Considerations for Choosing an SMB Stack in Embedded Devices

Introduction

This article is the second of two relating to the use of the SMB network file-sharing stack on embedded systems. The first article discussed what an SMB stack for embedded devices is and the use cases for an embedded SMB stack.

This article will discuss considerations that a development team may wish to address when choosing an embedded SMB stack.

Connected Way (https://www.connectedway.com) has been developing our embedded SMB stack for over 20 years. The perspectives and considerations I share in this document are the results of our experience in this market.

The article series is intended to be beneficial to product managers, development managers, and developers of embedded products that wish to consider the integration of SMB network file system features in their products.

Considerations in Choosing SMB Stack

Once you've decided to add either SMB client or server capabilities to your embedded product, how do you go about deciding on what SMB solution to integrate? SMB is a rich protocol stack. There are lots of features in SMB, and developers made a lot of design decisions when writing the software. Your selection will likely impact the success of your product, either from the bottom line, time to market, customer satisfaction, or support costs.

These are the most crucial concerns that clients have shared with me when considering an SMB solution.

Licensing

This is a big topic. The first decision is whether to go with an open-source solution or a proprietary one.

Open Source

The Samba project has developed a comprehensive open-source solution appropriate for Desktops, Servers, and embedded systems. Samba is licensed under GPLv3. Samba may seem an obvious choice if you're product firmware is based on Linux, but the Samba license is slightly different from the Linux license.

Linux is licensed under GPLv2. GPLv2 does not restrict the use or distribution of the software. It just restricts modifications to the covered software. GPLv3 is categorized as a Copyleft license. If you modify the code, you must publish your changes and allow others to use those changes. If you integrate directly with the software, you must publish the code you are integrating to Samba with the same GPLv3 license. Lastly, you must allow users of your product to modify, and replace the Samba software on your product.

The first two requirements may not be too onerous to most. You likely will not be modifying Samba, and you intend to use the Linux file system interfaces to Samba rather than integrating directly with the Samba libraries. Even under those conditions, you are obligated to share the software you use to configure Samba on your system. You will not be able to use the SMB client library but will need to use the VFS interface which requires you to mount shares before accessing. You will be limited to the Posix APIs to access content that provides a subset of the rich feature set of SMB.

The big issue for most is the fact that you must allow users of your product to replace the Samba software on your product. GPLv3 states:

> *"Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. "*

It is the installation consideration that essentially makes Samba incompatible with embedded. Unless you also provide users the ability to install their own software on the device you will want to utilize a different license.

Proprietary License

A proprietary license is negotiated between yourself and the developer although they may have a standard license they prefer to use. Generally, a proprietary license comes with a cost and with limitations on how you can distribute the resulting executables. There are two main classes of proprietary licenses: royalty-based, and royalty-free.

With a royalty-based license, you are obligated to pay a fee based on the number of installed executables. This is generally disagreeable because it is accounted for within the Bill of Materials (BOM) cost of the product. Essentially it impacts the product margins. It also requires sharing sensitive accounting data with the developer.

With a royalty-free license, the fee is generally assessed once when the software is delivered. There are a few variations of this type of license:

- whether the software and build environments are provided along with the license, and
- how many product models does the license cover.

These are important considerations and can significantly affect the overall cost of the software. It is one thing if you are integrating the software with one model of a product but another if you have a product line of similar models that run mostly the same software but differ in a way that won't impact the SMB software. You may want to consider a product line license.

Life Cycle Cost

Tied very much with the licensing arrangement is the overall Life Cycle cost. When calculating the life cycle cost, you need to consider the cost of integration, the cost of the license, and the cost of support and maintenance. One thing that may not seem intuitive is open source is not always free to the bottom line. There may still be a support cost, whether you do it yourself or subscribe to a service. There is also a cost of time and energy in integrating the product onto your platform. This may involve porting, configuring, sizing, and simply interfacing the software with your application.

Consider the cost of your engineers working with open-source software. One month of a developer's time working on the "free" software could easily exceed a proprietary license fee.

Proprietary software generally comes with maintenance and support. Depending on the agreement, the first year may be included in the license fee. Subsequent years typically are charged as a percentage of the original license fee. Maintenance will cover bug fixes, and software upgrades while support covers minor customizations and issue resolution.

Sometimes integration services are also included in the original license agreement.

