Samba's Way Toward SMB 3.0

MICHAEL ADAM

Michael Adam is a software engineer with a university background in mathematics and computer science. He leads the Samba team of SerNet GmbH in Germany. As a developer of the Samba software, Michael currently concentrates on the new protocol aspects in the SMB file server and is especially interested in clustering Samba. obnox@samba.org

The well-known Samba software has been a pioneer in clustered Server Message Block (SMB) file sharing since 2007. With Windows 8 and Server 2012, Microsoft has released version 3.0 of the SMB protocol, introducing a whole set of new features for all-active SMB clustering and server workloads. This article describes the interesting challenges that Samba faces in implementing SMB 3 and the missing features of SMB 2—most notably durable file handles—and provides details about the techniques used to solve the issues.

The Samba software [1] has provided open source SMB file, print, and authentication services on UNIX systems since 1992. When version 3.6 was released in 2011, Samba added support for version 2.0 of the SMB protocol.

In late 2011, Microsoft presented what is now called SMB 3.0 under the name of SMB 2.2 at the SNIA Storage Developer Conference [2]. At that time, the Samba engineers had developed a rather good understanding of most of the tasks for implementing SMB 2.1 and had started to work on the one big feature of SMB 2.0 that was missing from Samba's SMB 2. 0 implementation in Samba 3.6: durable file handles.

The announcement of SMB 2.2 got the Samba developers started because this touched an area Samba had pioneered since 2007: all-active clustering of SMB itself. So in parallel to the design and development of durable file handles, the developers dove into the exploration of the new SMB version 3.0. It turned out that in order to implement durable handles, some of the internal design characteristics of Samba that had to be revised and changed were also obstacles for the implementation of the SMB 3 features as well as most parts of SMB 2.1. So the developers took as holistic an approach as possible when implementing durable handles in order to pave the way for moving toward SMB 3.

A year has passed with the outcome that Samba 4.0, which has just been released, features durable handles and support for SMB 2.1—complete except for leases—and basic support for SMB 3.0, not yet including any of the more advanced features. Before we dive into the Samba-specific details, here is an overview of the various versions of SMB.

SMB 1, 2, 3...

The SMB protocol has been around for many years and is the basis of Windows' network file and print services. There are many other implementations of the protocol—Samba being the most popular open source implementation—generally

available in Linux and on most UNIX systems, and used in storage appliances, from small to very large, scale-out clustered systems.

With Windows Vista (released in 2007) and Windows Server 2008, Microsoft introduced version 2.0 of the SMB protocol. SMB 2.0 was essentially a cleanly re-designed variant in the long evolution of the SMB protocol. The number of calls was reduced, all file operations were handle-based, and chained requests were integrated into the protocol. One of the biggest additions of SMB 2.0 was the concept of durable file handles: durable handles can be reclaimed by the client after a short network outage with all lock and caching state, so as to allow for uninterrupted I/O.

SMB 2.1, introduced with Windows 7 and Windows Server 2008 R2 in late 2009, brought a couple of new features: leases, which are "oplocks done right," and handle caching introduced systematically with refined cache revocation rules as compared to the old batch oplocks; and applications were able to upgrade their caching mode without the need to break the mode in advance. Multi-credit (or large MTU) was a new mechanism that reduced the number of network round trips required for reading or writing large hunks of data by allowing for bigger data units to be transferred in a single read or write request.

Resilient file handles added certain guarantees to durable handles. Their usefulness was limited, though, because resiliency had to be requested by an aware application that used the published interface. Dynamic reauthentication allowed the client to reauthenticate a session proactively before the server signaled the session had expired. Branch cache enables machines in an office to cache file server contents locally after it has been fetched over a wide area network link. In summary, the major topics of SMB 2.1 were the reduction of network verbosity and increased reliability.

In September and October 2012, respectively, Windows Server 2012 and Windows 8 were published, along with version 3.0 of the SMB protocol. This version was called 2.2 until shortly before the official release, but was relabeled due to the scope of the additions.

The main topic of SMB 3 is all-active clustering, especially the scale-out and continuously available characteristic of file shares. With SMB 3, Microsoft for the first time specifically addressed server workloads, for Hyper-V and SQL. Apart from new signing and encryption mechanisms and the so-called secure negotiation, some of the major features of SMB 3 are: multi-channel as a mechanism for the client to bundle multiple transport connections (channels) into a single SMB session; persistent file handles are like durable handles with strong guarantees (in contrast to the resilient handles of SMB 2.1, these are transparent to the client); and support for RDMA-capable transports such as Infiniband or iWARP for reduced latency and CPU load imposed by network I/O operations.

