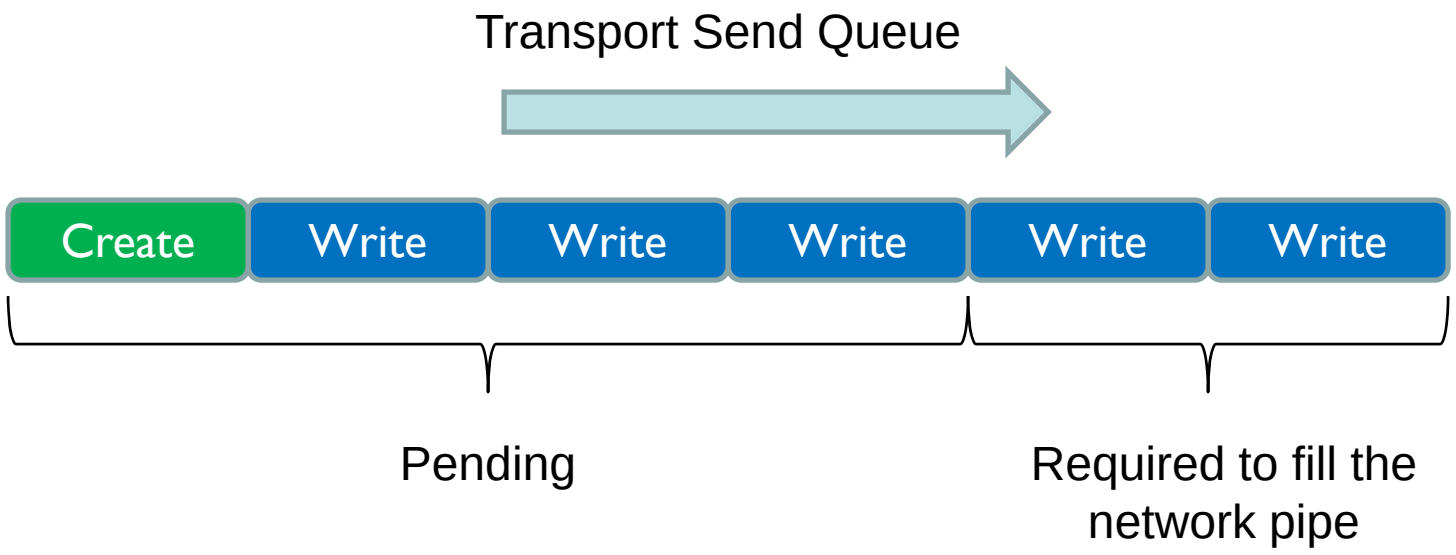
- Large MTU enables caching of large directories
  - 64k = 300-500 entries, 1 MB = 6k-8k
- Dynamic sizing enables cache scaling
- Exposing tunable parameters allows customizing cache for targeted workload
  - Sample customer engagement yielded 70%+ traffic reduction using focused non-default tuning

I/O Prioritization

App 1

App 2

- Foreground application synchronous operation may be pended behind large outgoing queue
- Many meta-data related app operations are inherently synchronous, so large data movement dominates the connection

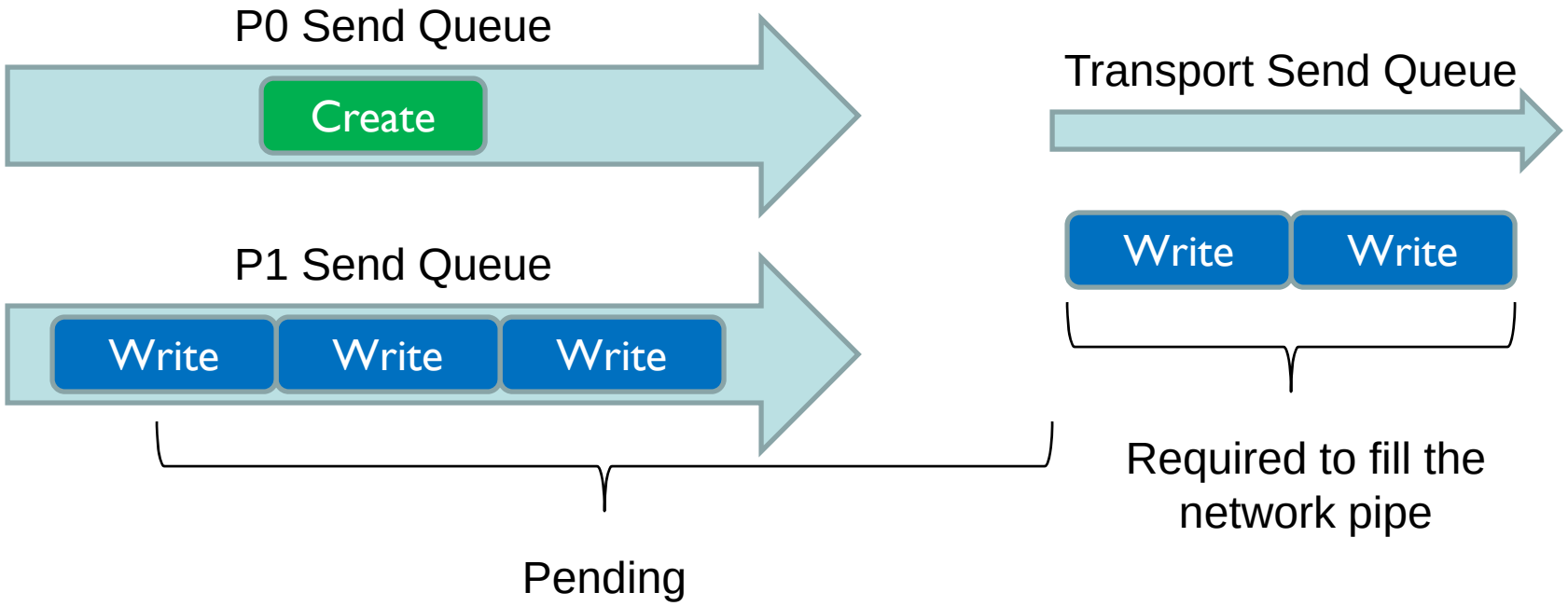


I/O Prioritization

App 1

App 2

- Prioritized send queue selection allows synchronous (and small) metadata operations to “promote” in the queue
- Client can still post sufficient data to keep pipe “filled”