Security Considerations

SMB has gotten a bad reputation in the past, perhaps deservedly, with respect to security vulnerabilities. That reputation was earned with version 1 of the SMB software. The SMBv2 and SMBv3 protocols have cured those vulnerabilities. For the most part, SMBv1 has been deprecated although it is in use by some systems.

Security generally involves three concepts: authentication, integrity, and confidentiality. All three of these concepts are in play with SMB.

The WannaCry Ransomware attack from 2017 exploited vulnerabilities in the SMBv1 protocol. At that time, Microsoft's recommendation was to disable SMBv1. That is still the recommendation and for that reason, the use of SMBv1 is discouraged. Table 1 compares the security capabilities of the various SMB versions:

	SMBv1	SMBv2	SMBv3.02	SMBv3.11
Authentication:				
NTLMv2	✓	✓	✓	✓
Kerberos	✓	✓	✓	✓
Integrity:				
HMAC256		✓		
AES-128-CMAC			✓	✓
Confidentiality:				
AES-128-CCM			✓	✓
AES-128-GCM				✓

Table 1. Security Features of SMB Versions

Authentication

Authentication is the process of logging in to the server. It's an important component in the overall security of any solution. In addition to establishing the rights to access the server, authentication sets up keys used in both the data integrity and data confidentiality steps.

There are two types of authentication in most SMB implementations although the protocol does support the addition of other methods. The two types are NTLMv2 and Kerberos.

NTLMv2 involves a handshake directly between the client and the server. The server provides a random pattern to the client which generates a hash over the server-provided random data, the user's password, and some additional data known by both the client and the server. This hash is then provided to the server. The password is never sent over the network and the hash cannot reveal the password. Using the hash, the server can infer whether the client knows the password without needing the client to send it.

This negotiation is performed in the context of SMB's session negotiate and setup handshakes. An initial blob for the authentication may be provided to the client in a negotiate response from the server. If provided, this blob may be used to seed an authentication blob sent in the first session setup request. The server, in turn, may reply with a challenge and some session-specific data in a session setup response with a MORE_PROCESSING_REQUIRED status code. The client can then provide another blob to the server in a follow-up session setup request that proves to the server that the client knows the password.

This exchange is illustrated in Figure 1.

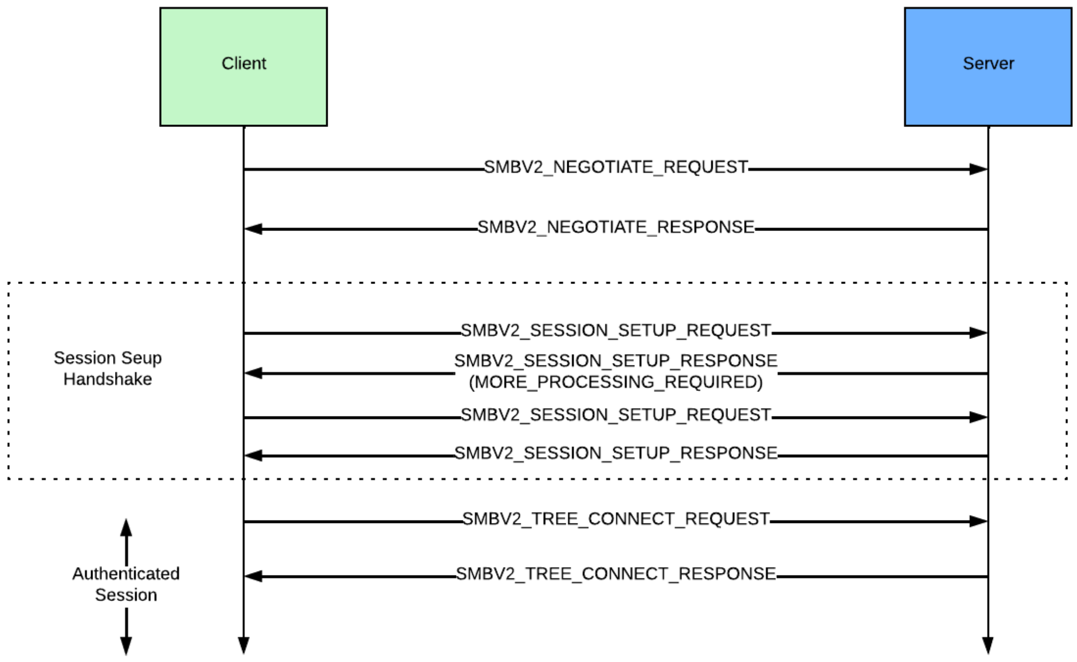
Client
Server
SMBV2_NEGOTIATE_REQUEST
SMBV2_NEGOTIATE_RESPONSE
Session Seup Handshake
SMBV2_SESSION_SETUP_REQUEST
SMBV2_SESSION_SETUP_RESPONSE (MORE_PROCESSING_REQUIRED)
SMBV2_SESSION_SETUP_REQUEST
SMBV2_SESSION_SETUP_RESPONSE
SMBV2_TREE_CONNECT_REQUEST
SMBV2_TREE_CONNECT_RESPONSE
Authenticated Session