Tasks for Samba

There are a lot of interesting features in the above list that Samba has not implemented or had not implemented in Samba 3.6. Specifically, durable handles of SMB 2.0, leases, multi-credit, dynamic reauthentication of SMB 2.1 and directory leases, the clustering concepts, persistent handles, multi-channel, and RDMA-support of SMB 3.0. To understand how the Samba developers have solved or are planning to

solve the items of this voluminous list of tasks, one first must know some details about the design and internal functionalities of the Samba software.

Design of Samba's File Server

The main daemon of Samba's file server is the smbd process. It listens on the TCP ports 445 and 139 for SMB connections and, for each new TCP connection, forks one `smbd` child process that is afterward exclusively in charge of handling this connection. There is hence essentially a 1:1 correspondence between `smbd` child processes and SMB-client TCP connections to the Samba server.

A certain amount of interprocess communication is needed between the `smbd` processes, mostly regarding locking and other conflicting operations on files. Because the locking semantics in SMB differ widely from what POSIX provides, the obvious UNIX/POSIX locking mechanisms through the file system and kernel are not enough here. Therefore, Samba has introduced a file system abstraction layer that implements the required features of a file system with the required semantics. This layer maintains a set of databases to store information that is not available directly in the underlying POSIX file system. These are most importantly `locking.tdb`, which is the database for open file handles, including share modes, oplocks, and `brlock.tdb`, which stores the byte range locks. On the other hand, the SMB-level pieces of information were kept purely in memory in previous releases of Samba, including the latest version 3.6, which included support for SMB 2.0.

The mentioned databases use Samba's own database implementation, the trivial database (TDB) [3]. It is a simple Berkeley DB-style key-value database that also supports multiple concurrent writers through record locks and memory mapping, and can be used for fast interprocess communication. The TDB databases are used virtually everywhere inside Samba.

In addition to the databases, there is the so-called messaging as a means of interprocess communication. The smbd processes can register themselves as interested in a certain database record, for example, representing a lock on a file. And the process holding the lock will send a message to the waiting process when it releases the lock so that the waiting process can try again to get hold of the lock. The messages are implemented by signals to trigger the other process and TDB records that hold the actual message content.

Samba, the Clustered SMB Pioneer

This architecture of Samba's file server was the main reason that the implementation of a clustered Samba file server was initially relatively easy to achieve: Samba was multi-process, and the important data for interprocess communication was serialized into the TDB databases anyway. So when the developers first started to experiment seriously with clustered Samba in 2006, the main task quickly became making the TDB databases clustered, because serving different TCP connections to a single node by different daemons on that node is in principle not much different from serving TCP connections to different nodes by processes on those nodes.

Some clever design had to be invented to make TDB clustered in a fashion that scaled well. Using a conventional clustered database as a substitute, or even storing the TDB files in the clustered file system to be used for sharing files with Samba, turned out to scale negatively, but the intent was to create a system that would scale positively (if not linearly) with the number of nodes in the cluster

with respect to the number of operations and the accumulated SMB throughput per time unit. In 2007, the clustered TDB software CTDB [4] was released and achieved this goal (see my paper about clustering Samba with CTDB [5]).

This clustering of course cannot be perfect, because it must be transparent to the client; the Samba server implementation cannot change the Windows SMB client software to be aware of the clustering. To the client, a Samba-CTDB cluster looks like a single SMB server with multiple network interfaces. The main issue is what happens when one node becomes unavailable. The IP addresses associated with that node are migrated to a different node, and the CTDB software sends so-called tickle-ACK packets to the clients previously connected to the now unavailable node to trigger a fast reconnect to the same IP address. But this reconnect means that the client loses all its open file handles with the associated caches (oplocks) and locks. Also, write or read operations that were in process when the node failure occurred are lost. Samba could not implement any retry or replay mechanisms without being able to modify the client.

This form of clustering was good enough, though, for the Samba-CTDB suite to become pretty popular quickly. Big installations started using it, and it was meanwhile used as a base technology in a couple of NAS appliances. The main point was that Windows at that time could not do all-active SMB clustering at all.