Durability++

Design Goals for Durability

- Seamlessly recover from network disconnects when possible
  - No application integration required
  - No side-effects on other opens
  - Opportunistic in nature (no guarantees)

The Moment of Truth

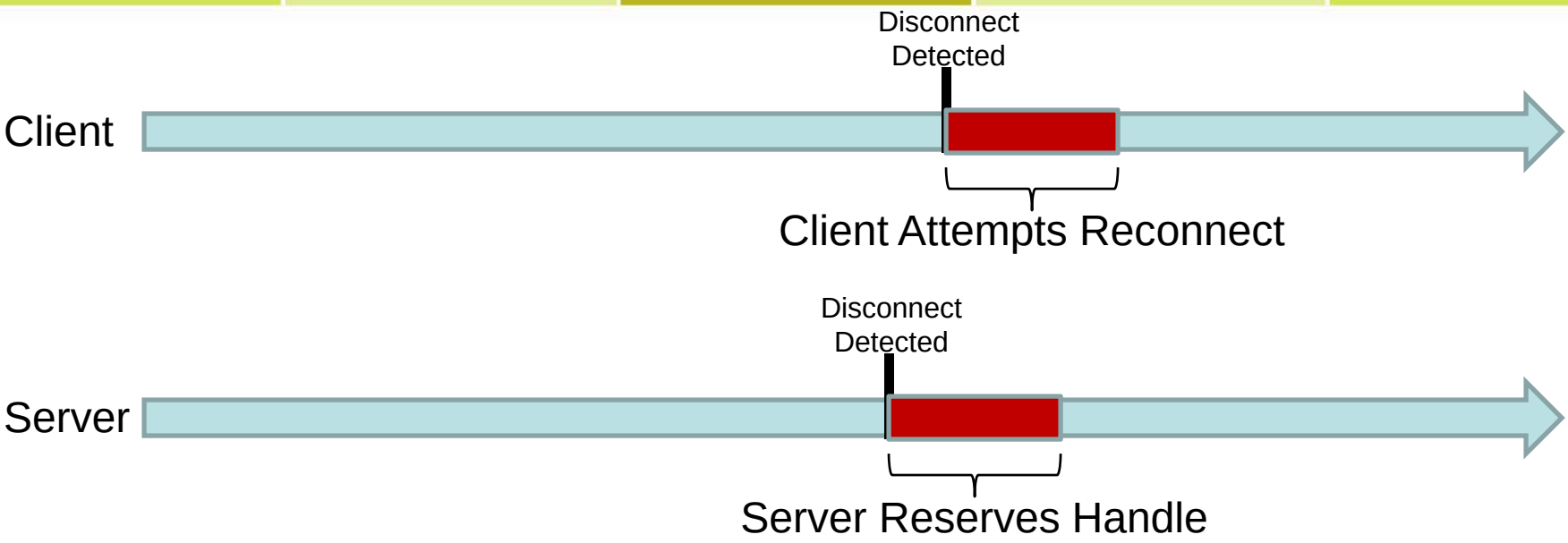
- Client1 opens 'FOO.DAT' for exclusive access
- Client1 is disconnected
- Client 2 attempts to open FOO.DAT

Who wins?

Resilient Handles

- Provide stricter guarantees of open recoverability
  - Application (or intermediary) requests functionality, specifies timeout
  - Other operations on the file may be affected (i.e. sharing violations) while the client is disconnected
  - Independent of oplock state (not affected by other actions on the server)

Recovery Time



- Without disconnect, there are no subsequent requests required to refresh state
- Application can customize limits based on tolerance
- Provides "windows of opportunity" in both directions
  - Either client or server could transition
- Disconnect detection is required, the faster the better
  - Keepalives, Resets, Timeouts, 3rd party notifications

New Concepts for Resiliency

- Oplock/Lease level may change while disconnected!
  - Server acknowledges on clients behalf in a reasonable time
  - Client must adjust caching behavior on non-exclusive opens
    - Locks flow to server
    - Open treated as write-through
- Client reconnect timeout is time based, not retry based
- Lock operations require handle-based "Sequence" to guarantee 'only-once' operation on replay

Minor Updates

SMB 2.1 Minor Updates

- Negotiate
  - Common negotiate may be 2-phase using 2.??? to select SMB2 dialect
    - Guarantees exchange of client SMB2 Negotiate
    - Prevents adding more dialects to the legacy SMB negotiate
- Write-Thru bit on individual SMB2 Write requests

Windows 7 Changes of Interest

- Multi-threaded Robocopy
  - /MT[:n] :: Do multi-threaded copies with n threads (default 8, range 1-128).
- CopyFile utilizes compounded metadata retrieval for increased performance
- Client extends SMB operation timeout based on async responses

Questions?