Figure 1. Sample SMBv2 Authentication Negotiation

Kerberos is the authentication mechanism used by Microsoft Active Directory (AD) Domains. Kerberos employs a ticketing system. To gain access to a server, a client must first obtain a ticket. The ticket is obtained from a domain controller on the network, which is not necessarily the same system as the file server. The complete mechanism of Kerberos authentication consists of numerous steps and is discussed in blogs and papers on the web. See Figure 2 for a sample Kerberos handshake. The ticket returned from the server proves that the client has authenticated. It can be used multiple times, throughout a lifetime, for access to multiple services on multiple servers. This provides a single and secure sign-on for the user.

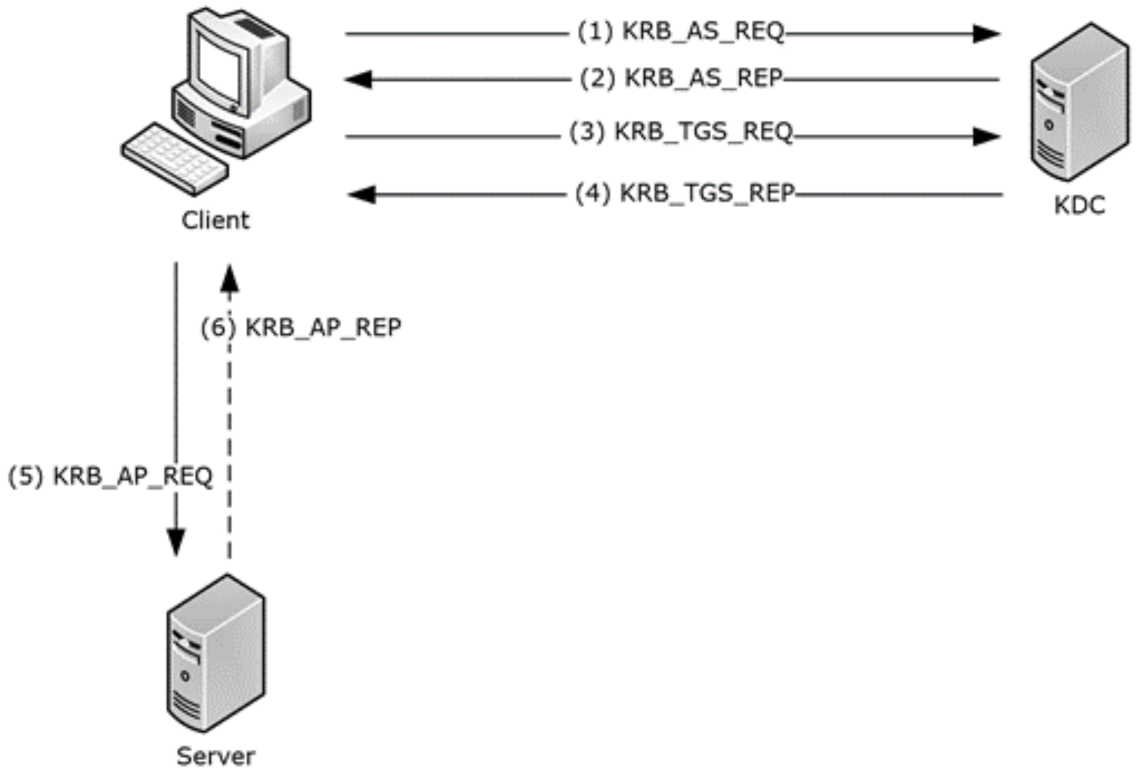

Figure 2. Sample Kerberos Authentication (https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-kile/b4af186e-b2ff-43f9-b18e-eedb366abf13)

Active Directory is generally deployed within an enterprise. Home networks will not support AD unless you have configured a Windows Server or Linux samba-ad-dc server.