This has changed now with SMB 3.0, which has introduced all-active clustering of SMB into the protocol.

Step 0: Client Implementation and Tests

Despite the availability of protocol documentation from Microsoft in the MSDN library [6] since 2008, after the court decision in the European Commission competition case, the initial step in Samba's development process toward understanding and implementing new features in the server is usually still to write tests. The official source about SMB 2 and 3 is the [MS-SMB2] document from MSDN [7]. At the time Samba started to explore SMB 3, the SMB 3 content of this document was still nascent, so test cases were inevitable. The tests are usually written as part of the smbtorture program. These tests are run against Windows and are extended until a good understanding of the protocol aspect is achieved, and as a next step the server implementation is extended until the tests are passed. The prerequisite for writing these tests for SMB features is a client implementation of these features. At the beginning of the exploration of durable handles and SMB 3.0, there were effectively four SMB client libraries in Samba: implementations for SMB 1 and SMB 2 in `source3/` and `source4/`.

To understand this, one has to know that the Samba code had been split into two code bases with the release of Samba 3.0 and the start of the Samba4 project in 2003. These two code bases were then known as the Samba3 and the Samba4 projects and were, despite the original intent, developed in parallel until they were merged again in 2008. From this time on, the code that was originally from the Samba3 tree was found in the source3 directory, and the `Samba4` code in the source4 directory. Although the code has been increasingly reconciled since the merge, the client implementations and test tools were still completely separate until last year. None of these client implementations was complete and each had its own problems.

As a first step, the developers created a common low-level base library for SMB1 and SMB2, and the existing libraries were turned into wrappers around this new library. The new client library is located in the files

```
libcli/smb/smbXcli_base.h
libcli/smb/smbXcli_base.c
```

With this client library a whole new set of tests have been written for durable and persistent handles, leases, multi-credit, multi-channel, and many more aspects of SMB 1 and 2. The test tool even uncovered a couple of bugs in the Windows Server 2012 prereleases. It still remains to unify the higher level libs into a single SMB client library that is used in all tests and in the smbclient command line client tool.

Implementing Durable Handles

The basic support for SMB 2.0 in Samba 3.6 could be added while keeping the original overall design paradigms explained above, namely the principle that the SMB-level pieces of information about sessions, share connections ("tree connects" in SMB speak), and open files were previously kept in memory and not marshaled into databases, because there was no need for interprocess communication at that level.

Durable file handles of SMB 2.0 created a new situation, in that they are a purely SMB-level concept. When the client is disconnected, the SMB server effectively keeps the file behind the durable handle open and gives it back to the client when it reconnects and reclaims the durable handle. More concretely, when a client reconnects after a short network outage, it uses a special form of the SMB2 session setup, the so-called session reconnect, which is characterized by the presence of the `PreviousSessionId` field. The server is to delete all tree connects associated to the session and close associated file handles except for the durable ones before replying to the session setup request. Thus, Samba needs to be able to look up SMB sessions, especially disconnected ones, by session ID, and tree connects by the tree ID. After establishing a new tree connect, the client requests to reconnect its durable handles by specifying the `FileId` in the durable handle reconnect create context. Hence, Samba needs to be able to look up open file handles by their file ID.

Furthermore, a reconnecting client will talk to a newly created `smbd` process, and hence the file handle in Samba can no longer be tied to a single `smbd` process. This shows that in order to implement durable handles, Samba needed a way to access session, tree connect, and file handle information from different processes, i.e., to create new SMB-level databases for sessions, tree connects, and open file handles.

After some initial thoughts about really keeping the files open until a client reconnects and tries to reclaim its durable handles, and then passing the open file to the new `smbd` via fd-passing, the developers chose a different initial approach: At disconnect, Samba closes the file and marks the entry in the `locking.tdb` database as disconnected. When the client reconnects the durable handle, Samba looks up the handle information in the corresponding databases, reopens the file, and reestablishes all modes and locks. This approach has the advantages of being easier to implement and of having a chance of succeeding when an SMB process gets killed, or when a reconnect happens to a different node in a Samba-CTDB cluster. One disadvantage is that it is not interoperable; between the server closing the file and the client reconnecting there is a possibility that a different application (such as NFS) could access the file without Samba noticing.