When available, Kerberos authentication is preferred. NTLMv2 is susceptible to man-in-the-middle attacks since NTLM only authenticates in one direction. That is, the server validates that the client knows the password, but the client does not validate that the server is whom it expects. For example, a rogue server could accept any authentication hash and trick the client into thinking it is accessing the intended server. Other SMB security mechanisms (integrity and confidentiality) can't help because they rely on authentication to function. This isn't as much of a concern within an enterprise, since the physical network is secure, but is the reason why NTLMv2 is a poor authentication mechanism for the internet.

Data Integrity

Message signing guards against man-in-the-middle attacks. These types of attacks were quite prevalent with SMB such that message signing should always be enabled on both sides of SMBv2 connections. Signing a message involves adding a hash to the message that is unique to the content of the message and a secret that is shared by both the client and the server. The secret is established during authentication. Someone who doesn't know the secret would be unable to generate a valid hash. The message itself can be read by anyone but the important aspect of a signed message is that it cannot be changed without changing the hash. Most SMB servers and clients will refuse to accept unsigned messages. SMBv2 with message signing should be considered a minimum level of security in any SMB deployment.

Confidentiality

It is generally a good idea to encrypt content during transmission to insure the confidentiality of the data. Encryption is supported in version 3 of SMB. It is up to the server to determine if encryption is optional or required on a connection and for the client to agree. If you wish to deploy encryption, you need to ensure that both your client and server support it. Encryption can be deployed server-wide, or on just certain shares (directory trees) on the server. If a share stores no confidential data, you can minimize performance overhead by disabling it on a share, while enforcing it on other shares. You may want to be sure that your SMB client supports encryption both server-wide and share-specific.

SMBv3.02 supports only AES-128-CCM while SMBv3.11 supports both AES-128-CCM and AES-128-GCM. AES stands for the type of encryption, 128 stands for the key size, and CCM/GCM stands for the authentication algorithm. CCM adds message authentication based on the unencrypted packet while GCM adds message authentication based on the encrypted packet. GCM is preferred because it is susceptible to fewer attacks.

Transport Layer Security (TLS)

Although technically not part of SMB, TLS can be used to further secure the SMB protocol. TLS is a form of Public Key Infrastructure (PKI). Most people encounter this daily when they visit a website using a "https" URL. There is a lot of information on the web about how PKI works. It involves the concept of asymmetric and symmetric keys. An asymmetric key involves a pair of keys, both different. One key can encrypt data, but it cannot decrypt it. The other key can decrypt data, but it can't encrypt it. Symmetric keys can both encrypt and decrypt data.

During TLS handshaking, the client contacts the server which returns the public portion of an asymmetric key pair. This is sent in the clear and anyone listening on the network can see it. With TLS, public keys do not need to be secure. The reason for this is that although anyone can see it and encrypt packets with it, only the server can decrypt packets that have been encrypted with it. The content will remain confidential. The client can create a symmetric key, encrypt it, and send it to the server. No one else but the server can see the symmetric key. In this way, both sides now have shared an encryption key and future packets can be secured by using it. This handshake is depicted in Figure 3. The details of the handshake are a bit different, and additional checks are performed. One important feature of TLS is that both sides can authenticate with the other by using the notion of certificates and certificate authorities. So, TLS and PKI provide a mechanism to verify that both the client and the server are communicating with whom they want to be communicating with and you can establish a session where data is exchanged confidentially.

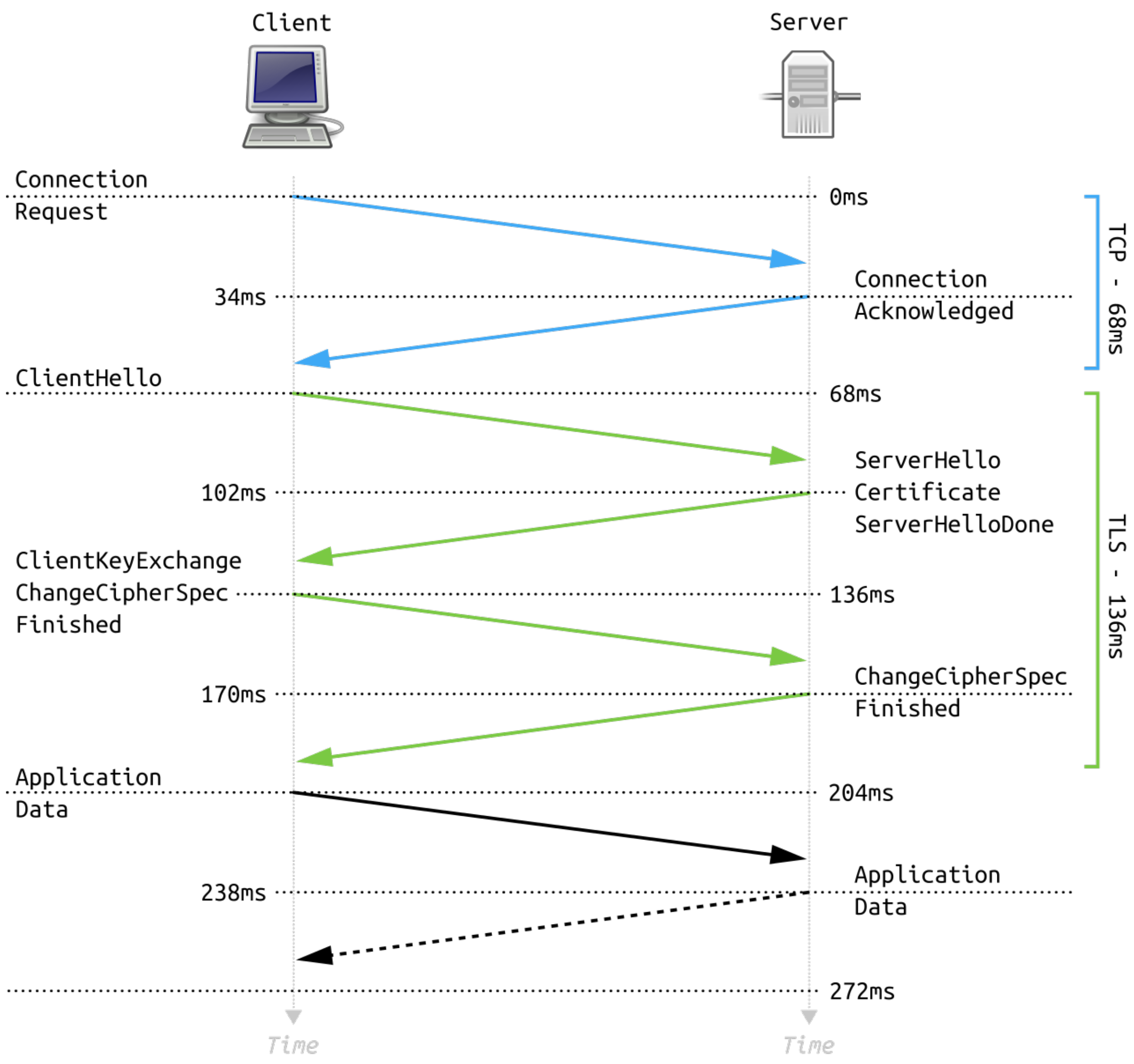

Figure 3. Sample TLS Key Exchange (By Fleshgrinder and The People from The Tango! Desktop Project.)

If you wrap an SMB session within TLS, you get added security and protection against man-in-the-middle attacks against NTLM authentication. VPNs use TLS as the security mechanism. SMB wrapped inside of TLS is as secure as SMB over a VPN.

Performance

Performance is a broad topic and can be looked at from various perspectives. Depending on your application, all perspectives may be important.

Throughput

Throughput is essentially how big is your pipe. If that pipe were carrying water, you may want to measure how much water can flow through that pipe in a unit of time. In the case of SMB, you want to know how much data can be transferred within a second. If it is too slow, data will back up, much like a slow drain. If it backs up for too long, the product will fail. Lots of things affect throughput: the speed of your processor, the speed of your storage, the speed of your network connections, but also, the efficiency of your SMB stack. Throughput is usually measured as some quantity of data per unit of time, (e.g., MB/sec).