In addition to the requirement for new databases, the assumption that the entries in the databases always refer to an existing `smbd` process had to be given up. This point is quite subtly important, since the lazy cleanup of the VFS-level databases relied on each entry being valid if and only if the process referred to by the entry existed. The new disconnected state enters a special new server ID token into the entries that lets the cleanup mechanism skip the pruning of the corresponding entry.

The introduction of the new SMB-level databases was in fact much more involved than one might guess, because the old structures mixed elements from the SMB/SMB2 layer, the FSA layer (see [MS-FSA] referred to by the [MS-SMB2], [MS-SMB], and [MS-CIFS] documents of the MSDN documentation [6]), and Samba's POSIX-VFS layer. The structures were also used across the various layers, so it was rather difficult to change a behavior in just one layer. This situation arose because the Samba code was not cleanly designed in a greenfield environment but was production code, the roots of which began at a time of limited understanding of the protocol and which grew over many years. Hence the introduction of the databases also required a cleanup and reworking of the SMB server.

The result was the `smbXsrv` system, a set of structures, databases, and attached code to form the core code of the protocol side of the SMB server. The data structures are defined in the idl (interface definition language) file

```
source3/librpc/idl/smbXsrv.idl
```

and these are the corresponding new databases:

```
smbXsrv_session_global.tdb
smbXsrv_tcon_global.tdb
smbXsrv_open_global.tdb
```

Now the separation between the SMB layer and the VFS layer is clearer. A couple of the old structures such as `connection_struct` and `user_struct` have been unburdened and moved to the VFS layer, and the old `sessionid.tdb` and `connections.tdb` databases are gone. With a lot of work required on the nitty-gritty details, this was the basis for the implementation of durable handles.

Low-Hanging Fruit

Based on the introduction of the smbXsrv system, a few tasks have become relatively easy and straightforward to implement.

1. *Session reconnect.* This is actually part of the durable handle implementation, but more on the client behavior side and not strictly part of the durable handle negotiation.
2. *Dynamic re-authentication.* This item of SMB 2.1 permits a client to proactively reauthenticate its session.
3. *Multi-Credit.* This feature of SMB 2.1 allows the client to consume multiple data units (so-called credits) in a single SMB request, resulting in a reduced number of network round trips required for the same high-level copy operations.

SMB in Samba 4.0

Samba 4.0.0 has been released on December 11, 2012. This is the first version of Samba shipping with the long-awaited Active Directory domain controller. That is, Samba 4.0 provides all the components required for an Active Directory server, most prominently LDAP, Kerberos, and DNS, along with a whole set of RPC

(remote procedure call) services. In contrast to the original plans of the Samba4 project, this release is made possible by the combination of the Active Directory server part with the advancement of the file server of the Samba3 releases. The original plans of the Samba4 project were to complete the file server of the Samba4 code base, but with the limited developer resources concentrated on the directory features while the production-proven Samba3 file server grew and matured. So the 4.0 release is also the direct continuation of the Samba 3.x file server releases. And for pure file-serving purposes, one can configure and run 4.0 in exactly the way familiar from version 3, omitting the Active Directory part.

Although 4.0 is clearly the Active Directory server release of Samba in the public perception, the changes discussed in this article also make it a big and important file server release.

With the support for durable file handles, Samba 4.0 now ships with full SMB 2.0 support. SMB 2.1 is supported, with the omission of leases, resilient handles, and branch cache. Basic support for SMB 3.0 is also provided, and this includes the new cryptographic algorithms, secure negotiation, and the new versions of the durable handle requests. All missing features are negotiable capabilities that a server need not offer, and hence Samba 4.0 is a correct SMB 3.0 server. Furthermore, these changes lay the foundation for further SMB 3 development currently in preparation.

The maximum SMB version that the server will offer can be controlled with the configuration parameter `max protocol`. The default is set to `SMB3`, but older protocol versions can be chosen with values such as `SMB2`, which is a synonym of `SMB2_10` (i.e., SMB 2.1), `SMB2_02` (i.e., SMB 2.0), and `NT1` (i.e., SMB 1). The full details can be found in the `smb.conf(5)` manual page.