Latency

Latency is essentially how long is your pipe. How long does it take something to travel from one end to the other? In the case of SMB, you want to know how long it takes to do an operation. For example, how long does it take to do a read of a 4K block? Latency is usually measured as some amount of time per quantity of data (e.g., ms/transaction).

Latency and Throughput are related, and often are simply inverses of one another, but not necessarily. In almost all software systems there is the notion of delay and buffering, sometimes referred to as queuing delay. For example, say a piece of software was written so that upon receiving a request, it placed the request on a queue, came back after some polling interval, and then processed the request. That polling interval becomes a delay. The number of packets queued throughout the polling interval becomes a queue depth. The true relationship between latency and throughput is defined by a theorem called "Little's Law" which is shown in Figure 4.

$$\underline{Work\ in\ Progress = Throughput\ \cdot Elapsed\ Time}$$

$$Throughput = \frac{Queue\ Depth}{Elapsed\ Time}$$

Figure 4. Little's Law

Overhead

Overhead is another concept like latency but instead of indicating the lifetime of a packet in the system, it relates to the amount of CPU time spent on that packet. Assuming zero queueing delays, then overhead and latency are close to the same thing. Overhead is measured in terms of CPU time per unit of measure (e.g., cpu-ms/packet)

Scalability

Scalability is a characteristic of a software program that identifies how it performs under load. In a perfect world, a system scales linearly. But there are limits to how much a software system can be expected to do and as the load increases, its performance decreases. When it decreases, it does not decrease linearly. Inefficiencies are often built into the design of a piece of software. For example, say that as packets are being processed, they are put in a queue waiting for some event. When that event occurs, the queue is searched for the relevant packet. As the load on the system increases, the length of the queue of packets increases, and the time to search the queue for the relevant packet also increases. Plus, as the load increases, and the number of packets increases, the number of events that cause the queue to be searched may also increase. So, not only does each search take longer, but the number of searches also increases. The result is that as the load increases, the performance (throughput, latency, and overhead) may be negatively impacted. See Figure 5 for a graph of performance results for a metric that does not scale linearly.

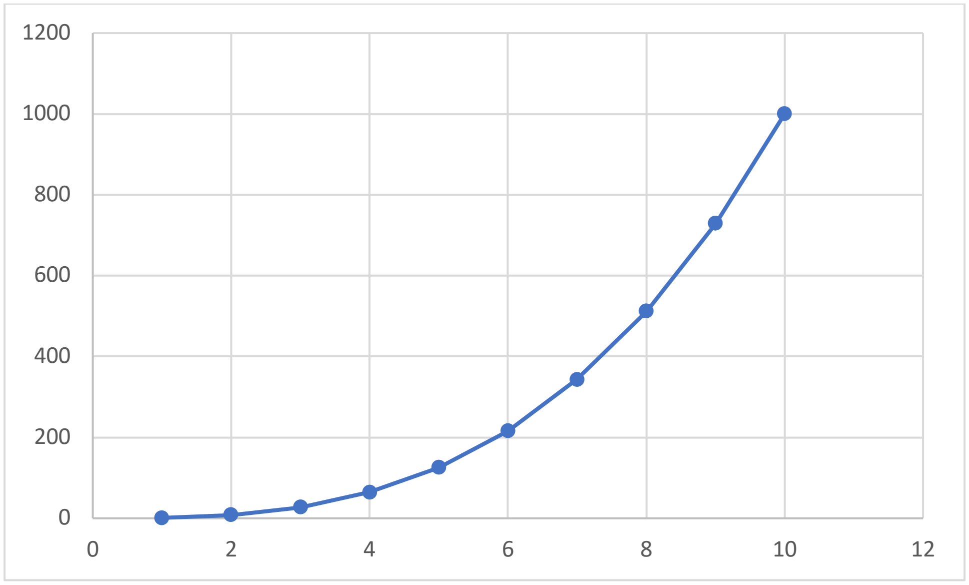

Figure 5. Example of Performance that Doesn't Scale

Scalability is measured as a percentage degradation per some measure. For example, a system's throughput may degrade according to a particular logarithmic formula based on the number of outstanding I/Os.

In a well-designed real-time software program, the scenario of decreased scalability under load is minimized. Instead of searching through increasingly deep queues, a real-time piece of software would be designed around events that maintain context. So, there would be no searching through queues to satisfy an event but rather there would be a direct relationship between an event and a queue element. Finding an element for an event takes the same amount of time whether there are two elements or 10,000 elements.

Therefore, the design of the underlying software can have a large impact on the scalability and hence the performance of the product under load. Whether the software has been designed with real-time in mind, whether it is event-driven, and how it performs under load are all important considerations.

An illustrative notion of this real-time event-driven architecture is synchronous vs asynchronous I/O. With synchronous I/O, when you issue an I/O, your program blocks and waits for the I/O to complete. If you have another I/O to issue, you must issue it in another thread of execution or wait until the previous I/O is complete before issuing the new one. Excessive threading impacts scalability. Waiting till the previous I/O is complete impacts throughput. With asynchronous I/O, you issue the I/O request and continue processing other tasks which may include issuing more I/O requests. As I/O completes, events are delivered to the application. Event processing may issue the next I/O in the sequence. The event context is associated with the buffer and the request. As a result, asynchronous I/O scales close to 100%.

Compatibility

Compatibility is the ability of a piece of software to interoperate with other software that serves the same purpose but has been implemented by different companies on different operating systems. It also implies interoperating with different versions of the same software. Because of the number of SMB implementations, it is difficult to create a definitive interoperability matrix, but my recommendations are shown in Table 2:

	SMBv2.1	**SMBv3.02**	**SMBv3.1.1**
Samba 4.18	✓	✓	✓
Samba 3.6	✓	✓	✓
Windows Server 2022	✓	✓	✓
Windows 11 Home	✓	✓	✓
Windows Server 2012	✓	✓	✓
Windows 10 Home	✓	✓	✓
Macos 13.1 (Ventura)	✓	✓	✓
Macos 11 (Big Sur)	✓	✓	✓

Table 2. Suggested SMB Protocol Interoperability Matrix

Ease of Integration

This goes without saying, but a big consideration is how easy it will be to integrate your application with the underlying SMB stack. By default, a client should need no special initialization or configuration. There are two basic models of SMB client access: The windows model, and the Linux model. On Windows, there is never a need to mount a remote share (although you can assign it to a drive letter). On Linux, you must mount the share for it to be available to applications. Which model is more natural for your developers?

An interesting way to ask the question is, "What is the smallest number of lines in a program that copies a file from the local system to a remote system including any necessary initialization of the SMB stack, and mounting of remote share?"

Miscellaneous

Suggested follow on questions are:

- Must the stack be explicitly initialized?
- Is any configuration necessary to use the client?
- How are remote shares accessed? Must they be explicitly mounted and dismounted?
- How is authentication information provided?
- If there is a configuration, can it be persistent?

- For the server, how are exports configured?
- Does the product support multiple network interfaces? Can SMB be enabled on select interfaces?
- Does the product discover network interfaces, or must they be configured separately? What happens if a network interface goes down?
- Does the product response to network topology changes?
- What is the file naming convention? Is there a flat namespace or do namespaces overlap?
- Can aliases be created for file paths?
- What is the API used for File I/O? Does it follow a Windows-like API or a Linux/Posix-like API or something different?
- Can the client be linked as a shared or static library?
- Can the client be deployed as a user space FUSE handler?
- Is browsing of servers supported? If so, using what protocols? NetBIOS? LanMan? Avahi? SSDP?
- Does it have a Java API? If so, is it compatible with java.io
- Does it have a Swift API?
- For the server, what kind of intrusion detection is there?
- For the server, how is authentication performed? How are users added to the system?
- For the server, is there any access control?

Summary

Hopefully, after reading this blog you have a good understanding of what are things to consider when evaluating an SMB product. The criteria you use can have a significant impact on the success of your product.

If you have questions or comments about this blog or our SMB product, feel free to contact me.

About the Author

Richard Schmitt, CEO of Connected Way, LLC. and Developer of ConnectedNAS.
rschmitt@connectedway.com
https://www.connectedway.com
Austin, TX

Richard has over 40 years of experience developing software for real-time embedded systems in various roles as CEO, CTO, Engineering Management, and Software Development. He founded Connected Way, formerly Blue Peach in 2002 to provide a multi-platform event-driven SMB stack for embedded platforms. Richard is keenly interested in embedded Linux security, and networking. If he's not answering his email, he's likely at the beach or mountain biking.