The durable handle feature can be turned on or off in the configuration via the parameter `durable handles`. The default is turned on, but this will only be effective if Samba's means for interoperability have been disabled. The reason for this is that with the current implementation, external access to a disconnected durable handle (e.g. with NFS or AFP protocol) would not be noticed, hence activating durable handles in that multi-protocol situation would open the door for data corruption. More concretely, the following settings in the configuration file `smb.conf` activate durable handles in Samba 4.0:

```
[global]
        durable handles = yes
        kernel oplocks = no
        kernel share modes = no
        posix locking = no
```

The remaining sections of this article describe the plans for the further development of SMB 2.1 and 3.0 features that are currently in preparation.

Leases

One of the interesting features of SMB 2.1 is leasing. As mentioned above, leases can be seen as SMB oplocks done right. Leases and oplocks are modes for caching data operations, opens, and closes on files. In principle, there are three primitives of caching: *read caching* allows caching of data read operations, *write caching* allows one to cache data writes and byte range locks, and *handle caching* allows caching of open and close operations.

The traditional SMB oplocks know three combination of these: *level 2 oplocks* provide read caching, *exclusive oplocks* provide read and write caching, and *batch oplocks* provide read, write, and handle caching. Leases come in four flavors: read leases, read handle leases, read/write leases, and read/write handle leases.

One important change is that by virtue of a so-called *lease key* that identifies the lease, clients can, in contrast to the case of oplocks, upgrade their caching mode without revoking their original lease. Furthermore, the maximum sharable caching mode is changed from read to read and handle, which is implemented by the new read handle leases.

Finally, SMB 3.0 introduces a new concept of *directory leases*: a lease on a directory allows the client to cache metadata operations on files in that directory in contrast to the data operations cached by leases on files.

Leases, including directory leases, will be implemented in Samba relatively soon. Based on the preparatory work on the `smbXsrv` system, the necessary steps are rather clearly arranged. The additions on the SMB protocol level are mostly obvious. The more subtle changes are required at the FSA layer. The developers have designed extensions to the format of the `locking.tdb` entries to cope with the additional caching modes on that level, so that both the semantics for oplocks and leases can be covered. The main work will lie in the extension of the existing cache handling routines, to reflect the broader cache revocation matrix. Although there is a certain level of protocol interoperability for SMB oplocks due to the so-called Linux kernel leases, this interoperability is already far from perfect, and for leases the offered semantics are simply too different. Therefore, as in the case of durable handles, Samba will probably not start off being able to offer leases in an interoperable environment, and definitely not for directory leases.

More SMB 3

The Samba developers are currently planning and designing the implementation of the large list of features of SMB 3. The remaining part of this paper describes the current plans to implement some of the most compelling features.

Clustering

From Samba's perspective, Windows finally embracing active-active clustering is very exciting. The most interesting question to start with is how well this can be integrated with Samba's CTDB clustering. Fortunately, initial investigations indicate that the concepts introduced by Windows are either orthogonal to Samba's clustering or capable of being integrated quite nicely. SMB 3.0 offers three clustering capabilities that can be attached to a share:

1. *Cluster:* The availability of the share can be monitored with the Witness service, an RPC service described in the [MS-SWN] document at [8]. The Witness service also allows the client to be notified of network interface changes. This is to speed up failovers in a highly available server. With the current state of research and testing, the Witness service will integrate cleanly with the CTDB clustering.
2. *Continuous availability:* This share allows for transparent failover of SMB clients. That is, in case of planned or unplanned outage of a cluster node, the SMB client is guaranteed to be able to reconnect to a different node without interruption of I/O, so the applications using the SMB file share will not notice. This

is based on certain retry and replay concepts, and it is the foundation and the prerequisite for persistent file handles.

3. *Scaleout:* This is the all-active nature of shares, that is the share is available on all nodes of the cluster simultaneously. It increases the available accumulated bandwidth for a share. Load can be balanced across the cluster nodes, in particular administratively triggered using the transparent SMB failover. The all-active nature also enables faster recovery of durable handles. Because the scale-out characteristic is the basis of Samba's CTDB clustering model, the understanding is that CTDB will enable building scale-out SMB3 clusters with Samba. The details of this are currently being worked out.

The bottom line is that SMB 3.0 adds clustering capabilities that CTDB clusters also have, but without client awareness. With SMB 3.0, the server introduces clustering infrastructure, while the main logic for failover and retry is in the client. This fact is the main reason that this will integrate with CTDB, so Samba developers will not need to invent a completely new clustering model.

Multi-Channel

As mentioned above, multi-channel is a mechanism for the client to bundle multiple transport connections (channels) into a single SMB session. The I/O load is spread across the channels, and the session is more robust against network failures: a session is intact as long as there is at least one intact channel on the session. Multi-channel is also the basis for RDMA support. The prerequisite for channel bundling is the new interface discovery control that the server has to offer. Based on the information this control delivers, the client decides which interfaces it uses.

The starting point for implementing multi-channel in Samba is to give up the assumption that smbd processes correspond bijectively to TCP connections to the SMB ports. The plan is to transfer TCP connections that belong to the same client to a common process by applying a technique called fd-passing to the TCP-socket. This transfer will be done at protocol negotiation time based on the client identifier called Client GUID. There are concrete plans for most of the details, but the complete implementation of multi-channel in Samba will be a rather large task, since the infrastructure for finding a server based on the Client GUID and passing a connection from one server to another needs to be created. Furthermore, the assumption that one smbd serves only one TCP connection is currently rooted firmly in the server code, and this needs to be carefully removed.

Persistent Handles

Persistent file handles are like durable file handles but with guarantees, whereas durable handles are a best-effort concept. This kind of file handle is only offered on continuously available shares. The SMB protocol mechanisms for obtaining and reconnecting persistent handles are in principle already available in Samba 4.0, although not activated: they are treated by special flags in the SMB3 version of the durable request. The difficult part of the implementation are the additional guarantees attached to persistent handles. In contrast to the durable handles, whose pieces of information are stored in the usual volatile databases, information for persistent handles will need to be stored persistently, so that a disconnected persistent handle will, for instance, survive a server restart. The details of the implementation are not designed yet, but this will definitely come with a performance penalty regarding opening files. But this is expected and accepted

for persistent handles; these are targeted at server workloads such as Hyper-V and SQL, which don't open and close files at a high frequency but rather work on a small number of open-end files for a long time, the important thing being that the open cannot be lost.

SMB over RDMA

The final SMB3 topic to be touched here is *SMB Direct*, the variant of SMB 3.0 that can use RDMA-capable transports such as Infiniband or iWARP-capable 10G Ethernet adapters for the data transfers. SMB direct uses a normal TCP connection as the initial connection, and this is always used for the protocol head. Then multi-channel is used to bind an RDMA-capable channel to the session for the payload data in reads and writes. This is a topic where much research is still needed, because it requires integration with special RDMA-capable network hardware since Windows does not offer a pure software iWARP implementation as Linux does. The transport abstraction is already largely designed and prototyped, but the work does not stop there; existing iWARP client libraries `libibverbs` or `librdmaca` must be integrated. They are currently not fork-safe, and they can't be used with fd-passing, which is the proposed mechanism for multi-channel session binds. So Samba needs some changes in such libraries.

Conclusion

Samba 4.0 will be an exciting release, not only because of the new Active Directory server component, but also as a file server release. The new release features basic SMB 3.0 support, and comes with support for durable file handles as the big new feature. The developers are currently busy designing and even starting the implementation in Samba of leases and many features of SMB 3, most notably support for scale-out and continuously available shares, multi-channel, and persistent handles. The main goal is to enable full support for SMB 3.0 clustering, e.g., running Hyper-V on a Samba-CTDB cluster with one of the next releases (4.1 or 4.2) of Samba.

References

[1] Samba, the open source SMB server for UNIX: http://www.samba.org/.

[2] The SNIA Storage Developer Conference 2011: http://www.snia.org/events/storage-developer2011.

[3] TDB, Samba's trivial database: http://tdb.samba.org/.

[4] CTDB, the clustered TDB project: http://ctdb.samba.org/.

[5] Michael Adam, "Clustered NAS for Everybody: Clustering Samba with CTDB," http://www.samba.org/~obnox/presentations/sambaXP-2009/samba-and-ctdb.pdf.

[6] MSDN Library, Open Specifications: Windows Protocols: http://msdn.microsoft.com/en-us/library/ms123401.aspx.

[7] MSDN Library, [MS-SMB2]: Server Message Block (SMB) Protocol Versions 2 and 3: http://msdn.microsoft.com/en-us/library/cc246482%28v=prot.20%29.aspx.

[8] MSDN Library, [MS-SWN]: Service Witness Protocol: http://msdn.microsoft.com/en-us/library/hh536748%28v=prot.20%29.aspx.